ADS Immersion 2025: Lösungsvorschlag zu Teil I

Version 25.07.2025

Aufgabe R0: Bibliotheken, Reproduzierbarkeit und zentrale Hilfsfunktionen

Laden Sie an dieser Stelle alle für die weiteren Aufgaben benötigten Pakete. Achten Sie darauf, ausschließlich Pakete zu laden, die tatsächlich benötigt und verwendet werden, um Effizienz und Lesbarkeit des Notebooks sicherzustellen. Stellen Sie zudem die Reproduzierbarkeit des Notebooks sicher. Definieren Sie außerdem zentrale Hilfsfunktionen; insbesondere sind die Funktionen `multiplot` und `get_binCI` aus der Vorlage *SWoF* einzubinden, um Mehrfachplots zu erstellen und binomiale Konfidenzintervalle zu berechnen.

Zunächst laden wir zentral an dieser Stelle alle Bibliotheken, die für die weitere Bearbeitung der Prüfungsaufgabe benötigt werden.

```
# Datenmanipulation und -management
library(dplyr)         # Intuitive Grammatik zur Manipulation von Daten
library(data.table)    # Hochleistungs-Datenrahmen für schnelle Manipulationen
library(tidyr)         # Datenaufbereitung und Umformen in einen aufgeräumten Zustand
library(readr)         # Effizientes Einlesen tabellarischer Daten (z. B. CSV-Dateien)
library(tibble)        # Datenrahmen mit verbesserter Benutzerfreundlichkeit
library(stringr)       # Stringverarbeitung mit konsistenten Funktionen
library(forcats)       # Einfaches Arbeiten mit faktorisierten/kategorischen Variablen
library(here)          # Dynamisches Arbeiten mit Pfaden relativ zum Projektverzeichnis

# Visualisierung
library(ggplot2)       # Kernpaket zur Erstellung vielseitiger Visualisierungen
library(scales)        # Hilfsfunktionen zum Formatieren und Skalieren in ggplot2
library(ggthemes)      # Zusätzliche ästhetische Themes für ggplot2
library(gridExtra)     # Anordnung mehrerer Diagramme auf einem Layout
library(RColorBrewer)  # Farbpaletten für ansprechende Visualisierungen
library(ggrepel)       # Vermeidung von sich überlappenden Textlabels in ggplot2
library(ggridges)      # Erstellung von Ridgeline-Plots zur Visualisierung von Verteilungen
library(GGally)        # Erweiterungen für ggplot2, z. B. Korrelationsplots
library(grid)          # Low-Level-Funktionen für benutzerdefinierte Layouts
library(corrplot)      # Visualisierung von Korrelationsmatrizen
library(plotly)        # Erstellung interaktiver und webfähiger Diagramme

# Explorative Datenanalyse und fehlende Werte
library(VIM)           # Visualisierung und Imputation von fehlenden Werten
library(hstats)        # Identifikation von Interaktionen mittels Friedmans H-Statistik

# Modellierung und Maschinelles Lernen
library(xgboost)       # Schnell und leistungsstarkes Gradient Boosting
library(nnet)          # Multinomiale logistische Regression
library(glmnet)        # Regularisierte lineare Modelle wie Lasso und Ridge Regression
library(mgcv)          # Erstellung von Generalized Additive Models (GAMs)
library(caret)         # Vereinheitlichte Schnittstelle für ML-Modelltraining und Tuning
library(MLmetrics)     # Bewertungsmetriken für Machine-Learning-Modelle
library(pROC)          # Analyse und Darstellung von ROC-Kurven

# Interaktive Daten
library(DT)            # Interaktive tabellarische Datenanzeige und -filterung
library(repr)          # Anpassung der Diagrammgrößen in Jupyter oder ähnlichen Umgebungen
```

Zum Zwecke der Reproduzierbarkeit des Notebooks setzen wir einen Seed und geben die verwendeten Paketversionen aus.

```
# Setze einen Seed
set.seed(42)

# Gebe die verwendeten Paketversionen aus
sessionInfo()
```

```
## R version 4.4.1 (2024-06-14 ucrt)
## Platform: x86_64-w64-mingw32/x64
## Running under: Windows 11 x64 (build 26100)
##
## Matrix products: default
##
##
## locale:
## [1] LC_COLLATE=German_Germany.utf8  LC_CTYPE=German_Germany.utf8
## [3] LC_MONETARY=German_Germany.utf8 LC_NUMERIC=C
## [5] LC_TIME=German_Germany.utf8
##
## time zone: Europe/Berlin
## tzcode source: internal
##
## attached base packages:
## [1] grid      stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] repr_1.1.7         DT_0.33            pROC_1.18.5        MLmetrics_1.1.3
##  [5] caret_6.0-94       lattice_0.22-6     mgcv_1.9-1         nlme_3.1-164
##  [9] glmnet_4.1-8       Matrix_1.7-0       nnet_7.3-19        xgboost_1.7.8.1
## [13] hstats_1.2.1       VIM_6.2.2          colorspace_2.1-1   plotly_4.10.4
## [17] corrplot_0.94      GGally_2.2.1       ggridges_0.5.6     ggrepel_0.9.5
## [21] RColorBrewer_1.1-3 gridExtra_2.3      ggthemes_5.1.0     scales_1.3.0
## [25] ggplot2_3.5.1      here_1.0.1         forcats_1.0.0      stringr_1.5.1
## [29] tibble_3.2.1       readr_2.1.5        tidyr_1.3.1        data.table_1.15.4
## [33] dplyr_1.1.4
##
## loaded via a namespace (and not attached):
##  [1] rlang_1.1.4          magrittr_2.0.3       e1071_1.7-14
##  [4] compiler_4.4.1       reshape2_1.4.4       vctrs_0.6.5
##  [7] pkgconfig_2.0.3      shape_1.4.6.1        fastmap_1.2.0
## [10] utf8_1.2.4           rmarkdown_2.28       prodlim_2024.06.25
## [13] tzdb_0.4.0           purrr_1.0.2          xfun_0.47
## [16] cachem_1.1.0         jsonlite_1.8.8       recipes_1.1.0
## [19] parallel_4.4.1       R6_2.5.1             bslib_0.8.0
## [22] stringi_1.8.4        vcd_1.4-12           ranger_0.16.0
## [25] parallelly_1.38.0    car_3.1-2            boot_1.3-30
## [28] rpart_4.1.23         lubridate_1.9.3      lmtest_0.9-40
## [31] jquerylib_0.1.4      Rcpp_1.0.13          iterators_1.0.14
## [34] knitr_1.48           future.apply_1.11.2  zoo_1.8-12
## [37] base64enc_0.1-3      timechange_0.3.0     splines_4.4.1
## [40] tidyselect_1.2.1     rstudioapi_0.17.1    abind_1.4-5
## [43] yaml_2.3.10          timeDate_4032.109    codetools_0.2-20
## [46] listenv_0.9.1        plyr_1.8.9           withr_3.0.1
## [49] evaluate_0.24.0      future_1.34.0        survival_3.6-4
## [52] ggstats_0.7.0        proxy_0.4-27         pillar_1.9.0
## [55] carData_3.0-5        stats4_4.4.1         foreach_1.5.2
## [58] generics_0.1.3       rprojroot_2.0.4      sp_2.1-4
## [61] hms_1.1.3            munsell_0.5.1        laeken_0.5.3
## [64] globals_0.16.3       class_7.3-22         glue_1.7.0
## [67] lazyeval_0.2.2       tools_4.4.1          robustbase_0.99-4
## [70] ModelMetrics_1.2.2.2 gower_1.0.1          ipred_0.9-15
## [73] cli_3.6.3            fansi_1.0.6          viridisLite_0.4.2
## [76] lava_1.8.0           gtable_0.3.5         DEoptimR_1.1-3
## [79] sass_0.4.9           digest_0.6.37        htmlwidgets_1.6.4
## [82] htmltools_0.5.8.1    lifecycle_1.0.4      hardhat_1.4.0
## [85] httr_1.4.7           MASS_7.3-60.2
```

Wie in der Aufgabenstellung gefordert, binden wir die beiden Funktionen `multiplot` und `get_binCI` aus der Vorlage *SWoF* ein.

```r
# Define multiple plot function
#
# ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objects)
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

 if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}

# function to extract binomial confidence levels
get_binCI <- function(x,n) as.list(setNames(binom.test(x,n)$conf.int, c("lwr", "upr")))
```

Außerdem definieren wir die Funktionen `plot_confusion_matrix` und `calculate_auc` . Die Funktion `calculate_auc` dient zur Berechnung der AUC (Area Under the (ROC-)Curve) und ermöglicht eine flexible Anwendung für verschiedene Szenarien. Die Funktion `plot_confusion_matrix` wird in den Aufgaben R4 und R6 Anwendung finden und dient zur Visualisierung von Konfusionsmatrizen.

```r
# Berechnung der AUC (Area Under the Curve)
calculate_auc <- function(true_labels, predicted_probs, positive_class = NULL, verbose = TRUE) {
  # Wenn positive_class angegeben ist, wird die Zielvariable binarisiert
  if (!is.null(positive_class)) {
    true_labels <- ifelse(true_labels == positive_class, 1, 0)
  }

  # Berechnung der ROC-Kurve und der AUC
  roc_curve <- roc(true_labels, predicted_probs)
  auc_value <- auc(roc_curve)

  # Optional: Ausgabe des AUC-Werts
  if (verbose) {
    cat(sprintf("AUC: %.4f\n", auc_value))
  }

  # Rückgabe des ROC-Objekts und des AUC-Werts
  list(roc_curve = roc_curve, auc = auc_value)
}

# Visualisierung von Konfusionsmatrizen
plot_confusion_matrix <- function(conf_matrix, title) {
  # Umwandlung der Konfusionsmatrix in ein Datenframe-Format
  cm_table <- as.data.frame(conf_matrix$table)
  colnames(cm_table) <- c("Vorhersage", "Referenz", "Frequenz")
  cm_table$FreqNorm <- cm_table$Frequenz / sum(cm_table$Frequenz)

  # Erzeugung eines ggplot2-Diagramms für die Konfusionsmatrix
  ggplot(cm_table, aes(x = Referenz, y = Vorhersage, fill = FreqNorm)) +
    geom_tile(color = "black") + # Erstellung der Rasterkacheln
    geom_text(aes(label = Frequenz), size = 3) +
    scale_fill_gradient(low = "white", high = "blue", name = "Anteil") + # Farbkodierung
    labs(title = title, x = "Referenz", y = "Vorhersage") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5, size = 10, face = "bold"))
}
```

Aufgabe R1: Datenaufbereitung

In dieser Aufgabe liegt der Fokus auf dem Einlesen, Filtern, Aufbereiten und Exportieren von Daten. Ziel ist es, grundlegende Vorverarbeitungsschritte durchzuführen und damit den Hauptdatensatz der Prüfung so aufzubereiten, dass er als Grundlage für die weiteren Aufgaben in Teil A sowie in Teil B genutzt werden kann.

a) Daten einlesen: Lesen Sie die Datei `diabetic_data.csv` aus den Prüfungsunterlagen ein. Stellen Sie dabei sicher, dass Zeichenketten als Faktoren (Datentyp: `factor`) eingelesen werden und dass fehlende Werte nicht explizit durch eine `na.strings`-Anweisung gesetzt werden. Geben Sie die Anzahl der Zeilen und Spalten des eingelesenen Datensatzes aus, zeigen Sie die ersten zehn Einträge an und stellen Sie sicher, dass die Datei korrekt eingelesen wurde. Diskutieren Sie abschließend kurz je einen Vorteil und einen Nachteil des obigen Ansatzes zur Behandlung der fehlenden Werte beim verwendeten Datensatz. Prüfen Sie abschließend, ob der Datensatz neben den Werten `?` auch weitere "echte" fehlende Werte (`NA`) aufweist.

Als Erstes lesen wir die Datei `diabetic_data.csv` mittels der Funktion `read.csv` ein und wandeln Sie durch die Funktion `as_tibble` in einen Tibble um. Beim Einlesen wird darauf geachtet, dass Zeichenketten als Faktoren eingelesen werden (`stringsAsFactors = TRUE`) und dass keine explizite `na.strings`-Anweisung zum Setzen von fehlenden Werten verwendet wird.

```r
# Lese Daten ein
data <- as_tibble(read.csv(here("2025/1_Erster_Ansatz", "diabetic_data.csv"), sep = ",", stringsAsFactors = TRUE))
```

Anschließend geben wir die Anzahl an Zeilen und Spalten des eingelesenen Datensatzes aus und zeigen die ersten zehn Einträge.

```r
# Gebe die Anzahl an Zeilen und Spalten aus
cat("Anzahl Zeilen des Datensatzes:", nrow(data),
    "\nAnzahl Spalten des Datensatzes:", ncol(data))
```

```
## Anzahl Zeilen des Datensatzes: 101766
## Anzahl Spalten des Datensatzes: 50
```

```r
# Zeige die ersten zehn Einträge
head(data, n = 10)
```

encounter_id <int>	patient_nbr <int>	race <fct>	gender <fct>	age <fct>	weight <fct>	admission_type_id <int>
2278392	8222157	Caucasian	Female	[0-10)	?	6
149190	55629189	Caucasian	Female	[10-20)	?	1
64410	86047875	AfricanAmerican	Female	[20-30)	?	1
500364	82442376	Caucasian	Male	[30-40)	?	1

encounter_id <int>	patient_nbr <int>	race <fct>	gender <fct>	age <fct>	weight <fct>	admission_type_id <int>
16680	42519267	Caucasian	Male	[40-50)	?	1
35754	82637451	Caucasian	Male	[50-60)	?	2
55842	84259809	Caucasian	Male	[60-70)	?	3
63768	114882984	Caucasian	Male	[70-80)	?	1
12522	48330783	Caucasian	Female	[80-90)	?	2
15738	63555939	Caucasian	Female	[90-100)	?	3

1-10 of 10 rows | 1-7 of 50 columns

Anhand des oben dargestellten Einblicks in den Datensatz erkennen wir, dass dieser korrekt eingelesen wurde: Fehlende Werte sind durch `?` kodiert, etwa in den Spalten `weight` und `medical_specialty`. Zudem wurden kategoriale Merkmale als Faktoren eingelesen und alle numerischen Merkmale – korrekterweise – als ganzzahlige Integer-Werte erkannt.

Ein Vorteil des obigen Ansatzes zur Behandlung der fehlenden Werten beim verwendeten Datensatz besteht darin, dass einige Machine-Learning-Verfahren nicht mit fehlenden Werten umgehen können. Durch die Kodierung fehlender Werte als eigene Kategorie `?` wird dieses Problem umgangen, da die fehlenden Werte als spezifische Kategorie behandelt werden und somit ohne zusätzliche Vorverarbeitung in derartige Modelle integriert werden können. Ein Nachteil dieses Ansatzes ist jedoch, dass `?` auch ein gültiger Wert oder eine Angabe für "unbekannt" sein kann. Dadurch können fehlende Werte fälschlicherweise mit echten `?`-Angaben gleichgesetzt werden, was die Datenanalyse verfälschen kann.

Schließlich zeigt der folgende Code, dass der Datensatz über die Werte `?` hinaus keine weiteren "echten" fehlenden Werte aufweist:

```
# Zählen der echten fehlenden Werte (NA) im Datensatz
anzahl_na <- sum(is.na(data))

# Ausgabe der Anzahl an echten fehlenden Werten (NA)
cat("Anzahl der echten fehlenden Werte (NA) im Datensatz: ", anzahl_na)
```

```
## Anzahl der echten fehlenden Werte (NA) im Datensatz:  0
```

b) Daten filtern: Sortieren Sie die Daten aufsteigend erst nach `patient_nbr` und dann nach `encounter_id`. Filtern Sie die Daten anschließend, sodass für jeden Patienten nur der jeweils erste Krankenhausaufenthalt (= niedrigste `encounter_id`) im Datensatz verbleibt. Überprüfen Sie die korrekte Durchführung der Sortierung und der Filterung, indem Sie einen geeigneten Vorher-Nachher-Vergleich anstellen. Schließen Sie danach alle Einträge aus, bei denen der Wert von `discharge_disposition_id` mit Versterben oder Hospiz in Verbindung steht (Werte `11`, `13`, `14`, `19`, `20`, `21`) und entfernen Sie die für die weitere Modellierung nicht benötigten Spalten `encounter_id` und `patient_nbr`. Erstellen Sie eine Häufigkeitstabelle der Werte von `discharge_disposition_id` und stellen Sie sicher, dass diese mit der Auflistung in Anhang 2 der Prüfungsaufgabe übereinstimmt, sowie dass der Datensatz nun aus 69.973 Zeilen und 48 Spalten besteht.

Zunächst sortieren wir die Daten aufsteigend erst nach `patient_nbr` und dann nach `encounter_id`. Um einen Vorher-Nachher-Vergleich anzustellen, zeigen wir darüber hinaus die ersten 12 Zeilen des sortierten Datensatzes.

```
# Sortiere die Daten aufsteigend nach patient_nbr und encounter_id
data <- data %>% arrange(patient_nbr, encounter_id)

# Zeige die ersten 12 Einträge des sortierten Datensatzes
data %>% select(patient_nbr, encounter_id) %>% head(12)
```

patient_nbr <int>	encounter_id <int>
135	24437208
135	26264286
378	29758806
729	189899286
774	64331490
927	14824206
1152	8380170
1152	30180318
1152	55533660
1152	80742510
1152	83281464
1305	66197028

1-12 of 12 rows

Anschließend filtern wir den Datensatz nach der ersten Einweisung und geben die ersten zehn Zeilen des resultierenden Datensatzes aus.

```r
# Behalte nur die erste Krankenhauseinweisung je Patient
data <- data %>% group_by(patient_nbr) %>% slice_head(n = 1) %>% ungroup()

# Zeige die ersten 7 Einträge des gefilterten Datensatzes
data %>% select(patient_nbr, encounter_id) %>% head(n = 7)
```

patient_nbr <int>	encounter_id <int>
135	24437208
378	29758806
729	189899286
774	64331490
927	14824206
1152	8380170
1305	66197028

7 rows

Aus den beiden oben ausgegebenen Ausschnitten der jeweiligen Datensätze ist erkennbar, dass sowohl die Sortierung als auch die Filterung korrekt durchgeführt wurden.

Als Nächstes werden alle Einträge entfernt, bei denen der Wert von `discharge_disposition_id` mit Tod oder Hospiz in Zusammenhang steht. Danach werden die beiden Spalten `encounter_id` und `patient_nbr` entfernt.

```r
# Filtern der Daten, sodass alle Fälle in Zusammenhang mit Tod oder Hospiz entfernt werden
data <- data %>% filter(!discharge_disposition_id %in% c(11, 13, 14, 19, 20, 21))

# Entferne die beiden für die weitere Modellierung nicht benötigten Spalten
data <- data %>% select(-c(encounter_id, patient_nbr))
```

Nun erzeugen wir eine Häufigkeitstabelle der Werte von `discharge_disposition_id` .

```r
# Erstellung der Häufigkeitstabelle der Werte von discharge_disposition_id
frequency_count <- data %>%
  group_by(discharge_disposition_id) %>%
  summarize(Anzahl = n(), .groups = 'drop') %>%
  arrange(discharge_disposition_id)

# Ausgabe der Häufigkeitstabelle
frequency_count
```

discharge_disposition_id <int>	Anzahl <int>
1	44317
2	1539
3	8784
4	541
5	913
6	8289
7	409
8	73
9	9
10	6
12	2
15	40
16	3
17	8
18	2474
22	1410
23	260
24	25
25	778
27	3
28	90

1-21 of 21 rows

Ein Vergleich mit den Werten aus Anhang 2 zeigt eine vollständige Übereinstimmung.

Abschließend zeigt der folgende Code, dass der Datensatz nun in der Tat aus 69.973 Zeilen und 48 Spalten besteht.

```
# Gebe die Anzahl an Zeilen und Spalten aus
cat("Anzahl Zeilen des Datensatzes:", nrow(data),
    "\nAnzahl Spalten des Datensatzes:", ncol(data))
```

```
## Anzahl Zeilen des Datensatzes: 69973
## Anzahl Spalten des Datensatzes: 48
```

c) Diagnosedaten analysieren: Der betrachtete Datensatz enthält die Primärdiagnose `diag_1` sowie die beiden Zusatzdiagnosen `diag_2` und `diag_3`, jeweils in der nominalen ICD9-Kodierung. Ermitteln Sie für jedes Diagnosemerkmal die Anzahl unterschiedlicher Diagnosen, die häufigste Diagnose und die Anzahl der Diagnosen, die weniger als zehn Mal vorkommen. Erörtern Sie kurz jeweils zwei Herausforderungen, die diese vergleichsweise hohe Anzahl an Kategorien bei der Datenvisualisierung sowie bei der Modellierung mit sich bringen kann.

Im folgenden Code wird eine Funktion definiert, die für eine gegebene Diagnose-Spalte die Anzahl unterschiedlicher Diagnosen, die häufigste Diagnose und die Anzahl der Diagnosen, die weniger als zehn Mal vorkommen, berechnet. Diese Funktion wird anschließend auf alle drei Diagnosefelder angewendet, und die Ergebnisse werden in einer Tabelle übersichtlich dargestellt.

```
# Funktion zur Analyse der Diagnose-Spalten
analyze_diagnosis <- function(column) {

  # 0. Entfernen von Diagnosen mit Häufigkeit 0
  column <- droplevels(column)

  # 1. Häufigkeiten aller tatsächlich vorhandenen Diagnosen
  freq <- table(column)

  # 2. Anzahl unterschiedlicher Diagnosen (nur die mit freq > 0)
  num_unique_diagnoses <- length(freq)

  # 3. Häufigste Diagnose
  most_frequent <- names(freq)[which.max(freq)]

  # 4. Diagnosen, die weniger als 10 Mal auftreten (aber mehr als 0)
  less_than_10 <- sum(freq < 10)

  return(list(
    "Anzahl Diagnosen"          = num_unique_diagnoses,
    "Häufigste Diagnose"        = most_frequent,
    "Anzahl seltene Diagnosen"  = less_than_10
  ))
}

# Analysiere jede Diagnose-Spalte separat
diag_1_results <- analyze_diagnosis(data$diag_1)
diag_2_results <- analyze_diagnosis(data$diag_2)
diag_3_results <- analyze_diagnosis(data$diag_3)

# Erzeuge einen data.frame zur Speicherung der Ergebnisse
results <- data.frame(
  Diagnosemerkmal = c("diag_1", "diag_2", "diag_3"),
  Anzahl_Diagnosen = c(diag_1_results$`Anzahl Diagnosen`,
                       diag_2_results$`Anzahl Diagnosen`,
                       diag_3_results$`Anzahl Diagnosen`),
  Häufigste_Diagnose = c(diag_1_results$`Häufigste Diagnose`,
                         diag_2_results$`Häufigste Diagnose`,
                         diag_3_results$`Häufigste Diagnose`),
  Anzahl_seltene_Diagnosen = c(diag_1_results$`Anzahl seltene Diagnosen`,
                               diag_2_results$`Anzahl seltene Diagnosen`,
                               diag_3_results$`Anzahl seltene Diagnosen`),
  stringsAsFactors = TRUE
)

# Print results
results
```

Diagnosemerkmal <fct>	Anzahl_Diagnosen <int>	Häufigste_Diagnose <fct>	Anzahl_seltene_Diagnosen <int>
diag_1	695	414	331
diag_2	724	250	388
diag_3	757	250	404

3 rows

Bei der Visualisierung von Merkmalen mit sehr vielen unterschiedlichen Ausprägungen, wie etwa hier bei `diag_1`, `diag_2` und `diag_3`, können sich unter anderem folgende Herausforderungen ergeben:

- *Überfrachtete Diagramme:* Eine Vielzahl an Kategorien kann zu überladenen Visualisierungen führen, insbesondere bei Beschriftungen. Bei Balken- oder Kuchendiagrammen sind darüber hinaus häufig viele Balken oder Segmente sichtbar, was die Lesbarkeit und Interpretation der Grafik erschwert. Wichtige Trends oder Muster können dabei leicht übersehen werden.
- *Unzureichende Differenzierung:* Wenn viele Diagnosen nur wenige Datenpunkte haben, gestaltet sich die visuelle Unterscheidung zwischen den Kategorien schwierig. Grafiken, die auf Häufigkeiten basieren, können wichtige Diagnosen in den Hintergrund drängen, was die Aussagekraft der Visualisierung mindert und zu falschen Interpretationen führt.

Hinsichtlich der Modellierung mit Machine-Learning-verfahren können sich unter anderem folgende Herausforderungen ergeben:

- *Sparsity der Daten:* Eine hohe Anzahl an Kategorien führt oft dazu, dass viele Diagnosen sehr selten auftreten. Insbesondere GLMs könnten Schwierigkeiten haben, zuverlässige Schätzungen zu liefern, wenn einige Kategorien nicht genügend Daten für eine robuste Modellierung bieten.
- *Overfitting-Risiko:* Bei der Modellierung mit Machine-Learning-Methoden kann eine Vielzahl an Kategorien das Risiko des Overfittings erhöhen. Modelle könnten beginnen, das Rauschen in den Daten anstatt der zugrunde liegenden Muster zu lernen, was die Generalisierbarkeit der Modelle auf neue, unbekannte Daten einschränkt und die Vorhersagegenauigkeit verringert.

d) Diagnosestammdaten aufbereiten: Lesen Sie die beiden Dateien `CCS_mapping_ICD9.csv` und `CCS_categories_ICD9.csv` aus den Prüfungsunterlagen ein und führen Sie einen geeigneten Left-Join anhand des Merkmals `category_id` durch, um die Informationen zu verknüpfen. Sortieren Sie die resultierenden Daten anschließend nach dem ICD9-kodierten Merkmal `code` und exportieren Sie das Ergebnis als CSV-Datei `icd9_data.csv` . Erstellen Sie zudem eine Häufigkeitstabelle der Diagnosegruppe `group` und stellen Sie sicher, dass diese mit der Auflistung in Anhang 2 der Prüfungsaufgabe übereinstimmt.

Zu Beginn lesen wir die beiden angegebenen Dateien mittels der Funktion `read.csv2` ein. Anschließend führen wir einen Left-Join des Datensatzes aus `CCS_mapping_ICD9.csv` mit dem Datensatz aus `CCS_categories_ICD9.csv` anhand des Merkmals `category_id` durch und sortieren den erhaltenen Datensatz gemäß des Merkmals `code` . Schließlich wird das Ergebnis als CSV-Datei exportiert.

```
# Mitgelieferte Diagnosegruppierungen einlesen
icd9_mapping <- read.csv2(here("2025/2_Datensätze", "CCS_mapping_ICD9.csv"), stringsAsFactors = TRUE)
icd9_categories <- read.csv2(here("2025/2_Datensätze", "CCS_categories_ICD9.csv"), stringsAsFactors = TRUE)

# Left-joinen, nach ICD sortieren und ggf. Mehrfachsätze entfernen
icd9_data <- icd9_mapping %>%
  left_join(icd9_categories, by = "category_id") %>%
  arrange(code) %>%
  distinct()  # hat im vorliegenden Fall keine Auswirkung

# Ergebnisdatei sichern
write.csv2(icd9_data, here("2025/2_Datensätze", "icd9_data.csv"), row.names = FALSE)
#write.csv2(icd9_data, here("C:/ml/data/DiabetesHospital/", "icd9_data.csv"), row.names = FALSE)
```

Schließlich erstellen wir eine Häufigkeitstabelle des Merkmals `group` im soeben erzeugten Datensatz.

```
# Häufigkeitsauszählung der Werte der Diagnosegruppe group
group_frequency_count <- icd9_data %>%
  group_by(group) %>%
  summarize(Anzahl = n(), .groups = 'drop') %>%
  arrange(group)

# Ausgabe der Häufigkeitsauszählung
group_frequency_count
```

group <fct>	**Anzahl** <int>
Circulatory	57
Diabetes	56
Digestive	60
Genitourinary	51
Injury	193
Musculoskeletal	32
n.a.	1
Neoplasms	100
Other	558
Respiratory	55

1-10 of 10 rows

Ein Vergleich mit den Werten aus Anhang 2 zeigt eine vollständige Übereinstimmung.

e) Diagnosegruppen zuspielen: Erstellen Sie zunächst einen Auszug `icd9` aus den aufbereiteten Diagnosegruppendaten, der nur die Spalten `code` und `group` enthält. Entfernen Sie dann sämtliche Punkte (‘.’) aus den Werten der Diagnosemerkmale `diag_1` , `diag_2` und `diag_3` des Hauptdatensatzes. Führen Sie anschließend für jedes der drei Diagnosemerkmale einen Left-Join mit dem erstellten Datensatz `icd9` durch, sodass die zugehörige Diagnosegruppe in einer neuen Spalte (`group_diag_1` , `group_diag_2` , `group_diag_3`) ergänzt wird. Beachten Sie, dass durch den Left-Join bei einigen Diagnosen fehlende Werte entstehen können. Ersetzen Sie diese fehlenden Werte durch den neuen Wert `Other` . Stellen Sie am Ende sicher, dass die Gruppen erfolgreich zugewiesen wurden, indem Sie beispielhaft zwei Zeilen auf ihre Korrektheit hinsichtlich der Zuordnungen der Diagnosegruppen überprüfen. Stellen Sie darüber hinaus sicher, dass alle Merkmale, deren Namen auf `_id` enden, den Datentyp `factor` besitzen.

Hier wird zunächst ein relevanter Auszug aus den aufbereiteten Diagnosegruppen-Daten erstellt, der die Spalten `code` und `group` enthält. Anschließend werden Punkte aus den Diagnosefeldern `diag_1`, `diag_2` und `diag_3` entfernt, bevor für jedes dieser Felder ein Left-Join auf die extrahierten Diagnosegruppen-Daten durchgeführt wird. Fehlende Werte in den neuen Diagnosegruppenspalten werden durch den neuen Wert `Other` ersetzt.

```
# Extrahiere relevante Spalten für den Left-Join
icd9 <- icd9_data %>% select(code, group)

# Entferne Punkte aus den Diagnosefeldern `diag_1`, `diag_2` und `diag_3`
data <- data %>% mutate(across(starts_with("diag_"), ~ gsub("\\.", "", .)))

# Führe Left-Joins für `diag_1`, `diag_2` und `diag_3` durch und benenne Spalten um
data <- data %>%
  left_join(icd9, by = c("diag_1" = "code")) %>%
  rename(group_diag_1 = group) %>%
  left_join(icd9, by = c("diag_2" = "code")) %>%
  rename(group_diag_2 = group) %>%
  left_join(icd9, by = c("diag_3" = "code")) %>%
  rename(group_diag_3 = group) %>%
  mutate(
    diag_1 = as.factor(diag_1),
    diag_2 = as.factor(diag_2),
    diag_3 = as.factor(diag_3)
  )

# Fehlende Werte in den neuen Diagnosegruppenspalten durch 'Other' ersetzen
data <- data %>% mutate(across(starts_with("group_diag_"), ~ replace_na(., "Other")))
```

Schließlich überprüfen wir, ob die Gruppen erfolgreich zugewiesen wurden, indem wir beispielhaft zwei Zeilen aus dem Hauptdatensatz und die entsprechenden Zuordnungen aus dem ICD9-Datensatz betrachten.

```
# Beispielhafte Überprüfung der Zuweisung der Diagnosegruppen
# Wählen Sie zwei zufällige Zeilen aus dem Hauptdatensatz aus
# Die Reproduzierbarkeit wurde durch den Seed gewahrt
sample_data_rows <- data %>%
  select(race, gender, age, diag_1, diag_2, diag_3, group_diag_1, group_diag_2, group_diag_3) %>%
  sample_n(2, )  # Seed wurde zu Beginn in Aufgabe R0 gesetzt

# Ausgabe der beiden zufälligen Einträge
sample_data_rows
```

race <fct>	gender <fct>	age <fct>	diag_1 <fct>	diag_2 <fct>	diag_3 <fct>	group_diag_1 <fct>	group_diag_2 <fct>	group_diag_3 <fct>
Caucasian	Male	[70-80)	414	V45	786	Circulatory	Diabetes	Respiratory
Caucasian	Female	[50-60)	574	276	250	Digestive	Other	Diabetes

2 rows

```
# Wählen Sie die entsprechenden Zeilen aus dem ICD9-Daten-Dataframe basierend auf den Diagnosecodes
sample_icd9_rows <- icd9_data %>%
  select(code, group) %>%
  filter(code %in% unlist(sample_data_rows[c("diag_1", "diag_2", "diag_3")]))

# Ausgabe der relevanten Zuordnungen
sample_icd9_rows
```

code <fct>	group <fct>
250	Diabetes
276	Other
414	Circulatory
574	Digestive
786	Respiratory
V45	Diabetes

6 rows

Bei beiden der oben dargestellten Zeilen wurden den jeweiligen drei Diagnosen die korrekten Diagnosegruppen zugeordnet.

Sicherstellen, dass alle Merkmale, deren Namen auf `_id` enden, den Datentyp `factor` besitzen.

```
# Setze den Datentyp jedes Merkmals, dessen Name auf "_id" endet, auf 'factor'
# und resette die Levels für alle Merkmale
data <- data %>%
  mutate(across(ends_with("_id"), as.factor)) %>%
  mutate(across(everything(), ~ if (is.factor(.x)) droplevels(.x) else .x))
```

f) Objektmerkmale bereinigen: Führen Sie die folgenden Schritte für jedes Merkmal vom Datentyp `factor` durch, mit Ausnahme der Merkmale `diag_1` , `diag_2` und `diag_3` : Überprüfen Sie, ob ein Merkmalswert im Datensatz weniger als zehn Mal vorkommt. Falls dies der Fall ist, ersetzen Sie ihn durch den häufigsten Wert des jeweiligen Merkmals. Erstellen Sie anschließend für jedes betroffene Merkmal eine Übersicht, die den Namen des Merkmals, den häufigsten Wert sowie die ersetzten Werte enthält. Analysieren Sie abschließend die Auswirkungen dieser Bereinigung auf die Anwendung von Machine-Learning-Modellen, insbesondere in Bezug auf Modellstabilität, Performance und Lauffähigkeit. Gehen Sie dabei jeweils auf zwei relevante Auswirkungen pro Aspekt ein (und nutzen Sie – bei Bedarf – die Logistische Regression als Beispiel sowie Erkenntnisse aus Aufgabe R3, um Ihre Argumentation zu stützen).

Zunächst verarbeiten wir alle Merkmale vom Typ `factor im Datensatz` , mit Ausnahme der Merkmale, deren Name mit `diag_` beginnt. Für jedes betroffene Merkmal führen wir folgende Schritte durch: Zählen der Häufigkeiten der Merkmalswerte, Ermitteln des häufigsten Werts des Merkmals und Identifizieren der Werte, die weniger als zehn Mal vorkommen. Diese seltenen Werte werden dann durch den häufigsten Wert ersetzt, wobei die ersetzten Werte protokolliert werden.

```
# Jedes Merkmal vom Typ "factor" (außer solche, deren Name mit "diag_" beginnt) verarbeiten
data <- data %>%
  mutate(across(
    where(is.factor),
    ~ {
      if (!startsWith(cur_column(), "diag_")) {
        # Häufigkeiten der Merkmalswerte zählen
        count_table <- table(.x)

        # Häufigsten Wert identifizieren
        most_frequent_value <- names(which.max(count_table))

        # Werte mit einer Häufigkeit von weniger als 10 identifizieren
        rare_values <- names(count_table[count_table < 10])

        if (length(rare_values) > 0) {
          # Seltene Werte durch den häufigsten Wert ersetzen und protokollieren
          cat(sprintf(
            "Im Merkmal '%s' wurden die folgenden seltenen Werte durch '%s' ersetzt:\n",
            cur_column(), most_frequent_value
          ))
          cat(paste(" -", rare_values, "\n", collapse = ""))

          # Rückgabe: Werte ersetzen und Faktor-Level beibehalten
          factor(
            replace(as.character(.x), .x %in% rare_values, most_frequent_value),
            levels = levels(.x)
          )
        } else {
          # Wenn keine seltenen Werte, einfach zurückgeben
          .x
        }
      } else {
        # Für "diag_"-Merkmale keine Änderung vornehmen
        .x
      }
    }
  ))
```

```
## Im Merkmal 'gender' wurden die folgenden seltenen Werte durch 'Female' ersetzt:
##  - Unknown/Invalid
## Im Merkmal 'weight' wurden die folgenden seltenen Werte durch '?' ersetzt:
##  - [175-200)
##  - >200
## Im Merkmal 'admission_type_id' wurden die folgenden seltenen Werte durch '1' ersetzt:
##  - 4
## Im Merkmal 'discharge_disposition_id' wurden die folgenden seltenen Werte durch '1' ersetzt:
##  - 9
##  - 10
##  - 12
##  - 16
##  - 17
##  - 27
## Im Merkmal 'admission_source_id' wurden die folgenden seltenen Werte durch '7' ersetzt:
##  - 10
##  - 11
##  - 13
##  - 14
##  - 22
##  - 25
## Im Merkmal 'payer_code' wurden die folgenden seltenen Werte durch '?' ersetzt:
##  - FR
## Im Merkmal 'medical_specialty' wurden die folgenden seltenen Werte durch '?' ersetzt:
##  - AllergyandImmunology
##  - Anesthesiology
##  - Cardiology-Pediatric
##  - DCPTEAM
##  - Dentistry
##  - Dermatology
##  - Endocrinology-Metabolism
##  - Neurophysiology
##  - OutreachServices
##  - Pathology
##  - Pediatrics-EmergencyMedicine
##  - Pediatrics-Hematology-Oncology
##  - Pediatrics-Neurology
##  - Pediatrics-Pulmonology
##  - Perinatology
##  - PhysicianNotFound
##  - Proctology
##  - Psychiatry-Addictive
##  - Psychiatry-Child/Adolescent
##  - Resident
##  - Speech
##  - SportsMedicine
##  - Surgery-Colon&Rectal
##  - Surgery-Maxillofacial
##  - Surgery-Pediatric
##  - Surgery-PlasticwithinHeadandNeck
## Im Merkmal 'nateglinide' wurden die folgenden seltenen Werte durch 'No' ersetzt:
##  - Down
## Im Merkmal 'chlorpropamide' wurden die folgenden seltenen Werte durch 'No' ersetzt:
##  - Down
##  - Up
## Im Merkmal 'acetohexamide' wurden die folgenden seltenen Werte durch 'No' ersetzt:
##  - Steady
## Im Merkmal 'miglitol' wurden die folgenden seltenen Werte durch 'No' ersetzt:
##  - Down
##  - Up
## Im Merkmal 'troglitazone' wurden die folgenden seltenen Werte durch 'No' ersetzt:
##  - Steady
## Im Merkmal 'glyburide.metformin' wurden die folgenden seltenen Werte durch 'No' ersetzt:
##  - Down
##  - Up
## Im Merkmal 'glipizide.metformin' wurden die folgenden seltenen Werte durch 'No' ersetzt:
##  - Steady
## Im Merkmal 'metformin.rosiglitazone' wurden die folgenden seltenen Werte durch 'No' ersetzt:
##  - Steady
## Im Merkmal 'metformin.pioglitazone' wurden die folgenden seltenen Werte durch 'No' ersetzt:
##  - Steady
```

Die durchgeführten Datenbereinigungen beeinflussen die Anwendung von Machine-Learning-Modellen, insbesondere der Logistischen Regression, in mehreren Aspekten:

Modellstabilität:

- Durch das Zusammenfassen seltener Werte wird die Modellstabilität erhöht, da das Modell auf einer größeren Datenbasis trainiert wird und robustere Schätzungen der Parameter ermöglicht.
- Eine Übervereinfachung kann jedoch zu Informationsverlust führen, da potenziell wichtige Muster oder Beziehungen, die in seltenen Kategorien existieren, nicht mehr abgebildet werden.

Performance:

- Die Bereinigung kann die Modellgenauigkeit verbessern, indem weniger spärliche Werte berücksichtigt werden, die andernfalls zu Overfitting führen könnten.

- Andererseits kann die Performance beeinträchtigt werden, wenn durch die Zusammenfassung relevante Informationen verloren gehen, die für die Vorhersagekraft wichtig sind.

Lauffähigkeit:

- Die Reduktion seltener Werte vereinfacht das Modell, spart Speicher und Rechenzeit, und führt so i. A. zu einer schnelleren Laufzeit.
- Seltene Werte können nur im Trainings- oder Testdatensatz auftreten, was bei der Aufteilung der Daten zu Problemen führen kann. Einige Modelle werfen Fehler, wenn sie auf Testdaten mit unbekannten Werten angewendet werden.

(Beispiel mit Logistischer Regression: Die Bereinigung reduziert potenzielle Multikollinearitätsprobleme, die durch seltene Kategorien mit wenigen Beobachtungen entstehen könnten. Die modellierten Odds Ratios sind dadurch konsistenter und zuverlässiger. Außerdem konnte die Logistische Regression in Aufgabe R3 schneller trainiert werden und zeigte stabilere Ergebnisse.)

g) Datensätze für binäre und ternäre Klassifikation erzeugen: Im Folgenden werden zwei Datensätze für die späteren Aufgaben erstellt: Ein Datensatz `diabetic_data_ter` für die ternäre Klassifikation von Krankenhaus-Wiedereinweisungen und ein Datensatz `diabetic_data_bin` für die binäre Klassifikation von Krankenhaus-Wiedereinweisungen.

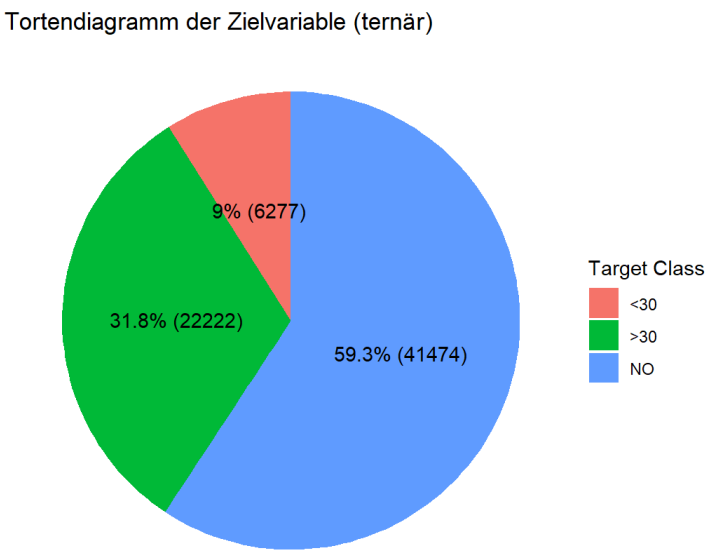
- **Datensatz `diabetic_data_ter`: Ersetzen Sie das Merkmal `readmitted` durch ein neues Merkmal `TARGET` mit dem Datentyp `factor`. Entfernen Sie zudem bei allen `factor`-Merkmalen die Levels, die nach den bisherigen Filterungen nicht mehr auftreten. Visualisieren Sie die Verteilung der ternären Zielvariable `TARGET` mithilfe eines Tortendiagramms.**
- **Datensatz `diabetic_data_bin`: Basierend auf dem Datensatz `diabetic_data_ter` filtern Sie alle Zeilen heraus, in denen `TARGET` den Wert `>30` aufweist. Anschließend ändern Sie das Merkmal `TARGET` unter Beibehaltung des Datentyps `factor` wie folgt: `TARGET` erhält den Wert `1`, wenn der Patient innerhalb von 30 Tagen erneut aufgenommen wurde (bisheriger `TARGET`-Wert: `<30`), und andernfalls den Wert `0`. Entfernen Sie zudem bei allen `factor`-Merkmalen die Levels, die nach den bisherigen Filterungen nicht mehr auftreten. Visualisieren Sie die Verteilung der binären Zielvariable `TARGET` mithilfe eines Tortendiagramms. Diskutieren Sie jeweils kurz die Imbalance dieser Zielvariable sowie deren potenzielle Auswirkungen auf die spätere Modellierung. Geben Sie die ersten zehn Zeilen des aufbereiteten Datensatzes aus und exportieren Sie anschließend den gesamten Datensatz als CSV-Datei unter dem Namen `diabetic_data_bin.csv`.**

Zunächst erstellen wir den Datensatz zur ternären Klassifikation gemäß den Schritten aus der Aufgabenstellung.

```
# Datensatz für ternäre Klassifikation erstellen (diabetic_data_ter)
# Ersetzen des Merkmals 'readmitted' durch 'TARGET' und setzen als Faktor
diabetic_data_ter <- data %>%
  mutate(TARGET = as.factor(case_when(
    readmitted == "NO" ~ "NO",
    readmitted == ">30" ~ ">30",
    readmitted == "<30" ~ "<30"
  ))) %>%
  select(-readmitted) %>%
  mutate(across(everything(), ~ if (is.factor(.x)) droplevels(.x) else .x))
```

Anschließend visualisierung wir die Verteilung der Zielvariablen in einem Tortendiagramm.

```
# Visualisierung der Verteilung der Zielvariablen 'TARGET' im ternären Datensatz
target_dist_ter <- diabetic_data_ter %>%
  group_by(TARGET) %>%
  summarize(count = n()) %>%
  mutate(percentage = count / sum(count) * 100)

ggplot(target_dist_ter, aes(x = "", y = count, fill = TARGET)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y") +
  geom_text(aes(label = paste0(round(percentage, 1), "% (", count, ")")),
            position = position_stack(vjust = 0.5)) +
  labs(fill = "Target Class", title = "Tortendiagramm der Zielvariable (ternär)") +
  theme_void() +
  theme(legend.position = "right")
```

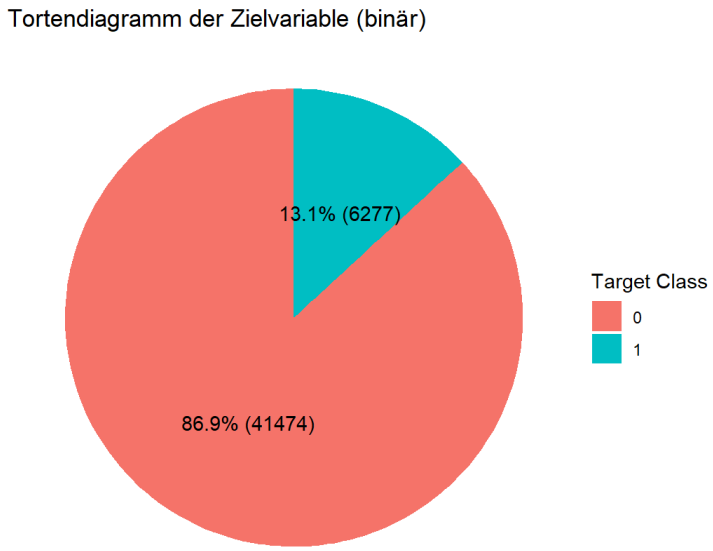


Als Nächstes erzeugen wir den Datensatz zur binären Klassifikation gemäß den Schritten in der Aufgabenstellung.

```r
# Datensatz für binäre Klassifikation erstellen (data_bin)
# Basierend auf diabetic_data_ter: Entfernen von '>30' und Zielvariable für binäre Klassifikation anpassen
diabetic_data_bin <- diabetic_data_ter %>%
  filter(TARGET != ">30") %>%
  mutate(TARGET = ifelse(TARGET == "<30", 1, 0)) %>%
  mutate(across(everything(), ~ if (is.factor(.x)) droplevels(.x) else .x))

# Zielvariable 'TARGET' als Faktor setzen
diabetic_data_bin$TARGET <- as.factor(diabetic_data_bin$TARGET)
```

Auch hier erzeugen wir ein Tortendiagramm zur Visualisierung der Verteilung der Zielvariablen.

```r
# Visualisierung der Verteilung der Zielvariablen 'TARGET' im binären Datensatz
target_dist_bin <- diabetic_data_bin %>%
  group_by(TARGET) %>%
  summarize(count = n()) %>%
  mutate(percentage = count / sum(count) * 100)

ggplot(target_dist_bin, aes(x = "", y = count, fill = TARGET)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y") +
  geom_text(aes(label = paste0(round(percentage, 1), "% (", count, ")")),
            position = position_stack(vjust = 0.5)) +
  labs(fill = "Target Class", title = "Tortendiagramm der Zielvariable (binär)") +
  theme_void() +
  theme(legend.position = "right")
```



Die Zielvariable `TARGET` zeigt eine klare Imbalance: Rund 87% der Fälle gehören zur Klasse `0` (keine Wiedereinweisung), während nur etwa 13% der Fälle die Klasse `1` (Wiedereinweisung) aufweisen.

Diese Ungleichverteilung kann dazu führen, dass Machine-Learning-Modelle die Mehrheit der Fälle bevorzugen und Schwierigkeiten haben, die Minderheitsklasse korrekt zu klassifizieren. Traditionelle Metriken wie Genauigkeit sind in diesem Fall i. d. R. nicht aussagekräftig. Stattdessen sollte auf Metriken wie die Fläche unter der ROC-Kurve (AUC) zurückgegriffen werden, um die Leistung der betrachteten Modelle besser zu bewerten. Zur Behandlung der Klassen-Imbalance könnten Methoden wie Oversampling der Minderheitsklasse oder Undersampling der Mehrheitsklasse sowie der Einsatz spezieller, auf imbalancierte Probleme zugeschnittene Machine-Learning-Algorithmen verwendet werden.

Zum Abschluss der Aufgabe geben wir die ersten zehn Zeilen aus und speichern den kompletten Datensatz als CSV-Datei.

```r
# Gebe die ersten 10 Zeilen des aufbereiteten Datensatzes aus
head(diabetic_data_bin, n = 10)
```

race <fct>	**gender** <fct>	**age** <fct>	**weight** <fct>	**admission_type_id** <fct>	**discharge_disposition_id** <fct>
Caucasian	Female	[50-60)	?	2	1
Caucasian	Female	[50-60)	?	3	1
Caucasian	Female	[80-90)	?	1	3
Caucasian	Female	[80-90)	?	1	1
AfricanAmerican	Female	[30-40)	?	1	1
Caucasian	Female	[60-70)	?	3	1
Caucasian	Female	[70-80)	?	2	3
Caucasian	Male	[60-70)	?	2	1
Caucasian	Female	[70-80)	?	3	1
Caucasian	Female	[80-90)	?	1	3

1-10 of 10 rows | 1-6 of 51 columns

```
# Speichere die resultierenden Daten
write.csv(diabetic_data_bin, "diabetic_data_bin.csv", row.names = FALSE)
```

Aufgabe R2: Explorative Datenanalyse und Visualisierung

In dieser Aufgabe sollen Code-Passagen aus der Vorlage *SWoF* entnommen und an einigen Stellen angepasst werden. Ziel der Anpassungen ist es, den Code mit minimalem Aufwand lauffähig zu machen, wobei der Datensatz `diabetic_data_bin` , der zum Abschluss von Aufgabe R1 erstellt wurde, als Input dient. Zu Beginn jeder der folgenden Teilaufgaben wird in Fettdruck angegeben, auf welche Abschnitte der Vorlage *SWoF* sich die jeweilige Teilaufgabe bezieht.

Hinweis: Bei allen in dieser Aufgabe erstellten Grafiken ist auf eine klare, konsistente und gut lesbare Darstellung zu achten. Bei Nichteinhaltung werden Punkte abgezogen.

a) 3 Overview: Wenden Sie die Funktionen `summary` und `glimpse` auf den Datensatz `diabetic_data_bin` an und diskutieren Sie die erhaltenen Resultate kurz.

Zunächst wenden wir die in R integrierte Funktion `summary` auf den Datensatz `diabetic_data_bin` an.

```
summary(diabetic_data_bin)
```

```
##               race          gender           age              weight
##  ?              : 1483   Female:25152   [70-80):11623   ?        :46283
##  AfricanAmerican: 8757   Male  :22599   [60-70):10736   [75-100) :  589
##  Asian          :  381                  [50-60): 8615   [50-75)  :  428
##  Caucasian      :35203                  [80-90): 7242   [100-125):  304
##  Hispanic       : 1088                  [40-50): 4885   [125-150):   70
##  Other          :  839                  [30-40): 2003   [25-50)  :   43
##                                         (Other): 2647   (Other)  :   34
##  admission_type_id discharge_disposition_id admission_source_id
##  1:24036           1      :30258            7      :24540
##  2: 8731           3      : 6029            1      :15312
##  3:10056           6      : 5189            17     : 3090
##  5: 2047           18     : 1921            4      : 2021
##  6: 2641           2      : 1101            6      : 1479
##  7:   18           22     : 1073            2      :  673
##  8:  222           (Other): 2180            (Other):  636
##  time_in_hospital   payer_code                medical_specialty
##  Min.   : 1.000   ?      :20378   ?                    :22642
##  1st Qu.: 2.000   MC     :13115   InternalMedicine     : 7341
##  Median : 3.000   HM     : 2821   Family/GeneralPractice: 3312
##  Mean   : 4.205   BC     : 2601   Emergency/Trauma     : 2875
##  3rd Qu.: 6.000   SP     : 2259   Cardiology           : 2811
##  Max.   :14.000   MD     : 1522   Surgery-General      : 1528
##                   (Other): 5055   (Other)              : 7242
##  num_lab_procedures num_procedures  num_medications number_outpatient
##  Min.   :  1.00     Min.   :0.000   Min.   : 1.00   Min.   : 0.000
##  1st Qu.: 30.00     1st Qu.:0.000   1st Qu.:10.00   1st Qu.: 0.000
##  Median : 44.00     Median :1.000   Median :14.00   Median : 0.000
##  Mean   : 42.36     Mean   :1.458   Mean   :15.56   Mean   : 0.233
##  3rd Qu.: 56.00     3rd Qu.:2.000   3rd Qu.:20.00   3rd Qu.: 0.000
##  Max.   :132.00     Max.   :6.000   Max.   :81.00   Max.   :36.000
##
##  number_emergency   number_inpatient      diag_1          diag_2
##  Min.   : 0.00000   Min.   : 0.0000   414    : 3641   250    : 3748
##  1st Qu.: 0.00000   1st Qu.: 0.0000   428    : 2219   276    : 3047
##  Median : 0.00000   Median : 0.0000   786    : 2063   428    : 2589
##  Mean   : 0.08157   Mean   : 0.1383   410    : 2006   401    : 2309
##  3rd Qu.: 0.00000   3rd Qu.: 0.0000   486    : 1488   427    : 2283
##  Max.   :37.00000   Max.   :12.0000   715    : 1418   599    : 1495
##                                       (Other):34916   (Other):32280
##     diag_3      number_diagnoses max_glu_serum A1Cresult     metformin
##  250    : 6522   Min.   : 1.000   >200:  608    >7  : 2003   Down  :  308
##  401    : 4787   1st Qu.: 5.000   >300:  429    >8  : 4279   No    :37346
##  276    : 2233   Median : 8.000   None:45565    None:38767   Steady: 9513
##  414    : 1748   Mean   : 7.102   Norm: 1149    Norm: 2702   Up    :  584
##  428    : 1695   3rd Qu.: 9.000
##  427    : 1646   Max.   :16.000
##  (Other):29120
##  repaglinide    nateglinide    chlorpropamide glimepiride    acetohexamide
##  Down  :   17   No    :47440   No    :47710   Down  :   92   No:47751
##  No    :47174   Steady:  300   Steady:   41   No    :45262
##  Steady:  505   Up    :   11                  Steady: 2233
##  Up    :   55                                 Up    :  164
##
##
##
##   glipizide      glyburide     tolbutamide    pioglitazone   rosiglitazone
##  Down  :  234   Down  :  266   No    :47739   Down  :   46   Down  :   56
##  No    :41845   No    :42500   Steady:   12   No    :44308   No    :44750
##  Steady: 5293   Steady: 4571                  Steady: 3287   Steady: 2850
##  Up    :  379   Up    :  414                  Up    :  110   Up    :   95
##
##
##
##    acarbose       miglitol     troglitazone  tolazamide    examide
##  No    :47638   No    :47743   No:47751     No    :47730   No:47751
##  Steady:  107   Steady:    8                Steady:   21
##  Up    :    6
##
##
##
##
##  citoglipton   insulin      glyburide.metformin glipizide.metformin
##  No:47751    Down  : 4776   No    :47433        No:47751
##              No    :23741   Steady:  318
##              Steady:14795
##              Up    : 4439
##
##
##
##  glimepiride.pioglitazone metformin.rosiglitazone metformin.pioglitazone
##  No:47751                 No:47751                No:47751
##
##
##
```

```
##
##
##
##  change     diabetesMed      group_diag_1          group_diag_2
##  Ch:21017   No :12040   Circulatory:14395   Circulatory  :14079
##  No:26734   Yes:35711   Other      : 8104   Other        :12432
##                         Respiratory: 6032   Diabetes     : 7141
##                         Digestive  : 4229   Respiratory  : 4461
##                         Diabetes   : 3895   Genitourinary: 3612
##                         Injury     : 3374   Digestive    : 1830
##                         (Other)    : 7722   (Other)      : 4196
##         group_diag_3   TARGET
##  Circulatory  :13433   0:41474
##  Other        :13108   1: 6277
##  Diabetes     : 9444
##  Respiratory  : 3017
##  Genitourinary: 2676
##  Digestive    : 1644
##  (Other)      : 4429
```

`summary` zeigt sowohl für die acht numerischen als auch für die 43 kategoriellen Merkmale grundlegende Statistiken. Für kategorielle Merkmale (wie z. B. `gender`) werden die Häufigkeiten jeder Kategorie (ggf. mit einer den Rest zusammenfassenden Kategorie `(Other)`) ausgegeben. Bei numerischen Merkmalen (wie z. B. `time_in_hospital`), werden Minimum, Maximum, Mittelwert und Quartile angezeigt. Die meisten Merkmale folgen keiner gleichmäßigen Verteilung, sondern weisen eine hohe Streuung auf; andererseits existerien nach der Datenaufbereitung auch Merkmale, welche nur einen einzigen Wert annehmen (z. B. `examide`).

Als Nächstes wenden wir die Funktion `glimpse` aus dem `dplyr`-Paket auf den Datensatz `diabetic_data_bin` an.

```
glimpse(diabetic_data_bin)
```

```
## Rows: 47,751
## Columns: 51
## $ race                     <fct> Caucasian, Caucasian, Caucasian, Caucasian, A…
## $ gender                   <fct> Female, Female, Female, Female, Female, Femal…
## $ age                      <fct> [50-60), [50-60), [80-90), [80-90), [30-40), …
## $ weight                   <fct> ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, …
## $ admission_type_id        <fct> 2, 3, 1, 1, 1, 3, 2, 2, 3, 1, 1, 1, 3, 3, 1, …
## $ discharge_disposition_id <fct> 1, 1, 3, 1, 1, 1, 3, 1, 1, 3, 1, 1, 5, 1, 1, …
## $ admission_source_id      <fct> 1, 1, 7, 7, 7, 1, 1, 1, 1, 7, 7, 7, 1, 1, 7, …
## $ time_in_hospital         <int> 8, 2, 4, 3, 5, 9, 12, 8, 1, 7, 4, 1, 4, 6, 4,…
## $ payer_code               <fct> ?, ?, MC, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,…
## $ medical_specialty        <fct> Cardiology, Surgery-Neuro, InternalMedicine, …
## $ num_lab_procedures       <int> 77, 49, 68, 46, 49, 52, 47, 57, 31, 77, 47, 3…
## $ num_procedures           <int> 6, 1, 2, 0, 0, 1, 2, 6, 1, 0, 4, 5, 2, 2, 3, …
## $ num_medications          <int> 33, 11, 23, 20, 5, 16, 18, 31, 9, 12, 16, 13,…
## $ number_outpatient        <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
## $ number_emergency         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
## $ number_inpatient         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
## $ diag_1                   <fct> 401, 722, 820, 274, 590, 491, 682, 414, 722, …
## $ diag_2                   <fct> 997, 305, 493, 427, 220, 428, 707, 584, 25001…
## $ diag_3                   <fct> 560, 250, E880, 416, 250, 25001, 560, 285, 41…
## $ number_diagnoses         <int> 8, 3, 9, 9, 3, 9, 9, 9, 7, 9, 7, 8, 9, 7, 6, …
## $ max_glu_serum            <fct> None, None, None, None, None, None, None, Non…
## $ A1Cresult                <fct> None, None, >7, >8, None, None, Norm, >8, Non…
## $ metformin                <fct> Steady, No, Steady, Steady, No, No, No, No, N…
## $ repaglinide              <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ nateglinide              <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ chlorpropamide           <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ glimepiride              <fct> No, No, No, No, Steady, No, Down, No, No, No,…
## $ acetohexamide            <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ glipizide                <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ glyburide                <fct> Down, No, No, Steady, No, No, No, No, No, No,…
## $ tolbutamide              <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ pioglitazone             <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ rosiglitazone            <fct> No, No, No, No, No, No, No, No, No, No, No, S…
## $ acarbose                 <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ miglitol                 <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ troglitazone             <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ tolazamide               <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ examide                  <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ citoglipton              <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ insulin                  <fct> Steady, No, No, No, No, Steady, Steady, Stead…
## $ glyburide.metformin      <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ glipizide.metformin      <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ glimepiride.pioglitazone <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ metformin.rosiglitazone  <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ metformin.pioglitazone   <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ change                   <fct> Ch, No, No, Ch, No, No, Ch, No, No, No, No, C…
## $ diabetesMed              <fct> Yes, No, Yes, Yes, Yes, Yes, Yes, Yes, No, Ye…
## $ group_diag_1             <fct> Circulatory, Musculoskeletal, Injury, Other, …
## $ group_diag_2             <fct> Injury, Other, Respiratory, Circulatory, Neop…
## $ group_diag_3             <fct> Digestive, Diabetes, Other, Circulatory, Diab…
## $ TARGET                   <fct> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
```

`glimpse` gibt einen schnellen Überblick über die Struktur des Datensatzes: Anzahl der Zeilen, Spalten und Datentypen. Es zeigt uns, dass alle kategoriellen Merkmale als `factor` und alle numerischen Merkmale – inhaltlich korrekt – als `integer` vorliegen. Fehlende Werte (z. B. in `weight`) sind sofort erkennbar.

b) 4 Individual feature visualisations: In der Vorlage *SWoF* wurden die im dortigen Datensatz verwendeten Merkmale in verschiedene Gruppen unterteilt (Abschnitte 4.1 bis 4.9). Entscheiden Sie sich für eine sinnvolle Einteilung der Merkmale des Datensatzes `diabetic_data_bin` (exklusive `diag_1`, `diag_2`, `diag_3`) in vier bis sieben Gruppen und begründen Sie diese. Erstellen Sie je Gruppe zu den jeweiligen Merkmalen Histogramme und Kerndichteschätzerplots, die denjenigen aus den Abschnitten 4.1 bis 4.9 der Vorlage entsprechen. Verwenden Sie dabei die Funktion `multiplot` und nutzen Sie eine logarithmierte y-Achse, wo erforderlich. Diskutieren Sie die wichtigsten Erkenntnisse aus den Visualisierungen, insbesondere mit Blick auf deren Bedeutung für die Vorhersage der Zielvariablen `TARGET`.

Die Merkmale des Datensatzes werden für die weitere Verarbeitung wie folgt thematisch gruppiert:

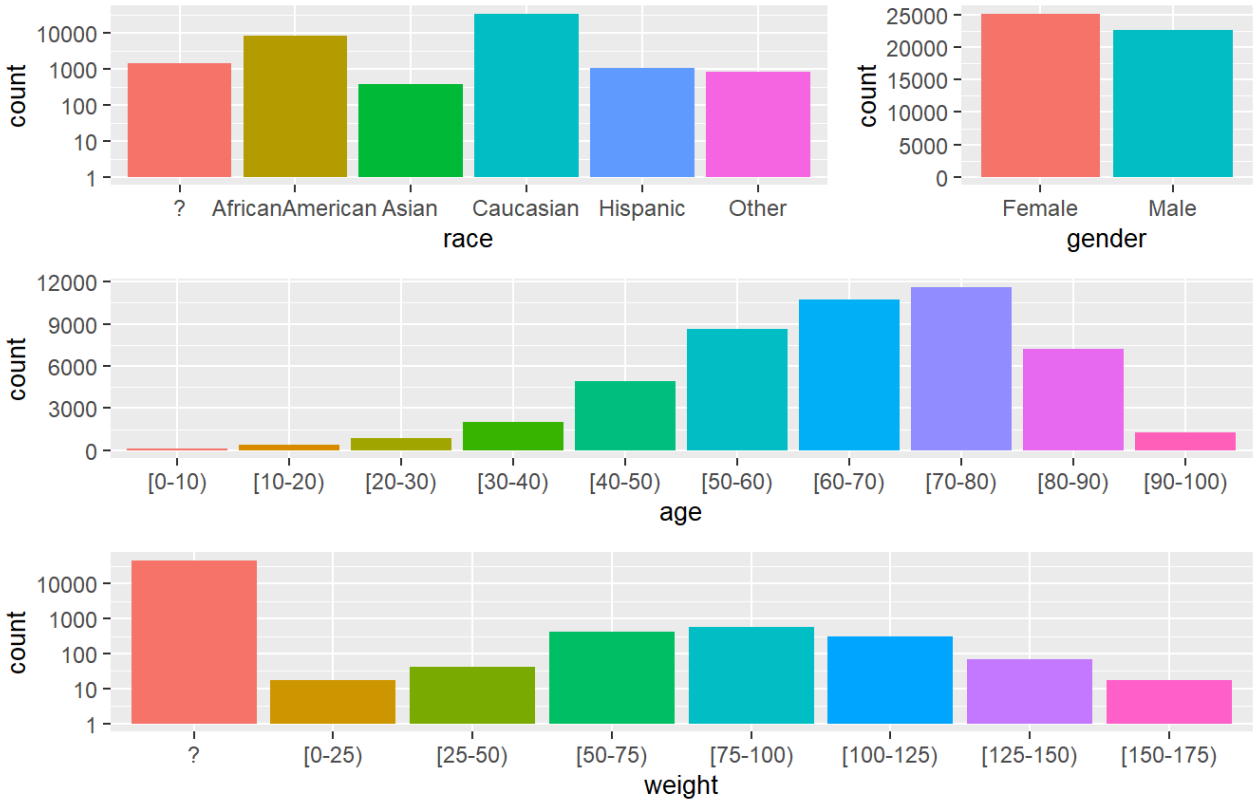
- *Kategorielle Merkmale zu persönlichen Informationen* (z. B. Geschlecht, Alter)
- *Kategorielle Merkmale zu Krankenhausaufenthaltsdetails* (z. B. Aufnahme- und Entlassungsmodalitäten)
- *Kategorielle Merkmale zu Versicherungs- und medizinischen Informationen* (z. B. Fachrichtung, Versicherungsdetails)
- *Kategorielle Merkmale zu Diagnosegruppen*
- *Kategorielle Merkmale zu diabetesbezogenen Merkmalen und Medikation* (z. B. Testergebnisse, Medikamenteneinsatz)
- *Numerische Merkmale zu Krankenhausressourcennutzung und Gesundheitszustand* (z. B. Anzahl der Prozeduren, Verweildauer im Krankenhaus)

Die vorgeschlagene Gruppierung ist sinnvoll, da sie eine thematische Trennung der Merkmale gewährleistet und verwandte Attribute zusammenfasst. Dies erleichtert sowohl die Interpretation der Ergebnisse als auch die gezielte Analyse der Variablen mit Blick auf die Zielvariable `TARGET`.

Kategorielle Merkmale zu persönlichen Informationen

Zunächst visualisieren wir die Histogramme der Merkmale, die persönliche Informationen der Patienten enthalten: `race`, `gender`, `age` und `weight`. Für die Merkmale `race` und `weight` wurde dabei eine logarithmierte y-Achse verwendet, da diese Merkmale jeweils eine starke Ungleichverteilung aufweisen.

```
# Gemeinsames Theme mit einheitlichen Schriftgrößen
common_theme <- theme(
  text = element_text(size = 10),             # Allgemeine Textgröße
  axis.title = element_text(size = 10),       # Schriftgröße der Achsentitel
  axis.text = element_text(size = 9),         # Schriftgröße der Achsenbeschriftungen
  legend.position = "none"                    # Legende deaktivieren
)

p1 <- diabetic_data_bin %>%
  ggplot(aes(race, fill = race)) +
  geom_bar() +
  scale_y_log10() +
  common_theme

p2 <- diabetic_data_bin %>%
  ggplot(aes(gender, fill = gender)) +
  geom_bar() +
  common_theme

p3 <- diabetic_data_bin %>%
  ggplot(aes(age, fill = age)) +
  geom_bar() +
  common_theme

# Definieren der korrekten Reihenfolge für die Gewichtskategorien
weight_levels <- c("?", "[0-25)", "[25-50)", "[50-75)", "[75-100)", "[100-125)", "[125-150)", "[150-175)", "[175-200)")

# Umwandlung von 'weight' in einen Faktor mit der obigen Reihenfolge
diabetic_data_bin$weight <- factor(diabetic_data_bin$weight, levels = weight_levels)

p4 <- diabetic_data_bin %>%
  ggplot(aes(weight, fill = weight)) +
  geom_bar() +
  scale_y_log10() +
  common_theme

# Erzeugen des Layouts und Verwenden der multiplot-Funktion
layout <- matrix(c(1,1,2,3,3,3,4,4,4),3,3,byrow=TRUE)
multiplot(p1, p2, p3, p4, layout=layout)
```



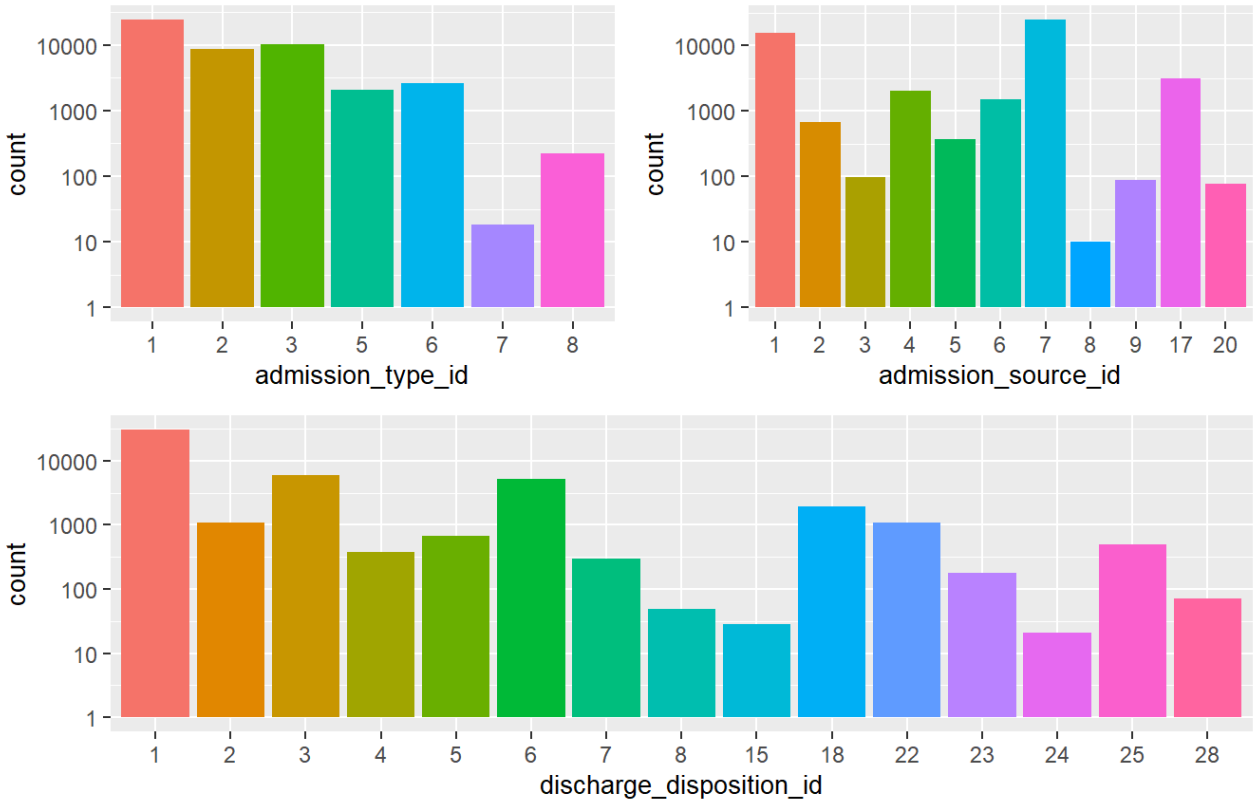
Die wichtigsten Erkenntnisse zu den einzelnen Merkmalen sind wie folgt:

- `race` : Der Großteil der Daten entfällt auf die Ausprägung `Caucasian` , gefolgt mit deutlichem Abstand von `AfricanAmerican` . Die verbleibenden Kategorien zeigen eine sehr geringe Häufigkeit. Es wäre sinnvoll zu prüfen, ob und wie genau ein Unterschied zwischen den Ausprägungen `?` und `Other` besteht.
- `gender` : Dieses Merkmal ist nahezu gleichmäßig zwischen `Female` und `Male` verteilt, mit einem leichten Überhang zugunsten von `Female` . Hinweis: Die seltene Ausprägung `Unknown/Invalid` wurde aufgrund der Datenaufbereitung in Aufgabe R1 entfernt.
- `age` : Die Verteilung auf die sortierten Altersgruppen weist eine leichte Linksschiefe auf: Die höheren Altersgruppen (z. B. `[70-80)`) treten am häufigsten auf, während die Häufigkeit in den jüngeren Altersgruppen allmählich abnimmt und in den sehr alten Altersgruppen rascher zurückgeht. Dies deutet darauf hin, dass die Mehrheit der Patienten älter ist, während Kinder und Jugendliche in der Stichprobe deutlich unterrepräsentiert sind.
- `weight` : Der bei weitem größte Teil der Daten (über 90 %) besitzt die Ausprägung `?` , was fehlende Werte signalisiert. Innerhalb der verbleibenden Werte ist die Verteilung auf die sortierten Gewichtsgruppen unimodal und konzentriert sich auf die mittleren Gewichtsgruppen. Der hohe Anteil fehlender Werte lässt vermuten, dass `weight` nur eine geringe oder keine Relevanz für die Prognose der Zielvariablen `TARGET` haben wird.

Kategorielle Merkmale zu Krankenhausaufenthaltsdetails

Nun visualisieren wir die Histogramme der Merkmale, die Details zu den Krankenhausaufenthalten der Patienten beschreiben: `admission_type_id` , `admission_source_id` und `discharge_disposition_id` . Für alle Merkmale wurde eine logarithmierte y-Achse verwendet, um der starken Ungleichverteilung Rechnung zu tragen.

```
p1 <- diabetic_data_bin %>%
  ggplot(aes(admission_type_id, fill = admission_type_id)) +
  geom_bar() +
  scale_y_log10() +
  common_theme

p2 <- diabetic_data_bin %>%
  ggplot(aes(admission_source_id, fill = admission_source_id)) +
  geom_bar() +
  scale_y_log10() +
  common_theme

p3 <- diabetic_data_bin %>%
  ggplot(aes(discharge_disposition_id, fill = discharge_disposition_id)) +
  geom_bar() +
  scale_y_log10() +
  common_theme

# Erzeugen des Layouts und Verwenden der multiplot-Funktion
layout <- matrix(c(1,2,3,3),2,2,byrow=TRUE)
multiplot(p1, p2, p3, layout=layout)
```



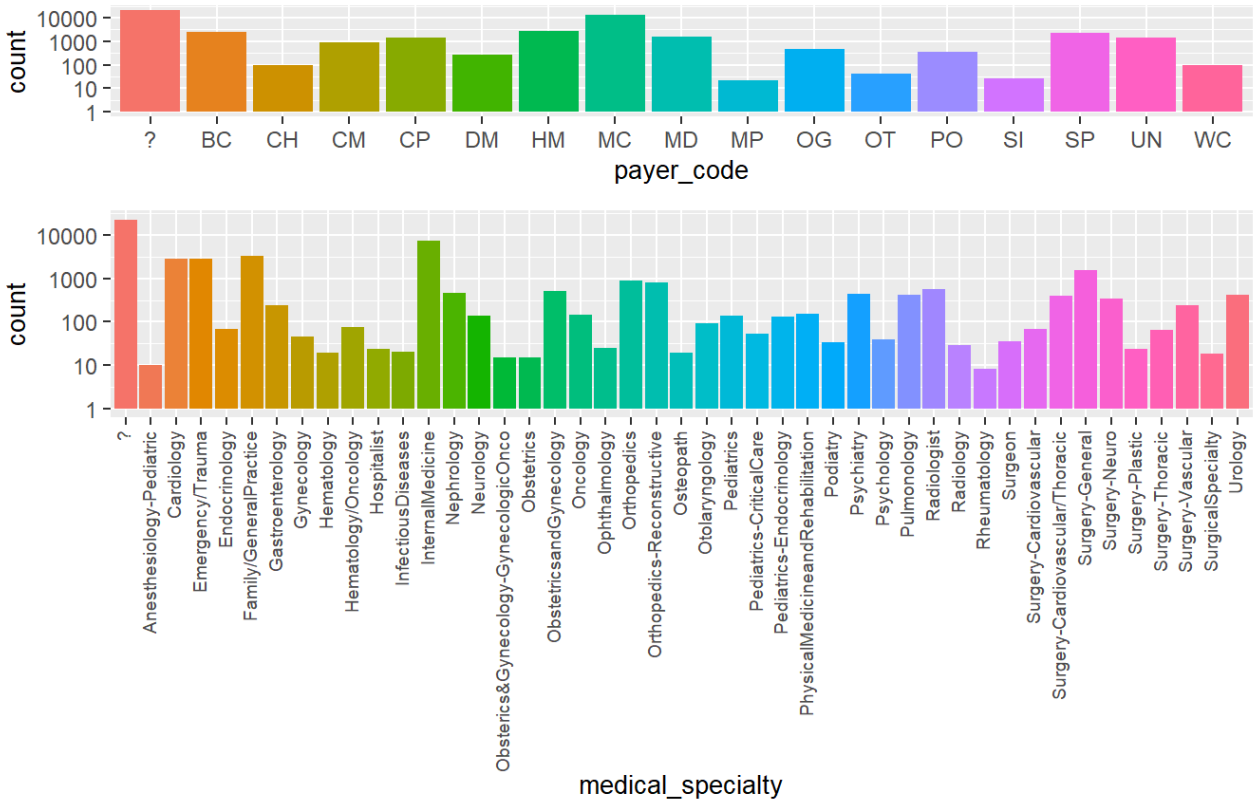
Die wichtigsten Erkenntnisse zu den einzelnen Merkmalen sind wie folgt:

- `admission_type_id` : Die meisten Patienten wurden unter den Kategorien `1` (Notfall), `2` (Dringend) und `3` (Elektiv) aufgenommen, wobei Notfälle mit Abstand am häufigsten auftreten. Die restlichen Kategorien (`5` , `6` , `7` , `8`) sind selten, insbesondere Kategorie `7` .
- `admission_source_id` : Die Kategorie `7` (Notaufnahme) ist hier mit deutlichem Abstand am häufigsten vertreten, gefolgt von Kategorie `1` (Arztüberweisung). Die übrigen Kategorien sind weniger häufig; insbesondere Kategorien wie `3` , `8` , `9` und `20` treten äußerst selten auf.
- `discharge_disposition_id` : Die Mehrheit der Patienten wurde unter den Kategorien `1` (Entlassung nach Hause), `3` (Verlegung in eine andere medizinische Einrichtung) und `6` (Überweisung in eine andere stationäre Einrichtung) entlassen. Einige Kategorien, wie `8` , `15` und `24` , treten vergleichsweise selten auf.

Kategorielle Merkmale zu Versicherungs- und medizinischen Informationen

Zunächst visualisieren wir die Histogramme der Merkmale, die Informationen zu Versicherungen und medizinischen Spezialgebieten der Patienten enthalten: `payer_code` und `medical_specialty` . Um die starke Ungleichverteilung in den Daten besser darzustellen, wurde bei beiden Merkmalen eine logarithmierte y-Achse verwendet. Zusätzlich wurde für das Merkmal `medical_specialty` eine um 90 Grad gedrehte Beschriftung der x-Achse verwendet, um die Lesbarkeit der langen Kategorienamen zu verbessern.

```
p1 <- diabetic_data_bin %>%
  ggplot(aes(payer_code, fill = payer_code)) +
  geom_bar() +
  scale_y_log10() +
  common_theme

p2 <- diabetic_data_bin %>%
  ggplot(aes(medical_specialty, fill = medical_specialty)) +
  geom_bar() +
  scale_y_log10() +
  common_theme +
  theme(
    legend.position = "none",
    axis.text.x = element_text(size = 7, angle = 90, hjust = 1, vjust = 0.3),
    axis.text.y = element_text(size = 9)
  )

# Erzeugen des Layouts und Verwenden der multiplot-Funktion
layout <- matrix(c(1,1,2,2,2,2,2,2),4,2,byrow=TRUE)
multiplot(p1, p2, layout=layout)
```



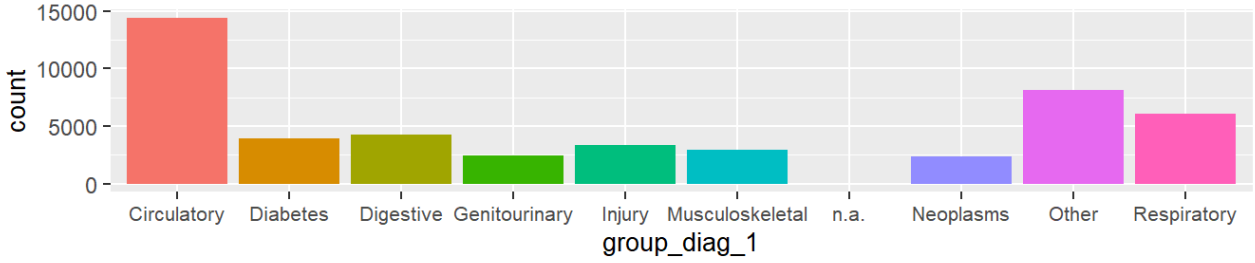
Die wichtigsten Erkenntnisse zu den einzelnen Merkmalen sind wie folgt:

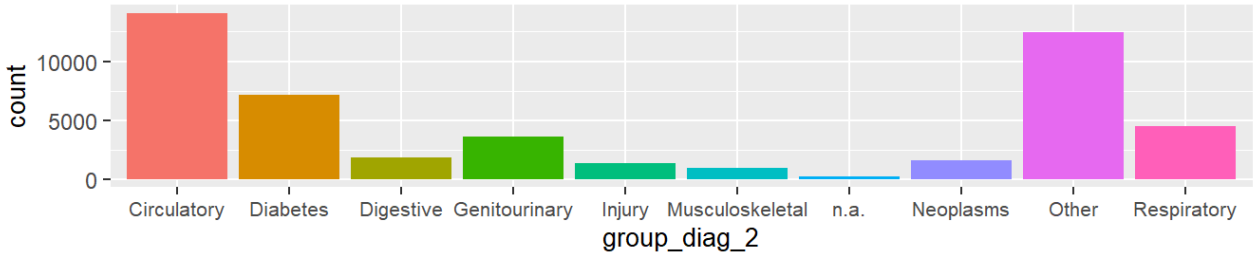
- `payer_code` : Die meisten Einträge sind in der Kategorie `?` , was fehlende Werte darstellt. Unter den definierten Kategorien treten `MC` (Medicare) und `BC` (Blue Cross) am häufigsten auf. Einige Kategorien wie `MP` und `SI` sind äußerst selten und könnten bei der Analyse nur eine begrenzte Rolle spielen.
- `medical_specialty` : Die Kategorie `?` ist erneut die häufigste, was einen hohen Anteil an fehlenden Werten anzeigt. Unter den definierten Spezialgebieten dominieren `InternalMedicine` , `Family/GeneralPractice` und `Cardiology` . Viele Spezialgebiete, wie etwa `Surgery-Cardiovascular` und `Orthopedics` , zeigen mittlere Häufigkeiten. Zahlreiche andere Kategorien, insbesondere in spezialisierten oder seltenen Bereichen wie `Rheumatology` und `Anesthesiology-Pediatric` , sind nur sehr selten anzutreffen.

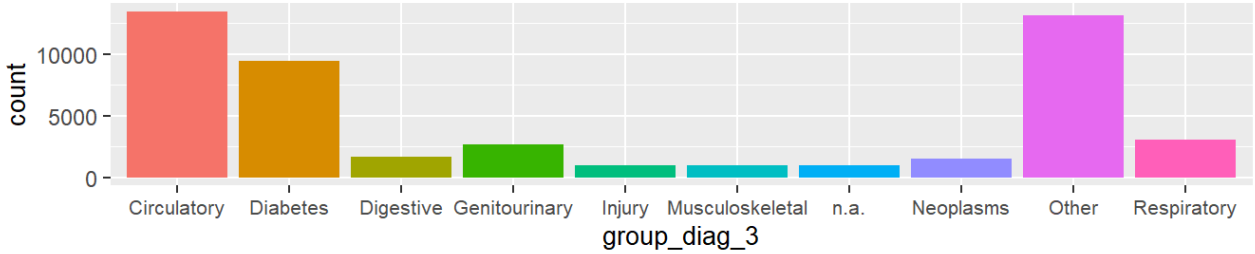
Kategorielle Merkmale zu Diagnosegruppen

Als Nächstes visualisieren wir die Häufigkeitsverteilungen der Merkmale, die Diagnosegruppen der Patienten betreffen: `group_diag_1` , `group_diag_2` und `group_diag_3` . Diese Merkmale wurden im Rahmen der Datenaufbereitung in Aufgabe R1 erzeugt.

```
p1 <- diabetic_data_bin %>%
  ggplot(aes(group_diag_1, fill = group_diag_1)) +
  geom_bar() +
  common_theme +
  theme(axis.text.x = element_text(size = 8))

p2 <- diabetic_data_bin %>%
  ggplot(aes(group_diag_2, fill = group_diag_2)) +
  geom_bar() +
  common_theme +
  theme(axis.text.x = element_text(size = 8))

p3 <- diabetic_data_bin %>%
  ggplot(aes(group_diag_3, fill = group_diag_3)) +
  geom_bar() +
  common_theme +
  theme(axis.text.x = element_text(size = 8))

# Erzeugen des Layouts und Verwenden der multiplot-Funktion
layout <- matrix(c(1,1,2,2,3,3),3,2,byrow=TRUE)
multiplot(p1, p2, p3, layout=layout)
```







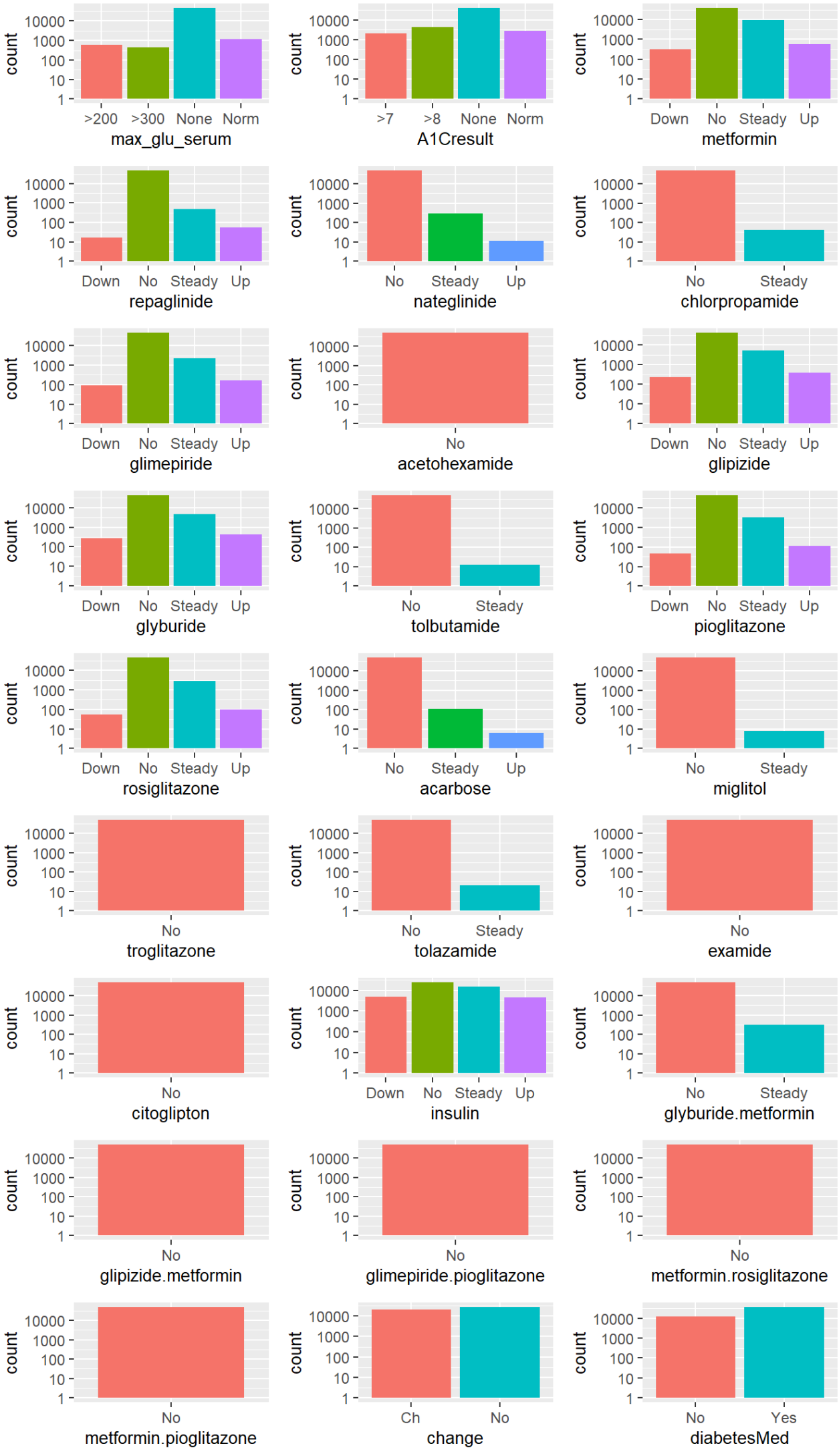
Die wichtigsten Erkenntnisse zu den einzelnen Merkmalen sind wie folgt:

- `group_diag_1` : Die häufigste Diagnosegruppe ist `Circulatory` , die Ausprägungen `Other` und `Respiratory` folgen mit deutlichem Abstand. `Digestive` , `Diabetes` , und `Injury` treten ebenfalls häufiger auf, während `Genitourinary` und `Musculoskeletal` moderat vertreten sind. Kategorien wie `Neoplasms` und insbesondere `n.a.` sind selten. Insgesamt dominiert `Circulatory` die Verteilung in dieser Gruppe.
- `group_diag_2` : Auch hier ist `Circulatory` die am meisten auftretende Kategorie, allerdings ist der Abstand zu `Other` weit weniger gering als bei `group_diag_1` . `Diabetes` zeigt in dieser Gruppe eine deutliche Zunahme im Vergleich zu `group_diag_1` . Die Kategorien `Digestive` , `Respiratory` , `Genitourinary` , und `Injury` treten weniger häufig auf, wobei `Genitourinary` etwas an Bedeutung gewinnt. `n.a.` bleibt selten.
- `group_diag_3` : `Circulatory` ist – ähnlich wie bei `group_diag_2` – zusammen mit `Other` die mit Abstand dominante Kategorie. `Diabetes` erreicht in dieser Gruppe seinen Höchstwert und überholt andere Kategorien deutlich; `Respiratory` nimmt im Vergleich zu den anderen Gruppen an Bedeutung ab. Die Häufigkeit von `Digestive` , `Genitourinary` , `Musculoskeletal` , und `Injury` bleibt niedrig, und `Neoplasms` sowie `n.a.` sind marginal vertreten.

Kategorielle Merkmale zu diabetesbezogenen Merkmalen und Medikation

Nun visualisieren wir die Histogramme der Merkmale, die Details zu Diagnosetests und der Medikation der Patienten beschreiben, darunter: `max_glu_serum` , `A1Cresult` , `change` und diverse Medikamente (z. B. `insulin` , `metformin` , `glipizide`). Für alle Merkmale wurde eine logarithmierte y-Achse verwendet, um der starken Ungleichverteilung Rechnung zu tragen.

```
features_to_plot <- c("max_glu_serum", "A1Cresult", "metformin", "repaglinide", "nateglinide", "chlorpropamide", "glimepirid
        e", "acetohexamide", "glipizide", "glyburide", "tolbutamide", "pioglitazone", "rosiglitazone", "acarbose", "miglito
        l", "troglitazone", "tolazamide", "examide", "citoglipton", "insulin", "glyburide.metformin", "glipizide.metformi
        n", "glimepiride.pioglitazone", "metformin.rosiglitazone", "metformin.pioglitazone", "change", "diabetesMed")

# Function to create individual plots
create_plot <- function(data, feature) {
  ggplot(data, aes_string(feature, fill = feature)) +
    geom_bar() +
    scale_y_log10() +
    scale_x_discrete(drop=FALSE) +
    #scale_y_continuous(trans = "log1p") +
    common_theme
}

# Generate plots and store them in a list
plots <- lapply(features_to_plot, create_plot, data = diabetic_data_bin)

# Erzeugen des Layouts und Verwenden der multiplot-Funktion
layout <- matrix(1:27, nrow = 9, ncol = 3, byrow = TRUE)
multiplot(plotlist = plots, layout=layout)
```



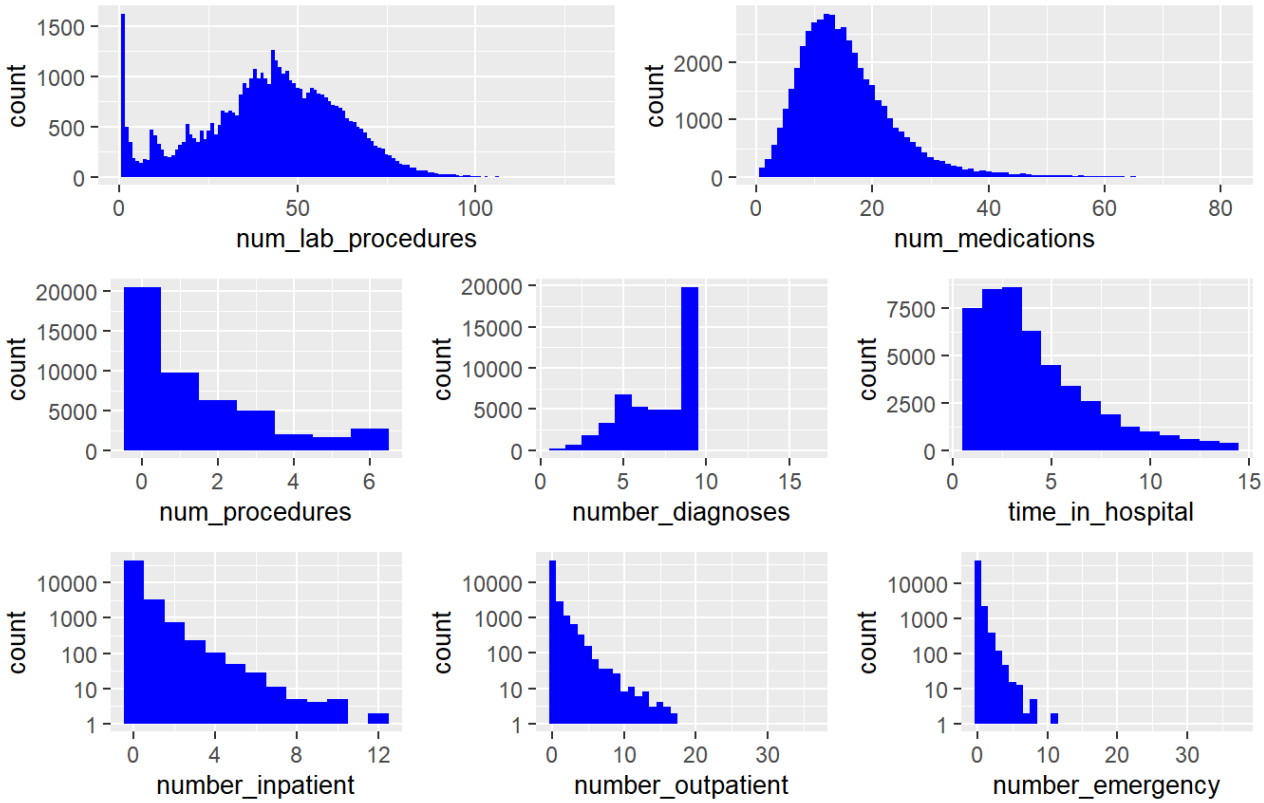
Die wichtigsten Erkenntnisse in dieser Merkmalsgruppe sind wie folgt:

- `max_glu_serum` und `A1Cresult` : Die Mehrheit der Werte bei beiden Merkmalen ist als `None` kodiert, was auf fehlende Testresultate schließen lässt. Die restlichen Werte konzentrieren sich auf Kategorien wie `>200` bei `max_glu_serum` und `>7` sowie `>8` bei `A1Cresult` .
- `insulin` : Hier zeigt sich eine deutliche Variabilität. Ein Großteil der Fälle fällt in die Kategorie `No` , gefolgt von `Steady` . Bemerkenswert ist der relativ hohe Anteil an `Up` und `Down` , was auf eine dynamische Anpassung bei der Insulingabe hinweist.
- `change` : Eine signifikante Anzahl an Fällen (`Ch`) deutet darauf hin, dass häufig Anpassungen in der Medikation vorgenommen werden. Ein erheblicher Anteil an `No` zeigt jedoch auch stabile Behandlungsverläufe.
- Medikamente mit häufigen Anpassungen: Bei einigen Medikamenten wie `metformin` , `glipizide` , `glyburide` , und `pioglitazone` zeigt sich ein Muster von Anpassungen in den Kategorien `Steady` , `Up` , und `Down` . Diese Medikamente werden offenbar regelmäßig genutzt und modifiziert.
- Praktisch ungenutzte Medikamente: Einige Medikamente wie `acarbose` , `miglitol` , `troglitazone` und Kombinationen wie `glyburide.metformin` werden nahezu ausschließlich in der Kategorie `No` angegeben, was auf eine seltene bis kaum vorhandene Verwendung hindeutet.
- Merkmale mit nur einem Wert: Einige Merkmale wie `examide` , `citoglipton` , und Kombinationen wie `metformin.rosiglitazone` haben nur eine Ausprägung (`No`), was bedeutet, dass sie keinen Informationsgehalt für die Vorhersage von Wiedereinweisungen liefern und aus der Modellierung entfernt werden könnten, um die Effizienz zu steigern.

Numerische Merkmale zu Krankenhausressourcennutzung und Gesundheitszustand

Die numerischen Merkmale des Datensatzes besitzen ganzzahlige Werte. Für einige dieser Merkmale verwenden wir eine logarithmierte Achse.

```
p1 <- diabetic_data_bin %>%
  ggplot(aes(num_lab_procedures, fill = num_lab_procedures)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  common_theme

p2 <- diabetic_data_bin %>%
  ggplot(aes(num_procedures, fill = num_procedures)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  common_theme

p3 <- diabetic_data_bin %>%
  ggplot(aes(number_diagnoses, fill = number_diagnoses)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  common_theme

p4 <- diabetic_data_bin %>%
  ggplot(aes(num_medications, fill = num_medications)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  common_theme

p5 <- diabetic_data_bin %>%
  ggplot(aes(number_emergency, fill = number_emergency)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  scale_y_log10() +
  common_theme

p6 <- diabetic_data_bin %>%
  ggplot(aes(time_in_hospital, fill = time_in_hospital)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  common_theme

p7 <- diabetic_data_bin %>%
  ggplot(aes(number_inpatient, fill = number_inpatient)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  scale_y_log10() +
  common_theme

p8 <- diabetic_data_bin %>%
  ggplot(aes(number_outpatient, fill = number_outpatient)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  scale_y_log10() +
  common_theme

#layout <- matrix(c(1,1,2,3,4,4,5,6,7,7,8,8),3,4,byrow=TRUE)
#multiplot(p1, p2, p3, p4, p5, p6, p7, p8, layout=layout)
layout <- matrix(c(1,1,1,2,2,2,3,3,4,4,5,5,6,6,7,7,8,8),3,6,byrow=TRUE)
multiplot(p1, p4, p2, p3, p6, p7, p8, p5, layout=layout)
```



Die wichtigsten Erkenntnisse in dieser Merkmalsgruppe sind wie folgt:

- `num_lab_procedures` : Der Großteil der Patienten hat etwa 40 bis 50 Labortests durchführen lassen, was auf umfangreiche diagnostische Untersuchungen hinweist. Eine kleinere Gruppe hat jedoch keine Labortests absolviert, was möglicherweise mit kürzeren Krankenhausaufenthalten zusammenhängt.
- `num_medications` : Die Verteilung zeigt eine Konzentration auf 10 bis 20 Medikamente, mit einem abnehmenden Trend bei höheren Zahlen. Dies weist darauf hin, dass viele Patienten mit einer moderaten Menge an Medikamenten behandelt wurden, während nur wenige eine umfangreichere Pharmakotherapie erhielten.
- `time_in_hospital` : Die Krankenhausaufenthaltsdauer ist meist kurz: die häufigste Aufenthaltsdauer beträgt 3 Tage. Längere Krankenhausaufenthalte sind seltener und deuten wahrscheinlich auf komplexere Krankheitsfälle hin.
- `number_diagnoses` : Die Mehrheit der Patienten hat etwa 8 bis 9 Diagnosen, was auf komplexe Krankheitsbilder schließen lässt. Sehr viele Diagnosen (>10) kommen nur selten vor.
- `number_inpatient` , `number_emergency` und `number_outpatient` : Alle drei Merkmale sind stark rechtsschief verteilt, viele Patienten hatten wenige bis keine Aufenthalte in diesen Kategorien. Die maximalen Aufenthalte lagen bei `number_inpatient` bei nur 12, bei `number_outpatient` bzw. `number_emergency` bei 36 bzw.

  37.
- `num_procedures` : Ein bemerkenswerter Anteil an Patienten hatte keine zusätzlichen medizinischen Verfahren (außer Labortests), und es gibt keinen konsistenten Anstieg bei häufigeren Eingriffen.

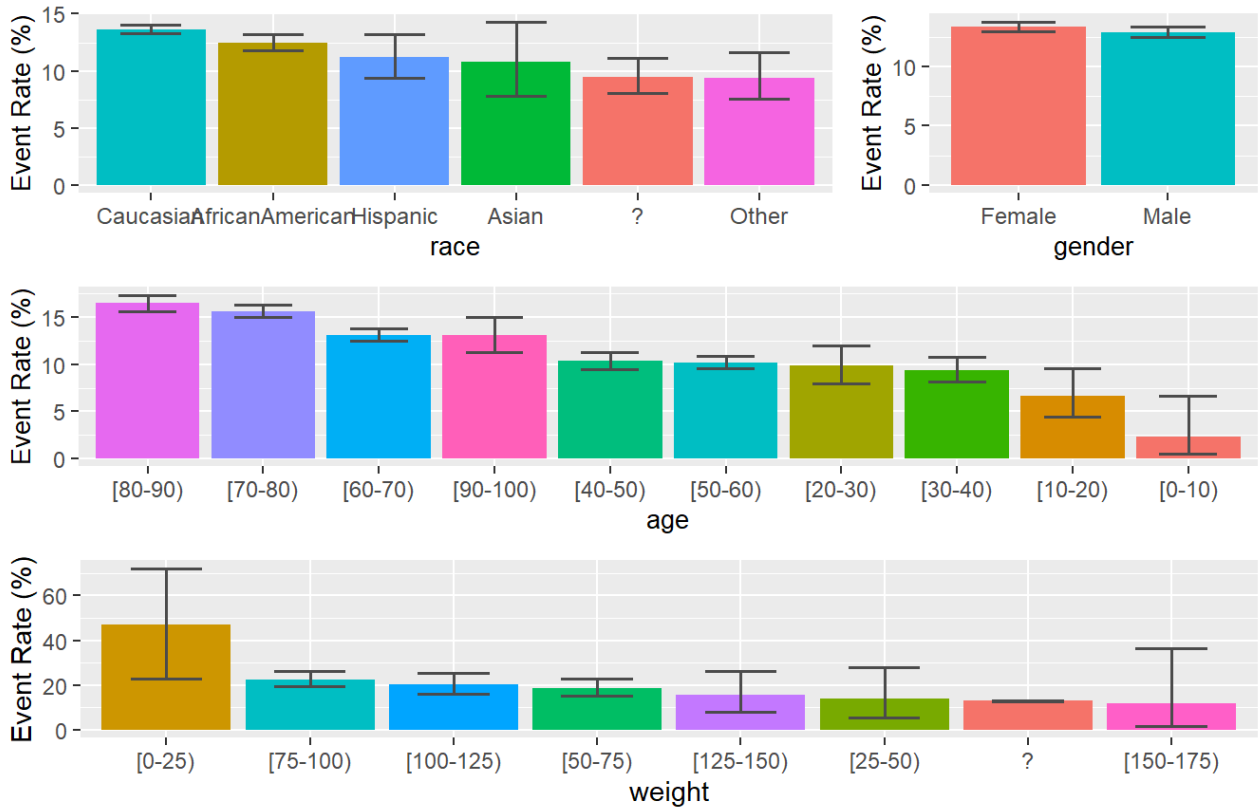
c) 5 Claim rates for individual features: Berechnen und visualisieren Sie die relativen Häufigkeiten der Zielvariable `TARGET` , also die Wiedereinweisungsquoten, für jedes kategorielle Merkmal, ähnlich wie in den Abschnitten 5.1 bis 5.4 der Vorlage. Für die numerischen Merkmale verfahren Sie entsprechend wie in den Abschnitten 5.5 und 5.6. Nutzen Sie dabei die Funktionen `multiplot` und `get_binCI` und unterteilen Sie die Visualisierungen entsprechend der in Teilaufgabe R2b) gewählten Gruppen. Diskutieren Sie die wichtigsten Erkenntnisse aus den Visualisierungen, insbesondere mit Blick auf deren Bedeutung für die Wiedereinweisungsprognose.

Wir verwenden dieselben Merkmalsgruppierungen wie in Teilaufgabe R2b):

- *Kategorielle Merkmale zu persönlichen Informationen* (z. B. Geschlecht, Alter)
- *Kategorielle Merkmale zu Krankenhausaufenthaltsdetails* (z. B. Aufnahme- und Entlassungsmodalitäten)
- *Kategorielle Merkmale zu Versicherungs- und medizinischen Informationen* (z. B. Fachrichtung, Versicherungsdetails)
- *Kategorielle Merkmale zu Diagnosegruppen*
- *Kategorielle Merkmale zu diabetesbezogenen Merkmalen und Medikation* (z. B. Testergebnisse, Medikamenteneinsatz)
- *Numerische Merkmale zu Krankenhausressourcennutzung und Gesundheitszustand* (z. B. Anzahl der Prozeduren, Verweildauer im Krankenhaus)

Kategorielle Merkmale zu persönlichen Informationen

```
p1 <- diabetic_data_bin %>%
  group_by(race, TARGET) %>%
  count() %>%
  spread(TARGET, n) %>%
  mutate(frac_claim = `1`/(`1`+`0`)*100,
         lwr = get_binCI(`1`,(`1`+`0`))[[1]]*100,
         upr = get_binCI(`1`,(`1`+`0`))[[2]]*100
         ) %>%
  ggplot(aes(reorder(race, -frac_claim, FUN = max), frac_claim, fill = race)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "race", y = "Event Rate (%)")

p2 <- diabetic_data_bin %>%
  group_by(gender, TARGET) %>%
  count() %>%
  spread(TARGET, n) %>%
  mutate(frac_claim = `1`/(`1`+`0`)*100,
         lwr = get_binCI(`1`,(`1`+`0`))[[1]]*100,
         upr = get_binCI(`1`,(`1`+`0`))[[2]]*100
         ) %>%
  ggplot(aes(reorder(gender, -frac_claim, FUN = max), frac_claim, fill = gender)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "gender", y = "Event Rate (%)")

p3 <- diabetic_data_bin %>%
  group_by(age, TARGET) %>%
  count() %>%
  spread(TARGET, n) %>%
  mutate(frac_claim = `1`/(`1`+`0`)*100,
         lwr = get_binCI(`1`,(`1`+`0`))[[1]]*100,
         upr = get_binCI(`1`,(`1`+`0`))[[2]]*100
         ) %>%
  ggplot(aes(reorder(age, -frac_claim, FUN = max), frac_claim, fill = age)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "age", y = "Event Rate (%)")

p4 <- diabetic_data_bin %>%
  group_by(weight, TARGET) %>%
  count() %>%
  spread(TARGET, n) %>%
  mutate(frac_claim = `1`/(`1`+`0`)*100,
         lwr = get_binCI(`1`,(`1`+`0`))[[1]]*100,
         upr = get_binCI(`1`,(`1`+`0`))[[2]]*100
         ) %>%
  ggplot(aes(reorder(weight, -frac_claim, FUN = max), frac_claim, fill = weight)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  theme(legend.position = "none") +
  labs(x = "weight", y = "Event Rate (%)")

# Erzeugen des Layouts und Verwenden der multiplot-Funktion
layout <- matrix(c(1,1,2,3,3,3,4,4,4),3,3,byrow=TRUE)
multiplot(p1, p2, p3, p4, layout=layout)
```

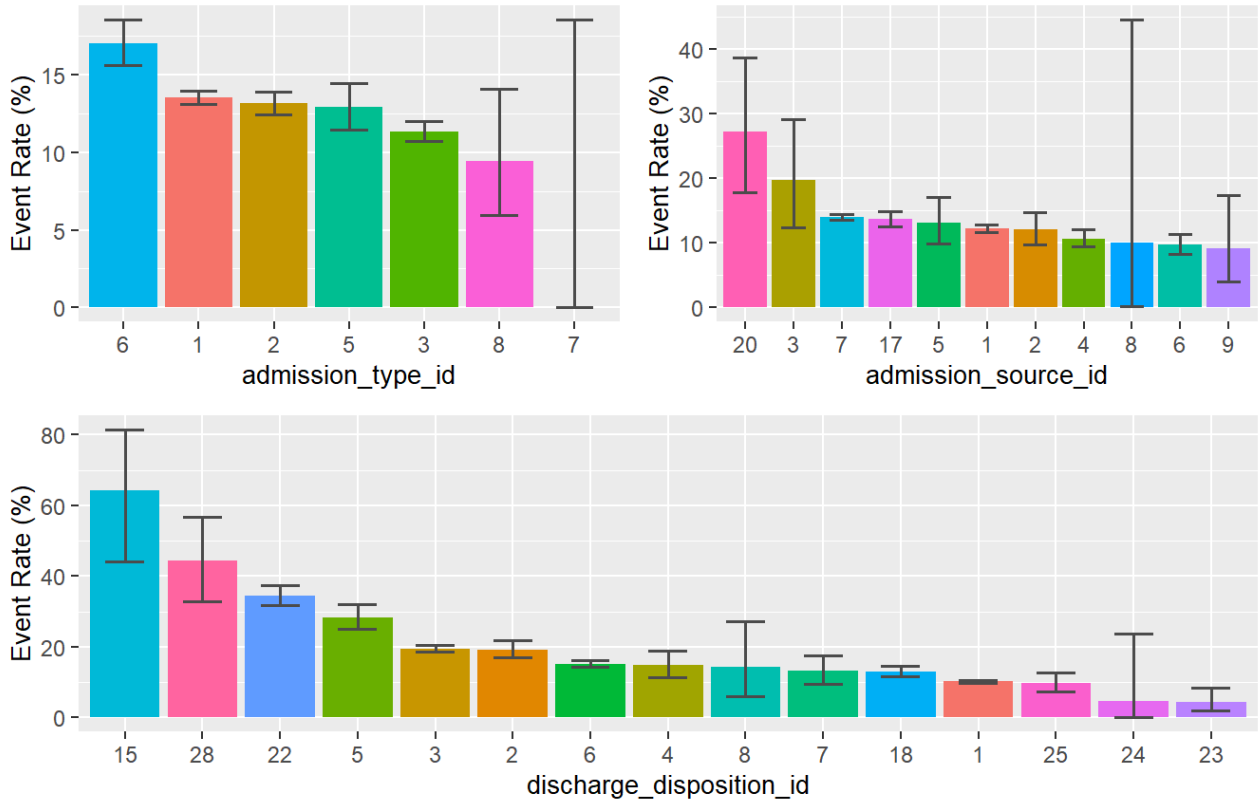


Die wichtigsten Erkenntnisse zu den einzelnen Merkmalen sind wie folgt:

- `race` : Höchste Wiedereinweisungsrate beim zugleich häufigsten Merkmal `Caucasian` , gefolgt von `AfricanAmerican` . Andere Kategorien mit moderaten Raten und geringer Absoluthäufigkeit.
- `gender` : Die Verteilung ist fast gleichmäßig (`Female: 0.134` , `Male: 0.129`) mit leicht höheren Wiedereinweisungsraten bei Frauen. Das Merkmal hat vermutlich wenig Einfluss auf die Zielvariable.
- `age` : Wiedereinweisungsraten steigen tendenziell mit dem Alter, der höchste Wert liegt in der Altersgruppe `[80-90)` . Junge Altersgruppen zeigen sehr niedrige Raten.
- `weight` : Über 90% der Werte fehlen, daher keine großen Aussagen tätigbar. In den extremen Randgruppen (`[0-25)` und `[150-175)`) zusätzlich sehr große Konfidenzintervalle.

Kategorielle Merkmale zu Krankenhausaufenthaltsdetails

```r
p1 <- diabetic_data_bin %>%
  group_by(admission_type_id, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(admission_type_id, -frac_claim, FUN = max), frac_claim, fill = admission_type_id)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "admission_type_id", y = "Event Rate (%)")

p2 <- diabetic_data_bin %>%
  group_by(admission_source_id, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(admission_source_id, -frac_claim, FUN = max), frac_claim, fill = admission_source_id)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "admission_source_id", y = "Event Rate (%)")

p3 <- diabetic_data_bin %>%
  group_by(discharge_disposition_id, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(discharge_disposition_id, -frac_claim, FUN = max), frac_claim, fill = discharge_disposition_id)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "discharge_disposition_id", y = "Event Rate (%)")

# Erzeugen des Layouts und Verwenden der multiplot-Funktion
layout <- matrix(c(1,2,3,3),2,2,byrow=TRUE)
multiplot(p1, p2, p3, layout=layout)
```



Die wichtigsten Erkenntnisse zu den einzelnen Merkmalen sind wie folgt:

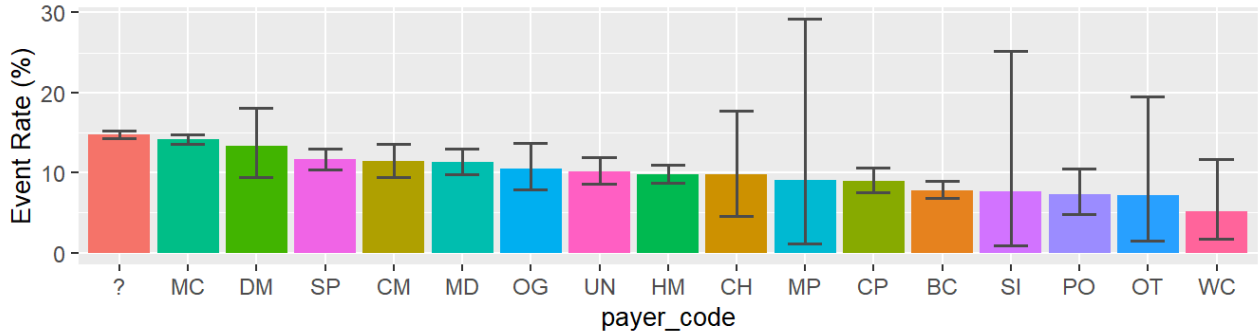
- `admission_type_id` : Patienten, die als `NULL` (Kategorie `6`) aufgenommen wurden, weisen die höchste Wiedereinweisungsrate (0.170) auf, was auf spezielle Aufnahmeumstände hinweisen könnte. Am häufigsten sind Aufnahmen mit Kategorie `Emergency` (`1` , 0.135), `Urgent` (`2` , 0.132) und `Elective` (`3` , 0.113), wobei letztere eine niedrigere Rate zeigt. Patienten von Trauma-Zentren (7) zeigen gar keine Wiedereinweisungen an.
- `admission_source_id` : Die Aufnahmequelle `Not Mapped` (20) weist mit 0.273 die höchste Wiedereinweisungsrate auf, ist aber selten und mit vergleichsweise großen Konfidenzintervallen behaftet. Häufige Kategorien wie `Emergency Room` (`7` , 0.140), `Physician Referral` (`1` , 0.122) und `NULL` (17, 0.137) zeigen ungefähr die durchschnittliche Wiedereinweisungsrate an. Die relativ großen Unterschiede in den

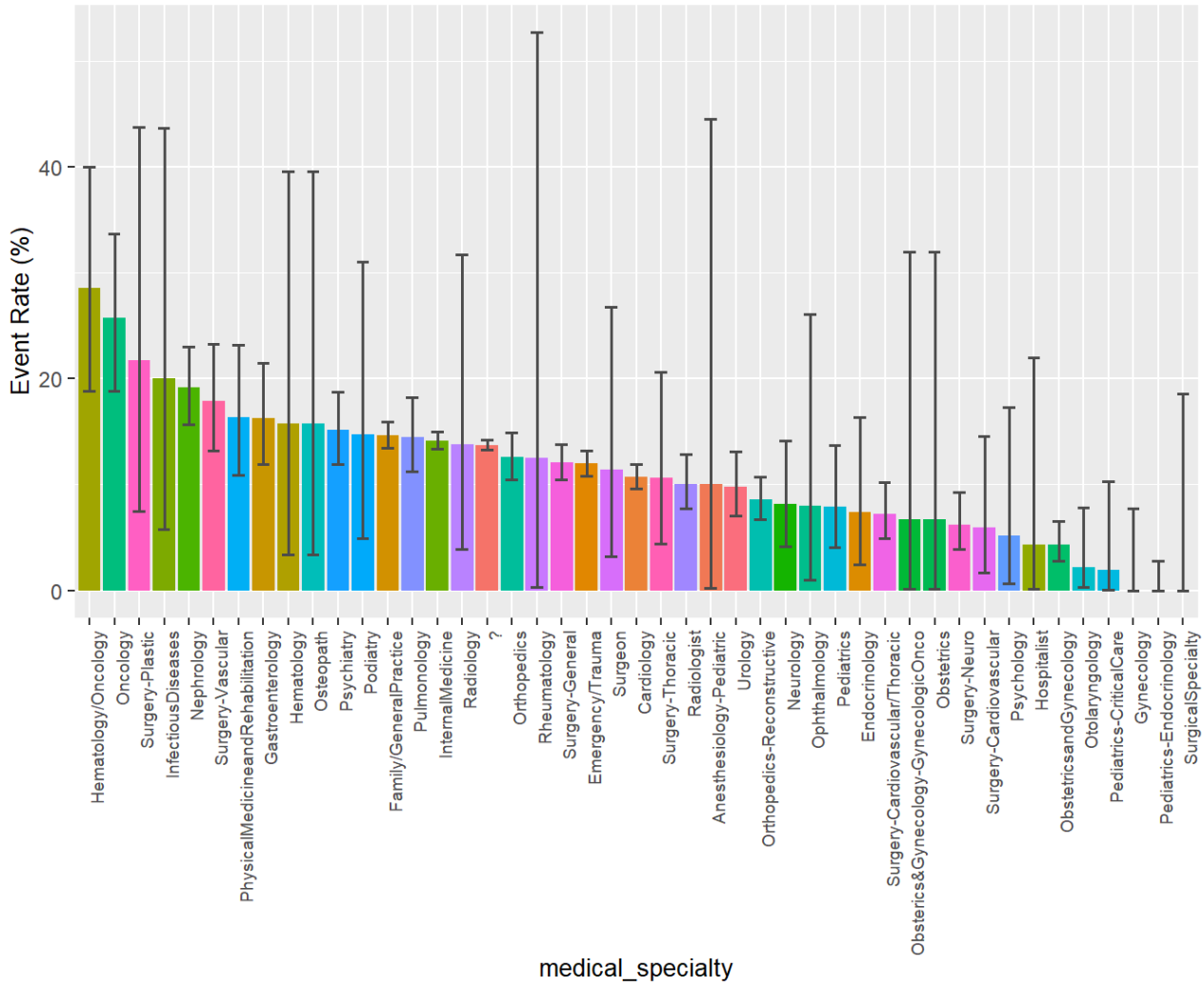
Wiedereinweisungsraten zwischen den Aufnahmequellen scheint für die Vorhersage wertvoll sein.

- `discharge_disposition_id` : Patienten, die in Einrichtungen wie Medicare-genehmigten Swing-Beds (Kategorie `15` , 0.643), psychiatrischen Krankenhäusern (`28` , 0.444) oder Reha-Kliniken (`22` , 0.346) entlassen wurden, weisen signifikant höhere Wiedereinweisungsraten auf als Patienten, die nach Hause entlassen wurden (häufigste Ausprägung, `1` , 0.102). Dies deutet darauf hin, dass die Entlassungsart ein starker Prädiktor für eine mögliche Wiedereinweisung ist. Seltenere Kategorien wie die Entlassung in Langzeitpflegeeinrichtungen (`23` , 0.0442) oder Pflegeeinrichtungen, die nach Medicaid zertifiziert, aber nicht nach Medicare zertifiziert sind, (`24` , 0.0476) weisen nur sehr geringere Wiedereinweisungsraten auf.

Kategorielle Merkmale zu Versicherungs- und medizinischen Informationen

```
p1 <- diabetic_data_bin %>%
  group_by(payer_code, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(payer_code, -frac_claim, FUN = max), frac_claim, fill = payer_code)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "payer_code", y = "Event Rate (%)")

p2 <- diabetic_data_bin %>%
  group_by(medical_specialty, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(medical_specialty, -frac_claim, FUN = max), frac_claim, fill = medical_specialty)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "medical_specialty", y = "Event Rate (%)") +
  theme(
    legend.position = "none",
    axis.text.x = element_text(size = 7, angle = 90, hjust = 1),  # Set x-axis text size
    axis.text.y = element_text(size = 9)  # Keep y-axis text size consistent with common theme
  )

# Erzeugen des Layouts und Verwenden der multiplot-Funktion
layout <- matrix(c(1,1,2,2,2,2,2,2),4,2,byrow=TRUE)
multiplot(p1, p2, layout=layout)
```



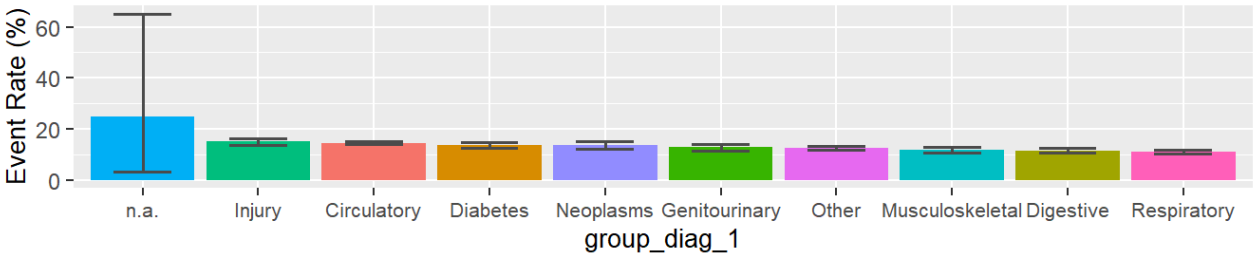


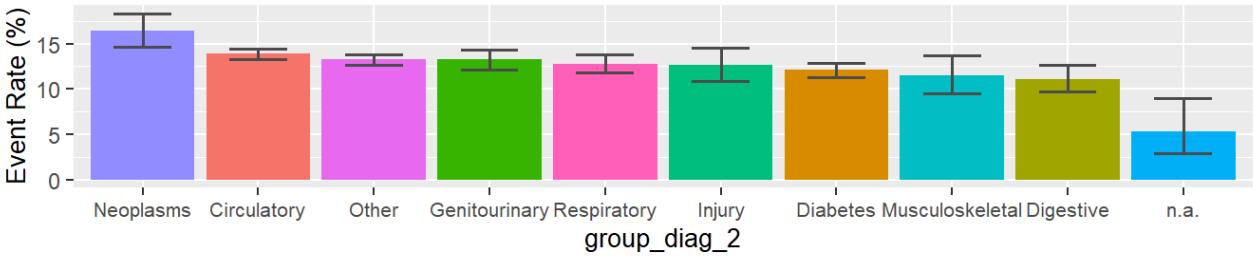
Die wichtigsten Erkenntnisse zu den einzelnen Merkmalen sind wie folgt:

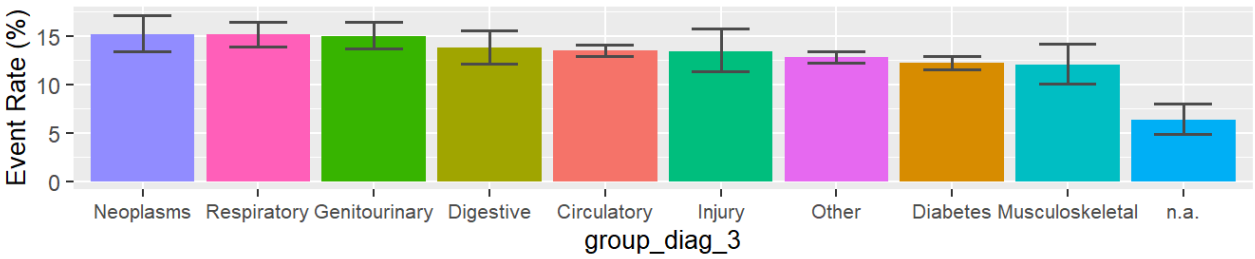
- `payer_code` : Die Ausprägung `?` (fehlender Wert) ist am häufigsten vertreten und weist eine leicht erhöhte Wiedereinweisungsrate (14,8%) auf. Die zweihäufigste Ausprägung Medicare (`MC` , 27,5%) zeigt ein ähnliches Verhalten. `DM` und `SP` haben vergleichsweise höhere Wiedereinweisungsraten (13,3% und 11,6%) bei moderater Häufigkeit. Seltenere Kategorien wie `WC` haben außergewöhnlich niedrige Werte (5,2%). Für `payer_code` scheint der überwiegende Anteil fehlender Werte problematisch, obwohl Kategorien wie `DM` und `SP` Hinweise auf Relevanz für die Zielvariable geben könnten.
- `medical_specialty` : `?` dominiert mit 47,4% und einem mittleren Wiedereinweisungsrate (13,7%), die ebenfalls häufig auftretende Kategorie `InternalMedicine` liegen bei 14,2%. Weniger verbreitete Fachrichtungen wie `Oncology` und `Surgery-Plastic` zeigen mit 25,7% und 21,7% hohe Wiedereinweisungsraten. Fachrichtungen wie `ObstetricsandGynecology` weisen hingegen extrem niedrige Werte (4,4%) auf. Spezialisierte Fachrichtungen wie `Oncology` und `Surgery-Plastic` könnten signifikant zur Vorhersage der Zielvariable beitragen. Fachrichtungen mit sehr niedrigen Wiedereinweisungsraten könnten hingegen ausgeschlossen werden, da sie vermutlich weniger Einfluss haben.

Kategorielle Merkmale zu Diagnosegruppen

```r
# Modify the first diagnosis group plot
p1 <- diabetic_data_bin %>%
  group_by(group_diag_1, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(group_diag_1, -frac_claim, FUN = max), frac_claim, fill = group_diag_1)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "group_diag_1", y = "Event Rate (%)") +
  theme(axis.text.x = element_text(size = 8))

# Modify the second diagnosis group plot
p2 <- diabetic_data_bin %>%
  group_by(group_diag_2, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(group_diag_2, -frac_claim, FUN = max), frac_claim, fill = group_diag_2)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "group_diag_2", y = "Event Rate (%)") +
  theme(axis.text.x = element_text(size = 8))

# Modify the third diagnosis group plot
p3 <- diabetic_data_bin %>%
  group_by(group_diag_3, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(group_diag_3, -frac_claim, FUN = max), frac_claim, fill = group_diag_3)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "group_diag_3", y = "Event Rate (%)") +
  theme(axis.text.x = element_text(size = 8))

# Define the layout
layout <- matrix(c(1, 1, 2, 2, 3, 3), 3, 2, byrow = TRUE)

# Use multiplot to display the modified plots
multiplot(p1, p2, p3, layout = layout)
```





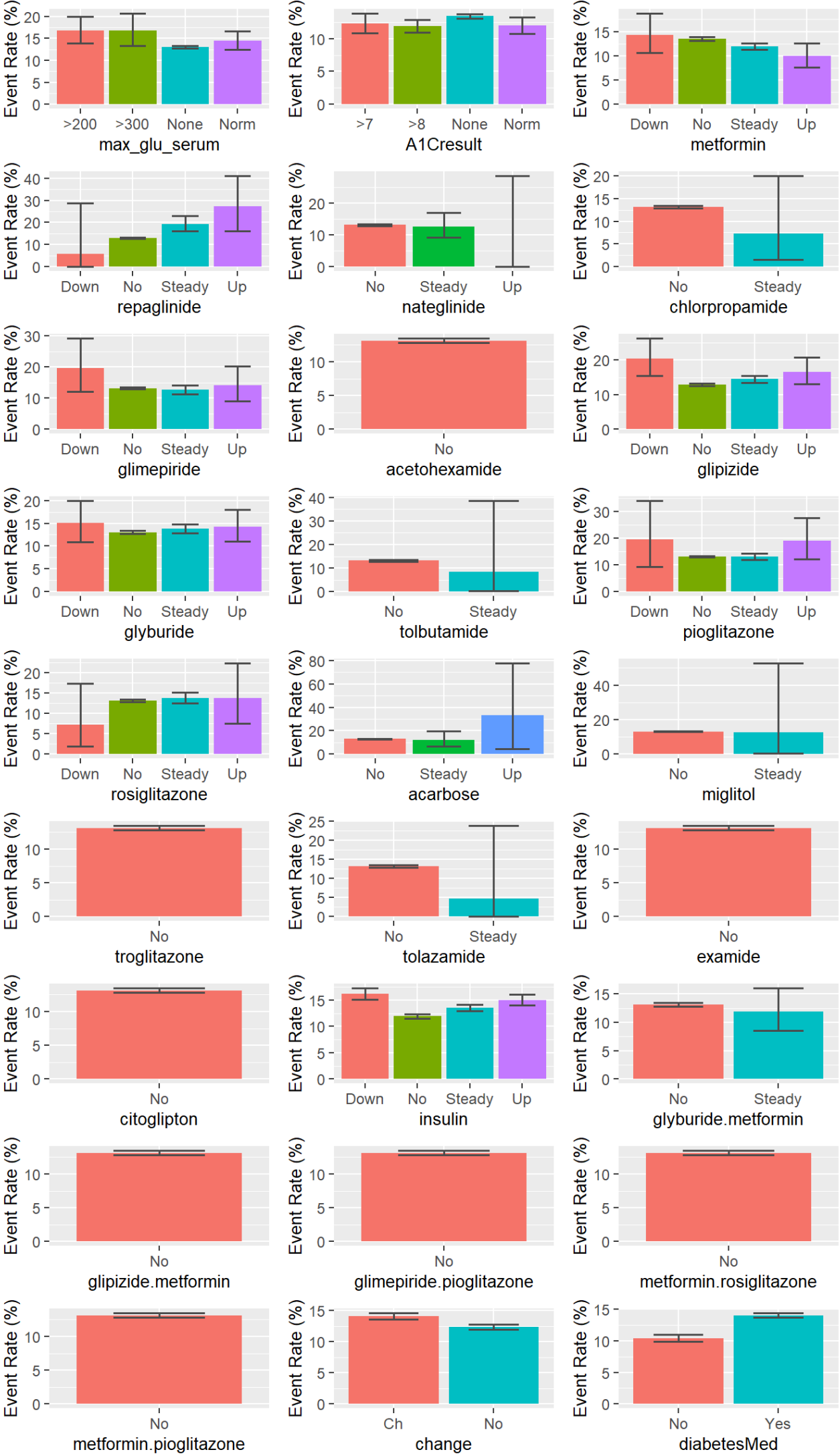


Die wichtigsten Erkenntnisse zu den einzelnen Merkmalen sind wie folgt:

- `group_diag_1` : Die am häufigsten auftretenden Diagnosegruppen `Circulatory` , `Other` resp. `Respiratory` weisen Wiedereinweisungsraten von 14.4%, 12.6% resp. 11.1% auf. Die höchste Wiedereinweisungsrate zeigt sich bei der Kategorie `Injury` (15%) sowie bei `n.a.` (25%), letztere ist jedoch extrem selten und daher statistisch wenig signifikant.
- `group_diag_2` : Diagnosen in den häufig auftretenden Gruppen `Circulatory` (13.9%), `Other` (13.2%) und `Respiratory` (12.8%) haben durchschnittliche Wiedereinweisungsraten. Die höchste Rate wurde bei der Kategorie `Neoplasms` (16.4%) beobachtet, während `n.a.` mit 5.35% besonders niedrig liegt, jedoch erneut sehr selten ist.
- `group_diag_3` : Die am häufigsten auftretenden Diagnosegruppe `Circulatory` (13.5%), `Other` (12.8%) und `Diabetes` (12.2%) zeigen unauffällige Wiedereinweisungsraten. Überdurchschnittliche Wiedereinweisungsraten treten bei `Neoplasms` (15.2%), `Respiratory` (15.2), `Genitourinary` (15.0%) und `. Kategorien wie n.a. (6.31%) zeigen sehr niedrige Wiederaufnahmeraten, treten jedoch selten auf.

Kategorielle Merkmale zu diabetesbezogenen Merkmalen und Medikation

```
# Function to create individual plots with fractions and error bars
create_plot <- function(data, feature) {
  data %>%
    group_by(.data[[feature]], TARGET) %>%  # Group by the feature and TARGET
    count() %>%
    spread(TARGET, n, fill = 0) %>%  # Spread the TARGET variable
    mutate(frac_claim = `1` / (`1` + `0`) * 100,  # Calculate fraction
           lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,  # Lower confidence interval
           upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%  # Upper confidence interval
    ggplot(aes_string(x = feature, y = "frac_claim", fill = feature)) +
    geom_col() +  # Use geom_col for bars
    geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +  # Add error bars
    scale_x_discrete(drop = FALSE) +  # Keep all levels
    common_theme +
    labs(x = feature, y = "Event Rate (%)")  # Update axis labels
}

# Generate plots and store them in a list
plots <- lapply(features_to_plot, create_plot, data = diabetic_data_bin)

# Define the layout matrix (3 rows, 2 columns)
layout <- matrix(1:27, nrow = 9, ncol = 3, byrow = TRUE)

# Use multiplot to display the plots with the specified layout
multiplot(plotlist = plots, layout = layout)
```



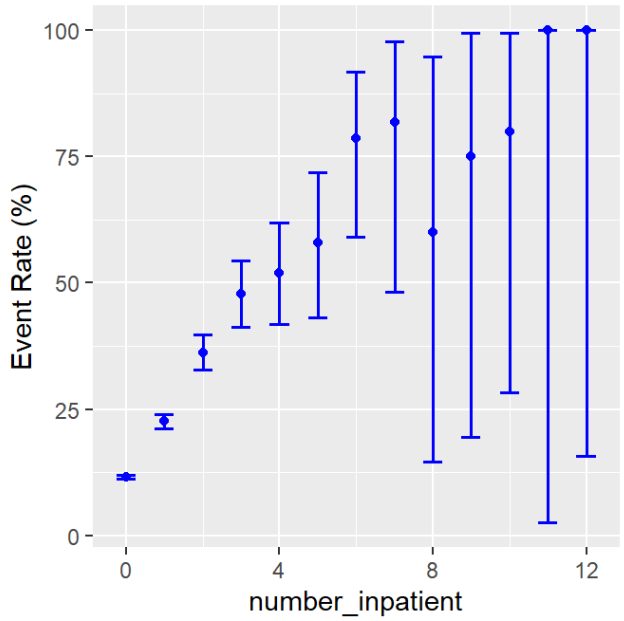
Die wichtigsten Erkenntnisse in dieser Merkmalsgruppe sind wie folgt:

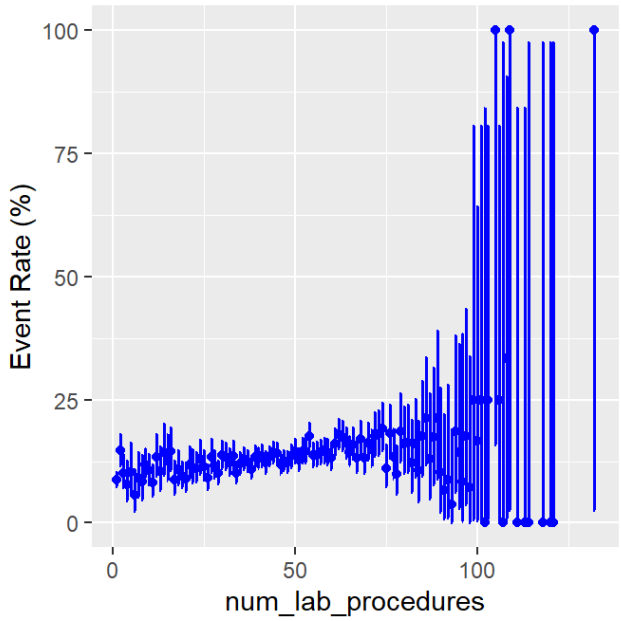
- `max_glu_serum` bzw. `A1Cresult` : Der überwiegende Anteil der Werte ist als `None` kodiert, was auf fehlende oder nicht durchgeführte Tests hinweist. Höhere Kategorien wie `>200` und `>300` bzw. `>7` haben höhere Wiedereinweisungsraten (z. B. `max_glu_serum` `>200` : 16.8%).
- `metformin` , `repaglinide` und `nateglinide` : Während `No` der dominierende Status ist, zeigen einzelne Kategorien wie `Up` oder `Steady` bemerkenswerte Wiedereinweisungsraten (`repaglinide` `Up` : 27.3%, `metformin` `Up` : 9.93%). Einige Medikamente wie `nateglinide` werden selten angewendet, haben jedoch hohe Wiedereinweisungsraten in speziellen Kategorien.
- Seltene oder kaum verwendete Medikamente: Medikamente wie `chlorpropamide` , `tolbutamide` , und Kombinationen wie `glimepiride.pioglitazone` sowie `metformin.pioglitazone` werden fast ausschließlich in der Kategorie `No` aufgeführt. Dies bedeutet, dass diese für die Vorhersage vermutlich keinen Informationswert haben.
- Medikamente mit variablen Status: Medikamente wie `glimepiride` , `glipizide` , `glyburide` , und `pioglitazone` besitzen nicht zu vernachlässigend wenige Anzahlen bei den Ausprägungen `Down` , `Steady` , und `Up` , wobei hohe Wiedereinweisungsraten oftmals in `Down` (`glimepiride` `Down` : 19.6%, `glipizide` `Down` : 20.5%) und `Up` (`glipizide` `Up` : 16.6%) Kategorien auffallen.
- `insulin` : Dieses Diabetes-Medikament zeigt ausgeprägte Variabilität zwischen den Kategorien, wobei `Down` (16.2%) und `Up` (15.0%) besonders häufig mit erhöhten Wiedereinweisungsraten verbunden sind.
- `change` und `diabetesMed` : Das fast gleichverteilte Merkmal `Ch` (Änderung der Medikation) weist eine höhere Wiedereinweisungsrate im Fälle einer Änderung der Medikation auf (`Ch` : 14.1%), was auf dynamische Behandlungsschemata schließen lässt. Patienten mit Diabetes-Medikation (`Yes`) haben eine deutlich höhere Wiedereinweisungsrate (14.0%) als solche ohne (`No` : 10.5%).
- Medikamente ohne Statusvariabilität: Merkmale wie `troglitazone` , `examide` , und `citoglipton` haben ausschließlich den Status `No` und tragen keinen Mehrwert für die Analyse. Diese könnten entfernt werden, um die Effizienz und Modellgenauigkeit zu verbessern.
- Auffällige Wiedereinweisungsmuster in spezifischen Kategorien: Medikamente wie `pioglitazone` (`Up` : 19.1%) und selten genutzte Optionen wie `acarbose` (`Up` : 33.3%) weisen besonders hohe Wiedereinweisungsraten auf und könnten besondere Aufmerksamkeit erfordern.

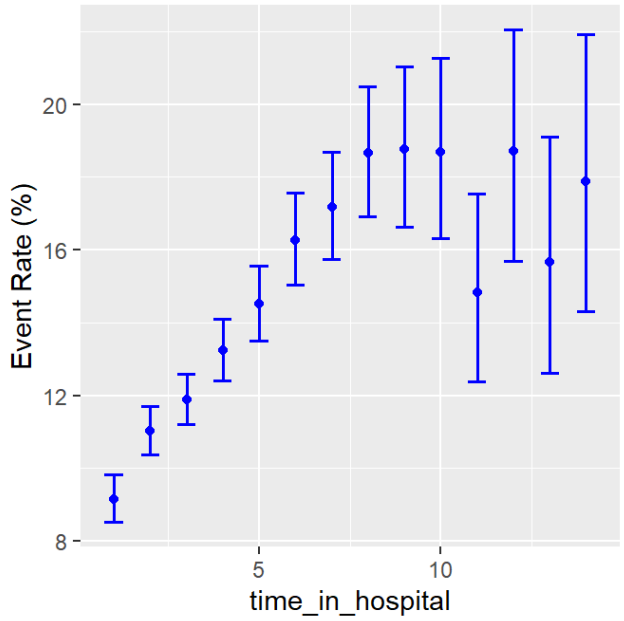
Zusammenfassend zeigt die Analyse eine hohe Dominanz von `No` in vielen Merkmalen, jedoch können spezifische Kategorien mit erhöhter Wiedereinweisungswahrscheinlichkeit gezielte Hinweise auf die Medikamentennutzung oder Anpassung geben. Merkmale ohne Variabilität könnten aus der Modellierung entfernt werden.

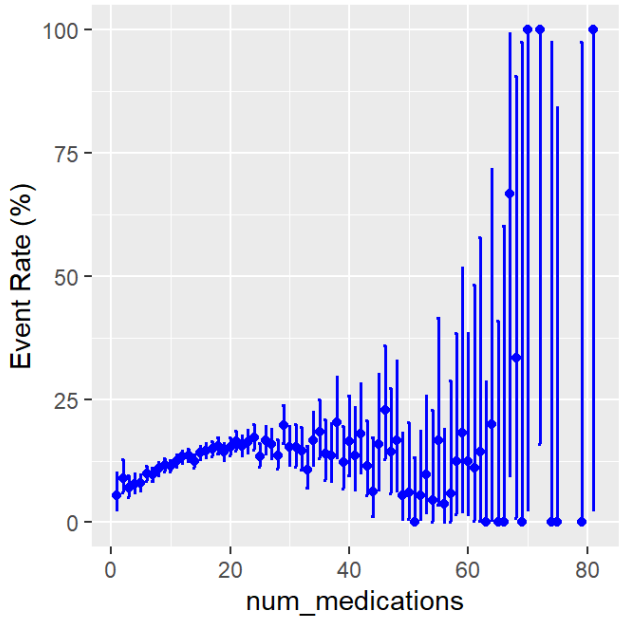
Numerische Merkmale zu Krankenhausressourcennutzung und Gesundheitszustand

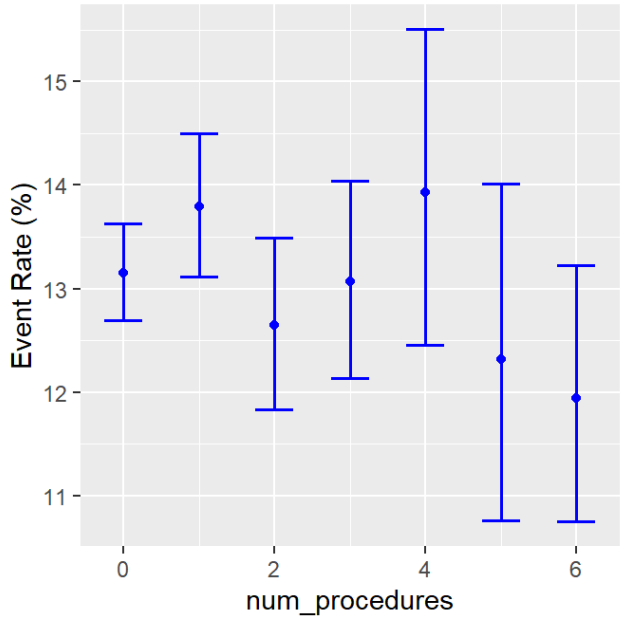
```
# Function to create individual plots with error bars
create_errorbar_plot <- function(data, feature) {
  data %>%
    group_by(!!sym(feature), TARGET) %>%
    count() %>%
    spread(TARGET, n, fill = 0) %>%
    mutate(frac_claim = `1`/(`1`+`0`)*100,
           lwr = get_binCI(`1`,(`1`+`0`))[[1]]*100,
           upr = get_binCI(`1`,(`1`+`0`))[[2]]*100) %>%
    ggplot(aes(x = !!sym(feature), y = frac_claim)) +
    geom_point(color = "blue") +
    geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "blue") +
    theme(legend.position = "none") +
    labs(x = feature, y = "Event Rate (%)")
}

# Define the features to plot
features <- c("number_inpatient", "num_lab_procedures", "time_in_hospital",
              "num_medications", "num_procedures", "number_diagnoses",
              "number_emergency", "number_outpatient")

# Generate the plots and store them in a list
plots <- lapply(features, create_errorbar_plot, data = diabetic_data_bin)

# Define the layout matrix (4 rows, 2 columns)
layout <- matrix(1:8, 4, 2, byrow=TRUE)

# Use multiplot to display the plots
multiplot(plotlist = plots, layout=layout)
```

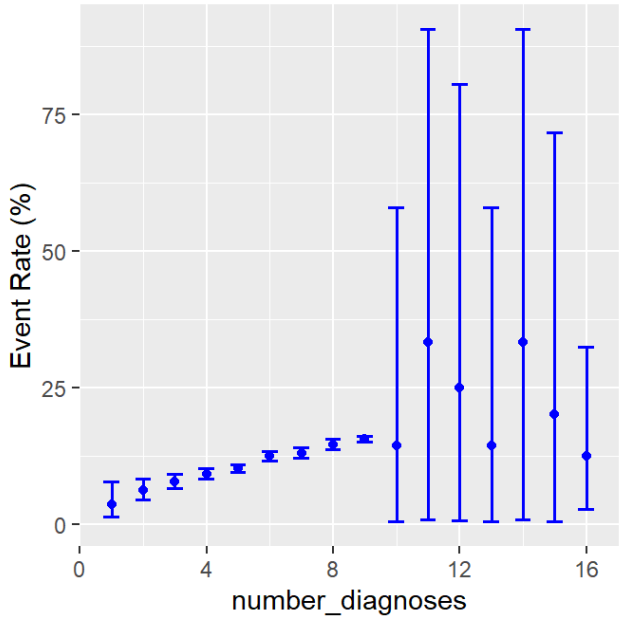

Event Rate (%)
number_inpatient

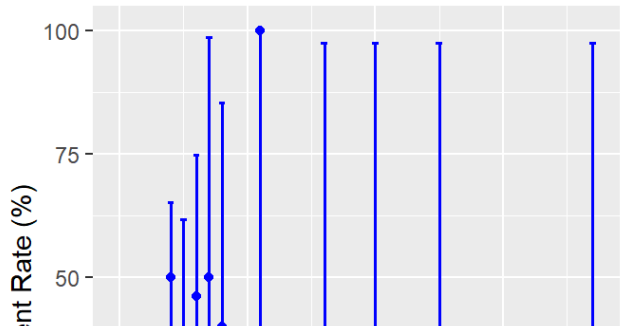

Event Rate (%)
num_lab_procedures

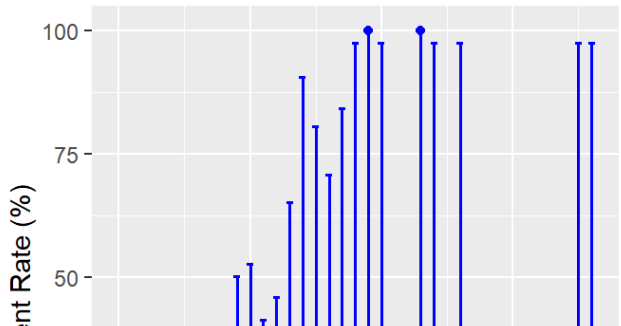

Event Rate (%)
time_in_hospital

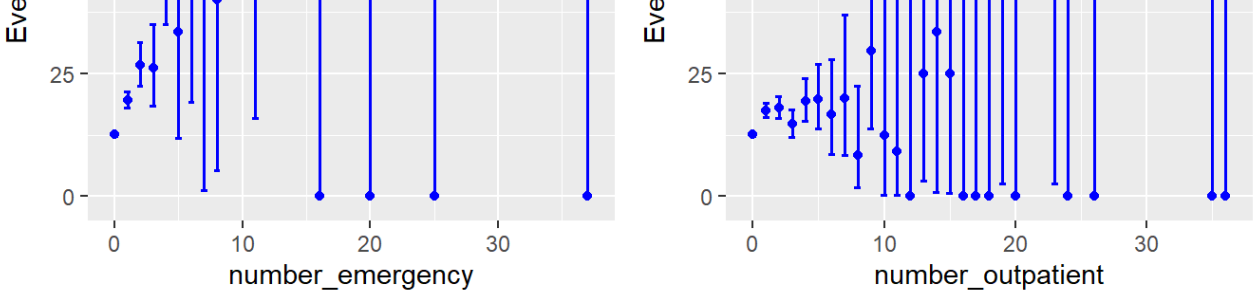

Event Rate (%)
num_medications


Event Rate (%)
num_procedures


Event Rate (%)
number_diagnoses







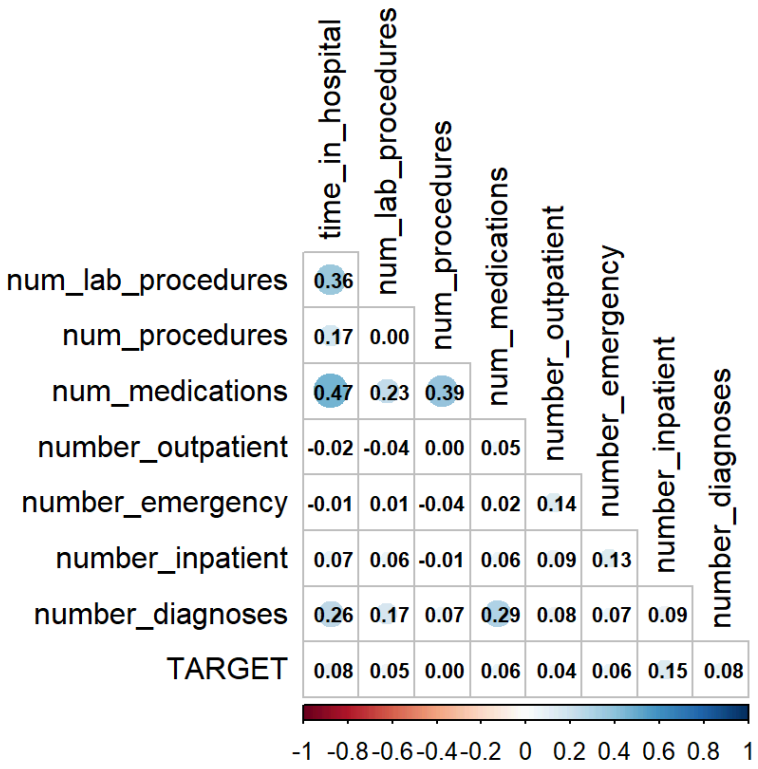
Die wichtigsten Erkenntnisse in dieser Merkmalsgruppe sind wie folgt:

- `number_inpatient` : Die Wiedereinweisungsraten steigen mit der Anzahl der stationären Aufenthalte kontinuierlich an und erreichen bei 6 Aufenthalten 78,6%. Ab 4 Aufenthalten gibt es jedoch nur noch sehr wenige Datenpunkte, was die Zuverlässigkeit der Ergebnisse einschränkt. Dieses Merkmal scheint ein vergleichsweise starker Prädiktor für die Wiedereinweisung zu sein.
- `num_lab_procedures` : Mit der Anzahl der Laborverfahren steigt tendenziell die relative Wiedereinweisungsrate, die bei höheren Anzahlen stark schwankt und höhere Konfidenzintervalle aufweist.
- `time_in_hospital` : Längere Krankenhausaufenthalte sind tendenziell mit höheren Wiedereinweisungsraten verbunden, insbesondere bei 6-10 Tagen Aufenthalt.
- `num_medications` : Mehr Medikamente entsprechen im Allgemeinen höheren Wiedereinweisungsraten, wobei die Streuung bei sehr hohen und seltenen Werten stark zunimmt.
- `num_procedures` : Die Wiedereinweisungsraten schwanken wenig und liegen im Bereich von 12,0% bis 13,9%, wobei die höchste Rate bei 4 Eingriffen (13,9%) und die niedrigste bei 6 Eingriffen (12,0%) liegt. Dieses Merkmal scheint keine großartige Prognosekraft zu haben.
- `number_diagnoses` : Eine höhere Anzahl von Diagnosen führt tendenziell zu einer erhöhten Wiedereinweisungsrate, mit deutlichen Sprüngen bei 9 oder mehr Diagnosen bei gleichzeitig viel höheren Konfidenzintervallen.
- `number_emergency` : Die Wiedereinweisungsraten steigen mit der Anzahl der Notfälle. Bei 0 Notfällen liegt die Rate bei 12,6%, während sie bei 6 Notfällen 46,2% erreicht. Es gibt auch einen starken Anstieg bei 4 Notfällen mit einer Wiedereinweisungsrate von 50%. Ab 3 sehr dünn besiedelte Datenbasis und somit große Konfidenzintervalle.
- `number_outpatient` : Patienten mit 1 bis 4 ambulanten Besuchen haben die höchsten Wiedereinweisungsraten, mit einem bemerkenswerten Höhepunkt bei 9 Besuchen.

d) 6 Multi-feature comparisons: Erstellen Sie eine Korrelationsmatrix wie in Abschnitt 6.1 der Vorlage. Diskutieren Sie die wichtigsten Erkenntnisse, insbesondere mit Blick auf deren Bedeutung für die Vorhersage der Zielvariablen.

Zunächst bestimmen wir die Korrelationsmatrix der numerischen Merkmale (inkl. `TARGET` , welches hierfür erst noch numerisch kodiert werden muss) und visualisieren diese im Anschluss.

```
# Verwende im neuen Datensatz TARGET numerisch kodiert
data_tmp <- diabetic_data_bin %>%
  mutate(TARGET = as.numeric(TARGET))

# Betrachte nur numerische Features für die spätere Korrelationsmatrix
numeric_features <- data_tmp %>%
  select_if(is.numeric)

# Berrechne Korrelationsmatrix
cor_matrix <- cor(numeric_features, use = "complete.obs", method = "spearman")

# Visualisiere Korrelationsmatrix
corrplot(cor_matrix, method = "circle", type = "lower", tl.col = "black",
         addCoef.col = "black", number.cex = 0.7, diag = FALSE)
```



Die wichtigsten Erkenntnisse aus der Korrelationsmatrix sind wie folgt:

- Die Zielvariable `TARGET` weist nur geringe Korrelationen mit anderen Variablen auf, wobei der stärkste Zusammenhang mit der Anzahl der Krankenhausaufenthalte (`number_inpatient`) besteht. Dies könnte darauf hindeuten, dass dieses Merkmal einen nicht zu vernachlässigenden Einfluss auf die Prognose der Wiedereinweisung innerhalb von 30 Tagen aufweisen wird.
- Die Dauer des Krankenhausaufenthalts (`time_in_hospital`) zeigt moderate bis mäßige positive Korrelationen mit der Anzahl der Medikamente (`num_medications`), der Anzahl der Labortests (`num_lab_procedures`) sowie der Anzahl der Diagnosen (`number_diagnoses`). Dies deutet darauf hin, dass Patienten, die länger im Krankenhaus bleiben, tendenziell komplexere Krankheitsbilder und mehr Behandlungen benötigen.
- Die Anzahl der Labortests (`num_lab_procedures`) und die Anzahl der weiteren Tests bzw. Behandlungen (`num_procedures`) haben eine schwache bis mäßige positive Korrelation. Auch dies deutet darauf hin, dass komplexere oder umfangreichere medizinische Untersuchungen und Behandlungen mit einer höheren Wahrscheinlichkeit für eine Wiedereinweisung innerhalb von 30 Tagen verbunden sein könnten.

Aufgabe R3: Datenaufteilung und XGBoost

In dieser Aufgabe teilen Sie die zwei Datensätze `diabetic_data_bin` (zur binären Klassifikation) und `diabetic_data_ter` (zur ternären Klassifikation) für die späteren Modellierungen in Trainings-, Validierungs- und Testdatensätze auf und wenden für die binäre Klassifikation mit XGBoost ein erstes Machine-Learning-Modell an. Ab Teilaufgabe R3b) sollen Code-Passagen aus der Vorlage *SWoF* entnommen und an einigen Stellen angepasst werden. Ziel der Anpassungen ist es, den Code mit minimalem Aufwand lauffähig zu machen, wobei die in Teilaufgabe R3a) aufbereiteten Trainings-, Validierungs- und Testdatensätze als Input dienen. In dieser und allen weiteren Aufgaben zur binären Klassifikation ist die "area under the ROC curve" (AUC) als Gütemaß von Machine-Learning-Modellen zu verwenden. Zu Beginn jeder der relevanten Teilaufgaben wird in Fettdruck angegeben, auf welche Abschnitte der Vorlage *SWoF* sich die jeweilige Teilaufgabe bezieht.

a) Datenaufteilung: Teilen Sie die beiden Datensätze `diabetic_data_bin` und `diabetic_data_ter` jeweils in Trainings-, Validierungs- und Testdatensätze im Verhältnis 70:15:15 auf. Achten Sie darauf, dass die Zielvariable `TARGET` bei der Aufteilung korrekt stratifiziert wird, um die Klassenverteilung beizubehalten. Geben Sie jeweils die Anzahl der Zeilen in den Trainings-, Validierungs- und Testdatensätzen sowie die Anteile der Klassen in den Teildatensätzen aus.

Um die späteren Modelle zu evaluieren, teilen wir die Datensätze `diabetic_data_bin` und `diabetic_data_ter` jeweils in Trainings-, Validierungs- und Testdatensätze auf, wobei 70% der Daten für das Training, 15% für die Validierung und 15% für das Testen verwendet werden. Diese Aufteilung stellt sicher, dass unsere Modelle auf einem großen Teil der Daten trainiert werden, während es immer noch auf nicht gesehenen Daten evaluiert wird, um ihre Güten zu beurteilen.

```r
# Funktion zur Aufteilung eines Datensatzes in Training, Validierung und Test
split_data <- function(data, target_variable, train_ratio = 0.7, val_ratio = 0.15) {
  # Erster Split: Training und Rest (Validation + Test)
  train_idx <- createDataPartition(data[[target_variable]], p = train_ratio, list = FALSE)
  train <- data[train_idx, ]
  rest <- data[-train_idx, ]

  # Zweiter Split: Validation und Test
  val_idx <- createDataPartition(rest[[target_variable]], p = val_ratio / (1 - train_ratio), list = FALSE)
  validation <- rest[val_idx, ]
  test <- rest[-val_idx, ]

  list(train = train, validation = validation, test = test)
}

# Aufteilung für binäre und ternäre Klassifikation
splits_binary <- split_data(diabetic_data_bin, "TARGET")
splits_ternary <- split_data(diabetic_data_ter, "TARGET")

# Einzelne Datensätze benennen
train_bin <- splits_binary$train
val_bin <- splits_binary$validation
test_bin <- splits_binary$test

train_ter <- splits_ternary$train
val_ter <- splits_ternary$validation
test_ter <- splits_ternary$test

# Funktion zur Ausgabe der Informationen für jede Aufteilung
print_split_info <- function(split, title) {
  cat(title, "Klassifikation:\n")
  cat("Train:", nrow(split$train), "Validation:", nrow(split$validation), "Test:", nrow(split$test), "\n")
  cat("TARGET-Verteilung (Train):", prop.table(table(split$train$TARGET)), "\n")
  cat("TARGET-Verteilung (Validation):", prop.table(table(split$validation$TARGET)), "\n")
  cat("TARGET-Verteilung (Test):", prop.table(table(split$test$TARGET)), "\n\n")
}

# Ergebnisse ausgeben
print_split_info(splits_binary, "Binäre")
```

```
## Binäre Klassifikation:
## Train: 33426 Validation: 7163 Test: 7162
## TARGET-Verteilung (Train): 0.8685454 0.1314546
## TARGET-Verteilung (Validation): 0.8684909 0.1315091
## TARGET-Verteilung (Test): 0.8686121 0.1313879
```

```r
print_split_info(splits_ternary, "Ternäre")
```

```
## Ternäre Klassifikation:
## Train: 48982 Validation: 10496 Test: 10495
## TARGET-Verteilung (Train): 0.08970642 0.3175861 0.5927075
## TARGET-Verteilung (Validation): 0.08974848 0.3175495 0.592702
## TARGET-Verteilung (Test): 0.08966174 0.3175798 0.5927585
```

Die Ausgabe bestätigt, dass die Aufteilung korrekt durchgeführt wurde. Für sowohl die binäre als auch die ternäre Klassifikation weisen die Trainings-, Validierungs- und Testdatensätze eine sehr ähnliche Klassenverteilung auf.

b) 8.3 XGBoost parameters and fitting: Verwenden Sie den Code aus dem angegebenen Abschnitt der Vorlage und stellen Sie sicher, dass er lauffähig ist. Passen Sie den Code sowie die Parameter dort an, wo es notwendig und sinnvoll ist, um das XGBoost-Modell richtig zu trainieren. Diskutieren Sie kurz die Bedeutung der verwendeten Parameter `colsample_bytree`, `subsample`, `max_depth` und `eta` im Kontext des XGBoost-Verfahrens und berechnen Sie den AUC-Wert auf Testdaten.

In dieser Aufgabe trainieren wir ein XGBoost-Modell zur Vorhersage von Krankenhauswiedereinweisungen. Dafür werden zunächst die Daten in das passende Format für XGBoost konvertiert. Anschließend passen wir die Parameter des Modells an, führen eine Kreuzvalidierung zur Bestimmung der optimalen Rundenzahl durch und trainieren das Modell. Abschließend evaluieren wir das Modell anhand der AUC-Metrik.

Zunächst konvertieren wir die Trainings- und Testdaten in das `DMatrix`-Format, das von XGBoost für effizientere Berechnungen genutzt wird.

```r
X_train <- train_bin %>% select(-TARGET)
X_val <- val_bin %>% select(-TARGET)
X_test <- test_bin %>% select(-TARGET)

# zur Umwandlung in binär (0/1) mit -1 versehen
y_train <- as.numeric(train_bin$TARGET) - 1
y_val <- as.numeric(val_bin$TARGET) - 1
y_test <- as.numeric(test_bin$TARGET) - 1

# Konvertiere Trainings-, Validierungs- und Testdaten in das DMatrix-Format
dtrain <- xgb.DMatrix(data.matrix(X_train), label = y_train)
dval <- xgb.DMatrix(data.matrix(X_val), label = y_val)
dtest <- xgb.DMatrix(data.matrix(X_test))
```

Nun definieren wir die Parameter des XGBoost-Modells. Dabei haben die Parameter folgende Bedeutung:

- `colsample_bytree` : Gibt den Anteil der Features an, die für das Training jedes Baums verwendet werden. Ein Wert von 0.7 bedeutet, dass 70 % der Features zufällig ausgewählt werden. Dies reduziert die Gefahr von Overfitting.
- `subsample` : Gibt den Anteil der Trainingsdaten an, der für das Training jedes Baums verwendet wird. Ein Wert von 0.7 sorgt für mehr Diversität zwischen den Bäumen.
- `max_depth` : Legt die maximale Tiefe der Bäume fest. Größere Werte erlauben komplexere Muster, erhöhen aber das Risiko von Overfitting.
- `eta` : Bestimmt die Lernrate, mit der die Modellparameter bei jedem Schritt aktualisiert werden. Kleinere Werte wie 0.1 führen zu langsameren, aber stabileren Verbesserungen.

Beachte, dass wir `eval_metric = "auc"` für die vorliegende binäre Klassifikation verwenden.

```
# Definiere die XGBoost-Parameter
xgb_params <- list(
  colsample_bytree = 0.7,
  subsample = 0.7,
  booster = "gbtree",
  max_depth = 5,
  eta = 0.1,
  eval_metric = "auc",
  objective = "binary:logistic",
  nthread = 4
)

watchlist <- list(train=dtrain, valid=dval)
```

Als Nächstes führen wir eine Kreuzvalidierung durch, um die optimale Anzahl an Iterationen (Boosting-Runden) zu bestimmen.

```
# Führe Kreuzvalidierung durch
xgb_cv <- xgb.cv(
  params = xgb_params,
  data = dtrain,
  early_stopping_rounds = 10,
  nfold = 5,
  nrounds = 100,
  maximize = TRUE,
  verbose = 0
)

# Speichere die optimale Anzahl an Runden
optimal_nrounds <- xgb_cv$best_iteration
```

Nun trainieren wir das Modell mit der optimalen Anzahl an Runden.

```
# Trainiere das XGBoost-Modell
gb_dt <- xgb.train(
  params = xgb_params,
  data = dtrain,
  print_every_n = 10,
  watchlist = watchlist,
  nrounds = optimal_nrounds,
  maximize = TRUE
)
```

```
## [1]  train-auc:0.668300  valid-auc:0.648974
## [11] train-auc:0.700069  valid-auc:0.679814
## [21] train-auc:0.707864  valid-auc:0.686211
## [31] train-auc:0.718680  valid-auc:0.690010
## [41] train-auc:0.733844  valid-auc:0.694602
## [51] train-auc:0.744228  valid-auc:0.699344
## [61] train-auc:0.752023  valid-auc:0.700282
## [71] train-auc:0.761650  valid-auc:0.702049
## [81] train-auc:0.768298  valid-auc:0.702361
## [91] train-auc:0.775590  valid-auc:0.701436
## [92] train-auc:0.776191  valid-auc:0.701548
```

Um die Modellleistung zu bewerten, berechnen wir den AUC-Wert auf den Testdaten.

```
# Berechnung und Ausgabe der AUC mit der calculate_auc Funktion
auc_xgb <- calculate_auc(y_test, predict(gb_dt, newdata = dtest))
```

```
## AUC: 0.7035
```

Das XGBoost-Modell wurde erfolgreich trainiert und erreicht einen guten AUC-Wert von ca. 0.7.

c) 8.4 Feature Importance: Berechnen Sie die Feature Importances des eben trainierten XGBoost-Modells und visualisieren Sie diese in einem geeigneten Plot. Identifizieren Sie die wichtigsten Merkmale und diskutieren Sie kurz qualitativ, warum diese für die Prognose von Wiedereinweisungen relevant sein könnten (es wird kein spezifisches Krankenversicherungs- oder Gesundheitswissen vorausgesetzt). Geben Sie kurz an, welche Implikationen die Ergebnisse für die Modellierung, die Vorhersagegüte und die Gestaltung eines präventiven Versicherungsschutzes haben könnten.

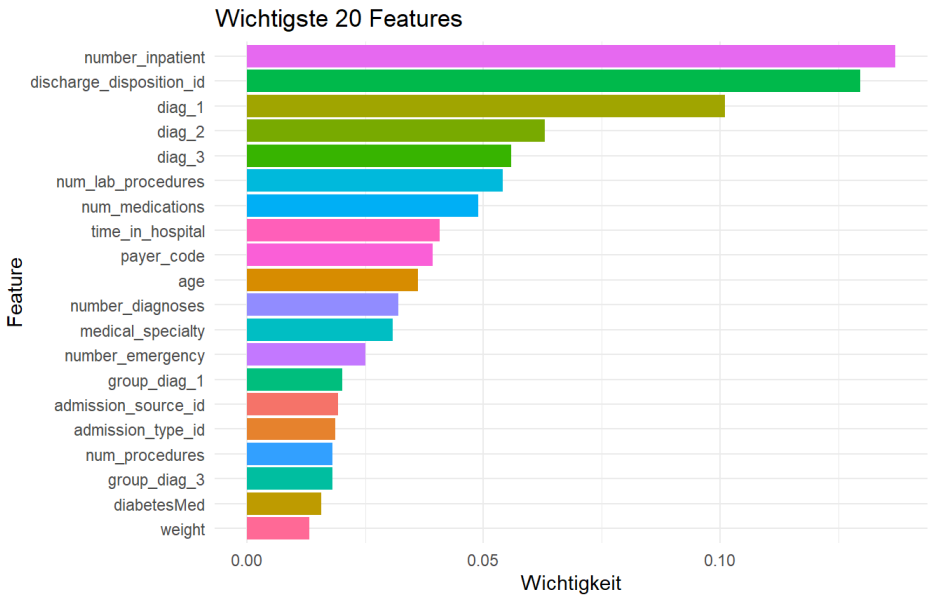
Die Analyse der Feature Importance ermöglicht es uns, die wichtigsten Variablen für die Vorhersage zu identifizieren. Dies gibt Einblick in die Daten und unterstützt die Modellinterpretation.

Zunächst berechnen wir die Feature Importance mit der Funktion `xgb.importance` und speichern die Ergebnisse in einem Tibble.

```
# Berechne Feature Importance
imp_matrix <- as_tibble(xgb.importance(
  feature_names = colnames(X_train),
  model = gb_dt
))
```

Um die Ergebnisse anschaulich darzustellen, visualisieren wir die 20 wichtigsten Features in einem Balkendiagramm.

```
# Visualisiere die 20 wichtigsten Features
imp_matrix %>%
  head(20) %>%
  ggplot(aes(x = reorder(Feature, Gain, FUN = max), y = Gain, fill = Feature)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  labs(
    title = "Wichtigste 20 Features",
    x = "Feature",
    y = "Wichtigkeit"
  ) +
  theme_minimal()
```



Abschließend diskutieren wir die wichtigsten Merkmale und deren Relevanz für die Prognose von Wiedereinweisungen.

Wichtigste Merkmale:

- `number_inpatient` : Die Anzahl der Krankenhausaufenthalte im Jahr vor der Einweisung spiegelt den Gesundheitszustand und die Komplexität des Patienten wider.
- `discharge_disposition_id` : Gibt Informationen über den Zustand des Patienten nach der Entlassung, wie z.

  B. Verlegung in eine Pflegeeinrichtung oder die Entlassung nach Hause.
- Diagnosen (`diag_1` , `diag_2` , `diag_3`): Entscheidend für die Einschätzung der Schwere und Art der Erkrankung.
- `num_lab_procedures` : Die Anzahl der durchgeführten Laboruntersuchungen gibt Hinweise auf die diagnostische Komplexität und den Schweregrad der Erkrankung. Ein hoher Wert kann auf schwerwiegende oder schwer zu diagnostizierende Zustände hinweisen.
- `num_medications` : Die Anzahl der verschriebenen Medikamente ist ein Indikator für die Behandlungsintensität und die Komplexität des Gesundheitszustands. Patienten mit vielen Medikamenten stehen oft unter höherem Risiko für unerwünschte Arzneimittelwirkungen oder schlechte Therapietreue, was eine erneute Einweisung wahrscheinlicher macht.

Die genannten Merkmale sind logisch mit dem Risiko einer Wiedereinweisung verknüpft. Zum Beispiel weisen häufigere Krankenhausaufenthalte oder eine höhere Zahl an Laboruntersuchungen und Medikamenten auf eine erhöhte Krankheitslast hin. Gleichzeitig geben Entlassungsgründe und Diagnosen spezifische Hinweise auf den Zustand des Patienten nach der Entlassung und die Wahrscheinlichkeit erneuter Komplikationen.

Die Ergebnisse legen darüber hinaus nahe, dass sich die Modellgenauigkeit durch Fokussierung auf diese Merkmale weiter verbessern lässt. Die Interpretierbarkeit der wichtigsten Features kann helfen, Interventionen zu entdecken (siehe Aufgabe R4) und zu priorisieren, um Krankenhauswiedereinweisungen zu reduzieren.

Aufgabe R4: Logistische Regressionen ohne und mit Interaktionen

Nachdem in Aufgabe R3 ein XGBoost-Modell für die binäre Klassifikation betrachtet wurde, wenden wir uns in dieser Aufgabe der Logistischen Regression zu. Dabei betrachten wir sowohl eine Logistische Regression ohne Interaktionen als auch ein Modell mit Interaktionen. Als Eingabedaten dienen die in Teilaufgabe R3a) aufbereiteten Trainings- und Testdatensätze.

a) Logistische Regression ohne Interaktionen: Trainieren Sie eine einfache Logistische Regression, die alle relevanten Prädiktoren (ohne Interaktionen) verwendet. Wählen Sie nur Merkmale aus, die mindestens zwei verschiedene Werte haben, und schließen Sie die Zielvariable sowie die Diagnosespalten (`diag_1` , `diag_2` , `diag_3`) aus. Diskutieren Sie in vier bis fünf Stichpunkten die wichtigsten Erkenntnisse aus den Koeffizienten der erhaltenen Logistischen Regression. Berechnen Sie den AUC-Wert dieses Modells auf den Testdaten und geben Sie das Ergebnis aus. Erstellen und visualisieren Sie eine Konfusionsmatrix, die die Vorhersagen des Modells mit den tatsächlichen Werten der Testdaten gegenüberstellt. Erörtern Sie zunächst kurz die Herausforderungen bei der Wahl eines geeigneten Schwellenwerts, beispielsweise in Bezug auf die Balance zwischen Sensitivität und Spezifität. Verwenden Sie danach den Schwellenwert 0,2 für die Erstellung der Konfusionsmatrix.

Wir implementieren nun eine einfache Logistische Regression unter Verwendung aller relevanten Prädiktoren, ohne Interaktionen, und wählen nur Merkmale aus, die mindestens zwei verschiedene Werte aufweisen. Die Zielvariable sowie die Diagnosespalten `diag_1` , `diag_2` und `diag_3` werden dabei ausgeschlossen (letztere aufgrund der zu großen Anzahl an verschiedenen Werten).

```
# Filtere nur die Merkmale aus, die mehr als einen einzigartigen Wert haben
valid_features <- colnames(train_bin)[sapply(train_bin, function(col) length(unique(col)) > 1)]

# Exkludiere die Zielvariable und die Diagnosespalten
valid_features <- setdiff(valid_features, c("TARGET", "diag_1", "diag_2", "diag_3"))

# Erstelle die Formel unter Verwendung der validen Merkmale
formula <- as.formula(paste("TARGET ~", paste(valid_features, collapse = " + ")))

# Fit der logistischen Regression
glm1 <- glm(formula, data = train_bin, family = binomial)
```

Zunächst geben wir die Koeffizienten der Logistischen Regression aus:

```
# Zusammenfassung des Modells
summary(glm1)
```

```
##
## Call:
## glm(formula = formula, family = binomial, data = train_bin)
##
## Coefficients:
##                                                  Estimate Std. Error
## (Intercept)                                    -1.734e+01  4.310e+02
## raceAfricanAmerican                             2.410e-01  1.189e-01
## raceAsian                                       2.064e-01  2.187e-01
## raceCaucasian                                   2.383e-01  1.128e-01
## raceHispanic                                   -1.098e-01  1.727e-01
## raceOther                                      -1.664e-01  1.858e-01
## genderMale                                     -1.513e-02  3.476e-02
## age[10-20)                                      4.379e-01  7.945e-01
## age[20-30)                                      4.268e-01  7.776e-01
## age[30-40)                                      3.537e-01  7.704e-01
## age[40-50)                                      5.438e-01  7.667e-01
## age[50-60)                                      4.041e-01  7.662e-01
## age[60-70)                                      6.382e-01  7.661e-01
## age[70-80)                                      7.401e-01  7.663e-01
## age[80-90)                                      6.366e-01  7.669e-01
## age[90-100)                                     3.949e-01  7.730e-01
## weight[0-25)                                    1.979e+00  6.288e-01
## weight[25-50)                                  -1.341e-02  4.721e-01
## weight[50-75)                                   2.329e-01  1.632e-01
## weight[75-100)                                  7.185e-01  1.274e-01
## weight[100-125)                                 7.958e-01  1.792e-01
## weight[125-150)                                 2.925e-01  4.138e-01
## weight[150-175)                                -1.337e+01  4.230e+02
## admission_type_id2                              5.563e-02  6.871e-02
## admission_type_id3                             -1.415e-01  7.969e-02
## admission_type_id5                              6.433e-02  1.194e-01
## admission_type_id6                              4.046e-01  9.789e-02
## admission_type_id7                             -1.344e+01  3.970e+02
## admission_type_id8                             -3.725e-01  3.030e-01
## discharge_disposition_id2                       6.012e-01  9.796e-02
## discharge_disposition_id3                       5.621e-01  5.516e-02
## discharge_disposition_id4                       3.630e-01  1.861e-01
## discharge_disposition_id5                       1.246e+00  1.116e-01
## discharge_disposition_id6                       2.744e-01  5.611e-02
## discharge_disposition_id7                       1.672e-01  2.261e-01
## discharge_disposition_id8                       5.256e-01  4.575e-01
## discharge_disposition_id15                      2.590e+00  5.682e-01
## discharge_disposition_id18                      2.038e-01  9.346e-02
## discharge_disposition_id22                      1.526e+00  8.874e-02
## discharge_disposition_id23                     -1.002e+00  4.228e-01
## discharge_disposition_id24                     -1.341e+01  4.147e+02
## discharge_disposition_id25                     -5.287e-01  2.119e-01
## discharge_disposition_id28                      1.969e+00  3.085e-01
## admission_source_id2                           -4.336e-01  1.576e-01
## admission_source_id3                            6.816e-01  2.870e-01
## admission_source_id4                           -4.632e-01  1.005e-01
## admission_source_id5                           -6.588e-01  2.020e-01
## admission_source_id6                           -2.201e-01  1.346e-01
## admission_source_id7                           -4.405e-02  7.022e-02
## admission_source_id8                            3.211e-01  1.137e+00
## admission_source_id9                            9.877e-02  4.517e-01
## admission_source_id17                          -4.984e-01  1.084e-01
## admission_source_id20                           7.077e-01  3.336e-01
## time_in_hospital                                9.672e-03  7.109e-03
## payer_codeBC                                   -4.426e-01  9.589e-02
## payer_codeCH                                   -8.684e-01  6.016e-01
## payer_codeCM                                   -5.265e-01  1.342e-01
## payer_codeCP                                   -3.890e-01  1.183e-01
## payer_codeDM                                   -3.408e-01  2.542e-01
## payer_codeHM                                   -2.174e-01  8.397e-02
## payer_codeMC                                   -2.620e-01  4.714e-02
## payer_codeMD                                   -1.430e-01  1.067e-01
## payer_codeMP                                   -2.255e-01  1.052e+00
## payer_codeOG                                   -1.419e-01  1.866e-01
## payer_codeOT                                   -8.230e-01  7.488e-01
## payer_codePO                                   -3.693e-01  2.360e-01
## payer_codeSI                                    8.250e-02  7.613e-01
## payer_codeSP                                   -1.362e-01  9.072e-02
## payer_codeUN                                   -3.001e-01  1.168e-01
## payer_codeWC                                   -9.847e-01  6.050e-01
## medical_specialtyAnesthesiology-Pediatric       3.862e-01  1.102e+00
## medical_specialtyCardiology                    -2.110e-01  8.361e-02
## medical_specialtyEmergency/Trauma              -1.171e-01  8.707e-02
## medical_specialtyEndocrinology                 -1.005e-01  4.845e-01
## medical_specialtyFamily/GeneralPractice         6.226e-02  6.876e-02
## medical_specialtyGastroenterology               3.827e-01  2.145e-01
## medical_specialtyGynecology                    -1.315e+01  2.469e+02
## medical_specialtyHematology                    -1.710e-01  7.836e-01
## medical_specialtyHematology/Oncology            6.077e-01  3.568e-01
```

```
## medical_specialtyHospitalist                          -3.666e-01  1.062e+00
## medical_specialtyInfectiousDiseases                    1.028e+00  6.061e-01
## medical_specialtyInternalMedicine                     -8.061e-02  5.269e-02
## medical_specialtyNephrology                            5.818e-01  1.484e-01
## medical_specialtyNeurology                            -3.646e-01  3.858e-01
## medical_specialtyObsterics&Gynecology-GynecologicOnco -1.375e+01  4.084e+02
## medical_specialtyObstetrics                           -1.308e+01  5.120e+02
## medical_specialtyObstetricsandGynecology              -6.278e-01  2.542e-01
## medical_specialtyOncology                              6.500e-01  2.379e-01
## medical_specialtyOphthalmology                        -4.669e-01  1.041e+00
## medical_specialtyOrthopedics                          -6.010e-02  1.437e-01
## medical_specialtyOrthopedics-Reconstructive           -8.398e-01  1.819e-01
## medical_specialtyOsteopath                             5.132e-01  6.708e-01
## medical_specialtyOtolaryngology                       -1.327e+01  1.800e+02
## medical_specialtyPediatrics                           -7.850e-01  4.482e-01
## medical_specialtyPediatrics-CriticalCare              -1.299e+01  2.408e+02
## medical_specialtyPediatrics-Endocrinology             -1.288e+01  1.525e+02
## medical_specialtyPhysicalMedicineandRehabilitation     6.026e-01  2.584e-01
## medical_specialtyPodiatry                              6.708e-03  6.376e-01
## medical_specialtyPsychiatry                            4.080e-01  1.683e-01
## medical_specialtyPsychology                           -1.057e+00  1.026e+00
## medical_specialtyPulmonology                           7.809e-02  1.667e-01
## medical_specialtyRadiologist                          -3.168e-01  1.833e-01
## medical_specialtyRadiology                             1.068e-01  6.427e-01
## medical_specialtyRheumatology                         -1.349e+01  7.129e+02
## medical_specialtySurgeon                              -3.843e-01  7.575e-01
## medical_specialtySurgery-Cardiovascular               -1.362e+00  7.314e-01
## medical_specialtySurgery-Cardiovascular/Thoracic      -6.686e-01  2.334e-01
## medical_specialtySurgery-General                      -1.240e-01  1.030e-01
## medical_specialtySurgery-Neuro                        -9.295e-01  3.094e-01
## medical_specialtySurgery-Plastic                       2.888e-02  8.112e-01
## medical_specialtySurgery-Thoracic                     -4.355e-01  5.393e-01
## medical_specialtySurgery-Vascular                      4.034e-01  2.131e-01
## medical_specialtySurgicalSpecialty                    -1.305e+01  4.156e+02
## medical_specialtyUrology                               3.596e-03  2.013e-01
## num_lab_procedures                                     1.009e-03  1.073e-03
## num_procedures                                        -1.365e-02  1.197e-02
## num_medications                                        6.129e-03  2.804e-03
## number_outpatient                                      2.066e-02  1.613e-02
## number_emergency                                       1.862e-01  3.574e-02
## number_inpatient                                       5.194e-01  2.664e-02
## number_diagnoses                                       5.896e-02  1.121e-02
## max_glu_serum>300                                      2.510e-03  2.206e-01
## max_glu_serumNone                                     -9.950e-02  1.621e-01
## max_glu_serumNorm                                      1.530e-01  1.800e-01
## A1Cresult>8                                           -8.322e-03  1.038e-01
## A1CresultNone                                          8.455e-02  8.726e-02
## A1CresultNorm                                          2.737e-04  1.110e-01
## metforminNo                                            1.583e-02  2.047e-01
## metforminSteady                                       -1.159e-01  2.046e-01
## metforminUp                                           -3.605e-01  2.664e-01
## repaglinideNo                                          1.355e+01  4.310e+02
## repaglinideSteady                                      1.375e+01  4.310e+02
## repaglinideUp                                          1.414e+01  4.310e+02
## nateglinideSteady                                     -7.337e-02  2.199e-01
## nateglinideUp                                         -1.500e+01  4.065e+02
## chlorpropamideSteady                                  -1.416e+00  1.038e+00
## glimepirideNo                                         -1.498e-01  3.581e-01
## glimepirideSteady                                     -2.594e-01  3.640e-01
## glimepirideUp                                         -1.671e-01  4.488e-01
## glipizideNo                                           -3.432e-01  2.112e-01
## glipizideSteady                                       -3.214e-01  2.118e-01
## glipizideUp                                           -1.720e-01  2.660e-01
## glyburideNo                                            1.394e-01  2.199e-01
## glyburideSteady                                        1.416e-01  2.199e-01
## glyburideUp                                           -1.882e-02  2.781e-01
## tolbutamideSteady                                     -8.027e-01  1.061e+00
## pioglitazoneNo                                         3.024e-02  4.751e-01
## pioglitazoneSteady                                    -5.866e-02  4.783e-01
## pioglitazoneUp                                         3.362e-01  5.579e-01
## rosiglitazoneNo                                        7.238e-01  6.247e-01
## rosiglitazoneSteady                                    7.724e-01  6.275e-01
## rosiglitazoneUp                                        7.623e-01  7.181e-01
## acarboseSteady                                        -5.144e-02  3.502e-01
## acarboseUp                                            -1.336e+01  1.022e+03
## miglitolSteady                                        -1.379e+01  7.014e+02
## tolazamideSteady                                      -1.365e+01  4.161e+02
## insulinNo                                             -1.452e-01  9.243e-02
## insulinSteady                                         -1.464e-01  7.064e-02
## insulinUp                                             -9.112e-02  7.260e-02
## glyburide.metforminSteady                              7.333e-02  2.076e-01
## changeNo                                               3.720e-02  6.446e-02
## diabetesMedYes                                         3.181e-01  6.277e-02
## group_diag_1Diabetes                                   2.248e-02  7.454e-02
## group_diag_1Digestive                                 -1.533e-01  7.051e-02
## group_diag_1Genitourinary                             -1.315e-01  8.559e-02
```

```
## group_diag_1Injury                          -2.101e-01  7.436e-02
## group_diag_1Musculoskeletal                 -2.615e-01  9.568e-02
## group_diag_1n.a.                             7.644e-01  8.948e-01
## group_diag_1Neoplasms                        3.086e-02  8.708e-02
## group_diag_1Other                           -2.186e-01  5.760e-02
## group_diag_1Respiratory                     -2.944e-01  6.168e-02
## group_diag_2Diabetes                         1.524e-01  6.053e-02
## group_diag_2Digestive                       -1.246e-01  1.012e-01
## group_diag_2Genitourinary                   -1.353e-01  7.051e-02
## group_diag_2Injury                          -1.291e-01  1.108e-01
## group_diag_2Musculoskeletal                 -1.284e-01  1.358e-01
## group_diag_2n.a.                            -1.035e-01  4.016e-01
## group_diag_2Neoplasms                        1.389e-01  9.602e-02
## group_diag_2Other                            7.959e-03  4.838e-02
## group_diag_2Respiratory                     -4.582e-02  6.462e-02
## group_diag_3Diabetes                         8.035e-02  5.306e-02
## group_diag_3Digestive                        1.158e-01  9.677e-02
## group_diag_3Genitourinary                    5.448e-02  7.516e-02
## group_diag_3Injury                          -4.534e-02  1.233e-01
## group_diag_3Musculoskeletal                  2.003e-02  1.245e-01
## group_diag_3n.a.                            -3.128e-02  1.849e-01
## group_diag_3Neoplasms                        4.652e-02  9.947e-02
## group_diag_3Other                           -3.052e-02  4.673e-02
## group_diag_3Respiratory                      4.111e-02  7.213e-02
##                                             z value Pr(>|z|)
## (Intercept)                                  -0.040 0.967908
## raceAfricanAmerican                           2.027 0.042631 *
## raceAsian                                     0.944 0.345285
## raceCaucasian                                 2.114 0.034537 *
## raceHispanic                                 -0.636 0.524903
## raceOther                                    -0.896 0.370332
## genderMale                                   -0.435 0.663327
## age[10-20)                                    0.551 0.581501
## age[20-30)                                    0.549 0.583102
## age[30-40)                                    0.459 0.646108
## age[40-50)                                    0.709 0.478137
## age[50-60)                                    0.527 0.597950
## age[60-70)                                    0.833 0.404777
## age[70-80)                                    0.966 0.334145
## age[80-90)                                    0.830 0.406510
## age[90-100)                                   0.511 0.609445
## weight[0-25)                                  3.148 0.001646 **
## weight[25-50)                                -0.028 0.977343
## weight[50-75)                                 1.427 0.153594
## weight[75-100)                                5.642 1.68e-08 ***
## weight[100-125)                               4.441 8.95e-06 ***
## weight[125-150)                               0.707 0.479654
## weight[150-175)                              -0.032 0.974778
## admission_type_id2                            0.810 0.418105
## admission_type_id3                           -1.775 0.075842 .
## admission_type_id5                            0.539 0.590055
## admission_type_id6                            4.133 3.58e-05 ***
## admission_type_id7                           -0.034 0.972995
## admission_type_id8                           -1.229 0.218901
## discharge_disposition_id2                     6.137 8.42e-10 ***
## discharge_disposition_id3                    10.190  < 2e-16 ***
## discharge_disposition_id4                     1.951 0.051064 .
## discharge_disposition_id5                    11.165  < 2e-16 ***
## discharge_disposition_id6                     4.891 1.00e-06 ***
## discharge_disposition_id7                     0.740 0.459591
## discharge_disposition_id8                     1.149 0.250639
## discharge_disposition_id15                    4.558 5.17e-06 ***
## discharge_disposition_id18                    2.180 0.029252 *
## discharge_disposition_id22                   17.190  < 2e-16 ***
## discharge_disposition_id23                   -2.369 0.017817 *
## discharge_disposition_id24                   -0.032 0.974200
## discharge_disposition_id25                   -2.496 0.012572 *
## discharge_disposition_id28                    6.383 1.73e-10 ***
## admission_source_id2                         -2.752 0.005927 **
## admission_source_id3                          2.375 0.017560 *
## admission_source_id4                         -4.609 4.04e-06 ***
## admission_source_id5                         -3.261 0.001109 **
## admission_source_id6                         -1.635 0.102016
## admission_source_id7                         -0.627 0.530472
## admission_source_id8                          0.282 0.777572
## admission_source_id9                          0.219 0.826929
## admission_source_id17                        -4.595 4.32e-06 ***
## admission_source_id20                         2.121 0.033885 *
## time_in_hospital                              1.360 0.173699
## payer_codeBC                                 -4.616 3.92e-06 ***
## payer_codeCH                                 -1.443 0.148885
## payer_codeCM                                 -3.924 8.72e-05 ***
## payer_codeCP                                 -3.289 0.001004 **
## payer_codeDM                                 -1.340 0.180132
## payer_codeHM                                 -2.589 0.009620 **
## payer_codeMC                                 -5.559 2.72e-08 ***
```

```
## payer_codeMD                                        -1.340 0.180280
## payer_codeMP                                        -0.214 0.830253
## payer_codeOG                                        -0.760 0.446957
## payer_codeOT                                        -1.099 0.271739
## payer_codePO                                        -1.565 0.117657
## payer_codeSI                                         0.108 0.913708
## payer_codeSP                                        -1.502 0.133205
## payer_codeUN                                        -2.569 0.010204 *
## payer_codeWC                                        -1.628 0.103604
## medical_specialtyAnesthesiology-Pediatric            0.350 0.726063
## medical_specialtyCardiology                         -2.524 0.011600 *
## medical_specialtyEmergency/Trauma                   -1.346 0.178457
## medical_specialtyEndocrinology                      -0.207 0.835719
## medical_specialtyFamily/GeneralPractice              0.905 0.365227
## medical_specialtyGastroenterology                    1.784 0.074359 .
## medical_specialtyGynecology                         -0.053 0.957505
## medical_specialtyHematology                         -0.218 0.827296
## medical_specialtyHematology/Oncology                 1.703 0.088578 .
## medical_specialtyHospitalist                        -0.345 0.729925
## medical_specialtyInfectiousDiseases                  1.696 0.089980 .
## medical_specialtyInternalMedicine                   -1.530 0.126071
## medical_specialtyNephrology                          3.921 8.82e-05 ***
## medical_specialtyNeurology                          -0.945 0.344560
## medical_specialtyObsterics&Gynecology-GynecologicOnco -0.034 0.973152
## medical_specialtyObstetrics                         -0.026 0.979615
## medical_specialtyObstetricsandGynecology            -2.470 0.013528 *
## medical_specialtyOncology                            2.732 0.006290 **
## medical_specialtyOphthalmology                      -0.449 0.653713
## medical_specialtyOrthopedics                        -0.418 0.675725
## medical_specialtyOrthopedics-Reconstructive         -4.618 3.88e-06 ***
## medical_specialtyOsteopath                           0.765 0.444182
## medical_specialtyOtolaryngology                     -0.074 0.941228
## medical_specialtyPediatrics                         -1.751 0.079866 .
## medical_specialtyPediatrics-CriticalCare            -0.054 0.956960
## medical_specialtyPediatrics-Endocrinology           -0.084 0.932670
## medical_specialtyPhysicalMedicineandRehabilitation   2.332 0.019697 *
## medical_specialtyPodiatry                            0.011 0.991605
## medical_specialtyPsychiatry                          2.424 0.015351 *
## medical_specialtyPsychology                         -1.030 0.302932
## medical_specialtyPulmonology                         0.469 0.639373
## medical_specialtyRadiologist                        -1.729 0.083852 .
## medical_specialtyRadiology                           0.166 0.868060
## medical_specialtyRheumatology                       -0.019 0.984899
## medical_specialtySurgeon                            -0.507 0.611946
## medical_specialtySurgery-Cardiovascular             -1.862 0.062600 .
## medical_specialtySurgery-Cardiovascular/Thoracic    -2.864 0.004178 **
## medical_specialtySurgery-General                    -1.205 0.228356
## medical_specialtySurgery-Neuro                      -3.005 0.002660 **
## medical_specialtySurgery-Plastic                     0.036 0.971604
## medical_specialtySurgery-Thoracic                   -0.807 0.419416
## medical_specialtySurgery-Vascular                    1.893 0.058385 .
## medical_specialtySurgicalSpecialty                  -0.031 0.974951
## medical_specialtyUrology                             0.018 0.985747
## num_lab_procedures                                   0.941 0.346769
## num_procedures                                      -1.140 0.254434
## num_medications                                      2.186 0.028834 *
## number_outpatient                                    1.281 0.200306
## number_emergency                                     5.209 1.90e-07 ***
## number_inpatient                                    19.499  < 2e-16 ***
## number_diagnoses                                     5.259 1.45e-07 ***
## max_glu_serum>300                                    0.011 0.990923
## max_glu_serumNone                                   -0.614 0.539337
## max_glu_serumNorm                                    0.850 0.395527
## A1Cresult>8                                         -0.080 0.936087
## A1CresultNone                                        0.969 0.332584
## A1CresultNorm                                        0.002 0.998033
## metforminNo                                          0.077 0.938363
## metforminSteady                                     -0.566 0.571076
## metforminUp                                         -1.353 0.175955
## repaglinideNo                                        0.031 0.974917
## repaglinideSteady                                    0.032 0.974555
## repaglinideUp                                        0.033 0.973820
## nateglinideSteady                                   -0.334 0.738638
## nateglinideUp                                       -0.037 0.970568
## chlorpropamideSteady                                -1.364 0.172422
## glimepirideNo                                       -0.418 0.675812
## glimepirideSteady                                   -0.713 0.476091
## glimepirideUp                                       -0.372 0.709702
## glipizideNo                                         -1.625 0.104095
## glipizideSteady                                     -1.517 0.129210
## glipizideUp                                         -0.647 0.517819
## glyburideNo                                          0.634 0.526142
## glyburideSteady                                      0.644 0.519829
## glyburideUp                                         -0.068 0.946058
## tolbutamideSteady                                   -0.756 0.449354
## pioglitazoneNo                                       0.064 0.949246
```

```
## pioglitazoneSteady                                -0.123 0.902387
## pioglitazoneUp                                     0.603 0.546737
## rosiglitazoneNo                                    1.159 0.246567
## rosiglitazoneSteady                                1.231 0.218373
## rosiglitazoneUp                                    1.062 0.288446
## acarboseSteady                                    -0.147 0.883239
## acarboseUp                                        -0.013 0.989575
## miglitolSteady                                    -0.020 0.984313
## tolazamideSteady                                  -0.033 0.973831
## insulinNo                                         -1.570 0.116304
## insulinSteady                                     -2.073 0.038199 *
## insulinUp                                         -1.255 0.209460
## glyburide.metforminSteady                          0.353 0.723953
## changeNo                                           0.577 0.563853
## diabetesMedYes                                     5.067 4.03e-07 ***
## group_diag_1Diabetes                               0.302 0.762968
## group_diag_1Digestive                             -2.174 0.029676 *
## group_diag_1Genitourinary                         -1.537 0.124372
## group_diag_1Injury                                -2.825 0.004721 **
## group_diag_1Musculoskeletal                       -2.733 0.006279 **
## group_diag_1n.a.                                   0.854 0.392958
## group_diag_1Neoplasms                              0.354 0.723103
## group_diag_1Other                                 -3.795 0.000148 ***
## group_diag_1Respiratory                           -4.773 1.81e-06 ***
## group_diag_2Diabetes                               2.518 0.011812 *
## group_diag_2Digestive                             -1.231 0.218319
## group_diag_2Genitourinary                         -1.919 0.054944 .
## group_diag_2Injury                                -1.165 0.243856
## group_diag_2Musculoskeletal                       -0.946 0.344176
## group_diag_2n.a.                                  -0.258 0.796625
## group_diag_2Neoplasms                              1.446 0.148083
## group_diag_2Other                                  0.165 0.869326
## group_diag_2Respiratory                           -0.709 0.478276
## group_diag_3Diabetes                               1.514 0.129999
## group_diag_3Digestive                              1.196 0.231514
## group_diag_3Genitourinary                          0.725 0.468518
## group_diag_3Injury                                -0.368 0.713116
## group_diag_3Musculoskeletal                        0.161 0.872178
## group_diag_3n.a.                                  -0.169 0.865654
## group_diag_3Neoplasms                              0.468 0.640059
## group_diag_3Other                                 -0.653 0.513647
## group_diag_3Respiratory                            0.570 0.568676
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 26015  on 33425  degrees of freedom
## Residual deviance: 23900  on 33237  degrees of freedom
## AIC: 24278
##
## Number of Fisher Scoring iterations: 14
```

Erkenntnisse aus den Koeffizienten:

- `discharge_disposition_id` : Hier werden viele Ausprägungen als signifikant angesehen, was in Übereinstimmung mit den Erkenntnissen der EDA aus Aufgabe R2 steht.
- `number_emergency` , `number_inpatient` , `number_diagnoses` : Die positiven Koeffizienten und deren Signifikanz weisen darauf hin, dass höhere Werte dieser Anzahlen mit einer höheren Wiedereinweisungsrate einhergehen.
- `num_lab_procedures` : Bemerkenswert ist, dass dieses Merkmal, welches in Teilaufgabe R3c) noch als das sechstwichtigste Merkmal identifiziert wurde, bei der Logistischen Regression nicht als signifikant eingestuft wird.
- `diag_1` , `diag_2` , `diag_3` : Die drei Diagnosemerkmale wurden bei der Feature Importance von XGBoost in Teilaufgabe R3c) noch als sehr wichtig eingestuft; bei der Logistischen Regression wurden diese Merkmale jedoch aufgrund deren sehr hohen Anzahlen an Ausprägungen weggelassen, um Instabilitäten und Overfitting zu verhindern.

Anschließend berechnen wir den AUC-Wert der Logistischen Regression auf den Testdaten.

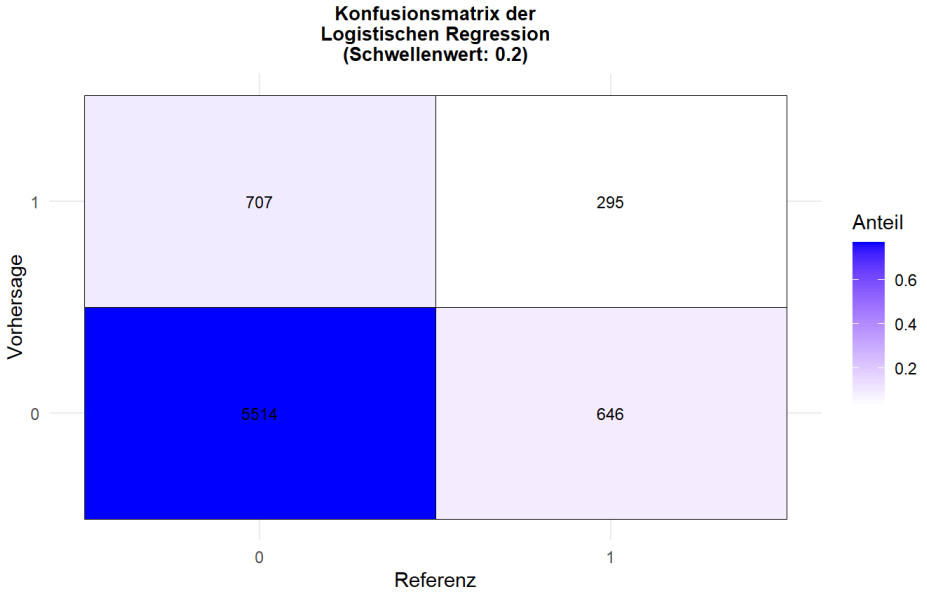
```
# Vorhersage der Wahrscheinlichkeiten für die Testdaten
test_bin_prob <- predict(glm1, newdata = test_bin, type = "response")

# Berechnung der AUC mit der angegebenen Funktion
auc_logreg_without <- calculate_auc(test_bin$TARGET, test_bin_prob, positive_class = 1)
```

```
## AUC: 0.6887
```

Die Wahl eines geeigneten Schwellenwerts ist bei einem unausgeglichenen Datensatz wie diesem besonders wichtig, da ein zu hoher Schwellenwert dazu führen kann, dass die Minderheitsklasse (ca. 13 %) zu selten erkannt wird, während ein zu niedriger Schwellenwert die Mehrheit falsch klassifizieren könnte. Der Schwellenwert beeinflusst die Balance zwischen Sensitivität (korrekte Erkennung der Minderheitsklasse) und Spezifität (korrekte Erkennung der Mehrheit). Ein bewusster Kompromiss zwischen diesen beiden Metriken ist erforderlich, abhängig vom Anwendungsfall und den Kosten von Fehlklassifikationen.

Als Nächstes erstellen wir eine Konfusionsmatrix, um die Vorhersagen des Modells mit den tatsächlichen Werten der Testdaten zu vergleichen, unter Verwendung des Schwellenwertes 0,2.

```
# Berechnung der vorhergesagten Klassenzugehörigkeiten (Schwellenwert = 0.2)
test_bin_pred <- ifelse(test_bin_prob > 0.2, 1, 0)

# Erstellen der Konfusionsmatrix
conf_matrix <- confusionMatrix(factor(test_bin_pred), factor(y_test))

# Visualisieren der Konfusionsmatrix
plot_confusion_matrix(conf_matrix, "Konfusionsmatrix der\nLogistischen Regression\n(Schwellenwert: 0.2)")
```



(Alternativer Ansatz über `nnet` wie später bei der ternärern Klassifikation:)

```
# Train logistic regression using nnet
logistic_model <- nnet::multinom(formula, data = train_bin, MaxNWts=1000000)
```

```
## # weights:  191 (190 variable)
## initial  value 23169.137657
## iter  10 value 14347.438195
## iter  20 value 13777.719291
## iter  30 value 13172.637854
## iter  40 value 12638.871275
## iter  50 value 12306.270894
## iter  60 value 12077.788395
## iter  70 value 11992.391944
## iter  80 value 11970.906152
## iter  90 value 11957.869795
## iter 100 value 11952.056036
## final  value 11952.056036
## stopped after 100 iterations
```

```
# Predict probabilities on the test set
probs <- predict(logistic_model, newdata = test_bin, type = "probs")

# Berechnung der AUC mit der angegebenen Funktion
auc_lr_nnet <- calculate_auc(y_test, probs, positive_class = 1)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
## AUC: 0.6892
```

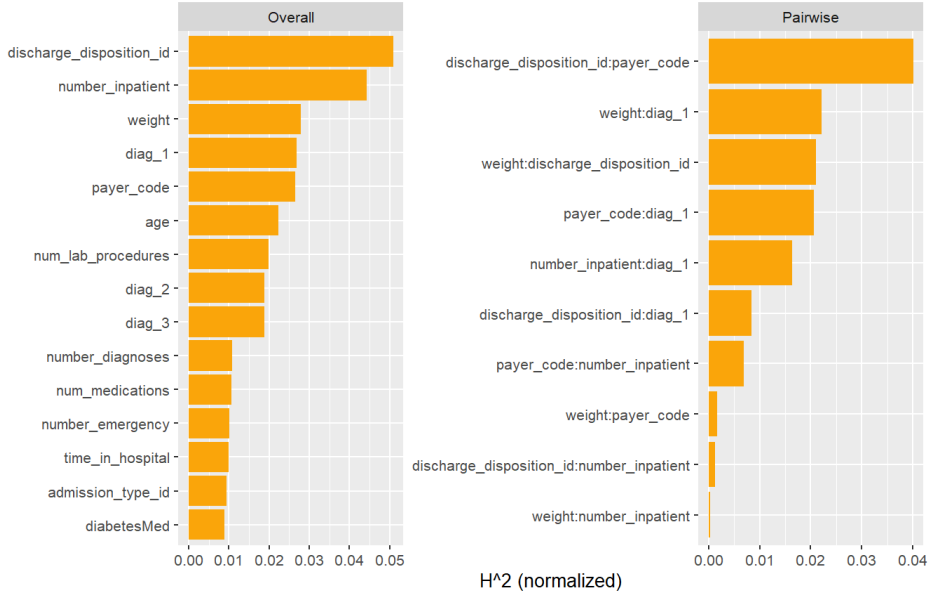
b) Logistische Regression mit Interaktionen: Verwenden Sie etwa das Paket `hstats` zur Identifikation der Wechselwirkungen zwischen den Prädiktoren des XGBoost-Modells aus Aufgabe R3. Erörtern Sie in wenigen Sätzen, wie diese Interaktionen bestimmt werden. Wählen Sie die drei signifikantesten Interaktionen aus, die weder `diag_1` , `diag_2` noch `diag_3` enthalten, und integrieren Sie diese in die Logistische Regression aus Teilaufgabe R4a), indem Sie die Regressionsformel entsprechend erweitern. Trainieren Sie dieses erweiterte Modell auf den Trainingsdaten und berechnen Sie dessen AUC-Wert auf den Testdaten und geben Sie das Ergebnis aus. Erstellen und visualisieren Sie eine Konfusionsmatrix, analog zu Teilaufgabe R4a).

Um die Interaktionen zwischen den Prädiktoren zu identifizieren, verwenden wir das `hstats` -Paket, das eine Methode zur Schätzung der Wechselwirkungen zwischen den Variablen in einem Modell bietet, zusammen mit dem XGBoost-Modell aus Aufgabe R3.

```
# Wende hstats zur Identifikation von Wechselwirkungen an
H <- hstats(gb_dt, X = data.matrix(X_train))
```

```
## 1-way calculations...
##   |                                                                      |
|   0%  |                                                                    |=
|   2%  |                                                                    |===
|   4%  |                                                                    |====
|   6%  |                                                                    |======
|   8%  |                                                                    |=======
|  10%  |                                                                    |========
|  12%  |                                                                    |==========
|  14%  |                                                                    |===========
|  16%  |                                                                    |=============
|  18%  |                                                                    |==============
|  20%  |                                                                    |================
|  22%  |                                                                    |=================
|  24%  |                                                                    |==================
|  26%  |                                                                    |====================
|  28%  |                                                                    |=====================
|  30%  |                                                                    |======================
|  32%  |                                                                    |========================
|  34%  |                                                                    |=========================
|  36%  |                                                                    |===========================
|  38%  |                                                                    |============================
|  40%  |                                                                    |==============================
|  42%  |                                                                    |================================
|  44%  |                                                                    |=================================
|  46%  |                                                                    |===================================
|  48%  |                                                                    |====================================
|  50%  |                                                                    |======================================
|  52%  |                                                                    |======================================
==                                  |  54%  |                                                                  |==
====================================                                  |  56%  |
|=========================================                              |  58%  |
|=========================================                              |  60%  |
|=========================================                              |  62%  |
|=============================================                          |  64%  |
|=============================================                          |  66%  |
|=================================================                      |  68%  |
|=================================================                      |  70%  |
|=====================================================                  |  72%  |
|=====================================================                  |  74%  |
|=========================================================              |  76%  |
|=========================================================              |  78%  |
|=============================================================          |  80%  |
|=============================================================          |  82%  |
|=================================================================      |  84%  |
|=================================================================      |  86%  |
|=====================================================================  |  88%  |
|=====================================================================  |  90%  |
|=========================================================================|  92%  |
|=========================================================================|  94%  |
|=========================================================================|  96%  |
|=========================================================================| 98%  |
|=========================================================================|100%
## 2-way calculations...
##   |                                                                      |
|   0%  |                                                                    |=======
|  10%  |                                                                    |==============
|  20%  |                                                                    |=====================
|  30%  |                                                                    |============================
|  40%  |                                                                    |===================================
|  50%  |                                                                    |===================================
======                              |  60%  |                                                                  |==
==============================================                        |  70%  |
|=============================================================          |  80%  |
|=================================================================      |  90%  |
|=========================================================================|100%
```

```
# Zeige die identifizierten Wechselwirkungen von hstats
plot(H) + theme_gray(base_size=10)
```



Die Interaktionen zwischen den Prädiktoren werden durch das H-Statistikverfahren aus der Arbeit von Friedman und Popescu ermittelt. Dieses Verfahren bewertet die Stärke von Wechselwirkungen durch Berechnung des Anteils der Vorhersagevarianz, der durch die Wechselwirkungen zwischen den Variablen erklärt wird. Für jede Variable und jedes Paar von Variablen wird der Beitrag ihrer Interaktion zur Vorhersage gemessen. Es werden sowohl Gesamtinteraktionen als auch paarweise Wechselwirkungen berechnet, wobei die Stärke der Interaktion durch den Anteil der erklärten Varianz dargestellt wird. Die größten Wechselwirkungen deuten darauf hin, welche Merkmale am stärksten miteinander in Beziehung stehen.

Als Nächstes identifizieren wir die drei signifikantesten Interaktionen, die weder `diag_1` , `diag_2` noch `diag_3` enthalten. Diese Interaktionen werden dann in die Formel der Logistischen Regression integriert.

```
# Extrahiere die drei wichtigsten Interaktionen ohne die Merkmale diag_1, diag_2 oder diag_3
top_interactions <- names(sort(h2_pairwise(H)$M[, 1], decreasing = TRUE))
top_interactions <- top_interactions[!grepl("diag_1|diag_2|diag_3", top_interactions)][1:3]
top_interactions <- gsub(":", " * ", top_interactions)
cat(paste("Top Interactions:", paste0(top_interactions, ", ")))
```

```
## Top Interactions: discharge_disposition_id * payer_code,  Top Interactions: weight * discharge_disposition_id,  Top Inter
actions: payer_code * number_inpatient,
```

```
# Kombiniere die ursprünglichen gültigen Merkmale und die formatierten Interaktionen
all_predictors <- c(valid_features, top_interactions)

# Erstelle die Formel unter Verwendung der gültigen Merkmale und der Top-Interaktionen
formula_with_interactions <- as.formula(paste("TARGET ~", paste(all_predictors, collapse = " + ")))
```

Nun trainieren wir das Logistische Regressionsmodell unter Verwendung der erweiterten Formel, die die identifizierten Interaktionen enthält.

```
# Trainiere das logistische Regressionsmodell mit Interaktionen
glm_with_interactions <- glm(formula_with_interactions, data = train_bin, family = binomial)

# Gib die Zusammenfassung des erweiterten Modells aus
#summary(glm_with_interactions)
```

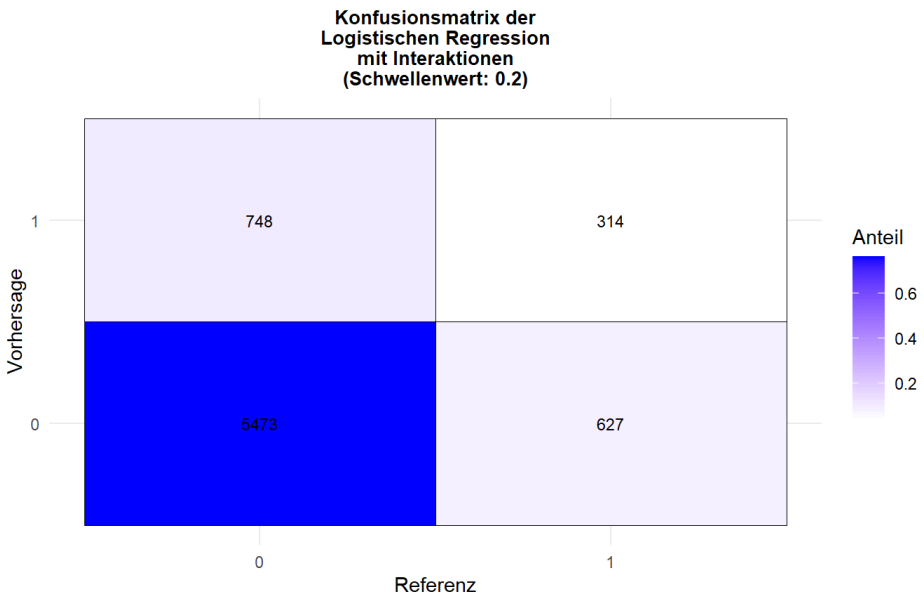
Im nächsten Schritt verwenden wir das trainierte Modell, um Wahrscheinlichkeiten für die Testdaten zu berechnen und den AUC-Wert zu bestimmen.

```
# Berechne die Vorhersagewahrscheinlichkeiten für die Testdaten
test_prob <- predict(glm_with_interactions, newdata = test_bin, type = "response")

# Berechne den AUC-Wert unter Verwendung der Funktion calculate_auc
auc_logreg_with <- calculate_auc(y_test, test_prob)
```

```
## AUC: 0.6896
```

Schließlich erstellen wir noch die zugehörige Konfusionsmatrix.

```
# Berechnung der vorhergesagten Klassenzugehörigkeiten (Schwellenwert = 0.2)
test_bin_pred_with <- ifelse(test_prob > 0.2, 1, 0)

# Erstellen der Konfusionsmatrix
conf_matrix_with <- confusionMatrix(factor(test_bin_pred_with), factor(y_test))

# Visualisieren der Konfusionsmatrix
plot_confusion_matrix(conf_matrix_with, "Konfusionsmatrix der\nLogistischen Regression\nmit Interaktionen\n(Schwellenwert:
        0.2)")
```



c) Modellvergleich und Diskussion: Vergleichen Sie die Ergebnisse der Logistischen Regressionen ohne und mit Interaktionen. Erötern Sie, ob die Hinzunahme der Interaktionen zu einer signifikanten Verbesserung der Modellgüte führt. Diskutieren Sie zudem kurz drei Ansätze, wie Interaktionen anders bzw. besser integriert werden können, um die Modellgüte zu steigern.

Die Ergebnisse der beiden Modelle zeigen eine lediglich sehr geringe Verbesserung der Modellgüte nach der Hinzunahme der Interaktionen: Die AUC der Logistischen Regression ohne Interaktionen beträgt 0.6887349, während das Modell mit Interaktionen eine AUC von 0.6896462 erzielt. Diese Veränderung ist marginal, was darauf hindeutet, dass die Berücksichtigung der Interaktionen keine signifikante Verbesserung der Modellleistung in diesem Fall bewirkt.

Ansatz 1: Gruppierung ähnlicher Merkmale

Ein Ansatz zur besseren Integration von Interaktionen ist die Gruppierung von ähnlichen Merkmalen, um ihre Wechselwirkungen explizit zu erfassen. Wenn mehrere Merkmale ähnliche Informationen liefern oder im selben Kontext relevant sind (z. B. `num_procedures` und `num_medications`), könnten diese gruppiert und ihre Kombinationen in das Modell integriert werden. So können potenziell wichtige Interaktionen aufgezeigt werden, die ansonsten unberücksichtigt bleiben würden.

Ansatz 2: Zusammenfassen von Wertebereichen

Ein weiterer Ansatz besteht darin, Wertebereiche von kontinuierlichen Merkmalen zu aggregieren, um Interaktionen besser sichtbar zu machen. Beispielsweise könnte `number_inpatient` in Kategorien unterteilt werden (z. B. 1-2, 3-5, 6+), was Interaktionen zwischen diesen Gruppen und anderen Variablen (wie `time_in_hospital`) genauer erfasst und modelliert. Diese Art der Kategorisierung hilft dem Modell, Muster in den Interaktionen zu erkennen, die bei kontinuierlichen Variablen möglicherweise zu komplex sind.

Ansatz 3: Berücksichtigung von Interaktionen höherer Ordnung

Zusätzlich könnte das Modell durch die Betrachtung von Interaktionen höherer Ordnung erweitert werden. Oft zeigen sich signifikante Wechselwirkungen nicht nur in einfachen Paaren von Merkmalen, sondern auch in der Kombination von mehreren Variablen. Beispielsweise könnten Wechselwirkungen zwischen `time_in_hospital` , `num_medications` und `number_emergency` entscheidend sein. Das Modell könnte von der Einführung solcher Interaktionen profitieren, da sie komplexe, aber wichtige Zusammenhänge aufdecken, die mit niedrigerer Ordnung nicht erfasst werden.

Aufgabe R5: Regularisierte Lineare Modelle und GAMs

In dieser Aufgabe werden Sie regularisierte Lineare Modelle (L1- und L2-Regularisierung) sowie Generalisierte Additive Modelle (GAMs) zur Modellierung einsetzen. Als Eingabedaten dienen die in Teilaufgabe R3a) aufbereiteten Trainings- und Testdatensätze. Verwenden Sie in allen Teilaufgaben die bereits in Teilaufgabe R4a) verwendeten Prädiktoren.

a) L1-Regularisierung (LASSO): Wenden Sie das LASSO-Verfahren (L1-Regularisierung) auf die Trainingsdaten an. Bestimmen Sie den optimalen Regularisierungsparameter lambda mittels Kreuzvalidierung, visualisieren Sie den Regularisierungsweg und geben Sie alle von Null verschiedenen Koeffizienten aus. Diskutieren Sie kurz drei mögliche Zusammenhänge zwischen den nicht-null Koeffizienten des LASSO-Modells und den in der EDA aus Aufgabe R2 gewonnenen Erkenntnissen. Berechnen Sie den AUC-Wert des Modells auf den Testdaten.

Zunächst erstellen wir eine für die weitere Verarbeitung passende Darstellung für die Prädiktoren der Trainings- und Testdaten. Als zugrunde liegende Formel wird diejenige aus Aufgabe R4a) wiederverwendet.

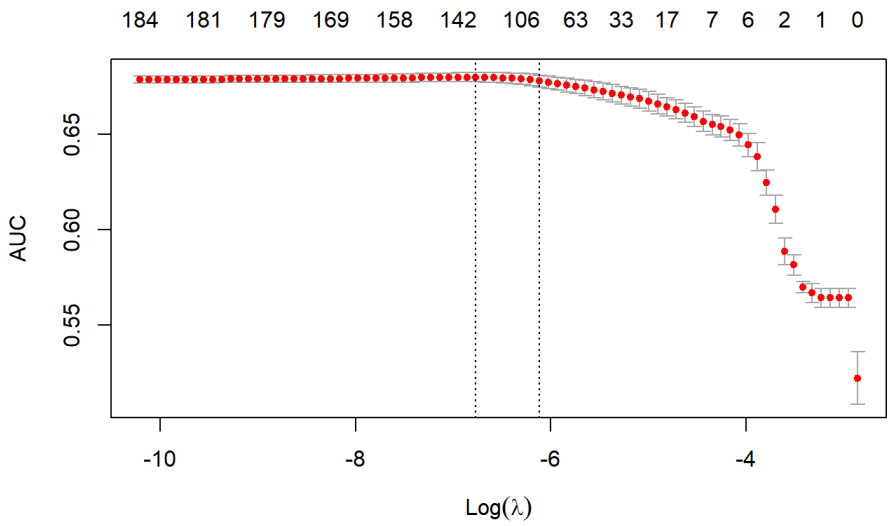
```
# Create a sparse matrix of the predictors (features) in the dataset
x_train <- sparse.model.matrix(formula, data = train_bin)[, -1]
x_test <- sparse.model.matrix(formula, data = test_bin)[, -1]
```

Nun wenden wir das LASSO-Verfahren (L1-Regularisierung) auf die Trainingsdaten an und bestimmen den optimalen Regularisierungsparameter lambda mittels Kreuzvalidierung. Wir verwenden den alpha-Wert von 1 für LASSO, und die Kreuzvalidierung erfolgt mit fünf Folds.

```
lasso_model <- cv.glmnet(x = x_train,
                         y = y_train,
                         alpha = 1,
                         family = "binomial",
                         nfolds = 5,
                         type.measure = "auc")
```

Wir visualisieren nun den Regularisierungsweg basierend auf den Ergebnissen der Kreuzvalidierung, um den Zusammenhang zwischen dem lambda-Wert und der Modellgüte zu sehen.

```
plot(lasso_model)
```



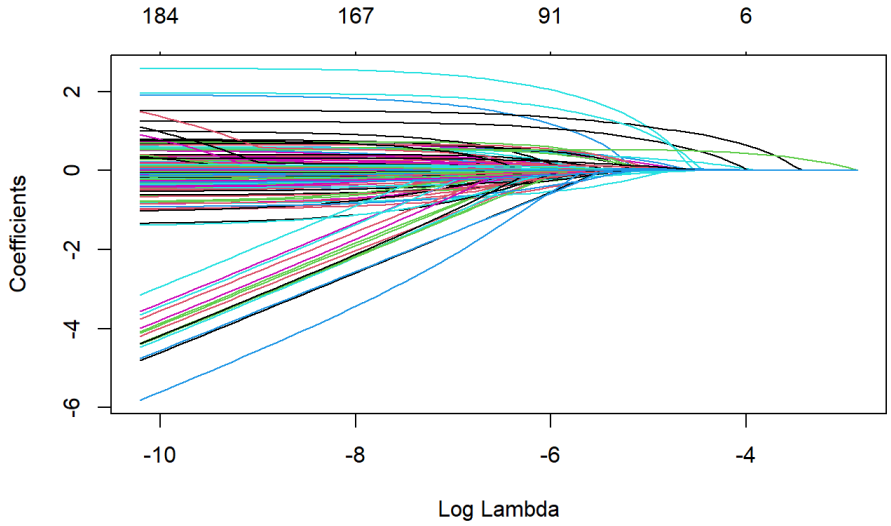
Nun extrahieren wir die Koeffizienten des Modells zum optimalen Regularisierungsparameter lambda (lambda.min), der durch die Kreuzvalidierung bestimmt wurde. Wir zeigen alle Koeffizienten an, die sich von Null unterscheiden.

```
# Extrahiere die Koeffizienten beim optimalen Lambda (lambda.min)
lasso_coef <- coef(lasso_model, s = "lambda.min")
optimal_lambda <- lasso_model$lambda.min
cat("Optimales Lambda:", optimal_lambda)
```

```
## Optimales Lambda: 0.001151469
```

Nicht gefordert, aber für das Verständnis der Regularisierung dennoch hilfreich: Der nachfolgende Plot visualitiert die Koeffizienten in Abhängigkeit von $\log(\lambda)$.

```
plot(lasso_model$glmnet.fit, xvar ='lambda')
```



Schließlich berechnen, speichern und printen wir die von Null verschiedenen Koeffizienten.

```
# In eine Matrix umwandeln und nicht-null Koeffizienten filtern
lasso_coef_matrix <- as.matrix(lasso_coef)

# Nicht-null Koeffizienten und deren Namen finden
non_zero_indices <- which(lasso_coef_matrix != 0)
non_zero_coefs <- lasso_coef_matrix[non_zero_indices]
non_zero_names <- rownames(lasso_coef_matrix)[non_zero_indices]

# Erstellen eines Dataframes mit zwei Spalten: "feature" und "coefficient"
non_zero_df <- data.frame(feature = non_zero_names, coefficient = non_zero_coefs, stringsAsFactors = FALSE)
non_zero_df
```

feature <chr>	coefficient <dbl>
(Intercept)	-2.9005310000
raceAfricanAmerican	0.0006113609
raceCaucasian	0.0179412466
raceHispanic	-0.2215149376
raceOther	-0.2649741035
age[10-20)	-0.0147021423
age[20-30)	-0.0268227997
age[30-40)	-0.1292924000
age[50-60)	-0.1287445415
age[60-70)	0.0574697270

1-10 of 136 rows Previous **1** 2 3 4 5 6 … 14 Next

Mit Blick auf die Wiedereinweisungsraten aus Teilaufgabe R2c) lassen sich u. a. folgende wichtige Zusammenhänge identifizieren:

- `discharge_disposition_id` : Ein sehr hoher Wert von `discharge_disposition_id15` steht im Einklang mit der hohen Wiedereinweisungsrate, die in der EDA für diese Ausprägung erkannt wurde.
- `number_inpatient` : Ein vergleichsweise hoher Wert von `number_inpatient` passt ins Bild, dass ein höherer Wert dieses Merkmals tendenziell mit einer höheren Wiedereinweisungsrate einhergeht.
- `gender` : Im LASSO-Modell weist dieses Merkmal einen Koeffizienten von Null auf. Dies spiegelt die Erkenntnis wider, dass die Wiedereinweisungsraten bei Männern und Frauen nahezu identisch sind.

Jetzt berechnen wir die Vorhersagen des LASSO-Modells auf dem Testdatensatz und werten die Modellleistung mit dem AUC-Wert aus.

```
# Bestimme die Prädiktionen des LASSO-Modells auf dem Testdatensatz
lasso_predictions <- predict(lasso_model, newx = x_test,
                             type = "response", s = "lambda.min")

# Berechne den AUC-Wert für das LASSO-Modell
auc_lasso <- calculate_auc(true_labels = y_test,
                           predicted_probs = lasso_predictions)
```

```
## Setting levels: control = 0, case = 1
```

```
## Warning in roc.default(true_labels, predicted_probs): Deprecated use a matrix
## as predictor. Unexpected results may be produced, please pass a numeric vector.
```

```
## Setting direction: controls < cases
```

```
## AUC: 0.6884
```

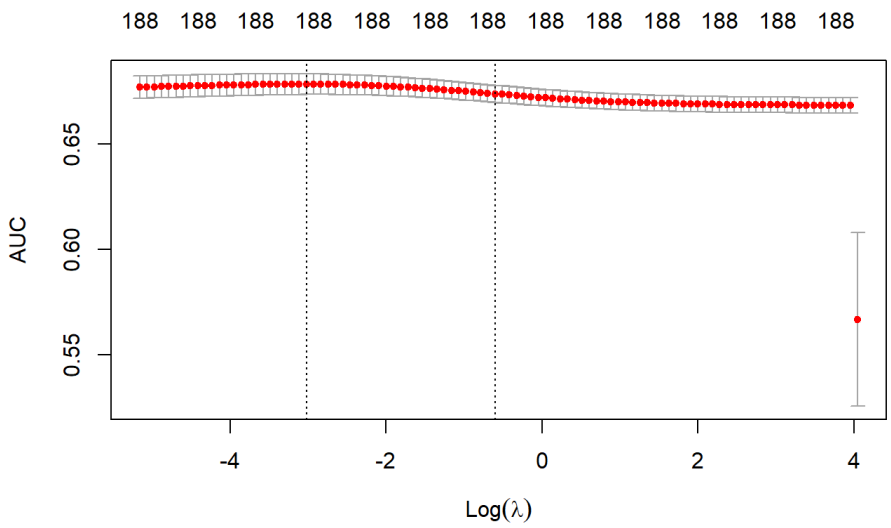
b) L2-Regularisierung (Ridge Regression): Wenden Sie das Ridge-Regression-Verfahren (L2-Regularisierung) auf die Trainingsdaten an. Bestimmen Sie den optimalen Regularisierungsparameter lambda mittels Kreuzvalidierung und visualisieren Sie den Regularisierungsweg. Berechnen Sie den AUC-Wert des Modells auf den Testdaten.

Wir verwenden die in Teilaufgabe R6a) erzeugten Versionen der Trainings- und Testdaten und wenden die Ridge Regression (L2-Regularisierung) auf die Trainingsdaten an. Dabei bestimmen wir bestimmen den optimalen Regularisierungsparameter lambda mithilfe von Kreuzvalidierung (alpha = 0 für Ridge Regression).

```
ridge_model <- cv.glmnet(x = x_train,
                         y = y_train,
                         alpha = 0,
                         family = "binomial",
                         nfolds = 5,
                         type.measure = "auc")
```

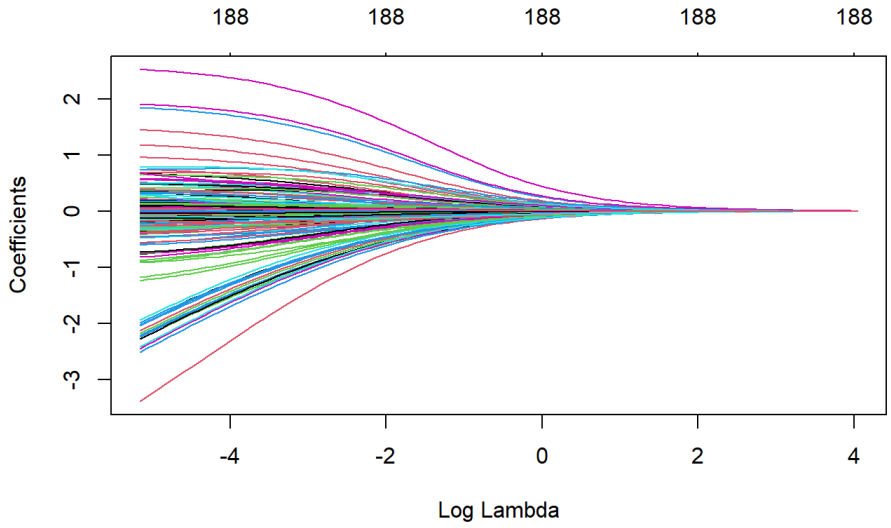
Zur Visualisierung der Modellgüte in Abhängigkeit von verschiedenen Regularisierungsparametern plotten wir die Kreuzvalidierungsergebnisse der Ridge Regression.

```
plot(ridge_model)
```



Nicht gefordert, aber für das Verständnis der Regularisierung dennoch hilfreich: Der nachfolgende Plot visualitiert die Koeffizienten in Abhängigkeit von $\log(\lambda)$.

```
plot(ridge_model$glmnet.fit, xvar ='lambda')
```



Nun extrahieren wir die Koeffizienten des Ridge-Modells für den optimalen Wert von lambda (lambda.min), der durch die Kreuzvalidierung identifiziert wurde, und geben die Koeffizienten des Ridge-Regression-Modells aus.

```
ridge_coef <- coef(ridge_model, s = "lambda.min")
optimal_lambda <- ridge_model$lambda.min
cat("Optimales Lambda:\n", optimal_lambda, "\n")
```

```
## Optimales Lambda:
##  0.04869844
```

Abschließend berechnen wir die Vorhersagen des Ridge-Regression-Modells für den Testdatensatz. Wir bewerten die Modellleistung durch den AUC-Wert.

```
# Bestimme die Prädiktionen des Ridge-Regression-Modells auf dem Testdatensatz
ridge_predictions <- predict(ridge_model, newx = x_test,
                             type = "response", s = "lambda.min")

# Berechne den AUC-Wert für die Ridge Regression.
auc_ridge <- calculate_auc(true_labels = y_test,
                           predicted_probs = ridge_predictions)
```

```
## AUC: 0.6865
```

c) Generalisiertes Additives Modell (GAM): Erstellen Sie ein Generalisiertes Additives Modell (GAM) auf den Trainingsdaten, beispielsweise mit dem Paket `mgcv`. Das GAM sollte so konzipiert sein, dass der AUC-Wert dieses Modells auf den Testdaten höher ist als der entsprechende Wert der Logistischen Regression aus Aufgabe R4a). Erläutern Sie, an Hand welcher Plots und Kurvenverläufe aus Aufgabe R2 gut geeignete Merkmale für die Erstellung eines passenden GAMs erkannt werden könnten, die in den bisherigen

Aufgabenteilen nicht betrachtete Effekte berücksichtigen. Benennen Sie die entsprechenden Merkmalskandidaten und visualisieren Sie die geschätzten Effekte von zwei dieser Merkmale, die als besonders vielversprechend für das GAM identifiziert wurden. Berechnen Sie den AUC-Wert des Modells auf den Testdaten.

Wir identifizieren zunächst die numerischen Prädiktoren aus den Trainingsdaten und erstellen eine Formel, in der diese Prädiktoren mit `s()` für glatte Terme versehen sind. Dies erlaubt es, nichtlineare Zusammenhänge besser zu modellieren. Nicht-numerische Prädiktoren werden der Formel unverändert hinzugefügt.

```
# Identifiziere numerische Prädiktoren (Spalten vom Typ integer)
numeric_features <- colnames(train_bin)[sapply(train_bin, is.integer)]

# Identifiziere nicht-numerische Prädiktoren (ohne die Zielvariable TARGET)
non_numeric_features <- setdiff(valid_features, numeric_features)

# Erstelle die Formel für numerische Prädiktoren mit glatten Termen (s())
numeric_formula <- paste("s(", numeric_features, ", k=4)", collapse = " + ")

# Kombiniere numerische und nicht-numerische Prädiktoren
formula_with_smooths <- as.formula(paste("TARGET ~", paste(c(numeric_formula, non_numeric_features), collapse = " + ")))

print(formula_with_smooths)
```

```
## TARGET ~ s(time_in_hospital, k = 4) + s(num_lab_procedures, k = 4) +
##     s(num_procedures, k = 4) + s(num_medications, k = 4) + s(number_outpatient,
##     k = 4) + s(number_emergency, k = 4) + s(number_inpatient,
##     k = 4) + s(number_diagnoses, k = 4) + race + gender + age +
##     weight + admission_type_id + discharge_disposition_id + admission_source_id +
##     payer_code + medical_specialty + max_glu_serum + A1Cresult +
##     metformin + repaglinide + nateglinide + chlorpropamide +
##     glimepiride + glipizide + glyburide + tolbutamide + pioglitazone +
##     rosiglitazone + acarbose + miglitol + tolazamide + insulin +
##     glyburide.metformin + change + diabetesMed + group_diag_1 +
##     group_diag_2 + group_diag_3
```

Nun trainieren wir das Generalisierte Additive Modell (GAM) auf den Trainingsdaten. Dieses Modell nutzt glatte Terme für die numerischen Prädiktoren und berücksichtigt sowohl lineare als auch nichtlineare Beziehungen.

```
# Trainiere ein GAM-Modell mit glatten Termen für nichtlineare Zusammenhänge
gam_model <- gam(
  formula_with_smooths,
  data = train_bin,
  family = binomial(link = "logit")
)

# Zeige die Zusammenfassung des GAM-Modells
summary(gam_model)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## TARGET ~ s(time_in_hospital, k = 4) + s(num_lab_procedures, k = 4) +
##     s(num_procedures, k = 4) + s(num_medications, k = 4) + s(number_outpatient,
##     k = 4) + s(number_emergency, k = 4) + s(number_inpatient,
##     k = 4) + s(number_diagnoses, k = 4) + race + gender + age +
##     weight + admission_type_id + discharge_disposition_id + admission_source_id +
##     payer_code + medical_specialty + max_glu_serum + A1Cresult +
##     metformin + repaglinide + nateglinide + chlorpropamide +
##     glimepiride + glipizide + glyburide + tolbutamide + pioglitazone +
##     rosiglitazone + acarbose + miglitol + tolazamide + insulin +
##     glyburide.metformin + change + diabetesMed + group_diag_1 +
##     group_diag_2 + group_diag_3
##
## Parametric coefficients:
##                                                  Estimate Std. Error
## (Intercept)                                    -2.235e+01  8.293e+03
## raceAfricanAmerican                             2.269e-01  1.190e-01
## raceAsian                                       2.159e-01  2.195e-01
## raceCaucasian                                   2.313e-01  1.128e-01
## raceHispanic                                   -1.184e-01  1.730e-01
## raceOther                                      -1.623e-01  1.861e-01
## genderMale                                     -7.776e-03  3.485e-02
## age[10-20)                                      4.353e-01  7.929e-01
## age[20-30)                                      3.969e-01  7.769e-01
## age[30-40)                                      3.001e-01  7.698e-01
## age[40-50)                                      4.915e-01  7.660e-01
## age[50-60)                                      3.521e-01  7.656e-01
## age[60-70)                                      5.837e-01  7.654e-01
## age[70-80)                                      6.853e-01  7.657e-01
## age[80-90)                                      5.792e-01  7.663e-01
## age[90-100)                                     3.417e-01  7.724e-01
## weight[0-25)                                    1.942e+00  6.310e-01
## weight[25-50)                                  -3.272e-02  4.721e-01
## weight[50-75)                                   2.154e-01  1.638e-01
## weight[75-100)                                  7.020e-01  1.283e-01
## weight[100-125)                                 7.778e-01  1.805e-01
## weight[125-150)                                 2.539e-01  4.159e-01
## weight[150-175)                                -3.869e+01  2.023e+07
## admission_type_id2                              5.833e-02  6.880e-02
## admission_type_id3                             -1.401e-01  7.982e-02
## admission_type_id5                              6.306e-02  1.197e-01
## admission_type_id6                              4.008e-01  9.820e-02
## admission_type_id7                             -3.873e+01  1.861e+07
## admission_type_id8                             -3.895e-01  3.045e-01
## discharge_disposition_id2                       5.932e-01  9.838e-02
## discharge_disposition_id3                       5.452e-01  5.539e-02
## discharge_disposition_id4                       3.453e-01  1.866e-01
## discharge_disposition_id5                       1.233e+00  1.120e-01
## discharge_disposition_id6                       2.690e-01  5.629e-02
## discharge_disposition_id7                       1.909e-01  2.254e-01
## discharge_disposition_id8                       4.964e-01  4.576e-01
## discharge_disposition_id15                      2.553e+00  5.688e-01
## discharge_disposition_id18                      2.092e-01  9.370e-02
## discharge_disposition_id22                      1.528e+00  8.913e-02
## discharge_disposition_id23                     -1.020e+00  4.234e-01
## discharge_disposition_id24                     -3.866e+01  1.937e+07
## discharge_disposition_id25                     -5.277e-01  2.122e-01
## discharge_disposition_id28                      1.981e+00  3.093e-01
## admission_source_id2                           -4.313e-01  1.576e-01
## admission_source_id3                            6.552e-01  2.880e-01
## admission_source_id4                           -4.555e-01  1.005e-01
## admission_source_id5                           -6.716e-01  2.018e-01
## admission_source_id6                           -2.146e-01  1.349e-01
## admission_source_id7                           -4.288e-02  7.034e-02
## admission_source_id8                            2.872e-01  1.136e+00
## admission_source_id9                            1.081e-01  4.529e-01
## admission_source_id17                          -4.938e-01  1.090e-01
## admission_source_id20                           6.932e-01  3.346e-01
## payer_codeBC                                   -4.519e-01  9.606e-02
## payer_codeCH                                   -8.764e-01  6.011e-01
## payer_codeCM                                   -5.371e-01  1.344e-01
## payer_codeCP                                   -3.900e-01  1.185e-01
## payer_codeDM                                   -3.997e-01  2.572e-01
## payer_codeHM                                   -2.334e-01  8.421e-02
## payer_codeMC                                   -2.708e-01  4.723e-02
## payer_codeMD                                   -1.512e-01  1.066e-01
## payer_codeMP                                   -2.982e-01  1.053e+00
## payer_codeOG                                   -1.481e-01  1.872e-01
## payer_codeOT                                   -8.269e-01  7.495e-01
## payer_codePO                                   -3.610e-01  2.363e-01
## payer_codeSI                                    2.201e-02  7.640e-01
```

```
## payer_codeSP                                          -1.353e-01  9.095e-02
## payer_codeUN                                          -3.011e-01  1.171e-01
## payer_codeWC                                          -9.977e-01  6.057e-01
## medical_specialtyAnesthesiology-Pediatric              3.313e-01  1.102e+00
## medical_specialtyCardiology                           -2.088e-01  8.385e-02
## medical_specialtyEmergency/Trauma                     -1.070e-01  8.715e-02
## medical_specialtyEndocrinology                        -1.229e-01  4.863e-01
## medical_specialtyFamily/GeneralPractice                6.150e-02  6.886e-02
## medical_specialtyGastroenterology                      3.981e-01  2.150e-01
## medical_specialtyGynecology                           -3.838e+01  1.151e+07
## medical_specialtyHematology                           -2.208e-01  7.884e-01
## medical_specialtyHematology/Oncology                   5.767e-01  3.585e-01
## medical_specialtyHospitalist                          -3.102e-01  1.059e+00
## medical_specialtyInfectiousDiseases                    9.753e-01  6.080e-01
## medical_specialtyInternalMedicine                     -8.709e-02  5.273e-02
## medical_specialtyNephrology                            5.573e-01  1.486e-01
## medical_specialtyNeurology                            -3.607e-01  3.865e-01
## medical_specialtyObsterics&Gynecology-GynecologicOnco -3.903e+01  1.937e+07
## medical_specialtyObstetrics                           -3.828e+01  2.373e+07
## medical_specialtyObstetricsandGynecology              -6.496e-01  2.546e-01
## medical_specialtyOncology                              6.326e-01  2.394e-01
## medical_specialtyOphthalmology                        -5.134e-01  1.043e+00
## medical_specialtyOrthopedics                          -6.811e-02  1.438e-01
## medical_specialtyOrthopedics-Reconstructive           -8.619e-01  1.821e-01
## medical_specialtyOsteopath                             4.473e-01  6.771e-01
## medical_specialtyOtolaryngology                       -3.853e+01  8.455e+06
## medical_specialtyPediatrics                           -7.671e-01  4.380e-01
## medical_specialtyPediatrics-CriticalCare              -3.828e+01  1.118e+07
## medical_specialtyPediatrics-Endocrinology             -3.809e+01  7.074e+06
## medical_specialtyPhysicalMedicineandRehabilitation     5.850e-01  2.592e-01
## medical_specialtyPodiatry                              4.608e-04  6.387e-01
## medical_specialtyPsychiatry                            4.203e-01  1.687e-01
## medical_specialtyPsychology                           -1.029e+00  1.027e+00
## medical_specialtyPulmonology                           6.231e-02  1.671e-01
## medical_specialtyRadiologist                          -3.144e-01  1.835e-01
## medical_specialtyRadiology                             1.091e-01  6.445e-01
## medical_specialtyRheumatology                         -3.866e+01  3.355e+07
## medical_specialtySurgeon                              -4.416e-01  7.634e-01
## medical_specialtySurgery-Cardiovascular               -1.314e+00  7.324e-01
## medical_specialtySurgery-Cardiovascular/Thoracic      -6.553e-01  2.409e-01
## medical_specialtySurgery-General                      -1.380e-01  1.031e-01
## medical_specialtySurgery-Neuro                        -9.678e-01  3.100e-01
## medical_specialtySurgery-Plastic                       5.542e-02  8.012e-01
## medical_specialtySurgery-Thoracic                     -3.943e-01  5.397e-01
## medical_specialtySurgery-Vascular                      3.899e-01  2.131e-01
## medical_specialtySurgicalSpecialty                    -3.828e+01  1.937e+07
## medical_specialtyUrology                              -1.672e-02  2.017e-01
## max_glu_serum>300                                      1.491e-03  2.207e-01
## max_glu_serumNone                                     -8.366e-02  1.623e-01
## max_glu_serumNorm                                      1.616e-01  1.803e-01
## A1Cresult>8                                           -1.135e-02  1.039e-01
## A1CresultNone                                          7.273e-02  8.732e-02
## A1CresultNorm                                         -3.785e-03  1.111e-01
## metforminNo                                            1.538e-02  2.050e-01
## metforminSteady                                       -1.126e-01  2.049e-01
## metforminUp                                           -3.557e-01  2.665e-01
## repaglinideNo                                          1.940e+01  8.293e+03
## repaglinideSteady                                      1.956e+01  8.293e+03
## repaglinideUp                                          1.998e+01  8.293e+03
## nateglinideSteady                                     -8.436e-02  2.198e-01
## nateglinideUp                                         -3.985e+01  2.122e+07
## chlorpropamideSteady                                  -1.406e+00  1.038e+00
## glimepirideNo                                         -1.518e-01  3.595e-01
## glimepirideSteady                                     -2.581e-01  3.653e-01
## glimepirideUp                                         -1.732e-01  4.504e-01
## glipizideNo                                           -3.362e-01  2.114e-01
## glipizideSteady                                       -3.211e-01  2.120e-01
## glipizideUp                                           -1.669e-01  2.661e-01
## glyburideNo                                            1.250e-01  2.198e-01
## glyburideSteady                                        1.318e-01  2.199e-01
## glyburideUp                                           -3.865e-02  2.780e-01
## tolbutamideSteady                                     -7.555e-01  1.060e+00
## pioglitazoneNo                                        -4.713e-02  4.670e-01
## pioglitazoneSteady                                    -1.401e-01  4.702e-01
## pioglitazoneUp                                         2.362e-01  5.520e-01
## rosiglitazoneNo                                        7.068e-01  6.237e-01
## rosiglitazoneSteady                                    7.478e-01  6.266e-01
## rosiglitazoneUp                                        7.525e-01  7.172e-01
## acarboseSteady                                        -5.480e-02  3.515e-01
## acarboseUp                                            -3.859e+01  4.745e+07
## miglitolSteady                                        -3.906e+01  3.355e+07
## tolazamideSteady                                      -3.891e+01  2.023e+07
## insulinNo                                             -1.447e-01  9.259e-02
## insulinSteady                                         -1.453e-01  7.079e-02
## insulinUp                                             -7.728e-02  7.268e-02
## glyburide.metforminSteady                              8.694e-02  2.080e-01
```

```
## changeNo                                  4.757e-02  6.466e-02
## diabetesMedYes                            3.029e-01  6.301e-02
## group_diag_1Diabetes                      2.389e-02  7.463e-02
## group_diag_1Digestive                    -1.652e-01  7.067e-02
## group_diag_1Genitourinary                -1.291e-01  8.578e-02
## group_diag_1Injury                       -2.270e-01  7.468e-02
## group_diag_1Musculoskeletal              -2.652e-01  9.593e-02
## group_diag_1n.a.                          6.987e-01  8.955e-01
## group_diag_1Neoplasms                     2.282e-02  8.728e-02
## group_diag_1Other                        -2.216e-01  5.769e-02
## group_diag_1Respiratory                  -3.050e-01  6.188e-02
## group_diag_2Diabetes                      1.550e-01  6.060e-02
## group_diag_2Digestive                    -1.360e-01  1.014e-01
## group_diag_2Genitourinary                -1.304e-01  7.060e-02
## group_diag_2Injury                       -1.273e-01  1.110e-01
## group_diag_2Musculoskeletal              -1.472e-01  1.362e-01
## group_diag_2n.a.                         -9.619e-02  4.015e-01
## group_diag_2Neoplasms                     1.373e-01  9.597e-02
## group_diag_2Other                         8.806e-03  4.847e-02
## group_diag_2Respiratory                  -5.022e-02  6.473e-02
## group_diag_3Diabetes                      8.096e-02  5.315e-02
## group_diag_3Digestive                     1.166e-01  9.691e-02
## group_diag_3Genitourinary                 5.116e-02  7.535e-02
## group_diag_3Injury                       -3.486e-02  1.236e-01
## group_diag_3Musculoskeletal               2.374e-02  1.243e-01
## group_diag_3n.a.                         -1.132e-02  1.843e-01
## group_diag_3Neoplasms                     4.385e-02  9.967e-02
## group_diag_3Other                        -2.900e-02  4.681e-02
## group_diag_3Respiratory                   3.874e-02  7.223e-02
##                                          z value Pr(>|z|)
## (Intercept)                               -0.003 0.997849
## raceAfricanAmerican                        1.907 0.056465 .
## raceAsian                                  0.984 0.325291
## raceCaucasian                              2.050 0.040355 *
## raceHispanic                              -0.684 0.493682
## raceOther                                 -0.872 0.383095
## genderMale                                -0.223 0.823415
## age[10-20)                                 0.549 0.583011
## age[20-30)                                 0.511 0.609494
## age[30-40)                                 0.390 0.696630
## age[40-50)                                 0.642 0.521158
## age[50-60)                                 0.460 0.645628
## age[60-70)                                 0.763 0.445743
## age[70-80)                                 0.895 0.370773
## age[80-90)                                 0.756 0.449749
## age[90-100)                                0.442 0.658208
## weight[0-25)                               3.078 0.002083 **
## weight[25-50)                             -0.069 0.944754
## weight[50-75)                              1.315 0.188385
## weight[75-100)                             5.471 4.48e-08 ***
## weight[100-125)                            4.310 1.63e-05 ***
## weight[125-150)                            0.610 0.541555
## weight[150-175)                            0.000 0.999998
## admission_type_id2                         0.848 0.396501
## admission_type_id3                        -1.755 0.079258 .
## admission_type_id5                         0.527 0.598232
## admission_type_id6                         4.081 4.48e-05 ***
## admission_type_id7                         0.000 0.999998
## admission_type_id8                        -1.279 0.200901
## discharge_disposition_id2                  6.030 1.64e-09 ***
## discharge_disposition_id3                  9.842  < 2e-16 ***
## discharge_disposition_id4                  1.850 0.064251 .
## discharge_disposition_id5                 11.004  < 2e-16 ***
## discharge_disposition_id6                  4.779 1.76e-06 ***
## discharge_disposition_id7                  0.847 0.397217
## discharge_disposition_id8                  1.085 0.277945
## discharge_disposition_id15                 4.489 7.15e-06 ***
## discharge_disposition_id18                 2.233 0.025539 *
## discharge_disposition_id22                17.141  < 2e-16 ***
## discharge_disposition_id23                -2.410 0.015971 *
## discharge_disposition_id24                 0.000 0.999998
## discharge_disposition_id25                -2.486 0.012901 *
## discharge_disposition_id28                 6.404 1.52e-10 ***
## admission_source_id2                      -2.737 0.006200 **
## admission_source_id3                       2.275 0.022883 *
## admission_source_id4                      -4.534 5.79e-06 ***
## admission_source_id5                      -3.329 0.000872 ***
## admission_source_id6                      -1.591 0.111555
## admission_source_id7                      -0.610 0.542171
## admission_source_id8                       0.253 0.800373
## admission_source_id9                       0.239 0.811399
## admission_source_id17                     -4.531 5.87e-06 ***
## admission_source_id20                      2.072 0.038278 *
## payer_codeBC                              -4.705 2.54e-06 ***
## payer_codeCH                              -1.458 0.144895
## payer_codeCM                              -3.996 6.44e-05 ***
```

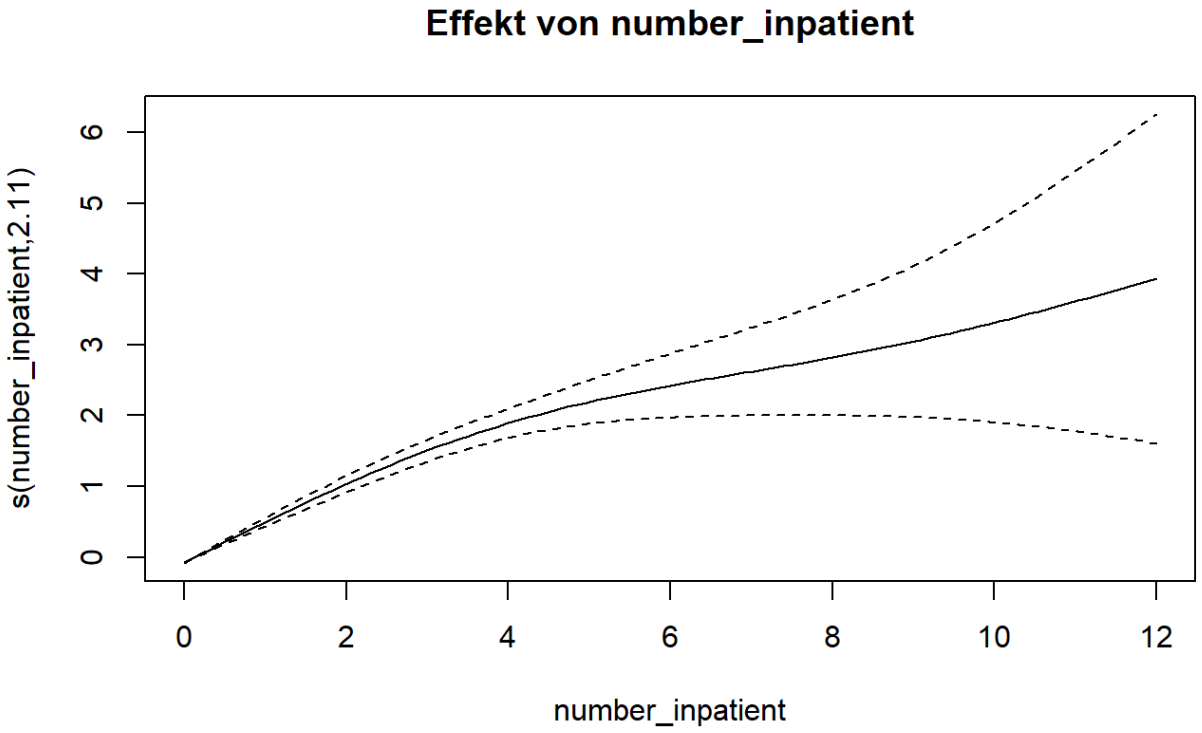
```
## payer_codeCP                                      -3.291 0.001000 ***
## payer_codeDM                                      -1.554 0.120209
## payer_codeHM                                      -2.772 0.005570 **
## payer_codeMC                                      -5.734 9.83e-09 ***
## payer_codeMD                                      -1.418 0.156107
## payer_codeMP                                      -0.283 0.776948
## payer_codeOG                                      -0.791 0.429053
## payer_codeOT                                      -1.103 0.269935
## payer_codePO                                      -1.528 0.126570
## payer_codeSI                                       0.029 0.977015
## payer_codeSP                                      -1.488 0.136755
## payer_codeUN                                      -2.572 0.010119 *
## payer_codeWC                                      -1.647 0.099533 .
## medical_specialtyAnesthesiology-Pediatric          0.301 0.763703
## medical_specialtyCardiology                       -2.490 0.012770 *
## medical_specialtyEmergency/Trauma                 -1.228 0.219489
## medical_specialtyEndocrinology                    -0.253 0.800556
## medical_specialtyFamily/GeneralPractice            0.893 0.371826
## medical_specialtyGastroenterology                  1.851 0.064105 .
## medical_specialtyGynecology                        0.000 0.999997
## medical_specialtyHematology                       -0.280 0.779452
## medical_specialtyHematology/Oncology               1.609 0.107667
## medical_specialtyHospitalist                      -0.293 0.769683
## medical_specialtyInfectiousDiseases                1.604 0.108649
## medical_specialtyInternalMedicine                 -1.652 0.098579 .
## medical_specialtyNephrology                        3.751 0.000176 ***
## medical_specialtyNeurology                        -0.933 0.350693
## medical_specialtyObsterics&Gynecology-GynecologicOnco  0.000 0.999998
## medical_specialtyObstetrics                        0.000 0.999999
## medical_specialtyObstetricsandGynecology          -2.552 0.010724 *
## medical_specialtyOncology                          2.643 0.008214 **
## medical_specialtyOphthalmology                    -0.492 0.622465
## medical_specialtyOrthopedics                      -0.474 0.635797
## medical_specialtyOrthopedics-Reconstructive       -4.734 2.21e-06 ***
## medical_specialtyOsteopath                         0.661 0.508843
## medical_specialtyOtolaryngology                    0.000 0.999996
## medical_specialtyPediatrics                       -1.751 0.079906 .
## medical_specialtyPediatrics-CriticalCare           0.000 0.999997
## medical_specialtyPediatrics-Endocrinology          0.000 0.999996
## medical_specialtyPhysicalMedicineandRehabilitation  2.257 0.024015 *
## medical_specialtyPodiatry                          0.001 0.999424
## medical_specialtyPsychiatry                        2.491 0.012726 *
## medical_specialtyPsychology                       -1.002 0.316347
## medical_specialtyPulmonology                       0.373 0.709225
## medical_specialtyRadiologist                      -1.713 0.086657 .
## medical_specialtyRadiology                         0.169 0.865626
## medical_specialtyRheumatology                      0.000 0.999999
## medical_specialtySurgeon                          -0.578 0.562959
## medical_specialtySurgery-Cardiovascular           -1.794 0.072772 .
## medical_specialtySurgery-Cardiovascular/Thoracic  -2.720 0.006528 **
## medical_specialtySurgery-General                  -1.339 0.180488
## medical_specialtySurgery-Neuro                    -3.122 0.001796 **
## medical_specialtySurgery-Plastic                   0.069 0.944853
## medical_specialtySurgery-Thoracic                 -0.731 0.464962
## medical_specialtySurgery-Vascular                  1.829 0.067349 .
## medical_specialtySurgicalSpecialty                 0.000 0.999998
## medical_specialtyUrology                          -0.083 0.933963
## max_glu_serum>300                                  0.007 0.994611
## max_glu_serumNone                                 -0.515 0.606240
## max_glu_serumNorm                                  0.896 0.370211
## A1Cresult>8                                       -0.109 0.912994
## A1CresultNone                                      0.833 0.404926
## A1CresultNorm                                     -0.034 0.972814
## metforminNo                                        0.075 0.940203
## metforminSteady                                   -0.549 0.582743
## metforminUp                                       -1.335 0.182033
## repaglinideNo                                      0.002 0.998134
## repaglinideSteady                                  0.002 0.998118
## repaglinideUp                                      0.002 0.998078
## nateglinideSteady                                 -0.384 0.701184
## nateglinideUp                                      0.000 0.999999
## chlorpropamideSteady                              -1.354 0.175714
## glimepirideNo                                     -0.422 0.672830
## glimepirideSteady                                 -0.707 0.479767
## glimepirideUp                                     -0.384 0.700614
## glipizideNo                                       -1.590 0.111730
## glipizideSteady                                   -1.514 0.129974
## glipizideUp                                       -0.627 0.530451
## glyburideNo                                        0.569 0.569541
## glyburideSteady                                    0.600 0.548776
## glyburideUp                                       -0.139 0.889439
## tolbutamideSteady                                 -0.713 0.476002
## pioglitazoneNo                                    -0.101 0.919629
## pioglitazoneSteady                                -0.298 0.765742
## pioglitazoneUp                                     0.428 0.668800
## rosiglitazoneNo                                    1.133 0.257164
```

```
## rosiglitazoneSteady                                  1.193 0.232708
## rosiglitazoneUp                                      1.049 0.294047
## acarboseSteady                                      -0.156 0.876113
## acarboseUp                                           0.000 0.999999
## miglitolSteady                                       0.000 0.999999
## tolazamideSteady                                     0.000 0.999998
## insulinNo                                           -1.562 0.118199
## insulinSteady                                       -2.053 0.040061 *
## insulinUp                                           -1.063 0.287670
## glyburide.metforminSteady                            0.418 0.676036
## changeNo                                             0.736 0.461896
## diabetesMedYes                                       4.808 1.53e-06 ***
## group_diag_1Diabetes                                 0.320 0.748874
## group_diag_1Digestive                               -2.338 0.019388 *
## group_diag_1Genitourinary                           -1.505 0.132241
## group_diag_1Injury                                  -3.040 0.002367 **
## group_diag_1Musculoskeletal                         -2.764 0.005703 **
## group_diag_1n.a.                                     0.780 0.435261
## group_diag_1Neoplasms                                0.261 0.793751
## group_diag_1Other                                   -3.842 0.000122 ***
## group_diag_1Respiratory                             -4.929 8.28e-07 ***
## group_diag_2Diabetes                                 2.558 0.010540 *
## group_diag_2Digestive                               -1.341 0.179818
## group_diag_2Genitourinary                           -1.847 0.064715 .
## group_diag_2Injury                                  -1.147 0.251373
## group_diag_2Musculoskeletal                         -1.081 0.279760
## group_diag_2n.a.                                    -0.240 0.810628
## group_diag_2Neoplasms                                1.431 0.152475
## group_diag_2Other                                    0.182 0.855845
## group_diag_2Respiratory                             -0.776 0.437824
## group_diag_3Diabetes                                 1.523 0.127723
## group_diag_3Digestive                                1.204 0.228723
## group_diag_3Genitourinary                            0.679 0.497216
## group_diag_3Injury                                  -0.282 0.777827
## group_diag_3Musculoskeletal                          0.191 0.848546
## group_diag_3n.a.                                    -0.061 0.950999
## group_diag_3Neoplasms                                0.440 0.659958
## group_diag_3Other                                   -0.619 0.535594
## group_diag_3Respiratory                              0.536 0.591731
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                          edf Ref.df  Chi.sq  p-value
## s(time_in_hospital)   2.853  2.983  14.884  0.00138 **
## s(num_lab_procedures) 1.938  2.285   4.715  0.10270
## s(num_procedures)     1.006  1.011   0.639  0.42878
## s(num_medications)    2.855  2.982  23.526 7.94e-05 ***
## s(number_outpatient)  1.807  2.140   2.910  0.28190
## s(number_emergency)   2.039  2.226  62.178  < 2e-16 ***
## s(number_inpatient)   2.112  2.426 394.325  < 2e-16 ***
## s(number_diagnoses)   1.001  1.002  20.792 5.95e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.071   Deviance explained = 8.46%
## UBRE = -0.27579  Scale est. = 1         n = 33426
```

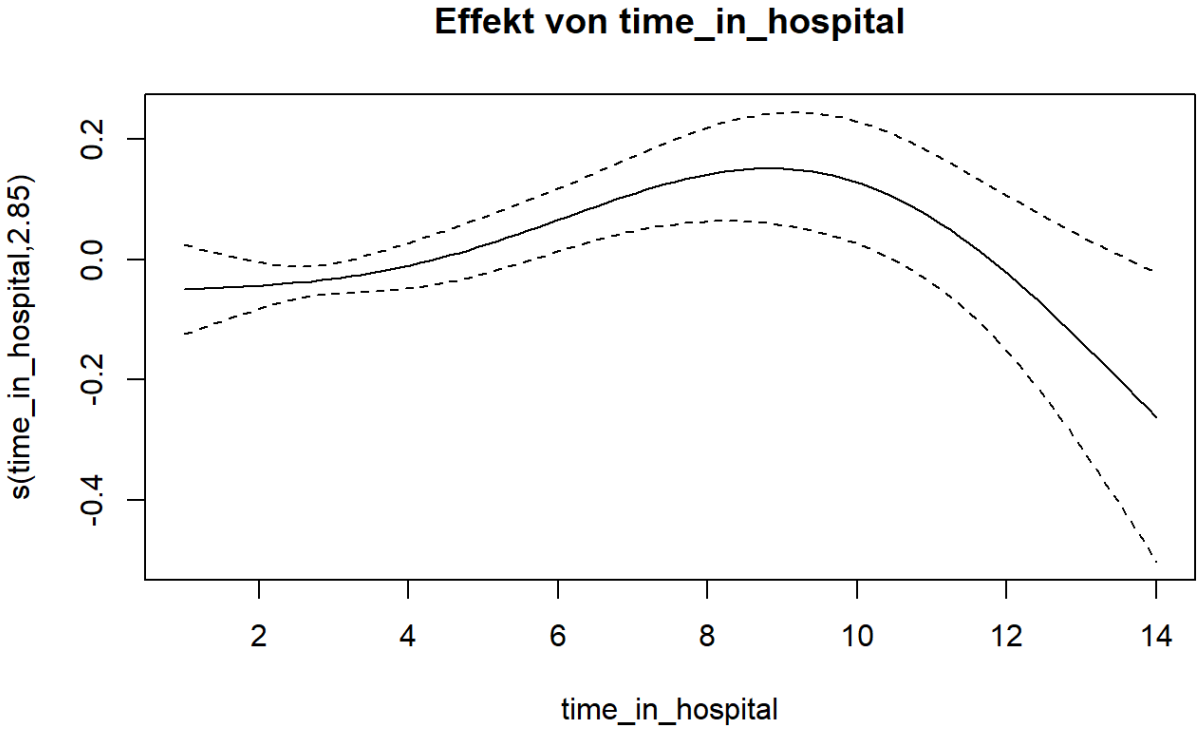
Insbesondere zeigen die Ereignisraten-Plots in Aufgabe R2c), dass die Merkmale `number_inpatient` und `time_in_hospital` aufgrund ihrer nicht-linearen Verläufe gut für die Verwendung in einem GAM geeignet sind. Beide Merkmale weisen zu Beginn einen linearen Anstieg auf, bevor sie ab einem gewissen Punkt ein Plateau erreichen. Ein weiteres geeignetes Merkmal ist `num_procedures` , da dieses in dem Plot in Aufgabe R2c) eine wellenartige Form zeigt. Diese Merkmale könnten in einem GAM zusätzliche, nicht-lineare Effekte erfassen und so zu einer verbesserten Modellgüte beitragen.

Abschließend visualisieren wir die geschätzten Effekte der Merkmale `number_inpatient` , `time_in_hospital` und `num_procedures` .

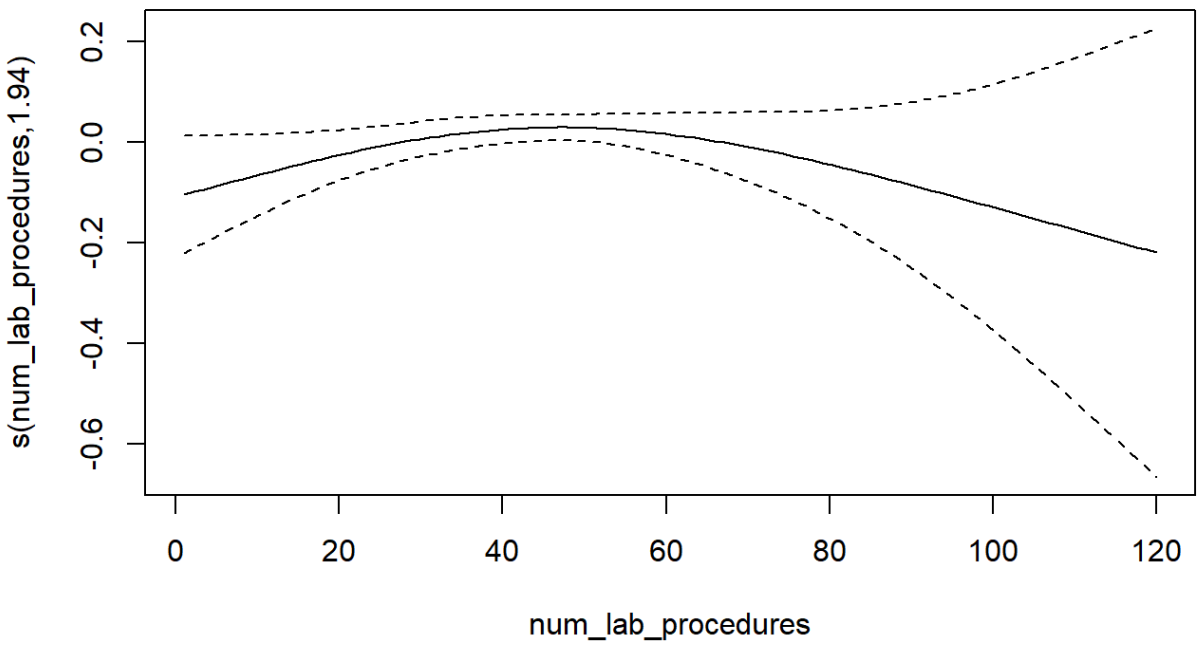
```
# Visualisiere die geschätzten Effekte für 'number_inpatient'
plot(gam_model, select = 7, scale  = 0, main = "Effekt von number_inpatient")
```




```
# Visualisiere die geschätzten Effekte für 'time_in_hospital'
plot(gam_model, select = 1, scale  = 0, main = "Effekt von time_in_hospital")
```




```
# Visualisiere die geschätzten Effekte für 'num_procedures'
plot(gam_model, select = 2, scale  = 0, main = "Effekt von num_procedures")
```



Um die Modellleistung zu bewerten, berechnen wir die Vorhersagen des GAM-Modells auf dem Testdatensatz und werten den AUC-Wert aus.

```
# Bestimme die Vorhersagen des GAM-Modells auf dem Testdatensatz
gam_predictions <- predict(gam_model, newdata = test_bin, type = "response")

# Berechne den AUC-Wert für das GAM-Modell
auc_gam <- calculate_auc(true_labels = y_test,
                         predicted_probs = gam_predictions,
                         verbose = TRUE)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
## AUC: 0.6896
```

d) Vergleich und Diskussion: Vergleichen Sie die Modellgüten der folgenden Ansätze:

- **Logistische Regression ohne Interaktionen aus Aufgabe R4a)**
- **Logistische Regression mit Interaktionen aus Aufgabe R4b)**
- **LASSO-Modell aus Aufgabe R5a)**
- **Ridge-Regression aus Aufgabe R5b)**
- **Generalisiertes Additives Modell aus Aufgabe R5c)**

Diskutieren Sie, welches dieser Modelle sich am besten für die vorliegende Prognoseaufgabe eignet und begründen Sie Ihre Entscheidung.

Um die verschiedenen Modelle zu vergleichen, betrachten wir die AUC-Werte der Vorhersagen auf den Testdaten. Die AUC-Werte geben Aufschluss über die Diskriminierungsfähigkeit der Modelle. Dazu erstellen wir eine übersichtliche Tabelle, die die AUC-Werte aller betrachteten Modelle zusammenfasst.

```
# Vergleich der AUC-Werte
model_comparison <- data.frame(
  Model = c("Log. Reg. ohne Interaktionen", "Log. Reg. mit Interaktionen", "LASSO", "Ridge", "GAM"),
  AUC = round(c(auc_logreg_without$auc, auc_logreg_with$auc,
                auc_lasso$auc, auc_ridge$auc, auc_gam$auc), 4)
)
model_comparison
```

Model <chr>	**AUC** <dbl>
Log. Reg. ohne Interaktionen	0.6887
Log. Reg. mit Interaktionen	0.6896
LASSO	0.6884
Ridge	0.6865
GAM	0.6896

5 rows

Grundsätzlich haben alle fünf Modelle eine ähnliche Güte mit Blick auf den AUC-Wert, wobei das GAM mit einem AUC von 0.6896 und die Logistische Regression mit Interaktionen mit einem AUC von 0.6896 die besten Werte aufweisen.

Allerdings bietet LASSO einen entscheidenden Vorteil in Bezug auf die Interpretierbarkeit: Durch die inhärente Modellselektion, bei der weniger bedeutende Merkmale durch Null-Koeffizienten ausgeschlossen werden, lässt sich klar erkennen, welche Variablen die wichtigsten Prädiktoren sind. Das macht LASSO besonders attraktiv, wenn Transparenz und Erklärung wichtiger sind als ein minimaler AUC-Vorteil.

Aufgrund der höheren Interpretierbarkeit und kürzeren Laufzeit bei nur minimal geringerem AUC-Wert eignet sich LASSO in unseren Augen unter den betrachteten Modellen am besten für die vorliegende Prognoseaufgabe.

Aufgabe R6: Ternäre Klassifikation

In dieser Aufgabe wenden Sie die Machine-Learning-Modelle Multinomiale Logistische Regression und XGBoost zur ternären Klassifikation an. Als Eingabedaten dienen die in Teilaufgabe R3a) aufbereiteten Trainings-, Validierungs- und Testdatensätze auf Basis von `diabetic_data_ter`.

a) Evaluationsmetriken bei binärer und ternärer Klassifikation: Erläutern Sie zunächst, ob und ggf. warum die bisher verwendete Evaluationsmetrik AUC für die bisher betrachtete binäre Klassifikation gut geeignet ist, und gehen Sie dabei kurz auf je zwei Vor- und Nachteile im Vergleich zur Accuracy ein. Erläutern Sie zudem, welche Besonderheiten bei der Bewertung von Mehrklassen-Klassifikationsmodellen zu beachten sind. Gehen Sie dabei auf die Evaluationsmetriken Accuracy, Balanced Accuracy und Makro-F1-Score ein und erklären Sie, in welchen Fällen diese Kennzahlen besonders nützlich sind. Schreiben Sie eine Funktion, die diese drei Kennzahlen berechnet und kompakt ausgibt.

AUC für binäre Klassifikation:

Die Evaluationsmetrik AUC (Area Under the ROC Curve) ist eine sehr gut geeignete Kennzahl für die binäre Klassifikation, da sie die Modellleistung in Bezug auf alle möglichen Schwellenwerte des Klassifikators bewertet. AUC misst die Fähigkeit eines Modells, zwischen den Klassen zu unterscheiden. Sie berücksichtigt also sowohl die falsch positiven als auch die falsch negativen Vorhersagen und liefert eine umfassende Einschätzung der Modellgüte.

Vor- und Nachteile im Vergleich zur Accuracy:

- *Vorteile:*
    1. Sensitivität gegenüber Klassifizierungsfehlern: AUC ist besonders nützlich, wenn die Daten unausgewogen sind, da sie die Leistung auch bei einer Klassifikation von Minderheitsklassen gut widerspiegelt.
    2. Schwellenunabhängigkeit: AUC bewertet die Klassifikationsleistung über alle möglichen Klassifikationsschwellen hinweg, wodurch sie flexibler ist und nicht auf eine spezifische Schwelle angewiesen ist.
- **Nachteile:**
    1. Kein Bezug zur Konfusionsmatrix: AUC berücksichtigt nicht die tatsächlichen Klassifikationszahlen und kann schwer zu interpretieren sein, wenn genauere Fehleranalysen erforderlich sind.
    2. Nicht leicht intuitiv: Der Wert der AUC ist nicht direkt mit einer Fehlerrate (z.B. Accuracy) zu verbinden, was die Interpretation für eine Zielgruppe ohne technisches Vorwissen erschweren kann.

Besonderheiten bei Mehrklassen-Klassifikation:

Bei der Mehrklassen-Klassifikation können klassische binäre Evaluationsmetriken wie AUC nicht direkt angewendet werden, da AUC nur die Leistung bei der Unterscheidung zwischen zwei Klassen misst. Um die Leistung eines Modells bei mehreren Klassen zu beurteilen, müssen Metriken entwickelt oder angepasst werden, die alle Klassen berücksichtigen. Hier kommen Kennzahlen wie Accuracy, Balanced Accuracy und Makro-F1-Score ins Spiel. Diese Metriken helfen dabei, das Modellverhalten bei Mehrklassenaufgaben fair zu bewerten, insbesondere bei unausgewogenen oder komplexen Datensätzen mit mehreren Klassen.

- *Accuracy*: Misst den Anteil der richtig klassifizierten Instanzen. Sie ist einfach zu berechnen, kann jedoch bei unausgewogenen Klassenverhältnissen irreführend sein.
- *Balanced Accuracy*: Diese Metrik berechnet die Genauigkeit für jede Klasse und gewichtet sie entsprechend der Häufigkeit jeder Klasse im Datensatz. Dadurch wird verhindert, dass die Dominanz größerer Klassen die Gesamtbewertung verzerrt. Sie ist besonders nützlich, wenn die Klassen im Datensatz unausgewogen sind, da sie der Modellleistung eine fairere Bewertung verleiht.
- *Makro-F1-Score*: Berechnet den F1-Score für jede Klasse und nimmt den Durchschnitt über alle Klassen hinweg. Diese Metrik ist besonders hilfreich, wenn ein Modell in einigen Klassen sehr gut, aber in anderen weniger gut ist. Sie stellt sicher, dass das Modell für alle Klassen eine ähnliche Leistung aufweist, unabhängig von der Häufigkeit der Klassen.

Besondere Nützlichkeit der Metriken:

- *Accuracy* ist vor allem dann hilfreich, wenn die Daten gut ausgeglichen sind.
- *Balanced Accuracy* ist besonders in Szenarien mit unausgewogenen Klassen sinnvoll.
- *Makro-F1-Score* kommt besonders zum Einsatz, wenn jede Klasse eine ähnliche Bedeutung hat und man sicherstellen möchte, dass das Modell in allen Klassen gut abschneidet.

Funktion zur Berechnung der Metriken:

```r
# Funktion zur Berechnung der Modellgüte
calculate_ternary_metrics <- function(y_true, y_pred) {
  # Berechnung der Accuracy
  accuracy <- sum(y_true == y_pred) / length(y_true)

  # Berechnung der Balanced Accuracy
  balanced_accuracy <- mean(sapply(unique(y_true), function(class) {
    tp <- sum(y_true == class & y_pred == class)
    fn <- sum(y_true == class & y_pred != class)
    tp / (tp + fn)
  }))

  # Berechnung des Makro-F1-Scores
  f1_score <- mean(sapply(unique(y_true), function(class) {
    tp <- sum(y_true == class & y_pred == class)
    fp <- sum(y_true != class & y_pred == class)
    fn <- sum(y_true == class & y_pred != class)
    precision <- tp / (tp + fp)
    recall <- tp / (tp + fn)
    if (is.nan(precision) | is.nan(recall)) {
      return(0)
    }
    (2 * precision * recall) / (precision + recall)
  }))

  # Rückgabe der Metriken
  return(list(
    Accuracy = accuracy,
    Balanced_Accuracy = balanced_accuracy,
    Macro_F1_Score = f1_score
  ))
}
```

b) Ternäre Klassifikation mit Multinomialer Logistischer Regression: Trainieren Sie eine Multinomiale Logistische Regression (z. B. mit der Funktion `multinom` aus dem Paket `nnet`) auf dem Trainingsdatensatz. Wählen Sie nur Merkmale aus, die mindestens zwei verschiedene Werte haben, und schließen Sie die Zielvariable sowie die Diagnosespalten (`diag_1`, `diag_2`, `diag_3`) aus. Optimieren Sie die Klassenschwellen, um die Verteilung der vorhergesagten Klassen an die der tatsächlichen Werte von `TARGET` anzupassen. Berechnen Sie die Modellgüte auf dem Testdatensatz mithilfe der optimierten Schwellen und der Funktion aus Teilaufgabe R6a). Erstellen und interpretieren Sie zusätzlich eine geeignete Konfusionsmatrix, die die Vorhersagen des Modells mit den tatsächlichen Werten der Testdaten gegenüberstellt.

Zur Modellierung der ternären Zielvariable `TARGET` verwenden wir eine Multinomiale Logistische Regression mittels `multinom` aus dem Paket `nnet`. Vorab entfernen wir Merkmale, die nur einen Wert haben, sowie die Zielvariable und irrelevante Diagnosespalten.

```r
# Filtere nur die Merkmale aus, die mehr als einen einzigartigen Wert haben
valid_features_ter <- colnames(train_ter)[sapply(train_ter, function(col) length(unique(col)) > 1)]

# Exkludiere die Zielvariable und die Diagnosespalten
valid_features_ter <- setdiff(valid_features, c("TARGET", "diag_1", "diag_2", "diag_3"))

# Erstelle die Formel unter Verwendung der validen Merkmale
formula_ter <- as.formula(paste("TARGET ~", paste(valid_features, collapse = " + ")))

# Trainieren der Multinomialen Logistischen Regression
multinom_model <- nnet::multinom(formula, data = train_ter, MaxNWts = 1000000)
```

```
## # weights:  570 (378 variable)
## initial  value 53812.227124
## iter  10 value 44837.777856
## iter  20 value 43989.214411
## iter  30 value 43740.508663
## iter  40 value 43341.673384
## iter  50 value 42798.357373
## iter  60 value 42224.819112
## iter  70 value 41831.874787
## iter  80 value 41525.611919
## iter  90 value 41384.496020
## iter 100 value 41316.363484
## final  value 41316.363484
## stopped after 100 iterations
```

Für die Klassenvorhersage definieren wir Schwellenwerte, die die Verteilung der tatsächlichen Klassen im Testdatensatz berücksichtigen. Mit der Funktion `calculate_ternary_metrics` aus Teilaufgabe R6a) berechnen wir anschließend die Modellgüte, wie Genauigkeit, Balanced Accuracy und Macro-F1-Score.

Zunächst werden die Schwellenwerte festgelegt und dann optimiert, um die Vorhersageverteilung den tatsächlichen Verteilungen der Zielvariable anzupassen. Nach der Optimierung erfolgt die Vorhersage und die Berechnung der Metriken zur Bewertung der Modellgüte.

```r
# Optimierungsfunktion für Klassenschwellen
optimize_thresholds <- function(probabilities, actual_values, initial_thresholds) {
  # Zielwert: Marginale Verteilung der vorhergesagten Klassen sollte der tatsächlichen ähneln
  target_distribution <- table(factor(actual_values, levels = c("<30", ">30", "NO"))) / length(actual_values)

  # Verlustfunktion zur Minimierung der Diskrepanz
  loss_function <- function(thresholds) {
    predicted_classes <- apply(probabilities, 1, function(row) {
      if (row[1] > thresholds[1]) return("<30")
      else if (row[2] > thresholds[2]) return(">30")
      else if (row[3] > thresholds[3]) return("NO")
      else return(c("<30", ">30", "NO")[which.max(row)])
    })
    # Berechne Verteilung der vorhergesagten Klassen
    predicted_distribution <- table(factor(predicted_classes, levels = c("<30", ">30", "NO"))) / length(predicted_classes)
    # Berechnung der Diskrepanz
    sum(abs(predicted_distribution - target_distribution))
  }

  # Optimierung der Schwellenwerte
  optimal_thresholds <- optim(
    par = initial_thresholds,            # Startwerte für Schwellen
    fn = loss_function,                  # Verlustfunktion
    method = "L-BFGS-B",                 # Optimierungsverfahren
    lower = c(0, 0, 0),                  # Untergrenzen
    upper = c(1, 1, 1)                   # Obergrenzen
  )$par

  return(optimal_thresholds)
}

# Vorhersage der Wahrscheinlichkeiten für den Testdatensatz
probabilities_multinom <- predict(multinom_model, newdata = test_ter, type = "probs")

# Klassenschwellen initialisieren
initial_thresholds <- c(0.1, 0.3, 0.5)

# Schwellenwerte optimieren
optimized_thresholds <- optimize_thresholds(probabilities_multinom, test_ter$TARGET, initial_thresholds)

# Optimierte Schwellenwerte anzeigen
cat("Optimierte Schwellenwerte:\n", optimized_thresholds, "\n")
```

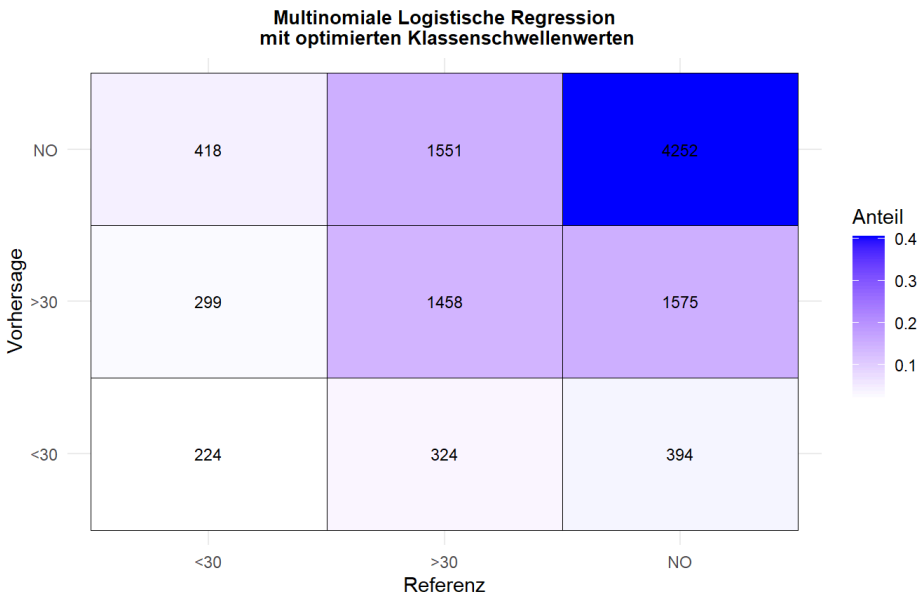
```
## Optimierte Schwellenwerte:
##  0.1670346 0.3523208 0.5
```

```r
# Vorhersage basierend auf optimierten Schwellen
predictions_optimized <- apply(probabilities_multinom, 1, function(row) {
  if (row[1] > optimized_thresholds[1]) return("<30")
  else if (row[2] > optimized_thresholds[2]) return(">30")
  else if (row[3] > optimized_thresholds[3]) return("NO")
  else return(c("<30", ">30", "NO")[which.max(row)])
})

# Konvertiere die tatsächlichen Werte und Vorhersagen zu Faktoren
y_test_factor <- factor(test_ter$TARGET, levels = c("<30", ">30", "NO"))
predictions_factor <- factor(predictions_optimized, levels = c("<30", ">30", "NO"))

# Berechnung der Metriken mit optimierten Schwellen
metrics_multinom_optimized <- calculate_ternary_metrics(y_test_factor, predictions_factor)

# Ergebnisse ausgeben
cat("Modellgüte mit optimierten Schwellen:\n",
    "Accuracy: ", metrics_multinom_optimized$Accuracy * 100, "%\n",
    "Balanced Accuracy: ", metrics_multinom_optimized$Balanced_Accuracy * 100, "%\n",
    "Macro-F1-Score: ", metrics_multinom_optimized$Macro_F1_Score * 100, "%\n")
```

```
## Modellgüte mit optimierten Schwellen:
##  Accuracy:  56.54121 %
##  Balanced Accuracy:  45.29933 %
##  Macro-F1-Score:  45.2973 %
```

Abschließend erstellen und visualisieren wir die Konfusionsmatrix basierend auf den optimierten Schwellenwerten.

```r
# Erstellung der Konfusionsmatrix
confusion_matrix_multinom <- confusionMatrix(predictions_factor, y_test_factor)

# Visualisierung der Konfusionsmatrix
cm_multinom <- plot_confusion_matrix(confusion_matrix_multinom, "Multinomiale Logistische Regression\n mit optimierten Klass
         enschwellenwerten")
cm_multinom
```



c) Ternäre Klassifikation mit XGBoost: Trainieren Sie ein XGBoost-Modell, analog zu Teilaufgabe R3b). Optimieren Sie die Klassenschwellen, um die Verteilung der vorhergesagten Klassen an die der tatsächlichen Werte von `TARGET` anzupassen. Berechnen Sie die Modellgüte auf dem Testdatensatz mithilfe der optimierten Schwellen und der Funktion aus Teilaufgabe R6a). Erstellen und interpretieren Sie zusätzlich eine geeignete Konfusionsmatrix, die die Vorhersagen des Modells den tatsächlichen Werten der Testdaten gegenüberstellt. Vergleichen Sie die Ergebnisse mit denen aus Teilaufgabe R6b) und erläutern Sie, welches Modell für die vorliegende Problemstellung aufgrund der Modellgüte und der Konfusionsmatrix besser geeignet ist.

In dieser Aufgabe trainieren wir ein XGBoost-Modell für eine ternäre Klassifikation, um die Modellgüte zu berechnen und eine Konfusionsmatrix zu erstellen. Wir verwenden denselben ternären Fehlermaß wie in Teilaufgabe R6a). Abschließend visualisieren und interpretieren wir die Konfusionsmatrix, um die Vorhersagen des Modells mit den tatsächlichen Werten der Testdaten gegenüberzustellen.

Zunächst konvertieren wir die Trainings- und Testdaten in das DMatrix-Format, das von XGBoost für effizientere Berechnungen genutzt wird. Wir stellen sicher, dass wir die richtigen Zielwerte für die ternäre Klassifikation haben.

```
# Aufbereitung der Trainings- und Testdaten
X_train <- train_ter %>% select(-TARGET)
X_val <- val_ter %>% select(-TARGET)
X_test <- test_ter %>% select(-TARGET)

# Konvertiere die Zielvariable in eine faktorisierte Form
y_train <- train_ter$TARGET
y_val <- val_ter$TARGET
y_test <- test_ter$TARGET

# Konvertiere Trainings- und Testdaten in das DMatrix-Format
dtrain <- xgb.DMatrix(data.matrix(X_train), label = as.numeric(y_train) - 1)
dval <- xgb.DMatrix(data.matrix(X_val), label = as.numeric(y_val) - 1)
dtest <- xgb.DMatrix(data.matrix(X_test))
```

Die Hyperparameter für das Modell werden wie folgt festgelegt:

```
# Define the XGBoost parameters for multiclass classification
xgb_params <- list(
  objective = "multi:softprob",  # Output probabilities per class
  num_class = 3,  # For 3 classes
  colsample_bytree = 0.7,
  subsample = 0.7,
  max_depth = 5,
  eta = 0.1,
  eval_metric = "mlogloss",
  seed = 123,
  nthread = 4
)

# Beobachtungslisten für Training und Test
watchlist <- list(train = dtrain, valid = dval)
```

Nun trainieren wir das Modell unter Verwendung der definierten Parameter. Die Kreuzvalidierung, wie sie auch in Teilaufgabe R6a) verwendet wurde, stellen wir zur Bestimmung der optimalen Iterationen ein:

```r
# Kreuzvalidierung durchführen
xgb_cv <- xgb.cv(
  params = xgb_params,
  data = dtrain,
  nfold = 5,
  nrounds = 100,
  early_stopping_rounds = 10,
  maximize = FALSE,
  verbose = 0
)

# Optimale Anzahl der Iterationen
optimal_nrounds <- xgb_cv$best_iteration
```

Nach der Bestimmung der besten Anzahl an Iterationen trainieren wir das finale Modell:

```r
# Trainiere das finale Modell mit der optimalen Anzahl der Iterationen
gb_dt <- xgb.train(
  params = xgb_params,
  data = dtrain,
  nrounds = optimal_nrounds,
  watchlist = watchlist,
  print_every_n = 10
)
```

```
## Warning in xgb.train(params = xgb_params, data = dtrain, nrounds =
## optimal_nrounds, : xgb.train: `seed` is ignored in R package.  Use `set.seed()`
## instead.
```

```
## [1]  train-mlogloss:1.066778 valid-mlogloss:1.067025
## [11] train-mlogloss:0.908670 valid-mlogloss:0.912190
## [21] train-mlogloss:0.860144 valid-mlogloss:0.867202
## [31] train-mlogloss:0.840738 valid-mlogloss:0.851868
## [41] train-mlogloss:0.830033 valid-mlogloss:0.845144
## [51] train-mlogloss:0.822236 valid-mlogloss:0.841343
## [61] train-mlogloss:0.816304 valid-mlogloss:0.839155
## [71] train-mlogloss:0.810853 valid-mlogloss:0.837473
## [81] train-mlogloss:0.806281 valid-mlogloss:0.836376
## [91] train-mlogloss:0.801896 valid-mlogloss:0.835498
## [100]    train-mlogloss:0.798304 valid-mlogloss:0.834876
```

Wie zuvor berechnen wir zunächst die optimalen Schwellenwerte und verwenden dann die Funktion `calculate_ternary_metrics`, um das XGBoost-Modell zu bewerten.

```r
# Vorhersage der Wahrscheinlichkeiten auf den Testdatensatz
probabilities <- predict(gb_dt, newdata = dtest)
probabilities <- matrix(probabilities, ncol = 3, byrow = TRUE)  # Reshape the predictions into a matrix

# Klassenschwellen initialisieren
initial_thresholds <- c(0.1, 0.3, 0.5)

# Optimierungsfunktion verwenden
optimized_thresholds <- optimize_thresholds(probabilities, y_test, initial_thresholds)

# Optimierte Schwellenwerte ausgeben
cat("Optimierte Schwellenwerte:\n", optimized_thresholds, "\n")
```

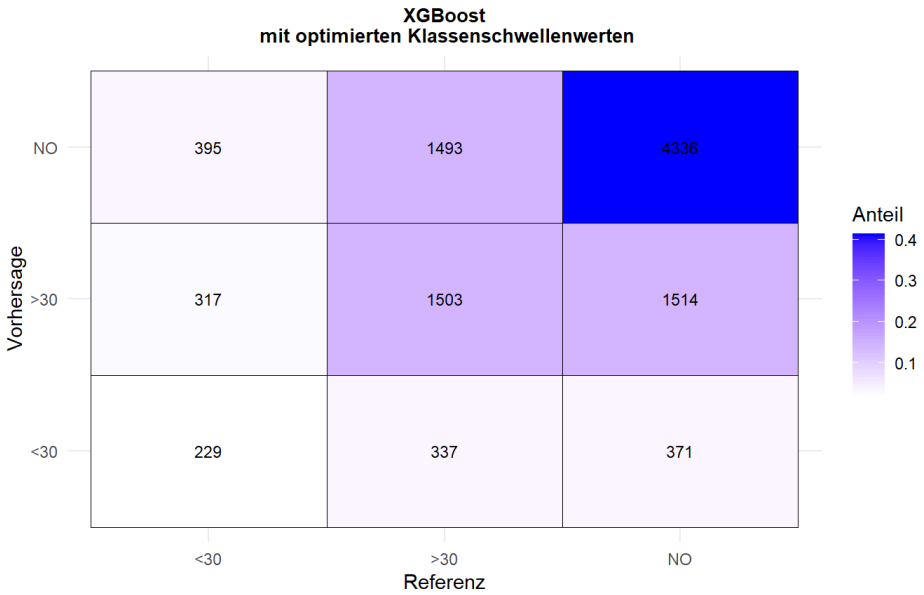
```
## Optimierte Schwellenwerte:
##  0.1468027 0.3462064 0.5
```

```r
# Vorhersage der Klassen basierend auf optimierten Schwellenwerten
predictions_optimized <- apply(probabilities, 1, function(row) {
  if (row[1] > optimized_thresholds[1]) return("<30")
  else if (row[2] > optimized_thresholds[2]) return(">30")
  else if (row[3] > optimized_thresholds[3]) return("NO")
  else return(c("<30", ">30", "NO")[which.max(row)])  # Default to max probability
})

# Konvertiere Zielwerte und Vorhersagen in Faktoren mit richtigen Levels
y_test_factor <- factor(y_test, levels = c("<30", ">30", "NO"))
predictions_factor <- factor(predictions_optimized, levels = c("<30", ">30", "NO"))

# Berechnung der Metriken mit optimierten Schwellenwerten
metrics_xgb_optimized <- calculate_ternary_metrics(y_test_factor, predictions_factor)

# Ergebnisse ausgeben
cat("Modellgüte mit optimierten Schwellenwerten:\n",
    "Accuracy: ", metrics_xgb_optimized$Accuracy * 100, "%\n",
    "Balanced Accuracy: ", metrics_xgb_optimized$Balanced_Accuracy * 100, "%\n",
    "Macro-F1-Score: ", metrics_xgb_optimized$Macro_F1_Score * 100, "%\n")
```

```
## Modellgüte mit optimierten Schwellenwerten:
##  Accuracy:  57.81801 %
##  Balanced Accuracy:  46.37658 %
##  Macro-F1-Score:  46.386 %
```

Wie bereits zuvor, erstellen wir nun die Konfusionsmatrix für das XGBoost-Modell und visualisieren sie unter Zugrundelegung der optimierten Schwellenwerte.

```
# Erstellung der Konfusionsmatrix
confusion_matrix_xgb_ter <- confusionMatrix(predictions_factor, y_test_factor)

# Visualisierung der Konfusionsmatrix
cm_xgb <- plot_confusion_matrix(confusion_matrix_xgb_ter, "XGBoost\n mit optimierten Klassenschwellenwerten")
cm_xgb
```



Zum Vergleich der Modellgüte stellen wir die zuvor berechneten Evaluationsmetriken der beiden Modelle in einer Tabelle gegenüber.

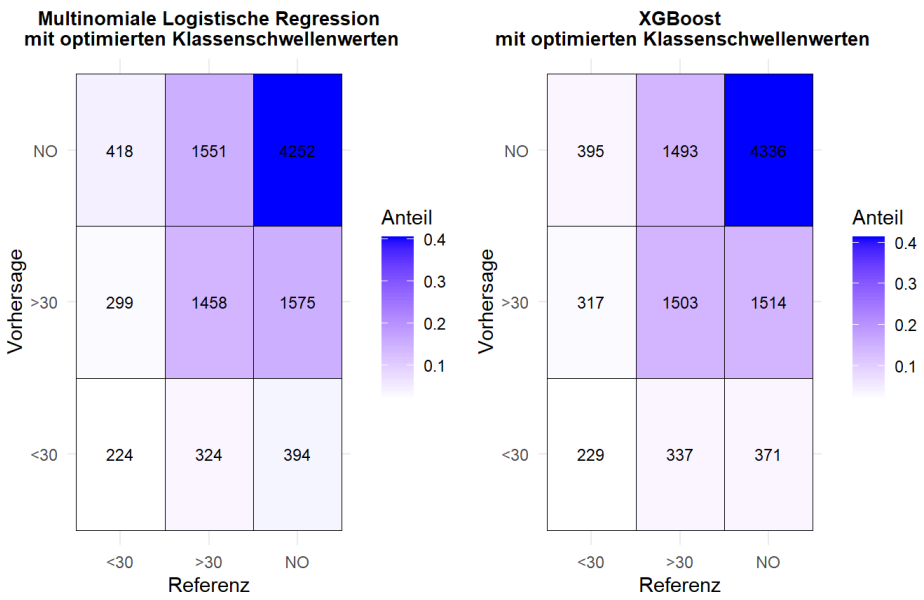
```
compare_model_performance <- function(metrics_multinom_optimized, metrics_xgb_optimized) {
  # Extract performance metrics for multinom model
  multinom_metrics <- data.frame(
    Model = "Multinomial",
    Accuracy = metrics_multinom_optimized$Accuracy * 100,
    Balanced_Accuracy = metrics_multinom_optimized$Balanced_Accuracy * 100,
    Macro_F1_Score = metrics_multinom_optimized$Macro_F1_Score * 100
  )

  # Extract performance metrics for xgb model
  xgb_metrics <- data.frame(
    Model = "XGBoost",
    Accuracy = metrics_xgb_optimized$Accuracy * 100,
    Balanced_Accuracy = metrics_xgb_optimized$Balanced_Accuracy * 100,
    Macro_F1_Score = metrics_xgb_optimized$Macro_F1_Score * 100
  )

  # Combine both dataframes into one
  performance_comparison <- rbind(multinom_metrics, xgb_metrics)

  return(performance_comparison)
}

# Assuming metrics_multinom_optimized and metrics_xgb_optimized are already defined
model_comparison <- compare_model_performance(metrics_multinom_optimized, metrics_xgb_optimized)
model_comparison
```

Model <chr>	Accuracy <dbl>	Balanced_Accuracy <dbl>	Macro_F1_Score <dbl>
Multinomial	56.54121	45.29933	45.2973
XGBoost	57.81801	46.37658	46.3860

2 rows

Zum Vergleich der Konfusionsmatrizen visualisieren wir die Ergebnisse beider Modelle nebeneinander.

```
multiplot(cm_multinom, cm_xgb, cols = 2)
```



Basierend auf der Modellgüte und der Analyse der Konfusionsmatrix lässt sich das XGBoost-Modell als besser geeignet für die vorliegende Problemstellung feststellen.

Erstens zeigen alle drei Metriken – Accuracy, Balanced Accuracy und Macro-F1-Score – für das XGBoost-Modell höhere Werte im Vergleich zum multinomialen Modell. Dies weist darauf hin, dass das XGBoost-Modell eine insgesamt bessere Leistung aufweist und sowohl die korrekten Vorhersagen häufiger trifft als auch eine ausgeglichenere Leistung in Bezug auf die verschiedenen Klassen liefert.

Zweitens zeigt die Konfusionsmatrix des XGBoost-Modells ein vorteilhafteres Verhalten, da die Fehlklassifikationen näher an den tatsächlichen Klassen liegen. Das bedeutet, dass das Modell bei falschen Vorhersagen tendenziell eine ähnliche Klasse wie die tatsächliche auswählt, anstatt die Klasse weit zu verfehlen. Zum Beispiel, wenn das tatsächliche Ergebnis `<30` ist, kommt es bei XGBoost häufiger vor, dass das Modell die falsche Klasse `>30` anstelle von `NO` vorhersagt, was näher an der richtigen Klasse liegt. Dies ist besonders wichtig, wenn die Kosten von Fehleinschätzungen in einigen Klassen höher sind als in anderen.

d) One-vs-One: Eine weitere Bewertungsmöglichkeit besteht darin, den AUC-Wert pro Klasse paarweise zu berechnen. Führen Sie dazu für das im Aufgabenteil b) trainierte Multinomiale Logistische Regressionsmodell eine One-vs-One-Analyse durch, bei der für jedes mögliche Klassenpaar von `TARGET` ein separates binäres Klassifikationsproblem aufgestellt wird und ermitteln Sie jeweils auf Basis der Testdaten den AUC-Wert. Vergleichen und interpretieren Sie das Ergebnis untereinander sowie mit dem Ergebnis der entsprechenden einfachen Logistischen Regression aus Aufgabe R4 (Binärklassifikation `<30` vs. `NO`).

```
# Auswahl aller Klassenpaare
class_pairs <- combn(levels(test_ter$TARGET), 2, simplify = FALSE)

# Initialisierung von Listen für Konfusionsmatrizen, AUC-Werte und zusätzliche Infos
conf_matrices <- list()
ovo_auc <- numeric(length(class_pairs))

# Iteration über jedes Klassenpaar und Berechnung der AUC
for (i in seq_along(class_pairs)) {
  pair <- class_pairs[[i]]

  # Daten für das aktuelle Klassenpaar filtern
  pair_data <- test_ter[test_ter$TARGET %in% pair, ]

  # Vorhersage der Wahrscheinlichkeiten für das aktuelle Klassenpaar
  pair_probs <- predict(multinom_model, newdata = pair_data, type = "probs")

  # Binarisierung der Zielvariable: erste Klasse = 1, zweite Klasse = 0
  binary_target <- ifelse(pair_data$TARGET == pair[1], 1, 0)

  # Berechnung der AUC mit der Funktion calculate_auc
  auc_result <- calculate_auc(binary_target, pair_probs[, pair[1]], verbose = FALSE)
  ovo_auc[i] <- auc_result$auc
}

# Ausgabe der AUC-Werte für jedes Klassenpaar
cat("\nAUC-Werte der One-vs-One Analyse für multinomiale Logistische Regression (ternär):\n")
```

```
##
## AUC-Werte der One-vs-One Analyse für multinomiale Logistische Regression (ternär):
```

```
for (i in seq_along(class_pairs)) {
  pair <- class_pairs[[i]]

  # Stellen Sie sicher, dass wir die AUC korrekt berechnen: pair[1] als positive Klasse (1) und pair[2] als negative Klasse
      (0)
  cat("Klassenpaar", pair[1], "vs", pair[2], ":", round(ovo_auc[i], 4), "\n")
}
```

```
## Klassenpaar <30 vs >30 : 0.6145
## Klassenpaar <30 vs NO : 0.6876
## Klassenpaar >30 vs NO : 0.6542
```

```
# Berechnung der AUC mit der calculate_auc Funktion
cat("\nVergleich mit Aufgabe R4, Binärklassifikation: Logistische Regression (via nnet) für '<30 vs. NO'\n")
```

```
##
## Vergleich mit Aufgabe R4, Binärklassifikation: Logistische Regression (via nnet) für '<30 vs. NO'
```

```
print(auc_lr_nnet$auc)
```

```
## Area under the curve: 0.6892
```

Der AUC-Wert für die beiden entferntesten Klassenpaare ‘ `<30` vs. `NO` ’, also Wiedereinweisung innerhalb von 30 Tagen gegenüber keine Wiedereinweisung, ist deutlich höher als der der beiden anderen, direkt aneinander grenzenden Klassenpaare. Insbesondere die Trennung der Klassen mit Wiedereinweisung ‘ `<30` vs. `>30` ’ (also innerhalb von 30 Tagen gegenüber später) fällt deutlich ab, hier kann das Modell die Klassen mit Abstand am schlechtesten unterscheiden.

Im Vergleich zum Ergebnis der Binärklassifikation mittels Logistischer Regression (ebenfalls mit nnet) in Aufgabe R4 fällt der entsprechende AUC-Wert ‘ `<30` vs. `NO` ’ der Multinomialen Logistischen Regression etwas schlechter aus. Der Unterschied kann im Hinblick auf die großen AUC-Unterschiede bei der Multinomialen Logistischen Regression als geringfügig erachtet werden.

e) Fazit: Ziehen Sie ein Fazit und gehen Sie dabei auf folgende Fragen ein: Welche Schlussfolgerungen lassen sich aus den Ergebnissen ziehen? Welchen Mehrwert hat die ternäre Klassifikation? Wie ist der Unterschied in der Datenbasis binär vs. ternär zu bewerten? Welche Modelle würden Sie abschließend empfehlen?

Eine Schlussfolgerung aus d) wäre, dass zumindest in diesem Fall die Multinomiale Logistische Regression die in den vorangegangenen Aufgaben zur Binärklassifikation erstellte einfache Logistische Regression ohne Abstriche bei der Klassifikationsleistung ersetzen könnte.

Ein möglicher Mehrwert der ternären Klassifikation für die Vorhersage von Wiedereinweisungen wären Aussagen zum “Graubereich” der späten Wiedereinweisungen nach 30 Tagen. Dieser Bereich wurde bei der Binärklassifikation entfernt, die Daten waren daher im Sinne einer binären Entscheidung unvollständig.

Neben dem bereits in Aufgabe R5 für die binäre Klassifikation empfohlenen, gut interpretierbaren und parametersparsamen LASSO-Modell kann im Hinblick auf eine hohe Prognosegüte das XGBoost-Modell empfohlen werden. Es liefert sowohl für die binäre als auch die ternäre Klassifikation mit Abstand die genauesten Prognosen. Neben dem hohen AUC-Wert sprechen dafür auch die bei der ternären Klassifikation gezeigten Konfusionsmatrizen und die ermittelten Kennzahlen Accuracy, Balanced Accuracy und Makro-F1-Score, die allesamt am besten sind.

Prüfung Immersion 13.4. bis 13.5.2025, Teil II (Python)
"Diabetes Hospital Readmission"
Fokus: Neuronale Netzwerke und Embeddings

WICHTIGE HINWEISE ZUR BEARBEITUNG DIESES AUFGABENTEILS

Der zweite Teil der Immersion-Prüfung ist in Python umzusetzen. Dabei soll ein bereits bestehendes und für geeignet befundenes Jupyter-Notebook als Vorlage verwendet und auf andere Daten und geänderte Fragestellungen angepasst werden.

Als Vorlage für die Bearbeitung der Aufgaben PT1, PT2, PT3 und PT7 b) dient das Notebook zum Anwendungsfall 4 der DAV (siehe https://aktuar.de/de/wissen/fachinformationen/detail/prognose-seltener-ereignisse-credit-scoring/) und muss gemäß Aufgabenstellung angepasst werden (d.h. Texte, Codes und Notebook-Zellen zielführend ändern, ergänzen, einfügen, löschen oder auskommentieren). Das als Vorlage zu verwendende Notebook befindet sich in einer um die Aufgabenstellung ergänzten und leicht bearbeiteten Version namens `template2-credit-scoring.ipynb` in den Prüfungsunterlagen und wird im Folgenden mit dem Kürzel CSN abgekürzt.

Die Aufgaben PT0, PT4 bis PT7 a) behandeln neue, bisher nicht in der Notebook-Vorlage behandelte Themen. Hierfür sind ausreichend viele Notebook-Zellen einzufügen und das Notebook ist der Aufgabenstellung entsprechend zu erweitern.

Für die Bearbeitung werden die im Aufgabenteil I (R-Notebook) erzeugten Datensätze `diabetic_data_bin.csv` und `icd9_data.csv` benötigt.

Bei der Überarbeitung ist darauf zu achten, dass Sektionsnummern des Ausgangsnotebooks CSN (in den ersten drei Aufgaben PT1 bis PT3) erhalten bleiben. Die dazugehörigen Überschriften können nach Bedarf angepasst werden. Inhaltlich sind die aus der Vorlage übernommenen Texte auf Deutsch zu übersetzen (z.B. mit DeepL) und insbesondere an die geänderten Sachverhalte (z.B. Daten und Ergebnisse) anzupassen. Kommentare in Code-Zellen müssen nicht übersetzt werden. Nicht benötigte Texte können gelöscht werden.

Aufgabe PT0: Aufsetzen einer Entwicklungsumgebung [Lernziel 5.1; 1 Punkt]

Für das Aufsetzen der Entwicklungsumgebung wird Python mit einer Version >= 3.10.11 empfohlen. Den Unterlagen wurde ein requirements.txt File hinzugefügt. Auf Basis dieses Files können die benötigten Pakete installiert werden. Die verwendete Python-Version ist auszugeben. Zudem sind die installierten Pakete übersichtlich (pro Zeile fünf Pakete mit Versionsnummer) auszugeben.

Lösungsvorschlag:

In [1]:
```
# !pip3 install -r /teamspace/studios/this_studio/PK_VADS_IMMERSION/2025/1_Erster_Ansatz/requirements.txt
```

In [2]:
```
# check python version
from platform import python_version

print(python_version())
```

```
3.10.16
```

In [3]:
```
import pkg_resources
installed_packages = list(pkg_resources.working_set)

# Anzahl der Pakete pro Zeile
packages_per_line = 5

# Ausgabe der Pakete in Gruppen
for i in range(0, len(installed_packages), packages_per_line):
    print(' | '.join([f"{pkg.project_name}=={pkg.version}".center(30) for pkg in installed_packages[i:i+packages_per_line]]))
```

```
        Mako==1.3.8           |         Markdown==3.7         |       MarkupSafe==3.0.2       |         PySocks==1.7.1        |          Py
YAML==6.0.2
      SQLAlchemy==2.0.36      |         absl-py==2.1.0        |       accelerate==0.34.2      |         adagio==0.2.6         |    aiohapp
yeyeballs==2.4.4
       aiohttp==3.11.11       |      aiohttp-cors==0.7.0      |        aiosignal==1.3.2       |        alembic==1.14.0        |     annota
ted-types==0.7.0
antlr4-python3-runtime==4.9.3 |         appdirs==1.4.4        |        asttokens==3.0.0       |       astunparse==1.6.3       |      async
-timeout==5.0.1
        attrs==24.3.0         |         autogluon==1.2        |    autogluon.common==1.2      |      autogluon.core==1.2      |    autoglu
on.features==1.2
  autogluon.multimodal==1.2   |     autogluon.tabular==1.2    |   autogluon.timeseries==1.2   |     beautifulsoup4==4.12.3    |          b
lis==0.7.11
        boto3==1.35.87        |       botocore==1.35.87       |       cachetools==5.5.0       |       catalogue==2.0.10       |        cat
boost==1.2.7
     certifi==2024.12.14      |   charset-normalizer==3.4.0   |          click==8.1.8         |      cloudpathlib==0.20.0     |       clou
dpickle==3.1.0
       colorama==0.4.6        |        colorful==0.5.6        |       colorlog==6.9.0         |          comm==0.2.2          |       conf
ection==0.1.5
       contourpy==1.3.1       |      coreforecast==0.0.12     |         cycler==0.12.1        |        cymem==2.0.10          |        dat
asets==3.2.0
       debugpy==1.8.11        |        decorator==5.1.1       |       defusedxml==0.7.1       |          dill==0.3.8          |         di
stlib==0.3.9
        einops==0.8.0         |        evaluate==0.4.3        |    exceptiongroup==1.2.2      |        executing==2.1.0       |         fa
stai==2.7.18
       fastcore==1.7.27       |      fastdownload==0.0.7      |      fastprogress==1.0.3      |        filelock==3.16.1       |      flatb
uffers==24.3.25
      fonttools==4.55.3       |       frozenlist==1.5.0       |           fs==2.4.16          |       fsspec==2024.9.0        |          f
ugue==0.9.1
        future==1.0.0         |          gast==0.6.0          |         gdown==5.2.0          |       gluonts==0.16.0         |    google-
api-core==2.24.0
     google-auth==2.37.0      |      google-pasta==0.2.0      | googleapis-common-protos==1.66.0 |        graphviz==0.20.3       |         g
reenlet==3.1.1
        grpcio==1.68.1        |          h5py==3.12.1         |    huggingface-hub==0.27.0    |        hyperopt==0.2.7        |
idna==3.10
       imageio==2.36.1        |   importlib-metadata==8.5.0   |       ipykernel==6.29.5       |        ipython==8.31.0        |       ipyw
idgets==8.1.5
         jedi==0.19.2         |        jinja2==3.1.5          |       jmespath==1.0.1         |         joblib==1.4.2         |       json
schema==4.21.1
jsonschema-specifications==2024.10.1 |    jupyter-client==8.6.3      |      jupyter-core==5.7.2      |   jupyterlab-widgets==3.0.13   |
keras==3.7.0
      kiwisolver==1.4.7       |        langcodes==3.5.0       |     language-data==1.3.0      |        lazy-loader==0.4       |        lib
clang==18.1.1
       lightgbm==4.5.0        |     lightning==2.5.0.post0    |  lightning-utilities==0.11.9  |     linkify-it-py==2.0.3      |        llv
mlite==0.43.0
      marisa-trie==1.2.1      |     markdown-it-py==3.0.0     |       matplotlib==3.10.0      |    matplotlib-inline==0.1.7   |     mdit-p
y-plugins==0.4.2
         mdurl==0.1.2         |        memray==1.15.0         |       ml-dtypes==0.4.1        |       mlforecast==0.13.4      |      model
-index==0.1.11
        mpmath==1.3.0         |        msgpack==1.1.0         |       multidict==6.1.0        |    multiprocess==0.70.16      |       murm
urhash==1.0.11
         namex==0.0.8         |     nest-asyncio==1.6.0       |       networkx==3.4.2         |        nlpaug==1.1.11         |         n
ltk==3.8.1
        numba==0.60.0         |        numpy==1.26.4          | nvidia-cublas-cu12==12.4.5.8  | nvidia-cuda-cupti-cu12==12.4.127 | nvidia-c
uda-nvrtc-cu12==12.4.127
nvidia-cuda-runtime-cu12==12.4.127 |  nvidia-cudnn-cu12==9.1.0.70   |  nvidia-cufft-cu12==11.2.1.3   | nvidia-curand-cu12==10.3.5.147 | nvidia
-cusolver-cu12==11.6.1.9
nvidia-cusparse-cu12==12.3.1.170 |     nvidia-ml-py3==7.352.0     |    nvidia-nccl-cu12==2.21.5    | nvidia-nvjitlink-cu12==12.4.127 |   nvidi
a-nvtx-cu12==12.4.127
       omegaconf==2.2.3       |       opencensus==0.11.4      |   opencensus-context==0.1.3   |      opendatalab==0.0.10      |         op
enmim==0.3.9
       openxlab==0.0.11       |       opt-einsum==3.4.0       |        optree==0.13.1         |         optuna==4.1.0         |       orde
red-set==4.1.0
       orjson==3.10.12        |       packaging==24.2         |         pandas==2.2.3         |          parso==0.8.4         |          p
atsy==1.0.1
      pdf2image==1.17.0       |         pexpect==4.9.0        |      pickleshare==0.7.5       |         pillow==11.0.0        |          p
ip==24.3.1
     platformdirs==4.3.6      |         plotly==5.24.1        |         preshed==3.0.9        |   prometheus-client==0.21.1   |     prompt
-toolkit==3.0.48
       propcache==0.2.1       |       proto-plus==1.25.0      |        protobuf==5.29.2       |         psutil==6.1.0         |       ptyp
rocess==0.7.0
       pure-eval==0.2.3       |         py4j==0.10.9.8        |         py-spy==0.4.0         |        pyarrow==18.1.0        |         py
asn1==0.6.1
    pyasn1-modules==0.4.1     |      pycryptodome==3.21.0     |        pydantic==2.10.4       |     pydantic-core==2.27.2     |        pyg
ments==2.18.0
       pyparsing==3.2.0       |      pytesseract==0.3.10      |  python-dateutil==2.9.0.post0 | pytorch-lightning==2.5.0.post0 | pytorch-me
tric-learning==2.3.0
         pytz==2024.2         |        pyzmq==26.2.0          |         ray==2.39.0           |      referencing==0.35.1      |         reg
ex==2024.11.6
       requests==2.32.3       |         rich==13.9.4          |        rpds-py==0.22.3        |           rsa==4.9            |       s3tr
ansfer==0.10.4
      safetensors==0.4.5      |      scikit-image==0.24.0     |      scikit-learn==1.5.2      |         scipy==1.14.1         |        sea
born==0.13.2
     sentencepiece==0.2.0     |         seqeval==1.2.2        |       setuptools==75.6.0      |       shellingham==1.5.4      |          s
ix==1.17.0
      smart-open==7.1.0       |         soupsieve==2.6        |          spacy==3.7.5         |      spacy-legacy==3.0.12     |      spacy
-loggers==1.0.5
         srsly==2.5.0         |       stack-data==0.6.3       |      statsforecast==1.7.8     |      statsmodels==0.14.4      |         sy
mpy==1.13.1
       tabulate==0.9.0        |        tenacity==9.0.0        |      tensorboard==2.18.0      |    tensorboardX==2.6.2.2      | tensorboar
d-data-server==0.7.2
      tensorflow==2.18.0      | tensorflow-io-gcs-filesystem==0.37.1 |        termcolor==2.5.0       |      text-unidecode==1.3      |
textual==1.0.0
         thinc==8.2.5         |      threadpoolctl==3.5.0     |      tifffile==2024.12.12     |          timm==1.0.3          |       toke
nizers==0.21.0
```

```
        toolz==0.12.1          |          torch==2.5.1          |      torchmetrics==1.2.1       |      torchvision==0.20.1       |          to
rnado==6.4.2
         tqdm==4.67.1          |       traitlets==5.14.3        |      transformers==4.47.1      |          triad==0.9.8          |          tr
iton==3.1.0
        typer==0.15.1          |   typing-extensions==4.12.2    |         tzdata==2024.2         |       uc-micro-py==1.0.3       |          ur
llib3==2.3.0
     utilsforecast==0.2.4      |      virtualenv==20.28.0       |         wasabi==1.1.3          |        wcwidth==0.2.13         |          we
asel==0.4.1
       werkzeug==3.1.3         |         wheel==0.45.1          |   widgetsnbextension==4.0.13   |       window-ops==0.0.15       |          wr
apt==1.17.0
        xgboost==2.0.2         |         xxhash==3.5.0          |          yarl==1.18.3          |          zipp==3.21.0          |       auto
command==2.2.2
   backports.tarfile==1.2.0    |         inflect==7.3.1         |   jaraco.collections==5.1.0    |    jaraco.context==5.3.0       |    jaraco.
functools==4.0.1
     jaraco.text==3.12.1       |     more-itertools==10.3.0     |          tomli==2.0.1          |        typeguard==4.3.0
/tmp/ipykernel_8762/3285228674.py:1: DeprecationWarning: pkg_resources is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_r
esources.html
  import pkg_resources
```

Aufgabe PT1: Möglichst einfache Erstellung eines guten Benchmarkmodells [Lernziel 5.1; 4 Punkte]

Basierend auf dem Notebook CSN (Part A) sind folgende Anpassungen durchzuführen:

- Die Zellen vor dem Text "Part A: Quick & Easy" mit Bezug zum ursprünglichen Thema Credit Scoring sind zu entfernen.
- In "Part A: Quick & Easy":
  - Alle im Notebook verwendeten Pakete sind in Sektion 1.1 zu ergänzen, nicht benötigte Imports sind zurückzubauen. Nutzen Sie einen `RANDOM_SEED` von 42.
  - Der in Sektion 1.2 verwendete Datensatz `../input/home-credit-default-risk/application_train.csv` ist durch den im R-Teil erzeugten Datensatz `diabetic_data_bin.csv` zu ersetzen. Welche Ergänzungen sind vorzunehmen, damit der Datensatz korrekt eingelesen wird?
  - Jedes eingelesene Feature, dessen Name auf „_id“ endet, ist auf den Datentyp „object“ zu setzen. Welchen Vorteil hat dieses Vorgehen?
  - Für Part A sind keine weiteren außer die im Notebook bereits vorhandenen Modifikationen am Datensatz durchzuführen. Falls einzelne Codeblöcke nicht benötigt werden, kommentieren Sie diese mit einer entsprechenden Anmerkung aus. Erinnerung: Beachten Sie hier und im restlichen Notebook den Allgemeinen Hinweis, dass Texte zu übersetzen und an den Datensatz anzupassen sind.

Lösungsvorschlag:

Part A: Schnell und einfach

1. Entwicklung eines Baseline-Modells.

In diesem allerersten Abschnitt werden wir schnell ein grundlegendes maschinelles Lernmodell für die binäre Klassifizierungsaufgabe der Vorhersage von Wiedereinweisungen bei Diabetes erstellen. Dieses Basismodell dient als Benchmark für alle nachfolgenden maschinellen Lernmodelle.

1.1 Einrichten der Modellierungsumgebung

Zunächst importieren wir alle erforderlichen Bibliotheken, kalibrieren die Visualisierungsparameter und setzen einen Zufallswert, um die Reproduzierbarkeit unserer Ergebnisse zu gewährleisten.

In [4]:
```python
import os
# Set environment variable to ignore all warnings
os.environ['PYTHONWARNINGS'] = 'ignore'

# Standard Python libraries for data analysis, scientific computing, and plotting
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
import re
from scipy.stats import uniform, loguniform

# Setting display options for pandas
pd.set_option('display.max_columns', None)
COLOR_LIGHT, COLOR_DARK = '#849CBE', '#00548A'

# Scikit-Learn modules for preprocessing and machine learning models
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import roc_auc_score

# Gradient boosting frameworks and logistic regression
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
import statsmodels.formula.api as smf

# Tensorflow, Keras
import tensorflow as tf
from keras import Input
from keras import Model
from keras import regularizers
```

```
from keras.api.layers import Embedding, Flatten, concatenate, Dense
from keras.src.utils import set_random_seed

# Ensure reproducibility
RANDOM_SEED = 42
set_random_seed(RANDOM_SEED)
```

```
2025-01-14 16:34:13.270422: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical resu
lts due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN
_OPTS=0`.
2025-01-14 16:34:13.471602: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register cuFFT factory: Attempting to reg
ister factory for plugin cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1736872453.532050    8762 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when
one has already been registered
E0000 00:00:1736872453.546478    8762 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS w
hen one has already been registered
2025-01-14 16:34:13.725524: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU inst
ructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI AVX512_BF16 AVX512_FP16 AVX_VNNI AMX_TILE AMX_INT8 AMX_BF16 FMA, in other opera
tions, rebuild TensorFlow with the appropriate compiler flags.
```

In [5]:
```
# Ergänzung: Zeitmessung Notebook
start_time_nb = time.time()
```

Um die Ressourcennutzung zu optimieren, passen wir die GPU-Speichereinstellungen von TensorFlow an und ermöglichen so eine effiziente Berechnung während unseres gesamten Modellierungsworkflows.

In [6]:
```
# Prevent TensorFlow From Fully Allocating GPU Memory
# Ref: https://www.tensorflow.org/guide/gpu#limiting_gpu_memory_growth
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
    # Currently, memory growth needs to be the same across GPU
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        logical_gpus = tf.config.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
    except RuntimeError as e:
        # Memory growth must be set before GPUs have been initialized
        print(e)
else:
    print("No GPU(s) found")
```

```
No GPU(s) found
```

```
2025-01-14 16:34:16.016609: E external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:152] failed call to cuInit: INTERNAL: CUDA error: Fai
led call to cuInit: UNKNOWN ERROR (303)
```

1.2 Laden und Prüfen der Eingabedaten

Als nächstes wird der erwähnte Datensatz geladen, und wir zeigen seine Dimensionen an, indem wir die Anzahl der Zeilen und Spalten ausgeben. Wir zeigen auch einige zufällig ausgewählte Zeilen, um den Inhalt und die Datenstruktur zu veranschaulichen.

Ergänzung: Beim Einlesen ist darauf zu achten, dass die im Datensatz vorhandenen "None" Werte nicht als Missings interpretiert werden (`na_values=[""], keep_default_na=False`).

In [7]:
```
# Load the dataset into a pandas DataFrame
# df_raw = pd.read_csv("diabetic_data_bin.csv", na_values=[""], keep_default_na=False)
df_raw = pd.read_csv("./PK_VADS_IMMERSION/2025/1_Erster_Ansatz/diabetic_data_bin.csv", na_values=[""], keep_default_na=False)

# Display the dimensions of the dataset (rows, columns)
print(f"Input Dataset dimensions (rows, columns): {df_raw.shape}")

# Display a random sample of 10 rows from the dataset to inspect data
df_raw.sample(n=7, random_state=RANDOM_SEED)
```

```
Input Dataset dimensions (rows, columns): (47751, 51)
```

Out[7]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_speci
47523	Asian	Female	[50-60)	?	3	6	1	1	?	Orthope
8582	Caucasian	Male	[40-50)	?	2	6	4	8	?	
42024	Caucasian	Female	[70-80)	?	1	3	7	3	MC	
15219	AfricanAmerican	Female	[20-30)	?	1	1	7	2	?	InternalMedi
45649	Caucasian	Female	[60-70)	?	1	2	7	1	?	
23448	Caucasian	Male	[80-90)	?	3	28	1	1	SP	
39945	Caucasian	Male	[40-50)	?	1	1	7	4	?	

In [8]:
```
# Aktuelle Datentypen
pd.DataFrame(df_raw.dtypes).T
```

Out[8]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_specialty	num_la
0	object	object	object	object	int64	int64	int64	int64	object	object	

In [9]:
```
# Step 1: Set the type of columns ending with "_id" to "object"
df_raw = df_raw.astype({col: 'object' for col in df_raw.columns if col.endswith('_id')})
```

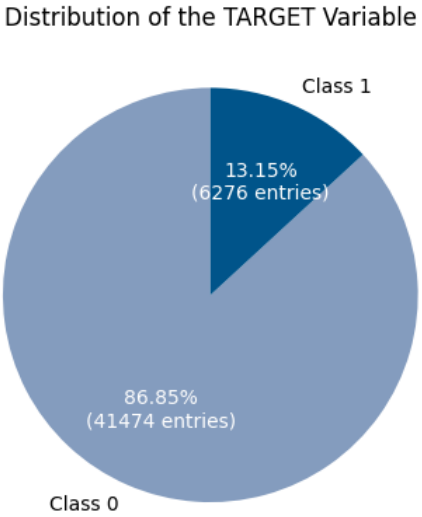
In [10]:
```
# Kontrolle der Datentypen
pd.DataFrame(df_raw.dtypes).T
```

Out[10]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_specialty	num_la
0	object	object	object	object	object	object	object	int64	object	object	

Die betroffenen Features (z.B. `discharge_disposition_id`) enthalten zwar numerische Werte, sind jedoch kategorielle Datentypen. Das wird mit dem Datentyp `object` sichtbar gemacht.

In [11]:
```
# Count the occurrences of each unique value in the 'TARGET' column
target_counts = df_raw['TARGET'].value_counts()

# Calculate the total number of entries
total_entries = len(df_raw['TARGET'])

# Define a custom formatting function for the pie chart labels
def custom_autopct(pct):
    absolute = int(pct/100.*total_entries)
    return "{:.2f}%\n({:d} entries)".format(pct, absolute)

# Create a pie chart to visualize the distribution of classes in the 'TARGET' column
pie_chart, _, autotexts = plt.pie(target_counts, labels=target_counts.index.map({0: 'Class 0', 1: 'Class 1'}),
                                  autopct=custom_autopct, startangle=90,
                                  colors=[COLOR_LIGHT, COLOR_DARK], wedgeprops={'edgecolor': 'none'})

# Set the color of the autopct texts to white
for autotext in autotexts:
    autotext.set_color('white')

# Plot the pie chart
plt.title('Distribution of the TARGET Variable')
plt.show()
```



Das Kreisdiagramm zeigt ein deutliches Ungleichgewicht in der Verteilung der abhängigen Variable 'TARGET'. Die Klasse '1' erscheint deutlich seltener als Klasse '0'. In diesen Fällen bietet sich die Accuracy als Kennzahl nicht an, wohl aber die Fläche unterhalb der ROC (Receiver Operating Characteristic) Kurve.

1.3 Ersetzen fehlender kategorialer Werte

Während viele Tools für maschinelles Lernen (ML) die in Teil B beschriebenen komplexeren Datenvorbereitungsschritte erfordern, ist für CatBoost, das ML-Tool unserer Wahl, nur die unten beschriebene minimale Datenvorbereitung erforderlich, indem fehlende kategoriale Werte durch einen vordefinierten Wert ersetzt werden.

In [12]:
```
# Identify categorical features by data type 'object'
categorical_features = list(df_raw.select_dtypes(include=['object']).columns)

# Compute the number of unique values for each categorical feature and reset the index to make 'Feature Names' a column
categorical_feature_counts = df_raw[categorical_features].nunique().reset_index()

# Set the column names to 'Feature' and 'Number of Unique Values'
categorical_feature_counts.columns = ['Categorical Feature', 'Number of Unique Values']
```

```
# Sort the DataFrame by 'Number of Unique Values' in descending order
categorical_feature_counts.sort_values(by='Number of Unique Values', ascending=False, inplace=True)

# Display the resulting DataFrame
display(categorical_feature_counts.style.hide(axis='index'))
```

Categorical Feature	Number of Unique Values
diag_3	724
diag_2	678
diag_1	661
medical_specialty	45
payer_code	17
discharge_disposition_id	15
admission_source_id	11
group_diag_1	10
group_diag_3	10
age	10
group_diag_2	10
weight	8
admission_type_id	7
race	6
rosiglitazone	4
metformin	4
max_glu_serum	4
A1Cresult	4
insulin	4
pioglitazone	4
glyburide	4
glimepiride	4
glipizide	4
repaglinide	4
acarbose	3
nateglinide	3
gender	2
diabetesMed	2
chlorpropamide	2
tolbutamide	2
change	2
miglitol	2
glyburide.metformin	2
tolazamide	2
acetohexamide	1
citoglipton	1
troglitazone	1
examide	1
glipizide.metformin	1
metformin.pioglitazone	1
glimepiride.pioglitazone	1
metformin.rosiglitazone	1

Als Nächstes ermitteln wir die kategorialen Merkmale, die fehlende Werte enthalten, und berechnen den Prozentsatz der fehlenden Werte.

In [13]:
```
# Define a function that returns a DataFrame with features containing missing values
# and the percentage of missing values for each of these features
def missing_values_percentage(data):
    # Calculate percentages and create a DataFrame
    nan_percentages = data.isna().sum() * 100 / len(data)
    missing_values_df = pd.DataFrame({
        'Categorical Feature': nan_percentages.index,
        'Missing Percent': nan_percentages.values
```

```
    })
    # Format the 'Missing Percent' column as a percentage string with two decimal places
    missing_values_df['Missing Percent'] = missing_values_df['Missing Percent'].apply(lambda x: f'{x:.2f}%')
    # Filter out features without missing values and sort by percentage
    missing_values_df = missing_values_df[missing_values_df['Missing Percent'] != '0.00%']
    missing_values_df.sort_values(by='Missing Percent', ascending=False, inplace=True)
    return missing_values_df

# Calculate the proportion of missing values for the categorical features
missing_values_df = missing_values_percentage(df_raw[categorical_features])

# Display the resulting DataFrame without the index using the new recommended Styler.hide method
missing_values_df_styled = missing_values_df.style.hide(axis='index')
display(missing_values_df_styled)
```

Categorical Feature	Missing Percent

Um fehlende Werte in kategorialen Spalten zu berücksichtigen, ohne Daten zu imputieren oder Datensätze zu löschen, ordnen wir Instanzen fehlender Daten einer eigenen Kategorie zu, `__missingValue__`. Diese Strategie ermöglicht es unseren Algorithmen für maschinelles Lernen, fehlende Werte als eigenständiges informatives Merkmal zu verarbeiten, wodurch die volle Breite des Datensatzes und die Integrität des Inputs für die Modellierung erhalten bleiben.

Ergänzung: Da der Datensatz keine fehlenden Werte enthält, wird die nächste Zelle nicht benötigt und auskommentiert.

In [14]:
```
# Replace missing values with a new category '__missingValue__' in all categorical columns
# df_raw[categorical_features] = df_raw[categorical_features].fillna('__missingValue__')
```

1.4 Aufteilung der Daten in Trainings-, Validierungs- und Testteilmengen

Wenn wir einen Klassifikator für maschinelles Lernen trainieren, wollen wir nicht nur, dass er lernt, die Trainingsdaten zu modellieren oder sie sich einfach zu merken (ML-Modelle können groß und komplex genug sein, um dies zu tun). Wir wollen, dass das Modell auf Daten verallgemeinert, die es noch nie gesehen hat. Daher wird die Leistung des Modells in der Regel an einer Testmenge gemessen, die aus Beispielen besteht, die noch nie zuvor gesehen wurden. Aus Gründen der Konsistenz mit künftigen Modelloptimierungsstrategien teilen wir unseren Datensatz nach dem Zufallsprinzip in drei Teile auf:

- Trainingsdaten: 70 % der Daten für das Modelltraining
- Validierungsdaten: 15 %, um unsere Modelle nach dem Training zu validieren und möglicherweise über Änderungen zu entscheiden
- Testdaten: 15% für die endgültige Messung der Verallgemeinerungsleistung aller relevanten Modelle

Diese Aufteilung ermöglicht ein effektives Modelltraining, die Abstimmung der Hyperparameter, um eine Überanpassung zu verhindern, und eine unvoreingenommene Bewertung der Leistung bei neuen, ungesehenen Daten.

In [15]:
```
# Constants for data split ratios
TRAIN_RATIO = 0.7
VAL_TEST_RATIO = 0.5  # Splitting the remaining 30% equally into validation and test

# Separate the features (X) and the target (y)
X_raw = df_raw.drop(columns=['TARGET'], axis=1)
y = df_raw['TARGET']

# Split the dataset into a training set and a combined validation and test set
X_raw_train, X_raw_val_test, y_train, y_val_test = train_test_split(X_raw, y, train_size=TRAIN_RATIO, random_state=RANDOM_SEED)

# Further split the combined validation and test set into separate validation and test sets
X_raw_val, X_raw_test, y_val, y_test = train_test_split(X_raw_val_test, y_val_test, test_size=VAL_TEST_RATIO, random_state=RANDOM_SEED)

# Verify the dimensions of the training, validation, and test sets by displaying (rows, columns)
print(f"Training set dimensions (rows, columns):   {X_raw_train.shape}")
print(f"Validation set dimensions (rows, columns):  {X_raw_val.shape}")
print(f"Test set dimensions (rows, columns):        {X_raw_test.shape}")
```

```
Training set dimensions (rows, columns):   (33425, 50)
Validation set dimensions (rows, columns):  (7163, 50)
Test set dimensions (rows, columns):        (7163, 50)
```

Überprüfen Sie für die drei Stichproben die Mittelwerte und Standardabweichungen der Variablen TARGET `y`:

In [16]:
```
# Calculate TARGET mean and std for each set
stats = pd.DataFrame({
    'Mean': [y_train.mean(), y_val.mean(), y_test.mean()],
    'Standard Deviation': [y_train.std(), y_val.std(), y_test.std()]
}, index=['Train', 'Validation', 'Test']).round(4)

print(stats)
```

```
             Mean  Standard Deviation
Train       0.1314              0.3378
Validation  0.1312              0.3377
Test        0.1319              0.3384
```

Diese zufällig erzeugten Unterschiede in den Mittelwerten der TARGET-Variablen erscheinen zunächst gering, können sich aber auf die endgültige Auswertung auswirken.

1.5 Training eines Standard-CatBoost-Klassifikators (baseline model)

In [17]:
```
# Einschub: Daten für AutoGluon (PT7) vorbereiten
Xy_raw_train = X_raw_train.copy()
Xy_raw_val = X_raw_val.copy()
Xy_raw_test = X_raw_test.copy()
```

```
Xy_raw_train['TARGET'] = y_train
Xy_raw_val['TARGET'] = y_val
Xy_raw_test['TARGET'] = y_test
```

Wir fahren fort, indem wir das CatBoost-Modell auf unseren vorbereiteten Datensatz anwenden, wobei wir die Standardeinstellungen für eine unkomplizierte Bewertung ohne Hyperparameter-Tuning verwenden.

In [18]:
```
# Start timer to calculate the running time of training the CatBoost model
start_time = time.time()

# Fit the CatBoostClassifier on the training data
CB1 = CatBoostClassifier(eval_metric='AUC', random_seed=RANDOM_SEED)
CB1.fit(X_raw_train, y_train, cat_features=categorical_features, logging_level='Silent')

# Calculate and print the running time in seconds
# Einschub: Für die spätere Anwendung von autogluon wird die Zeit von CatBoost gespeichert.
elapsed_time_CB1 = time.time() - start_time
print(f"Elapsed time (sec): {elapsed_time_CB1:.2f}")
```

```
Elapsed time (sec): 65.59
```

Um zu zeigen, wie umfangreich die Anpassungsmöglichkeiten für die Verfeinerung des CatBoost-Modells sind, stellen wir eine vollständige Liste der Hyperparameter und ihrer Standardeinstellungen vor.

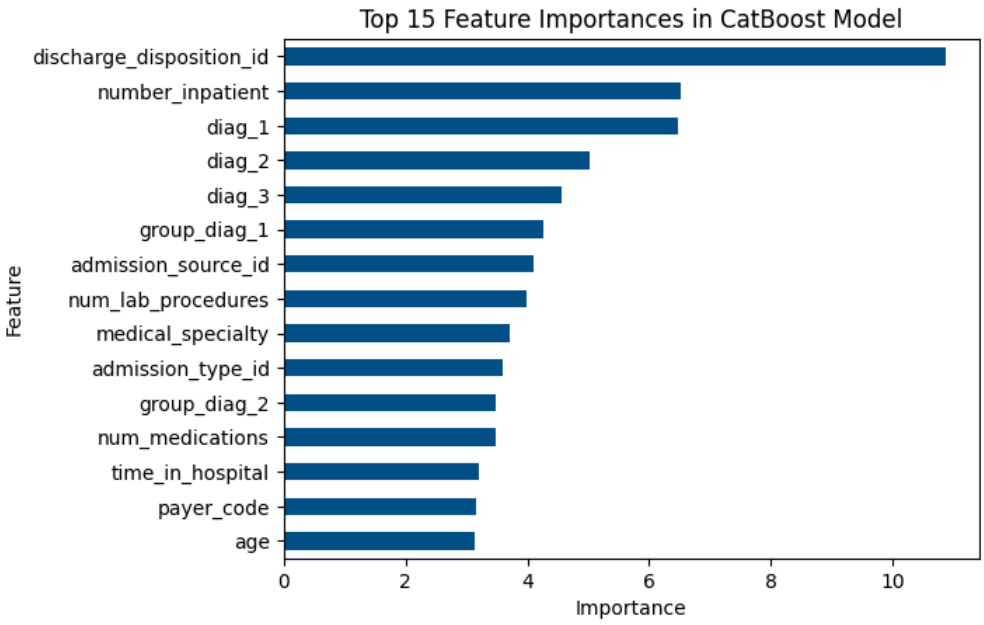
In [19]:
```
# Display all model hyperparameters in a DataFrame
hyperparams_list = [(k, v) for k, v in CB1.get_all_params().items()]
hyperparams_df = pd.DataFrame(hyperparams_list, columns=['Hyperparameter', 'Value'])
display(hyperparams_df.style.hide(axis='index'))
```

Hyperparameter	Value
nan_mode	Min
eval_metric	AUC
combinations_ctr	['Borders:CtrBorderCount=15:CtrBorderType=Uniform:TargetBorderCount=1:TargetBorderType=MinEntropy:Prior=0/1:Prior=0.5/1:Prior=1/1', 'Counter:CtrBorderCount=15:CtrBorderType=Uniform:Prior=0/1']
iterations	1000
sampling_frequency	PerTree
fold_permutation_block	0
leaf_estimation_method	Newton
random_score_type	NormalWithModelSizeDecrease
counter_calc_method	SkipTest
grow_policy	SymmetricTree
penalties_coefficient	1
boosting_type	Plain
model_shrink_mode	Constant
feature_border_type	GreedyLogSum
ctr_leaf_count_limit	18446744073709551615
bayesian_matrix_reg	0.100000
one_hot_max_size	2
eval_fraction	0
force_unit_auto_pair_weights	False
l2_leaf_reg	3
random_strength	1
rsm	1
boost_from_average	False
max_ctr_complexity	4
model_size_reg	0.500000
simple_ctr	['Borders:CtrBorderCount=15:CtrBorderType=Uniform:TargetBorderCount=1:TargetBorderType=MinEntropy:Prior=0/1:Prior=0.5/1:Prior=1/1', 'Counter:CtrBorderCount=15:CtrBorderType=Uniform:Prior=0/1']
pool_metainfo_options	{'tags': {}}
subsample	0.800000
use_best_model	False
class_names	[0, 1]
random_seed	42
depth	6
ctr_target_border_count	1
posterior_sampling	False
has_time	False
store_all_simple_ctr	False
border_count	254
classes_count	0
auto_class_weights	None
sparse_features_conflict_fraction	0
leaf_estimation_backtracking	AnyImprovement
best_model_min_trees	1
model_shrink_rate	0
min_data_in_leaf	1
loss_function	Logloss
learning_rate	0.046101
score_function	Cosine
task_type	CPU
leaf_estimation_iterations	10
bootstrap_type	MVS
max_leaves	64

Hyperparameter	Value
permutation_count	4

Um erste Einblicke in das Verhalten des trainierten Modells zu gewinnen, visualisieren wir die eingebauten Merkmalsimporte, die die einflussreichsten Merkmale für die Vorhersagen hervorheben.

In [20]:
```
# Extract the 15 most important features and create a Series for plotting
feature_importances = pd.Series(CB1.feature_importances_, index=X_raw_train.columns).nlargest(15)

# Plot the feature importances
feature_importances.plot(kind='barh', color=COLOR_DARK)
plt.gca().invert_yaxis()
plt.title('Top 15 Feature Importances in CatBoost Model')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



Ergänzung:

Die obige Darstellung des Feature Importance Plots zeigt, dass die einflussreichsten Prädiktoren `discharge_disposition_id` sowie `number_impatient` sind. Dann folgen die drei Diagnoseinformationen `diag_1`, `diag_2` und `diag_3`. `discharge_disposition_id` enthält Informationen darüber, wohin die Patienten entlassen werden. Beispielsweise bedeuten die IDs 1, 6 und 8 Entlassung nach Hause (sowie die Art der dortigen Versorgung). Das zweitwichtigste Merkmal ist `number_inpatient`, d. h. die Anzahl der stationären Besuche im letzten Jahr.

Als nächstes wird die Leistung unseres CatBoost-Modells anhand der Validierungsmenge zu bewerten, indem wir die Fläche unter der Receiver Operating Characteristic Curve (AUC) als unsere gewählte Bewertungsmetrik verwenden (siehe oben).

In [21]:
```
# Calculate the AUC score using the model's prediction probabilities for the positive class
auc_CB1 = roc_auc_score(y_val, CB1.predict_proba(X_raw_val)[:, 1])

# Print the validation AUC
print(f'The validation AUC of the CatBoost model with standard hyperparameters is: {auc_CB1:.6f}')
```

```
The validation AUC of the CatBoost model with standard hyperparameters is: 0.707710
```

Zusammenfassend lässt sich sagen, dass Teil A eine solide Grundlage für unsere binäre Klassifizierungsaufgabe unter Verwendung eines CatBoost-Modells mit minimaler Datenvorbereitung und ohne Anpassung der Hyperparameter geschaffen hat. Dies ist bereits ein komplexes Modell, aber es ist sehr einfach zu erstellen und für seine hohe Vorhersagekraft mit Standardparametern bekannt. Dieser Benchmark dient als Referenzpunkt, um den Wert der in den folgenden Abschnitten erstellten komplexeren Modelle zu bewerten.

Aufgabe PT2: Logistische Regression und Merkmalsanalyse [Lernziel 5.1 / 2.2; 10 Punkte]

Basierend auf dem Notebook CSN (Part B) sind folgende Anpassungen durchzuführen:

- a) In Abschnitt 2.1 sind nur notwendige Operationen durchzuführen. Was kann (begründet) auskommentiert werden?
- b) Für die logistische Regression in Abschnitt 2.2 ist folgendes Vorgehen zu wählen: Orientieren Sie sich an der Feature-Importance-Liste von CatBoost, und konstruieren Sie ein Modell mit mindestens fünf Features. Beachten Sie dabei die Merkmalsskala. Wo ergeben sich Probleme und wie sind diese zu begründen?
- c) Aus Abschnitt 2.3 sind nur relevante Bestandteile zu bearbeiten. Was kann (begründet) auskommentiert werden?
- d) Die Analyse der kategoriellen Merkmale in Abschnitt 3.1 ist auf `group_diag_1`, `group_diag_2`, `group_diag_3`, `age`, `medical_specialty`, `payer_code` und `discharge_disposition_id` zu beschränken.
- e) Bei der Analyse der numerischen Merkmale in Abschnitt 3.2 sind `number_outpatient`, `number_inpatient`, `num_procedures`, `num_lab_procedures`, `num_medications`, `time_in_hospital`, `number_emergency` sowie `number_diagnoses` zu berücksichtigen.

- Abschnitt 3.3 sowie 4 (Enhancing CatBoost's Performance with New Features) werden an dieser Stelle nicht berücksichtigt.
- Abschnitt 5 wird nicht berücksichtigt (folgt allerdings später im Rahmen von PT7).

- f) Abschnitt 6 ist auszuführen:
  - In Abschnitt 6.3 ist der `non_event_factor` begründet anzupassen.
  - Im Anschluss an Abschnitt 6.4 ist ein neuer Abschnitt 6.5 einzufügen, in dem die bisherigen Ergebnisse der Modelle gegenübergestellt werden sollen. Orientieren Sie sich dabei an Abschnitt 4.2 des CSN Notebooks.

Lösungsvorschlag:

Part B: Gewinnung von Erkenntnissen aus Daten und Modellierung

Teil B konzentriert sich auf die Verfeinerung unserer Strategie des maschinellen Lernens, um wertvolle Erkenntnisse zu gewinnen. Wir beginnen mit einer gründlichen Untersuchung der logistischen Regression, einem Eckpfeiler der versicherungsmathematischen Analytik, in ihrer klassischen, unregulierten Form. Anschließend gehen wir zu einer gezielten explorativen Datenanalyse und zum Feature Engineering über, um die Vorhersagekraft unserer Modelle zu verbessern. Ein weiterer Schwerpunkt liegt auf der Erklärbarkeit von Modellen. Um die Voraussetzungen für die fortgeschrittenen Modellierungstechniken in Teil C zu schaffen, schließen wir mit wichtigen Vorverarbeitungsaktivitäten ab: Kodierung, Skalierung und Subsampling.

2. Logistische Regression: Ein klassischer Ansatz zur Klassifizierung

In diesem Abschnitt befassen wir uns mit der logistischen Regression, einer klassischen und grundlegenden Technik für binäre Klassifizierungsaufgaben. Da diese Methode nicht mit fehlenden Merkmalswerten umgehen kann, beginnen wir mit der Imputation fehlender numerischer Werte durch Medianwerte. Anschließend implementieren und evaluieren wir ein klassisches logistisches Regressionsmodell mit niedrigen Parametern. Anschließend passen wir das CatBoost-Modell auf der Grundlage der modifizierten Datensätze an, vergleichen die Leistung des neuen Modells mit der des Benchmark-Modells und interpretieren das Ergebnis.

2.1 Aufbereitung der Daten: Imputation fehlender Zahlen mit Median

Ergänzung: Wie oben gezeigt sind bei den numerischen Variablen keine Missings vorhanden. Der entsprechende Abschnitt wird daher auskommentiert.

In diesem Unterabschnitt gehen wir die Herausforderung fehlender Daten an, indem wir diese fehlenden Werte imputieren. Dies ist ein wesentlicher Schritt, da die logistische Regression vollständige Datensätze benötigt, um genaue Vorhersagen und Modellparameter zu berechnen. Fehlende Werte für kategoriale Merkmale mit einer eindeutigen Kategorie haben wir bereits in Unterabschnitt 1.3 imputiert. Bei numerischen Merkmalen wählen wir den Median, da er robust gegenüber Ausreißern ist und die Verteilung von schiefen Datensätzen effektiv bewahrt. Obwohl die Median-Imputation die Datenvariabilität verringern und möglicherweise zu Verzerrungen führen kann, wenn die fehlenden Daten nicht zufällig sind, bietet ihre Einfachheit einen praktischen Ansatz für die Erstellung eines Basismodells bei binären Klassifizierungsaufgaben. Der folgende Codeausschnitt zeigt, wie wir fehlende Werte in numerischen Merkmalen unter Verwendung des aus den Trainingsdaten berechneten Medians imputieren, um Datenverluste zu vermeiden.

In [22]:
```
# Identify numerical features
numerical_features = list(X_raw_train.select_dtypes(exclude=['object']).columns)

# Copy the Input Data
df_pre = df_raw.copy()

num_missing = df_pre.isnull().sum().sum()
print(f"\nNumber of missing values before imputation: {num_missing}")

# Replace missing values for numeric features by median value from the training data
#for col in numerical_features:
#    median_value = X_raw_train[col].median()
#    df_pre[col] = df_pre[col].fillna(median_value)

# Check for remaining missing values:
# num_missing = df_pre.isnull().sum().sum()
# print(f"\nNumber of missing values after imputation: {num_missing}")
```

```
Number of missing values before imputation: 0
```

Während unser CatBoost-Modell keine Probleme mit sehr seltenen Merkmalswerten hatte, kann dies zu Problemen bei der kommenden logistischen Regression führen. Daher ersetzen wir die extrem seltenen Werte mit unbekanntem Geschlecht (Unknown/Invalid) durch den Wert „Female“.

Ergänzung: Eine Ersetzung wurde bereits im R-Notebook vorgenommen, daher wird der entsprechende Teil auskommentiert.

In [23]:
```
# print("\nGender_Frequency: ", df_pre.CODE_GENDER.value_counts())
# df_pre.loc[df_pre["CODE_GENDER"] != "M", "CODE_GENDER"] = "F"
```

Wie bereits in Unterabschnitt 1.4 beschrieben, werden wir unseren vorverarbeiteten Datensatz entsprechend in separate Trainings-, Validierungs- und Testdatensätze aufteilen. Dieser entscheidende Schritt, der für das Training, die Feinabstimmung und die Evaluierung unserer maschinellen Lernmodelle unerlässlich ist, stellt sicher, dass unser Modell an Daten validiert und getestet werden kann, die es während des Trainingsprozesses nicht gesehen hat.

In [24]:
```
# Constants for data split ratios
TRAIN_RATIO = 0.7
VAL_TEST_RATIO = 0.5  # Splitting the remaining 30% equally into validation and test

# Separate the features (X) and the target (y) after preprocessing
X_pre = df_pre.drop(['TARGET'], axis=1)
y_pre = df_pre['TARGET']
```

```
# Split the preprocessed dataset into a training set and a combined validation and test set
X_pre_train, X_pre_val_test, y_pre_train, y_pre_val_test = train_test_split(X_pre, y_pre, train_size=TRAIN_RATIO, random_state=RANDOM_SEED)

# Further split the combined validation and test set into separate validation and test sets
X_pre_val, X_pre_test, y_pre_val, y_pre_test = train_test_split(X_pre_val_test, y_pre_val_test, test_size=VAL_TEST_RATIO, random_state=RANDOM

# Verify the dimensions of the training, validation, and test sets by displaying (rows, columns)
print(f"Preprocessed training set dimensions (rows, columns):   {X_pre_train.shape}")
print(f"Preprocessed validation set dimensions (rows, columns): {X_pre_val.shape}")
print(f"Preprocessed test set dimensions (rows, columns):       {X_pre_test.shape}")
```

```
Preprocessed training set dimensions (rows, columns):   (33425, 50)
Preprocessed validation set dimensions (rows, columns): (7163, 50)
Preprocessed test set dimensions (rows, columns):       (7163, 50)
```

2.2 Implementierung der logistischen Regression: Fitting und Leistungsbewertung

Zur Vorbereitung einer logistischen Regressionsanalyse, die den formelbasierten Modellierungsstil von R widerspiegelt, werden die Merkmale und die Zielvariable in einem einzigen Datenrahmen zusammengefasst. Dieses Setup erleichtert die Verwendung der Formel-API von statsmodels, um das logistische Regressionsmodell kurz und bündig zu definieren und anzupassen. Mit Blick auf potenzielle Überanpassung und Konvergenzprobleme wählen wir nach und nach Merkmale aus der ersten CatBoost Liste aus. Dabei kommt es bereits zu Konvergenzproblemen, falls versucht wird "diag_1" - "diag_3" in die Liste mit aufzunehmen. Grund hierfür sind die bei diesen Merkmalen bestehenden vielen Klassen, die teilweise nur schwach besetzt sind. Diese werde daher für das weitere Vorgehen ignoriert. Ein ähnliches Problem entsteht, falls versucht wird "medical_specialty" zu verwenden (siehe Auswertung zur Anzahl der Ausprägungen in Teil 1). Daher wird an diesem Punkt mit der weiteren Aufnahme von Variablen gestoppt. Insgesamt beinhaltet das Modell damit **fünf** Variablen. Nachdem das Modell angepasst wurde, geben wir einen umfassenden Überblick über die Ergebnisse der logistischen Regression, um die Interpretation und Analyse zu erleichtern.

In [25]:
```
# Combine features and target variable into a single dataframe for formula-based modeling
Xy_pre_train = X_pre_train.copy()
Xy_pre_train['TARGET'] = y_pre_train

# Start timer to calculate the running time of training the logistic regression model
start_time = time.time()

#Define the logistic regression model using statsmodels' formula API and fit it to the training data
# LR1 = smf.logit(formula="TARGET ~  C(discharge_disposition_id) + C(admission_type_id ) + number_inpatient + num_medications + time_in_hospi
LR1 = smf.logit(formula="TARGET ~  C(discharge_disposition_id) + number_inpatient + C(admission_source_id ) + num_lab_procedures + C(admissio
# Calculate and print the running time in seconds
elapsed_time_LR1 = time.time() - start_time
print(f"Elapsed time (sec): {elapsed_time_LR1:.2f}")

# Display a summary of the logistic regression model results
print(LR1.summary())
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.367773
         Iterations: 100
Elapsed time (sec): 1.41
                           Logit Regression Results
==============================================================================
Dep. Variable:                 TARGET   No. Observations:                33425
Model:                          Logit   Df Residuals:                    33392
Method:                           MLE   Df Model:                           32
Date:                Tue, 14 Jan 2025   Pseudo R-squ.:                0.05466
Time:                        16:35:24   Log-Likelihood:                -12293.
converged:                      False   LL-Null:                       -13004.
Covariance Type:            nonrobust   LLR p-value:                9.991e-279
=================================================================================================================
                                    coef    std err          z      P>|z|      [0.025      0.975]
-----------------------------------------------------------------------------------------------------------------
Intercept                        -2.3953      0.075    -31.915      0.000      -2.542      -2.248
C(discharge_disposition_id)[T.2]  0.6584      0.096      6.878      0.000       0.471       0.846
C(discharge_disposition_id)[T.3]  0.6928      0.047     14.721      0.000       0.601       0.785
C(discharge_disposition_id)[T.4]  0.6174      0.170      3.634      0.000       0.284       0.950
C(discharge_disposition_id)[T.5]  1.2349      0.110     11.215      0.000       1.019       1.451
C(discharge_disposition_id)[T.6]  0.3661      0.053      6.854      0.000       0.261       0.471
C(discharge_disposition_id)[T.7]  0.0611      0.225      0.271      0.786      -0.380       0.502
C(discharge_disposition_id)[T.8]  0.4333      0.487      0.890      0.373      -0.521       1.388
C(discharge_disposition_id)[T.15] 2.8934      0.499      5.800      0.000       1.916       3.871
C(discharge_disposition_id)[T.18] 0.3374      0.087      3.869      0.000       0.166       0.508
C(discharge_disposition_id)[T.22] 1.5649      0.080     19.558      0.000       1.408       1.722
C(discharge_disposition_id)[T.23] -1.0820     0.459     -2.359      0.018      -1.981      -0.183
C(discharge_disposition_id)[T.24] -0.4830     1.038     -0.465      0.642      -2.517       1.551
C(discharge_disposition_id)[T.25] -0.2688     0.193     -1.392      0.164      -0.647       0.110
C(discharge_disposition_id)[T.28] 1.9307      0.288      6.699      0.000       1.366       2.496
C(admission_source_id)[T.2]      -0.2535      0.159     -1.590      0.112      -0.566       0.059
C(admission_source_id)[T.3]       0.6065      0.301      2.018      0.044       0.017       1.196
C(admission_source_id)[T.4]      -0.3454      0.096     -3.601      0.000      -0.533      -0.157
C(admission_source_id)[T.5]      -0.5875      0.207     -2.840      0.005      -0.993      -0.182
C(admission_source_id)[T.6]      -0.3602      0.123     -2.922      0.003      -0.602      -0.119
C(admission_source_id)[T.7]      -0.0090      0.061     -0.147      0.883      -0.129       0.111
C(admission_source_id)[T.8]      -0.3973      1.090     -0.364      0.716      -2.535       1.740
C(admission_source_id)[T.9]      -0.5037      0.473     -1.066      0.287      -1.430       0.423
C(admission_source_id)[T.17]     -0.2122      0.099     -2.154      0.031      -0.405      -0.019
C(admission_source_id)[T.20]      0.7764      0.307      2.525      0.012       0.174       1.379
C(admission_type_id)[T.2]         0.0498      0.060      0.836      0.403      -0.067       0.167
C(admission_type_id)[T.3]        -0.1179      0.070     -1.675      0.094      -0.256       0.020
C(admission_type_id)[T.5]         0.0928      0.107      0.863      0.388      -0.118       0.303
C(admission_type_id)[T.6]         0.3578      0.092      3.887      0.000       0.177       0.538
C(admission_type_id)[T.7]       -24.3879    1.2e+05     -0.000      1.000   -2.35e+05    2.35e+05
C(admission_type_id)[T.8]        -0.3404      0.289     -1.178      0.239      -0.907       0.226
number_inpatient                  0.5897      0.026     23.034      0.000       0.540       0.640
num_lab_procedures                0.0037      0.001      4.082      0.000       0.002       0.006
=================================================================================================================
/home/zeus/miniconda3/envs/cloudspace/lib/python3.10/site-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optim
ization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

Aus der Feature Importance Liste wurden fünf Variablen ausgewählt. Bei der Angabe der Merkmale (an R angelehnt) ist darauf zu achten, dass kategorielle Varialben mit dem Präfix `C` angegeben werden.

Da unser logistisches Regressionsmodell nun trainiert ist, bewerten wir seine Leistung auf dem Validierungsset durch Berechnung des AUC, der uns eine einzige Metrik liefert, die die Fähigkeit des Modells zur Unterscheidung zwischen den positiven und negativen Klassen zusammenfasst.

In [26]:
```
# Evaluate the performance of the logistic regression model on the validation data
# and calculate the Area Under the Curve (AUC) for the ROC
auc_LR1 = roc_auc_score(y_pre_val, LR1.predict(X_pre_val))

# Print the validation AUC
print(f'The validation AUC of the logistic regression model is: {auc_LR1:.6f}')
```

```
The validation AUC of the logistic regression model is: 0.661211
```

2.3 Data imputation impact: Neubewertung von CatBoost mit Medianersatz

Da ein Medianersatz nicht stattgefunden hat, wird dieser Abschnitt nicht berücksichtigt und auskommentiert.

In [27]:
```
# # Start timer to calculate the running time of training the CatBoost model with median replacement
# start_time = time.time()

# # Initialize a CatBoostClassifier and fit it to the preprocessed training data
# CB2 = CatBoostClassifier(eval_metric='AUC', random_seed=RANDOM_SEED)
# CB2.fit(X_pre_train, y_train, cat_features=categorical_features, logging_level='Silent')

# # Calculate and print the running time in seconds
# elapsed_time_CB2 = time.time() - start_time
# print(f"Elapsed time (sec): {elapsed_time_CB2:.2f}")
```

In [28]:
```
# # Evaluate the performance of the CatBoost model with median replacement on the
# # validation data and calculate the Area Under the Curve (AUC) for the ROC
# auc_CB2 = roc_auc_score(y_val, CB2.predict_proba(X_pre_val)[:,1])
# print(f'The validation AUC of the CatBoost model with median replacement is:   {auc_CB2:.6f}')
# print(f'The validation AUC of the CatBoost model with minimum replacement was: {auc_CB1:.6f}')
```

3. Explorative Datenanalyse und Feature Engineering

In Abschnitt 3 führen wir eine gezielte explorative Datenanalyse (EDA) durch, bei der wir die wichtigsten kategorialen und numerischen Variablen des Datensatzes analysieren, und erkunden dann das Feature-Engineering, um neue Attribute zu entwickeln, die das Potenzial haben, die Vorhersagefähigkeiten unserer Modelle zu verbessern.

Dabei werden wir uns auf eine auswahl zentraler numerischer Merkmale konzentrieren, die in Unterabschnitt 1.5 identifiziert wurden, sowie auf kategorischer Merkmale von besonderem Interessen.

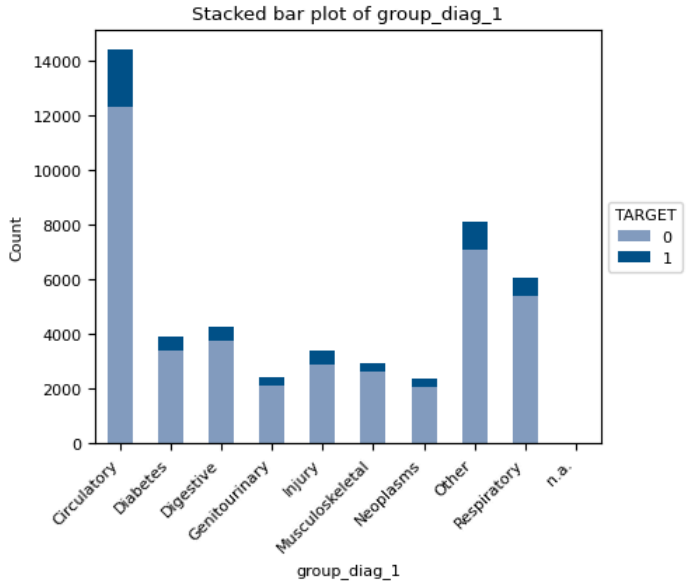
3.1 Analyse der kategorialen Merkmale

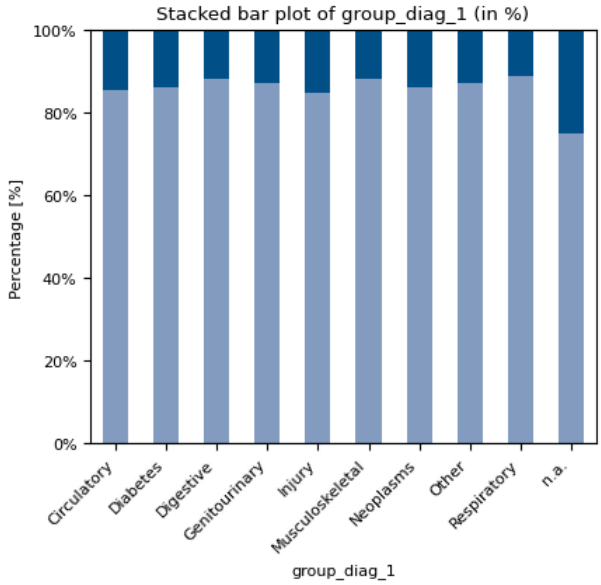
Zunächst untersuchen wir sieben ausgewählte kategoriale Variablen - `group_diag_1` , `group_diag_2` , `group_diag_3` , `age` , `medical_specialty` , `payer_code` und `discharge_disposition_id` -, um ihren Einfluss auf die Wiedereinweisung zu ermitteln. Unsere Analyse umfasst:

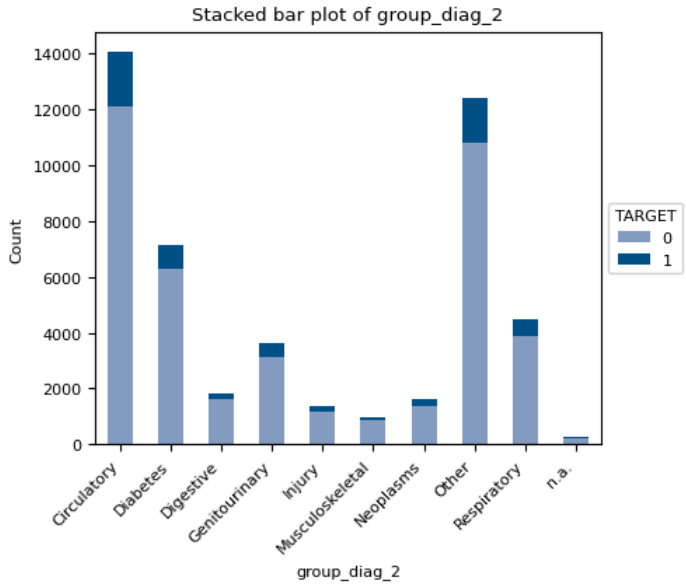
1. Erstellung von gestapelten Balkendiagrammen, um die Anzahl der Personen und die prozentuale Verteilung auf die verschiedenen Kategorien im Hinblick auf das Zielergebnis zu vergleichen.
2. Vergleich der rohen Zahl der Wiedereinweisungen mit dem Anteil innerhalb jeder Kategorie, um potenzielle Vorhersagemuster aufzudecken.

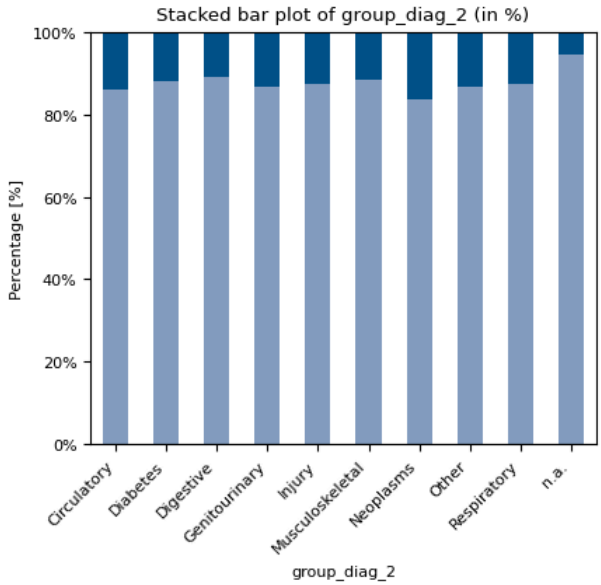
Indem wir sowohl die Anzahl als auch die prozentualen Verteilungen nebeneinander untersuchen, können wir besser verstehen, wie jedes kategoriale Merkmal mit der Wahrscheinlichkeit korreliert, dass eine Person eine Wiedereinweisung erfährt.

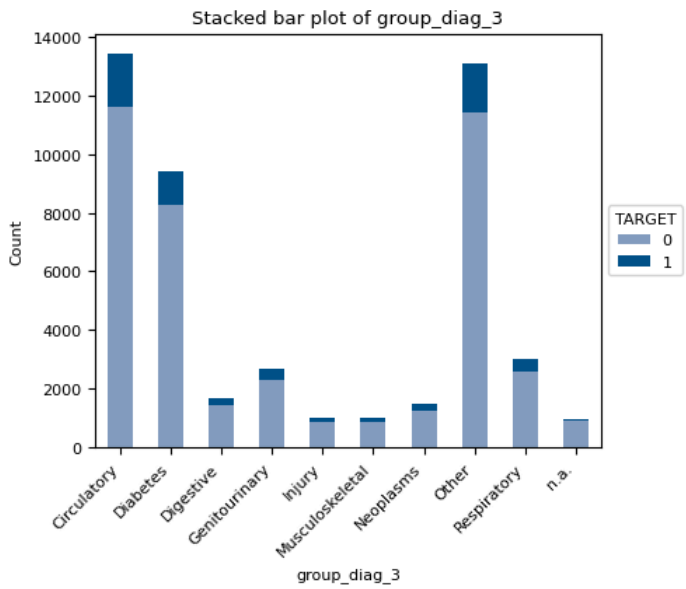
In [29]:
```
# Wird in Aufgabe PT4d noch benötigt

# List of selected categorical features to analyze
selected_categorical_features = ['group_diag_1', 'group_diag_2', 'group_diag_3', 'age', 'discharge_disposition_id', 'medical_specialty', 'pay

# Number of rows/columns for the subplot grid
n_cols = 2  # doubled to fit count and percentage plots side by side
n_rows = len(selected_categorical_features)  # one row for each feature

# Set up the matplotlib figure
plt.figure(figsize=(n_cols * 5, n_rows * 5))

# Correct font size
plt.rcParams.update({'font.size': 8})

# Loop through the number of categorical features
for idx, feature in enumerate(selected_categorical_features):
    # Create a crosstab for stacked bar plot structure
    ctab = pd.crosstab(df_pre[feature], df_pre['TARGET'])

    # (1st column) add a new subplot iteratively for count values
    ax1 = plt.subplot(n_rows, n_cols, idx * n_cols + 1)

    # Create a stacked bar plot for count values
    ctab.plot(kind="bar", stacked=True, color=[COLOR_LIGHT, COLOR_DARK], edgecolor="none", ax=ax1)

    # Additional plot settings for count subplot
    ax1.set_title(f'Stacked bar plot of {feature}')
    ax1.set_xlabel(feature)
    ax1.set_ylabel('Count')
    ax1.legend(title='TARGET', loc='center left', bbox_to_anchor=(1, 0.5))
    plt.xticks(rotation=45, ha='right')

    # (2nd column) add a new subplot iteratively for percentage values
    ax2 = plt.subplot(n_rows, n_cols, idx * n_cols + 2)

    # Normalize the crosstab by row and multiply by 100 to convert to percentages
    ctab_normalized = ctab.div(ctab.sum(axis=1), axis=0) * 100

    # Create a stacked bar plot for percentage values
    ctab_normalized.plot(kind="bar", stacked=True, color=[COLOR_LIGHT, COLOR_DARK], edgecolor="none", ax=ax2)

    # Additional plot settings for percentage subplot
    ax2.set_title(f'Stacked bar plot of {feature} (in %)')
    ax2.set_xlabel(feature)
    ax2.set_ylabel('Percentage [%]')
    ax2.legend().remove()
    plt.xticks(rotation=45, ha='right')

    # Make yticks be in percentages for the percentage subplot
    ax2.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, loc: "{:.0f}%".format(int(x))))
    ax2.set_ylim(0, 100)

# Preventing subplots from being too close to each other and display the final plot
plt.tight_layout()
plt.show()
```

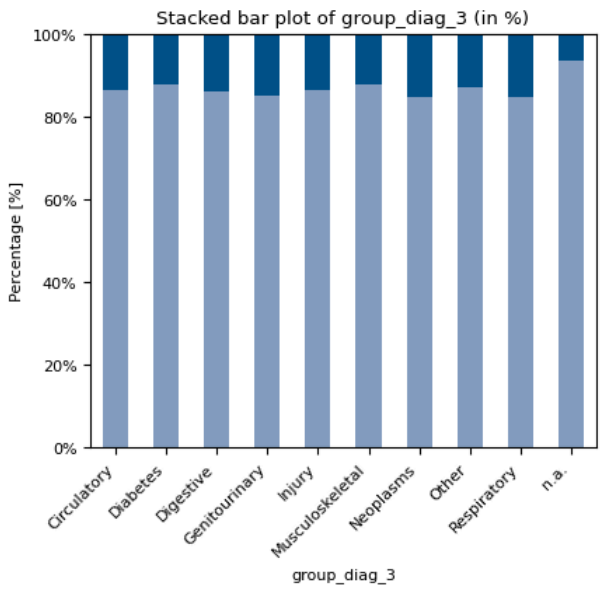

Stacked bar plot of group_diag_1
Count
TARGET
0
1
Circulatory
Diabetes
Digestive
Genitourinary
Injury
Musculoskeletal
Neoplasms
Other
Respiratory
n.a.
group_diag_1

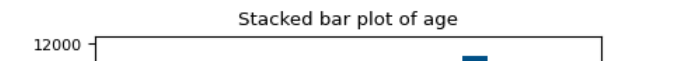

Stacked bar plot of group_diag_1 (in %)
Percentage [%]
group_diag_1


Stacked bar plot of group_diag_2
Count
TARGET
0
1
group_diag_2

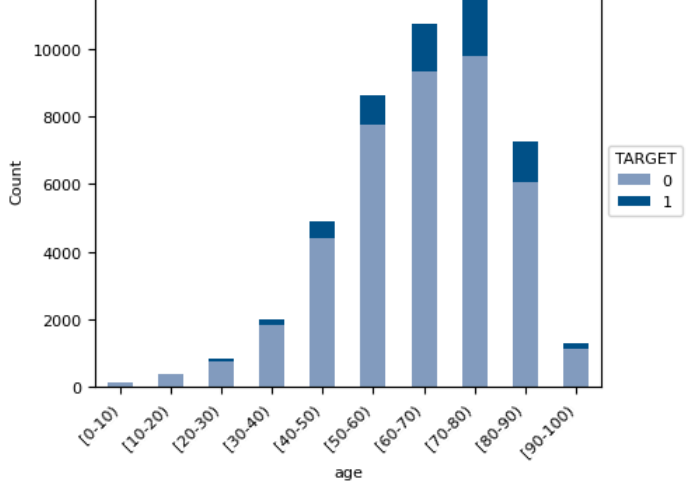

Stacked bar plot of group_diag_2 (in %)
Percentage [%]
group_diag_2

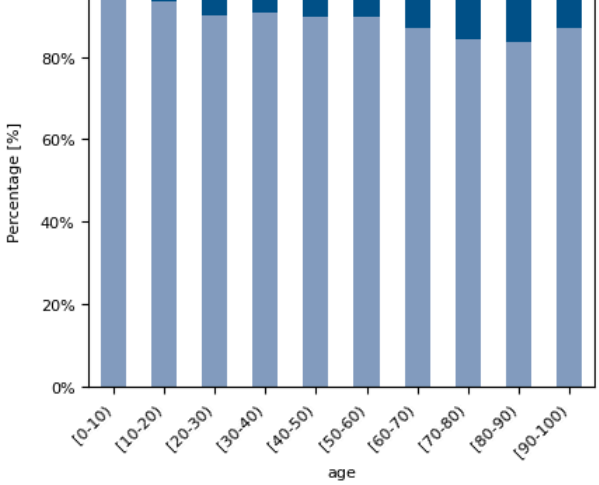

Stacked bar plot of group_diag_3
Count
TARGET
0
1
group_diag_3

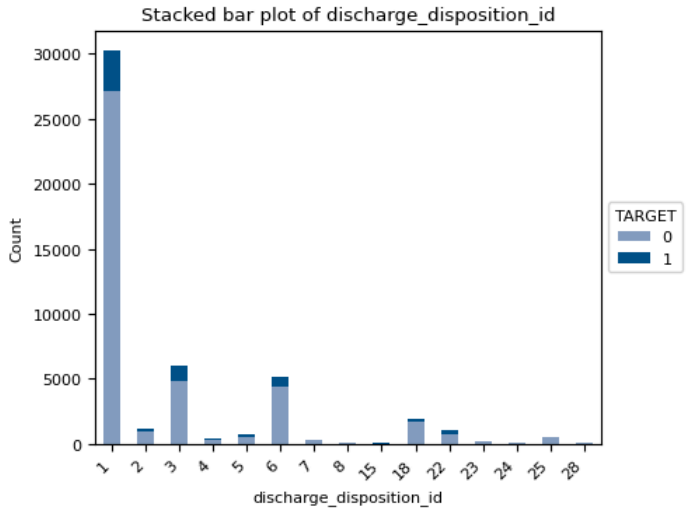

Stacked bar plot of group_diag_3 (in %)
Percentage [%]
group_diag_3

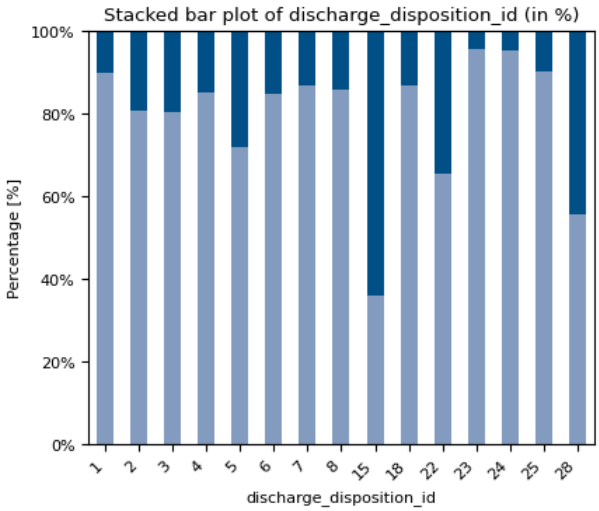

Stacked bar plot of age

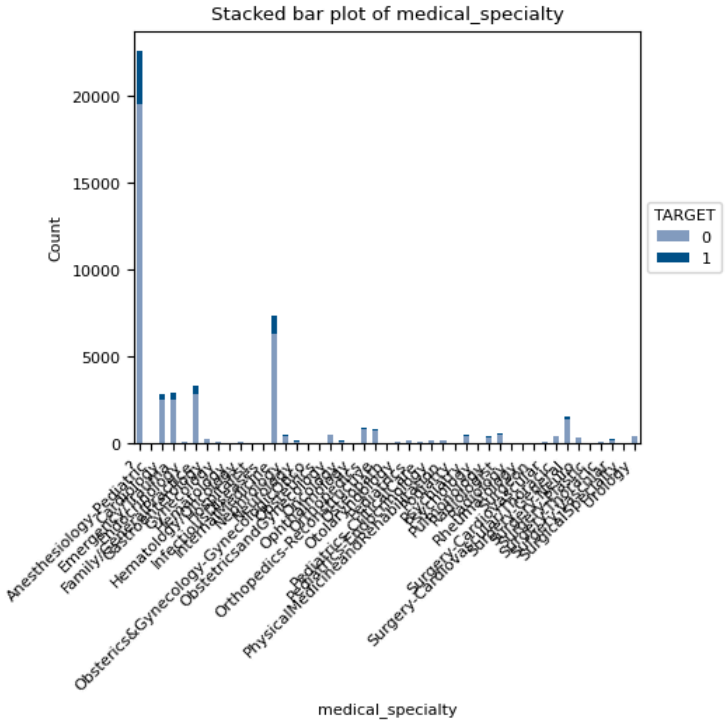

Stacked bar plot of age (in %)

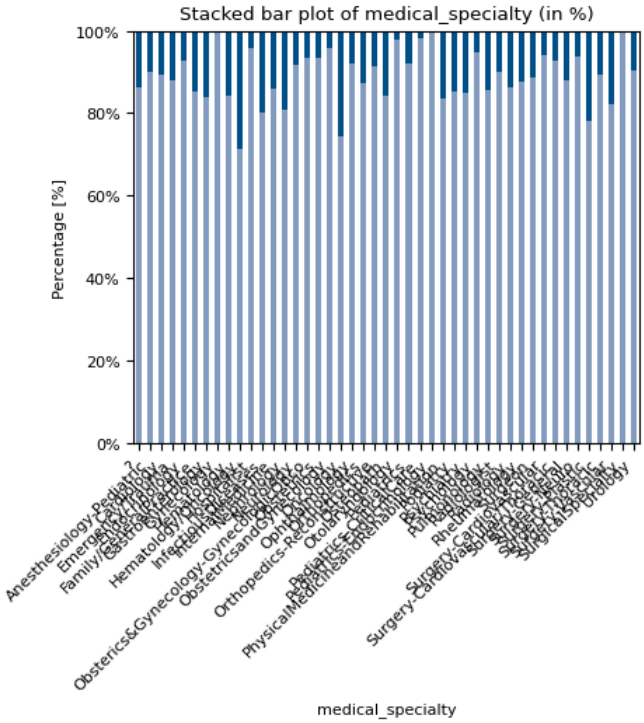

Count
age
TARGET
0
1
[0-10)
[10-20)
[20-30)
[30-40)
[40-50)
[50-60)
[60-70)
[70-80)
[80-90)
[90-100)

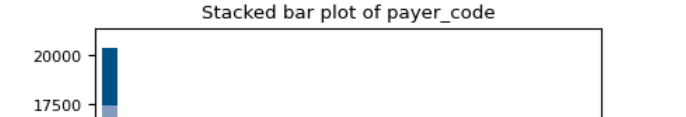

Percentage [%]
age
[0-10)
[10-20)
[20-30)
[30-40)
[40-50)
[50-60)
[60-70)
[70-80)
[80-90)
[90-100)

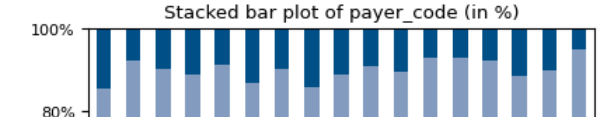

Stacked bar plot of discharge_disposition_id
Count
discharge_disposition_id
TARGET
0
1

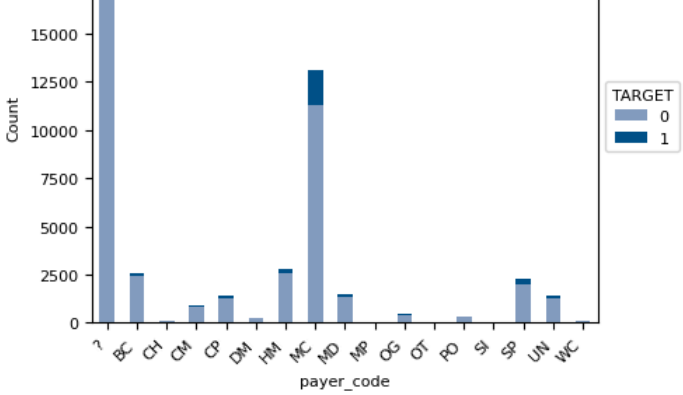

Stacked bar plot of discharge_disposition_id (in %)
Percentage [%]
discharge_disposition_id

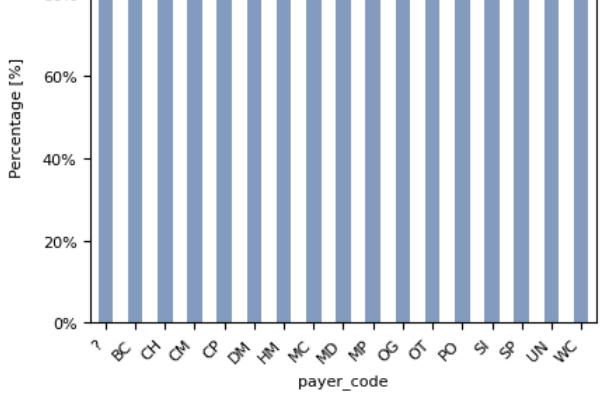

Stacked bar plot of medical_specialty
Count
medical_specialty
TARGET
0
1


Stacked bar plot of medical_specialty (in %)
Percentage [%]
medical_specialty


Stacked bar plot of payer_code


Stacked bar plot of payer_code (in %)



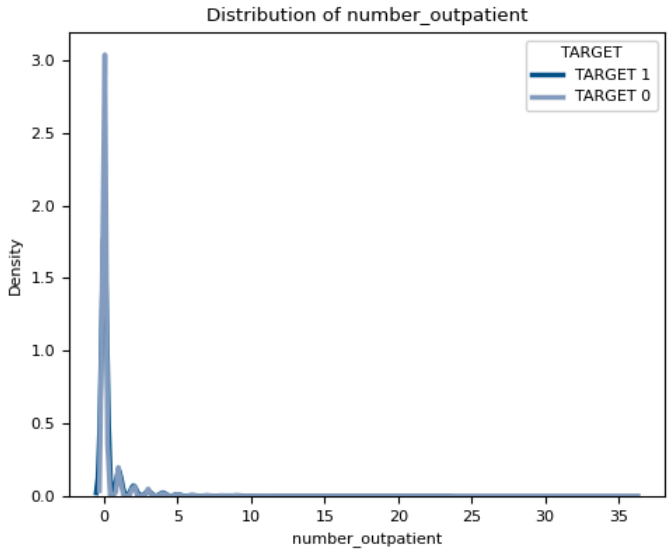


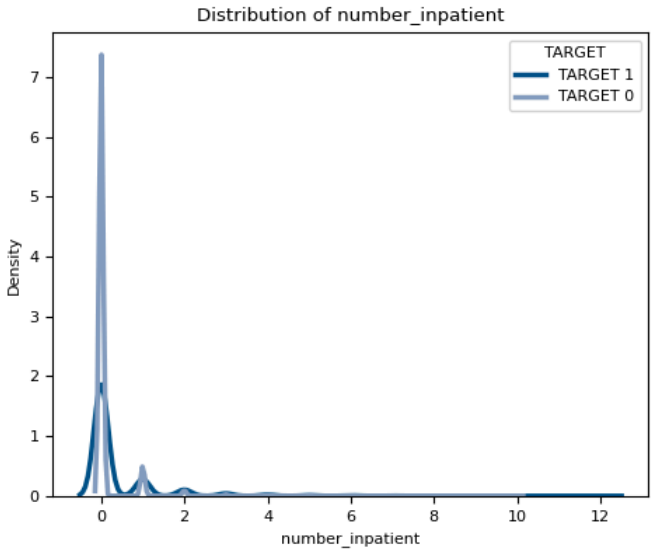
Entsprechende Interpretationen wurden bereits im R-Notebook ausgeführt und werden hier nicht nochmals angegeben.

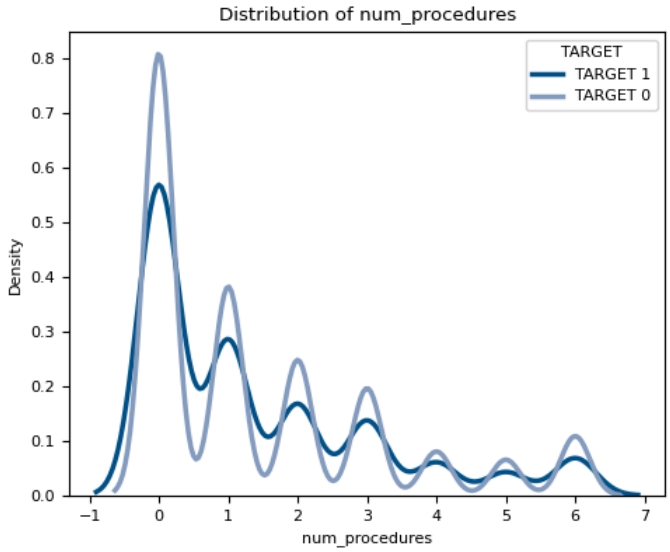
3.2 Analyse der numerischen Merkmale

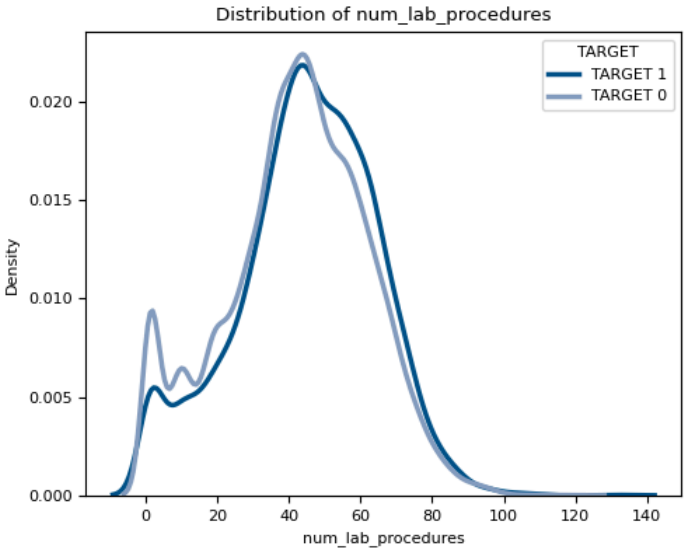
Als Nächstes analysieren wir numerische Schlüsselvariablen, um Muster im Zusammenhang mit den Wiedereinweisungen zu erkennen. Mithilfe von Kernel Density Estimation (KDE) Diagrammen beobachten wir die Dichteverteilung ausgewählter numerischer Merkmale. Durch den Vergleich dieser Verteilungen können wir feststellen, welche numerischen Prädiktoren eine signifikante Vorhersagekraft haben könnten - ein entscheidender Schritt für ein effektives Feature-Engineering und die anschließende Modellbildung bei binären Klassifizierungsaufgaben.

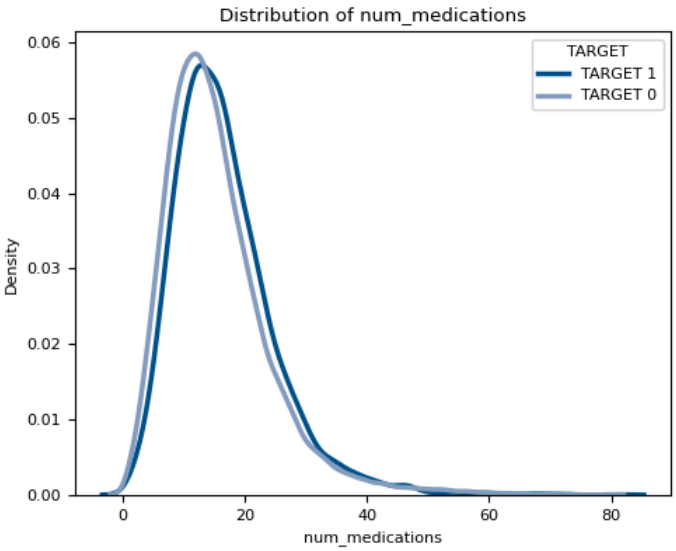
In [30]:
```
# Wird in Aufgabe PT4d noch benötigt

# List of selected numerical features to analyze
selected_numerical_features = ['number_outpatient', 'number_inpatient', 'num_procedures',
                               'num_lab_procedures','num_medications', 'time_in_hospital',
                               'number_emergency', 'number_diagnoses']

# number of plots, set up the subplot grid
n_plots = len(selected_numerical_features)
n_cols = 2  # adjust the number of columns based on your preference
n_rows = (n_plots + n_cols - 1) // n_cols

# set up the matplotlib figure
plt.figure(figsize=(n_cols * 5, n_rows * 4))

# loop through the list of numerical features
for idx, feature in enumerate(selected_numerical_features):
    ax = plt.subplot(n_rows, n_cols, idx + 1)

    # Create a color mapping based on TARGET values assuming values are 0 and 1
    color_mapping = {0: COLOR_LIGHT, 1: COLOR_DARK}

    class_vals = df_pre['TARGET'].unique()
    for val in class_vals:
        subset = df_pre[df_pre['TARGET'] == val]

        # Draw the KDE for each class
        sns.kdeplot(subset[feature], ax=ax, label=f'TARGET {val}', color=color_mapping[val], linewidth=2.5)

    # additional plot settings
    ax.set_title(f'Distribution of {feature}')
    ax.set_xlabel(f'{feature}')
    ax.set_ylabel('Density')
    ax.legend(title='TARGET')

# Preventing subplots from being too close to each other and display the final plot
plt.tight_layout()
plt.show()
```

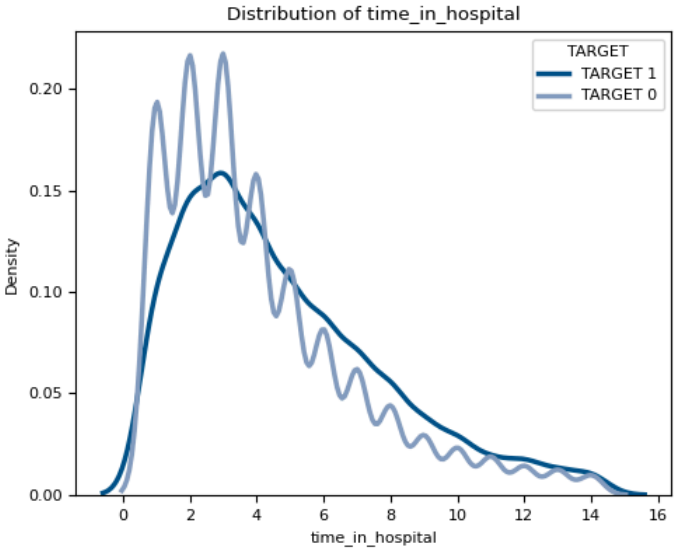

Distribution of number_outpatient
TARGET
TARGET 1
TARGET 0
Density
number_outpatient

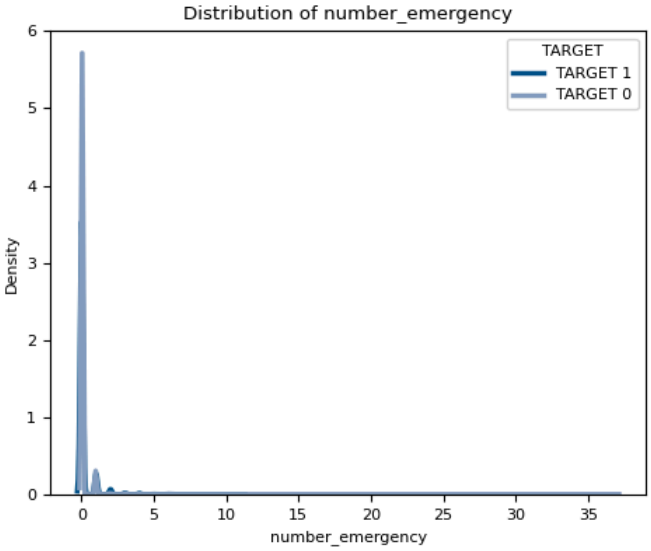

Distribution of number_inpatient
TARGET
TARGET 1
TARGET 0
Density
number_inpatient

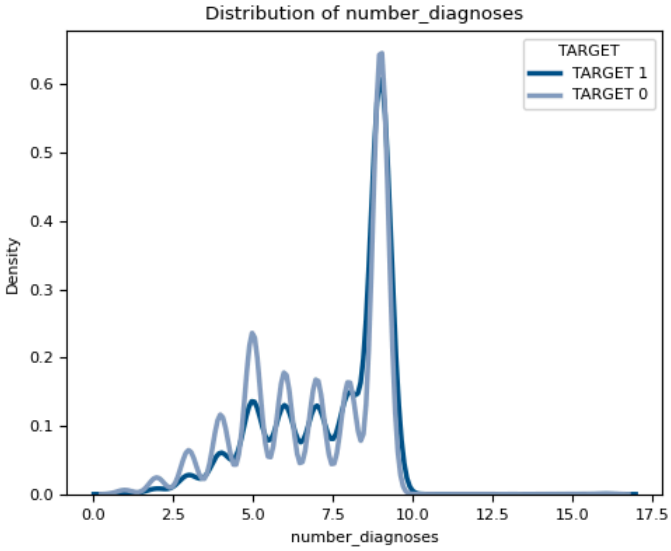

Distribution of num_procedures
TARGET
TARGET 1
TARGET 0
Density
num_procedures


Distribution of num_lab_procedures
TARGET
TARGET 1
TARGET 0
Density
num_lab_procedures


Distribution of num_medications
TARGET
TARGET 1
TARGET 0
Density
num_medications


Distribution of time_in_hospital
TARGET
TARGET 1
TARGET 0
Density
time_in_hospital


Distribution of number_emergency
TARGET
TARGET 1
TARGET 0
Density
number_emergency


Distribution of number_diagnoses
TARGET
TARGET 1
TARGET 0
Density
number_diagnoses

Entsprechende Interpretationen wurden bereits im R-Notebook ausgeführt und werden hier nicht nochmals angegeben.

3.3 Feature Engineering: Erstellen neuer Merkmale (wird nicht berücksichtigt)

4. Verbesserung der Leistung von CatBoost durch neue Features (wird nicht berücksichtigt)

5. Modelle interpretieren für ein besseres Verständnis (wird nicht berücksichtigt)

6. Vorverarbeitung von Daten: Kodierung, Skalierung und Subsampling

In diesem Abschnitt konzentrieren wir uns auf zwei wesentliche Vorverarbeitungsschritte: Kodierung, um kategoriale Daten maschinenlesbar zu machen, was für viele Algorithmen des maschinellen Lernens unerlässlich ist, und Skalierung, um die Werte numerischer Merkmale zu normalisieren, was die Effizienz der Algorithmen unterstützt. Außerdem bekämpfen wir die Unausgewogenheit der Klassen durch den Einsatz von Subsampling-Methoden, wodurch ein ausgewogenerer Trainingsdatensatz entsteht und die Recheneffizienz unserer maschinellen Lernmodelle erhöht wird.

6.1 Kodierung kategorischer Daten und Skalierung numerischer Werte

Im Allgemeinen benötigen Modelle für maschinelles Lernen Eingabedaten in numerischem Format, um mathematische Berechnungen durchführen zu können. Reale Datensätze - wie der hier betrachtete - enthalten jedoch häufig kategorische oder Textdaten. Daher ist eine Kodierung erforderlich, um diese nicht numerischen Daten in eine numerische Form umzuwandeln, die von maschinellen Lernalgorithmen verstanden und verarbeitet werden kann.

Die beiden am häufigsten verwendeten Kodierungsmethoden sind die folgenden:

- Label-Encoding: Wandelt jeden Wert in einer Spalte in eine Zahl um. Gut für ordinale Daten, d.h. wenn Kategorien eine inhärente Ordnung haben.
- One-Hot-Encoding: Erzeugt eine binäre Spalte für jede Kategorie/jedes Label in der Spalte. Dies ist nützlich für nominale Variablen, bei denen die Kategorien keine Reihenfolge oder Priorität haben.

Im Folgenden werden wir One-Hot-Encoding auf unseren Datensatz anwenden, um ihn für maschinelle Lernmethoden wie Künstliche Neuronale Netze oder XGBoost

In [31]:
```
# Output the dimensions of the dataset prior to dummy encoding
print("Shape of the dataset before dummy encoding: ", df_pre.shape)

# Perform dummy encoding on categorical features, removing the first level to avoid dummy variable trap
df = pd.get_dummies(df_pre, drop_first=True)

# Output the dimensions of the dataset after dummy encoding to show the change
print("Shape of the dataset after dummy encoding: ", df.shape)
```

```
Shape of the dataset before dummy encoding:  (47751, 51)
Shape of the dataset after dummy encoding:  (47751, 2249)
```

In [32]:
```
# Change columns names (LightGBM does not support special JSON characters in feature names)
# Wird benötigt, um z.B. die Altersklassen entsprechend anzupassen. Beispiel: 'age_[50-60)': 'age_5060'
new_names = {col: re.sub(r'[^A-Za-z0-9_]+', '', col) for col in df.columns}
new_n_list = list(new_names.values())
new_names = {col: f'{new_col}_{i}' if new_col in new_n_list[:i] else new_col for i, (col, new_col) in enumerate(new_names.items())}
df = df.rename(columns=new_names)
df.head(1)
```

Out[32]:

	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpatient	number_emergency	number_inpatient	number_diagnoses
0	8	77	6	33	0	0	0	8

In [33]:
```
# Generate the samples once again

# Split dataset in 70% training, 15% validation and 15% test and devide into feature matrix x and label y
X = df.drop(['TARGET'], axis=1)
y = df.TARGET
X_train, X_valtest, y_train, y_valtest = train_test_split(X, y, train_size=0.70, random_state=RANDOM_SEED)
X_val, X_test, y_val, y_test = train_test_split(X_valtest, y_valtest, test_size=0.50, random_state=RANDOM_SEED)
print('Sample sizes for training, validation and test: ', X_train.shape, X_val.shape, X_test.shape)
```

```
Sample sizes for training, validation and test:  (33425, 2248) (7163, 2248) (7163, 2248)
```

Die Skalierung ist beim maschinellen Lernen für die Konsistenz, die verbesserte Leistung und die Erfüllung der Algorithmusanforderungen von entscheidender Bedeutung:

1. Gleichmäßigkeit: Verschiedene Merkmale können unterschiedliche Skalen haben, wie z. B. das Alter von 0 bis 100 und das Einkommen von Tausenden bis Millionen. Dies kann Algorithmen dazu verleiten, Merkmale mit größeren Bereichen überzubewerten.
2. Effizienz: Algorithmen wie Gradient Descent und K-Nearest-Neighbors (KNN) profitieren von Merkmalen auf einer ähnlichen Skala, was zu einer besseren Leistung und schnelleren Konvergenz führt. KNN wird zum Beispiel stark von Merkmalen mit größeren Werten beeinflusst.
3. Normalisierungsanforderung: Bestimmte Algorithmen für maschinelles Lernen, wie z. B. neuronale Netze, erfordern eine Skalierung der Eingaben, damit das Modelltraining effektiv durchgeführt werden kann.

Typische Skalierungstechniken sind die Min-Max-Skalierung, bei der die Merkmale auf einen Bereich von 0 bis 1 eingestellt werden, und die Standard-Skalierung, bei der die Merkmale auf einen Mittelwert von Null und eine Einheitsvarianz eingestellt werden. Der geeignete Skalierungsansatz hängt von dem verwendeten Algorithmus und den Datenmerkmalen ab. Wir werden in unserer Analyse die Standard-Skalierung verwenden.

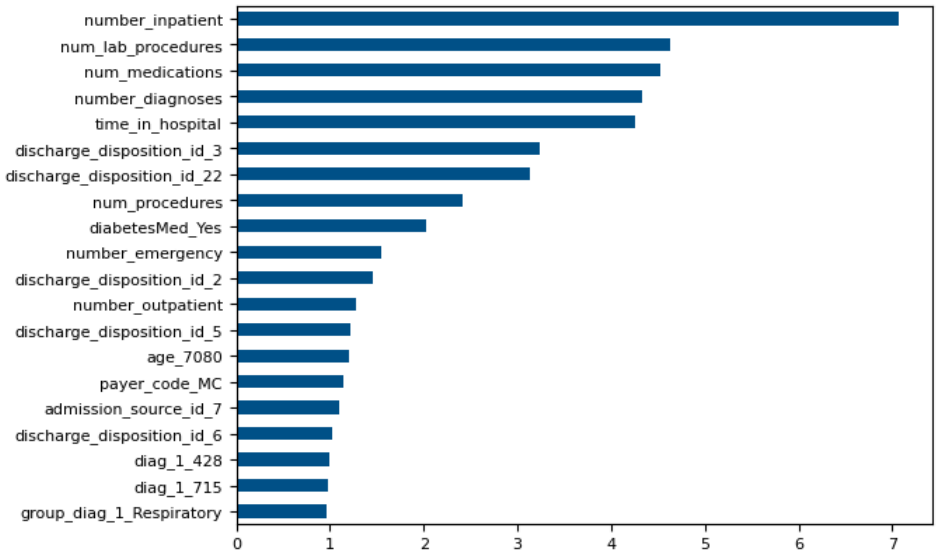
In [34]:
```
# Generate list: feature names
feature_names=list(X.columns)

# Feature scaling using StandardScaler based on training data distributions
scaler = StandardScaler()
scaler.fit(X_train)
X_train = pd.DataFrame(scaler.transform(X_train), columns = feature_names)
X_val = pd.DataFrame(scaler.transform(X_val), columns = feature_names)

# Skalierung vorziehen (vorher Teil von PT7)
X_test = pd.DataFrame(scaler.transform(X_test), columns = feature_names)
```

6.2 Haben Kodierung und Skalierung Auswirkungen auf das Benchmark-Modell?

In [35]:
```
# Repeat CB with one-hot-encoded and scaled data
start_time = time.time()

CB4 = CatBoostClassifier(eval_metric='AUC', random_seed=RANDOM_SEED)
CB4.fit(X_train, y_train, logging_level='Silent')

# Calculate and print the running time in seconds
elapsed_time_CB4 = time.time() - start_time
print(f"Elapsed time (sec): {elapsed_time_CB4:.2f}")
```

```
Elapsed time (sec): 10.07
```

In [36]:
```
# CatBoost: Plot most the top 20 most important features
pd.Series(CB4.feature_importances_, index=X_train.columns).nlargest(20).plot(kind='barh', color=COLOR_DARK).invert_yaxis()
plt.show()
```



Im Vergleich zu der Feature Importance List des "quick" Modells ist das Merkmal `discharge_disposition_id` (wichtigstes Merkmal in der "quick" Liste) mit mehreren Ausprägungen vertreut in der Liste vertreten. Weiterhin wichtig bleibt das Merkmal `number_inpatient`. Kategorielle Ausprägungen wie z.B. das Alter sind nur noch mit den relevanten Klassen (in diesem Fall die Klasse `7080`) in der Liste vertreten. Zu beachten ist, dass es deutlich mehr Merkmale im betrachteten Datensatz gibt (Steigerung von ca. 50 Merkmalen zu ca. 2250 Merkmalen d.h. Faktor 45!). Änderungen in der Reihenfolge sind u.a. dadurch erklärbar.

In [37]:
```
# Calculate and dispay the model AUC
auc_CB4 = roc_auc_score(y_val, CB4.predict_proba(X_val)[:,1])
print('The validation AUC of the CatBoost model after encoding and scaling is:   {:.6f}'.format(auc_CB4))
#print('The validation AUC of the CatBoost model before encoding and scaling was: {:.6f}'.format(auc_CB3))
```

```
The validation AUC of the CatBoost model after encoding and scaling is:   0.702311
```

In [38]:
```
print(f"Elapsed time of the CatBoost model after encoding and scaling (sec):  {elapsed_time_CB4:.2f}")
#print(f"Elapsed time of the CatBoost model before encoding and scaling (sec): {elapsed_time_CB3:.2f}")
```

```
Elapsed time of the CatBoost model after encoding and scaling (sec):  10.07
```

Zusammenfassung und Interpretation:

Durch die Kodierung der kategorialen Daten hat sich die Anzahl der Merkmale deutlich erhöht (wie bereits erwähnt). Der AUC-Wert hat sich nur geringfügig verringert (von ca. 0.707 auf 0.702). Die Laufzeit ist deutlich geringer (von ca. 57 auf 7 Sekunden). Offenbar führt die direkte Verwendung der kategorialen Merkmale innerhalb von CatBoost zu einem stark erhöhten Rechenaufwand, der jedoch nur zu einer minimalen Verbesserung der Vorhersagequalität führt.

6.3 Subsampling der Trainingsdaten

Analog der Analyse in CSN sind Wiedereinweisungen eher seltene Ereignisse, so dass der Datensatz in Bezug auf `TARGET` unausgewogen ist. Im Datensatz steht eine Wiedereinweisung durchschnittlich 6.5 Fällen gegenüber, bei denen es keine Wiedereinweisung gibt. Analog zu CSN untersuchen wir hier, ob eine Reduktion des Übergewichts auf 1:3 reduziert werden kann, ohne dass sich die resultierende Modellqualität verschlechtert, und ob die reduzierte Datenmenge das Modelltraining beschleunigen kann. Daher wird der `non_event_factor` auf 2 angepasst.

In [39]:
```
non_event_factor = 2 # thus the event rate should be 26% (instead of 8%)

# Save complete training data
X_train_all = X_train.copy(deep=True)
y_train_all = y_train.copy(deep=True)
```

In [40]:
```
# Subsampling: enhance event rate to 1:nov_event_factor
Xs = pd.concat([X_train.reset_index(drop=True), y_train.reset_index(drop=True)], axis=1)
Xs = pd.concat([Xs[Xs['TARGET']==1], Xs[Xs['TARGET']==0].sample(
          n = non_event_factor * len(Xs[Xs['TARGET']==1] ), random_state=RANDOM_SEED)],axis=0).sort_index(ascending=True)

# Over-write complete training data with subsample
X_train = Xs.drop(columns=["TARGET"])
y_train = Xs["TARGET"]

print("\nSize of training data set after subsampling: ", X_train.shape)
print("\nTARGET-distribution after subsampling:\n", y_train.value_counts())
print("\nReduction of data volume (%): ", round((len(y_train)/len(y_train_all)-1)*100))
```

```
Size of training data set after subsampling:  (13176, 2248)

TARGET-distribution after subsampling:
 TARGET
0    8784
1    4392
Name: count, dtype: int64

Reduction of data volume (%):  -61
```

6.4 Beeinflusst das Subsampling das Benchmark-Modell?

In [41]:
```
# Repeat CB with subsampled data
start_time = time.time()

CB5 = CatBoostClassifier(eval_metric='AUC', random_seed=RANDOM_SEED)
CB5.fit(X_train, y_train, logging_level='Silent')

# Calculate and print the running time in seconds
elapsed_time_CB5 = time.time() - start_time
print(f"Elapsed time (sec): {elapsed_time_CB5:.2f}")
```

```
Elapsed time (sec): 6.57
```

In [42]:
```
# Calculate and dispay the model AUC
auc_CB5 = roc_auc_score(y_val, CB5.predict_proba(X_val)[:,1])
print('The validation AUC of the CatBoost model with subsampled data is: {:.6f}'.format(auc_CB5))
```

```
The validation AUC of the CatBoost model with subsampled data is: 0.699061
```

Zusammenfassung und Schlussfolgerung:

Die kleinere Menge an Traingsdaten reduziert die Trainingszeit von CatBoost nochmal. Allerdings wird die Modellleistung nicht verbessert. Somit kann das Subsampling von Trainingsdaten eine Schlüsselkomponente zur Beschleunigung des Modelltrainings und insbesondere der Hyperparameterabstimmung sein. Im Folgenden konzentrieren wir uns auf diesen Subsampling-Datensatz.

6.5 Modellbewertung und -vergleich (neuer Abschnitt auf Basis von Abschnitt 4.2 des Notebooks CSN)

Modellvergleich vorbereiten:

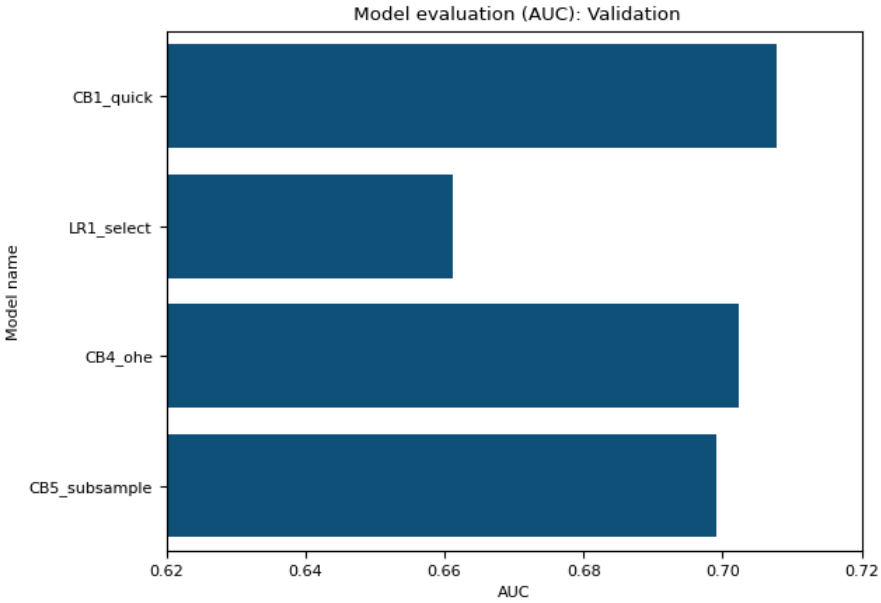
In [43]:
```
# Data structures for model names and AUC
mname = []
mauc = []
dict = {'Model name': mname,'AUC': mauc, }
```

In [44]:
```
# Store name and AUC of prevoisly fitted models
mname.append("CB1_quick")
mauc.append(auc_CB1)
mname.append("LR1_select")
mauc.append(auc_LR1)
mname.append("CB4_ohe")
mauc.append(auc_CB4)
mname.append("CB5_subsample")
mauc.append(auc_CB5)
```

In [45]:
```
def plot_auc(d,x1,x2,t):
    df_eval = pd.DataFrame(d)
    #sns.set_style('darkgrid')
    plt.title(" Model evaluation (AUC): " + t)
    sns.barplot(data = df_eval, x = "AUC", y = "Model name", color = COLOR_DARK)
    plt.xlim(x1, x2)
    plt.show()
```

Vergleichen Sie die Modellleistung (AUC - höher ist besser):

In [46]:
```
plot_auc(dict,0.62,0.72, "Validation")
```



Das erste CatBoost Modell (CB1_quick) ist im Vergleich nach AUC das Beste Modell, die logistische Regression ist deutlich schlechter. Wie bereits erwähnt beschleunigen die Skalierung und das Subsampling das Training der Modelle, die Modellgüte steigt dadurch allerdings nicht.

Aufgabe PT3: Modelloptimierung anpassen und erweitern [Lernziel 5.1; 3 Punkte]

- a) Die Abschnitte 7 und 8 sind nicht zu bearbeiten und sollen entfernt werden.
- b) Die Abschnitte 9 und 10 sind zu bearbeiten und anzupassen. Nehmen Sie je nach verwendeter Hardware (CPU/GPU) passende Einstellungen vor. Welche Änderungen an den HP-Grids sind vorzunehmen? Interpretieren Sie die Ergebnisse.
- c) Der Abschnitt 11 ist zunächst zu überspringen.

Lösungsvorschlag:

Part C: Tuning and Applying Machine Learning Models

In Teil C geht es um die Nutzung des vollen Potenzials unserer Modelle durch die Feinabstimmung der Hyperparameter für regularisierte logistische Regression, künstliche neuronale Netze und die Gradientenbaum-Boosting-Modelle CatBoost, LightGBM und XGBoost. Wir führen eine detaillierte vergleichende Bewertung anhand von Validierungs- und Testdatensätzen durch und untersuchen die Anwendung dieser Modelle in Hochrisikobereichen. Abschließend fassen wir die wichtigsten Erkenntnisse zusammen, die wir bei unserer Untersuchung des maschinellen Lernens gewonnen haben. Zum Abschluss reflektieren wir die wichtigsten Erkenntnisse aus unserer Reise zum maschinellen Lernen.

7. Regularisierte logistische Regression zur Vermeidung von Überanpassung (wird nicht berücksichtigt)

8. Optimizing a Feed-Forward Neural Network (wird nicht berücksichtigt)

9. Hyperparameter Optimization in CatBoost

Nun wollen wir untersuchen, ob wir die Lernrate und die Baumtiefe von CatBoost optimieren können, indem wir eine kostspielige kreuzvalidierte randomisierte Suche durchführen (Anpassung von 40 Modellen).

Im ersten Codeblock schaffen wir die Voraussetzungen für die Optimierung der Hyperparameter des CatBoostClassifier-Modells. Wir definieren ein Parameterraster, das den Bereich der zu untersuchenden Werte für die Lernrate und die Tiefe der Bäume angibt. Hier wird „loguniform" für die Lernrate verwendet, um aus einer logarithmischen Verteilung zu wählen und die Suche über verschiedene Skalen zu optimieren. In ähnlicher Weise stellt eine Liste von Ganzzahlen mögliche Werte für die Tiefe der Bäume dar.

In [47]:
```
# Define the hyperparameters grid to search
param_grid_CB ={'learning_rate': loguniform(0.015, 0.06), # loguniform(0.015, 0.06),
                'depth':         [5,6,7,8,9]}

# Define variables for results reports
sel_params_CB = ['param_learning_rate','param_depth','mean_test_score','rank_test_score']
```

Nach der anfänglichen Einrichtung implementiert unser nächster Codeblock die eigentliche Hyperparameter-Optimierung mit RandomizedSearchCV, einer Strategie, die den Parameterraum zufällig abtastet und eine vierfache Kreuzvalidierung durchführt. Wir wählen eine vernünftige Anzahl von Iterationen (`n_iter=10`), um ein Gleichgewicht zwischen Rechenzeit und Gründlichkeit der Suche herzustellen. Die Ergebnisse werden aufgezeichnet, einschließlich der Laufzeit und der besten gefundenen Parameter.

In [48]:
```
# CatBoostClassifier: Hyper parameter optimization with RandomizedSearchCV
tic = time.time()

CB_rs = RandomizedSearchCV(CatBoostClassifier(iterations=1000, eval_metric='AUC'), param_grid_CB,
                           cv=4, n_iter=10, random_state=RANDOM_SEED, scoring='roc_auc') # n_jobs=-1,

CB_rs.fit(X_train, y_train, logging_level='Silent')

# Print the runtime as well as the five best hyper parameter constellations
print("time (sec):" + "%6.0f" % (time.time() - tic))
print("Best hyper parameters:",CB_rs.best_params_)
pd.DataFrame(CB_rs.cv_results_)[sel_params_CB].sort_values("rank_test_score").head()
```

```
time (sec):   373
Best hyper parameters: {'depth': 7, 'learning_rate': 0.017229876161789517}
```

Out[48]:

	param_learning_rate	param_depth	mean_test_score	rank_test_score
3	0.017230	7	0.693159	1
5	0.018285	8	0.692142	2
8	0.019300	6	0.691830	3
7	0.040811	6	0.691635	4
6	0.015434	7	0.691492	5

Nachdem wir die besten Hyperparameter ermittelt haben, verwenden wir sie, um ein neues CatBoostClassifier-Modell mit der gesamten Trainingsmenge zu erstellen. Bei diesem Ansatz werden die optimierten Parameter genutzt, die theoretisch ein genaueres Modell ergeben sollten.

In [49]:
```
# CatBoostClassifier: Fit new model on all folds (with best parameters)
tic = time.time()
CB6 = CatBoostClassifier(**CB_rs.best_params_, iterations=1000, eval_metric='AUC')
CB6.fit(X_train, y_train, logging_level='Silent')
print("time (sec):" + "%6.0f" % (time.time() - tic))
```

```
time (sec):     7
```

Schließlich bewerten wir die Leistung des abgestimmten Modells, indem wir den AUC-Wert anhand der Validierungsmenge berechnen.

In [50]:
```
# Calculate and dispay the model AUC
auc_CB6 = roc_auc_score(y_val, CB6.predict_proba(X_val)[:,1])
print('The validation AUC of the CatBoost model with tuned hyperparameters is: {:.6f}'.format(auc_CB6))
print('Comparison: The val. AUC of the CatBoost model with subsampled data is: {:.6f}'.format(auc_CB5))
```

```
The validation AUC of the CatBoost model with tuned hyperparameters is: 0.700055
Comparison: The val. AUC of the CatBoost model with subsampled data is: 0.699061
```

Das Training des Modells benötigt deutlich mehr Zeit. Das resultierende Modell ist allerdings leicht besser.

10. HP-Tuning LightGBM and XGBoost

In diesem Abschnitt richten wir unsere Aufmerksamkeit auf die wichtigsten Konkurrenten von CatBoost: LightGBM und XGBoost. Indem wir die Leistung von Grafikprozessoren zur Beschleunigung der Berechnungen nutzen und einen umfangreicheren Prozess zur Abstimmung der Hyperparameter durchführen, intensivieren wir unsere Bemühungen zur Modellanpassung erheblich, indem wir die Anzahl der bewerteten Modelle verdoppeln. Dieser rigorose Vergleich zielt darauf ab, die Leistung dieser führenden Algorithmen unter einem ähnlichen Optimierungsregime zu vergleichen.

10.1 LightGBM: HP-Abstimmung und Bewertung

In [51]:
```
# Parameter search range for LightGBM
param_grid_LGB = {'learning_rate':    loguniform(0.01, 0.05),
                  'num_leaves':       [32,40,45,50,55],
                  'subsample':        uniform(0,1),
                  'colsample_bytree': uniform(0,1),
                  'verbose':          [-1]}

# Create a list of variables for displaying the cross-validation result
sel_params_LGB = ['param_learning_rate', 'param_num_leaves', 'param_subsample',
                  'param_colsample_bytree', 'mean_test_score', 'rank_test_score']
```

In [52]:
```
# LGBMClassifier: Hyperparameter optimization with RandomizedSearchCV
tic = time.time()

LGB_rs = RandomizedSearchCV(LGBMClassifier(device='cpu', n_estimators=1000),
                            param_grid_LGB, cv=4, n_iter=20, scoring='roc_auc',
                            random_state=RANDOM_SEED) # n_jobs=-1,
LGB_rs.fit(X_train, y_train)

# Output the runtime and best parameters (as well as the runner-ups)
print("Elapsed time (sec):" + "%6.0f" % (time.time() - tic))
print("Best Parameters:", LGB_rs.best_params_)
pd.DataFrame(LGB_rs.cv_results_)[sel_params_LGB].sort_values("rank_test_score").head()
```

```
Elapsed time (sec):   229
Best Parameters: {'colsample_bytree': 0.33370861113902184, 'learning_rate': 0.012585185474052117, 'num_leaves': 45, 'subsample': 0.020584494295802447, 'verbose': -1}
```

Out[52]:

	param_learning_rate	param_num_leaves	param_subsample	param_colsample_bytree	mean_test_score	rank_test_score
2	0.012585	45	0.020584	0.333709	0.688414	1
8	0.011104	50	0.942202	0.170524	0.688042	2
4	0.016318	55	0.431945	0.183405	0.684956	3
15	0.018693	40	0.539692	0.325330	0.684851	4
9	0.018595	40	0.097672	0.563288	0.684122	5

In [53]:
```
# LGBMClassifier: Create a new model on all folds using the best parameters
tic = time.time()
LGB = LGBMClassifier(**LGB_rs.best_params_, device='cpu', n_estimators=1000)
LGB.fit(X_train, y_train)
print("Elapsed time (sec):" + "%6.0f" % (time.time() - tic))
```

```
Elapsed time (sec):     4
```

In [54]:
```
# Calculate and dispay the model AUC
auc_LGB = roc_auc_score(y_val, LGB.predict_proba(X_val)[:,1])
print('The validation AUC of the LightGBM model with tuned hyperparameters is: {:.6f}'.format(auc_LGB))
```

```
The validation AUC of the LightGBM model with tuned hyperparameters is: 0.699489
```

10.2 XGBoost: HP-tuning and evaluation

In [55]:
```
# Parameter search range for XGBoost
param_grid_XGB = {'learning_rate':    loguniform(0.01, 0.05),
                  'max_depth':        [5,6,7,8,9],
                  'subsample':        uniform(0,1),
                  'colsample_bytree': uniform(0,1)}

# Create a list of variables for displaying the cross-validation result
sel_params_XGB = ['param_learning_rate', 'param_max_depth', 'param_subsample',
                  'param_colsample_bytree', 'mean_test_score', 'rank_test_score']
```

In [56]:
```
# XGBClassifier: Hyperparameter optimization with RandomizedSearchCV
tic = time.time()

XGB_rs = RandomizedSearchCV(XGBClassifier(n_estimators=1000, tree_method="hist", device="cpu"),
                            param_grid_XGB, cv=4, n_iter=20, scoring="roc_auc",
                            n_jobs=-1, random_state=RANDOM_SEED)
XGB_rs.fit(X_train, y_train)

# Output the runtime and best parameters (as well as the runner-ups)
print("Elapsed time (sec):" + "%6.0f" % (time.time() - tic))
print("Best Parameters:", XGB_rs.best_params_)
pd.DataFrame(XGB_rs.cv_results_)[sel_params_XGB].sort_values("rank_test_score").head()
```

```
Elapsed time (sec):  1128
Best Parameters: {'colsample_bytree': 0.7553614103176525, 'learning_rate': 0.01982308318513928, 'max_depth': 6, 'subsample': 0.5677003278199915}
```

Out[56]:

	param_learning_rate	param_max_depth	param_subsample	param_colsample_bytree	mean_test_score	rank_test_score
12	0.019823	6	0.567700	0.755361	0.691729	1
15	0.018693	6	0.539692	0.325330	0.691102	2
8	0.011104	8	0.942202	0.170524	0.690764	3
10	0.020307	8	0.495177	0.684233	0.689811	4
4	0.016318	9	0.431945	0.183405	0.689663	5

In [57]:
```
# XGBClassifier: Create a new model on all folds using the best parameters
tic = time.time()
XGB = XGBClassifier(**XGB_rs.best_params_, n_estimators=1000, tree_method="hist", eval_metric="auc")
XGB.fit(X_train, y_train)
print("Elapsed time (sec):" + "%6.0f" % (time.time() - tic))
```

```
Elapsed time (sec):    52
```

In [58]:
```
# Calculate and display the model AUC
auc_XGB = roc_auc_score(y_val, XGB.predict_proba(X_val)[:,1])
print('The validation AUC of the XGBoost model with tuned hyperparameters is: {:.6f}'.format(auc_XGB))
```

```
The validation AUC of the XGBoost model with tuned hyperparameters is: 0.700008
```

Bei den HP-Grids ist darauf zu achten, dass die verwendeten Parameter nicht an den äußeren Rändern des angegebenen Wertebereichs liegen. Ist das der Fall, kann das ein Hinweis auf weiteren Optimierungsbedarf geben. Die Wertebereiche sollten angepasst werden. Im Vergleich zum CSN-Notebook ist das für das HP-Tuning durchgeführt worden. Die berechneten AUC-Werte liegen hinter dem ursprünglichen Cat-Boost Modell zurück. Im nächsten Abschnitt werden diese Modelle insgesamt u.a. mit den noch folgenden Modellen verglichen.

Damit ist zunächst das Ende der Modellierungsansätze aus dem Notebook CSN erreicht.

Aufgabe PT4: DXG-Diagnosegruppen bilden und Erzeugung von Embeddings vorbereiten

Aufgabe PT4-1: [Lernziel 5.1; 5 Punkte]

Vorbereitend für die Abschnitte PT4 und PT5 werden für die kategoriellen Merkmale Anpassungen zur Erstellung von Embeddings durchgeführt.

Hinweis: In diesem Aufgabenabschnitt wird der Umfang des Notebooks CSN verlassen. Nachfolgende Aufgaben sind in der vorgegebenen Reihenfolge abzuarbeiten.

a) Lesen Sie den in den Materialien vorhandenen Datensatz `icd9_data.csv` ein und geben Sie die ersten drei Zeilen aus.

b) Mergen Sie den ursprünglichen Datensatz (aus Aufgabenteil PT1 1.2) mit dem Datensatz aus Teil a). Dabei sollen nur die Spalten `group`, `L2code` und `category` aus dem Datensatz aus a) erhalten bleiben. Im Anschluss sind drei Zeilen des neuen Datensatzes auszugeben.

c) Bezogen auf den Datensatz aus PT4-1 b): Welche Kategorien haben, bezogen auf die Wiedereinweisung, die höchsten absoluten Anteile? Geben Sie die drei höchsten Kategorien aus.

d) Das Feature `category` besitzt Ausprägungen, die teilweise schwach besetzt sind. Um auch für dieses Merkmal stabile Embeddings erzeugen zu können, soll wie folgt vorgegangen werden:

- Ergänzen Sie den Datensatz aus Teil b) um die Spalte `sum_target_cg`, die mit Hilfe des Ergebnisses aus Teil c) zu befüllen ist. Damit sollten die Ausprägungen von `category` die zugehörigen absoluten Anteile zugeordnet bekommen.
- Abschließend soll die Spalte `hdiag` hinzugefügt werden: Falls die Kategorie einer Zeile öfter als 20 Mal im Datensatz vorkommt, ist der Wert von `category` zu übernehmen. Falls nicht, dann ist der Wert die Kombination des Strings "o." und dem Wert des Features `group`. Kürzen Sie den Inhalt der neuen Spalte im Anschluss auf maximal 30 Zeichen. Verifizieren Sie die Ergebnisse exemplarisch an zwei beliebigen Zeilen.

Lösungsvorschlag:

Zu a):

In [59]:
```
# Diagnosegruppierungsdaten einlesen
# icd9_data = pd.read_csv("C:/ml/data/DiabetesHospital/icd9_data.csv", sep=";")
# icd9_data = pd.read_csv("../input/diabeteshospital/icd9_data.csv", sep=";")
# icd9_data = pd.read_csv("../2_Datensätze/icd9_data.csv", sep=";")
icd9_data = pd.read_csv("./PK_VADS_IMMERSION/2025//2_Datensätze/icd9_data.csv", sep=";")
icd9_data.head(3)
```

Out[59]:

	category_id	code	group	chapter	chapter_id	L2code	category	X
0	0	?	n.a.	n.a	0	n.a.	Not available	NaN
1	135	1	Digestive	Diseases of the digestive system	9	c9.1	Intestinal infection	NaN
2	1	10	Other	Infectious and parasitic diseases	1	c1.1	Tuberculosis	NaN

Zu b):

In [60]:
```
df_diag = pd.merge(df_raw, icd9_data, how='left', left_on= 'diag_1', right_on= 'code')
df_diag = df_diag.drop(columns=['category_id', 'code', 'chapter','chapter_id', 'X'], axis=1)
df_diag.head(3)
```

Out[60]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_specialty	num_
0	Caucasian	Female	[50-60)	?	2	1	1	8	?	Cardiology	
1	Caucasian	Female	[50-60)	?	3	1	1	2	?	Surgery-Neuro	
2	Caucasian	Female	[80-90)	?	1	3	7	4	MC	InternalMedicine	

Zu c):

In [61]:
```
# Schritt 1: Über "category" gruppieren und die Variable TARGET summieren
df_sum = df_diag.groupby('category', as_index=False)['TARGET'].sum().rename(columns={'TARGET': 'sum_target_cg'})
df_sum.sort_values(by='sum_target_cg', ascending=False).head(3)
```

Out[61]:

	category	sum_target_cg
60	Congestive heart failure; nonhypertensive	443
62	Coronary atherosclerosis and other heart disease	429
5	Acute cerebrovascular disease	324

Zu d):

In [62]:
```
# Schritt 2: Das Ergebnis über "category" dem DataFrame df_diag zuordnen (Merge)
df_diag = df_diag.merge(df_sum, on='category', how='left')

min_TARGET = 20

# Schritt 3: Neues Feld "h_diag" berechnen: Wenn TARGET zu selten, dann grobe Krankheitsgruppe verwenden und diese mit 'o.' kennzeichen (da n
df_diag['hdiag'] = df_diag.apply(
    lambda row: 'o.' + row['group'] if row['sum_target_cg'] < min_TARGET else row['category'], axis=1
```

```
)

# Länge kürzen
df_diag['hdiag'] = df_diag['hdiag'].str[:30]

# Ergebnis auszählen
df_diag['hdiag'].value_counts()
```

Out[62]:
```
hdiag
Coronary atherosclerosis and o    3835
Congestive heart failure; nonh    2219
Other lower respiratory diseas    2170
Diabetes mellitus without comp    2007
Acute myocardial infarction       2006
                                  ...
Acute posthemorrhagic anemia       135
Chronic ulcer of skin               98
Maintenance chemotherapy; radi      83
Other liver diseases                72
o.n.a.                               8
Name: count, Length: 72, dtype: int64
```

In [63]:
```
# Beispiel mit category >= 20
df_diag[["sum_target_cg", "category", "group", "hdiag"]].iloc[0]
```

Out[63]:
```
sum_target_cg                       22
category         Essential hypertension
group                      Circulatory
hdiag            Essential hypertension
Name: 0, dtype: object
```

In [64]:
```
# Beispiel mit category < 20
df_diag[["sum_target_cg", "category", "group", "hdiag"]].iloc[3]
```

Out[64]:
```
sum_target_cg                                    6
category         Gout and other crystal arthropathies
group                                        Other
hdiag                                      o.Other
Name: 3, dtype: object
```

Aufgabe PT4-2: [Lernziel 5.1 / 2.2; 10 Punkte]

a) Nach erfolgter Anpassung des Merkmals `category` in der vorherigen Aufgabe werden weitere relevante, kategorielle Merkmale angepasst. Dabei soll wie folgt vorgegangen werden:

Falls die Anzahl der Merkmalsausprägung kleiner als der Grenzwert 20 ist, dann ist

- bei dem Merkmal `discharge_disposition_id` die Ausprägung durch `99`,
- bei dem Merkmal `medical_specialty` die Ausprägung durch `ZZ`,
- bei dem Merkmal `payer_code` die Ausprägung durch `ZZ`,
- bei dem Merkmal `admission_type_id` die Ausprägung durch `99`,
- bei dem Merkmal `age` durch die Ausprägung `[0,20)`

zu ersetzen. Nach erfolgter Ersetzung sind die Merkmalswerte mit ihrer Häufigkeit auszugeben.

b) Visualisieren Sie die unter a) sowie PT4-1 erstellten Merkmale geeignet. Mindestens eine Auffälligkeit ist dabei genauer zu analysieren.

Lösungsvorschlag:

Zu a)

In [65]:
```
# discharge_disposition:

# Schritt 1: Über MERKMAL gruppieren und die Variable TARGET summieren
df_sum = df_diag.groupby('discharge_disposition_id', as_index=False)['TARGET'].sum().rename(columns={'TARGET': 'sum_target_dd'})

# Schritt 2: Das Ergebnis über MERKMAL dem DataFrame df_diag zuordnen (Merge)
df_diag = df_diag.merge(df_sum, on='discharge_disposition_id', how='left')

# Schritt 3: Neues Feld  berechnen: Wenn TARGET zu selten, zusammenfassen zu Restgruppe
df_diag['ddisp'] = df_diag.apply(
    lambda row: 99 if row['sum_target_dd'] < min_TARGET else row['discharge_disposition_id'], axis=1)

# Ergebnis auszählen
df_diag['ddisp'].value_counts()
```

Out[65]:
```
ddisp
1     30258
3      6029
6      5189
18     1921
2      1101
22     1073
5       664
25      494
4       375
7       296
99      279
28       72
Name: count, dtype: int64
```

In [66]:
```
# medical_specialty:

# Schritt 1: Über MERKMAL gruppieren und die Variable TARGET summieren
df_sum = df_diag.groupby('medical_specialty', as_index=False)['TARGET'].sum().rename(columns={'TARGET': 'sum_target_ms'})

# Schritt 2: Das Ergebnis über MERKMAL dem DataFrame df_diag zuordnen (Merge)
df_diag = df_diag.merge(df_sum, on='medical_specialty', how='left')

# Schritt 3: Neues Feld  berechnen: Wenn TARGET zu selten, zusammenfassen zu Restgruppe
df_diag['mspec'] = df_diag.apply(
    lambda row: 'ZZ' if row['sum_target_ms'] < min_TARGET else row['medical_specialty'], axis=1)

# Ergebnis auszählen
df_diag['mspec'].value_counts()
```

Out[66]:
```
mspec
?                                 22642
InternalMedicine                   7341
Family/GeneralPractice             3312
Emergency/Trauma                   2875
Cardiology                         2811
Surgery-General                    1528
ZZ                                 1126
Orthopedics                         891
Orthopedics-Reconstructive          806
Radiologist                         576
ObstetricsandGynecology             506
Nephrology                          460
Psychiatry                          450
Pulmonology                         415
Urology                             409
Surgery-Cardiovascular/Thoracic     402
Surgery-Neuro                       340
Gastroenterology                    246
Surgery-Vascular                    241
PhysicalMedicineandRehabilitation   153
Oncology                            144
Hematology/Oncology                  77
Name: count, dtype: int64
```

In [67]:
```
# payer_code:

# Schritt 1: Über MERKMAL gruppieren und die Variable TARGET summieren
df_sum = df_diag.groupby('payer_code', as_index=False)['TARGET'].sum().rename(columns={'TARGET': 'sum_target_pc'})

# Schritt 2: Das Ergebnis über MERKMAL dem DataFrame df_diag zuordnen (Merge)
df_diag = df_diag.merge(df_sum, on='payer_code', how='left')

# Schritt 3: Neues Feld  berechnen: Wenn TARGET zu selten, zusammenfassen zu Restgruppe
df_diag['pcode'] = df_diag.apply(
    lambda row: 'ZZ' if row['sum_target_pc'] < min_TARGET else row['payer_code'], axis=1)

# Ergebnis auszählen
df_diag['pcode'].value_counts()
```

Out[67]:
```
pcode
?     20378
MC    13115
HM     2821
BC     2601
SP     2259
MD     1522
CP     1391
UN     1385
CM      913
OG      475
PO      357
ZZ      278
DM      256
Name: count, dtype: int64
```

In [68]:
```
# admission_type:

# Schritt 1: Über MERKMAL gruppieren und die Variable TARGET summieren
df_sum = df_diag.groupby('admission_type_id', as_index=False)['TARGET'].sum().rename(columns={'TARGET': 'sum_target_at'})

# Schritt 2: Das Ergebnis über MERKMAL dem DataFrame df_diag zuordnen (Merge)
df_diag = df_diag.merge(df_sum, on='admission_type_id', how='left')

# Schritt 3: Neues Feld  berechnen: Wenn TARGET zu selten, zusammenfassen zu Restgruppe
df_diag['atype'] = df_diag.apply(
    lambda row: 99 if row['sum_target_at'] < min_TARGET else row['admission_type_id'], axis=1)

# Ergebnis auszählen
df_diag['atype'].value_counts()
```

Out[68]:
```
atype
1     24036
3     10056
2      8731
6      2641
5      2047
8       222
99       18
Name: count, dtype: int64
```

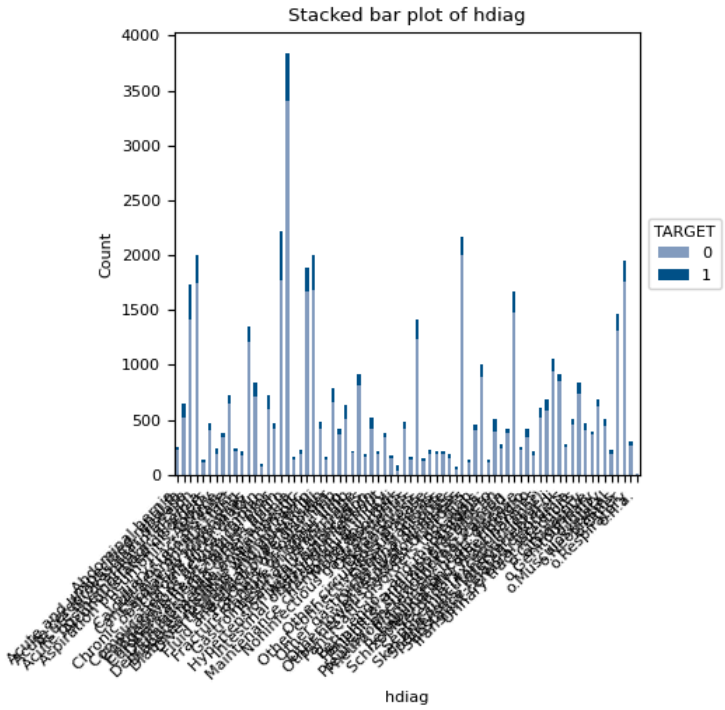
In [69]:
```python
# age:

# Schritt 1: Über MERKMAL gruppieren und die Variable TARGET summieren
df_sum = df_diag.groupby('age', as_index=False)['TARGET'].sum().rename(columns={'TARGET': 'sum_target_age'})

# Schritt 2: Das Ergebnis über MERKMAL dem DataFrame df_diag zuordnen (Merge)
df_diag = df_diag.merge(df_sum, on='age', how='left')

# Schritt 3: Neues Feld  berechnen: Wenn TARGET zu selten, zusammenfassen zu Restgruppe
df_diag['agegr'] = df_diag.apply(
    lambda row: '[0-20)' if row['sum_target_age'] < min_TARGET else row['age'], axis=1)

# Ergebnis auszählen
df_diag['agegr'].value_counts()
```

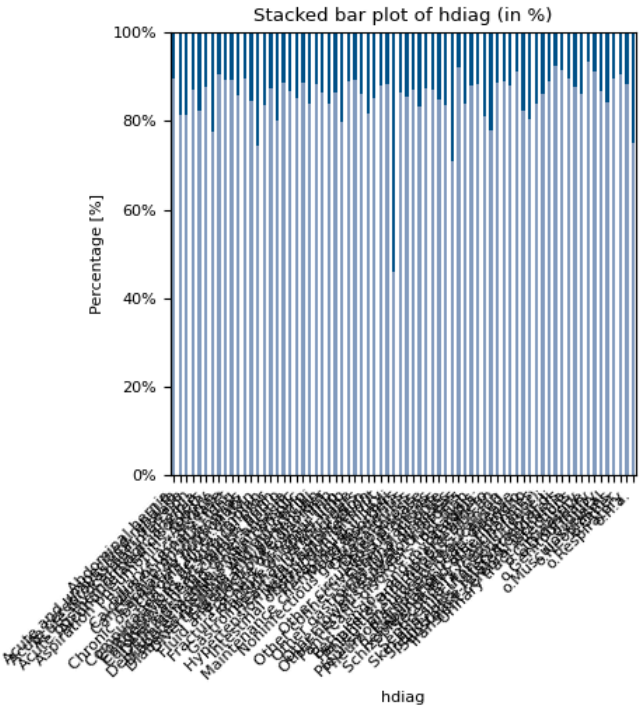
Out[69]:
```
agegr
[70-80)     11623
[60-70)     10736
[50-60)      8615
[80-90)      7242
[40-50)      4885
[30-40)      2003
[90-100)     1282
[20-30)       844
[10-20)       392
[0-20)        129
Name: count, dtype: int64
```

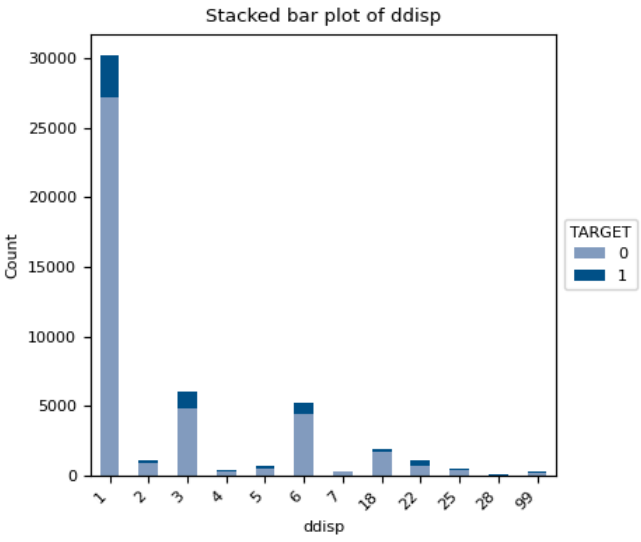
Zu b)

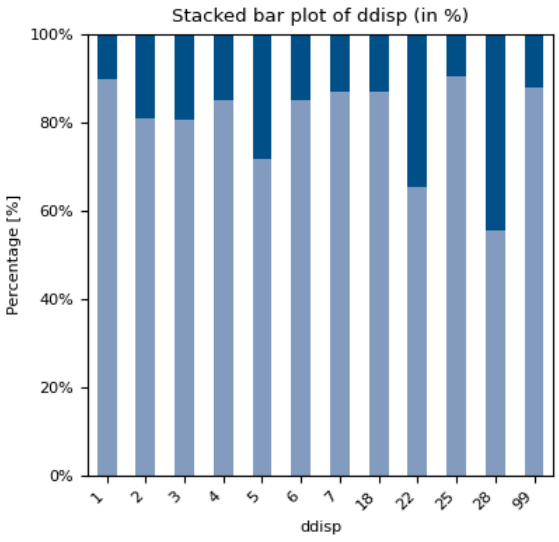
In [70]:
```python
# List of selected categorical features to analyze
selected_categorical_features = ['hdiag', 'ddisp','mspec', 'pcode', 'atype', 'agegr']

# Number of rows/columns for the subplot grid
n_cols = 2  # doubled to fit count and percentage plots side by side
n_rows = len(selected_categorical_features)  # one row for each feature

# Set up the matplotlib figure
plt.figure(figsize=(n_cols * 5, n_rows * 5))

# Correct font size
plt.rcParams.update({'font.size': 8})

# Loop through the number of categorical features
for idx, feature in enumerate(selected_categorical_features):
    # Create a crosstab for stacked bar plot structure
    ctab = pd.crosstab(df_diag[feature], df_diag['TARGET'])

    # (1st column) add a new subplot iteratively for count values
    ax1 = plt.subplot(n_rows, n_cols, idx * n_cols + 1)

    # Create a stacked bar plot for count values
    ctab.plot(kind="bar", stacked=True, color=[COLOR_LIGHT, COLOR_DARK], edgecolor="none", ax=ax1)

    # Additional plot settings for count subplot
    ax1.set_title(f'Stacked bar plot of {feature}')
    ax1.set_xlabel(feature)
    ax1.set_ylabel('Count')
    ax1.legend(title='TARGET', loc='center left', bbox_to_anchor=(1, 0.5))
    plt.xticks(rotation=45, ha='right')

    # (2nd column) add a new subplot iteratively for percentage values
    ax2 = plt.subplot(n_rows, n_cols, idx * n_cols + 2)

    # Normalize the crosstab by row and multiply by 100 to convert to percentages
    ctab_normalized = ctab.div(ctab.sum(axis=1), axis=0) * 100

    # Create a stacked bar plot for percentage values
    ctab_normalized.plot(kind="bar", stacked=True, color=[COLOR_LIGHT, COLOR_DARK], edgecolor="none", ax=ax2)

    # Additional plot settings for percentage subplot
    ax2.set_title(f'Stacked bar plot of {feature} (in %)')
    ax2.set_xlabel(feature)
    ax2.set_ylabel('Percentage [%]')
    ax2.legend().remove()
    plt.xticks(rotation=45, ha='right')

    # Make yticks be in percentages for the percentage subplot
    ax2.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, loc: "{:.0f}%".format(int(x))))
    ax2.set_ylim(0, 100)

# Preventing subplots from being too close to each other and display the final plot
```

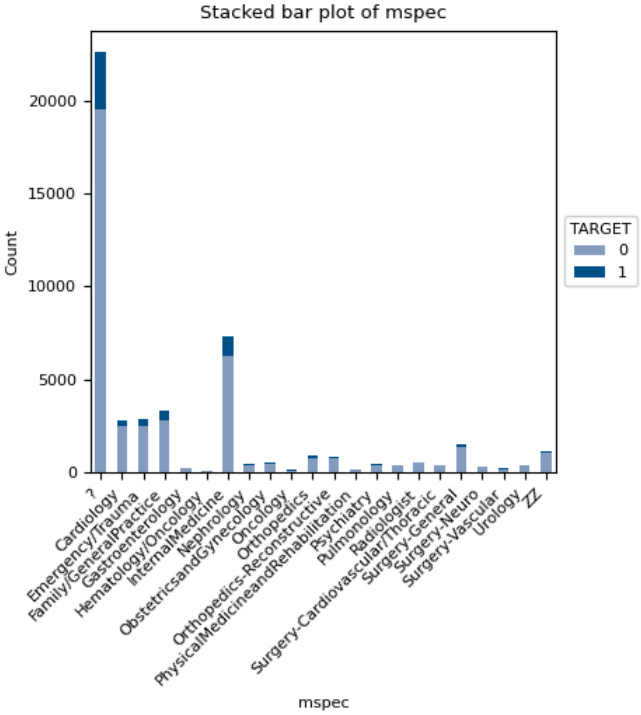
```
plt.tight_layout()
plt.show()
```

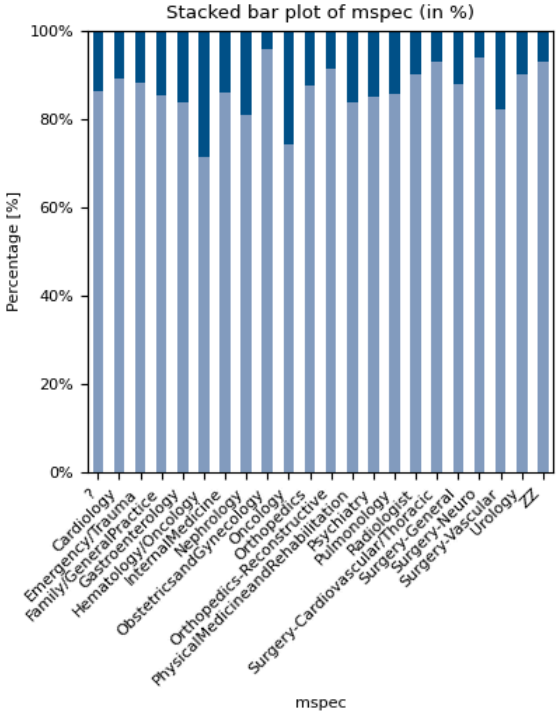

Stacked bar plot of hdiag
Count
TARGET
0
1
hdiag

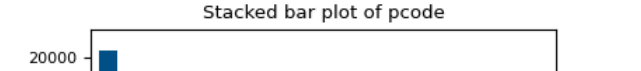

Stacked bar plot of hdiag (in %)
Percentage [%]
hdiag


Stacked bar plot of ddisp
Count
TARGET
0
1
ddisp

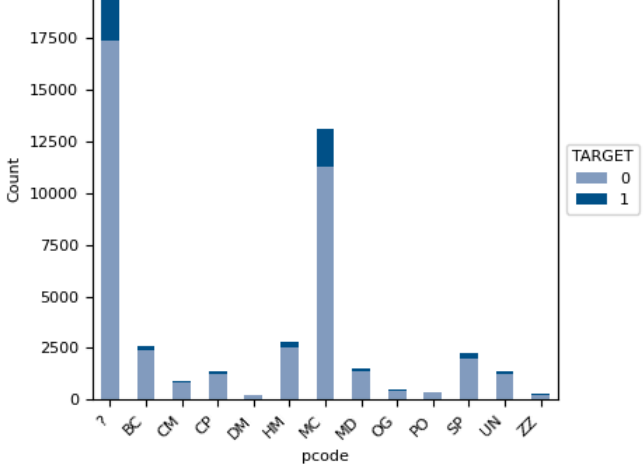

Stacked bar plot of ddisp (in %)
Percentage [%]
ddisp

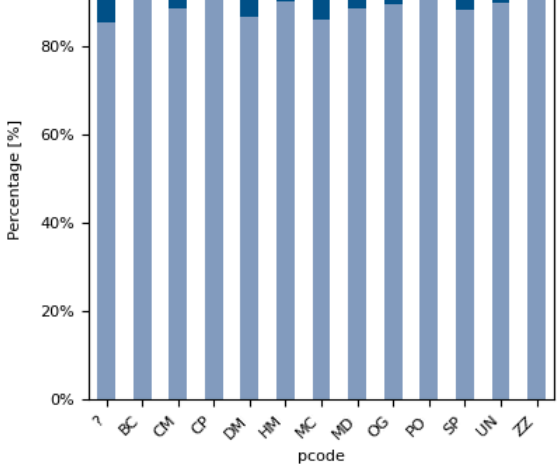

Stacked bar plot of mspec
Count
TARGET
0
1
mspec

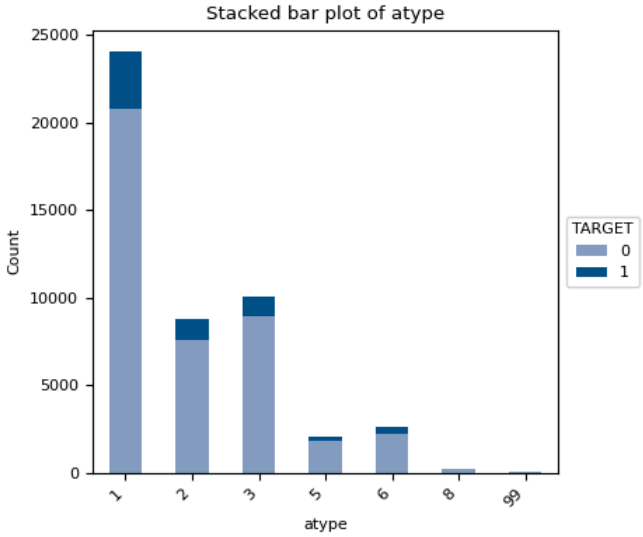

Stacked bar plot of mspec (in %)
Percentage [%]
mspec

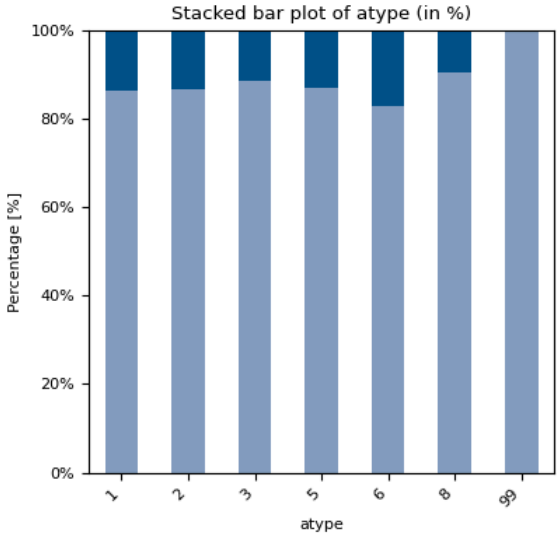

Stacked bar plot of pcode

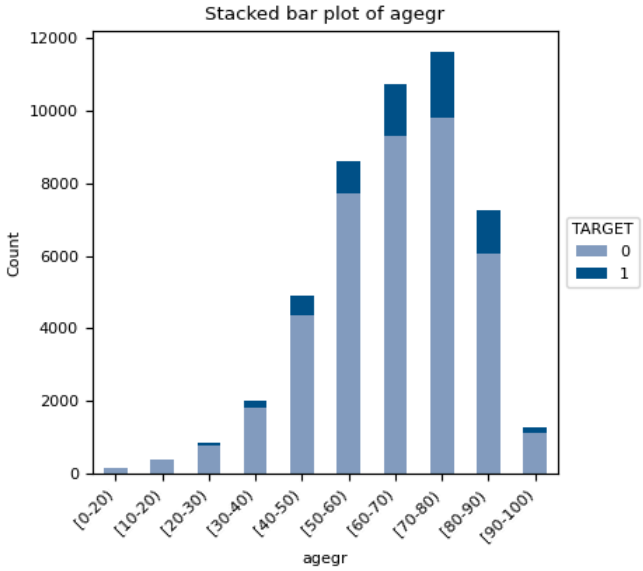

Stacked bar plot of pcode (in %)

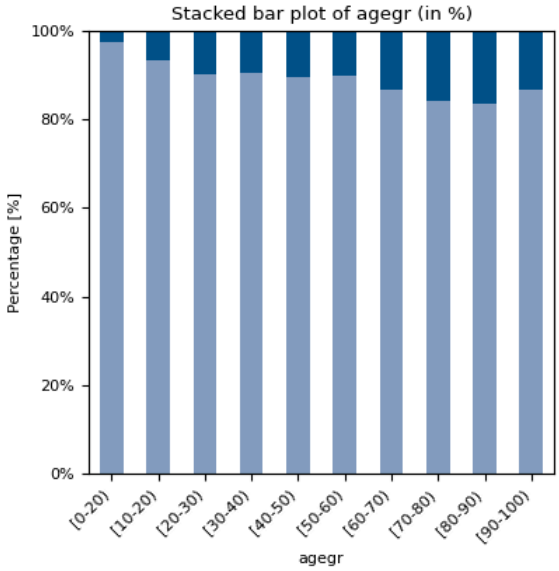












Die Graphik zu `hdiag` (zusammengefasstes Merkmal `category`) zeigt weiterhin viele Ausprägungen (und dadurch einen schwer lesbaren Text auf der holizontalen Achse). Bei genauer Betrachtung erkennt man in der rechten Graphik einen tiefen blauen Balken, d.h. eine auffällige Ausprägung mit einer hohen Wahrscheinlichkeit eine Wiedereinweisung. Um die Ausprägung zu ermitteln, werden die Wiedereinweisungswahrscheinlichkeiten berechnet und der höhe nach sortiert:

In [71]:
```
# Check hdiag: Krankheiten mit den höchsten Wiedereinweisungswahrscheinlicheiten
df_diag.groupby('hdiag', as_index=False)['TARGET'].mean().sort_values(by='TARGET', ascending=False).head()
```

Out[71]:

	hdiag	TARGET
34	Maintenance chemotherapy; radi	0.542169
43	Other liver diseases	0.291667
13	Chronic ulcer of skin	0.255102
71	o.n.a.	0.250000
6	Aspiration pneumonitis; food/v	0.223629

Aus dieser Liste sieht man, dass es sich bei diesem Merkmal um Chemotherapie handelt. Vermehrte Krankenhausbesuche sind verständlich und erklärbar.

Bei dem Feature mspec ist ebenfalls ein auffälliger blauer Balken zu erkennen. Da die Zusammenfassung von dünn besetzten Merkmalsausprägungen für eine gut lesbare Darstellung gesorgt hat, kann das Merkmal direkt abgelesen werden: Hämatologie/Okologie. Ermittlung der Wiedereinweisungswahrscheinlichkeiten nach `mspec`:

In [72]:
```
# Check mspec: Medical Specialty mit den höchsten Wiedereinweisungswahrscheinlicheiten
df_diag.groupby('mspec', as_index=False)['TARGET'].mean().sort_values(by='TARGET', ascending=False).head()
```

Out[72]:

	mspec	TARGET
5	Hematology/Oncology	0.285714
9	Oncology	0.256944
7	Nephrology	0.191304
19	Surgery-Vascular	0.178423
12	PhysicalMedicineandRehabilitation	0.163399

Aufgabe PT5: Erzeugung stabiler Diagnose-Embeddings mit performantem neuronalem Netzwerk

Aufgabe PT5-1: [Lernziel 5.1; 6 Punkte]

a) Für die in den Aufgabenteilen PT4-1 und PT4-2 erzeugten kategoriellen Variablen ist ein Label-Encoding durchzuführen. Geben Sie von dem neuen Datensatz drei zufällige Zeilen aus.

b) Skalieren Sie die numerischen Features `number_inpatient`, `num_lab_procedures`, `number_diagnoses` sowie `num_medications`. Demonstrieren Sie die Wirkungsweise der Skalierung.

c) Zum Trainieren eines neuronalen Netzwerkes soll ein Datensatz bestehend aus den kategoriellen Merkmalen aus PT4-2 a) sowie den numerischen Features aus Teil b) dieser Aufgabe erstellt werden. Dieser ist im Verhältnis 30%/15%/15% (Anteil Training, Anteil Validierung, Anteil Test) aufzuteilen.

Lösungsvorschlag:

Zu a):

In [73]:
```
# Liste der nominalen Merkmale für Embeddings anlegen
emb_features = ['hdiag', 'ddisp','mspec', 'pcode', 'atype', 'agegr']
num_emb_features = []

# Nominale Merkmale "faktorisieren" (label encoding), 0,1,2,...
for i in emb_features:
    df_diag[f'num_{i}'] = df_diag[f'{i}'].factorize()[0]
    num_emb_features.append(f'num_{i}')
num_emb_features
```

Out[73]:
```
['num_hdiag', 'num_ddisp', 'num_mspec', 'num_pcode', 'num_atype', 'num_agegr']
```

In [74]:
```
df_diag.sample(n=3)
```

Out[74]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_specialty
47523	Asian	Female	[50-60)	?	3	6	1	1	?	Orthopedics
8582	Caucasian	Male	[40-50)	?	2	6	4	8	?	?
42024	Caucasian	Female	[70-80)	?	1	3	7	3	MC	?

Zu b):

In [75]:
```
# Skalierung der numerischen Daten mit StandardScaler
num_features = ['number_inpatient', 'num_lab_procedures', 'number_diagnoses', 'num_medications']
scaler = StandardScaler()
```

```
df_scaled = df_diag.copy(deep=True)
df_scaled[num_features] = scaler.fit_transform(df_diag[num_features])
# Kennzahlen anzeigen
df_scaled.describe()
```

Out[75]:

	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpatient	number_emergency	number_inpatient	number_diagn
count	47751.000000	4.775100e+04	47751.000000	4.775100e+04	47751.000000	47751.000000	4.775100e+04	4.775100e
mean	4.205106	5.356859e-17	1.458043	-4.850934e-17	0.232979	0.081569	4.761653e-18	-1.047564
std	2.914298	1.000010e+00	1.764084	1.000010e+00	0.927995	0.445226	1.000010e+00	1.000010e
min	1.000000	-2.087767e+00	0.000000	-1.709703e+00	0.000000	0.000000	-2.595729e-01	-2.981599e
25%	2.000000	-6.237921e-01	0.000000	-6.528603e-01	0.000000	0.000000	-2.595729e-01	-1.027122e
50%	3.000000	8.295457e-02	1.000000	-1.831526e-01	0.000000	0.000000	-2.595729e-01	4.387350
75%	6.000000	6.887375e-01	2.000000	5.214089e-01	0.000000	0.000000	-2.595729e-01	9.273541
max	14.000000	4.525362e+00	6.000000	7.684451e+00	36.000000	37.000000	2.226972e+01	4.347688e

In [76]:
```
df_scaled.head(3)
```

Out[76]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_specialty	num_
0	Caucasian	Female	[50-60)	?	2	1	1	8	?	Cardiology	
1	Caucasian	Female	[50-60)	?	3	1	1	2	?	Surgery-Neuro	
2	Caucasian	Female	[80-90)	?	1	3	7	4	MC	InternalMedicine	

Zu c):

In [77]:
```
# Generate the samples once again: X-y-split
feature_list = num_features + num_emb_features
xn = df_scaled[feature_list]
yn = df_scaled.TARGET
xn.head(3)
```

Out[77]:

	number_inpatient	num_lab_procedures	number_diagnoses	num_medications	num_hdiag	num_ddisp	num_mspec	num_pcode	num_atype	num_agegr
0	-0.259573	1.748858	0.438735	2.047959	0	0	0	0	0	0
1	-0.259573	0.335364	-2.004361	-0.535433	1	0	1	0	1	0
2	-0.259573	1.294520	0.927354	0.873690	2	1	2	1	2	1

In [78]:
```
# Split dataset in 70% training, 15% validation and 15% test and devide into feature matrix x and label y
xn_train, xn_valtest, yn_train, yn_valtest = train_test_split(xn, yn, train_size=0.70, random_state=RANDOM_SEED)
xn_val, xn_test, yn_val, yn_test = train_test_split(xn_valtest, yn_valtest, test_size=0.50, random_state=RANDOM_SEED)
print('Sample sizes for training, validation and test: ', xn_train.shape, xn_val.shape, xn_test.shape)
```

```
Sample sizes for training, validation and test:  (33425, 10) (7163, 10) (7163, 10)
```

Aufgabe PT5-2: [Lernziel 5.1; 20 Punkte]

a) Für die Embeddings der kategoriellen Features ist ein neuronales Netzwerk zu erstellen. Dabei ist Folgendes zu berücksichtigen:

- Als Eingabe erhält das Netzwerk den Datensatz aus PT5-1 c).
- Für jedes kategorielle Merkmal soll ein Embedding mit zwei Dimensionen erzeugt werden.
- Die Anzahl der Hidden Layers kann frei gewählt werden. Die finale Entscheidung ist zu begründen.

b) Fitten Sie das unter a) definierte Netzwerk. Plotten Sie den Loss und Validation Loss über die Epochen des Trainings. Kommentieren Sie den Verlauf. Berechnen Sie im Anschluss den Validation Loss.

c) Um im nächsten Abschnitt eine Beurteilung der berechneten Embeddings vornehmen zu können, soll das Netzwerk aus Teil a) geklont und im Anschluss erneut gefittet werden. Der Validation Loss ist zu berechnen und auszugeben.

d) Extrahieren Sie die Embeddings aus beiden Netzwerken und plotten Sie diese paarweise nebeneinander. Welche Auffälligkeiten gibt es? Wie sind die Embeddings bezüglich Stabilität zu beurteilen?

Lösungsvorschlag:

Zu a):

In [79]:
```
# Feaures für die Embeddings
print(emb_features)
```

```
['hdiag', 'ddisp', 'mspec', 'pcode', 'atype', 'agegr']
```

In [80]:
```python
# Anzahl Ausprägungen der Embedding-Merkmale auszählen und speichern als len_name
for i in emb_features:
    # Create the variable name dynamically and assign the length
    var_name = f"len_{i}"
    globals()[var_name] = len(xn_train[f'num_{i}'].unique())
    print(var_name)
    print(globals()[var_name])
```

```
len_hdiag
72
len_ddisp
12
len_mspec
22
len_pcode
13
len_atype
7
len_agegr
10
```

Das folgende Netzwerk berücksichtigt die (skalierten) numerischen Variablen sowie die per label Encoding modifizierten kategoriellen Variablen. Nach der Angabe der Embedding Layers werden diese über Concatenate mit den numerischen Merkmalen zusammengeführt. Es werden insgesamt zwei hidden Layers berücksichtigt, da damit bereits eine schnelle Konvergenz erreicht werden kann.

In [81]:
```python
# Numerical input
in_num = Input(shape=(4,), name='in_num')

# Define input dimension dictionary for each feature
input_dims = {'hdiag': len_hdiag, 'ddisp': len_ddisp, 'mspec': len_mspec, 'pcode': len_pcode, 'atype': len_atype, 'agegr': len_agegr}

# Dictionary to store the input and embedding layers
input_layers = {}
embedding_layers = {}

# Loop through each feature to create Input, Embedding, and Flatten layers
for feature in emb_features:
    # Create Input Layer
    input_layers[feature] = Input(shape=(1,), name=f'in_{feature}')

    # Create Embedding and Flatten layers
    embedding = Embedding(input_dim=input_dims[feature], output_dim=2, name=f'emb_{feature}')(input_layers[feature])
    embedding_layers[feature] = Flatten()(embedding)

# Concatenate the numerical input and all embedding layers
concatenated_input = concatenate([in_num] + list(embedding_layers.values()), name='concate')

# Define hidden layers
hidden1 = Dense(64, kernel_regularizer=regularizers.l2(0.0001), activation='relu')(concatenated_input)
hidden2 = Dense(32, kernel_regularizer=regularizers.l2(0.0001), activation='relu')(hidden1)
hidden3 = Dense(16, kernel_regularizer=regularizers.l2(0.0001), activation='relu')(hidden2)

# Output Layer
output = Dense(1, activation='sigmoid')(hidden3)
#output = Dense(1, activation='sigmoid')(hidden2)

# Instantiate and compile the model
NNemb = Model(inputs=[in_num] + list(input_layers.values()), outputs=output)
NNemb.compile(optimizer='adam', loss='binary_crossentropy', metrics=['AUC'])

# Summary
NNemb.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
in_hdiag (InputLayer)	(None, 1)	0	-
in_ddisp (InputLayer)	(None, 1)	0	-
in_mspec (InputLayer)	(None, 1)	0	-
in_pcode (InputLayer)	(None, 1)	0	-
in_atype (InputLayer)	(None, 1)	0	-
in_agegr (InputLayer)	(None, 1)	0	-
emb_hdiag (Embedding)	(None, 1, 2)	144	in_hdiag[0][0]
emb_ddisp (Embedding)	(None, 1, 2)	24	in_ddisp[0][0]
emb_mspec (Embedding)	(None, 1, 2)	44	in_mspec[0][0]
emb_pcode (Embedding)	(None, 1, 2)	26	in_pcode[0][0]
emb_atype (Embedding)	(None, 1, 2)	14	in_atype[0][0]
emb_agegr (Embedding)	(None, 1, 2)	20	in_agegr[0][0]
in_num (InputLayer)	(None, 4)	0	-
flatten (Flatten)	(None, 2)	0	emb_hdiag[0][0]
flatten_1 (Flatten)	(None, 2)	0	emb_ddisp[0][0]
flatten_2 (Flatten)	(None, 2)	0	emb_mspec[0][0]
flatten_3 (Flatten)	(None, 2)	0	emb_pcode[0][0]
flatten_4 (Flatten)	(None, 2)	0	emb_atype[0][0]
flatten_5 (Flatten)	(None, 2)	0	emb_agegr[0][0]
concate (Concatenate)	(None, 16)	0	in_num[0][0], flatten[0][0], flatten_1[0][0], flatten_2[0][0], flatten_3[0][0], flatten_4[0][0], flatten_5[0][0]
dense (Dense)	(None, 64)	1,088	concate[0][0]
dense_1 (Dense)	(None, 32)	2,080	dense[0][0]
dense_2 (Dense)	(None, 16)	528	dense_1[0][0]
dense_3 (Dense)	(None, 1)	17	dense_2[0][0]

Total params: 3,985 (15.57 KB)

Trainable params: 3,985 (15.57 KB)

Non-trainable params: 0 (0.00 B)

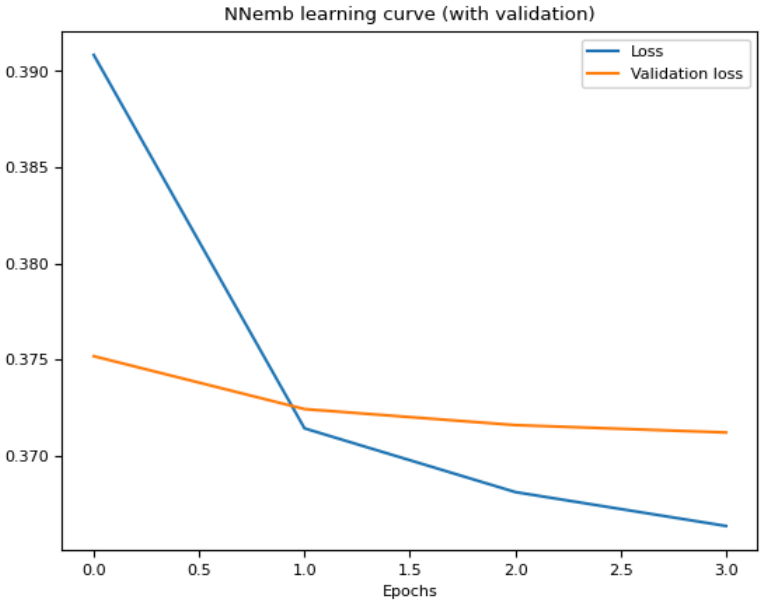
Zu b):

In [82]:
```
# Fitting
tic = time.time()

# Prepare the input data using a list comprehension
input_data = [xn_train[num_features]] + [np.asarray(xn_train[feature]) for feature in num_emb_features]

# Fit the model
fit = NNemb.fit(input_data, np.asarray(yn_train), epochs=4, verbose=0, validation_split=0.2)

runtime_nnemb = time.time() - tic

print("Dauer NNemb (sec):" + "%6.0f" % (runtime_nnemb))
```

```
Dauer NNemb (sec):    10
```

In [83]:
```
try:
    plot = pd.DataFrame.from_dict({"Loss": fit.history["loss"],
```

```
                                    "Validation loss": fit.history["val_loss"],
                                   })
    plot.index.name = "Epochs"
    plot = plot.plot(title="NNemb learning curve (with validation)")
except Exception:
    pass
```



Interpretation:

An der Graphik ist zu sehen, dass das Netzwerk bereits nach kurzer Zeit konvergiert. Daher kann davon ausgegangen werden, dass die verwendete Architektur für das Problem passend ist.

In [84]:
```
# Prepare the input data using a list comprehension
input_val_data = [xn_val[num_features]] + [np.asarray(xn_val[feature]) for feature in num_emb_features]

yn_val_pred = NNemb.predict(input_val_data)[:,0]
auc_NNemb = roc_auc_score(yn_val, yn_val_pred)

mname.append("NNemb")
mauc.append(auc_NNemb)

print('The validation AUC is: {:.6f}'.format(auc_NNemb))

# Prepare test data
input_test_data = [xn_test[num_features]] + [np.asarray(xn_test[feature]) for feature in num_emb_features]
yn_test_pred = NNemb.predict(input_test_data)[:,0]
```

```
224/224 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
The validation AUC is: 0.684513
224/224 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

Zu c):

In [85]:
```
# Modell nochmals anlegen und kompilieren

# Create a copy of the test model (with freshly initialized weights) and compile.
NNembC = tf.keras.models.clone_model(NNemb)
NNembC.compile(optimizer='adam', loss='binary_crossentropy', metrics=['AUC'])

# Fitting
fitC = NNembC.fit(input_data, np.asarray(yn_train), epochs=4, verbose = 0, validation_split = 0.2)

yn_val_predC = NNembC.predict(input_val_data)[:,0]
auc_NNembC = roc_auc_score(yn_val, yn_val_predC)

mname.append("NNembC")
mauc.append(auc_NNembC)

print('The validation AUC is: {:.6f}'.format(auc_NNembC))
```

```
224/224 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
The validation AUC is: 0.683469
```

zu d):

In [86]:
```
# Definiere Funktion, die die 2-dim. Embedding-Gewichte einer spezifischen Schicht mit Labels in einen DataFrame schreibt
def weights_emb(prefix, feature, model, layer):
    name = f"wgt_{prefix}_{feature}"
    # Get the specified layer from the model and create a DataFrame from its weights
    globals()[name] = model.layers[layer]
    globals()[name] = pd.DataFrame(globals()[name].get_weights()[0], columns=["dim_0", "dim_1"])
    # Group by `n_um_<feature>`, take the last value of `<feature>`, and assign it to the DataFrame
    globals()[name][feature] = df_diag.groupby(f"num_{feature}")[feature].last().values
    globals()[name] = globals()[name].set_index(feature)
```

In [87]:
```
# Funktion für alle Embedding-Merkmale (inkl. clone) anwenden
for idx, feature in enumerate(emb_features, start=6):
    weights_emb('NNemb', feature, NNemb, idx)
    weights_emb('NNembC', feature, NNembC, idx)

# Embeddinggewichte für ein Merkmal (transponiert) anzeigen
wgt_NNembC_mspec.T
```

Out[87]:

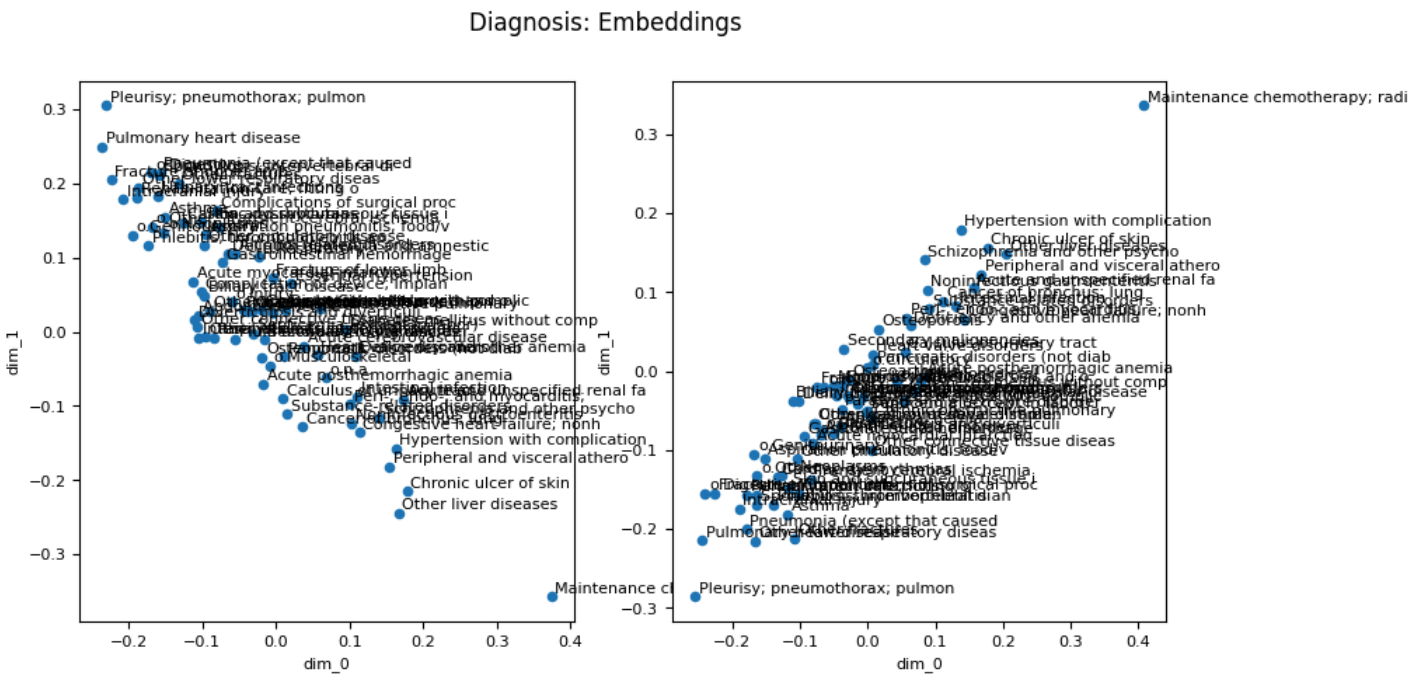
mspec	Cardiology	Surgery-Neuro	InternalMedicine	Orthopedics-Reconstructive	Surgery-General	Surgery-Cardiovascular/Thoracic	Emergency/Trauma	Nephrology	Family/GeneralPractice
dim_0	0.099369	0.273834	0.088721	0.304053	0.052265	0.178722	0.046021	-0.061752	-0.054524
dim_1	-0.174990	-0.290567	0.023822	-0.305305	-0.041070	-0.146024	0.009892	0.177952	0.075656

In [88]:
```
# Definiere Funktion um zwei Embeddings nebeneinander zu plotten
def plot_embeddings(df1, df2,text):
    # Set up subplots
    fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
    fig.suptitle(f"{text}: Embeddings", fontsize=12)

    # Plot the first DataFrame
    df1.plot(ax=ax1, kind="scatter", x="dim_0", y="dim_1")
    def annotate1(row, ax=ax1):
        ax.annotate(row.name, row.values[:2], xytext=(2, 2), textcoords="offset points", size=8)
    df1.apply(annotate1, axis="columns")

    # Plot the second DataFrame
    df2.plot(ax=ax2, kind="scatter", x="dim_0", y="dim_1")
    def annotate2(row, ax=ax2):
        ax.annotate(row.name, row.values[:2], xytext=(2, 2), textcoords="offset points", size=8)
    df2.apply(annotate2, axis="columns")

    plt.show()
```

Zweidimensionale Embeddings plotten und analysieren:

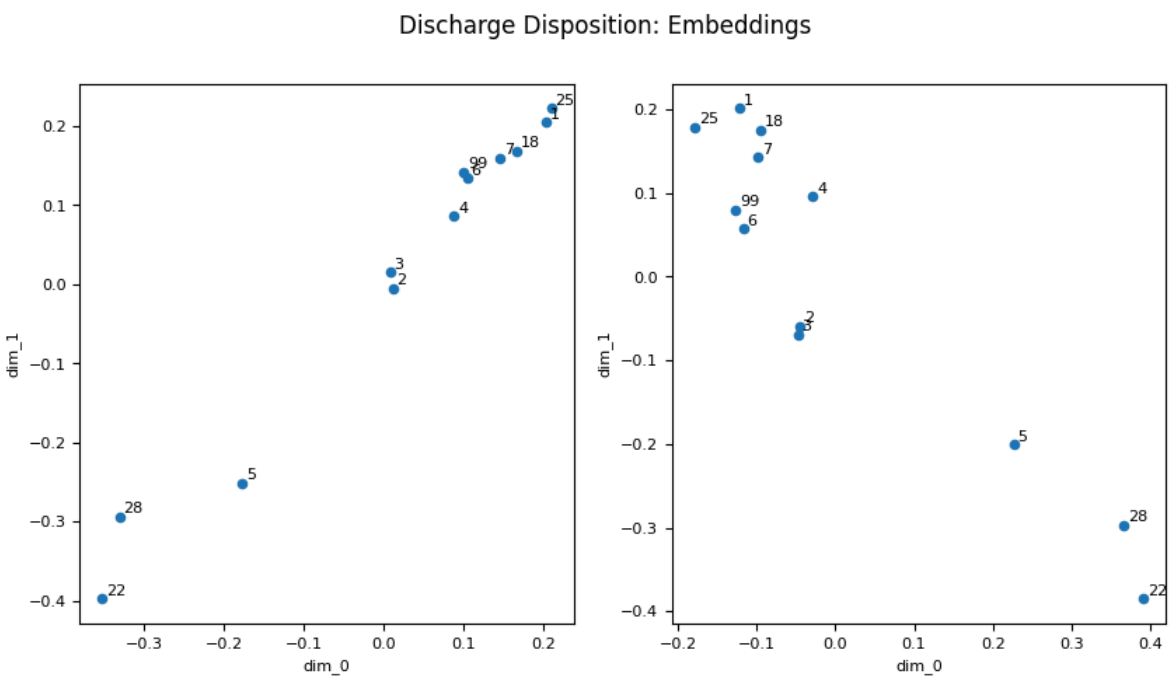
1. Diagnose-Embedding

In [89]:
```
plot_embeddings(wgt_NNemb_hdiag, wgt_NNembC_hdiag, "Diagnosis")
```



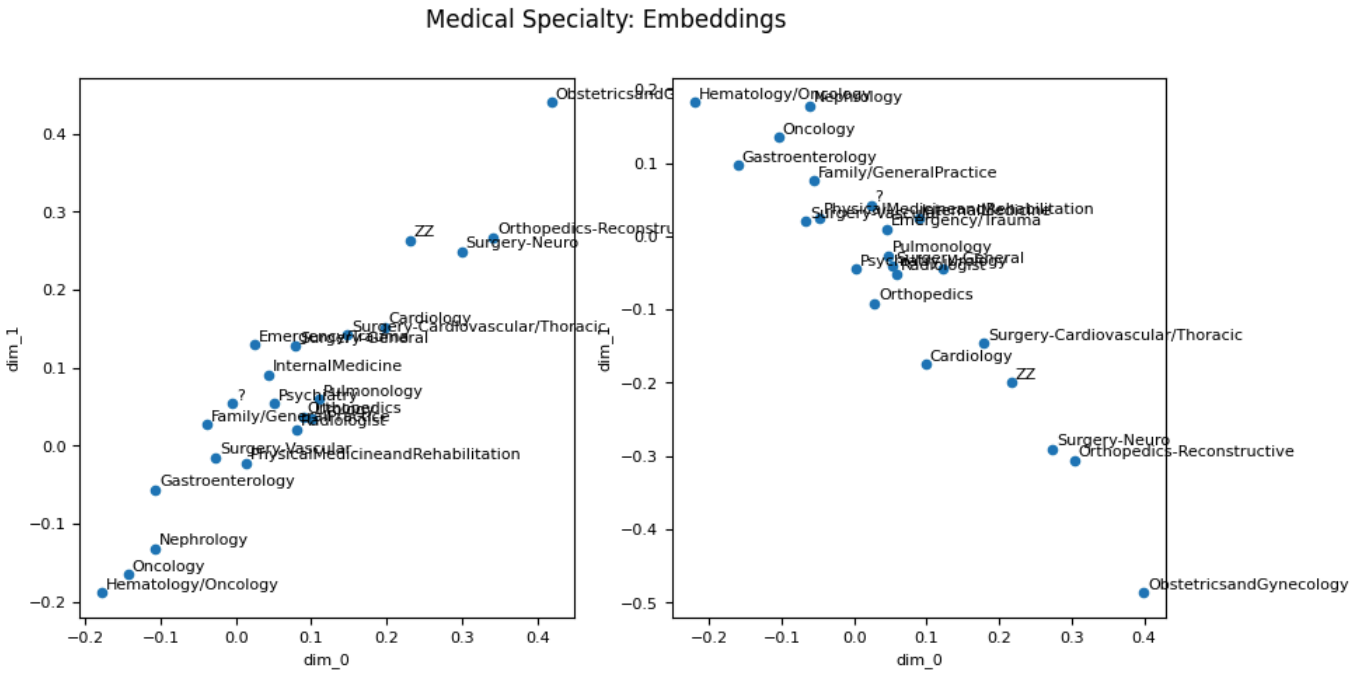
2. Discharge Disposition

In [90]:
```
plot_embeddings(wgt_NNemb_ddisp,wgt_NNembC_ddisp, "Discharge Disposition")
```



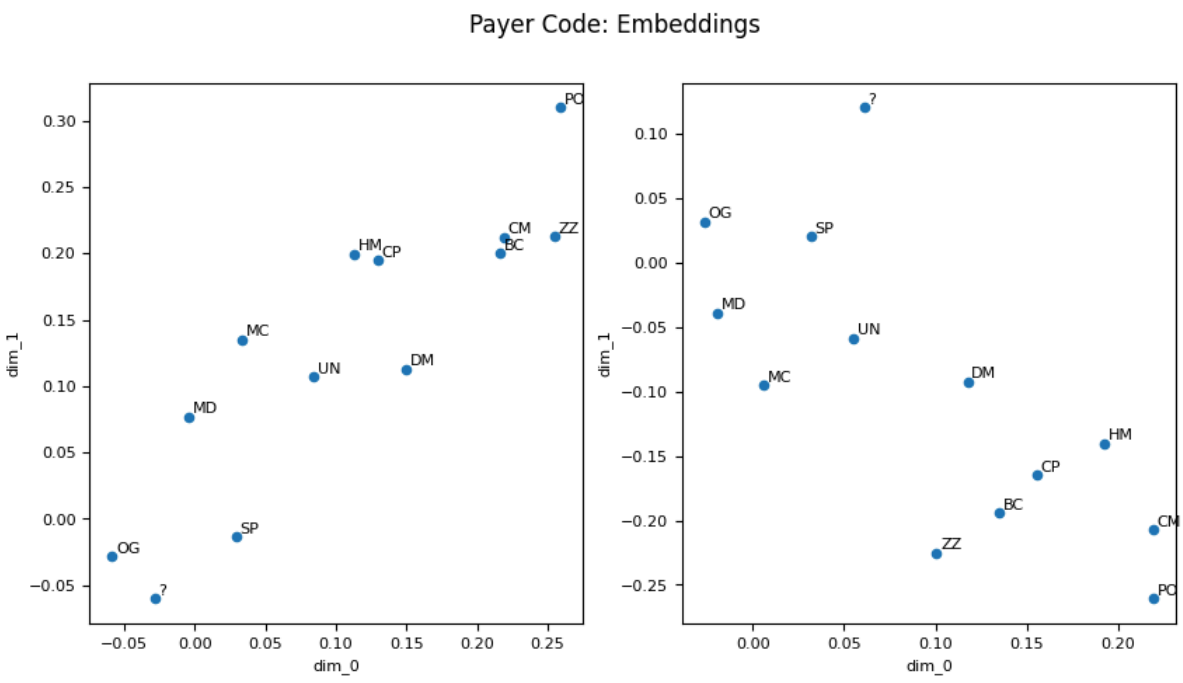
3. Medical Specialty

```
In [91]: plot_embeddings(wgt_NNemb_mspec, wgt_NNembC_mspec, "Medical Specialty")
```



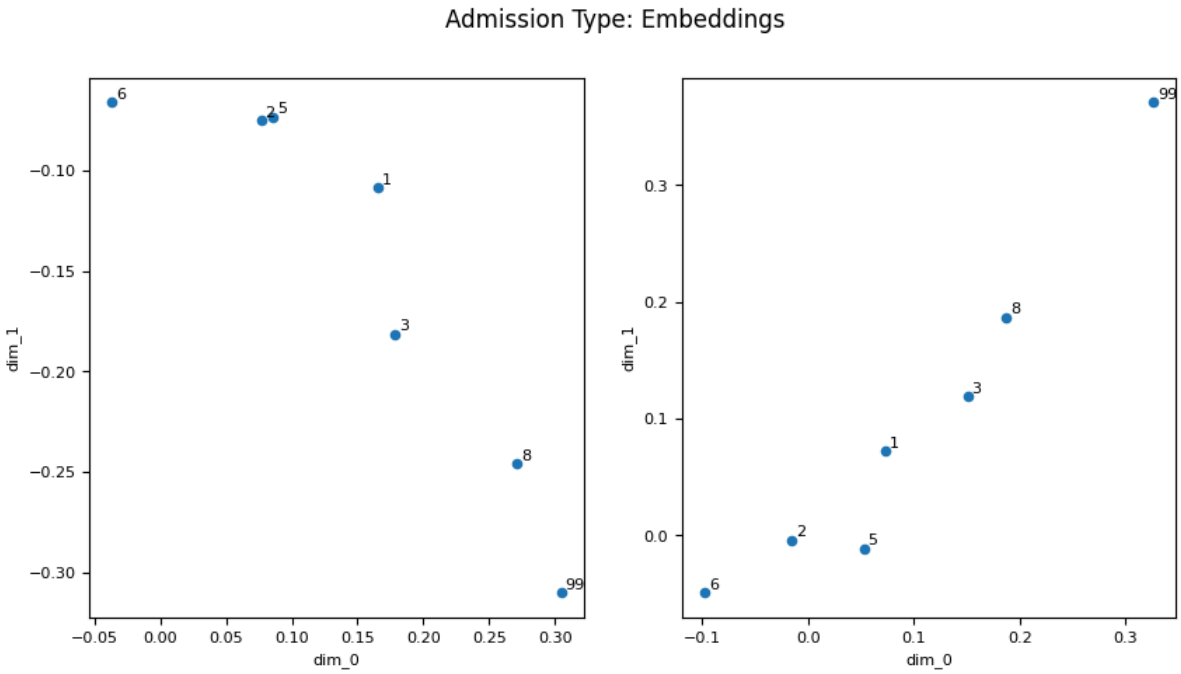
4. Payer Code

```
In [92]: plot_embeddings(wgt_NNemb_pcode, wgt_NNembC_pcode, "Payer Code")
```



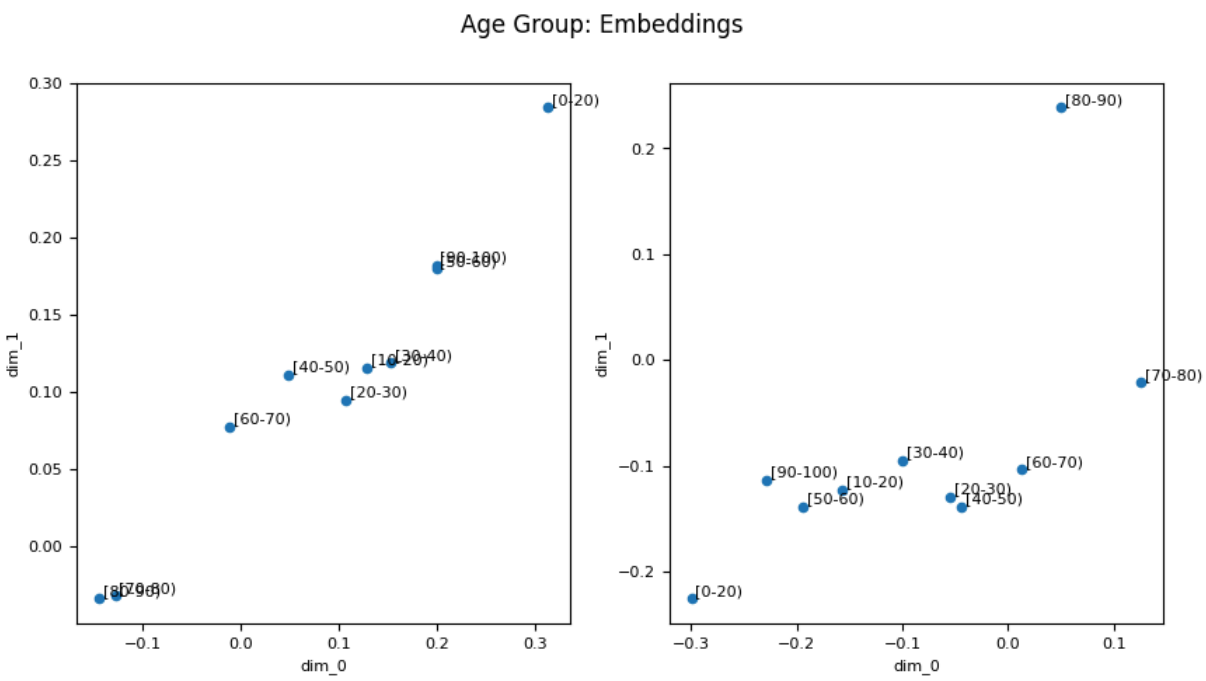
5. Admission Type

In [93]:
```
plot_embeddings(wgt_NNemb_atype, wgt_NNembC_atype, "Admission Type")
```



6. Age Group

In [94]:
```
plot_embeddings(wgt_NNemb_agegr, wgt_NNembC_agegr, "Age Group")
```



Zur Beurteilung der Stabilität der Embeddings werden diese paarweise vergleichen. Da sich die Netzwerkarchitektur (durch das Kloning) nicht verändert hat, wird Stabilität wie folgt bewertet: Im paarweise Vergleich können Unterschiede auftreten (es kann nicht erwartet werden, dass die Bilder identisch sind). Kleinere Abweichungen sowie insbesondere Drehungen und Spiegelungen sind zu erwarten. Deutlich davon abweichende Beobachtungen sprechen gegen eine Stabilität der Embeddings.

1. Diagnose-Embidding (hdiag): Wie bereits in Teil PT4-2 b) erwähnt zeigt sich auch bei den Embeddings der deutliche Unterschied beim Merkmal `Maintenance chemotherapy` in beiden Graphiken. Ansonsten sind sowohl Drehung als auch Spiegelung zu beobachten.
2. Discharge Disposition Embedding: Eine deutliche Ähnlichkeit der beiden Embedding-Darstellungen ist erkennbar. Man kann grob drei Cluster an Datenpunkten erkennen: Die beiden Punkte 22 und 28, dazu isoliert die 5 und die restlichen Datenpunkte.
3. Medical Specialty Embedding: Auch hier zeigen sich deutliche Ähnlichkeiten der beiden Bilder. Auffallend ist das Merkmal `Obstetrics and Gynecology` (Geburtshilfe und Gynäkologie), welches einen deutlichen Abstand zu den restlichen Datenpunkten aufweist.

Ähnliche Beobachtungen gelten auch für die Bilder der Embeddings von `payer code`, `Admission Type` und `Age Group`. Daher kann von einer Stabilität der Embeddings ausgegangen werden.

Aufgabe PT6: Embeddings joinen und bei Modellierung verwenden

Aufgabe PT6-1: [Lernziel 5.1; 15 Punkte]

Mit den erzeugten Embeddings in PT5 soll untersucht werden, inwieweit die Verwendung dieser (anstatt der ursprünglichen kategoriellen Variablen) Auswirkungen auf die Modellierungsergebnisse (d.h. den AUC-Score) hat.

a) Erweitern Sie den Datensatz aus Aufgabenteil PT5-1 a) um die entsprechenden Ergebnisse der Embeddings. Entfernen Sie nicht benötigte Features aus dem Datensatz. Dieser Datensatz ist im Anschluss in die Teile Training (70%), Validierung (15%) und Test (15%) zu trennen.

b) Aufbauend auf dem Datensatz in a): Führen Sie eine logistische Regression unter Berücksichtigung der Embedding-Merkmale durch. Orientieren Sie sich dabei an dem Modell aus Abschnitt 2.2. Was kann im Vergleich hierzu gesagt werden?

c) Analog zu Teil b): Führen Sie CatBoost unter Berücksichtigung der Embedding-Merkmale durch. Was kann im Vergleich zum Ergebnis in Abschnitt 1.5 (Baseline-Modell) gesagt werden?

d) Bewerten Sie abschließend die Wirkung der Embeddings in den betrachteten Modellen. Nutzen Sie dafür ein Balkendiagramm, welches die bisherigen Modelle (d.h. insbesondere das erste berechnet CatBoost Modell, die log. Regression sowie die Modelle aus dem One-Hot Encoding und Subsampling) mit den Validierungsergebnissen zeigt. Die Bewertung sollte unter anderem auch eine Aussage zur Nützlichkeit sowie Ihre eigene Empfehlung enthalten.

Lösungsvorschlag:

Zu a):

In [95]:
```
# Ausgangsdatensatz zur Ergänzung um Embedding-Merkmale
df_diag.head()
```

Out[95]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_specialty
0	Caucasian	Female	[50-60)	?	2	1	1	8	?	Cardiology
1	Caucasian	Female	[50-60)	?	3	1	1	2	?	Surgery-Neuro
2	Caucasian	Female	[80-90)	?	1	3	7	4	MC	InternalMedicine
3	Caucasian	Female	[80-90)	?	1	1	7	3	?	InternalMedicine
4	AfricanAmerican	Female	[30-40)	?	1	1	7	5	?	InternalMedicine

In [96]:
```
# Embedding dim0 und dim1 mit Merkmals-Prefix versehen (ggf. via Liste umsetzen, s.o.)
wgt_NNemb_hdiag = wgt_NNemb_hdiag.add_prefix('hdiag_')
wgt_NNemb_ddisp = wgt_NNemb_ddisp.add_prefix('ddisp_')
wgt_NNemb_mspec = wgt_NNemb_mspec.add_prefix('mspec_')
wgt_NNemb_pcode = wgt_NNemb_pcode.add_prefix('pcode_')
wgt_NNemb_atype = wgt_NNemb_atype.add_prefix('atype_')
wgt_NNemb_agegr = wgt_NNemb_agegr.add_prefix('agegr_')
```

In [97]:
```
# Aussehen der Embedding-Ergebnisse (beispielhaft):
wgt_NNemb_hdiag.head()
```

Out[97]:

hdiag	hdiag_dim_0	hdiag_dim_1
Essential hypertension	0.020485	0.064476
Spondylosis; intervertebral di	-0.157980	0.211726
Fracture of neck of femur (hip	-0.044707	0.028998
o.Other	-0.168215	0.141595
Urinary tract infections	-0.160506	0.183398

In [98]:
```
# Embedding-Werte ergänzen
df_emb = df_diag.merge(wgt_NNemb_hdiag, on='hdiag', how='left')
df_emb = df_emb.merge(wgt_NNemb_ddisp, on='ddisp', how='left')
df_emb = df_emb.merge(wgt_NNemb_mspec, on='mspec', how='left')
df_emb = df_emb.merge(wgt_NNemb_pcode, on='pcode', how='left')
df_emb = df_emb.merge(wgt_NNemb_atype, on='atype', how='left')
df_emb = df_emb.merge(wgt_NNemb_agegr, on='agegr', how='left')
df_emb.head(3)
```

Out[98]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_specialty	num_
0	Caucasian	Female	[50-60)	?	2	1	1	8	?	Cardiology	
1	Caucasian	Female	[50-60)	?	3	1	1	2	?	Surgery-Neuro	
2	Caucasian	Female	[80-90)	?	1	3	7	4	MC	InternalMedicine	

In [99]:
```
# Entfernung nicht benötigter Merkmale
df_emb = df_emb.drop(columns=['admission_type_id', 'discharge_disposition_id', 'payer_code', 'medical_specialty',  'diag_1', 'diag_2', 'diag_
                              'group_diag_1', 'group_diag_2', 'group_diag_3', 'group', 'L2code', 'hdiag', 'category', 'sum_target_cg',
                              'ddisp', 'sum_target_dd', 'mspec', 'sum_target_ms', 'pcode', 'sum_target_pc', 'atype', 'sum_target_at',
                              'agegr', 'sum_target_age', 'num_hdiag', 'num_ddisp', 'num_mspec', 'num_pcode', 'num_atype', 'num_agegr'], axis=

df_emb.head()
```

Out[99]:

	race	gender	age	weight	admission_source_id	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpatier
0	Caucasian	Female	[50-60)	?	1	8	77	6	33	
1	Caucasian	Female	[50-60)	?	1	2	49	1	11	
2	Caucasian	Female	[80-90)	?	7	4	68	2	23	
3	Caucasian	Female	[80-90)	?	7	3	46	0	20	
4	AfricanAmerican	Female	[30-40)	?	7	5	49	0	5	

In [100…

```
# Generierung von Train-, Validierungs- und Testdatensatz
xe = df_emb.drop(columns=['TARGET'], axis=1)
ye = df_emb.TARGET
# Split dataset in 70% training, 15% validation and 15% test and devide into feature matrix x and label y
xe_train, xe_valtest, ye_train, ye_valtest = train_test_split(xe, ye, train_size=0.70, random_state=RANDOM_SEED)
xe_val, xe_test, ye_val, ye_test = train_test_split(xe_valtest, ye_valtest, test_size=0.50, random_state=RANDOM_SEED)
print('Sample sizes for training, validation and test: ', xe_train.shape, xe_val.shape, xe_test.shape)
```

```
Sample sizes for training, validation and test:  (33425, 52) (7163, 52) (7163, 52)
```

zu b):

In [101…

```
Xy_emb_train = xe_train.copy()
Xy_emb_train['TARGET'] = ye_train

LR_emb = smf.logit(formula="TARGET ~ number_inpatient + num_lab_procedures + hdiag_dim_0 + hdiag_dim_1 + ddisp_dim_0 + ddisp_dim_1 + mspec_di
                   data=Xy_emb_train).fit()

# Display a summary of the logistic regression model results
print(LR_emb.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.362732
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                 TARGET   No. Observations:                33425
Model:                          Logit   Df Residuals:                    33410
Method:                           MLE   Df Model:                           14
Date:                Tue, 14 Jan 2025   Pseudo R-squ.:                 0.06762
Time:                        17:06:23   Log-Likelihood:                -12124.
converged:                       True   LL-Null:                       -13004.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================================
                         coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------------------
Intercept             -1.3692      0.100    -13.667      0.000      -1.566      -1.173
number_inpatient       0.5626      0.026     22.023      0.000       0.513       0.613
num_lab_procedures     0.0030      0.001      3.404      0.001       0.001       0.005
hdiag_dim_0            0.8384      0.351      2.386      0.017       0.150       1.527
hdiag_dim_1           -1.6836      0.347     -4.858      0.000      -2.363      -1.004
ddisp_dim_0           -2.8379      1.151     -2.466      0.014      -5.094      -0.582
ddisp_dim_1           -0.0184      1.063     -0.017      0.986      -2.101       2.064
mspec_dim_0           -1.5127      0.448     -3.376      0.001      -2.391      -0.634
mspec_dim_1           -0.7748      0.545     -1.422      0.155      -1.843       0.293
pcode_dim_0           -1.1870      0.464     -2.560      0.010      -2.096      -0.278
pcode_dim_1           -0.4693      0.301     -1.559      0.119      -1.059       0.121
atype_dim_0           -0.9263      0.414     -2.238      0.025      -1.738      -0.115
atype_dim_1            0.0088      0.653      0.013      0.989      -1.271       1.288
agegr_dim_0           -0.7819      0.658     -1.189      0.234      -2.071       0.507
agegr_dim_1           -0.1026      1.017     -0.101      0.920      -2.096       1.891
==============================================================================================
```

In [102…

```
# Evaluate the performance of the logistic regression model on the validation data
# and calculate the Area Under the Curve (AUC) for the ROC
auc_LR_emb = roc_auc_score(ye_val, LR_emb.predict(xe_val))

mname.append("LR_emb")
mauc.append(auc_LR_emb)

# Print the validation AUC
print(f'The validation AUC of the logistic regression model is: {auc_LR_emb:.6f}')
```

```
The validation AUC of the logistic regression model is: 0.669898
```

In Aufgabe PT2 wurde eine logistische Regression durchgeführt, deren Merkmale sich an der Feature Importance von CatBoost orierntiert haben. Konkret wurden die fünf Merkmale `discharge_disposition_id`, `number_inpatient`, `admission_source_id`, `num_lab_procedures` und `admission_type_id`. Der Validation Score bei diesem Modell lag bei ca. 0.6612. Bei dem vorliegenden Modell werden aufgrund der Embeddings deutlich mehr Features verwendet (insgesamt 14). Allerdings ist nur ein sehr geringer Anstieg des zugehörigen AUC-Wertes zu beobachten.

zu c):

In [103…

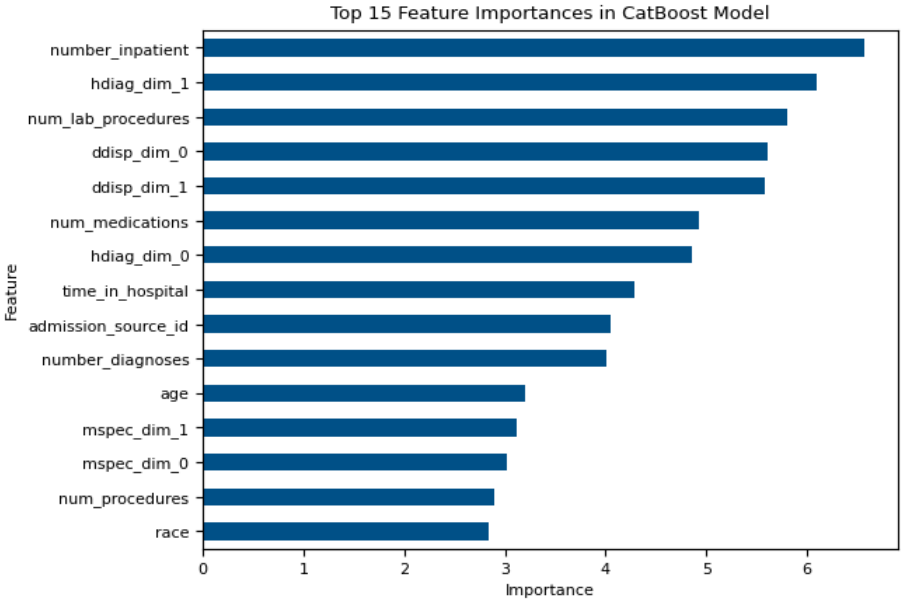
```
# Identify categorical features by data type 'object'
categorical_features_emb = list(xe_train.select_dtypes(include=['object']).columns)
```

```
# Start timer to calculate the running time of training the CatBoost model
start_time = time.time()

# Fit the CatBoostClassifier on the training data
CB_emb = CatBoostClassifier(eval_metric='AUC', random_seed=RANDOM_SEED)
CB_emb.fit(xe_train, ye_train, cat_features=categorical_features_emb, logging_level='Silent')

# Calculate and print the running time in seconds
elapsed_time_CB_emb = time.time() - start_time
print(f"Elapsed time (sec): {elapsed_time_CB_emb:.2f}")
```

```
Elapsed time (sec): 45.33
```

In [104…

```
# Extract the 15 most important features and create a Series for plotting
feature_importances = pd.Series(CB_emb.feature_importances_, index=xe_train.columns).nlargest(15)

# Plot the feature importances
feature_importances.plot(kind='barh', color=COLOR_DARK)
plt.gca().invert_yaxis()
plt.title('Top 15 Feature Importances in CatBoost Model')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



In [105…

```
# Calculate the AUC score using the model's prediction probabilities for the positive class
auc_CB_emb = roc_auc_score(ye_val, CB_emb.predict_proba(xe_val)[:, 1])

mname.append("CB_emb")
mauc.append(auc_CB_emb)

# Print the validation AUC
print(f'The validation AUC of the CatBoost model with standard hyperparameters is: {auc_CB_emb:.6f}')
```

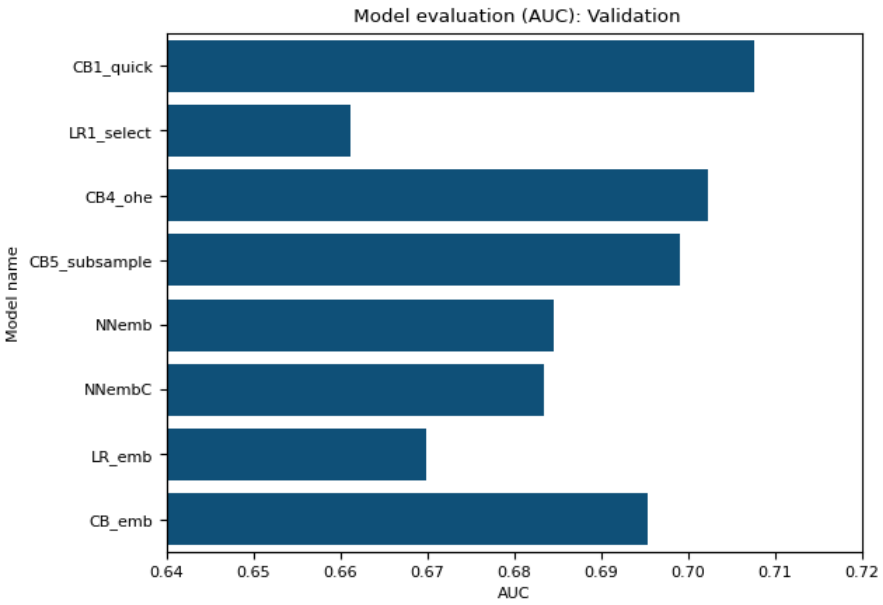
```
The validation AUC of the CatBoost model with standard hyperparameters is: 0.695381
```

Das Ergebnis von CatBoost unter Berücksichtigung der Embeddings liegt hinter dem Ergebnis des ersten "CB-quick" zurück. Bei der Trainingszeit liegen bei auf ungefähr gleichem Niveau.

zu d):

In [106…

```
plot_auc(dict,0.64,0.72, "Validation")
```



Das Bild zeigt, dass der erste durchgeführte CatBoost Algorithmus von den in diesem Notebook durchgeführten Modellen, gemessen am AUC, nicht überboten werden konnte. Das war so nicht zu erwarten, da insbesondere über Hyperparameter-Tuning deutlich mehr Zeit (verglichen mit der Trainingszeit von CatBoost) investiert worden ist. Allerdings sind die Unterschiede, wie aus der Graphik zu sehen, minimal. Über Techniken wie Skalierung bzw. Codierung ist es möglich, die Trainingszeiten für die Baumverfahren deutlich (d.h. auf wenige Sekunden) zu senken, was bei größeren Datensätzen einen Vorteil bringen kann. Die logistischen Regressionen, die die geringste Trainingszeit aufweisen, liegen deutlich zurück. Die Hinzunahme von Embeddings bei dem durchgeführten CatBoost Verfahren bringt ähnlich gute Ergebnisse wie "CB-Quick", und hat zusätzlich den Vorteil das über die Visualisierung der Embeddings neue Muster bzw. Erkenntnisse über die Datenpunkte gewonnen werden kann.

Aufgabe PT7: AutoML sowie Modellbewertung und Anwendung

Aufgabe PT7-1: [Lernziel 5.1; 16 Punkte]

- a) Über AutoML oder ein beliebiges anderes, bisher nicht verwendetes Modell oder Modellensemble (z.B. Stacking, Blending) soll versucht werden, den AUC zu verbessern. Ein Beispiel für AutoML ist AutoGluon, siehe https://auto.gluon.ai/stable/index.html. Dabei ist, analog zum bisherigen Vorgehen, mit einem Trainings-, Validierungs- und Testset zu arbeiten und die Modellgüte zu bestimmen.
- b) Am Ende des Notebooks soll Abschnitt 11. vollständig angepasst werden und alles danach Folgende (12., Appendix) gelöscht werden. In 11.1 sollen alle in den vorangegangenen Abschnitten erstellten bzw. optimierten Modelle (ohne Search-CV-Einzelmodelle) bewertet werden. In 11.2 soll eine Auswahl von mindestens 6 Modellen ins Lift Chart einfließen und ein geeignetes Modell für die nachfolgenden Wahrscheinlichkeits- und Perzentilbetrachtungen ausgewählt werden.

Lösungsvorschlag:

Zu a):

In [107…
```
from autogluon.tabular import TabularPredictor
# save_path = 'C:/ml/model_data/AutoGluon'
save_path = '/teamspace/studios/this_studio/PK_VADS_IMMERSION/2025/1_Erster_Ansatz/AutoGluon'
```

In [108…
```
AG1 = TabularPredictor(label='TARGET', eval_metric='roc_auc', path=save_path, verbosity=0).fit(
    Xy_raw_train, presets='best_quality', time_limit=elapsed_time_CB1*5
)

print(AG1.feature_metadata)
```

```
2025-01-14 17:07:13,259 INFO worker.py:1810 -- Started a local Ray instance. View the dashboard at 127.0.0.1:8265
2025-01-14 17:11:26,049 ERROR worker.py:422 -- Unhandled error (suppress with 'RAY_IGNORE_UNHANDLED_ERRORS=1'): The worker died unexpectedly w
hile executing this task. Check python-core-worker-*.log files for more information.
2025-01-14 17:11:26,053 ERROR worker.py:422 -- Unhandled error (suppress with 'RAY_IGNORE_UNHANDLED_ERRORS=1'): The worker died unexpectedly w
hile executing this task. Check python-core-worker-*.log files for more information.
2025-01-14 17:11:26,058 ERROR worker.py:422 -- Unhandled error (suppress with 'RAY_IGNORE_UNHANDLED_ERRORS=1'): The worker died unexpectedly w
hile executing this task. Check python-core-worker-*.log files for more information.
('category', [])  : 26 | ['race', 'age', 'weight', 'admission_type_id', 'discharge_disposition_id', ...]
('int', [])       :  8 | ['time_in_hospital', 'num_lab_procedures', 'num_procedures', 'num_medications', 'number_outpatient', ...]
('int', ['bool']) :  8 | ['gender', 'chlorpropamide', 'tolbutamide', 'miglitol', 'tolazamide', ...]
```

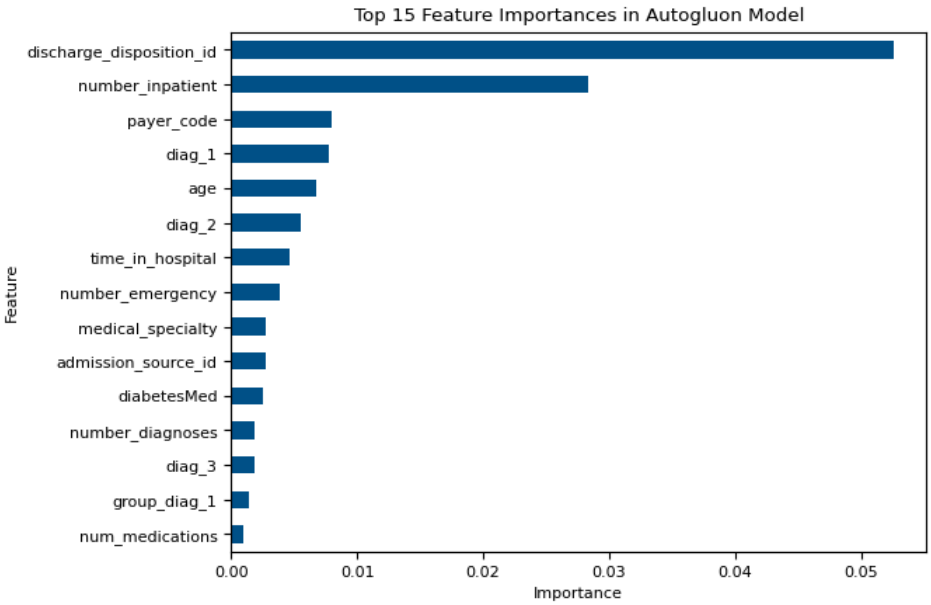
In [109…
```
fi_AG1 = AG1.feature_importance(Xy_raw_val)
fi_AG1.head(15)
```

Out[109…

	importance	stddev	p_value	n	p99_high	p99_low
discharge_disposition_id	0.052553	0.004280	0.000005	5	0.061367	0.043740
number_inpatient	0.028313	0.005016	0.000113	5	0.038640	0.017986
payer_code	0.007971	0.002712	0.001388	5	0.013556	0.002386
diag_1	0.007718	0.002183	0.000692	5	0.012212	0.003224
age	0.006789	0.002548	0.001992	5	0.012036	0.001543
diag_2	0.005571	0.001560	0.000667	5	0.008783	0.002359
time_in_hospital	0.004660	0.000663	0.000048	5	0.006025	0.003294
number_emergency	0.003831	0.001661	0.003354	5	0.007251	0.000411
medical_specialty	0.002744	0.003034	0.056610	5	0.008992	-0.003504
admission_source_id	0.002716	0.001600	0.009585	5	0.006010	-0.000578
diabetesMed	0.002553	0.004746	0.147697	5	0.012325	-0.007220
number_diagnoses	0.001904	0.001260	0.013884	5	0.004498	-0.000689
diag_3	0.001901	0.003349	0.136528	5	0.008796	-0.004993
group_diag_1	0.001362	0.001788	0.081772	5	0.005043	-0.002318
num_medications	0.000918	0.000679	0.019477	5	0.002316	-0.000479

In [110…

```
# Extract the 15 most important features and create a Series for plotting
feature_importances = pd.Series(fi_AG1.importance, index=X_raw_train.columns).nlargest(15)

# Plot the feature importances
feature_importances.plot(kind='barh', color=COLOR_DARK)
plt.gca().invert_yaxis()
plt.title('Top 15 Feature Importances in Autogluon Model')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



In [111…

```
AG1.leaderboard(Xy_raw_val)
```

Out[111…

	model	score_test	score_val	eval_metric	pred_time_test	pred_time_val	fit_time	pred_time_test_marginal	pred_time_val_marginal
0	WeightedEnsemble_L2	0.707573	0.705871	roc_auc	1.657650	5.327779	129.306103	0.002921	0.005486
1	WeightedEnsemble_L3	0.707516	0.705816	roc_auc	4.851315	9.586839	173.729563	0.004465	0.005346
2	CatBoost_BAG_L2	0.706983	0.703483	roc_auc	3.871634	8.236626	161.743931	0.176424	0.332731
3	LightGBMXT_BAG_L2	0.705698	0.699386	roc_auc	3.968258	8.323869	138.815352	0.273049	0.419973
4	CatBoost_BAG_L1	0.702294	0.698429	roc_auc	0.166325	0.379653	98.258661	0.166325	0.379653
5	LightGBMXT_BAG_L1	0.700023	0.700309	roc_auc	0.250850	0.491814	6.379445	0.250850	0.491814
6	RandomForestEntr_BAG_L2	0.694401	0.687717	roc_auc	4.670426	9.248762	142.442906	0.975217	1.344867
7	LightGBM_BAG_L2	0.693054	0.687949	roc_auc	3.941173	8.123795	139.200786	0.245964	0.219900
8	RandomForestGini_BAG_L1	0.689840	0.666698	roc_auc	0.330917	1.313724	6.154113	0.330917	1.313724
9	RandomForestGini_BAG_L2	0.689543	0.681757	roc_auc	4.672383	9.207322	142.686785	0.977173	1.303427
10	RandomForestEntr_BAG_L1	0.687699	0.674011	roc_auc	0.317233	1.300562	6.094200	0.317233	1.300562
11	ExtraTreesGini_BAG_L1	0.684561	0.664138	roc_auc	0.424174	1.422875	4.253977	0.424174	1.422875
12	LightGBM_BAG_L1	0.682677	0.682165	roc_auc	0.165229	0.413665	7.107664	0.165229	0.413665
13	ExtraTreesEntr_BAG_L1	0.679172	0.663207	roc_auc	1.719617	1.397880	3.680745	1.719617	1.397880
14	KNeighborsUnif_BAG_L1	0.554138	0.529792	roc_auc	0.140255	0.591957	0.105804	0.140255	0.591957
15	KNeighborsDist_BAG_L1	0.552965	0.533303	roc_auc	0.180609	0.591766	0.093476	0.180609	0.591766

In [112…

```
y_val_AG1 = AG1.predict_proba(X_raw_val).iloc[:, 1]
y_val_AG1.head(5)  # some example predictions
```

Out[112…

```
19120    0.140591
8718     0.144903
36646    0.530546
40896    0.075810
20211    0.218620
Name: 1, dtype: float64
```

In [113…

```
# Calculate the AUC score using the model's prediction probabilities for the positive class
auc_AG1 = roc_auc_score(y_val, y_val_AG1)

# Print the validation AUC
print(f'The validation AUC of AutoGluon is: {auc_AG1:.6f}')
```

```
The validation AUC of AutoGluon is: 0.707573
```

In [114…

```
AG1.evaluate(Xy_raw_val)
```

Out[114…

```
{'roc_auc': 0.7075725431737446,
 'accuracy': 0.8705849504397599,
 'balanced_accuracy': 0.5100758681760524,
 'mcc': 0.11103681531037865,
 'f1': 0.04136504653567735,
 'precision': 0.7407407407407407,
 'recall': 0.02127659574468085}
```

In [115…

```
AG1.evaluate(Xy_raw_test)
```

Out[115…

```
{'roc_auc': 0.708640046562208,
 'accuracy': 0.868770068407092,
 'balanced_accuracy': 0.5066837020359053,
 'mcc': 0.07996046579894195,
 'f1': 0.028925619834710745,
 'precision': 0.6086956521739131,
 'recall': 0.014814814814814815}
```

11. Modellbewertung und Anwendung

11.1 Bewertung des Modells: Validierung und Testsatz

Nun, da die Modellierung abgeschlossen ist, ist es an der Zeit, unsere Modelle anhand von Testdaten zu bewerten und die Ergebnisse mit den Validierungsdaten und zwischen den verschiedenen Modellen von Interesse zu vergleichen. Wir betrachten alle abgestimmten Modelle.

In [116…

```
# Calulate model AUC based on test data
auc_CB1_test = roc_auc_score(y_test, CB1.predict_proba(X_raw_test)[:, 1])
auc_LR1_test = roc_auc_score(y_test, LR1.predict(X_pre_test))

# Feature scaling using StandardScaler based on training data distributions (6.1)
# X_test = pd.DataFrame(scaler.transform(X_test), columns = feature_names)
auc_CB4_test = roc_auc_score(y_test, CB4.predict_proba(X_test)[:,1])
auc_CB5_test = roc_auc_score(y_test, CB4.predict_proba(X_test)[:,1])
auc_CB6_test = roc_auc_score(y_test, CB6.predict_proba(X_test)[:,1])
auc_LGB_test = roc_auc_score(y_test, LGB.predict_proba(X_test)[:,1])
auc_XGB_test = roc_auc_score(y_test, XGB.predict_proba(X_test)[:,1])

# NNemb, NNembC
```

```
input_test_data = [xn_test[num_features]] + [np.asarray(xn_test[feature]) for feature in num_emb_features]
y_test_NNemb = NNemb.predict(input_test_data)[:,0]
auc_NNemb_test = roc_auc_score(y_test, y_test_NNemb)
y_test_NNembC = NNembC.predict(input_test_data)[:,0]
auc_NNembC_test = roc_auc_score(y_test, y_test_NNembC)

# auc_NNemb_test = roc_auc_score(yn_test, yn_test_pred)
auc_LR_emb_test = roc_auc_score(ye_test, LR_emb.predict(xe_test))
auc_CB_emb_test = roc_auc_score(ye_test, CB_emb.predict_proba(xe_test)[:, 1])
auc_AG1_test = roc_auc_score(y_test, AG1.predict_proba(X_raw_test).iloc[:, 1])
```

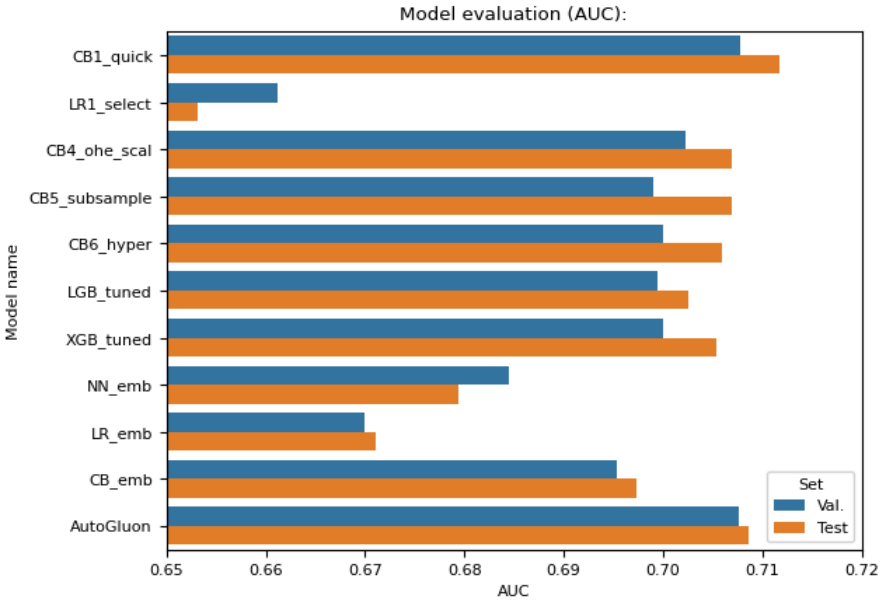
```
224/224 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
224/224 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

In [117…

```
# Data structures for model names, val/test-sample and AUC
mdict = {'Model name': [], 'Set': [], 'AUC': []}

# List of models and their AUC scores on validation and test sets
models_auc = [
    ("CB1_quick",       auc_CB1, auc_CB1_test),
    ("LR1_select",      auc_LR1, auc_LR1_test),
    ("CB4_ohe_scal",    auc_CB4, auc_CB4_test),
    ("CB5_subsample",   auc_CB5, auc_CB5_test),
    ("CB6_hyper",       auc_CB6, auc_CB6_test),
    ("LGB_tuned",       auc_LGB, auc_LGB_test),
    ("XGB_tuned",       auc_XGB, auc_XGB_test),
    ("NN_emb",          auc_NNemb, auc_NNemb_test),
    ("LR_emb",          auc_LR_emb, auc_LR_emb_test),
    ("CB_emb",          auc_CB_emb, auc_CB_emb_test),
    ("AutoGluon",       auc_AG1, auc_AG1_test),
]

# Populate the dictionary by looping over the models and their AUC scores
for model_name, auc_val, auc_test in models_auc:
    mdict['Model name'].extend([model_name] * 2)  # Model name twice for Val. and Test
    mdict['Set'].extend(["Val.", "Test"])
    mdict['AUC'].extend([auc_val, auc_test])
```

In [118…

```
df_eval = pd.DataFrame(mdict)
plt.title("Model evaluation (AUC):")
sns.barplot(data=df_eval, x="AUC", y="Model name",hue="Set")
plt.xlim(0.65, 0.72)
plt.show()
```



Im Gegensatz zum Notebook CSN schneiden nicht alle Modelle auf den Testdaten besser ab. Auch hier ist zu sehen, dass das "Quick-CB" Modell bereits sehr gute AUC Scores auf dem Trainings- und Testdatensatz erreicht. Weitere Einzelverfahren, wie XGB oder LGMB kommen zwar in die Nähe, können diese Werte allerdings (auch unter Einsatz von Hyperparametertuning) nicht erreichen. Um einen höheren AUC-Wert als "Quick-CB" zu erreichen, sind weitere Anstrengungen wie AutoML (hier: AutoGluon) nötig, wobei keine Einzelmodelle, sondern Kombinationen von Modellen zum Einsatz kommen.

11.2 High risk prediction

In Unterabschnitt 11.1 haben wir das Modell mit der höchsten Fläche unter der Receiver-Operating-Characteristic-Kurve (AUC) als das beste Modell auf der Grundlage aller Wiedereinweisungen von geringem bis hohem Risiko betrachtet. Da dem Hochrisikobereich aus gesundheitlichen Gründen in der Regel die größte Aufmerksamkeit zuteil wird, ist die Leistung der Modelle in diesem Bereich für die Anwendung entscheidender. Für diese Auswertung eignet sich das Lift-Chart. Ein kumulatives Lift-Chart zeigt die Verbesserung, die ein Modell im Vergleich zu einer zufälligen Schätzung bietet und misst die Veränderung in Form eines Lift-Scores.

In [119…

```
# Make preditions for test data
y_test_LR1 = LR1.predict(X_pre_test)
y_test_CB1 = CB1.predict_proba(X_raw_test)[:, 1]
y_test_AG1 = AG1.predict_proba(X_raw_test).iloc[:, 1]
```

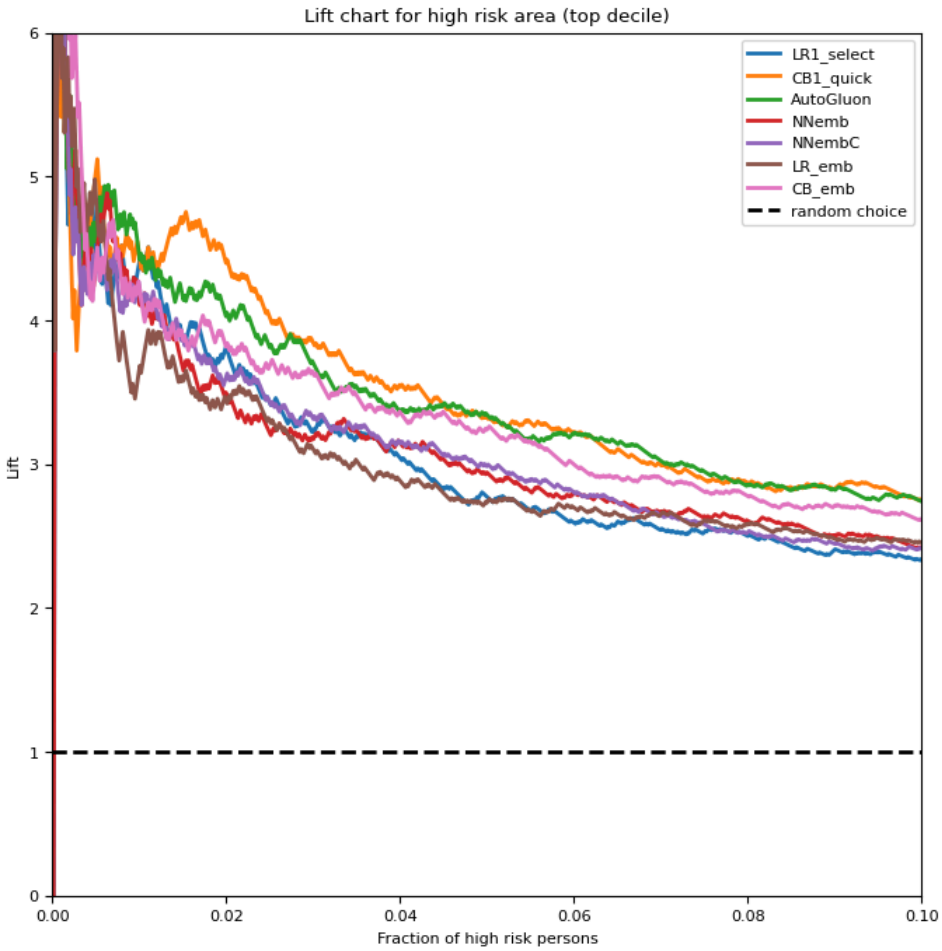
```
y_test_LR_emb = LR_emb.predict(xe_test)
y_test_CB_emb = CB_emb.predict_proba(xe_test)[:, 1]
```

In [120…

```
# Core of scikit-plot function cumulative_gain_curve (necessary, since scypi release 1.12.0 broke compatibility with scikit-plot version 0.3.
# Source: https://github.com/reiinakano/scikit-plot/blob/26007fbf9f05e915bd0f6acb86850b01b00944cf/scikitplot/helpers.py
def cumulative_gain_curve1(y_true, y_score):
    """This binary classification function generates the points necessary to plot the Cumulative Gain"""
    y_true, y_score = np.asarray(y_true), np.asarray(y_score)
    # make y_true a boolean vector
    y_true = (y_true == 1)
    sorted_indices = np.argsort(y_score)[::-1]
    y_true = y_true[sorted_indices]
    gains = np.cumsum(y_true)
    percentages = np.arange(start=1, stop=len(y_true) + 1)
    gains = gains / float(np.sum(y_true))
    percentages = percentages / float(len(y_true))
    gains = np.insert(gains, 0, [0])
    percentages = np.insert(percentages, 0, [0])
    return percentages, gains

# Calculate cumulative gains for lift chart
def calculate_gains_and_percentages(model_predictions, y_true):
    """Calculate gains and percentages for a given model's predictions."""
#    percentages, gains = cumulative_gain_curve1(y_true, model_predictions[:, 1])
    percentages, gains = cumulative_gain_curve1(y_true, model_predictions)
    gains_adjusted = gains[1:] / percentages[1:]  # Adjust gains starting from the second element
    return percentages[1:], gains_adjusted
```

In [121…

```
# Select models and prepare true and predicted values
y_true = np.array(np.ravel(y_test))
model_predictions = {
    'LR1_select': np.array(y_test_LR1),
    'CB1_quick': np.array(y_test_CB1),
    'AutoGluon': np.array(y_test_AG1),
    'NNemb': np.array(y_test_NNemb),
    'NNembC': np.array(y_test_NNembC),
    'LR_emb': np.array(y_test_LR_emb),
    'CB_emb': np.array(y_test_CB_emb)
}

# Initialize lift chart
fig, ax = plt.subplots(figsize=(8, 8))
ax.set_title("Lift chart for high risk area (top decile)")
ax.set_xlabel('Fraction of high risk persons')
ax.set_ylabel('Lift')
plt.xlim(0, 0.1)
plt.ylim(0, 6.0)

# Calculate and plot lift score for each model
for label, predictions in model_predictions.items():
    percentages, gains_adjusted = calculate_gains_and_percentages(predictions, y_true)
    ax.plot(percentages, gains_adjusted, lw=2, label=label)

# Plot random choice baseline
ax.plot([0, 1], [1, 1], 'k--', lw=2, label='random choice')
ax.legend(loc='upper right')
plt.show()
```



Das kumulative Lift-Diagramm zeigt den Faktor (Lift-Score), um den das Modell die „Trefferquote“ verbessern kann. In unserem Fall gibt es kein eindeutiges Gewinnermodell für das oberste Dezil. Alle Gradient-Tree-Boosting-Modelle haben einen ähnlichen Lift-Score. Daher können wir unser erstes „schnelles und einfaches“ Basismodell für die Definition des oberen Risikobereichs verwenden. Im Folgenden konzentrieren wir uns auf das oberste Dezil und erstellen eine Liste der Personen, bei denen eine hohe Wahrscheinlichkeit einer Wiedereinweisung vorliegt. Da wir den wahren Wert für unsere Testdaten kennen, vergleichen wir ihn mit den vorhergesagten Werten.

In [122…

```
# Merge predicted an true values into a DataFrame
df = pd.DataFrame({
    'PredProbability': y_test_CB1,
    'TrueValue': y_test
})

# Sort by Probability in descending order
df_sorted = df.sort_values(by='PredProbability', ascending=False)
df_sorted.head()
```

Out[122…

	PredProbability	TrueValue
34138	0.882116	1
45194	0.863774	1
33582	0.853748	1
33042	0.831136	1
20453	0.817978	0

In [123…

```
# Creating a 'Percentile' column based on quantile-based discretization into 100 bins
df_sorted['Percentile'] = pd.qcut(df_sorted['PredProbability'], 100, labels=range(1, 101))

# Calculate means of predicted and true values for the top 10 percentiles
percentile_means = df_sorted.groupby('Percentile', observed=True).mean().tail(10)

# Printing the results for the top 10 percentiles
print(percentile_means[::-1])  # This reverses the order to start with the highest probabilities
```

```
            PredProbability  TrueValue
Percentile
100                0.629849   0.583333
99                 0.474494   0.583333
98                 0.414685   0.352113
97                 0.374268   0.347222
96                 0.343633   0.319444
95                 0.317046   0.366197
94                 0.292885   0.208333
93                 0.271637   0.267606
92                 0.255991   0.333333
91                 0.244785   0.263889
```

Obwohl auch hier die vorhergesagten Wahrscheinlichkeiten etwas zu hoch erscheinen, funktioniert die Rangfolge der Perzentile sehr gut und kann dazu verwendet werden, über die zu ergreifenden Maßnahmen zu entscheiden.

In [124…

```
# Zeitmessung Notebook:
elapsed_time_nb = time.time() - start_time_nb
print(f"Elapsed time entire notebook (min): {elapsed_time_nb/60:.2f}")
```

```
Elapsed time entire notebook (min): 41.20
```

ADS Immersion 2025: Proposed Solution to Part I

Version 25.07.2025

Task R0: Libraries, Reproducibility, and Core Helper Functions

At this point, load all packages required for the subsequent tasks. Ensure that you load only those packages that are actually needed and used to maintain the efficiency and readability of the notebook. Additionally, ensure the reproducibility of the notebook. Define core helper functions as well; specifically, incorporate the functions `multiplot` and `get_binCI` from the *SWoF* template to generate multiple plots and compute binomial confidence intervals.

First, we load all required libraries in one place to support the further processing of the exam task.

```
# Data manipulation and management
library(dplyr)         # Intuitive grammar for data manipulation
library(data.table)    # High-performance data frames for fast manipulations
library(tidyr)         # Data cleaning and reshaping into tidy formats
library(readr)         # Efficient reading of tabular data (e.g., CSV files)
library(tibble)        # Data frames with enhanced usability
library(stringr)       # String processing with consistent functions
library(forcats)       # Easy handling of factor/categorical variables
library(here)          # Dynamic path handling relative to the project root

# Visualization
library(ggplot2)       # Core package for creating versatile visualizations
library(scales)        # Helper functions for formatting and scaling in ggplot2
library(ggthemes)      # Additional aesthetic themes for ggplot2
library(gridExtra)     # Arrange multiple plots in a single layout
library(RColorBrewer)  # Color palettes for appealing visualizations
library(ggrepel)       # Avoid overlapping text labels in ggplot2
library(ggridges)      # Create ridgeline plots for distribution visualization
library(GGally)        # Extensions for ggplot2, e.g., correlation plots
library(grid)          # Low-level functions for custom layouts
library(corrplot)      # Visualization of correlation matrices
library(plotly)        # Create interactive and web-ready plots

# Exploratory data analysis and missing values
library(VIM)           # Visualization and imputation of missing values
library(hstats)        # Interaction detection using Friedman's H-statistic

# Modeling and machine learning
library(xgboost)       # Fast and powerful gradient boosting
library(nnet)          # Multinomial logistic regression
library(glmnet)        # Regularized linear models such as Lasso and Ridge
library(mgcv)          # Generalized Additive Models (GAMs)
library(caret)         # Unified interface for ML model training and tuning
library(MLmetrics)     # Evaluation metrics for machine learning models
library(pROC)          # Analysis and plotting of ROC curves

# Interactive data
library(DT)            # Interactive tabular data display and filtering
library(repr)          # Adjust plot sizes in Jupyter or similar environments
```

For the sake of notebook reproducibility, we set a seed and output the package versions used.

```
# Set a seed
set.seed(42)

# Output the used package versions
sessionInfo()
```

```
## R version 4.4.1 (2024-06-14 ucrt)
## Platform: x86_64-w64-mingw32/x64
## Running under: Windows 11 x64 (build 26100)
##
## Matrix products: default
##
##
## locale:
## [1] LC_COLLATE=German_Germany.utf8  LC_CTYPE=German_Germany.utf8
## [3] LC_MONETARY=German_Germany.utf8 LC_NUMERIC=C
## [5] LC_TIME=German_Germany.utf8
##
## time zone: Europe/Berlin
## tzcode source: internal
##
## attached base packages:
## [1] grid      stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] repr_1.1.7         DT_0.33            pROC_1.18.5        MLmetrics_1.1.3
##  [5] caret_6.0-94       lattice_0.22-6     mgcv_1.9-1         nlme_3.1-164
##  [9] glmnet_4.1-8       Matrix_1.7-0       nnet_7.3-19        xgboost_1.7.8.1
## [13] hstats_1.2.1       VIM_6.2.2          colorspace_2.1-1   plotly_4.10.4
## [17] corrplot_0.94      GGally_2.2.1       ggridges_0.5.6     ggrepel_0.9.5
## [21] RColorBrewer_1.1-3 gridExtra_2.3      ggthemes_5.1.0     scales_1.3.0
## [25] ggplot2_3.5.1      here_1.0.1         forcats_1.0.0      stringr_1.5.1
## [29] tibble_3.2.1       readr_2.1.5        tidyr_1.3.1        data.table_1.15.4
## [33] dplyr_1.1.4
##
## loaded via a namespace (and not attached):
##  [1] rlang_1.1.4          magrittr_2.0.3       e1071_1.7-14
##  [4] compiler_4.4.1       reshape2_1.4.4       vctrs_0.6.5
##  [7] pkgconfig_2.0.3      shape_1.4.6.1        fastmap_1.2.0
## [10] utf8_1.2.4           rmarkdown_2.28       prodlim_2024.06.25
## [13] tzdb_0.4.0           purrr_1.0.2          xfun_0.47
## [16] cachem_1.1.0         jsonlite_1.8.8       recipes_1.1.0
## [19] parallel_4.4.1       R6_2.5.1             bslib_0.8.0
## [22] stringi_1.8.4        vcd_1.4-12           ranger_0.16.0
## [25] parallelly_1.38.0    car_3.1-2            boot_1.3-30
## [28] rpart_4.1.23         lubridate_1.9.3      lmtest_0.9-40
## [31] jquerylib_0.1.4      Rcpp_1.0.13          iterators_1.0.14
## [34] knitr_1.48           future.apply_1.11.2  zoo_1.8-12
## [37] base64enc_0.1-3      timechange_0.3.0     splines_4.4.1
## [40] tidyselect_1.2.1     rstudioapi_0.17.1    abind_1.4-5
## [43] yaml_2.3.10          timeDate_4032.109    codetools_0.2-20
## [46] listenv_0.9.1        plyr_1.8.9           withr_3.0.1
## [49] evaluate_0.24.0      future_1.34.0        survival_3.6-4
## [52] ggstats_0.7.0        proxy_0.4-27         pillar_1.9.0
## [55] carData_3.0-5        stats4_4.4.1         foreach_1.5.2
## [58] generics_0.1.3       rprojroot_2.0.4      sp_2.1-4
## [61] hms_1.1.3            munsell_0.5.1        laeken_0.5.3
## [64] globals_0.16.3       class_7.3-22         glue_1.7.0
## [67] lazyeval_0.2.2       tools_4.4.1          robustbase_0.99-4
## [70] ModelMetrics_1.2.2.2 gower_1.0.1          ipred_0.9-15
## [73] cli_3.6.3            fansi_1.0.6          viridisLite_0.4.2
## [76] lava_1.8.0           gtable_0.3.5         DEoptimR_1.1-3
## [79] sass_0.4.9           digest_0.6.37        htmlwidgets_1.6.4
## [82] htmltools_0.5.8.1    lifecycle_1.0.4      hardhat_1.4.0
## [85] httr_1.4.7           MASS_7.3-60.2
```

As required in the task description, we include the functions `multiplot` and `get_binCI` from the *SWoF* template.

```r
# Define multiple plot function
#
# ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objects)
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

 if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}

# function to extract binomial confidence levels
get_binCI <- function(x,n) as.list(setNames(binom.test(x,n)$conf.int, c("lwr", "upr")))
```

We also define the functions `plot_confusion_matrix` and `calculate_auc`. The `calculate_auc` function is used to compute the AUC (Area Under the ROC Curve) and allows flexible application in different scenarios. The `plot_confusion_matrix` function will be used in tasks R4 and R6 to visualize confusion matrices.

```r
# Compute AUC (Area Under the Curve)
calculate_auc <- function(true_labels, predicted_probs, positive_class = NULL, verbose = TRUE) {
  # If positive_class is provided, binarize the target variable
  if (!is.null(positive_class)) {
    true_labels <- ifelse(true_labels == positive_class, 1, 0)
  }

  # Compute ROC curve and AUC
  roc_curve <- roc(true_labels, predicted_probs)
  auc_value <- auc(roc_curve)

  # Optionally print the AUC value
  if (verbose) {
    cat(sprintf("AUC: %.4f\n", auc_value))
  }

  # Return ROC object and AUC value
  list(roc_curve = roc_curve, auc = auc_value)
}

# Visualization of confusion matrices
plot_confusion_matrix <- function(conf_matrix, title) {
  # Convert confusion matrix to data frame format
  cm_table <- as.data.frame(conf_matrix$table)
  colnames(cm_table) <- c("Prediction", "Reference", "Frequency")
  cm_table$FreqNorm <- cm_table$Frequency / sum(cm_table$Frequency)

  # Create a ggplot2 plot for the confusion matrix
  ggplot(cm_table, aes(x = Reference, y = Prediction, fill = FreqNorm)) +
    geom_tile(color = "black") + # Create tiles
    geom_text(aes(label = Frequency), size = 3) +
    scale_fill_gradient(low = "white", high = "blue", name = "Proportion") + # Color coding
    labs(title = title, x = "Reference", y = "Prediction") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5, size = 10, face = "bold"))
}
```

Task R1: Data Preprocessing

This task focuses on reading, filtering, preparing, and exporting data. The aim is to perform basic preprocessing steps to prepare the main dataset for use as the foundation for the subsequent tasks in Part I as well as Part II.

a) Reading Data: Load the file `diabetic_data.csv` from the examination materials. Ensure that character strings are read as factors (data type: `factor`) and that missing values are not explicitly set using a `na.strings` statement. Output the number of rows and columns of the imported dataset, print the first ten entries, and verify the file is correctly imported. Briefly discuss one advantage and one disadvantage of the above approach to handling missing values in this dataset. Finally, check whether the dataset contains not only the values `?` but also other "real" missing values (`NA`).

First, we read the file `diabetic_data.csv` using the `read.csv` function and convert it into a tibble using the `as_tibble` function. During import, we ensure that character strings are read as factors (`stringsAsFactors = TRUE`) and that no explicit `na.strings` instruction is used to define missing values.

```r
# Read data
data <- as_tibble(read.csv(here("2025/1_Erster_Ansatz", "diabetic_data.csv"), sep = ",", stringsAsFactors = TRUE))
```

Next, we print the number of rows and columns of the imported dataset and display the first ten entries.

```r
# Output number of rows and columns
cat("Number of rows in the dataset:", nrow(data),
    "\nNumber of columns in the dataset:", ncol(data))
```

```
## Number of rows in the dataset: 101766
## Number of columns in the dataset: 50
```

```r
# Show the first ten entries
head(data, n = 10)
```

encounter_id <int>	patient_nbr <int>	race <fct>	gender <fct>	age <fct>	weight <fct>	admission_type_id <int>
2278392	8222157	Caucasian	Female	[0-10)	?	6
149190	55629189	Caucasian	Female	[10-20)	?	1
64410	86047875	AfricanAmerican	Female	[20-30)	?	1
500364	82442376	Caucasian	Male	[30-40)	?	1
16680	42519267	Caucasian	Male	[40-50)	?	1

encounter_id <int>	patient_nbr <int>	race <fct>	gender <fct>	age <fct>	weight <fct>	admission_type_id <int>
35754	82637451	Caucasian	Male	[50-60)	?	2
55842	84259809	Caucasian	Male	[60-70)	?	3
63768	114882984	Caucasian	Male	[70-80)	?	1
12522	48330783	Caucasian	Female	[80-90)	?	2
15738	63555939	Caucasian	Female	[90-100)	?	3

1-10 of 10 rows | 1-7 of 50 columns

Based on the above inspection of the dataset, we see that the file was correctly imported: missing values are encoded using `?`, for example in the columns `weight` and `medical_specialty`. In addition, categorical features were correctly read as factors, and all numeric features were identified as integer values.

One advantage of the above approach to handling missing values in this dataset is that some machine learning algorithms cannot handle missing values. By encoding missing values as a separate category `?`, this problem is bypassed because missing values are treated as a distinct category and can be incorporated into such models without additional preprocessing. However, a disadvantage of this approach is that `?` may also be a valid value or an indication of "unknown". This may result in missing values being falsely equated with genuine `?` entries, potentially distorting the data analysis.

Finally, the following code confirms that beyond the `?` values, the dataset does not contain any additional "true" missing values (`NA`):

```
# Count actual missing values (NA) in the dataset
num_na <- sum(is.na(data))

# Output number of actual missing values (NA)
cat("Number of actual missing values (NA) in the dataset: ", num_na)
```

```
## Number of actual missing values (NA) in the dataset:  0
```

b) Filtering Data: Sort the data in ascending order, first by `patient_nbr`, then by `encounter_id`. Next, filter the dataset so that only the first hospitalization for each patient (= the lowest `encounter_id`) remains. Verify the correct execution of sorting and filtering by conducting an appropriate before-and-after comparison. Then, exclude all entries where the value of `discharge_disposition_id` is associated with death or hospice (values `11`, `13`, `14`, `19`, `20`, `21`). Remove the columns `encounter_id` and `patient_nbr`, as they are not required for further modeling. Create a frequency table of the `discharge_disposition_id` values and ensure that it matches the table in Appendix 2 of the exam. Finally, verify that the dataset now consists of 69,973 rows and 48 columns.

First, we sort the data in ascending order by `patient_nbr` and then by `encounter_id`. To enable a before-and-after comparison, we additionally display the first 12 rows of the sorted dataset.

```
# Sort the data in ascending order by patient_nbr and encounter_id
data <- data %>% arrange(patient_nbr, encounter_id)

# Show the first 12 entries of the sorted dataset
data %>% select(patient_nbr, encounter_id) %>% head(12)
```

patient_nbr <int>	encounter_id <int>
135	24437208
135	26264286
378	29758806
729	189899286
774	64331490
927	14824206
1152	8380170
1152	30180318
1152	55533660
1152	80742510
1152	83281464
1305	66197028

1-12 of 12 rows

Next, we filter the dataset to retain only the first hospitalization per patient and display the first seven rows of the resulting dataset.

```
# Keep only the first hospitalization per patient
data <- data %>% group_by(patient_nbr) %>% slice_head(n = 1) %>% ungroup()

# Show the first 7 entries of the filtered dataset
data %>% select(patient_nbr, encounter_id) %>% head(n = 7)
```

patient_nbr <int>	encounter_id <int>
135	24437208
378	29758806
729	189899286
774	64331490
927	14824206
1152	8380170
1305	66197028

7 rows

From the above outputs, we can confirm that both sorting and filtering have been correctly applied.

Next, we remove all entries where `discharge_disposition_id` indicates death or hospice. We also remove the `encounter_id` and `patient_nbr` columns since they are not needed for further modeling.

```
# Filter out cases related to death or hospice
data <- data %>% filter(!discharge_disposition_id %in% c(11, 13, 14, 19, 20, 21))

# Remove columns not needed for modeling
data <- data %>% select(-c(encounter_id, patient_nbr))
```

Now we create a frequency table of the values of `discharge_disposition_id` .

```
# Create frequency table of discharge_disposition_id values
frequency_count <- data %>%
  group_by(discharge_disposition_id) %>%
  summarize(Count = n(), .groups = 'drop') %>%
  arrange(discharge_disposition_id)

# Output the frequency table
frequency_count
```

discharge_disposition_id <int>	Count <int>
1	44317
2	1539
3	8784
4	541
5	913
6	8289
7	409
8	73
9	9
10	6
12	2
15	40
16	3
17	8
18	2474
22	1410
23	260
24	25
25	778
27	3
28	90

1-21 of 21 rows

A comparison with Appendix 2 shows that the values match exactly.

Finally, the following code verifies that the dataset now contains 69,973 rows and 48 columns.

```r
# Output the number of rows and columns
cat("Number of rows in the dataset:", nrow(data),
    "\nNumber of columns in the dataset:", ncol(data))
```

```
## Number of rows in the dataset: 69973
## Number of columns in the dataset: 48
```

c) Analyzing Diagnosis Data: The dataset contains the primary diagnosis `diag_1` as well as two secondary diagnoses `diag_2` and `diag_3`, each in the nominal ICD9 coding. For each diagnosis variable, determine the number of unique diagnoses, the most frequent diagnosis, and the number of diagnoses that occur fewer than ten times. Briefly discuss two challenges each that this relatively large number of categories can pose for data visualization and modeling.

The following code defines a function that, for a given diagnosis column, calculates the number of unique diagnoses, the most frequent diagnosis, and the number of diagnoses that occur fewer than ten times. This function is then applied to all three diagnosis variables, and the results are clearly presented in a summary table.

```r
# Function to analyze diagnosis columns
analyze_diagnosis <- function(column) {

  # 0. Remove levels with zero frequency
  column <- droplevels(column)

  # 1. Frequency of all actually present diagnoses
  freq <- table(column)

  # 2. Number of unique diagnoses (only those with freq > 0)
  num_unique_diagnoses <- length(freq)

  # 3. Most frequent diagnosis
  most_frequent <- names(freq)[which.max(freq)]

  # 4. Diagnoses occurring fewer than 10 times (but more than 0)
  less_than_10 <- sum(freq < 10)

  return(list(
    "Number of Diagnoses"     = num_unique_diagnoses,
    "Most Frequent Diagnosis" = most_frequent,
    "Rare Diagnoses Count"     = less_than_10
  ))
}

# Analyze each diagnosis column separately
diag_1_results <- analyze_diagnosis(data$diag_1)
diag_2_results <- analyze_diagnosis(data$diag_2)
diag_3_results <- analyze_diagnosis(data$diag_3)

# Create a data frame to store the results
results <- data.frame(
  Diagnosis_Variable = c("diag_1", "diag_2", "diag_3"),
  Number_of_Diagnoses = c(diag_1_results$`Number of Diagnoses`,
                          diag_2_results$`Number of Diagnoses`,
                          diag_3_results$`Number of Diagnoses`),
  Most_Frequent_Diagnosis = c(diag_1_results$`Most Frequent Diagnosis`,
                              diag_2_results$`Most Frequent Diagnosis`,
                              diag_3_results$`Most Frequent Diagnosis`),
  Rare_Diagnoses_Count = c(diag_1_results$`Rare Diagnoses Count`,
                           diag_2_results$`Rare Diagnoses Count`,
                           diag_3_results$`Rare Diagnoses Count`),
  stringsAsFactors = TRUE
)

# Print results
results
```

Diagnosis_Variable <fct>	Number_of_Diagnoses <int>	Most_Frequent_Diagnosis <fct>	Rare_Diagnoses_Count <int>
diag_1	695	414	331
diag_2	724	250	388
diag_3	757	250	404

3 rows

When visualizing features with a large number of distinct values, such as `diag_1`, `diag_2`, and `diag_3`, several challenges may arise:

- *Overloaded plots:* A high number of categories can lead to cluttered visualizations, especially with labels. In bar or pie charts, this often results in many bars or segments, which can impair readability and interpretation. Important trends or patterns may be overlooked.
- *Lack of differentiation:* When many diagnoses occur infrequently, visually distinguishing between categories becomes difficult. Frequency-based charts may push important diagnoses into the background, reducing the informativeness of the visual and potentially leading to misinterpretation.

In terms of modeling with machine learning methods, the following challenges are common:

- *Data sparsity:* A large number of categories often means many diagnoses are rare. Especially generalized linear models (GLMs) may struggle to produce reliable estimates when some categories lack sufficient data for robust modeling.
- *Risk of overfitting:* When modeling with machine learning techniques, many categories can increase the risk of overfitting. Models may start to learn noise instead of true patterns, reducing their generalizability and predictive performance on unseen data.

d) Preparing Diagnosis Reference Data: Load the two files `CCS_mapping_ICD9.csv` and `CCS_categories_ICD9.csv` from the examination materials and perform an appropriate left join using the feature `category_id` to link the information. Next, sort the resulting data by the ICD9-coded feature `code` and export the result as a CSV file `icd9_data.csv`. Create a frequency table for the diagnosis group `group` and ensure it matches the table in Appendix 2 of the exam.

First, we read the two specified files using the `read.csv2` function. Then we perform a left join of the dataset from `CCS_mapping_ICD9.csv` with the dataset from `CCS_categories_ICD9.csv` using the feature `category_id` and sort the resulting dataset by the feature `code`. Finally, the result is exported as a CSV file.

```
# Read diagnosis grouping files
icd9_mapping <- read.csv2(here("2025/2_Datensätze", "CCS_mapping_ICD9.csv"), stringsAsFactors = TRUE)
icd9_categories <- read.csv2(here("2025/2_Datensätze", "CCS_categories_ICD9.csv"), stringsAsFactors = TRUE)

# Perform left join, sort by ICD9 code, and remove duplicates if present
icd9_data <- icd9_mapping %>%
  left_join(icd9_categories, by = "category_id") %>%
  arrange(code) %>%
  distinct()  # Has no effect in this specific case

# Save the resulting file
write.csv2(icd9_data, here("2025/2_Datensätze", "icd9_data.csv"), row.names = FALSE)
```

Finally, we create a frequency table of the feature `group` in the newly created dataset.

```
# Count frequency of each diagnosis group
group_frequency_count <- icd9_data %>%
  group_by(group) %>%
  summarize(Count = n(), .groups = 'drop') %>%
  arrange(group)

# Output frequency table
group_frequency_count
```

group <fct>	Count <int>
Circulatory	57
Diabetes	56
Digestive	60
Genitourinary	51
Injury	193
Musculoskeletal	32
n.a.	1
Neoplasms	100
Other	558
Respiratory	55

1-10 of 10 rows

A comparison with the values in Appendix 2 shows complete agreement.

e) Assigning Diagnosis Groups: First, create a subset `icd9` from the prepared diagnosis group data containing only the columns `code` and `group`. Then, remove all periods ('.') from the values of the diagnosis features `diag_1`, `diag_2`, and `diag_3` in the main dataset. For each of the three diagnosis features, perform a left join with the created dataset `icd9` so that the corresponding diagnosis group is added in a new column (`group_diag_1`, `group_diag_2`, `group_diag_3`). Note that the left join may result in missing values for some diagnoses. Replace these missing values with the placeholder value `Other`. Ensure that the groups are correctly assigned by checking two example rows. Additionally, ensure that all features whose names end with `_id` are of data type `factor`.

We begin by creating a relevant subset from the prepared diagnosis group data, containing only the columns `code` and `group`. Then, periods are removed from the diagnosis fields `diag_1`, `diag_2`, and `diag_3`. For each of these fields, a left join is performed on the extracted diagnosis group data. Missing values in the new diagnosis group columns are replaced with the value `Other`.

```
# Extract relevant columns for left join
icd9 <- icd9_data %>% select(code, group)

# Remove periods from diagnosis fields `diag_1`, `diag_2`, and `diag_3`
data <- data %>% mutate(across(starts_with("diag_"), ~ gsub("\\.", "", .)))

# Perform left joins for `diag_1`, `diag_2`, and `diag_3` and rename columns
data <- data %>%
  left_join(icd9, by = c("diag_1" = "code")) %>%
  rename(group_diag_1 = group) %>%
  left_join(icd9, by = c("diag_2" = "code")) %>%
  rename(group_diag_2 = group) %>%
  left_join(icd9, by = c("diag_3" = "code")) %>%
  rename(group_diag_3 = group) %>%
  mutate(
    diag_1 = as.factor(diag_1),
    diag_2 = as.factor(diag_2),
    diag_3 = as.factor(diag_3)
  )

# Replace missing values in the new diagnosis group columns with 'Other'
data <- data %>% mutate(across(starts_with("group_diag_"), ~ replace_na(., "Other")))
```

Next, we verify that the groups were correctly assigned by examining two randomly selected rows from the main dataset and matching diagnosis codes with the ICD9 reference data.

```
# Randomly select two rows for validation
sample_data_rows <- data %>%
  select(race, gender, age, diag_1, diag_2, diag_3, group_diag_1, group_diag_2, group_diag_3) %>%
  sample_n(2)  # Seed was set in Task R0

# Output the sample rows
sample_data_rows
```

race <fct>	gender <fct>	age <fct>	diag_1 <fct>	diag_2 <fct>	diag_3 <fct>	group_diag_1 <fct>	group_diag_2 <fct>	group_diag_3 <fct>
Caucasian	Male	[70-80)	414	V45	786	Circulatory	Diabetes	Respiratory
Caucasian	Female	[50-60)	574	276	250	Digestive	Other	Diabetes

2 rows

```
# Retrieve matching entries from the ICD9 reference data
sample_icd9_rows <- icd9_data %>%
  select(code, group) %>%
  filter(code %in% unlist(sample_data_rows[c("diag_1", "diag_2", "diag_3")]))

# Output the relevant ICD9 mappings
sample_icd9_rows
```

code <fct>	group <fct>
250	Diabetes
276	Other
414	Circulatory
574	Digestive
786	Respiratory
V45	Diabetes

6 rows

For both sample entries above, the assigned diagnosis groups correctly match the respective diagnosis codes.

Finally, we ensure that all features ending in `_id` are of type `factor` and reset unused factor levels.

```
# Convert all features ending with '_id' to factor and reset levels
data <- data %>%
  mutate(across(ends_with("_id"), as.factor)) %>%
  mutate(across(everything(), ~ if (is.factor(.x)) droplevels(.x) else .x))
```

f) For each feature of data type `factor`, except for `diag_1`, `diag_2`, and `diag_3`, perform the following steps: Check if a feature value occurs in the dataset fewer than ten times. If so, replace it with the most frequent value for that feature. Then, create an overview for each affected feature, listing the feature name, the most frequent value, and the values that were replaced. Finally, analyze the impact of this cleaning process on the application of machine learning models, especially in terms of model stability, performance, and executability. Discuss two relevant effects for each aspect (and, if needed, support your argument using logistic regression and insights from Task R3).

We begin by processing all features of type `factor` in the dataset, except those whose names start with `diag_`. For each affected feature, we count the frequency of its values, determine the most frequent value, and identify values that occur fewer than ten times. These rare values are then replaced with the most frequent value, and the replacements are logged.

```r
# Process all factor features except those starting with "diag_"
data <- data %>%
  mutate(across(
    where(is.factor),
    ~ {
      if (!startsWith(cur_column(), "diag_")) {
        # Count value frequencies
        count_table <- table(.x)

        # Identify the most frequent value
        most_frequent_value <- names(which.max(count_table))

        # Identify rare values (frequency < 10)
        rare_values <- names(count_table[count_table < 10])

        if (length(rare_values) > 0) {
          # Log the replacements
          cat(sprintf(
            "In feature '%s', the following rare values were replaced with '%s':\n",
            cur_column(), most_frequent_value
          ))
          cat(paste(" -", rare_values, "\n", collapse = ""))

          # Replace rare values with the most frequent one
          factor(
            replace(as.character(.x), .x %in% rare_values, most_frequent_value),
            levels = levels(.x)
          )
        } else {
          .x  # No changes needed
        }
      } else {
        .x  # Do not modify diag_* features
      }
    }
  ))
```

```
## In feature 'gender', the following rare values were replaced with 'Female':
##  - Unknown/Invalid
## In feature 'weight', the following rare values were replaced with '?':
##  - [175-200)
##  - >200
## In feature 'admission_type_id', the following rare values were replaced with '1':
##  - 4
## In feature 'discharge_disposition_id', the following rare values were replaced with '1':
##  - 9
##  - 10
##  - 12
##  - 16
##  - 17
##  - 27
## In feature 'admission_source_id', the following rare values were replaced with '7':
##  - 10
##  - 11
##  - 13
##  - 14
##  - 22
##  - 25
## In feature 'payer_code', the following rare values were replaced with '?':
##  - FR
## In feature 'medical_specialty', the following rare values were replaced with '?':
##  - AllergyandImmunology
##  - Anesthesiology
##  - Cardiology-Pediatric
##  - DCPTEAM
##  - Dentistry
##  - Dermatology
##  - Endocrinology-Metabolism
##  - Neurophysiology
##  - OutreachServices
##  - Pathology
##  - Pediatrics-EmergencyMedicine
##  - Pediatrics-Hematology-Oncology
##  - Pediatrics-Neurology
##  - Pediatrics-Pulmonology
##  - Perinatology
##  - PhysicianNotFound
##  - Proctology
##  - Psychiatry-Addictive
##  - Psychiatry-Child/Adolescent
##  - Resident
##  - Speech
##  - SportsMedicine
##  - Surgery-Colon&Rectal
##  - Surgery-Maxillofacial
##  - Surgery-Pediatric
##  - Surgery-PlasticwithinHeadandNeck
## In feature 'nateglinide', the following rare values were replaced with 'No':
##  - Down
## In feature 'chlorpropamide', the following rare values were replaced with 'No':
##  - Down
##  - Up
## In feature 'acetohexamide', the following rare values were replaced with 'No':
##  - Steady
## In feature 'miglitol', the following rare values were replaced with 'No':
##  - Down
##  - Up
## In feature 'troglitazone', the following rare values were replaced with 'No':
##  - Steady
## In feature 'glyburide.metformin', the following rare values were replaced with 'No':
##  - Down
##  - Up
## In feature 'glipizide.metformin', the following rare values were replaced with 'No':
##  - Steady
## In feature 'metformin.rosiglitazone', the following rare values were replaced with 'No':
##  - Steady
## In feature 'metformin.pioglitazone', the following rare values were replaced with 'No':
##  - Steady
```

The performed data cleaning impacts the use of machine learning models, especially logistic regression, in several aspects:

Model Stability:

- Combining rare values improves model stability by increasing the training base and yielding more robust parameter estimates.
- However, oversimplification may lead to information loss by eliminating potentially meaningful patterns present in rare categories.

Performance:

- Cleaning can enhance model accuracy by reducing sparsity and mitigating overfitting to rare categories.
- Conversely, predictive performance may suffer if relevant predictive signals in rare values are discarded.

Executability:

- Reducing rare values simplifies the model, saves memory and computation time, and typically results in faster runtime.

- Rare values that appear only in training or test sets can cause issues during data splits. Some models may even error out when encountering unknown values in test data.

(Example from Logistic Regression: Cleaning minimizes multicollinearity risks associated with low-frequency categories, resulting in more consistent and interpretable odds ratios. Moreover, the logistic regression in Task R3 trained faster and produced more stable outcomes after applying this preprocessing step.)

g) Creating Datasets for Binary and Ternary Classification: Two datasets should be created for later tasks: A dataset `diabetic_data_ter` for the ternary classification of hospital readmissions and a dataset `diabetic_data_bin` for the binary classification of hospital readmissions.

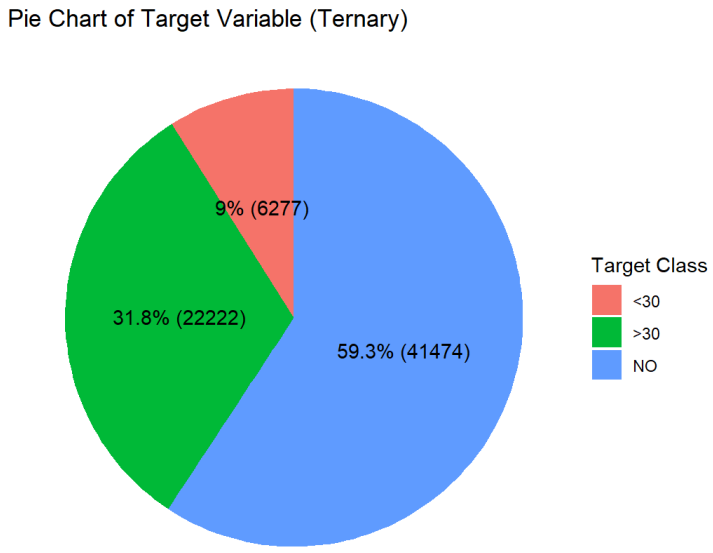
- **Dataset `diabetic_data_ter`: Replace the feature `readmitted` with a new feature `TARGET` of data type `factor`. Also, remove any levels from `factor` features that no longer occur after the previous filtering steps. Visualize the distribution of the ternary target variable `TARGET` with a pie chart.**
- **Dataset `diabetic_data_bin`: Based on the dataset `diabetic_data_ter`, filter out all rows where `TARGET` has the value `>30`. Then, modify the feature `TARGET` while keeping the data type `factor` as follows: Assign `TARGET` the value `1` if the patient was readmitted within 30 days (former `TARGET` value: `<30`), and assign the value `0` otherwise. Also, remove any levels from `factor` features that no longer occur after the previous filtering steps. Visualize the distribution of the binary target variable `TARGET` with a pie chart. Briefly discuss the imbalance of this target variable and its potential impact on later modeling. Display the first ten rows of the preprocessed dataset and then export the entire dataset as a CSV file named `diabetic_data_bin.csv`.**

We start by creating the dataset for ternary classification according to the steps described in the task.

```
# Create dataset for ternary classification (diabetic_data_ter)
# Replace 'readmitted' with 'TARGET' and convert to factor
diabetic_data_ter <- data %>%
  mutate(TARGET = as.factor(case_when(
    readmitted == "NO" ~ "NO",
    readmitted == ">30" ~ ">30",
    readmitted == "<30" ~ "<30"
  ))) %>%
  select(-readmitted) %>%
  mutate(across(everything(), ~ if (is.factor(.x)) droplevels(.x) else .x))
```

Next, we visualize the distribution of the ternary target variable using a pie chart.

```
# Visualize distribution of 'TARGET' in ternary dataset
target_dist_ter <- diabetic_data_ter %>%
  group_by(TARGET) %>%
  summarize(count = n()) %>%
  mutate(percentage = count / sum(count) * 100)

ggplot(target_dist_ter, aes(x = "", y = count, fill = TARGET)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y") +
  geom_text(aes(label = paste0(round(percentage, 1), "% (", count, ")")),
            position = position_stack(vjust = 0.5)) +
  labs(fill = "Target Class", title = "Pie Chart of Target Variable (Ternary)") +
  theme_void() +
  theme(legend.position = "right")
```

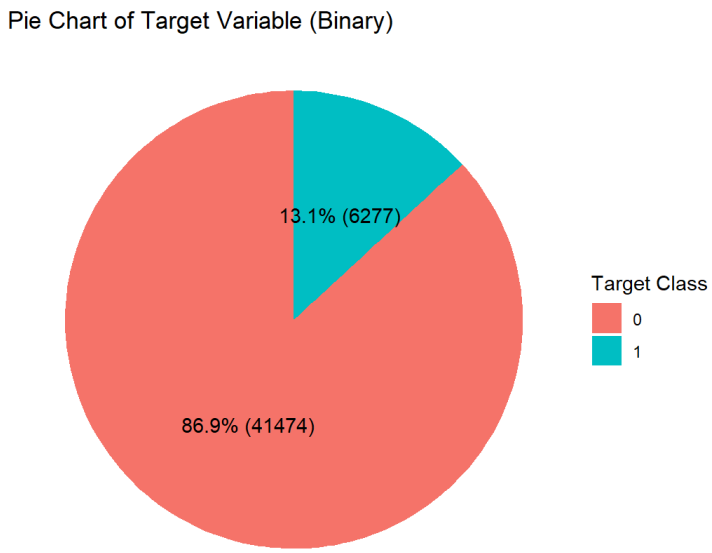


We then create the binary classification dataset as specified.

```r
# Create binary classification dataset (diabetic_data_bin)
# Based on diabetic_data_ter: remove '>30' and adjust target variable
diabetic_data_bin <- diabetic_data_ter %>%
  filter(TARGET != ">30") %>%
  mutate(TARGET = ifelse(TARGET == "<30", 1, 0)) %>%
  mutate(across(everything(), ~ if (is.factor(.x)) droplevels(.x) else .x))

# Set TARGET as factor
diabetic_data_bin$TARGET <- as.factor(diabetic_data_bin$TARGET)
```

We also generate a pie chart to visualize the distribution of the binary target variable.

```r
# Visualize distribution of 'TARGET' in binary dataset
target_dist_bin <- diabetic_data_bin %>%
  group_by(TARGET) %>%
  summarize(count = n()) %>%
  mutate(percentage = count / sum(count) * 100)

ggplot(target_dist_bin, aes(x = "", y = count, fill = TARGET)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y") +
  geom_text(aes(label = paste0(round(percentage, 1), "% (", count, ")")),
            position = position_stack(vjust = 0.5)) +
  labs(fill = "Target Class", title = "Pie Chart of Target Variable (Binary)") +
  theme_void() +
  theme(legend.position = "right")
```



The `TARGET` variable exhibits clear imbalance: approximately 87% of cases belong to class `0` (no readmission), while only about 13% fall into class `1` (readmission).

This imbalance may cause machine learning models to favor the majority class, making it harder to correctly classify the minority class. Accuracy is generally not a meaningful metric in this scenario. Instead, metrics such as the Area Under the ROC Curve (AUC) should be used to better assess model performance. To address class imbalance, techniques such as oversampling the minority class, undersampling the majority class, or using algorithms specifically designed for imbalanced data can be employed.

Finally, we display the first ten rows and export the dataset as a CSV file.

```r
# Display the first 10 rows of the processed dataset
head(diabetic_data_bin, n = 10)
```

race <fct>	gender <fct>	age <fct>	weight <fct>	admission_type_id <fct>	discharge_disposition_id <fct>
Caucasian	Female	[50-60)	?	2	1
Caucasian	Female	[50-60)	?	3	1
Caucasian	Female	[80-90)	?	1	3
Caucasian	Female	[80-90)	?	1	1
AfricanAmerican	Female	[30-40)	?	1	1
Caucasian	Female	[60-70)	?	3	1
Caucasian	Female	[70-80)	?	2	3
Caucasian	Male	[60-70)	?	2	1
Caucasian	Female	[70-80)	?	3	1
Caucasian	Female	[80-90)	?	1	3

1-10 of 10 rows | 1-6 of 51 columns

```
# Save the resulting dataset
write.csv(diabetic_data_bin, "diabetic_data_bin.csv", row.names = FALSE)
```

Task R2: Exploratory Data Analysis and Visualization

In this task, code snippets from the *SWoF* template must be extracted and modified in certain places. The objective of these modifications is to ensure the code runs with minimal adjustments, using the dataset `diabetic_data_bin` , which was created at the end of Task R1, as input. At the beginning of each relevant subtask, the corresponding section of the *SWoF* template is indicated in bold.

Note: For all visualizations created in this task, ensure clarity, consistency, and readability. Points will be deducted for failing to meet this requirement.

a) 3 Overview: Apply the functions `summary` and `glimpse` to the dataset `diabetic_data_bin` and briefly discuss the results.

We start by applying R's built-in `summary` function to the dataset `diabetic_data_bin` .

```
summary(diabetic_data_bin)
```

```
##               race          gender           age              weight
##  ?              : 1483   Female:25152   [70-80):11623   ?        :46283
##  AfricanAmerican: 8757   Male  :22599   [60-70):10736   [75-100) :  589
##  Asian          :  381                  [50-60): 8615   [50-75)  :  428
##  Caucasian      :35203                  [80-90): 7242   [100-125):  304
##  Hispanic       : 1088                  [40-50): 4885   [125-150):   70
##  Other          :  839                  [30-40): 2003   [25-50)  :   43
##                                         (Other): 2647   (Other)  :   34
##  admission_type_id discharge_disposition_id admission_source_id
##  1:24036           1      :30258            7      :24540
##  2: 8731           3      : 6029            1      :15312
##  3:10056           6      : 5189            17     : 3090
##  5: 2047           18     : 1921            4      : 2021
##  6: 2641           2      : 1101            6      : 1479
##  7:   18           22     : 1073            2      :  673
##  8:  222           (Other): 2180            (Other):  636
##  time_in_hospital   payer_code                medical_specialty
##  Min.   : 1.000   ?      :20378   ?                    :22642
##  1st Qu.: 2.000   MC     :13115   InternalMedicine     : 7341
##  Median : 3.000   HM     : 2821   Family/GeneralPractice: 3312
##  Mean   : 4.205   BC     : 2601   Emergency/Trauma     : 2875
##  3rd Qu.: 6.000   SP     : 2259   Cardiology           : 2811
##  Max.   :14.000   MD     : 1522   Surgery-General      : 1528
##                   (Other): 5055   (Other)              : 7242
##  num_lab_procedures num_procedures  num_medications number_outpatient
##  Min.   :  1.00     Min.   :0.000   Min.   : 1.00   Min.   : 0.000
##  1st Qu.: 30.00     1st Qu.:0.000   1st Qu.:10.00   1st Qu.: 0.000
##  Median : 44.00     Median :1.000   Median :14.00   Median : 0.000
##  Mean   : 42.36     Mean   :1.458   Mean   :15.56   Mean   : 0.233
##  3rd Qu.: 56.00     3rd Qu.:2.000   3rd Qu.:20.00   3rd Qu.: 0.000
##  Max.   :132.00     Max.   :6.000   Max.   :81.00   Max.   :36.000
##
##  number_emergency   number_inpatient      diag_1          diag_2
##  Min.   : 0.00000   Min.   : 0.0000   414    : 3641   250    : 3748
##  1st Qu.: 0.00000   1st Qu.: 0.0000   428    : 2219   276    : 3047
##  Median : 0.00000   Median : 0.0000   786    : 2063   428    : 2589
##  Mean   : 0.08157   Mean   : 0.1383   410    : 2006   401    : 2309
##  3rd Qu.: 0.00000   3rd Qu.: 0.0000   486    : 1488   427    : 2283
##  Max.   :37.00000   Max.   :12.0000   715    : 1418   599    : 1495
##                                       (Other):34916   (Other):32280
##     diag_3      number_diagnoses max_glu_serum A1Cresult     metformin
##  250    : 6522   Min.   : 1.000   >200:  608    >7  : 2003   Down  :  308
##  401    : 4787   1st Qu.: 5.000   >300:  429    >8  : 4279   No    :37346
##  276    : 2233   Median : 8.000   None:45565    None:38767   Steady: 9513
##  414    : 1748   Mean   : 7.102   Norm: 1149    Norm: 2702   Up    :  584
##  428    : 1695   3rd Qu.: 9.000
##  427    : 1646   Max.   :16.000
##  (Other):29120
##  repaglinide    nateglinide    chlorpropamide glimepiride    acetohexamide
##  Down  :   17   No    :47440   No    :47710   Down  :   92   No:47751
##  No    :47174   Steady:  300   Steady:   41   No    :45262
##  Steady:  505   Up    :   11                  Steady: 2233
##  Up    :   55                                 Up    :  164
##
##
##
##   glipizide      glyburide     tolbutamide    pioglitazone   rosiglitazone
##  Down  :  234   Down  :  266   No    :47739   Down  :   46   Down  :   56
##  No    :41845   No    :42500   Steady:   12   No    :44308   No    :44750
##  Steady: 5293   Steady: 4571                  Steady: 3287   Steady: 2850
##  Up    :  379   Up    :  414                  Up    :  110   Up    :   95
##
##
##
##    acarbose       miglitol     troglitazone  tolazamide    examide
##  No    :47638   No    :47743   No:47751     No    :47730   No:47751
##  Steady:  107   Steady:    8                Steady:   21
##  Up    :    6
##
##
##
##
##  citoglipton   insulin      glyburide.metformin glipizide.metformin
##  No:47751    Down  : 4776   No    :47433        No:47751
##              No    :23741   Steady:  318
##              Steady:14795
##              Up    : 4439
##
##
##
##  glimepiride.pioglitazone metformin.rosiglitazone metformin.pioglitazone
##  No:47751                 No:47751                No:47751
##
##
##
```

```
##
##
##
##  change     diabetesMed      group_diag_1          group_diag_2
##  Ch:21017   No :12040   Circulatory:14395   Circulatory  :14079
##  No:26734   Yes:35711   Other      : 8104   Other        :12432
##                         Respiratory: 6032   Diabetes     : 7141
##                         Digestive  : 4229   Respiratory  : 4461
##                         Diabetes   : 3895   Genitourinary: 3612
##                         Injury     : 3374   Digestive    : 1830
##                         (Other)    : 7722   (Other)      : 4196
##         group_diag_3   TARGET
##  Circulatory  :13433   0:41474
##  Other        :13108   1: 6277
##  Diabetes     : 9444
##  Respiratory  : 3017
##  Genitourinary: 2676
##  Digestive    : 1644
##  (Other)      : 4429
```

The `summary` function provides basic statistics for all features – eight numeric and 43 categorical ones. For categorical features (e.g., `gender`), it lists the frequency of each category (possibly summarizing infrequent ones under `(Other)`). For numeric features (e.g., `time_in_hospital`), it reports the minimum, maximum, mean, and quartiles. Most features do not follow a uniform distribution and exhibit considerable variance. In contrast, some features (e.g., `examide`) show only a single value due to earlier preprocessing.

Next, we use the `glimpse` function from the `dplyr` package to get a quick overview of the dataset's structure.

```
glimpse(diabetic_data_bin)
```

```
## Rows: 47,751
## Columns: 51
## $ race                    <fct> Caucasian, Caucasian, Caucasian, Caucasian, A…
## $ gender                  <fct> Female, Female, Female, Female, Female, Femal…
## $ age                     <fct> [50-60), [50-60), [80-90), [80-90), [30-40), …
## $ weight                  <fct> ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, …
## $ admission_type_id       <fct> 2, 3, 1, 1, 1, 3, 2, 2, 3, 1, 1, 1, 3, 3, 1, …
## $ discharge_disposition_id <fct> 1, 1, 3, 1, 1, 1, 3, 1, 1, 3, 1, 1, 5, 1, 1, …
## $ admission_source_id     <fct> 1, 1, 7, 7, 7, 1, 1, 1, 1, 7, 7, 7, 1, 1, 7, …
## $ time_in_hospital        <int> 8, 2, 4, 3, 5, 9, 12, 8, 1, 7, 4, 1, 4, 6, 4,…
## $ payer_code              <fct> ?, ?, MC, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,…
## $ medical_specialty       <fct> Cardiology, Surgery-Neuro, InternalMedicine, …
## $ num_lab_procedures      <int> 77, 49, 68, 46, 49, 52, 47, 57, 31, 77, 47, 3…
## $ num_procedures          <int> 6, 1, 2, 0, 0, 1, 2, 6, 1, 0, 4, 5, 2, 2, 3, …
## $ num_medications         <int> 33, 11, 23, 20, 5, 16, 18, 31, 9, 12, 16, 13,…
## $ number_outpatient       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
## $ number_emergency        <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
## $ number_inpatient        <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
## $ diag_1                  <fct> 401, 722, 820, 274, 590, 491, 682, 414, 722, …
## $ diag_2                  <fct> 997, 305, 493, 427, 220, 428, 707, 584, 25001…
## $ diag_3                  <fct> 560, 250, E880, 416, 250, 25001, 560, 285, 41…
## $ number_diagnoses        <int> 8, 3, 9, 9, 3, 9, 9, 9, 7, 9, 7, 8, 9, 7, 6, …
## $ max_glu_serum           <fct> None, None, None, None, None, None, None, Non…
## $ A1Cresult               <fct> None, None, >7, >8, None, None, Norm, >8, Non…
## $ metformin               <fct> Steady, No, Steady, Steady, No, No, No, No, N…
## $ repaglinide             <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ nateglinide             <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ chlorpropamide          <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ glimepiride             <fct> No, No, No, No, Steady, No, Down, No, No, No,…
## $ acetohexamide           <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ glipizide               <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ glyburide               <fct> Down, No, No, Steady, No, No, No, No, No, No,…
## $ tolbutamide             <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ pioglitazone            <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ rosiglitazone           <fct> No, No, No, No, No, No, No, No, No, No, No, S…
## $ acarbose                <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ miglitol                <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ troglitazone            <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ tolazamide              <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ examide                 <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ citoglipton             <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ insulin                 <fct> Steady, No, No, No, No, Steady, Steady, Stead…
## $ glyburide.metformin     <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ glipizide.metformin     <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ glimepiride.pioglitazone <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ metformin.rosiglitazone  <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ metformin.pioglitazone   <fct> No, No, No, No, No, No, No, No, No, No, No, N…
## $ change                  <fct> Ch, No, No, Ch, No, No, Ch, No, No, No, No, C…
## $ diabetesMed             <fct> Yes, No, Yes, Yes, Yes, Yes, Yes, Yes, No, Ye…
## $ group_diag_1            <fct> Circulatory, Musculoskeletal, Injury, Other, …
## $ group_diag_2            <fct> Injury, Other, Respiratory, Circulatory, Neop…
## $ group_diag_3            <fct> Digestive, Diabetes, Other, Circulatory, Diab…
## $ TARGET                  <fct> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
```

The `glimpse` function outputs the number of rows and columns, along with the data type of each feature. It confirms that all categorical features are stored as `factor` and all numeric features correctly as `integer`. Missing values (e.g., in `weight`) are immediately visible.

b) 4 Individual feature visualisations: In the *SWoF* template, the features in the dataset are classified into different groups (Sections 4.1 to 4.9). Define a meaningful grouping of the features in the dataset `diabetic_data_bin` (excluding `diag_1`, `diag_2`, `diag_3`) into four to seven groups and justify your choice. For each group, create histograms and kernel density estimator plots for the respective features, ensuring they resemble those in Sections 4.1 to 4.9 of the template. Use the function `multiplot` and apply a logarithmic y-axis where appropriate. Discuss the key insights derived from the visualizations, particularly in terms of their relevance for predicting the target variable `TARGET`.

The features in the dataset are grouped thematically as follows for further processing:

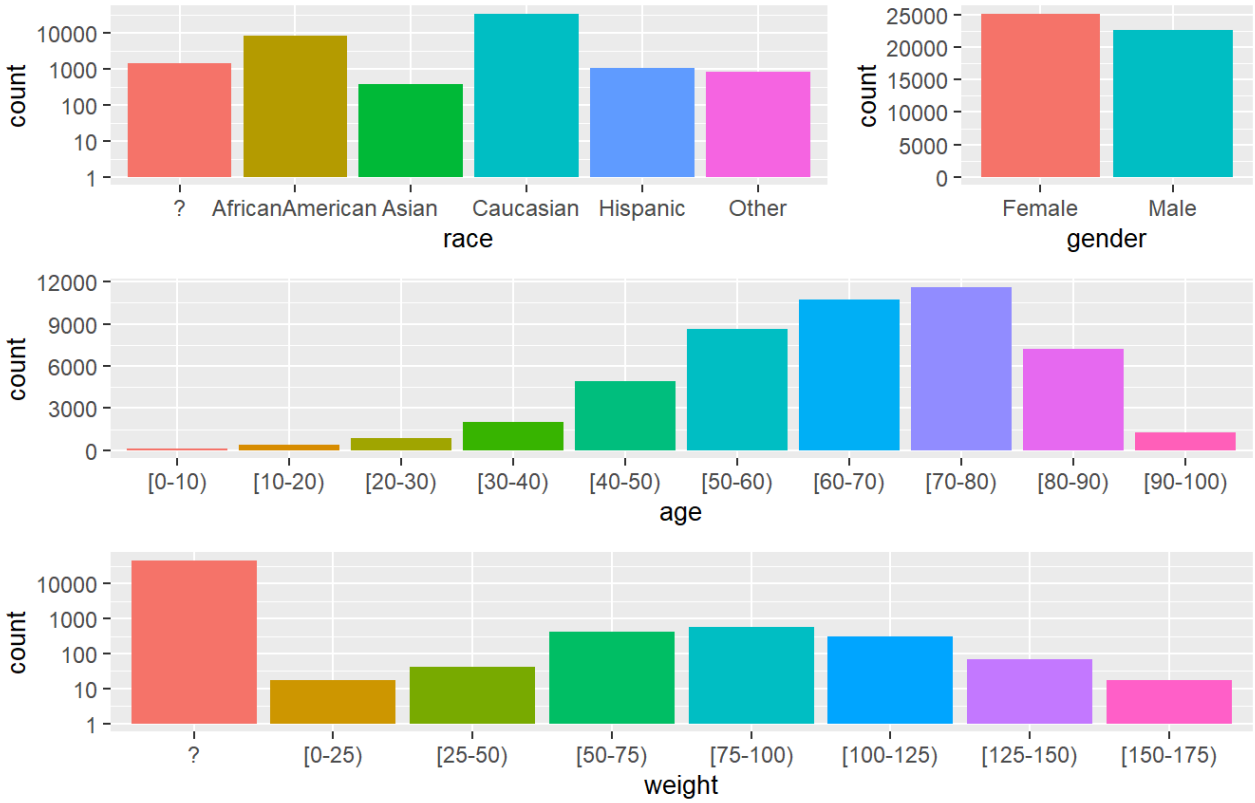
- **Categorical features related to personal information** (e.g., gender, age)
- **Categorical features related to hospital stay details** (e.g., admission and discharge modalities)
- **Categorical features related to insurance and medical information** (e.g., specialty, insurance details)
- **Categorical features related to diagnosis groups**
- **Categorical features related to diabetes-specific information and medication** (e.g., test results, medication use)
- **Numerical features related to hospital resource use and patient health status** (e.g., number of procedures, length of hospital stay)

This grouping is useful because it ensures a thematic separation of features and brings together related attributes. It facilitates interpretation and targeted analysis of variables with respect to the target variable `TARGET`.

Categorical features related to personal information

We begin by visualizing the histograms of features that describe personal information: `race`, `gender`, `age`, and `weight`. A logarithmic y-axis is used for `race` and `weight` due to strong skewness.

```
# Shared theme for uniform text sizes
common_theme <- theme(
  text = element_text(size = 10),
  axis.title = element_text(size = 10),
  axis.text = element_text(size = 9),
  legend.position = "none"
)

p1 <- diabetic_data_bin %>%
  ggplot(aes(race, fill = race)) +
  geom_bar() +
  scale_y_log10() +
  common_theme

p2 <- diabetic_data_bin %>%
  ggplot(aes(gender, fill = gender)) +
  geom_bar() +
  common_theme

p3 <- diabetic_data_bin %>%
  ggplot(aes(age, fill = age)) +
  geom_bar() +
  common_theme

weight_levels <- c("?", "[0-25)", "[25-50)", "[50-75)", "[75-100)", "[100-125)", "[125-150)", "[150-175)", "[175-200)")
diabetic_data_bin$weight <- factor(diabetic_data_bin$weight, levels = weight_levels)

p4 <- diabetic_data_bin %>%
  ggplot(aes(weight, fill = weight)) +
  geom_bar() +
  scale_y_log10() +
  common_theme

layout <- matrix(c(1,1,2,3,3,3,4,4,4),3,3,byrow=TRUE)
multiplot(p1, p2, p3, p4, layout=layout)
```



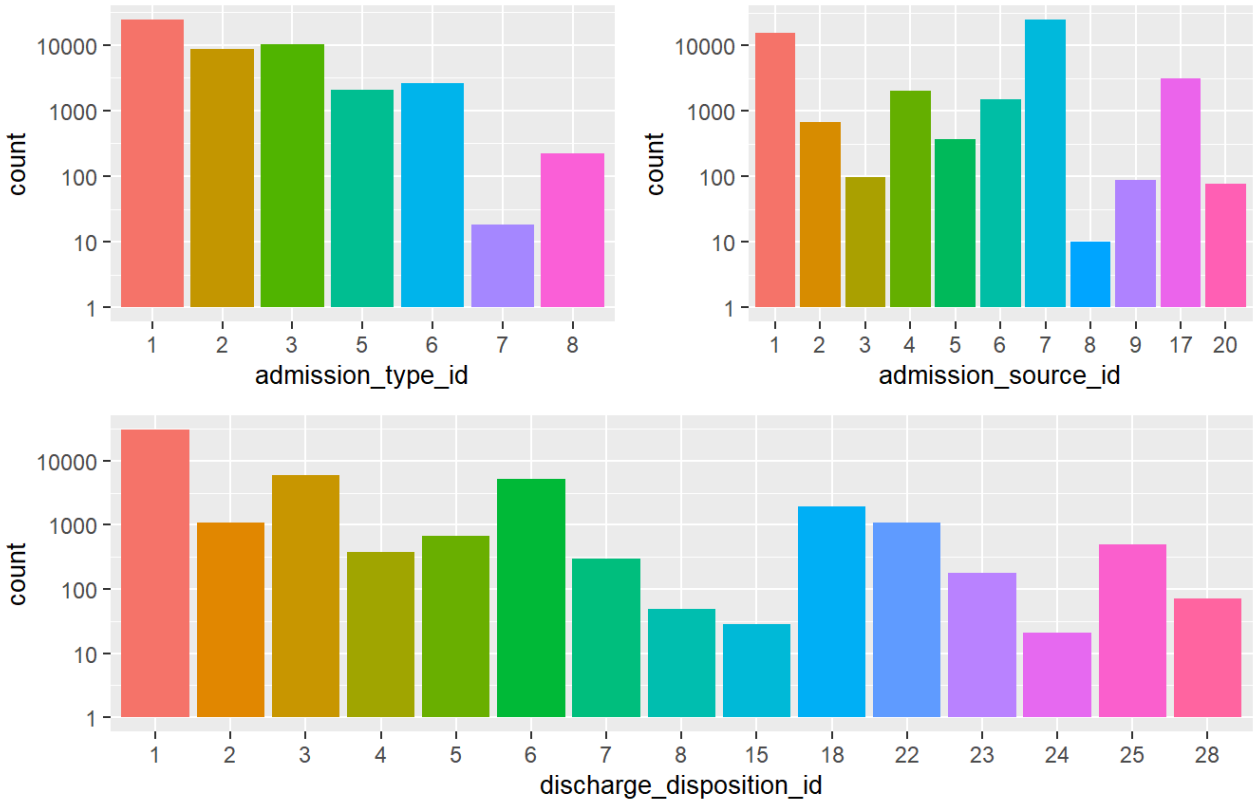
Key insights:

- `race` : Predominantly `Caucasian` , followed by `AfricanAmerican` ; other categories are rare. The difference between `?` and `Other` may need investigation.
- `gender` : Nearly balanced between `Female` and `Male` ; `Unknown/Invalid` has been removed in prior steps.
- `age` : Left-skewed distribution; older age groups (e.g., `[70-80)`) are more frequent.
- `weight` : Over 90% of data marked as `?` (missing); remaining values center on mid-range weights. The high amount of missing data limits predictive usefulness.

Categorical features related to hospital stay details

Now we visualize the features that describe hospital stay details: `admission_type_id` , `admission_source_id` , and `discharge_disposition_id` , using logarithmic y-axes due to high imbalance.

```
p1 <- diabetic_data_bin %>%
  ggplot(aes(admission_type_id, fill = admission_type_id)) +
  geom_bar() +
  scale_y_log10() +
  common_theme

p2 <- diabetic_data_bin %>%
  ggplot(aes(admission_source_id, fill = admission_source_id)) +
  geom_bar() +
  scale_y_log10() +
  common_theme

p3 <- diabetic_data_bin %>%
  ggplot(aes(discharge_disposition_id, fill = discharge_disposition_id)) +
  geom_bar() +
  scale_y_log10() +
  common_theme

layout <- matrix(c(1,2,3,3),2,2,byrow=TRUE)
multiplot(p1, p2, p3, layout=layout)
```



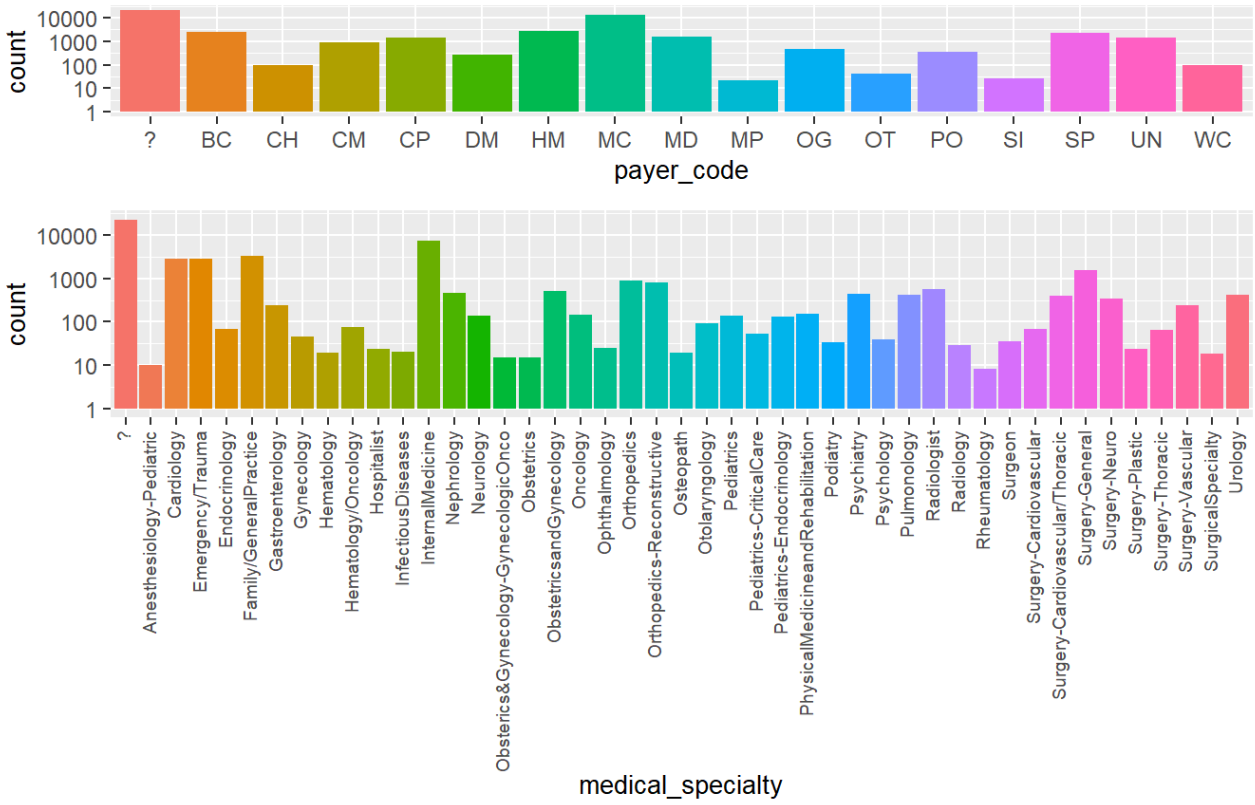
Key Insights:

- `admission_type_id` : Most patients were admitted as `1` (Emergency), `2` (Urgent), or `3` (Elective), with `1` being the most frequent.
- `admission_source_id` : Category `7` (Emergency Room) is dominant, followed by `1` (Physician Referral).
- `discharge_disposition_id` : Most patients discharged to `1` (Home), `3` (Another medical facility), or `6` (Another inpatient facility); some categories (e.g., `8` , `15` , `24`) are rare.

Categorical features related to insurance and medical information

We now visualize histograms for the features `payer_code` and `medical_specialty` . Due to the high imbalance, a logarithmic y-axis is used. Long x-axis labels are rotated for readability.

```
p1 <- diabetic_data_bin %>%
  ggplot(aes(payer_code, fill = payer_code)) +
  geom_bar() +
  scale_y_log10() +
  common_theme

p2 <- diabetic_data_bin %>%
  ggplot(aes(medical_specialty, fill = medical_specialty)) +
  geom_bar() +
  scale_y_log10() +
  common_theme +
  theme(
    legend.position = "none",
    axis.text.x = element_text(size = 7, angle = 90, hjust = 1, vjust = 0.3),
    axis.text.y = element_text(size = 9)
  )

layout <- matrix(c(1,1,2,2,2,2,2,2),4,2,byrow=TRUE)
multiplot(p1, p2, layout=layout)
```



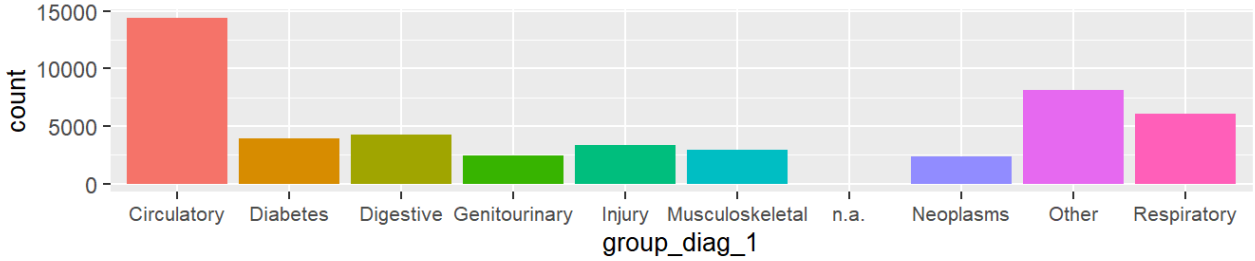
Key Insights:

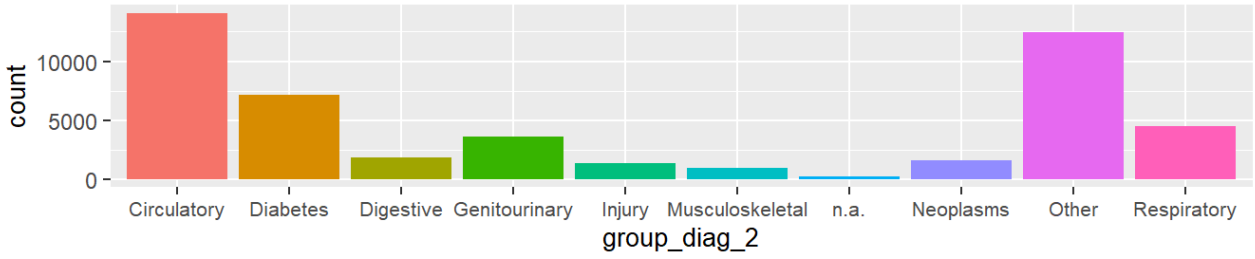
- `payer_code` : Most entries are `?` (missing). Among defined values, `MC` and `BC` dominate. Categories like `MP` and `SI` are very rare.
- `medical_specialty` : Again, `?` is most frequent. Top specialties include `InternalMedicine` , `Family/GeneralPractice` , and `Cardiology` . Rare specialties like `Rheumatology` and `Anesthesiology-Pediatric` occur infrequently.

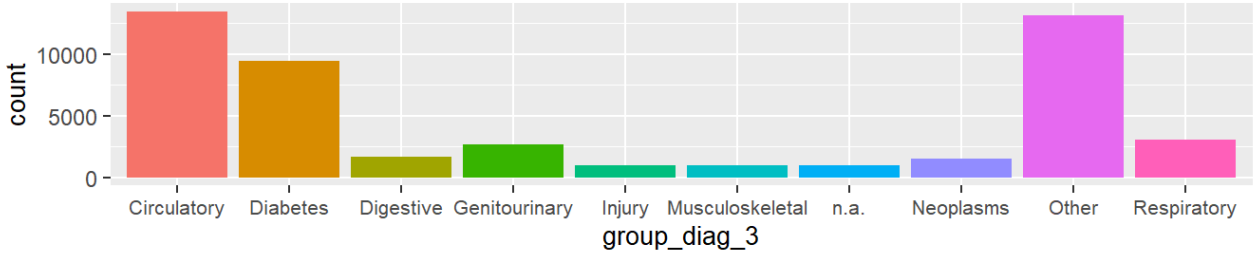
Categorical features related to diagnosis groups

Next, we visualize the distribution of diagnosis group features `group_diag_1` , `group_diag_2` , and `group_diag_3` which were created in Task 1.

```
p1 <- diabetic_data_bin %>%
  ggplot(aes(group_diag_1, fill = group_diag_1)) +
  geom_bar() +
  common_theme +
  theme(axis.text.x = element_text(size = 8))

p2 <- diabetic_data_bin %>%
  ggplot(aes(group_diag_2, fill = group_diag_2)) +
  geom_bar() +
  common_theme +
  theme(axis.text.x = element_text(size = 8))

p3 <- diabetic_data_bin %>%
  ggplot(aes(group_diag_3, fill = group_diag_3)) +
  geom_bar() +
  common_theme +
  theme(axis.text.x = element_text(size = 8))

layout <- matrix(c(1,1,2,2,3,3),3,2,byrow=TRUE)
multiplot(p1, p2, p3, layout=layout)
```







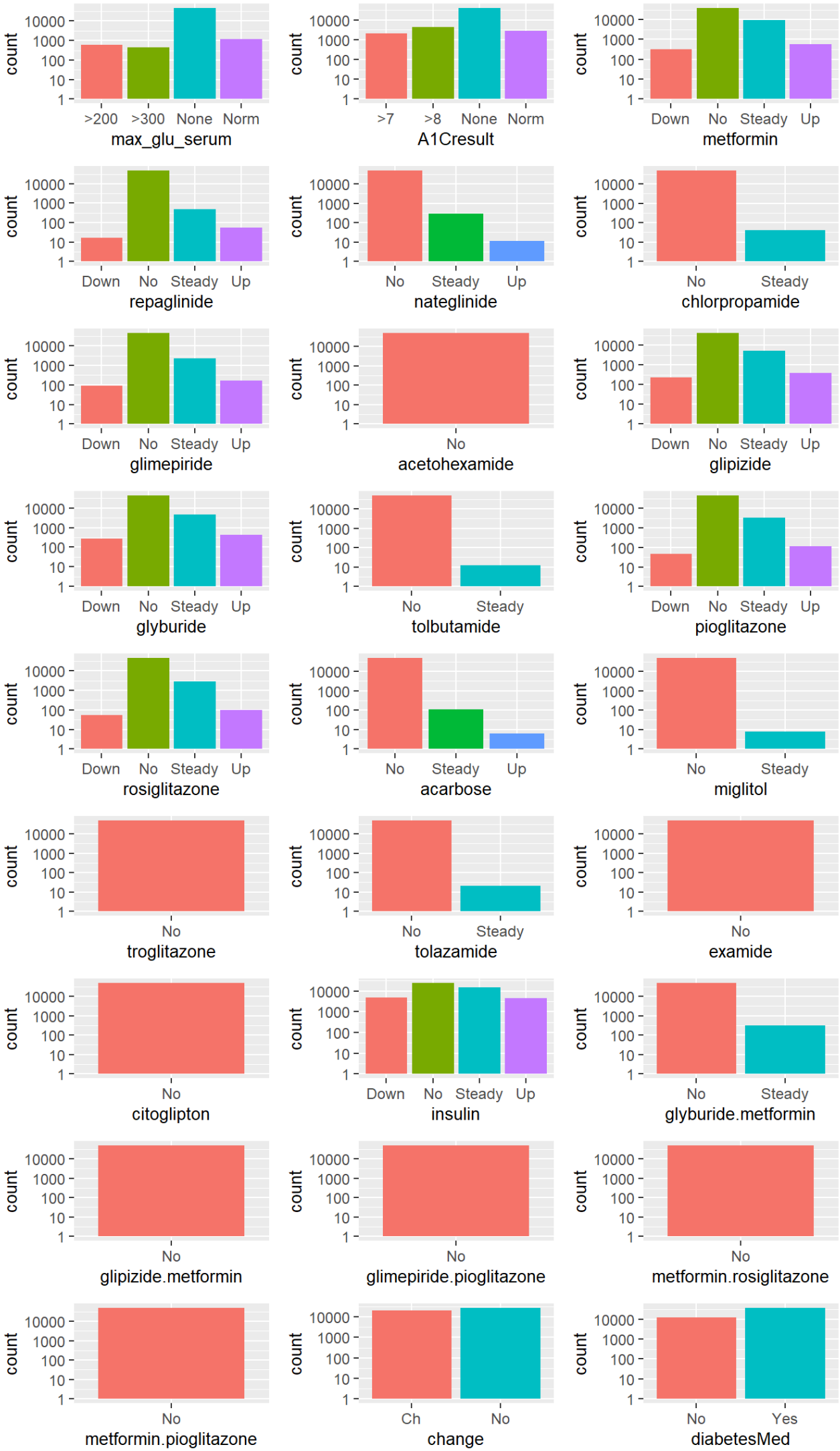
Key Insights:

- `group_diag_1` : Dominated by `Circulatory` , followed by `Other` , `Respiratory` , `Digestive` , `Diabetes` , and `Injury` . `Neoplasms` and `n.a.` are rare.
- `group_diag_2` : Again `Circulatory` leads, with smaller gap to `Other` . `Diabetes` increases compared to `group_diag_1` . `Genitourinary` gains importance.
- `group_diag_3` : `Circulatory` and `Other` dominate. `Diabetes` peaks here. `Respiratory` , `Digestive` , and others are rare. `Neoplasms` and `n.a.` are marginal.

Categorical features related to diabetes-specific information and medication

We now visualize the histograms of features that describe diagnostic tests and patient medication, including: `max_glu_serum` , `A1Cresult` , `change` , and various medications (e.g., `insulin` , `metformin` , `glipizide`). A logarithmic y-axis is used for all features due to the strong imbalance in their distributions.

```
features_to_plot <- c("max_glu_serum", "A1Cresult", "metformin", "repaglinide", "nateglinide", "chlorpropamide", "glimepirid
         e", "acetohexamide", "glipizide", "glyburide", "tolbutamide", "pioglitazone", "rosiglitazone", "acarbose", "miglito
         l", "troglitazone", "tolazamide", "examide", "citoglipton", "insulin", "glyburide.metformin", "glipizide.metformi
         n", "glimepiride.pioglitazone", "metformin.rosiglitazone", "metformin.pioglitazone", "change", "diabetesMed")

create_plot <- function(data, feature) {
  ggplot(data, aes_string(feature, fill = feature)) +
    geom_bar() +
    scale_y_log10() +
    scale_x_discrete(drop=FALSE) +
    common_theme
}

plots <- lapply(features_to_plot, create_plot, data = diabetic_data_bin)
layout <- matrix(1:27, nrow = 9, ncol = 3, byrow = TRUE)
multiplot(plotlist = plots, layout=layout)
```



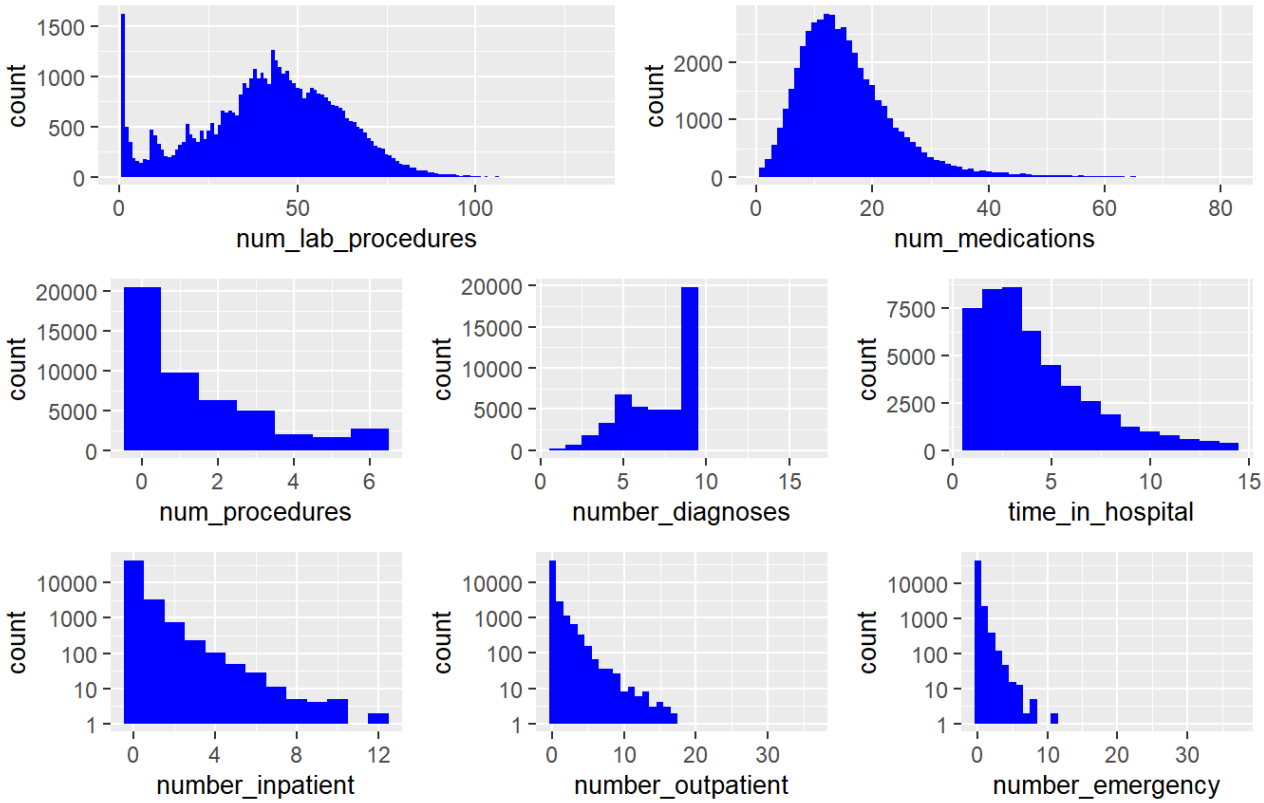
Key Insights:

- `max_glu_serum` and `A1Cresult`: The majority of values in both features are coded as `None`, indicating missing test results. Remaining values are concentrated in categories such as `>200` (`max_glu_serum`) and `>7` / `>8` (`A1Cresult`).
- `insulin`: Shows clear variability. Most values fall into the `No` category, followed by `Steady`. Notably, there is a substantial portion of `Up` and `Down`, indicating dynamic insulin adjustments.
- `change`: A significant number of `Ch` values suggest that medication regimens are frequently adjusted. However, a large number of `No` entries also point to stable treatment courses.
- Frequently adjusted medications: Features like `metformin`, `glipizide`, `glyburide`, and `pioglitazone` exhibit patterns across the categories `Steady`, `Up`, and `Down`, suggesting regular use and dosage modifications.
- Practically unused medications: Some drugs such as `acarbose`, `miglitol`, `troglitazone`, and combinations like `glyburide.metformin` are almost exclusively labeled `No`, indicating minimal or no usage.
- Features with only one value: Variables such as `examide`, `citoglipton`, and drug combinations like `metformin.rosiglitazone` contain only the value `No`, implying they carry no predictive information and could be removed during modeling to improve efficiency.

Numerical features related to hospital resource use and patient health status

The numerical features contained in our dataset are essentially integer-valued features. We use logarithmic axes for some of them.

```
p1 <- diabetic_data_bin %>%
  ggplot(aes(num_lab_procedures, fill = num_lab_procedures)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  common_theme

p2 <- diabetic_data_bin %>%
  ggplot(aes(num_procedures, fill = num_procedures)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  common_theme

p3 <- diabetic_data_bin %>%
  ggplot(aes(number_diagnoses, fill = number_diagnoses)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  common_theme

p4 <- diabetic_data_bin %>%
  ggplot(aes(num_medications, fill = num_medications)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  common_theme

p5 <- diabetic_data_bin %>%
  ggplot(aes(number_emergency, fill = number_emergency)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  scale_y_log10() +
  common_theme

p6 <- diabetic_data_bin %>%
  ggplot(aes(time_in_hospital, fill = time_in_hospital)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  common_theme

p7 <- diabetic_data_bin %>%
  ggplot(aes(number_inpatient, fill = number_inpatient)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  scale_y_log10() +
  common_theme

p8 <- diabetic_data_bin %>%
  ggplot(aes(number_outpatient, fill = number_outpatient)) +
  geom_histogram(fill = "blue", binwidth = 1) +
  scale_y_log10() +
  common_theme

layout <- matrix(c(1,1,1,2,2,2,3,3,4,4,5,5,6,6,7,7,8,8),3,6,byrow=TRUE)
multiplot(p1, p4, p2, p3, p6, p7, p8, p5, layout=layout)
```



Key Insights:

- `num_lab_procedures` : Most patients underwent around 40 to 50 lab tests, indicating extensive diagnostic assessments. A small subset had no lab tests, possibly reflecting shorter hospital stays.
- `num_medications` : The distribution centers around 10 to 20 medications, with a declining trend at higher counts. This suggests most patients received a moderate number of drugs, while a few underwent more intensive pharmacological treatment.
- `time_in_hospital` : Hospital stays are generally short, with 3 days being the most common duration. Longer stays are rare and likely correspond to more complex medical cases.
- `number_diagnoses` : Most patients have around 8 to 9 diagnoses, indicating complex health profiles. Very high diagnosis counts (>10) are rare.
- `number_inpatient` , `number_emergency` , and `number_outpatient` : All three features are strongly right-skewed. Many patients had few or no admissions in these categories. The maximum observed values were 12 (`number_inpatient`), 36 (`number_outpatient`), and 37 (`number_emergency`).
- `num_procedures` : A significant portion of patients underwent no additional medical procedures beyond lab tests, and there is no clear trend of increasing frequency with higher procedure counts.

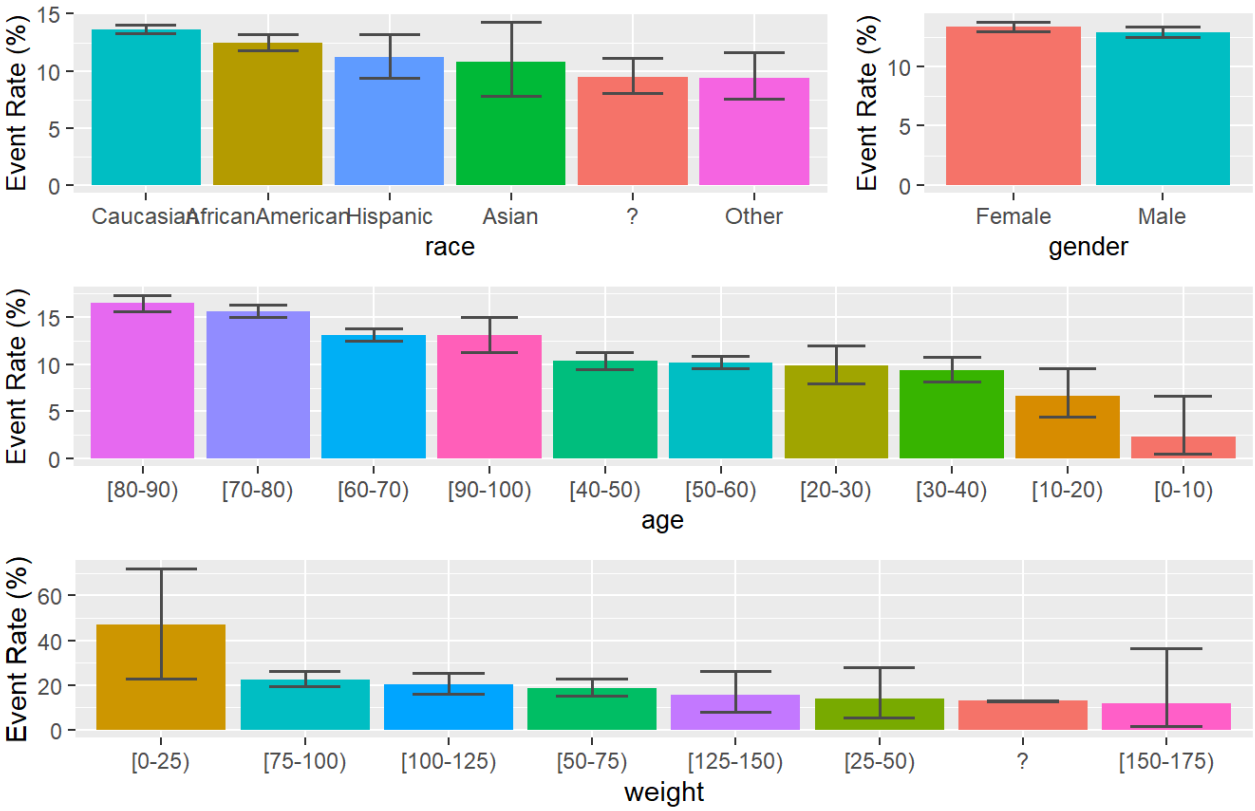
c) 5 Claim rates for individual features: Calculate and visualize the relative frequencies of the target variable `TARGET` (i.e., the readmission rates) for each categorical feature, similar to Sections 5.1 to 5.4 of the template. For numerical features, proceed as described in Sections 5.5 and 5.6. Use the functions `multiplot` and `get_binCI` , and group the visualizations according to the grouping defined in subtask R2b). Discuss the key insights from the visualizations, particularly in terms of their relevance for predicting the target variable readmission prediction.

We use the same grouping as in Task R2b):

- **Categorical features related to personal information** (e.g., gender, age)
- **Categorical features related to hospital stay details** (e.g., admission and discharge modalities)
- **Categorical features related to insurance and medical information** (e.g., specialty, insurance details)
- **Categorical features related to diagnosis groups**
- **Categorical features related to diabetes-specific information and medication** (e.g., test results, medication use)
- **Numerical features related to hospital resource use and patient health status** (e.g., number of procedures, length of hospital stay)

Categorical features related to personal information

```
p1 <- diabetic_data_bin %>%
  group_by(race, TARGET) %>%
  count() %>%
  spread(TARGET, n) %>%
  mutate(frac_claim = `1`/(`1`+`0`)*100,
         lwr = get_binCI(`1`,(`1`+`0`))[[1]]*100,
         upr = get_binCI(`1`,(`1`+`0`))[[2]]*100
         ) %>%
  ggplot(aes(reorder(race, -frac_claim, FUN = max), frac_claim, fill = race)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "race", y = "Event Rate (%)")

p2 <- diabetic_data_bin %>%
  group_by(gender, TARGET) %>%
  count() %>%
  spread(TARGET, n) %>%
  mutate(frac_claim = `1`/(`1`+`0`)*100,
         lwr = get_binCI(`1`,(`1`+`0`))[[1]]*100,
         upr = get_binCI(`1`,(`1`+`0`))[[2]]*100
         ) %>%
  ggplot(aes(reorder(gender, -frac_claim, FUN = max), frac_claim, fill = gender)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "gender", y = "Event Rate (%)")

p3 <- diabetic_data_bin %>%
  group_by(age, TARGET) %>%
  count() %>%
  spread(TARGET, n) %>%
  mutate(frac_claim = `1`/(`1`+`0`)*100,
         lwr = get_binCI(`1`,(`1`+`0`))[[1]]*100,
         upr = get_binCI(`1`,(`1`+`0`))[[2]]*100
         ) %>%
  ggplot(aes(reorder(age, -frac_claim, FUN = max), frac_claim, fill = age)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "age", y = "Event Rate (%)")

p4 <- diabetic_data_bin %>%
  group_by(weight, TARGET) %>%
  count() %>%
  spread(TARGET, n) %>%
  mutate(frac_claim = `1`/(`1`+`0`)*100,
         lwr = get_binCI(`1`,(`1`+`0`))[[1]]*100,
         upr = get_binCI(`1`,(`1`+`0`))[[2]]*100
         ) %>%
  ggplot(aes(reorder(weight, -frac_claim, FUN = max), frac_claim, fill = weight)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  theme(legend.position = "none") +
  labs(x = "weight", y = "Event Rate (%)")

layout <- matrix(c(1,1,2,3,3,3,4,4,4),3,3,byrow=TRUE)
multiplot(p1, p2, p3, p4, layout=layout)
```

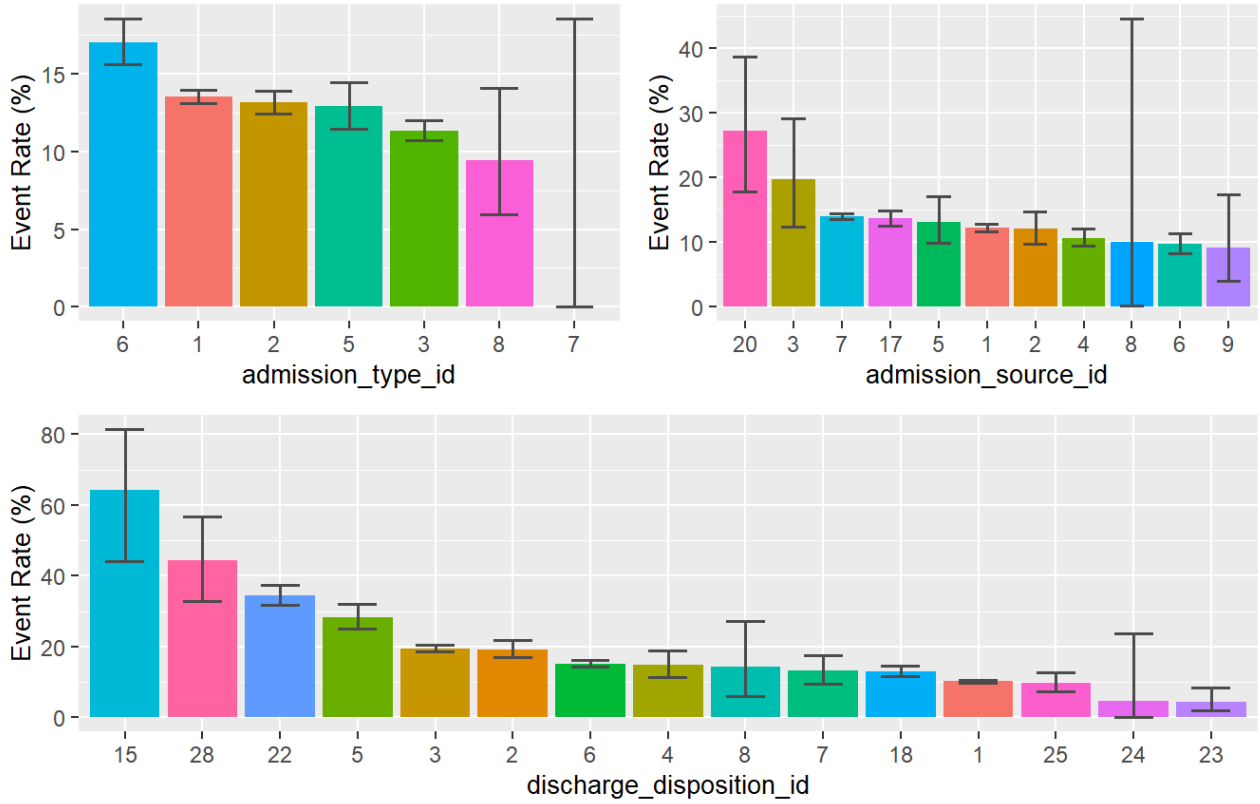


Key Insights:

- `race` : The highest readmission rate is observed for the most frequent group, `Caucasian` , followed by `AfricanAmerican` . Other categories have moderate rates and occur much less frequently.
- `gender` : The distribution is nearly balanced (`Female: 0.134` , `Male: 0.129`), with slightly higher readmission rates for females. This suggests the feature likely has little impact on the target variable.
- `age` : Readmission rates tend to increase with age, with the highest rate in the `[80-90)` age group. Younger age groups show very low readmission rates.
- `weight` : Over 90% of the values are missing, so no strong conclusions can be drawn. Additionally, in the extreme ranges (`[0-25)` and `[150-175)`), the confidence intervals are very large.

Categorical features related to hospital stay details

```
p1 <- diabetic_data_bin %>%
  group_by(admission_type_id, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(admission_type_id, -frac_claim, FUN = max), frac_claim, fill = admission_type_id)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "admission_type_id", y = "Event Rate (%)")

p2 <- diabetic_data_bin %>%
  group_by(admission_source_id, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(admission_source_id, -frac_claim, FUN = max), frac_claim, fill = admission_source_id)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "admission_source_id", y = "Event Rate (%)")

p3 <- diabetic_data_bin %>%
  group_by(discharge_disposition_id, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(discharge_disposition_id, -frac_claim, FUN = max), frac_claim, fill = discharge_disposition_id)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "discharge_disposition_id", y = "Event Rate (%)")

layout <- matrix(c(1,2,3,3),2,2,byrow=TRUE)
multiplot(p1, p2, p3, layout=layout)
```



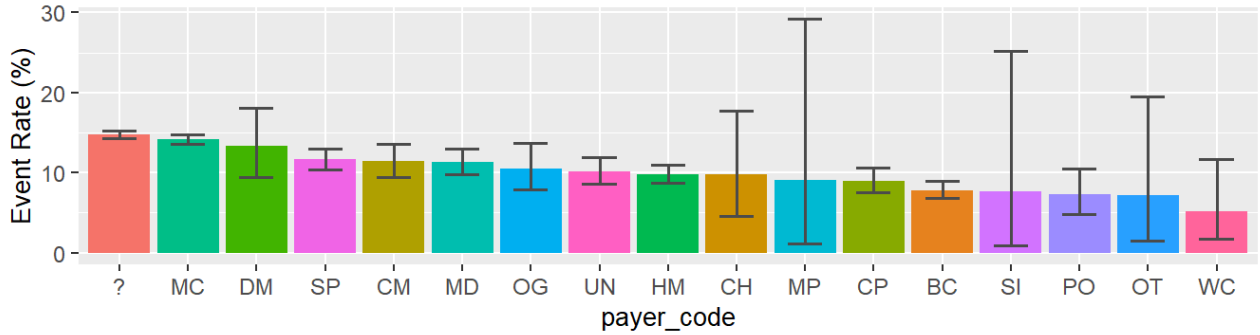
Key Insights:

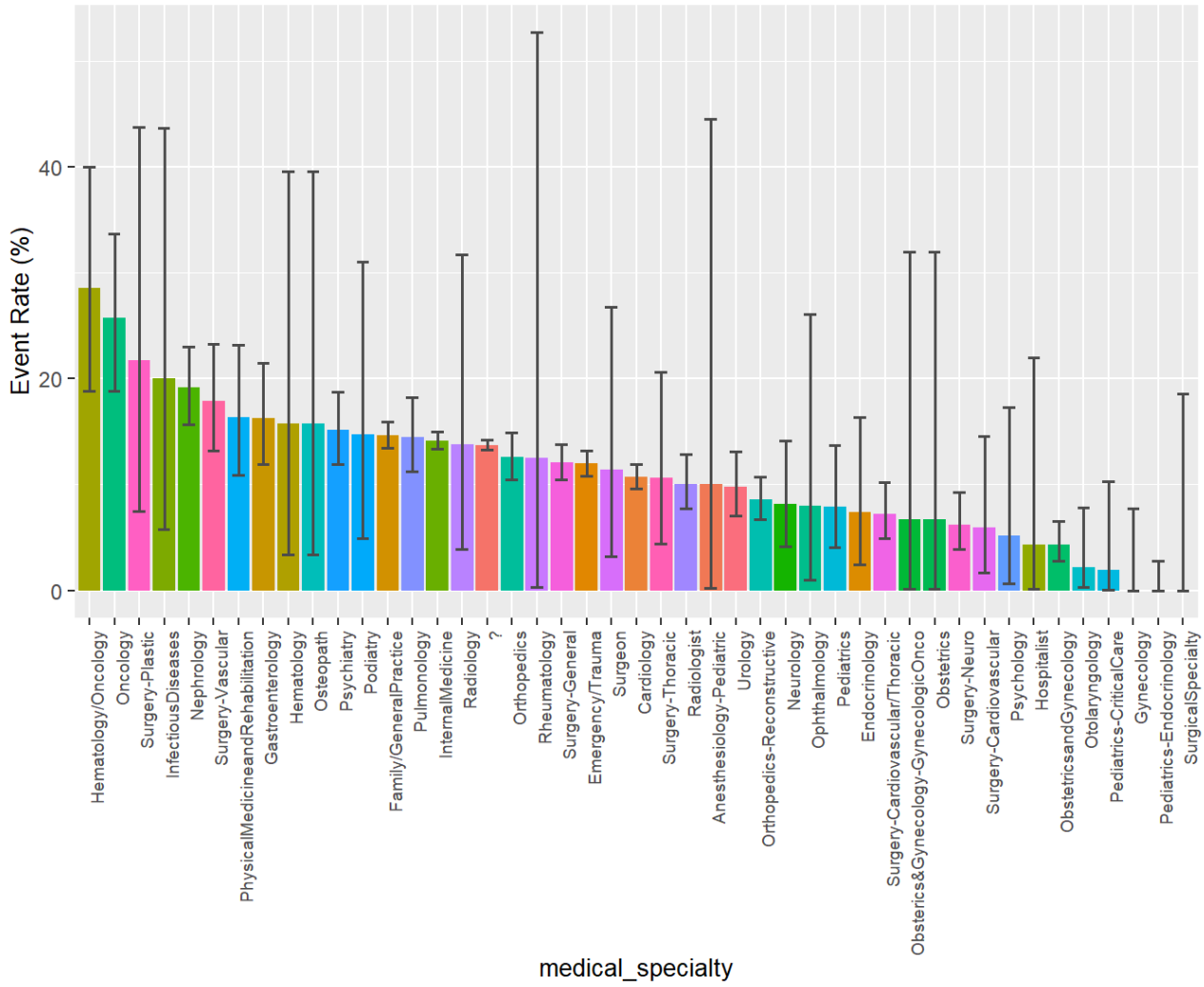
- `admission_type_id` : Patients admitted as `NULL` (category `6`) show the highest readmission rate (0.170), possibly indicating special admission circumstances. Most common types are `Emergency` (`1` , 0.135), `Urgent` (`2` , 0.132), and `Elective` (`3` , 0.113), with the latter showing a lower rate. Patients from trauma centers (7) show no readmissions.
- `admission_source_id` : The source `Not Mapped` (20) has the highest readmission rate (0.273) but is rare and associated with wide confidence intervals. Common categories like `Emergency Room` (`7` , 0.140), `Physician Referral` (`1` , 0.122), and `NULL` (17, 0.137) show average rates. The notable differences between sources suggest this feature could be valuable for prediction.

- `discharge_disposition_id` : Patients discharged to facilities such as Medicare-approved swing beds (category `15` , 0.643), psychiatric hospitals (`28` , 0.444), or rehab clinics (`22` , 0.346) have significantly higher readmission rates than those discharged home (most common value, `1` , 0.102). This suggests discharge type is a strong predictor of readmission. Rare categories like discharge to long-term care facilities (`23` , 0.0442) or Medicaid-certified but not Medicare-certified facilities (`24` , 0.0476) show much lower rates.

Categorical features related to insurance and medical information

```
p1 <- diabetic_data_bin %>%
  group_by(payer_code, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(payer_code, -frac_claim, FUN = max), frac_claim, fill = payer_code)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "payer_code", y = "Event Rate (%)")

p2 <- diabetic_data_bin %>%
  group_by(medical_specialty, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(medical_specialty, -frac_claim, FUN = max), frac_claim, fill = medical_specialty)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "medical_specialty", y = "Event Rate (%)") +
  theme(
    legend.position = "none",
    axis.text.x = element_text(size = 7, angle = 90, hjust = 1),  # Set x-axis text size
    axis.text.y = element_text(size = 9)  # Keep y-axis text size consistent with common theme
  )

layout <- matrix(c(1,1,2,2,2,2,2,2),4,2,byrow=TRUE)
multiplot(p1, p2, layout=layout)
```



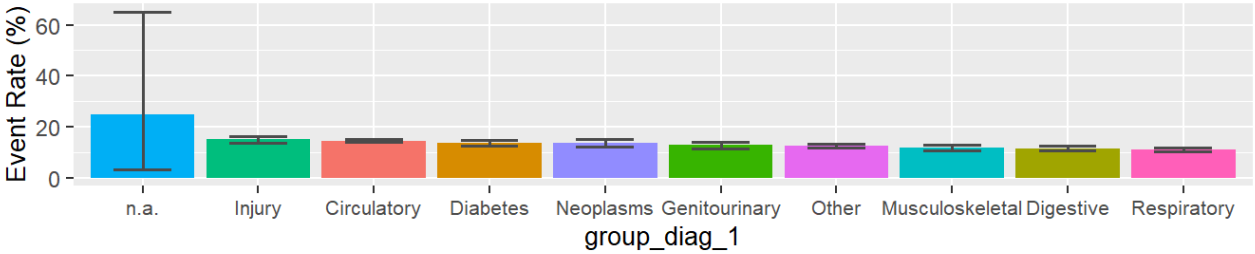


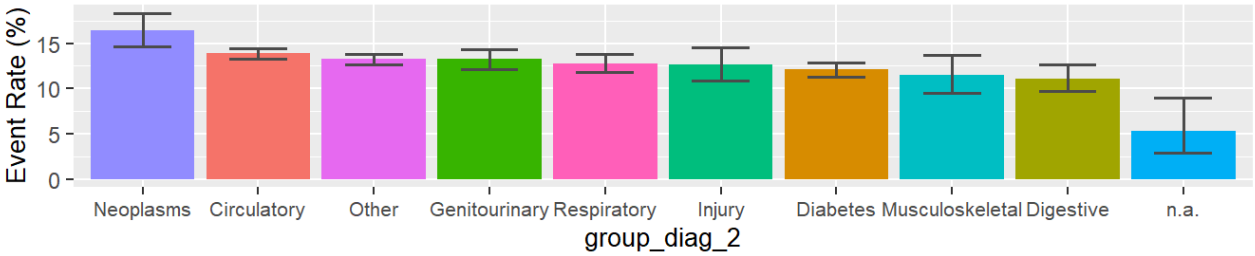
Key Insights:

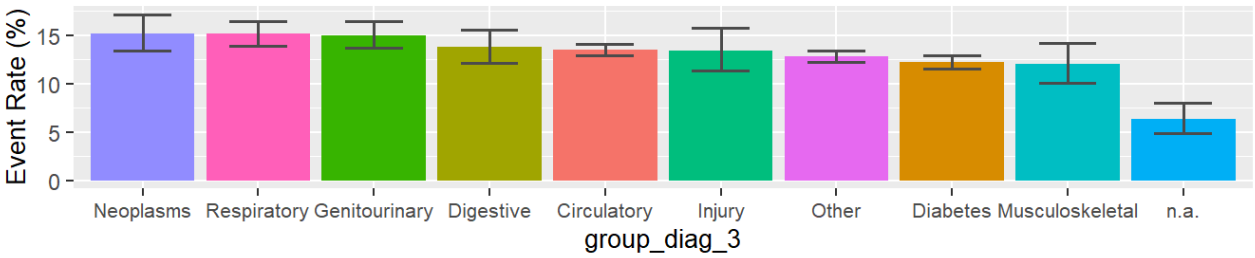
- `payer_code` : The most common value is `?` (missing), which shows a slightly elevated readmission rate (14.8%). The second most frequent value, Medicare (`MC` , 27.5%), shows similar behavior. `DM` and `SP` have comparatively higher readmission rates (13.3% and 11.6%) at moderate frequency. Rare categories such as `WC` have unusually low rates (5.2%). While the high proportion of missing values is problematic, certain categories like `DM` and `SP` may offer predictive value.
- `medical_specialty` : The category `?` dominates with 47.4% and a medium readmission rate (13.7%), while the also frequent `InternalMedicine` is slightly higher (14.2%). Less common specialties such as `Oncology` and `Surgery-Plastic` show high readmission rates (25.7% and 21.7%), whereas specialties like `ObstetricsandGynecology` have extremely low rates (4.4%). Specialized fields such as `Oncology` and `Surgery-Plastic` may significantly contribute to prediction, while those with very low rates might be excluded due to minimal influence.

Categorical features related to diagnosis groups

```r
p1 <- diabetic_data_bin %>%
  group_by(group_diag_1, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(group_diag_1, -frac_claim, FUN = max), frac_claim, fill = group_diag_1)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "group_diag_1", y = "Event Rate (%)") +
  theme(axis.text.x = element_text(size = 8))

p2 <- diabetic_data_bin %>%
  group_by(group_diag_2, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(group_diag_2, -frac_claim, FUN = max), frac_claim, fill = group_diag_2)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "group_diag_2", y = "Event Rate (%)") +
  theme(axis.text.x = element_text(size = 8))

p3 <- diabetic_data_bin %>%
  group_by(group_diag_3, TARGET) %>%
  count() %>%
  spread(TARGET, n, fill = 0) %>%
  mutate(frac_claim = `1` / (`1` + `0`) * 100,
         lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,
         upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%
  ggplot(aes(reorder(group_diag_3, -frac_claim, FUN = max), frac_claim, fill = group_diag_3)) +
  geom_col() +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +
  common_theme +
  labs(x = "group_diag_3", y = "Event Rate (%)") +
  theme(axis.text.x = element_text(size = 8))

layout <- matrix(c(1, 1, 2, 2, 3, 3), 3, 2, byrow = TRUE)

multiplot(p1, p2, p3, layout = layout)
```







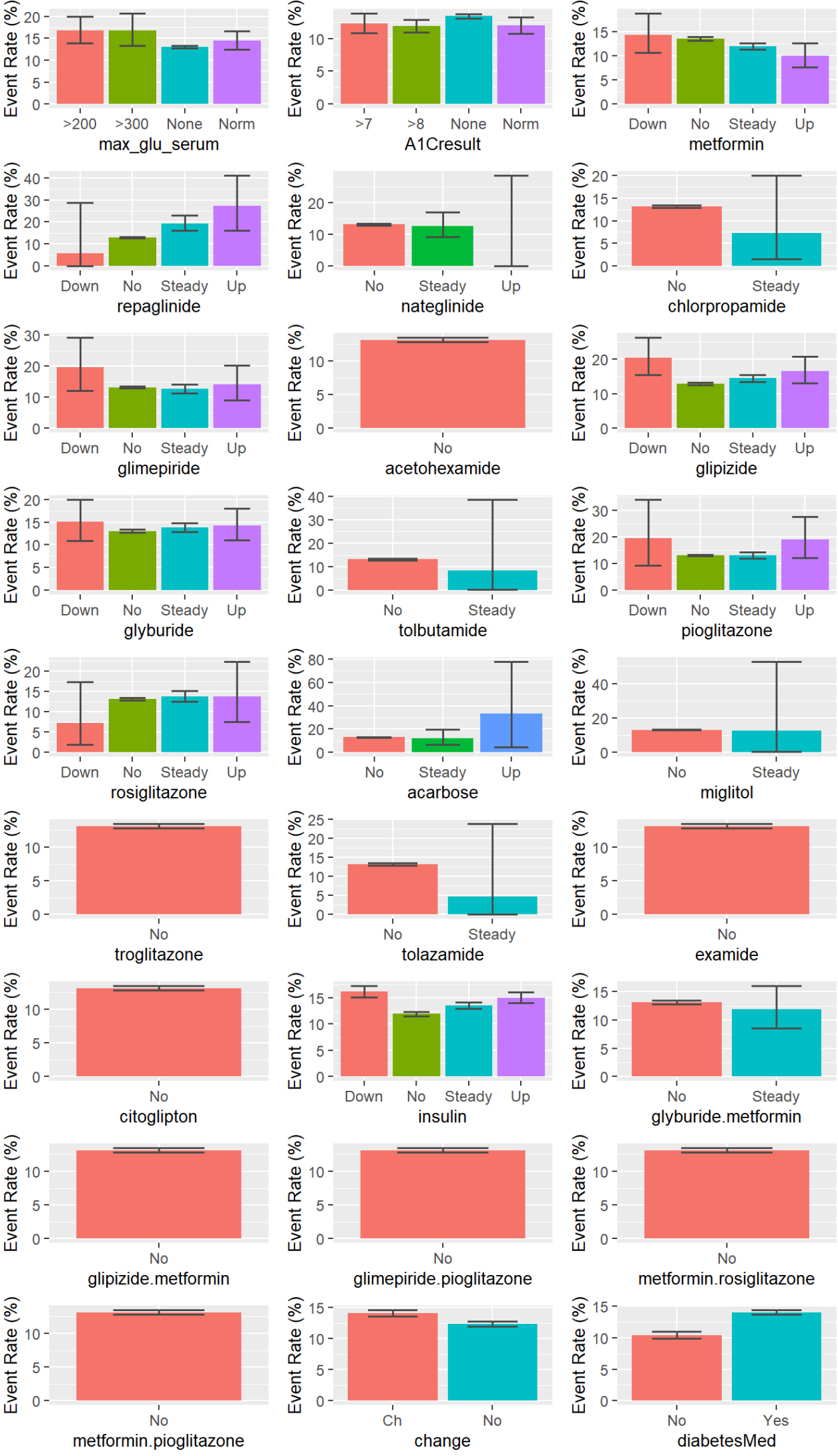
Key Insights:

- `group_diag_1` : The most common diagnosis groups `Circulatory` , `Other` , and `Respiratory` show readmission rates of 14.4%, 12.6%, and 11.1% respectively. The highest rate appears in the `Injury` category (15%) and in `n.a.` (25%), although the latter is extremely rare and thus statistically insignificant.

- `group_diag_2` : Diagnoses in frequent groups such as `Circulatory` (13.9%), `Other` (13.2%), and `Respiratory` (12.8%) have average readmission rates. The highest rate is observed for `Neoplasms` (16.4%), while `n.a.` is particularly low (5.35%) but also very rare.
- `group_diag_3` : The most common diagnosis groups `Circulatory` (13.5%), `Other` (12.8%), and `Diabetes` (12.2%) show unremarkable readmission rates. Above-average rates occur in `Neoplasms` (15.2%), `Respiratory` (15.2%), and `Genitourinary` (15.0%). Categories such as `n.a.` (6.31%) have very low rates but are also rare.

Categorical features related to diabetes-specific information and medication

```
# Function to create individual plots with fractions and error bars
create_plot <- function(data, feature) {
  data %>%
    group_by(.data[[feature]], TARGET) %>%  # Group by the feature and TARGET
    count() %>%
    spread(TARGET, n, fill = 0) %>%  # Spread the TARGET variable
    mutate(frac_claim = `1` / (`1` + `0`) * 100,  # Calculate fraction
           lwr = get_binCI(`1`, (`1` + `0`))[[1]] * 100,  # Lower confidence interval
           upr = get_binCI(`1`, (`1` + `0`))[[2]] * 100) %>%  # Upper confidence interval
    ggplot(aes_string(x = feature, y = "frac_claim", fill = feature)) +
    geom_col() +  # Use geom_col for bars
    geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "gray30") +  # Add error bars
    scale_x_discrete(drop = FALSE) +  # Keep all levels
    common_theme +
    labs(x = feature, y = "Event Rate (%)")  # Update axis labels
}

# Generate plots and store them in a list
plots <- lapply(features_to_plot, create_plot, data = diabetic_data_bin)

# Define the layout matrix (3 rows, 2 columns)
layout <- matrix(1:27, nrow = 9, ncol = 3, byrow = TRUE)

# Use multiplot to display the plots with the specified layout
multiplot(plotlist = plots, layout = layout)
```



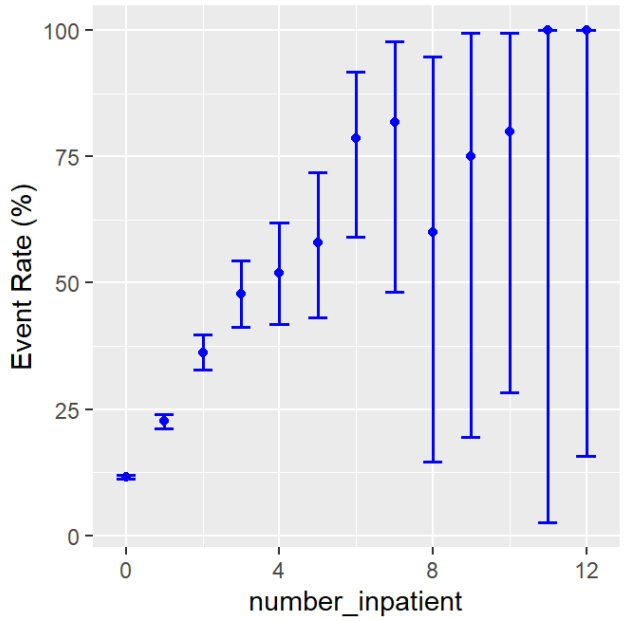
Key Insights:

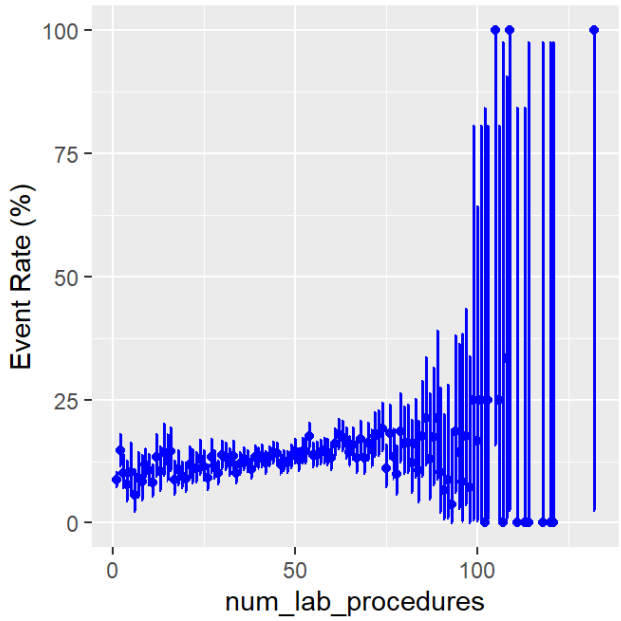
- `max_glu_serum` and `A1Cresult`: The majority of values are coded as `None`, indicating missing or unperformed tests. Higher categories such as `>200` and `>300` (or `>7` for `A1Cresult`) have increased readmission rates (e.g., `max_glu_serum >200`: 16.8%).
- `metformin`, `repaglinide`, and `nateglinide`: While `No` is the dominant status, certain categories like `Up` or `Steady` show notable readmission rates (`repaglinide Up`: 27.3%, `metformin Up`: 9.93%). Some medications like `nateglinide` are used rarely but have high rates in specific statuses.
- Rare or barely used medications: Drugs like `chlorpropamide`, `tolbutamide`, and combinations such as `glimepiride.pioglitazone` and `metformin.pioglitazone` appear almost exclusively as `No`. These likely provide no predictive value.
- Medications with variable statuses: Drugs like `glimepiride`, `glipizide`, `glyburide`, and `pioglitazone` have non-negligible counts in `Down`, `Steady`, and `Up`, with high readmission rates often observed in `Down` (`glimepiride Down`: 19.6%, `glipizide Down`: 20.5%) and `Up` (`glipizide Up`: 16.6%).
- `insulin`: This diabetes drug shows notable variation across categories, with `Down` (16.2%) and `Up` (15.0%) being particularly associated with elevated readmission rates.
- `change` and `diabetesMed`: The nearly evenly distributed `change` feature indicates higher readmission rates when medication was changed (`Ch`: 14.1%), suggesting dynamic treatment schemes. Patients on diabetes medication (`Yes`) have a significantly higher readmission rate (14.0%) than those without (`No`: 10.5%).
- Medications without status variability: Features like `troglitazone`, `examide`, and `citoglipton` only have the `No` status and contribute no analytical value. These could be removed to improve efficiency and model performance.
- Distinctive readmission patterns in specific categories: Drugs like `pioglitazone` (`Up`: 19.1%) and rarely used options such as `acarbose` (`Up`: 33.3%) show very high readmission rates and may warrant special attention.

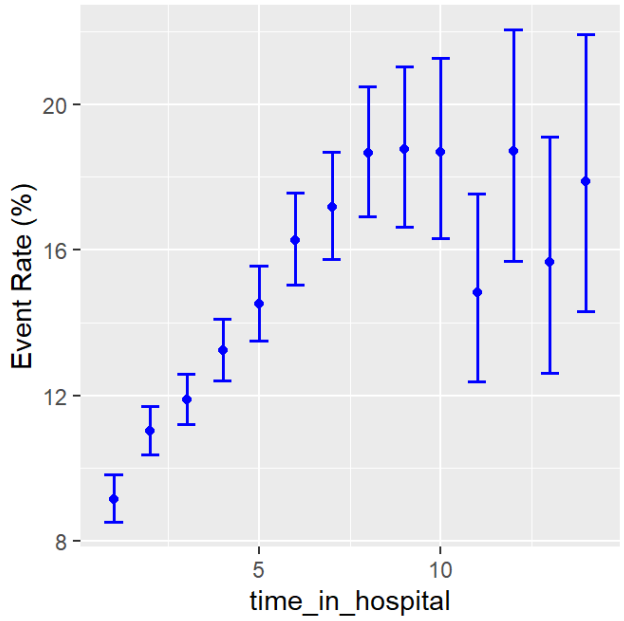
In summary, the analysis shows a dominance of the `No` category in many features, but specific statuses with higher readmission probabilities offer targeted insights into medication usage or adjustments. Features with no variability could be excluded from modeling.

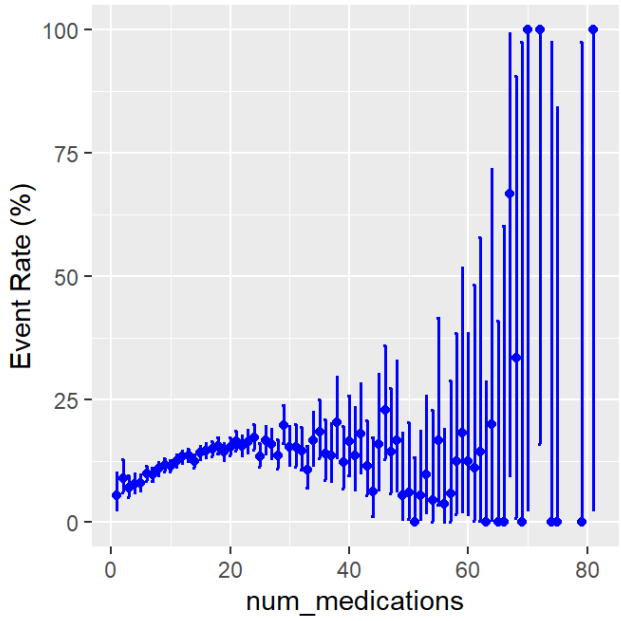
Numerical features related to hospital resource use and patient health status

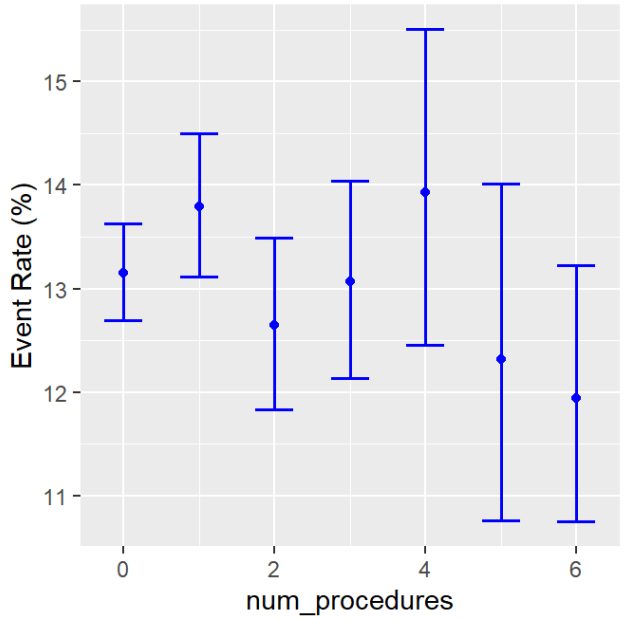
```
# Function to create individual plots with error bars
create_errorbar_plot <- function(data, feature) {
  data %>%
    group_by(!!sym(feature), TARGET) %>%
    count() %>%
    spread(TARGET, n, fill = 0) %>%
    mutate(frac_claim = `1`/(`1`+`0`)*100,
           lwr = get_binCI(`1`,(`1`+`0`))[[1]]*100,
           upr = get_binCI(`1`,(`1`+`0`))[[2]]*100) %>%
    ggplot(aes(x = !!sym(feature), y = frac_claim)) +
    geom_point(color = "blue") +
    geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.5, size = 0.7, color = "blue") +
    theme(legend.position = "none") +
    labs(x = feature, y = "Event Rate (%)")
}

# Define the features to plot
features <- c("number_inpatient", "num_lab_procedures", "time_in_hospital",
              "num_medications", "num_procedures", "number_diagnoses",
              "number_emergency", "number_outpatient")

# Generate the plots and store them in a list
plots <- lapply(features, create_errorbar_plot, data = diabetic_data_bin)

# Define the layout matrix (4 rows, 2 columns)
layout <- matrix(1:8, 4, 2, byrow=TRUE)

# Use multiplot to display the plots
multiplot(plotlist = plots, layout=layout)
```

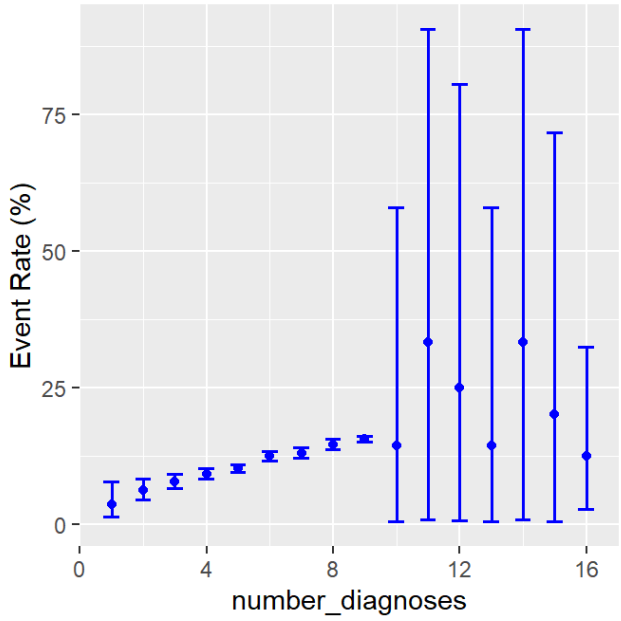

Event Rate (%)
number_inpatient

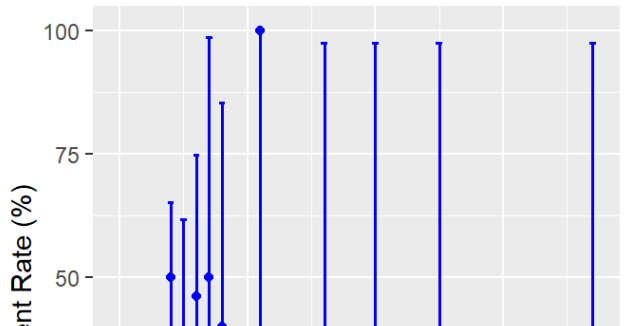

Event Rate (%)
num_lab_procedures

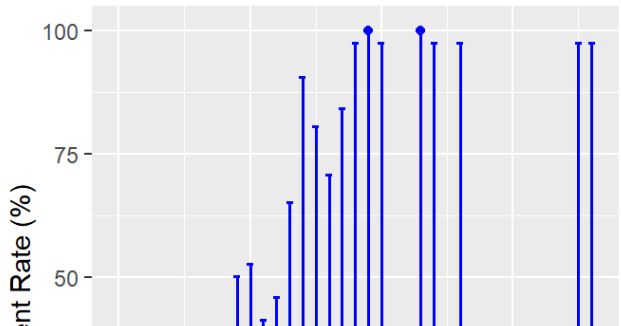

Event Rate (%)
time_in_hospital

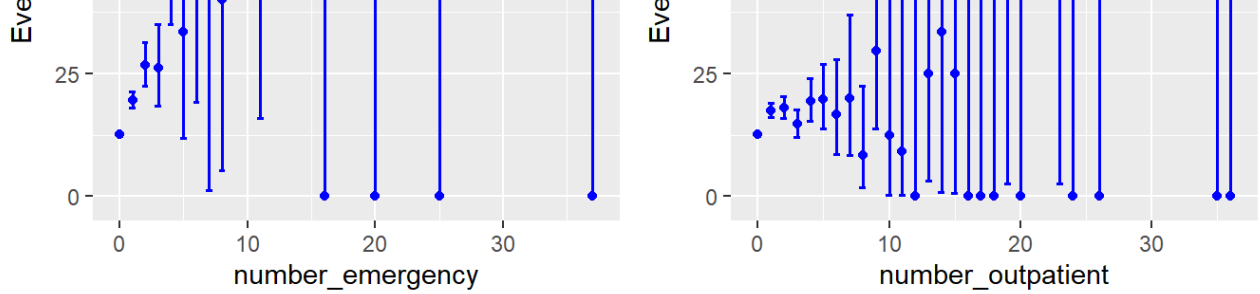

Event Rate (%)
num_medications


Event Rate (%)
num_procedures


Event Rate (%)
number_diagnoses







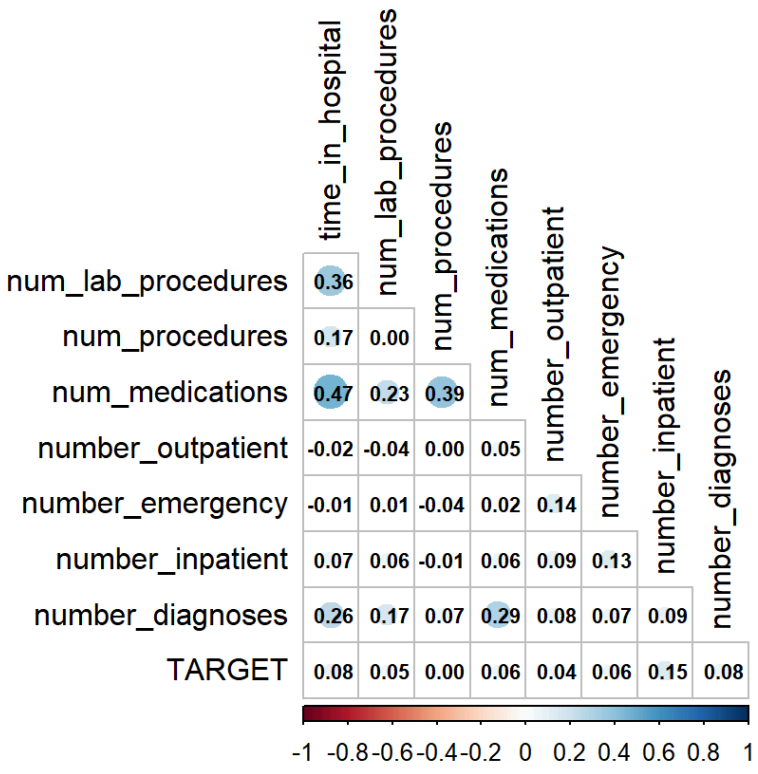
Key Insights:

- `number_inpatient` : Readmission rates increase steadily with the number of inpatient stays, reaching 78.6% at 6 stays. However, data becomes sparse from 4 stays onward, limiting reliability. This feature appears to be a strong predictor of readmission.
- `num_lab_procedures` : Higher lab procedure counts tend to correspond with increased readmission rates, though variability increases and confidence intervals widen at high values.
- `time_in_hospital` : Longer hospital stays are generally associated with higher readmission rates, particularly between 6–10 days.
- `num_medications` : A higher number of medications generally correlates with increased readmission rates, though the variance increases substantially at very high (and rare) values.
- `num_procedures` : Readmission rates show little fluctuation, ranging from 12.0% to 13.9%, with the highest rate at 4 procedures and the lowest at 6. This suggests low predictive value.
- `number_diagnoses` : More diagnoses tend to be associated with higher readmission rates, with a marked increase for 9 or more diagnoses, though these values have wider confidence intervals.
- `number_emergency` : Readmission rates rise with the number of emergency visits—from 12.6% at 0 emergencies to 46.2% at 6. There's a notable spike at 4 emergencies (50%), though data becomes sparse from 3 visits onward, increasing uncertainty.
- `number_outpatient` : Patients with 1 to 4 outpatient visits show the highest readmission rates, with a notable peak at 9 visits.

d) 6 Multi-feature comparisons: Create a correlation matrix similar to Section 6.1 of the template. Discuss the key insights, especially with regard to their relevance for predicting the target variable.

First, we compute the correlation matrix of the numerical features (including `TARGET` , which must first be encoded numerically), and then visualize it.

```
# Use numerically encoded TARGET in a temporary dataset
data_tmp <- diabetic_data_bin %>%
  mutate(TARGET = as.numeric(TARGET))

# Select only numerical features for the correlation matrix
numeric_features <- data_tmp %>%
  select_if(is.numeric)

# Compute correlation matrix
cor_matrix <- cor(numeric_features, use = "complete.obs", method = "spearman")

# Visualize the correlation matrix
corrplot(cor_matrix, method = "circle", type = "lower", tl.col = "black",
         addCoef.col = "black", number.cex = 0.7, diag = FALSE)
```



Key Insights:

- `TARGET` (readmission) shows only weak correlations with other variables. The strongest correlation is with `number_inpatient`, suggesting this feature may have a non-negligible impact on predicting 30-day readmissions.
- `time_in_hospital` exhibits moderate positive correlations with `num_medications`, `num_lab_procedures`, and `number_diagnoses`. This implies that patients with longer hospital stays tend to have more complex medical conditions requiring additional diagnostics and treatments.
- `num_lab_procedures` and `num_procedures` show a weak to moderate positive correlation. This further suggests that more extensive diagnostic and procedural activity may be associated with a higher likelihood of readmission.

Overall, while most numerical features show only limited correlation with the target, their interrelationships highlight shared clinical context (e.g., disease severity or care complexity), which may be informative in multivariate modeling.

Task R3: Data Splitting and XGBoost

In this task, you will split the two datasets `diabetic_data_bin` (for binary classification) and `diabetic_data_ter` (for ternary classification) into training, validation, and test datasets for subsequent modeling and apply a first machine learning model with XGBoost for binary classification. Starting from subtask R3b), code snippets from the *SWoF* template must be extracted and modified in certain places. The goal of these modifications is to ensure that the code runs with minimal adjustments, using the training, validation, and test datasets prepared in subtask R3a) as input. For this and all subsequent tasks related to binary classification, the "area under the ROC curve" (AUC) is to be used as the evaluation metric for machine learning models. At the beginning of each relevant subtask, the corresponding section of the *SWoF* template is indicated in bold.

a) Data Splitting: Split the two datasets `diabetic_data_bin` and `diabetic_data_ter` into training, validation, and test datasets in a 70:15:15 ratio. Ensure that the target variable `TARGET` is correctly stratified during the splitting process to maintain class distribution. Output the number of rows in the training, validation, and test datasets, as well as the class proportions in each subsets.

To evaluate later models, we split the datasets `diabetic_data_bin` and `diabetic_data_ter` into training, validation, and test sets, using a 70/15/15 split. This ensures the models are trained on a large portion of the data while still being evaluated on unseen samples to assess their performance.

```
# Function to split a dataset into training, validation, and test sets
split_data <- function(data, target_variable, train_ratio = 0.7, val_ratio = 0.15) {
  # First split: Training and remaining (Validation + Test)
  train_idx <- createDataPartition(data[[target_variable]], p = train_ratio, list = FALSE)
  train <- data[train_idx, ]
  rest <- data[-train_idx, ]

  # Second split: Validation and Test
  val_idx <- createDataPartition(rest[[target_variable]], p = val_ratio / (1 - train_ratio), list = FALSE)
  validation <- rest[val_idx, ]
  test <- rest[-val_idx, ]

  list(train = train, validation = validation, test = test)
}

# Apply split for binary and ternary classification
splits_binary <- split_data(diabetic_data_bin, "TARGET")
splits_ternary <- split_data(diabetic_data_ter, "TARGET")

# Assign individual datasets
train_bin <- splits_binary$train
val_bin <- splits_binary$validation
test_bin <- splits_binary$test

train_ter <- splits_ternary$train
val_ter <- splits_ternary$validation
test_ter <- splits_ternary$test

# Function to print split information
print_split_info <- function(split, title) {
  cat(title, "Classification:\n")
  cat("Train:", nrow(split$train), "Validation:", nrow(split$validation), "Test:", nrow(split$test), "\n")
  cat("TARGET distribution (Train):", prop.table(table(split$train$TARGET)), "\n")
  cat("TARGET distribution (Validation):", prop.table(table(split$validation$TARGET)), "\n")
  cat("TARGET distribution (Test):", prop.table(table(split$test$TARGET)), "\n\n")
}

# Output results
print_split_info(splits_binary, "Binary")
```

```
## Binary Classification:
## Train: 33426 Validation: 7163 Test: 7162
## TARGET distribution (Train): 0.8685454 0.1314546
## TARGET distribution (Validation): 0.8684909 0.1315091
## TARGET distribution (Test): 0.8686121 0.1313879
```

```
print_split_info(splits_ternary, "Ternary")
```

```
## Ternary Classification:
## Train: 48982 Validation: 10496 Test: 10495
## TARGET distribution (Train): 0.08970642 0.3175861 0.5927075
## TARGET distribution (Validation): 0.08974848 0.3175495 0.592702
## TARGET distribution (Test): 0.08966174 0.3175798 0.5927585
```

The output confirms that the split was performed correctly. For both the binary and ternary classification tasks, the training, validation, and test sets have very similar class distributions.

b) 8.3 XGBoost parameters and fitting: Use the code from the indicated section of the template and ensure that it runs correctly. Adjust the code and parameters where necessary and appropriate to properly train the XGBoost model. Briefly discuss the role of the hyperparameters `colsample_bytree`, `subsample`, `max_depth`, and `eta` in the context of XGBoost, and compute the AUC score on the test dataset.

In this task, we train an XGBoost model to predict hospital readmissions. First, we convert the data into the appropriate format for XGBoost. We then tune the model parameters, perform cross-validation to determine the optimal number of rounds, and train the model. Finally, we evaluate its performance using the AUC metric.

We begin by converting the training and test datasets into the `DMatrix` format, which XGBoost uses for efficient computation.

```
X_train <- train_bin %>% select(-TARGET)
X_val <- val_bin %>% select(-TARGET)
X_test <- test_bin %>% select(-TARGET)

# Convert TARGET to binary (0/1) by subtracting 1
y_train <- as.numeric(train_bin$TARGET) - 1
y_val <- as.numeric(val_bin$TARGET) - 1
y_test <- as.numeric(test_bin$TARGET) - 1

# Convert training, validation, and test data to DMatrix format
dtrain <- xgb.DMatrix(data.matrix(X_train), label = y_train)
dval <- xgb.DMatrix(data.matrix(X_val), label = y_val)
dtest <- xgb.DMatrix(data.matrix(X_test))
```

Next, we define the parameters of the XGBoost model. The parameters are interpreted as follows:

- `colsample_bytree` : Proportion of features used for training each tree. A value of 0.7 means 70% of features are randomly selected, which reduces the risk of overfitting.
- `subsample` : Fraction of training data used for each tree. A value of 0.7 introduces diversity across trees.
- `max_depth` : Maximum depth of each tree. Larger values allow more complex patterns but increase the risk of overfitting.
- `eta` : Learning rate. Smaller values like 0.1 result in slower but more stable learning.

Note that we use `eval_metric = "auc"` for binary classification.

```
# Define XGBoost parameters
xgb_params <- list(
  colsample_bytree = 0.7,
  subsample = 0.7,
  booster = "gbtree",
  max_depth = 5,
  eta = 0.1,
  eval_metric = "auc",
  objective = "binary:logistic",
  nthread = 4
)

watchlist <- list(train = dtrain, valid = dval)
```

We then perform cross-validation to determine the optimal number of boosting rounds.

```
# Perform cross-validation
xgb_cv <- xgb.cv(
  params = xgb_params,
  data = dtrain,
  early_stopping_rounds = 10,
  nfold = 5,
  nrounds = 100,
  maximize = TRUE,
  verbose = 0
)

# Store optimal number of rounds
optimal_nrounds <- xgb_cv$best_iteration
```

We now train the model using the optimal number of rounds.

```
# Train the XGBoost model
gb_dt <- xgb.train(
  params = xgb_params,
  data = dtrain,
  print_every_n = 10,
  watchlist = watchlist,
  nrounds = optimal_nrounds,
  maximize = TRUE
)
```

```
## [1]  train-auc:0.668300  valid-auc:0.648974
## [11] train-auc:0.700069  valid-auc:0.679814
## [21] train-auc:0.707864  valid-auc:0.686211
## [31] train-auc:0.718680  valid-auc:0.690010
## [41] train-auc:0.733844  valid-auc:0.694602
## [51] train-auc:0.744228  valid-auc:0.699344
## [61] train-auc:0.752023  valid-auc:0.700282
## [71] train-auc:0.761650  valid-auc:0.702049
## [81] train-auc:0.768298  valid-auc:0.702361
## [91] train-auc:0.775590  valid-auc:0.701436
## [92] train-auc:0.776191  valid-auc:0.701548
```

To evaluate model performance, we compute the AUC on the test data.

```
# Compute and output AUC using the calculate_auc function
auc_xgb <- calculate_auc(y_test, predict(gb_dt, newdata = dtest))
```

```
## AUC: 0.7035
```

The XGBoost model was successfully trained and achieved a solid AUC score of approximately 0.7.

c) 8.4 Feature Importance: Compute the feature importances of the XGBoost model just trained and visualize them using an appropriate plot. Identify the most important features and provide a brief qualitative discussion on why they may be relevant for predicting readmissions (no specific knowledge of health insurance or healthcare is required). Finally, briefly outline the potential implications of these findings for model development, predictive performance, and the design of preventive insurance strategies.

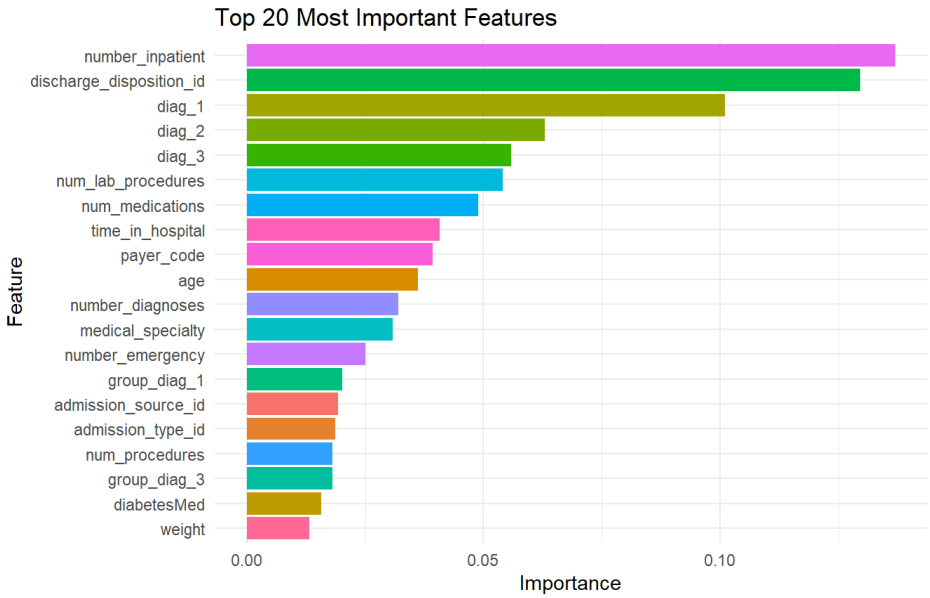
Feature importance analysis allows us to identify the most influential variables for prediction. This provides insights into the data and supports model interpretability.

We first compute feature importance using the `xgb.importance` function and store the results in a tibble:

```
# Compute feature importance
imp_matrix <- as_tibble(xgb.importance(
  feature_names = colnames(X_train),
  model = gb_dt
))
```

To visualize the results, we plot the top 20 most important features in a bar chart:

```
# Visualize the top 20 important features
imp_matrix %>%
  head(20) %>%
  ggplot(aes(x = reorder(Feature, Gain, FUN = max), y = Gain, fill = Feature)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  labs(
    title = "Top 20 Most Important Features",
    x = "Feature",
    y = "Importance"
  ) +
  theme_minimal()
```



We now discuss the most important features and their relevance for predicting hospital readmissions.

Key Features:

- `number_inpatient` : The number of inpatient visits in the year prior to the index admission reflects the patient's health status and complexity.
- `discharge_disposition_id` : Indicates the patient's condition upon discharge, such as transfer to a care facility or discharge to home.
- Diagnoses (`diag_1` , `diag_2` , `diag_3`): Critical for assessing the severity and type of illness.
- `num_lab_procedures` : The number of lab tests performed gives insight into diagnostic complexity and severity. A high count may indicate serious or difficult-to-diagnose conditions.
- `num_medications` : The number of prescribed medications is a proxy for treatment intensity and complexity. Patients on many medications may be at higher risk of adverse drug reactions or poor adherence, increasing the likelihood of readmission.

These features are logically connected to readmission risk. For instance, more frequent hospital stays or a higher number of lab procedures and medications suggest a greater disease burden. Discharge reasons and diagnoses provide specific information about the patient's condition post-discharge and the likelihood of complications.

The findings also suggest that model accuracy could be further improved by focusing on these features. The interpretability of key features can help uncover and prioritize interventions (see Task R4) to reduce hospital readmissions.

Task R4: Logistic Regressions Without and With Interactions

Following the examination of an XGBoost model for binary classification in Task R3, this task focuses on Logistic Regression. Both a Logistic Regression without interactions and a model with interactions will be explored. The training and test datasets prepared in subtask R3a) serve as input data.

a) Logistic Regression Without Interactions: Train a basic Logistic Regression that uses all relevant predictors (without interactions). Select only features that have at least two distinct values, and exclude the target variable as well as the diagnosis columns (`diag_1` , `diag_2` , `diag_3`). Briefly discuss, in four to five bullet points, the key insights derived from the coefficients of the resulting Logistic Regression. Compute the AUC score of this model on the test dataset and output the result. Generate and visualize a confusion matrix that compares the model's predictions with the actual values in the test dataset. Before constructing the confusion matrix, briefly discuss the challenges of choosing an appropriate threshold, particularly in balancing sensitivity and specificity. Then, use a threshold of 0.2 to generate the matrix.

We now implement a simple logistic regression using all relevant predictors, excluding interactions. Only features with at least two unique values are selected. The target variable and the diagnosis columns `diag_1` , `diag_2` , and `diag_3` are excluded (the latter due to a large number of unique values).

```
# Select only features with more than one unique value
valid_features <- colnames(train_bin)[sapply(train_bin, function(col) length(unique(col)) > 1)]

# Exclude target variable and diagnosis columns
valid_features <- setdiff(valid_features, c("TARGET", "diag_1", "diag_2", "diag_3"))

# Create formula using valid features
formula <- as.formula(paste("TARGET ~", paste(valid_features, collapse = " + ")))

# Fit logistic regression model
glm1 <- glm(formula, data = train_bin, family = binomial)
```

We first display the coefficients of the logistic regression model:

```
# Model summary
summary(glm1)
```

```
##
## Call:
## glm(formula = formula, family = binomial, data = train_bin)
##
## Coefficients:
##                                                  Estimate Std. Error
## (Intercept)                                    -1.734e+01  4.310e+02
## raceAfricanAmerican                             2.410e-01  1.189e-01
## raceAsian                                       2.064e-01  2.187e-01
## raceCaucasian                                   2.383e-01  1.128e-01
## raceHispanic                                   -1.098e-01  1.727e-01
## raceOther                                      -1.664e-01  1.858e-01
## genderMale                                     -1.513e-02  3.476e-02
## age[10-20)                                      4.379e-01  7.945e-01
## age[20-30)                                      4.268e-01  7.776e-01
## age[30-40)                                      3.537e-01  7.704e-01
## age[40-50)                                      5.438e-01  7.667e-01
## age[50-60)                                      4.041e-01  7.662e-01
## age[60-70)                                      6.382e-01  7.661e-01
## age[70-80)                                      7.401e-01  7.663e-01
## age[80-90)                                      6.366e-01  7.669e-01
## age[90-100)                                     3.949e-01  7.730e-01
## weight[0-25)                                    1.979e+00  6.288e-01
## weight[25-50)                                  -1.341e-02  4.721e-01
## weight[50-75)                                   2.329e-01  1.632e-01
## weight[75-100)                                  7.185e-01  1.274e-01
## weight[100-125)                                 7.958e-01  1.792e-01
## weight[125-150)                                 2.925e-01  4.138e-01
## weight[150-175)                                -1.337e+01  4.230e+02
## admission_type_id2                              5.563e-02  6.871e-02
## admission_type_id3                             -1.415e-01  7.969e-02
## admission_type_id5                              6.433e-02  1.194e-01
## admission_type_id6                              4.046e-01  9.789e-02
## admission_type_id7                             -1.344e+01  3.970e+02
## admission_type_id8                             -3.725e-01  3.030e-01
## discharge_disposition_id2                       6.012e-01  9.796e-02
## discharge_disposition_id3                       5.621e-01  5.516e-02
## discharge_disposition_id4                       3.630e-01  1.861e-01
## discharge_disposition_id5                       1.246e+00  1.116e-01
## discharge_disposition_id6                       2.744e-01  5.611e-02
## discharge_disposition_id7                       1.672e-01  2.261e-01
## discharge_disposition_id8                       5.256e-01  4.575e-01
## discharge_disposition_id15                      2.590e+00  5.682e-01
## discharge_disposition_id18                      2.038e-01  9.346e-02
## discharge_disposition_id22                      1.526e+00  8.874e-02
## discharge_disposition_id23                     -1.002e+00  4.228e-01
## discharge_disposition_id24                     -1.341e+01  4.147e+02
## discharge_disposition_id25                     -5.287e-01  2.119e-01
## discharge_disposition_id28                      1.969e+00  3.085e-01
## admission_source_id2                           -4.336e-01  1.576e-01
## admission_source_id3                            6.816e-01  2.870e-01
## admission_source_id4                           -4.632e-01  1.005e-01
## admission_source_id5                           -6.588e-01  2.020e-01
## admission_source_id6                           -2.201e-01  1.346e-01
## admission_source_id7                           -4.405e-02  7.022e-02
## admission_source_id8                            3.211e-01  1.137e+00
## admission_source_id9                            9.877e-02  4.517e-01
## admission_source_id17                          -4.984e-01  1.084e-01
## admission_source_id20                           7.077e-01  3.336e-01
## time_in_hospital                                9.672e-03  7.109e-03
## payer_codeBC                                   -4.426e-01  9.589e-02
## payer_codeCH                                   -8.684e-01  6.016e-01
## payer_codeCM                                   -5.265e-01  1.342e-01
## payer_codeCP                                   -3.890e-01  1.183e-01
## payer_codeDM                                   -3.408e-01  2.542e-01
## payer_codeHM                                   -2.174e-01  8.397e-02
## payer_codeMC                                   -2.620e-01  4.714e-02
## payer_codeMD                                   -1.430e-01  1.067e-01
## payer_codeMP                                   -2.255e-01  1.052e+00
## payer_codeOG                                   -1.419e-01  1.866e-01
## payer_codeOT                                   -8.230e-01  7.488e-01
## payer_codePO                                   -3.693e-01  2.360e-01
## payer_codeSI                                    8.250e-02  7.613e-01
## payer_codeSP                                   -1.362e-01  9.072e-02
## payer_codeUN                                   -3.001e-01  1.168e-01
## payer_codeWC                                   -9.847e-01  6.050e-01
## medical_specialtyAnesthesiology-Pediatric       3.862e-01  1.102e+00
## medical_specialtyCardiology                    -2.110e-01  8.361e-02
## medical_specialtyEmergency/Trauma              -1.171e-01  8.707e-02
## medical_specialtyEndocrinology                 -1.005e-01  4.845e-01
## medical_specialtyFamily/GeneralPractice         6.226e-02  6.876e-02
## medical_specialtyGastroenterology               3.827e-01  2.145e-01
## medical_specialtyGynecology                    -1.315e+01  2.469e+02
## medical_specialtyHematology                    -1.710e-01  7.836e-01
## medical_specialtyHematology/Oncology            6.077e-01  3.568e-01
```

```
## medical_specialtyHospitalist                          -3.666e-01  1.062e+00
## medical_specialtyInfectiousDiseases                    1.028e+00  6.061e-01
## medical_specialtyInternalMedicine                     -8.061e-02  5.269e-02
## medical_specialtyNephrology                            5.818e-01  1.484e-01
## medical_specialtyNeurology                            -3.646e-01  3.858e-01
## medical_specialtyObsterics&Gynecology-GynecologicOnco -1.375e+01  4.084e+02
## medical_specialtyObstetrics                           -1.308e+01  5.120e+02
## medical_specialtyObstetricsandGynecology              -6.278e-01  2.542e-01
## medical_specialtyOncology                              6.500e-01  2.379e-01
## medical_specialtyOphthalmology                        -4.669e-01  1.041e+00
## medical_specialtyOrthopedics                          -6.010e-02  1.437e-01
## medical_specialtyOrthopedics-Reconstructive           -8.398e-01  1.819e-01
## medical_specialtyOsteopath                             5.132e-01  6.708e-01
## medical_specialtyOtolaryngology                       -1.327e+01  1.800e+02
## medical_specialtyPediatrics                           -7.850e-01  4.482e-01
## medical_specialtyPediatrics-CriticalCare              -1.299e+01  2.408e+02
## medical_specialtyPediatrics-Endocrinology             -1.288e+01  1.525e+02
## medical_specialtyPhysicalMedicineandRehabilitation     6.026e-01  2.584e-01
## medical_specialtyPodiatry                              6.708e-03  6.376e-01
## medical_specialtyPsychiatry                            4.080e-01  1.683e-01
## medical_specialtyPsychology                           -1.057e+00  1.026e+00
## medical_specialtyPulmonology                           7.809e-02  1.667e-01
## medical_specialtyRadiologist                          -3.168e-01  1.833e-01
## medical_specialtyRadiology                             1.068e-01  6.427e-01
## medical_specialtyRheumatology                         -1.349e+01  7.129e+02
## medical_specialtySurgeon                              -3.843e-01  7.575e-01
## medical_specialtySurgery-Cardiovascular               -1.362e+00  7.314e-01
## medical_specialtySurgery-Cardiovascular/Thoracic      -6.686e-01  2.334e-01
## medical_specialtySurgery-General                      -1.240e-01  1.030e-01
## medical_specialtySurgery-Neuro                        -9.295e-01  3.094e-01
## medical_specialtySurgery-Plastic                       2.888e-02  8.112e-01
## medical_specialtySurgery-Thoracic                     -4.355e-01  5.393e-01
## medical_specialtySurgery-Vascular                      4.034e-01  2.131e-01
## medical_specialtySurgicalSpecialty                    -1.305e+01  4.156e+02
## medical_specialtyUrology                               3.596e-03  2.013e-01
## num_lab_procedures                                     1.009e-03  1.073e-03
## num_procedures                                        -1.365e-02  1.197e-02
## num_medications                                        6.129e-03  2.804e-03
## number_outpatient                                      2.066e-02  1.613e-02
## number_emergency                                       1.862e-01  3.574e-02
## number_inpatient                                       5.194e-01  2.664e-02
## number_diagnoses                                       5.896e-02  1.121e-02
## max_glu_serum>300                                      2.510e-03  2.206e-01
## max_glu_serumNone                                     -9.950e-02  1.621e-01
## max_glu_serumNorm                                      1.530e-01  1.800e-01
## A1Cresult>8                                           -8.322e-03  1.038e-01
## A1CresultNone                                          8.455e-02  8.726e-02
## A1CresultNorm                                          2.737e-04  1.110e-01
## metforminNo                                            1.583e-02  2.047e-01
## metforminSteady                                       -1.159e-01  2.046e-01
## metforminUp                                           -3.605e-01  2.664e-01
## repaglinideNo                                          1.355e+01  4.310e+02
## repaglinideSteady                                      1.375e+01  4.310e+02
## repaglinideUp                                          1.414e+01  4.310e+02
## nateglinideSteady                                     -7.337e-02  2.199e-01
## nateglinideUp                                         -1.500e+01  4.065e+02
## chlorpropamideSteady                                  -1.416e+00  1.038e+00
## glimepirideNo                                         -1.498e-01  3.581e-01
## glimepirideSteady                                     -2.594e-01  3.640e-01
## glimepirideUp                                         -1.671e-01  4.488e-01
## glipizideNo                                           -3.432e-01  2.112e-01
## glipizideSteady                                       -3.214e-01  2.118e-01
## glipizideUp                                           -1.720e-01  2.660e-01
## glyburideNo                                            1.394e-01  2.199e-01
## glyburideSteady                                        1.416e-01  2.199e-01
## glyburideUp                                           -1.882e-02  2.781e-01
## tolbutamideSteady                                     -8.027e-01  1.061e+00
## pioglitazoneNo                                         3.024e-02  4.751e-01
## pioglitazoneSteady                                    -5.866e-02  4.783e-01
## pioglitazoneUp                                         3.362e-01  5.579e-01
## rosiglitazoneNo                                        7.238e-01  6.247e-01
## rosiglitazoneSteady                                    7.724e-01  6.275e-01
## rosiglitazoneUp                                        7.623e-01  7.181e-01
## acarboseSteady                                        -5.144e-02  3.502e-01
## acarboseUp                                            -1.336e+01  1.022e+03
## miglitolSteady                                        -1.379e+01  7.014e+02
## tolazamideSteady                                      -1.365e+01  4.161e+02
## insulinNo                                             -1.452e-01  9.243e-02
## insulinSteady                                         -1.464e-01  7.064e-02
## insulinUp                                             -9.112e-02  7.260e-02
## glyburide.metforminSteady                              7.333e-02  2.076e-01
## changeNo                                               3.720e-02  6.446e-02
## diabetesMedYes                                         3.181e-01  6.277e-02
## group_diag_1Diabetes                                   2.248e-02  7.454e-02
## group_diag_1Digestive                                 -1.533e-01  7.051e-02
## group_diag_1Genitourinary                             -1.315e-01  8.559e-02
```

```
## group_diag_1Injury                          -2.101e-01  7.436e-02
## group_diag_1Musculoskeletal                 -2.615e-01  9.568e-02
## group_diag_1n.a.                             7.644e-01  8.948e-01
## group_diag_1Neoplasms                        3.086e-02  8.708e-02
## group_diag_1Other                           -2.186e-01  5.760e-02
## group_diag_1Respiratory                     -2.944e-01  6.168e-02
## group_diag_2Diabetes                         1.524e-01  6.053e-02
## group_diag_2Digestive                       -1.246e-01  1.012e-01
## group_diag_2Genitourinary                   -1.353e-01  7.051e-02
## group_diag_2Injury                          -1.291e-01  1.108e-01
## group_diag_2Musculoskeletal                 -1.284e-01  1.358e-01
## group_diag_2n.a.                            -1.035e-01  4.016e-01
## group_diag_2Neoplasms                        1.389e-01  9.602e-02
## group_diag_2Other                            7.959e-03  4.838e-02
## group_diag_2Respiratory                     -4.582e-02  6.462e-02
## group_diag_3Diabetes                         8.035e-02  5.306e-02
## group_diag_3Digestive                        1.158e-01  9.677e-02
## group_diag_3Genitourinary                    5.448e-02  7.516e-02
## group_diag_3Injury                          -4.534e-02  1.233e-01
## group_diag_3Musculoskeletal                  2.003e-02  1.245e-01
## group_diag_3n.a.                            -3.128e-02  1.849e-01
## group_diag_3Neoplasms                        4.652e-02  9.947e-02
## group_diag_3Other                           -3.052e-02  4.673e-02
## group_diag_3Respiratory                      4.111e-02  7.213e-02
##                                             z value Pr(>|z|)
## (Intercept)                                  -0.040 0.967908
## raceAfricanAmerican                           2.027 0.042631 *
## raceAsian                                     0.944 0.345285
## raceCaucasian                                 2.114 0.034537 *
## raceHispanic                                 -0.636 0.524903
## raceOther                                    -0.896 0.370332
## genderMale                                   -0.435 0.663327
## age[10-20)                                    0.551 0.581501
## age[20-30)                                    0.549 0.583102
## age[30-40)                                    0.459 0.646108
## age[40-50)                                    0.709 0.478137
## age[50-60)                                    0.527 0.597950
## age[60-70)                                    0.833 0.404777
## age[70-80)                                    0.966 0.334145
## age[80-90)                                    0.830 0.406510
## age[90-100)                                   0.511 0.609445
## weight[0-25)                                  3.148 0.001646 **
## weight[25-50)                                -0.028 0.977343
## weight[50-75)                                 1.427 0.153594
## weight[75-100)                                5.642 1.68e-08 ***
## weight[100-125)                               4.441 8.95e-06 ***
## weight[125-150)                               0.707 0.479654
## weight[150-175)                              -0.032 0.974778
## admission_type_id2                            0.810 0.418105
## admission_type_id3                           -1.775 0.075842 .
## admission_type_id5                            0.539 0.590055
## admission_type_id6                            4.133 3.58e-05 ***
## admission_type_id7                           -0.034 0.972995
## admission_type_id8                           -1.229 0.218901
## discharge_disposition_id2                     6.137 8.42e-10 ***
## discharge_disposition_id3                    10.190  < 2e-16 ***
## discharge_disposition_id4                     1.951 0.051064 .
## discharge_disposition_id5                    11.165  < 2e-16 ***
## discharge_disposition_id6                     4.891 1.00e-06 ***
## discharge_disposition_id7                     0.740 0.459591
## discharge_disposition_id8                     1.149 0.250639
## discharge_disposition_id15                    4.558 5.17e-06 ***
## discharge_disposition_id18                    2.180 0.029252 *
## discharge_disposition_id22                   17.190  < 2e-16 ***
## discharge_disposition_id23                   -2.369 0.017817 *
## discharge_disposition_id24                   -0.032 0.974200
## discharge_disposition_id25                   -2.496 0.012572 *
## discharge_disposition_id28                    6.383 1.73e-10 ***
## admission_source_id2                         -2.752 0.005927 **
## admission_source_id3                          2.375 0.017560 *
## admission_source_id4                         -4.609 4.04e-06 ***
## admission_source_id5                         -3.261 0.001109 **
## admission_source_id6                         -1.635 0.102016
## admission_source_id7                         -0.627 0.530472
## admission_source_id8                          0.282 0.777572
## admission_source_id9                          0.219 0.826929
## admission_source_id17                        -4.595 4.32e-06 ***
## admission_source_id20                         2.121 0.033885 *
## time_in_hospital                              1.360 0.173699
## payer_codeBC                                 -4.616 3.92e-06 ***
## payer_codeCH                                 -1.443 0.148885
## payer_codeCM                                 -3.924 8.72e-05 ***
## payer_codeCP                                 -3.289 0.001004 **
## payer_codeDM                                 -1.340 0.180132
## payer_codeHM                                 -2.589 0.009620 **
## payer_codeMC                                 -5.559 2.72e-08 ***
```

```
## payer_codeMD                                        -1.340 0.180280
## payer_codeMP                                        -0.214 0.830253
## payer_codeOG                                        -0.760 0.446957
## payer_codeOT                                        -1.099 0.271739
## payer_codePO                                        -1.565 0.117657
## payer_codeSI                                         0.108 0.913708
## payer_codeSP                                        -1.502 0.133205
## payer_codeUN                                        -2.569 0.010204 *
## payer_codeWC                                        -1.628 0.103604
## medical_specialtyAnesthesiology-Pediatric            0.350 0.726063
## medical_specialtyCardiology                         -2.524 0.011600 *
## medical_specialtyEmergency/Trauma                   -1.346 0.178457
## medical_specialtyEndocrinology                      -0.207 0.835719
## medical_specialtyFamily/GeneralPractice              0.905 0.365227
## medical_specialtyGastroenterology                    1.784 0.074359 .
## medical_specialtyGynecology                         -0.053 0.957505
## medical_specialtyHematology                         -0.218 0.827296
## medical_specialtyHematology/Oncology                 1.703 0.088578 .
## medical_specialtyHospitalist                        -0.345 0.729925
## medical_specialtyInfectiousDiseases                  1.696 0.089980 .
## medical_specialtyInternalMedicine                   -1.530 0.126071
## medical_specialtyNephrology                          3.921 8.82e-05 ***
## medical_specialtyNeurology                          -0.945 0.344560
## medical_specialtyObsterics&Gynecology-GynecologicOnco -0.034 0.973152
## medical_specialtyObstetrics                         -0.026 0.979615
## medical_specialtyObstetricsandGynecology            -2.470 0.013528 *
## medical_specialtyOncology                            2.732 0.006290 **
## medical_specialtyOphthalmology                      -0.449 0.653713
## medical_specialtyOrthopedics                        -0.418 0.675725
## medical_specialtyOrthopedics-Reconstructive         -4.618 3.88e-06 ***
## medical_specialtyOsteopath                           0.765 0.444182
## medical_specialtyOtolaryngology                     -0.074 0.941228
## medical_specialtyPediatrics                         -1.751 0.079866 .
## medical_specialtyPediatrics-CriticalCare            -0.054 0.956960
## medical_specialtyPediatrics-Endocrinology           -0.084 0.932670
## medical_specialtyPhysicalMedicineandRehabilitation   2.332 0.019697 *
## medical_specialtyPodiatry                            0.011 0.991605
## medical_specialtyPsychiatry                          2.424 0.015351 *
## medical_specialtyPsychology                         -1.030 0.302932
## medical_specialtyPulmonology                         0.469 0.639373
## medical_specialtyRadiologist                        -1.729 0.083852 .
## medical_specialtyRadiology                           0.166 0.868060
## medical_specialtyRheumatology                       -0.019 0.984899
## medical_specialtySurgeon                            -0.507 0.611946
## medical_specialtySurgery-Cardiovascular             -1.862 0.062600 .
## medical_specialtySurgery-Cardiovascular/Thoracic    -2.864 0.004178 **
## medical_specialtySurgery-General                    -1.205 0.228356
## medical_specialtySurgery-Neuro                      -3.005 0.002660 **
## medical_specialtySurgery-Plastic                     0.036 0.971604
## medical_specialtySurgery-Thoracic                   -0.807 0.419416
## medical_specialtySurgery-Vascular                    1.893 0.058385 .
## medical_specialtySurgicalSpecialty                  -0.031 0.974951
## medical_specialtyUrology                             0.018 0.985747
## num_lab_procedures                                   0.941 0.346769
## num_procedures                                      -1.140 0.254434
## num_medications                                      2.186 0.028834 *
## number_outpatient                                    1.281 0.200306
## number_emergency                                     5.209 1.90e-07 ***
## number_inpatient                                    19.499  < 2e-16 ***
## number_diagnoses                                     5.259 1.45e-07 ***
## max_glu_serum>300                                    0.011 0.990923
## max_glu_serumNone                                   -0.614 0.539337
## max_glu_serumNorm                                    0.850 0.395527
## A1Cresult>8                                         -0.080 0.936087
## A1CresultNone                                        0.969 0.332584
## A1CresultNorm                                        0.002 0.998033
## metforminNo                                          0.077 0.938363
## metforminSteady                                     -0.566 0.571076
## metforminUp                                         -1.353 0.175955
## repaglinideNo                                        0.031 0.974917
## repaglinideSteady                                    0.032 0.974555
## repaglinideUp                                        0.033 0.973820
## nateglinideSteady                                   -0.334 0.738638
## nateglinideUp                                       -0.037 0.970568
## chlorpropamideSteady                                -1.364 0.172422
## glimepirideNo                                       -0.418 0.675812
## glimepirideSteady                                   -0.713 0.476091
## glimepirideUp                                       -0.372 0.709702
## glipizideNo                                         -1.625 0.104095
## glipizideSteady                                     -1.517 0.129210
## glipizideUp                                         -0.647 0.517819
## glyburideNo                                          0.634 0.526142
## glyburideSteady                                      0.644 0.519829
## glyburideUp                                         -0.068 0.946058
## tolbutamideSteady                                   -0.756 0.449354
## pioglitazoneNo                                       0.064 0.949246
```

```
## pioglitazoneSteady                                  -0.123 0.902387
## pioglitazoneUp                                       0.603 0.546737
## rosiglitazoneNo                                      1.159 0.246567
## rosiglitazoneSteady                                  1.231 0.218373
## rosiglitazoneUp                                      1.062 0.288446
## acarboseSteady                                      -0.147 0.883239
## acarboseUp                                          -0.013 0.989575
## miglitolSteady                                      -0.020 0.984313
## tolazamideSteady                                    -0.033 0.973831
## insulinNo                                           -1.570 0.116304
## insulinSteady                                       -2.073 0.038199 *
## insulinUp                                           -1.255 0.209460
## glyburide.metforminSteady                            0.353 0.723953
## changeNo                                             0.577 0.563853
## diabetesMedYes                                       5.067 4.03e-07 ***
## group_diag_1Diabetes                                 0.302 0.762968
## group_diag_1Digestive                               -2.174 0.029676 *
## group_diag_1Genitourinary                           -1.537 0.124372
## group_diag_1Injury                                  -2.825 0.004721 **
## group_diag_1Musculoskeletal                         -2.733 0.006279 **
## group_diag_1n.a.                                     0.854 0.392958
## group_diag_1Neoplasms                                0.354 0.723103
## group_diag_1Other                                   -3.795 0.000148 ***
## group_diag_1Respiratory                             -4.773 1.81e-06 ***
## group_diag_2Diabetes                                 2.518 0.011812 *
## group_diag_2Digestive                               -1.231 0.218319
## group_diag_2Genitourinary                           -1.919 0.054944 .
## group_diag_2Injury                                  -1.165 0.243856
## group_diag_2Musculoskeletal                         -0.946 0.344176
## group_diag_2n.a.                                    -0.258 0.796625
## group_diag_2Neoplasms                                1.446 0.148083
## group_diag_2Other                                    0.165 0.869326
## group_diag_2Respiratory                             -0.709 0.478276
## group_diag_3Diabetes                                 1.514 0.129999
## group_diag_3Digestive                                1.196 0.231514
## group_diag_3Genitourinary                            0.725 0.468518
## group_diag_3Injury                                  -0.368 0.713116
## group_diag_3Musculoskeletal                          0.161 0.872178
## group_diag_3n.a.                                    -0.169 0.865654
## group_diag_3Neoplasms                                0.468 0.640059
## group_diag_3Other                                   -0.653 0.513647
## group_diag_3Respiratory                              0.570 0.568676
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 26015  on 33425  degrees of freedom
## Residual deviance: 23900  on 33237  degrees of freedom
## AIC: 24278
##
## Number of Fisher Scoring iterations: 14
```

Insights from the Coefficients:

- `discharge_disposition_id` : Many categories are deemed significant, aligning with EDA findings from task R2.
- `number_emergency` , `number_inpatient` , `number_diagnoses` : Positive and significant coefficients indicate that higher counts are associated with increased readmission rates.
- `num_lab_procedures` : Interestingly, this feature, which ranked as the sixth most important in task R3c), is not deemed significant in logistic regression.
- `diag_1` , `diag_2` , `diag_3` : Although identified as highly important by XGBoost in task R3c), these features were excluded from the logistic regression due to the high number of categories, which could cause instability or overfitting.

Next, we compute the AUC score of the logistic regression model on the test data:

```
# Predict probabilities for test data
test_bin_prob <- predict(glm1, newdata = test_bin, type = "response")

# Compute AUC using the provided function
auc_logreg_without <- calculate_auc(test_bin$TARGET, test_bin_prob, positive_class = 1)
```

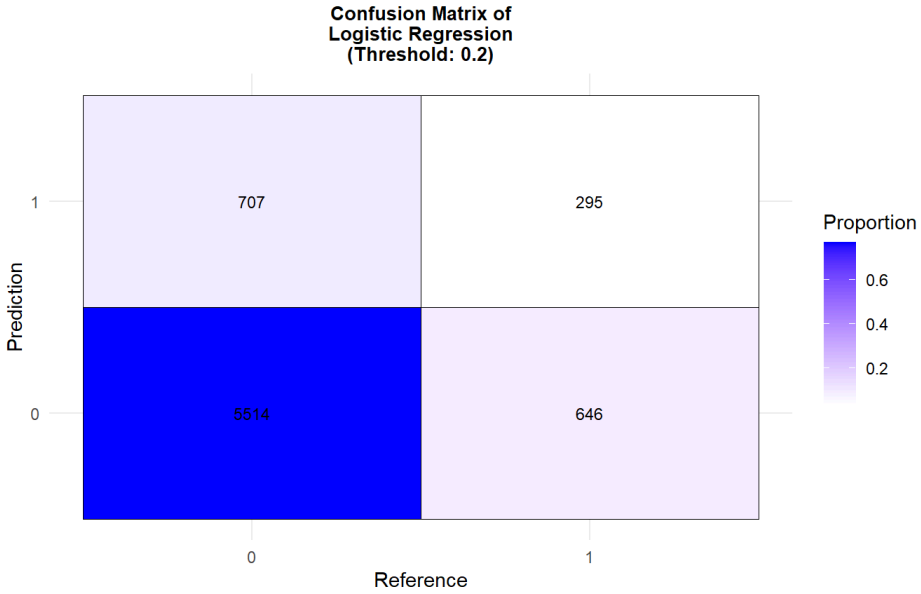
```
## AUC: 0.6887
```

Choosing an appropriate threshold is especially important in imbalanced datasets like this one. A high threshold may result in under-detection of the minority class (~13%), while a low threshold may misclassify too many majority class instances.

The threshold affects the trade-off between sensitivity (correctly identifying positives) and specificity (correctly identifying negatives). The optimal threshold depends on the use case and the cost of misclassification.

We now generate a confusion matrix using a threshold of 0.2:

```
# Predict class labels using threshold 0.2
test_bin_pred <- ifelse(test_bin_prob > 0.2, 1, 0)

# Generate confusion matrix
conf_matrix <- confusionMatrix(factor(test_bin_pred), factor(y_test))

# Visualize confusion matrix
plot_confusion_matrix(conf_matrix, "Confusion Matrix of\nLogistic Regression\n(Threshold: 0.2)")
```



(Alternative approach using `nnet`, as applied in the ternary classification:)

```
# Train logistic regression using nnet
logistic_model <- nnet::multinom(formula, data = train_bin, MaxNWts=1000000)
```

```
## # weights:  191 (190 variable)
## initial  value 23169.137657
## iter  10 value 14347.438195
## iter  20 value 13777.719291
## iter  30 value 13172.637854
## iter  40 value 12638.871275
## iter  50 value 12306.270894
## iter  60 value 12077.788395
## iter  70 value 11992.391944
## iter  80 value 11970.906152
## iter  90 value 11957.869795
## iter 100 value 11952.056036
## final  value 11952.056036
## stopped after 100 iterations
```

```
# Predict probabilities on the test set
probs <- predict(logistic_model, newdata = test_bin, type = "probs")

# Compute AUC using the provided function
auc_lr_nnet <- calculate_auc(y_test, probs, positive_class = 1)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
## AUC: 0.6892
```

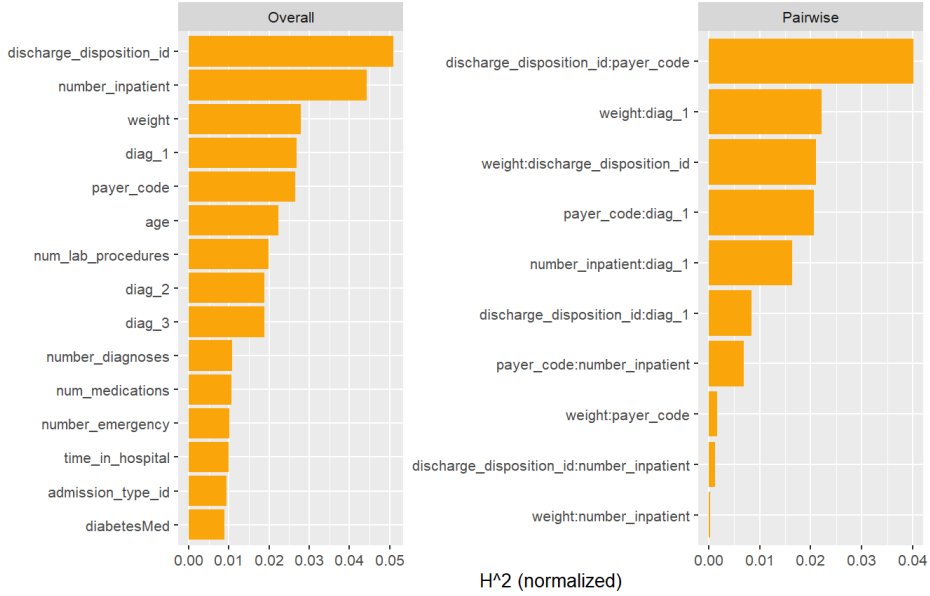
b) Logistic Regression With Interactions: Use, for example, the `hstats` package to identify interactions between the predictors of the XGBoost model from Task R3. Briefly explain in a few sentences how these interactions are detected. Select the three most significant interactions that do not involve `diag_1`, `diag_2`, or `diag_3`, and incorporate these interactions into the Logistic Regression from subtask R4a) by extending the regression formula accordingly. Train this enhanced model on the training dataset, compute its AUC score on the test dataset, and output the result. Generate and visualize a confusion matrix, analogous to subtask R4a).

To identify interactions between the predictors, we use the `hstats` package, which provides a method to estimate interactions among variables in a model, together with the XGBoost model from Task R3.

```
# Apply hstats to identify interactions
H <- hstats(gb_dt, X = data.matrix(X_train))
```

```
## 1-way calculations...
##   |                                                                      |
|   0%  |                                                                    |=
|   2%  |                                                                    |===
|   4%  |                                                                    |====
|   6%  |                                                                    |======
|   8%  |                                                                    |=======
|  10%  |                                                                    |========
|  12%  |                                                                    |==========
|  14%  |                                                                    |===========
|  16%  |                                                                    |=============
|  18%  |                                                                    |==============
|  20%  |                                                                    |================
|  22%  |                                                                    |==================
|  24%  |                                                                    |===================
|  26%  |                                                                    |====================
|  28%  |                                                                    |======================
|  30%  |                                                                    |=======================
|  32%  |                                                                    |========================
|  34%  |                                                                    |==========================
|  36%  |                                                                    |===========================
|  38%  |                                                                    |============================
|  40%  |                                                                    |==============================
|  42%  |                                                                    |================================
|  44%  |                                                                    |=================================
|  46%  |                                                                    |==================================
|  48%  |                                                                    |====================================
|  50%  |                                                                    |=====================================
|  52%  |                                                                    |=====================================
==                                |  54%  |                                                                  |==
====================================                                  |  56%  |
|=========================================                               |  58%  |
|=========================================                               |  60%  |
|=========================================                               |  62%  |
|=============================================                           |  64%  |
|=============================================                           |  66%  |
|=================================================                       |  68%  |
|=================================================                       |  70%  |
|=====================================================                   |  72%  |
|=====================================================                   |  74%  |
|=========================================================               |  76%  |
|=========================================================               |  78%  |
|=============================================================           |  80%  |
|=============================================================           |  82%  |
|=================================================================       |  84%  |
|=================================================================       |  86%  |
|=====================================================================   |  88%  |
|=====================================================================   |  90%  |
|=========================================================================|  92%  |
|=========================================================================|  94%  |
|=========================================================================|  96%  |
|=========================================================================| 98%  |
|=========================================================================|| 100%
## 2-way calculations...
##   |                                                                      |
|   0%  |                                                                    |=======
|  10%  |                                                                    |==============
|  20%  |                                                                    |=====================
|  30%  |                                                                    |============================
|  40%  |                                                                    |===================================
|  50%  |                                                                    |===================================
======                            |  60%  |                                                                  |==
================================================                      |  70%  |
|=================================================                       |  80%  |
|=================================================================       |  90%  |
|=========================================================================|| 100%
```

```
# Show the identified interactions from hstats
plot(H) + theme_gray(base_size=10)
```



The interactions between predictors are detected using the H-statistic method from the work of Friedman and Popescu. This method evaluates the strength of interactions by calculating the portion of prediction variance explained by the interactions among variables. For each variable and each pair of variables, it measures the contribution of their interaction to the model's predictions. Both overall interactions and pairwise interactions are computed, with the strength of the interaction represented as a proportion of explained variance. The most significant interactions indicate which features are most strongly related.

Next, we identify the three most important interactions that do not involve `diag_1`, `diag_2`, or `diag_3`. These interactions will be incorporated into the logistic regression formula.

```
# Extract the top three interactions excluding diag_1, diag_2, diag_3
top_interactions <- names(sort(h2_pairwise(H)$M[, 1], decreasing = TRUE))
top_interactions <- top_interactions[!grepl("diag_1|diag_2|diag_3", top_interactions)][1:3]
top_interactions <- gsub(":", " * ", top_interactions)
cat(paste("Top Interactions:", paste0(top_interactions, ", ")))
```

```
## Top Interactions: discharge_disposition_id * payer_code,  Top Interactions: weight * discharge_disposition_id,  Top Inter
actions: payer_code * number_inpatient,
```

```
# Combine the original valid features with the formatted interactions
all_predictors <- c(valid_features, top_interactions)

# Create the formula using the valid features and the top interactions
formula_with_interactions <- as.formula(paste("TARGET ~", paste(all_predictors, collapse = " + ")))
```

Now we train the logistic regression model using the extended formula that includes the identified interactions.

```
# Train logistic regression model with interactions
glm_with_interactions <- glm(formula_with_interactions, data = train_bin, family = binomial)

# Output the summary of the enhanced model
#summary(glm_with_interactions)
```

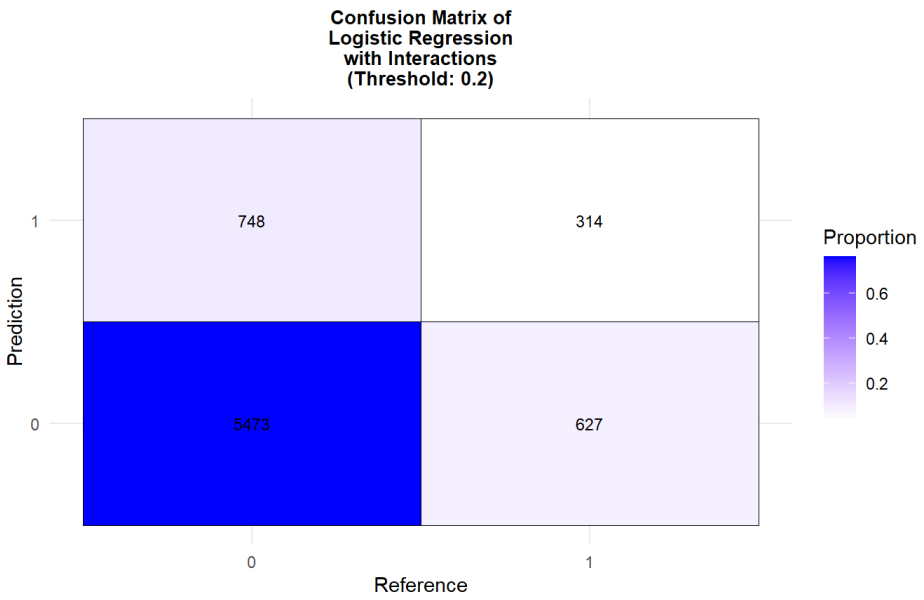
Next, we use the trained model to predict probabilities on the test set and compute the AUC score.

```
# Calculate predicted probabilities for the test data
test_prob <- predict(glm_with_interactions, newdata = test_bin, type = "response")

# Compute AUC using the calculate_auc function
auc_logreg_with <- calculate_auc(y_test, test_prob)
```

```
## AUC: 0.6896
```

Finally, we generate the corresponding confusion matrix.

```
# Predict class membership (threshold = 0.2)
test_bin_pred_with <- ifelse(test_prob > 0.2, 1, 0)

# Create confusion matrix
conf_matrix_with <- confusionMatrix(factor(test_bin_pred_with), factor(y_test))

# Visualize the confusion matrix
plot_confusion_matrix(conf_matrix_with, "Confusion Matrix of\nLogistic Regression\nwith Interactions\n(Threshold: 0.2)")
```



c) Model Comparison and Discussion: Compare the results of the Logistic Regressions without and with interactions. Discuss whether including interactions led to a significant improvement in model performance. Additionally, briefly discuss three approaches to integrating interactions differently or more effectively to further improve model quality.

The results of the two models show only a very slight improvement in model performance after including the interactions: The AUC of the logistic regression without interactions is 0.6887349, while the model with interactions achieves an AUC of 0.6896462. This change is marginal, suggesting that accounting for the interactions does not significantly improve model performance in this case.

Approach 1: Grouping Similar Features

One way to better integrate interactions is to group similar features in order to explicitly capture their interactions. If several features provide similar information or are relevant in the same context (e.g., `num_procedures` and `num_medications`), they could be grouped and their combinations included in the model. This can reveal potentially important interactions that would otherwise remain unaccounted for.

Approach 2: Aggregating Value Ranges

Another approach is to aggregate value ranges of continuous features to make interactions more visible. For example, `number_inpatient` could be divided into categories (e.g., 1-2, 3-5, 6+), allowing interactions between these groups and other variables (like `time_in_hospital`) to be captured and modeled more effectively. This type of categorization helps the model detect patterns in interactions that may be too complex with continuous variables.

Approach 3: Considering Higher-Order Interactions

The model could also be enhanced by considering higher-order interactions. Significant interactions often occur not only in simple pairs of features but also in combinations of multiple variables. For instance, interactions between `time_in_hospital`, `num_medications`, and `number_emergency` may be crucial. Introducing such interactions into the model could help uncover complex but important relationships that cannot be captured through lower-order interactions.

Task R5: Regularized Linear Models and GAMs

In this task, you will apply regularized linear models (L1 and L2 regularization) as well as Generalized Additive Models (GAMs) for modeling. The training and test datasets prepared in subtask R3a) will serve as input data. In all subtasks, use the same set of predictors as in subtask R4a).

a) L1 Regularization (LASSO): Apply the LASSO method (L1 regularization) to the training data. Determine the optimal regularization parameter lambda using cross-validation, visualize the regularization path, and output all non-zero coefficients. Briefly discuss three possible relationships between the non-zero coefficients of the LASSO model and the insights gained from the EDA in Task R2. Compute the AUC score of the model on the test data.

First, we create a representation of the predictors from the training and test data suitable for further processing. We reuse the formula defined in Task R4a).

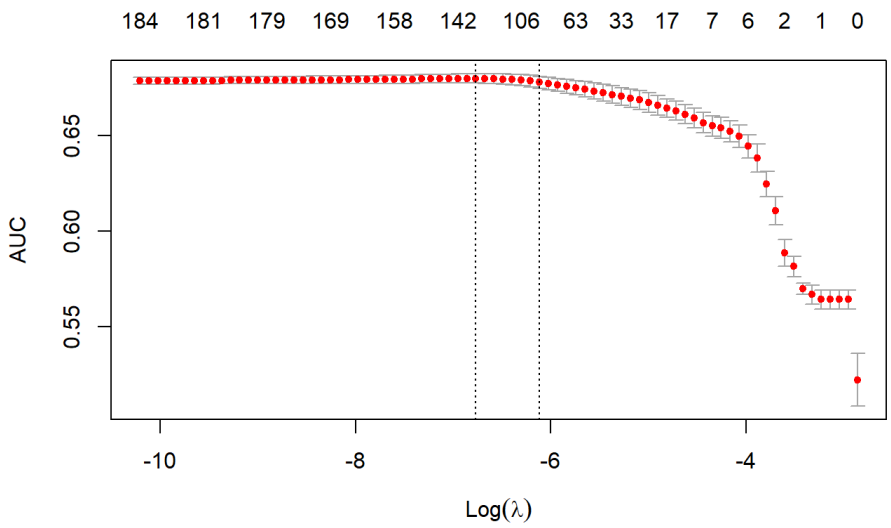
```
# Create a sparse matrix of the predictors (features) in the dataset
x_train <- sparse.model.matrix(formula, data = train_bin)[, -1]
x_test <- sparse.model.matrix(formula, data = test_bin)[, -1]
```

Now we apply the LASSO method (L1 regularization) to the training data and determine the optimal regularization parameter lambda using cross-validation. We use an alpha value of 1 for LASSO and perform 5-fold cross-validation.

```
lasso_model <- cv.glmnet(x = x_train,
                         y = y_train,
                         alpha = 1,
                         family = "binomial",
                         nfolds = 5,
                         type.measure = "auc")
```

Next, we visualize the regularization path based on the cross-validation results to see the relationship between lambda and model performance.

```
plot(lasso_model)
```



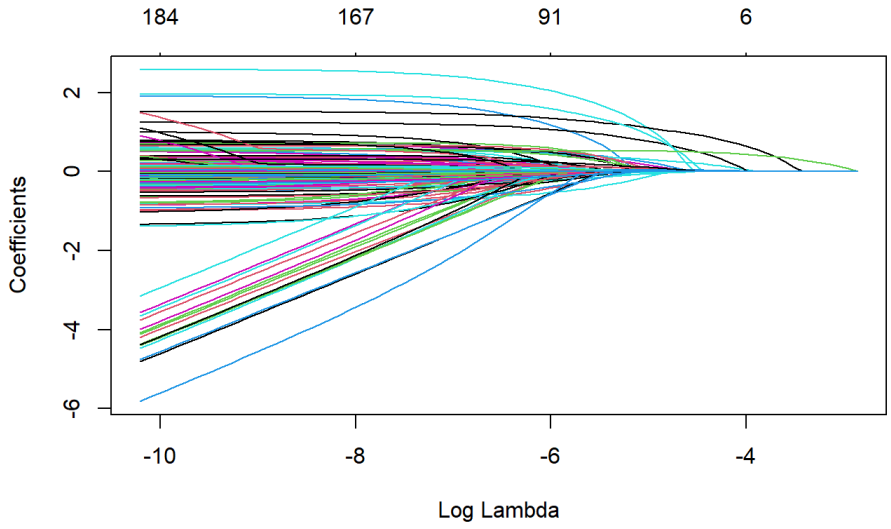
We now extract the model coefficients at the optimal lambda value (lambda.min) as determined by cross-validation. We display all coefficients that are different from zero.

```
# Extract coefficients at the optimal lambda (lambda.min)
lasso_coef <- coef(lasso_model, s = "lambda.min")
optimal_lambda <- lasso_model$lambda.min
cat("Optimal Lambda:", optimal_lambda)
```

```
## Optimal Lambda: 0.001151469
```

Optional but useful for understanding regularization: the following plot shows the coefficients as a function of $\log(\lambda)$.

```
plot(lasso_model$glmnet.fit, xvar ='lambda')
```



Finally, we compute, store, and print the non-zero coefficients.

```
# Convert to matrix and filter non-zero coefficients
lasso_coef_matrix <- as.matrix(lasso_coef)

# Find non-zero coefficients and their names
non_zero_indices <- which(lasso_coef_matrix != 0)
non_zero_coefs <- lasso_coef_matrix[non_zero_indices]
non_zero_names <- rownames(lasso_coef_matrix)[non_zero_indices]

# Create a dataframe with two columns: "feature" and "coefficient"
non_zero_df <- data.frame(feature = non_zero_names, coefficient = non_zero_coefs, stringsAsFactors = FALSE)
non_zero_df
```

feature <chr>	coefficient <dbl>
(Intercept)	-2.9005310000
raceAfricanAmerican	0.0006113609
raceCaucasian	0.0179412466

feature <chr>	coefficient <dbl>
raceHispanic	-0.2215149376
raceOther	-0.2649741035
age[10-20)	-0.0147021423
age[20-30)	-0.0268227997
age[30-40)	-0.1292924000
age[50-60)	-0.1287445415
age[60-70)	0.0574697270

1-10 of 136 rows Previous **1** 2 3 4 5 6 … 14 Next

Considering the readmission rates from subtask R2c), we can identify the following important relationships:

- `discharge_disposition_id` : A very high value for `discharge_disposition_id15` aligns with the high readmission rate observed in the EDA for this category.
- `number_inpatient` : A relatively high coefficient for `number_inpatient` is consistent with the pattern that a higher value of this feature tends to correlate with higher readmission rates.
- `gender` : This feature has a coefficient of zero in the LASSO model. This reflects the insight that readmission rates for men and women are nearly identical.

Now we compute the predictions from the LASSO model on the test dataset and evaluate its performance using the AUC score.

```
# Predict test set probabilities using the LASSO model
lasso_predictions <- predict(lasso_model, newx = x_test,
                             type = "response", s = "lambda.min")

# Compute AUC for the LASSO model
auc_lasso <- calculate_auc(true_labels = y_test,
                           predicted_probs = lasso_predictions)
```

```
## Setting levels: control = 0, case = 1
```

```
## Warning in roc.default(true_labels, predicted_probs): Deprecated use a matrix
## as predictor. Unexpected results may be produced, please pass a numeric vector.
```

```
## Setting direction: controls < cases
```

```
## AUC: 0.6884
```

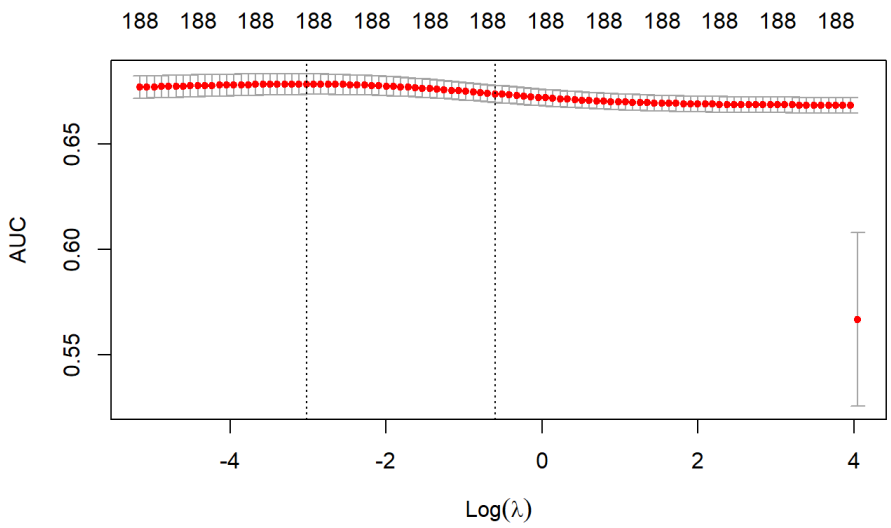
b) L2 Regularization (Ridge Regression): Apply the Ridge Regression method (L2 regularization) to the training data. Determine the optimal regularization parameter lambda using cross-validation and visualize the regularization path. Compute the AUC score of the model on the test data.

We use the training and test datasets prepared in subtask R6a) and apply Ridge Regression (L2 regularization) to the training data. We determine the optimal regularization parameter lambda using cross-validation (alpha = 0 for Ridge Regression).

```
ridge_model <- cv.glmnet(x = x_train,
                         y = y_train,
                         alpha = 0,
                         family = "binomial",
                         nfolds = 5,
                         type.measure = "auc")
```

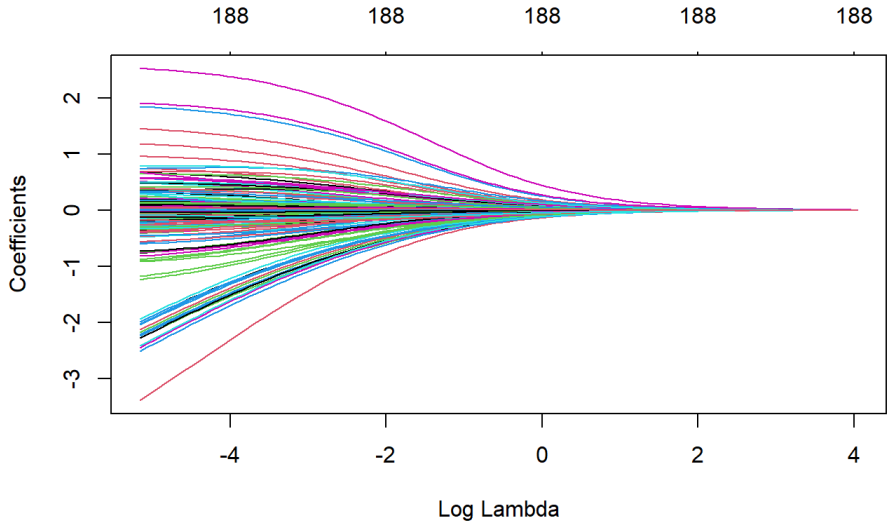
To visualize the model performance depending on different regularization parameters, we plot the cross-validation results of the Ridge Regression.

```
plot(ridge_model)
```



Not required, but helpful for understanding regularization: The following plot visualizes the coefficients as a function of $\log(\lambda)$.

```
plot(ridge_model$glmnet.fit, xvar ='lambda')
```



Now we extract the coefficients of the Ridge model for the optimal value of lambda (lambda.min), which was identified via cross-validation, and print the coefficients of the Ridge regression model.

```
ridge_coef <- coef(ridge_model, s = "lambda.min")
optimal_lambda <- ridge_model$lambda.min
cat("Optimal Lambda:\n", optimal_lambda, "\n")
```

```
## Optimal Lambda:
##  0.04869844
```

Finally, we compute the predictions of the Ridge regression model for the test dataset. We evaluate the model performance using the AUC score.

```
# Predict the test dataset using the Ridge regression model
ridge_predictions <- predict(ridge_model, newx = x_test,
                             type = "response", s = "lambda.min")

# Compute the AUC score for the Ridge regression
auc_ridge <- calculate_auc(true_labels = y_test,
                           predicted_probs = ridge_predictions)
```

```
## AUC: 0.6865
```

c) Generalized Additive Model (GAM): Train a Generalized Additive Model (GAM) on the training data, for example using the `mgcv` package. Design the GAM such that the AUC score achieved of this model on the test data is higher than that of the Logistic Regression model from Task R4a). Explain how the plots and curve patterns from Task R2 help identify relevant features for constructing a suitable GAM, taking into account effects that were overlooked in the previous task parts. Specify the corresponding feature candidates and visualize the estimated effects of two of these features, which have been identified as particularly promising for the GAM. Compute the AUC score of the model on the test data.

First, we identify the numeric predictors in the training data and create a formula where these predictors are wrapped in `s()` for smooth terms. This allows us to better model nonlinear relationships. Non-numeric predictors are included in the formula unchanged.

```r
# Identify numeric predictors (columns of type integer)
numeric_features <- colnames(train_bin)[sapply(train_bin, is.integer)]

# Identify non-numeric predictors (excluding the target variable TARGET)
non_numeric_features <- setdiff(valid_features, numeric_features)

# Create the formula for numeric predictors with smooth terms (s())
numeric_formula <- paste("s(", numeric_features, ", k=4)", collapse = " + ")

# Combine numeric and non-numeric predictors
formula_with_smooths <- as.formula(paste("TARGET ~", paste(c(numeric_formula, non_numeric_features), collapse = " + ")))

print(formula_with_smooths)
```

```
## TARGET ~ s(time_in_hospital, k = 4) + s(num_lab_procedures, k = 4) +
##     s(num_procedures, k = 4) + s(num_medications, k = 4) + s(number_outpatient,
##     k = 4) + s(number_emergency, k = 4) + s(number_inpatient,
##     k = 4) + s(number_diagnoses, k = 4) + race + gender + age +
##     weight + admission_type_id + discharge_disposition_id + admission_source_id +
##     payer_code + medical_specialty + max_glu_serum + A1Cresult +
##     metformin + repaglinide + nateglinide + chlorpropamide +
##     glimepiride + glipizide + glyburide + tolbutamide + pioglitazone +
##     rosiglitazone + acarbose + miglitol + tolazamide + insulin +
##     glyburide.metformin + change + diabetesMed + group_diag_1 +
##     group_diag_2 + group_diag_3
```

Now we train the Generalized Additive Model (GAM) on the training data. This model uses smooth terms for numeric predictors and accounts for both linear and nonlinear relationships.

```r
# Train a GAM model with smooth terms for nonlinear relationships
gam_model <- gam(
  formula_with_smooths,
  data = train_bin,
  family = binomial(link = "logit")
)

# Show the summary of the GAM model
summary(gam_model)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## TARGET ~ s(time_in_hospital, k = 4) + s(num_lab_procedures, k = 4) +
##     s(num_procedures, k = 4) + s(num_medications, k = 4) + s(number_outpatient,
##     k = 4) + s(number_emergency, k = 4) + s(number_inpatient,
##     k = 4) + s(number_diagnoses, k = 4) + race + gender + age +
##     weight + admission_type_id + discharge_disposition_id + admission_source_id +
##     payer_code + medical_specialty + max_glu_serum + A1Cresult +
##     metformin + repaglinide + nateglinide + chlorpropamide +
##     glimepiride + glipizide + glyburide + tolbutamide + pioglitazone +
##     rosiglitazone + acarbose + miglitol + tolazamide + insulin +
##     glyburide.metformin + change + diabetesMed + group_diag_1 +
##     group_diag_2 + group_diag_3
##
## Parametric coefficients:
##                                                  Estimate Std. Error
## (Intercept)                                    -2.235e+01  8.293e+03
## raceAfricanAmerican                             2.269e-01  1.190e-01
## raceAsian                                       2.159e-01  2.195e-01
## raceCaucasian                                   2.313e-01  1.128e-01
## raceHispanic                                   -1.184e-01  1.730e-01
## raceOther                                      -1.623e-01  1.861e-01
## genderMale                                     -7.776e-03  3.485e-02
## age[10-20)                                      4.353e-01  7.929e-01
## age[20-30)                                      3.969e-01  7.769e-01
## age[30-40)                                      3.001e-01  7.698e-01
## age[40-50)                                      4.915e-01  7.660e-01
## age[50-60)                                      3.521e-01  7.656e-01
## age[60-70)                                      5.837e-01  7.654e-01
## age[70-80)                                      6.853e-01  7.657e-01
## age[80-90)                                      5.792e-01  7.663e-01
## age[90-100)                                     3.417e-01  7.724e-01
## weight[0-25)                                    1.942e+00  6.310e-01
## weight[25-50)                                  -3.272e-02  4.721e-01
## weight[50-75)                                   2.154e-01  1.638e-01
## weight[75-100)                                  7.020e-01  1.283e-01
## weight[100-125)                                 7.778e-01  1.805e-01
## weight[125-150)                                 2.539e-01  4.159e-01
## weight[150-175)                                -3.869e+01  2.023e+07
## admission_type_id2                              5.833e-02  6.880e-02
## admission_type_id3                             -1.401e-01  7.982e-02
## admission_type_id5                              6.306e-02  1.197e-01
## admission_type_id6                              4.008e-01  9.820e-02
## admission_type_id7                             -3.873e+01  1.861e+07
## admission_type_id8                             -3.895e-01  3.045e-01
## discharge_disposition_id2                       5.932e-01  9.838e-02
## discharge_disposition_id3                       5.452e-01  5.539e-02
## discharge_disposition_id4                       3.453e-01  1.866e-01
## discharge_disposition_id5                       1.233e+00  1.120e-01
## discharge_disposition_id6                       2.690e-01  5.629e-02
## discharge_disposition_id7                       1.909e-01  2.254e-01
## discharge_disposition_id8                       4.964e-01  4.576e-01
## discharge_disposition_id15                      2.553e+00  5.688e-01
## discharge_disposition_id18                      2.092e-01  9.370e-02
## discharge_disposition_id22                      1.528e+00  8.913e-02
## discharge_disposition_id23                     -1.020e+00  4.234e-01
## discharge_disposition_id24                     -3.866e+01  1.937e+07
## discharge_disposition_id25                     -5.277e-01  2.122e-01
## discharge_disposition_id28                      1.981e+00  3.093e-01
## admission_source_id2                           -4.313e-01  1.576e-01
## admission_source_id3                            6.552e-01  2.880e-01
## admission_source_id4                           -4.555e-01  1.005e-01
## admission_source_id5                           -6.716e-01  2.018e-01
## admission_source_id6                           -2.146e-01  1.349e-01
## admission_source_id7                           -4.288e-02  7.034e-02
## admission_source_id8                            2.872e-01  1.136e+00
## admission_source_id9                            1.081e-01  4.529e-01
## admission_source_id17                          -4.938e-01  1.090e-01
## admission_source_id20                           6.932e-01  3.346e-01
## payer_codeBC                                   -4.519e-01  9.606e-02
## payer_codeCH                                   -8.764e-01  6.011e-01
## payer_codeCM                                   -5.371e-01  1.344e-01
## payer_codeCP                                   -3.900e-01  1.185e-01
## payer_codeDM                                   -3.997e-01  2.572e-01
## payer_codeHM                                   -2.334e-01  8.421e-02
## payer_codeMC                                   -2.708e-01  4.723e-02
## payer_codeMD                                   -1.512e-01  1.066e-01
## payer_codeMP                                   -2.982e-01  1.053e+00
## payer_codeOG                                   -1.481e-01  1.872e-01
## payer_codeOT                                   -8.269e-01  7.495e-01
## payer_codePO                                   -3.610e-01  2.363e-01
## payer_codeSI                                    2.201e-02  7.640e-01
```

```
## payer_codeSP                                          -1.353e-01  9.095e-02
## payer_codeUN                                          -3.011e-01  1.171e-01
## payer_codeWC                                          -9.977e-01  6.057e-01
## medical_specialtyAnesthesiology-Pediatric              3.313e-01  1.102e+00
## medical_specialtyCardiology                           -2.088e-01  8.385e-02
## medical_specialtyEmergency/Trauma                     -1.070e-01  8.715e-02
## medical_specialtyEndocrinology                        -1.229e-01  4.863e-01
## medical_specialtyFamily/GeneralPractice                6.150e-02  6.886e-02
## medical_specialtyGastroenterology                      3.981e-01  2.150e-01
## medical_specialtyGynecology                           -3.838e+01  1.151e+07
## medical_specialtyHematology                           -2.208e-01  7.884e-01
## medical_specialtyHematology/Oncology                   5.767e-01  3.585e-01
## medical_specialtyHospitalist                          -3.102e-01  1.059e+00
## medical_specialtyInfectiousDiseases                    9.753e-01  6.080e-01
## medical_specialtyInternalMedicine                     -8.709e-02  5.273e-02
## medical_specialtyNephrology                            5.573e-01  1.486e-01
## medical_specialtyNeurology                            -3.607e-01  3.865e-01
## medical_specialtyObsterics&Gynecology-GynecologicOnco -3.903e+01  1.937e+07
## medical_specialtyObstetrics                           -3.828e+01  2.373e+07
## medical_specialtyObstetricsandGynecology              -6.496e-01  2.546e-01
## medical_specialtyOncology                              6.326e-01  2.394e-01
## medical_specialtyOphthalmology                        -5.134e-01  1.043e+00
## medical_specialtyOrthopedics                          -6.811e-02  1.438e-01
## medical_specialtyOrthopedics-Reconstructive           -8.619e-01  1.821e-01
## medical_specialtyOsteopath                             4.473e-01  6.771e-01
## medical_specialtyOtolaryngology                       -3.853e+01  8.455e+06
## medical_specialtyPediatrics                           -7.671e-01  4.380e-01
## medical_specialtyPediatrics-CriticalCare              -3.828e+01  1.118e+07
## medical_specialtyPediatrics-Endocrinology             -3.809e+01  7.074e+06
## medical_specialtyPhysicalMedicineandRehabilitation     5.850e-01  2.592e-01
## medical_specialtyPodiatry                              4.608e-04  6.387e-01
## medical_specialtyPsychiatry                            4.203e-01  1.687e-01
## medical_specialtyPsychology                           -1.029e+00  1.027e+00
## medical_specialtyPulmonology                           6.231e-02  1.671e-01
## medical_specialtyRadiologist                          -3.144e-01  1.835e-01
## medical_specialtyRadiology                             1.091e-01  6.445e-01
## medical_specialtyRheumatology                         -3.866e+01  3.355e+07
## medical_specialtySurgeon                              -4.416e-01  7.634e-01
## medical_specialtySurgery-Cardiovascular               -1.314e+00  7.324e-01
## medical_specialtySurgery-Cardiovascular/Thoracic      -6.553e-01  2.409e-01
## medical_specialtySurgery-General                      -1.380e-01  1.031e-01
## medical_specialtySurgery-Neuro                        -9.678e-01  3.100e-01
## medical_specialtySurgery-Plastic                       5.542e-02  8.012e-01
## medical_specialtySurgery-Thoracic                     -3.943e-01  5.397e-01
## medical_specialtySurgery-Vascular                      3.899e-01  2.131e-01
## medical_specialtySurgicalSpecialty                    -3.828e+01  1.937e+07
## medical_specialtyUrology                              -1.672e-02  2.017e-01
## max_glu_serum>300                                      1.491e-03  2.207e-01
## max_glu_serumNone                                     -8.366e-02  1.623e-01
## max_glu_serumNorm                                      1.616e-01  1.803e-01
## A1Cresult>8                                           -1.135e-02  1.039e-01
## A1CresultNone                                          7.273e-02  8.732e-02
## A1CresultNorm                                         -3.785e-03  1.111e-01
## metforminNo                                            1.538e-02  2.050e-01
## metforminSteady                                       -1.126e-01  2.049e-01
## metforminUp                                           -3.557e-01  2.665e-01
## repaglinideNo                                          1.940e+01  8.293e+03
## repaglinideSteady                                      1.956e+01  8.293e+03
## repaglinideUp                                          1.998e+01  8.293e+03
## nateglinideSteady                                     -8.436e-02  2.198e-01
## nateglinideUp                                         -3.985e+01  2.122e+07
## chlorpropamideSteady                                  -1.406e+00  1.038e+00
## glimepirideNo                                         -1.518e-01  3.595e-01
## glimepirideSteady                                     -2.581e-01  3.653e-01
## glimepirideUp                                         -1.732e-01  4.504e-01
## glipizideNo                                           -3.362e-01  2.114e-01
## glipizideSteady                                       -3.211e-01  2.120e-01
## glipizideUp                                           -1.669e-01  2.661e-01
## glyburideNo                                            1.250e-01  2.198e-01
## glyburideSteady                                        1.318e-01  2.199e-01
## glyburideUp                                           -3.865e-02  2.780e-01
## tolbutamideSteady                                     -7.555e-01  1.060e+00
## pioglitazoneNo                                        -4.713e-02  4.670e-01
## pioglitazoneSteady                                    -1.401e-01  4.702e-01
## pioglitazoneUp                                         2.362e-01  5.520e-01
## rosiglitazoneNo                                        7.068e-01  6.237e-01
## rosiglitazoneSteady                                    7.478e-01  6.266e-01
## rosiglitazoneUp                                        7.525e-01  7.172e-01
## acarboseSteady                                        -5.480e-02  3.515e-01
## acarboseUp                                            -3.859e+01  4.745e+07
## miglitolSteady                                        -3.906e+01  3.355e+07
## tolazamideSteady                                      -3.891e+01  2.023e+07
## insulinNo                                             -1.447e-01  9.259e-02
## insulinSteady                                         -1.453e-01  7.079e-02
## insulinUp                                             -7.728e-02  7.268e-02
## glyburide.metforminSteady                              8.694e-02  2.080e-01
```

```
## changeNo                                  4.757e-02  6.466e-02
## diabetesMedYes                            3.029e-01  6.301e-02
## group_diag_1Diabetes                      2.389e-02  7.463e-02
## group_diag_1Digestive                    -1.652e-01  7.067e-02
## group_diag_1Genitourinary                -1.291e-01  8.578e-02
## group_diag_1Injury                       -2.270e-01  7.468e-02
## group_diag_1Musculoskeletal              -2.652e-01  9.593e-02
## group_diag_1n.a.                          6.987e-01  8.955e-01
## group_diag_1Neoplasms                     2.282e-02  8.728e-02
## group_diag_1Other                        -2.216e-01  5.769e-02
## group_diag_1Respiratory                  -3.050e-01  6.188e-02
## group_diag_2Diabetes                      1.550e-01  6.060e-02
## group_diag_2Digestive                    -1.360e-01  1.014e-01
## group_diag_2Genitourinary                -1.304e-01  7.060e-02
## group_diag_2Injury                       -1.273e-01  1.110e-01
## group_diag_2Musculoskeletal              -1.472e-01  1.362e-01
## group_diag_2n.a.                         -9.619e-02  4.015e-01
## group_diag_2Neoplasms                     1.373e-01  9.597e-02
## group_diag_2Other                         8.806e-03  4.847e-02
## group_diag_2Respiratory                  -5.022e-02  6.473e-02
## group_diag_3Diabetes                      8.096e-02  5.315e-02
## group_diag_3Digestive                     1.166e-01  9.691e-02
## group_diag_3Genitourinary                 5.116e-02  7.535e-02
## group_diag_3Injury                       -3.486e-02  1.236e-01
## group_diag_3Musculoskeletal               2.374e-02  1.243e-01
## group_diag_3n.a.                         -1.132e-02  1.843e-01
## group_diag_3Neoplasms                     4.385e-02  9.967e-02
## group_diag_3Other                        -2.900e-02  4.681e-02
## group_diag_3Respiratory                   3.874e-02  7.223e-02
##                                          z value Pr(>|z|)
## (Intercept)                               -0.003 0.997849
## raceAfricanAmerican                        1.907 0.056465 .
## raceAsian                                  0.984 0.325291
## raceCaucasian                              2.050 0.040355 *
## raceHispanic                              -0.684 0.493682
## raceOther                                 -0.872 0.383095
## genderMale                                -0.223 0.823415
## age[10-20)                                 0.549 0.583011
## age[20-30)                                 0.511 0.609494
## age[30-40)                                 0.390 0.696630
## age[40-50)                                 0.642 0.521158
## age[50-60)                                 0.460 0.645628
## age[60-70)                                 0.763 0.445743
## age[70-80)                                 0.895 0.370773
## age[80-90)                                 0.756 0.449749
## age[90-100)                                0.442 0.658208
## weight[0-25)                               3.078 0.002083 **
## weight[25-50)                             -0.069 0.944754
## weight[50-75)                              1.315 0.188385
## weight[75-100)                             5.471 4.48e-08 ***
## weight[100-125)                            4.310 1.63e-05 ***
## weight[125-150)                            0.610 0.541555
## weight[150-175)                            0.000 0.999998
## admission_type_id2                         0.848 0.396501
## admission_type_id3                        -1.755 0.079258 .
## admission_type_id5                         0.527 0.598232
## admission_type_id6                         4.081 4.48e-05 ***
## admission_type_id7                         0.000 0.999998
## admission_type_id8                        -1.279 0.200901
## discharge_disposition_id2                  6.030 1.64e-09 ***
## discharge_disposition_id3                  9.842  < 2e-16 ***
## discharge_disposition_id4                  1.850 0.064251 .
## discharge_disposition_id5                 11.004  < 2e-16 ***
## discharge_disposition_id6                  4.779 1.76e-06 ***
## discharge_disposition_id7                  0.847 0.397217
## discharge_disposition_id8                  1.085 0.277945
## discharge_disposition_id15                 4.489 7.15e-06 ***
## discharge_disposition_id18                 2.233 0.025539 *
## discharge_disposition_id22                17.141  < 2e-16 ***
## discharge_disposition_id23                -2.410 0.015971 *
## discharge_disposition_id24                 0.000 0.999998
## discharge_disposition_id25                -2.486 0.012901 *
## discharge_disposition_id28                 6.404 1.52e-10 ***
## admission_source_id2                      -2.737 0.006200 **
## admission_source_id3                       2.275 0.022883 *
## admission_source_id4                      -4.534 5.79e-06 ***
## admission_source_id5                      -3.329 0.000872 ***
## admission_source_id6                      -1.591 0.111555
## admission_source_id7                      -0.610 0.542171
## admission_source_id8                       0.253 0.800373
## admission_source_id9                       0.239 0.811399
## admission_source_id17                     -4.531 5.87e-06 ***
## admission_source_id20                      2.072 0.038278 *
## payer_codeBC                              -4.705 2.54e-06 ***
## payer_codeCH                              -1.458 0.144895
## payer_codeCM                              -3.996 6.44e-05 ***
```

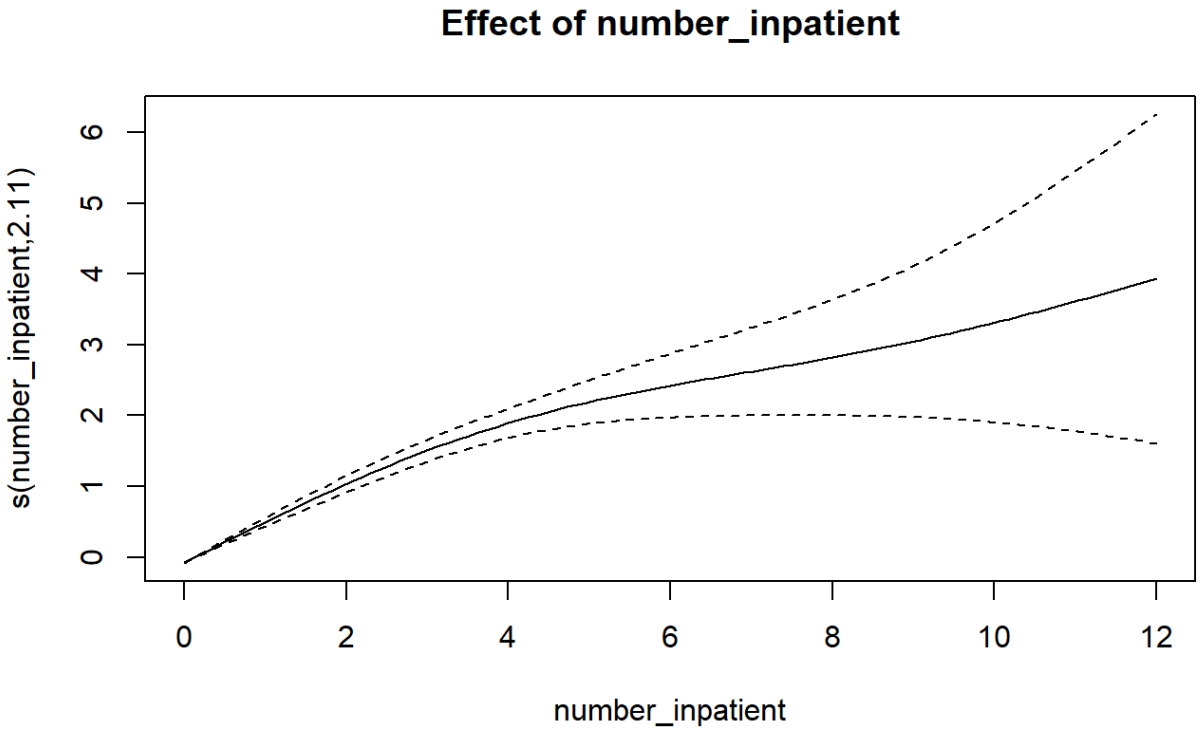
```
## payer_codeCP                                      -3.291 0.001000 ***
## payer_codeDM                                      -1.554 0.120209
## payer_codeHM                                      -2.772 0.005570 **
## payer_codeMC                                      -5.734 9.83e-09 ***
## payer_codeMD                                      -1.418 0.156107
## payer_codeMP                                      -0.283 0.776948
## payer_codeOG                                      -0.791 0.429053
## payer_codeOT                                      -1.103 0.269935
## payer_codePO                                      -1.528 0.126570
## payer_codeSI                                       0.029 0.977015
## payer_codeSP                                      -1.488 0.136755
## payer_codeUN                                      -2.572 0.010119 *
## payer_codeWC                                      -1.647 0.099533 .
## medical_specialtyAnesthesiology-Pediatric          0.301 0.763703
## medical_specialtyCardiology                       -2.490 0.012770 *
## medical_specialtyEmergency/Trauma                 -1.228 0.219489
## medical_specialtyEndocrinology                    -0.253 0.800556
## medical_specialtyFamily/GeneralPractice            0.893 0.371826
## medical_specialtyGastroenterology                  1.851 0.064105 .
## medical_specialtyGynecology                        0.000 0.999997
## medical_specialtyHematology                       -0.280 0.779452
## medical_specialtyHematology/Oncology               1.609 0.107667
## medical_specialtyHospitalist                      -0.293 0.769683
## medical_specialtyInfectiousDiseases                1.604 0.108649
## medical_specialtyInternalMedicine                 -1.652 0.098579 .
## medical_specialtyNephrology                        3.751 0.000176 ***
## medical_specialtyNeurology                        -0.933 0.350693
## medical_specialtyObsterics&Gynecology-GynecologicOnco  0.000 0.999998
## medical_specialtyObstetrics                        0.000 0.999999
## medical_specialtyObstetricsandGynecology          -2.552 0.010724 *
## medical_specialtyOncology                          2.643 0.008214 **
## medical_specialtyOphthalmology                    -0.492 0.622465
## medical_specialtyOrthopedics                      -0.474 0.635797
## medical_specialtyOrthopedics-Reconstructive       -4.734 2.21e-06 ***
## medical_specialtyOsteopath                         0.661 0.508843
## medical_specialtyOtolaryngology                    0.000 0.999996
## medical_specialtyPediatrics                       -1.751 0.079906 .
## medical_specialtyPediatrics-CriticalCare           0.000 0.999997
## medical_specialtyPediatrics-Endocrinology          0.000 0.999996
## medical_specialtyPhysicalMedicineandRehabilitation  2.257 0.024015 *
## medical_specialtyPodiatry                          0.001 0.999424
## medical_specialtyPsychiatry                        2.491 0.012726 *
## medical_specialtyPsychology                       -1.002 0.316347
## medical_specialtyPulmonology                       0.373 0.709225
## medical_specialtyRadiologist                      -1.713 0.086657 .
## medical_specialtyRadiology                         0.169 0.865626
## medical_specialtyRheumatology                      0.000 0.999999
## medical_specialtySurgeon                          -0.578 0.562959
## medical_specialtySurgery-Cardiovascular           -1.794 0.072772 .
## medical_specialtySurgery-Cardiovascular/Thoracic  -2.720 0.006528 **
## medical_specialtySurgery-General                  -1.339 0.180488
## medical_specialtySurgery-Neuro                    -3.122 0.001796 **
## medical_specialtySurgery-Plastic                   0.069 0.944853
## medical_specialtySurgery-Thoracic                 -0.731 0.464962
## medical_specialtySurgery-Vascular                  1.829 0.067349 .
## medical_specialtySurgicalSpecialty                 0.000 0.999998
## medical_specialtyUrology                          -0.083 0.933963
## max_glu_serum>300                                  0.007 0.994611
## max_glu_serumNone                                 -0.515 0.606240
## max_glu_serumNorm                                  0.896 0.370211
## A1Cresult>8                                       -0.109 0.912994
## A1CresultNone                                      0.833 0.404926
## A1CresultNorm                                     -0.034 0.972814
## metforminNo                                        0.075 0.940203
## metforminSteady                                   -0.549 0.582743
## metforminUp                                       -1.335 0.182033
## repaglinideNo                                      0.002 0.998134
## repaglinideSteady                                  0.002 0.998118
## repaglinideUp                                      0.002 0.998078
## nateglinideSteady                                 -0.384 0.701184
## nateglinideUp                                      0.000 0.999999
## chlorpropamideSteady                              -1.354 0.175714
## glimepirideNo                                     -0.422 0.672830
## glimepirideSteady                                 -0.707 0.479767
## glimepirideUp                                     -0.384 0.700614
## glipizideNo                                       -1.590 0.111730
## glipizideSteady                                   -1.514 0.129974
## glipizideUp                                       -0.627 0.530451
## glyburideNo                                        0.569 0.569541
## glyburideSteady                                    0.600 0.548776
## glyburideUp                                       -0.139 0.889439
## tolbutamideSteady                                 -0.713 0.476002
## pioglitazoneNo                                    -0.101 0.919629
## pioglitazoneSteady                                -0.298 0.765742
## pioglitazoneUp                                     0.428 0.668800
## rosiglitazoneNo                                    1.133 0.257164
```

```
## rosiglitazoneSteady                                  1.193 0.232708
## rosiglitazoneUp                                      1.049 0.294047
## acarboseSteady                                      -0.156 0.876113
## acarboseUp                                           0.000 0.999999
## miglitolSteady                                       0.000 0.999999
## tolazamideSteady                                     0.000 0.999998
## insulinNo                                           -1.562 0.118199
## insulinSteady                                       -2.053 0.040061 *
## insulinUp                                           -1.063 0.287670
## glyburide.metforminSteady                            0.418 0.676036
## changeNo                                             0.736 0.461896
## diabetesMedYes                                       4.808 1.53e-06 ***
## group_diag_1Diabetes                                 0.320 0.748874
## group_diag_1Digestive                               -2.338 0.019388 *
## group_diag_1Genitourinary                           -1.505 0.132241
## group_diag_1Injury                                  -3.040 0.002367 **
## group_diag_1Musculoskeletal                         -2.764 0.005703 **
## group_diag_1n.a.                                     0.780 0.435261
## group_diag_1Neoplasms                                0.261 0.793751
## group_diag_1Other                                   -3.842 0.000122 ***
## group_diag_1Respiratory                             -4.929 8.28e-07 ***
## group_diag_2Diabetes                                 2.558 0.010540 *
## group_diag_2Digestive                               -1.341 0.179818
## group_diag_2Genitourinary                           -1.847 0.064715 .
## group_diag_2Injury                                  -1.147 0.251373
## group_diag_2Musculoskeletal                         -1.081 0.279760
## group_diag_2n.a.                                    -0.240 0.810628
## group_diag_2Neoplasms                                1.431 0.152475
## group_diag_2Other                                    0.182 0.855845
## group_diag_2Respiratory                             -0.776 0.437824
## group_diag_3Diabetes                                 1.523 0.127723
## group_diag_3Digestive                                1.204 0.228723
## group_diag_3Genitourinary                            0.679 0.497216
## group_diag_3Injury                                  -0.282 0.777827
## group_diag_3Musculoskeletal                          0.191 0.848546
## group_diag_3n.a.                                    -0.061 0.950999
## group_diag_3Neoplasms                                0.440 0.659958
## group_diag_3Other                                   -0.619 0.535594
## group_diag_3Respiratory                              0.536 0.591731
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                          edf Ref.df  Chi.sq  p-value
## s(time_in_hospital)   2.853  2.983  14.884  0.00138 **
## s(num_lab_procedures) 1.938  2.285   4.715  0.10270
## s(num_procedures)     1.006  1.011   0.639  0.42878
## s(num_medications)    2.855  2.982  23.526 7.94e-05 ***
## s(number_outpatient)  1.807  2.140   2.910  0.28190
## s(number_emergency)   2.039  2.226  62.178  < 2e-16 ***
## s(number_inpatient)   2.112  2.426 394.325  < 2e-16 ***
## s(number_diagnoses)   1.001  1.002  20.792 5.95e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.071   Deviance explained = 8.46%
## UBRE = -0.27579  Scale est. = 1         n = 33426
```

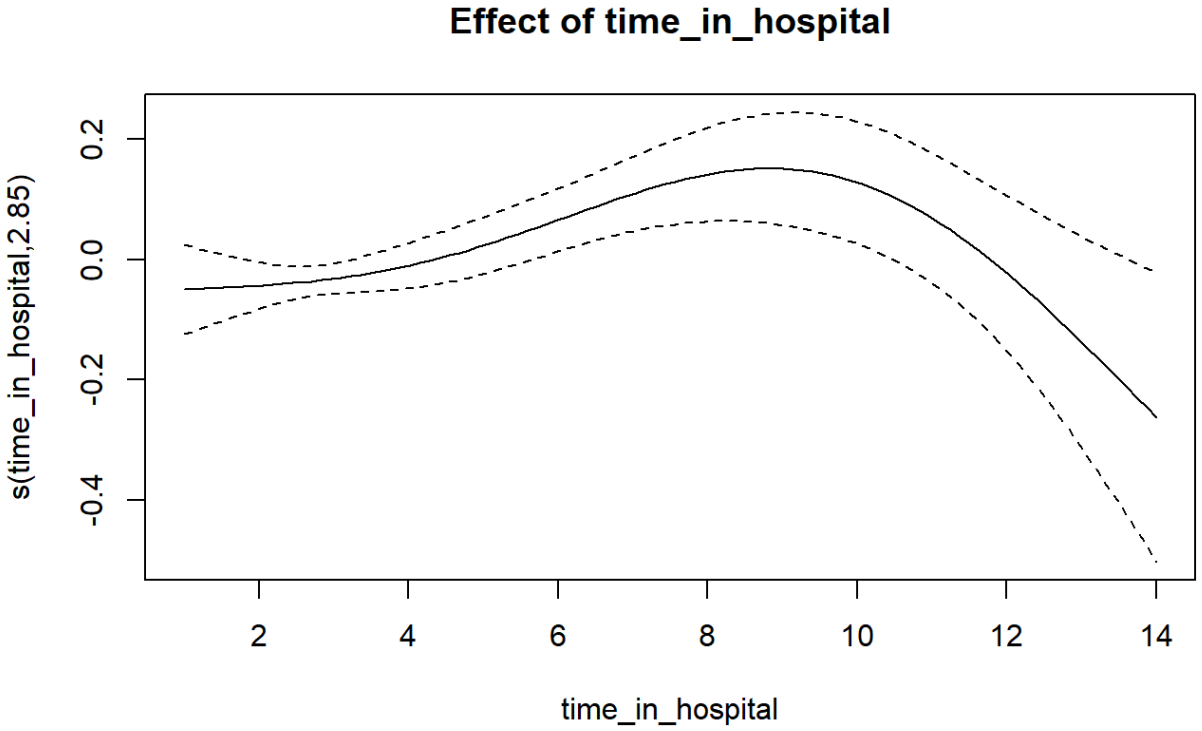
In particular, the event rate plots from Task R2c) show that the features `number_inpatient` and `time_in_hospital` are well-suited for use in a GAM due to their nonlinear patterns. Both features initially increase linearly before reaching a plateau. Another promising feature is `num_procedures`, which exhibited a wave-like shape in the Task R2c) plot. These features can capture additional nonlinear effects in a GAM, potentially improving model performance.

Next, we visualize the estimated effects of the features `number_inpatient`, `time_in_hospital`, and `num_procedures`.

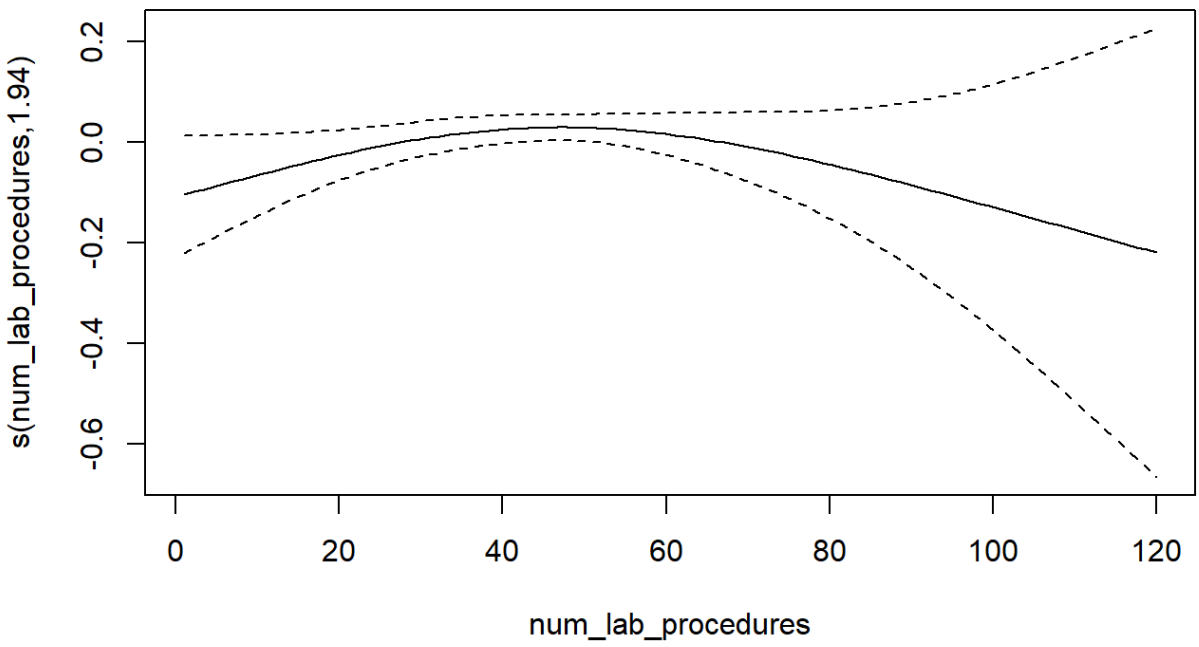
```
# Visualize the estimated effect for 'number_inpatient'
plot(gam_model, select = 7, scale  = 0, main = "Effect of number_inpatient")
```




```
# Visualize the estimated effect for 'time_in_hospital'
plot(gam_model, select = 1, scale  = 0, main = "Effect of time_in_hospital")
```




```
# Visualize the estimated effect for 'num_procedures'
plot(gam_model, select = 2, scale  = 0, main = "Effect of num_procedures")
```



To evaluate the model performance, we compute the predictions of the GAM model on the test dataset and calculate the AUC score.

```
# Predict the test dataset using the GAM model
gam_predictions <- predict(gam_model, newdata = test_bin, type = "response")

# Compute the AUC score for the GAM model
auc_gam <- calculate_auc(true_labels = y_test,
                         predicted_probs = gam_predictions,
                         verbose = TRUE)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
## AUC: 0.6896
```

d) Comparison and Discussion: Compare the model performance of the following approaches:

- **Logistic Regression without interactions from subtask R4a)**
- **Logistic Regression with interactions from subtask R4b)**
- **LASSO model from subtask R5a)**
- **Ridge Regression from subtask R5b)**
- **Generalized Additive Model from subtask R5c)**

Discuss which of these models is best suited for the given prediction task and provide a justified explanation for your choice.

To evaluate the models, we look at the AUC scores of their predictions on the test data. AUC provides insight into each model's ability to distinguish between classes. We summarize the AUC scores of all models in the table below:

```
# Compare AUC values
model_comparison <- data.frame(
  Model = c("Log. Reg. without Interactions", "Log. Reg. with Interactions", "LASSO", "Ridge", "GAM"),
  AUC = round(c(auc_logreg_without$auc, auc_logreg_with$auc,
                auc_lasso$auc, auc_ridge$auc, auc_gam$auc), 4)
)
model_comparison
```

Model <chr>	AUC <dbl>
Log. Reg. without Interactions	0.6887
Log. Reg. with Interactions	0.6896
LASSO	0.6884
Ridge	0.6865
GAM	0.6896

5 rows

All five models show similar performance in terms of AUC, with the GAM (0.6896) and Logistic Regression with interactions (0.6896) achieving the highest scores.

However, the LASSO model offers a significant advantage in interpretability. By performing inherent feature selection, it eliminates less important variables by setting their coefficients to zero. This makes it easier to identify key predictors, which is especially valuable when model transparency and interpretability are priorities.

Given its better interpretability and shorter runtime, and only slightly lower AUC, we consider LASSO to be the most suitable model for this prediction task among the ones evaluated.

Task R6: Ternary Classification

In this task, you will apply the machine learning models Multinomial Logistic Regression and XGBoost for ternary classification. The training, validation, and test datasets based on `diabetic_data_ter` , prepared in subtask R3a), will serve as input data.

a) Evaluation Metrics for Binary and Ternary Classification: First, explain whether and why the AUC evaluation metric used so far is well suited for binary classification. Briefly discuss two advantages and disadvantages of AUC compared to Accuracy. Describe the key considerations when evaluating multiclass classification models. Discuss the evaluation metrics Accuracy, Balanced Accuracy, and Macro-F1 Score, and explain in which situation each of these is particularly useful. Implement a function that computes these three metrics and outputs them in a concise format.

AUC for Binary Classification:

The evaluation metric AUC (Area Under the ROC Curve) is a very suitable metric for binary classification as it assesses model performance across all possible classification thresholds. AUC measures the model's ability to distinguish between the two classes, considering both false positive and false negative rates, thereby providing a comprehensive assessment of model quality.

Advantages and Disadvantages Compared to Accuracy:

- *Advantages:*
    1. **Sensitive to classification errors**: AUC is particularly useful for imbalanced datasets because it captures the model's performance on minority classes well.
    2. **Threshold-independent**: AUC evaluates performance across all possible decision thresholds, making it more flexible and not reliant on a fixed threshold.
- *Disadvantages:*
    1. **No direct link to confusion matrix**: AUC does not reflect exact classification counts and can be harder to interpret when detailed error analysis is needed.
    2. **Less intuitive**: The AUC value does not directly translate to an error rate (like Accuracy), making it harder to interpret for non-technical audiences.

Special Considerations for Multiclass Classification:

Standard binary metrics like AUC are not directly applicable in multiclass classification, as AUC only evaluates the ability to distinguish between two classes. For multiclass problems, evaluation metrics must account for all class predictions. Accuracy, Balanced Accuracy, and Macro-F1 Score are commonly used metrics for this purpose, especially with imbalanced or complex datasets.

- *Accuracy*: Measures the proportion of correctly classified instances. Easy to compute but can be misleading for imbalanced classes.
- *Balanced Accuracy*: Computes accuracy per class and averages them, correcting for class imbalance. Useful when class distribution is skewed.
- *Macro-F1 Score*: Computes the F1 Score for each class and averages them. Useful when all classes are equally important, ensuring the model performs well across all classes.

Use Cases:

- *Accuracy* is suitable when data is well-balanced.
- *Balanced Accuracy* is useful in imbalanced class scenarios.
- *Macro-F1 Score* is appropriate when each class is equally important and performance should be balanced across them.

Function to Compute Metrics:

```r
# Function to calculate model evaluation metrics
calculate_ternary_metrics <- function(y_true, y_pred) {
  # Accuracy
  accuracy <- sum(y_true == y_pred) / length(y_true)

  # Balanced Accuracy
  balanced_accuracy <- mean(sapply(unique(y_true), function(class) {
    tp <- sum(y_true == class & y_pred == class)
    fn <- sum(y_true == class & y_pred != class)
    tp / (tp + fn)
  }))

  # Macro-F1 Score
  f1_score <- mean(sapply(unique(y_true), function(class) {
    tp <- sum(y_true == class & y_pred == class)
    fp <- sum(y_true != class & y_pred == class)
    fn <- sum(y_true == class & y_pred != class)
    precision <- tp / (tp + fp)
    recall <- tp / (tp + fn)
    if (is.nan(precision) | is.nan(recall)) {
      return(0)
    }
    (2 * precision * recall) / (precision + recall)
  }))

  # Return metrics
  return(list(
    Accuracy = accuracy,
    Balanced_Accuracy = balanced_accuracy,
    Macro_F1_Score = f1_score
  ))
}
```

b) Ternary Classification with Multinomial Logistic Regression: Train a Multinomial Logistic Regression model (e.g., using the `multinom` function from the `nnet` package) on the training dataset. Select only features that have at least two distinct values, and exclude the target variable as well as the diagnosis columns (`diag_1` , `diag_2` , `diag_3`). Optimize the class thresholds to ensure that the predicted class distribution closely matches the actual distribution of `TARGET` . Evaluate the model's performance on the test dataset using the optimized thresholds and the function from subtask R6a). Additionally, generate and interpret an appropriate confusion matrix that compares the model's predictions with the actual values in the test data.

We model the ternary target variable `TARGET` using multinomial logistic regression via the `multinom` function from the `nnet` package. Before training, we exclude features with only one unique value, the target variable, and irrelevant diagnosis columns.

```r
# Filter only features with more than one unique value
valid_features_ter <- colnames(train_ter)[sapply(train_ter, function(col) length(unique(col)) > 1)]

# Exclude target variable and diagnosis columns
valid_features_ter <- setdiff(valid_features, c("TARGET", "diag_1", "diag_2", "diag_3"))

# Create formula using valid features
formula_ter <- as.formula(paste("TARGET ~", paste(valid_features, collapse = " + ")))

# Train the Multinomial Logistic Regression model
multinom_model <- nnet::multinom(formula, data = train_ter, MaxNWts = 1000000)
```

```
## # weights:  570 (378 variable)
## initial  value 53812.227124
## iter  10 value 44837.777856
## iter  20 value 43989.214411
## iter  30 value 43740.508663
## iter  40 value 43341.673384
## iter  50 value 42798.357373
## iter  60 value 42224.819112
## iter  70 value 41831.874787
## iter  80 value 41525.611919
## iter  90 value 41384.496020
## iter 100 value 41316.363484
## final  value 41316.363484
## stopped after 100 iterations
```

To generate class predictions, we define thresholds that reflect the actual class distribution in the test data. We use the `calculate_ternary_metrics` function from subtask R6a) to assess model quality in terms of Accuracy, Balanced Accuracy, and Macro-F1 Score.

We first set initial thresholds and then optimize them to align the predicted class distribution with the true distribution. After optimization, we make predictions and evaluate the model.

```
# Threshold optimization function
optimize_thresholds <- function(probabilities, actual_values, initial_thresholds) {
  # Target: predicted class distribution should resemble the actual one
  target_distribution <- table(factor(actual_values, levels = c("<30", ">30", "NO"))) / length(actual_values)

  # Loss function to minimize the distribution mismatch
  loss_function <- function(thresholds) {
    predicted_classes <- apply(probabilities, 1, function(row) {
      if (row[1] > thresholds[1]) return("<30")
      else if (row[2] > thresholds[2]) return(">30")
      else if (row[3] > thresholds[3]) return("NO")
      else return(c("<30", ">30", "NO")[which.max(row)])
    })
    predicted_distribution <- table(factor(predicted_classes, levels = c("<30", ">30", "NO"))) / length(predicted_classes)
    sum(abs(predicted_distribution - target_distribution))
  }

  # Optimize thresholds
  optimal_thresholds <- optim(
    par = initial_thresholds,
    fn = loss_function,
    method = "L-BFGS-B",
    lower = c(0, 0, 0),
    upper = c(1, 1, 1)
  )$par

  return(optimal_thresholds)
}

# Predict probabilities for test set
probabilities_multinom <- predict(multinom_model, newdata = test_ter, type = "probs")

# Initialize thresholds
initial_thresholds <- c(0.1, 0.3, 0.5)

# Optimize thresholds
optimized_thresholds <- optimize_thresholds(probabilities_multinom, test_ter$TARGET, initial_thresholds)

# Show optimized thresholds
cat("Optimized thresholds:\n", optimized_thresholds, "\n")
```

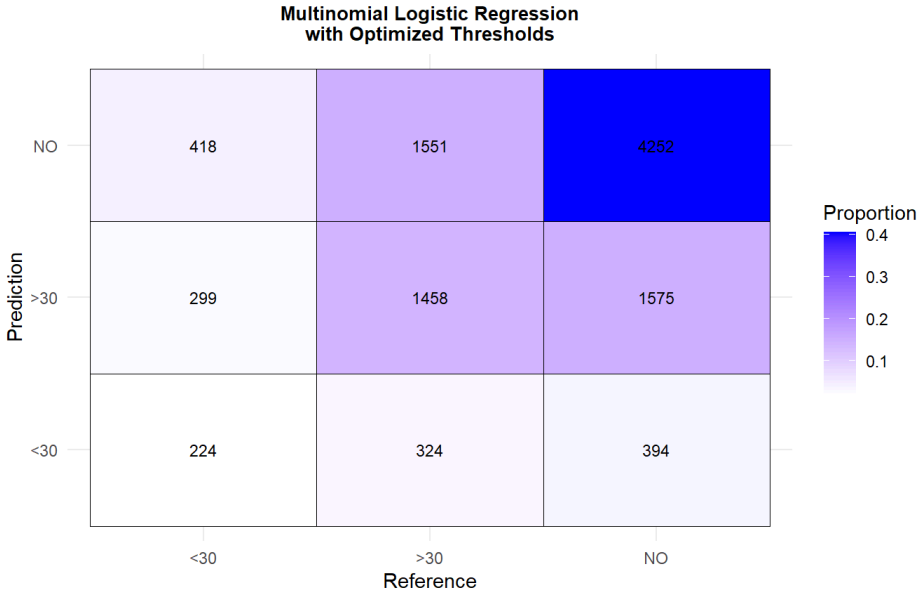
```
## Optimized thresholds:
##  0.1670346 0.3523208 0.5
```

```
# Predict classes using optimized thresholds
predictions_optimized <- apply(probabilities_multinom, 1, function(row) {
  if (row[1] > optimized_thresholds[1]) return("<30")
  else if (row[2] > optimized_thresholds[2]) return(">30")
  else if (row[3] > optimized_thresholds[3]) return("NO")
  else return(c("<30", ">30", "NO")[which.max(row)])
})

# Convert true values and predictions to factors
y_test_factor <- factor(test_ter$TARGET, levels = c("<30", ">30", "NO"))
predictions_factor <- factor(predictions_optimized, levels = c("<30", ">30", "NO"))

# Compute metrics
metrics_multinom_optimized <- calculate_ternary_metrics(y_test_factor, predictions_factor)

# Output results
cat("Model performance with optimized thresholds:\n",
    "Accuracy: ", metrics_multinom_optimized$Accuracy * 100, "%\n",
    "Balanced Accuracy: ", metrics_multinom_optimized$Balanced_Accuracy * 100, "%\n",
    "Macro-F1-Score: ", metrics_multinom_optimized$Macro_F1_Score * 100, "%\n")
```

```
## Model performance with optimized thresholds:
##  Accuracy:  56.54121 %
##  Balanced Accuracy:  45.29933 %
##  Macro-F1-Score:  45.2973 %
```

Finally, we create and visualize the confusion matrix using the optimized predictions.

```
# Create confusion matrix
confusion_matrix_multinom <- confusionMatrix(predictions_factor, y_test_factor)

# Visualize the confusion matrix
cm_multinom <- plot_confusion_matrix(confusion_matrix_multinom, "Multinomial Logistic Regression\nwith Optimized Threshold
        s")
cm_multinom
```



c) Ternary Classification with XGBoost: Train an XGBoost model, analogous to subtask R3b). Optimize the class thresholds to ensure that the predicted class distribution closely matches the actual distribution of `TARGET`. Evaluate the model's performance on the test dataset using the optimized thresholds and the function from subtask R6a). Additionally, generate and interpret an appropriate confusion matrix that compares the model's predictions with the actual values in the test data. Compare the results with those from subtask R6b) and discuss which model (Multinomial Logistic Regression or XGBoost) is better suited for this problem, based on model performance and the confusion matrix.

We train an XGBoost model for ternary classification to evaluate model performance and generate a confusion matrix. We use the same error metrics as in subtask R6a). Finally, we visualize and interpret the confusion matrix to compare the model's predictions to the actual test values.

We first convert the training and test datasets to DMatrix format for efficient computation using XGBoost, ensuring that the target variable is formatted correctly for ternary classification.

```
# Prepare training and test data
X_train <- train_ter %>% select(-TARGET)
X_val <- val_ter %>% select(-TARGET)
X_test <- test_ter %>% select(-TARGET)

# Convert the target variable to a factor
y_train <- train_ter$TARGET
y_val <- val_ter$TARGET
y_test <- test_ter$TARGET

# Convert training and test data into DMatrix format
dtrain <- xgb.DMatrix(data.matrix(X_train), label = as.numeric(y_train) - 1)
dval <- xgb.DMatrix(data.matrix(X_val), label = as.numeric(y_val) - 1)
dtest <- xgb.DMatrix(data.matrix(X_test))
```

We then define the hyperparameters for the model:

```
# Define the XGBoost parameters for multiclass classification
xgb_params <- list(
  objective = "multi:softprob",  # Output probabilities for each class
  num_class = 3,  # For 3 classes
  colsample_bytree = 0.7,
  subsample = 0.7,
  max_depth = 5,
  eta = 0.1,
  eval_metric = "mlogloss",
  seed = 123,
  nthread = 4
)

# Define training and validation watchlist
watchlist <- list(train = dtrain, valid = dval)
```

Next, we perform cross-validation to determine the optimal number of boosting rounds:

```r
# Perform cross-validation
xgb_cv <- xgb.cv(
  params = xgb_params,
  data = dtrain,
  nfold = 5,
  nrounds = 100,
  early_stopping_rounds = 10,
  maximize = FALSE,
  verbose = 0
)

# Retrieve the optimal number of rounds
optimal_nrounds <- xgb_cv$best_iteration
```

Using the optimal number of rounds, we train the final model:

```r
# Train the final model
gb_dt <- xgb.train(
  params = xgb_params,
  data = dtrain,
  nrounds = optimal_nrounds,
  watchlist = watchlist,
  print_every_n = 10
)
```

```
## Warning in xgb.train(params = xgb_params, data = dtrain, nrounds =
## optimal_nrounds, : xgb.train: `seed` is ignored in R package.  Use `set.seed()`
## instead.
```

```
## [1]  train-mlogloss:1.066778 valid-mlogloss:1.067025
## [11] train-mlogloss:0.908670 valid-mlogloss:0.912190
## [21] train-mlogloss:0.860144 valid-mlogloss:0.867202
## [31] train-mlogloss:0.840738 valid-mlogloss:0.851868
## [41] train-mlogloss:0.830033 valid-mlogloss:0.845144
## [51] train-mlogloss:0.822236 valid-mlogloss:0.841343
## [61] train-mlogloss:0.816304 valid-mlogloss:0.839155
## [71] train-mlogloss:0.810853 valid-mlogloss:0.837473
## [81] train-mlogloss:0.806281 valid-mlogloss:0.836376
## [91] train-mlogloss:0.801896 valid-mlogloss:0.835498
## [100]    train-mlogloss:0.798304 valid-mlogloss:0.834876
```

We will now compute the optimal thresholds and evaluate the model using the `calculate_ternary_metrics` function.

```r
# Predict probabilities on the test dataset
probabilities <- predict(gb_dt, newdata = dtest)
probabilities <- matrix(probabilities, ncol = 3, byrow = TRUE)  # Reshape predictions into a matrix

# Initialize class thresholds
initial_thresholds <- c(0.1, 0.3, 0.5)

# Use the optimization function to find best thresholds
optimized_thresholds <- optimize_thresholds(probabilities, y_test, initial_thresholds)

# Output the optimized thresholds
cat("Optimized thresholds:\n", optimized_thresholds, "\n")
```

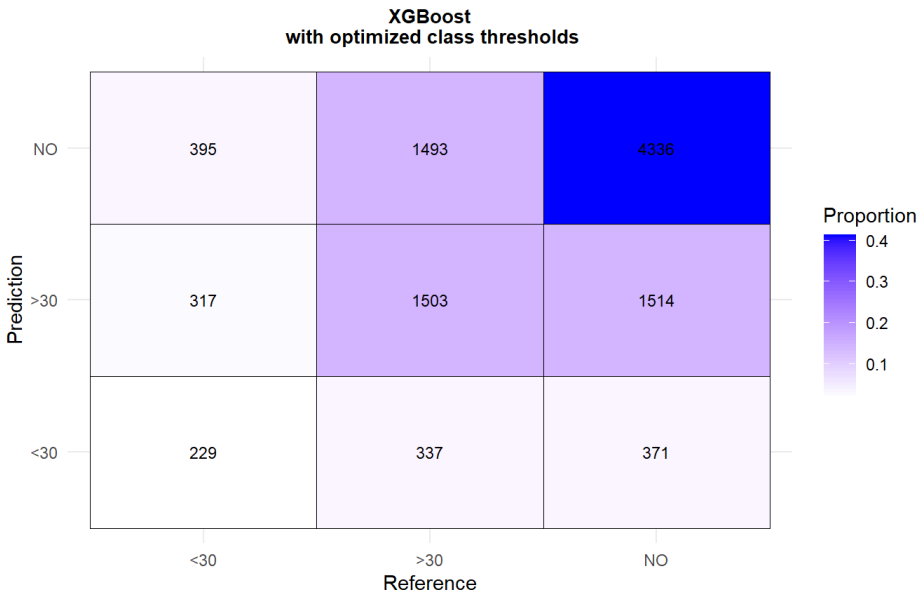
```
## Optimized thresholds:
##  0.1468027 0.3462064 0.5
```

```r
# Predict classes based on optimized thresholds
predictions_optimized <- apply(probabilities, 1, function(row) {
  if (row[1] > optimized_thresholds[1]) return("<30")
  else if (row[2] > optimized_thresholds[2]) return(">30")
  else if (row[3] > optimized_thresholds[3]) return("NO")
  else return(c("<30", ">30", "NO")[which.max(row)])  # Default to max probability
})

# Convert true labels and predictions to factors with correct levels
y_test_factor <- factor(y_test, levels = c("<30", ">30", "NO"))
predictions_factor <- factor(predictions_optimized, levels = c("<30", ">30", "NO"))

# Calculate metrics using optimized thresholds
metrics_xgb_optimized <- calculate_ternary_metrics(y_test_factor, predictions_factor)

# Print model performance results
cat("Model performance with optimized thresholds:\n",
    "Accuracy: ", metrics_xgb_optimized$Accuracy * 100, "%\n",
    "Balanced Accuracy: ", metrics_xgb_optimized$Balanced_Accuracy * 100, "%\n",
    "Macro-F1-Score: ", metrics_xgb_optimized$Macro_F1_Score * 100, "%\n")
```

```
## Model performance with optimized thresholds:
##  Accuracy:  57.81801 %
##  Balanced Accuracy:  46.37658 %
##  Macro-F1-Score:  46.386 %
```

As before, we now create the confusion matrix for the XGBoost model and visualize it using the optimized thresholds.

```
# Create the confusion matrix
confusion_matrix_xgb_ter <- confusionMatrix(predictions_factor, y_test_factor)

# Visualize the confusion matrix
cm_xgb <- plot_confusion_matrix(confusion_matrix_xgb_ter, "XGBoost\n with optimized class thresholds")
cm_xgb
```



To compare model performance, we present the previously computed evaluation metrics of both models side-by-side in a table.

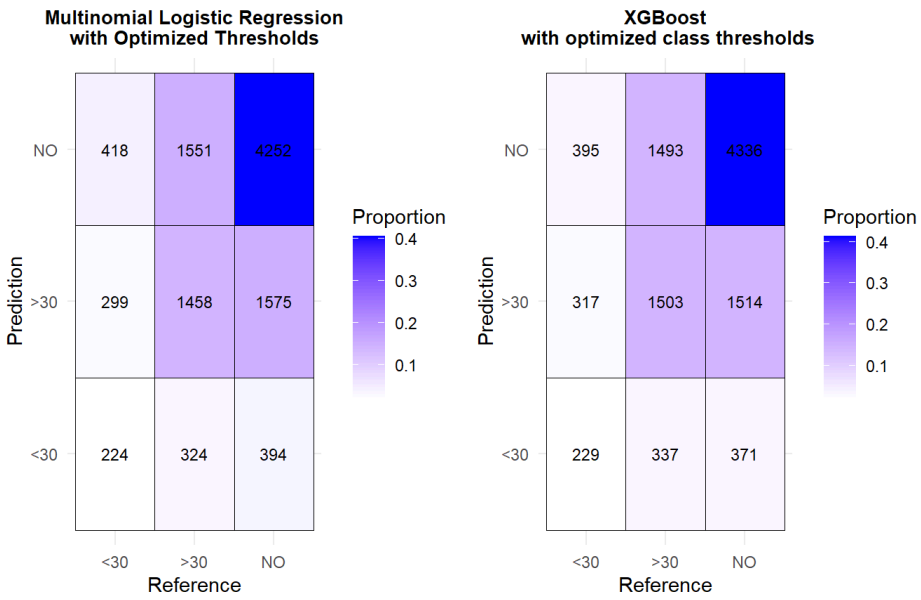
```
compare_model_performance <- function(metrics_multinom_optimized, metrics_xgb_optimized) {
  # Extract performance metrics for multinom model
  multinom_metrics <- data.frame(
    Model = "Multinomial",
    Accuracy = metrics_multinom_optimized$Accuracy * 100,
    Balanced_Accuracy = metrics_multinom_optimized$Balanced_Accuracy * 100,
    Macro_F1_Score = metrics_multinom_optimized$Macro_F1_Score * 100
  )

  # Extract performance metrics for xgb model
  xgb_metrics <- data.frame(
    Model = "XGBoost",
    Accuracy = metrics_xgb_optimized$Accuracy * 100,
    Balanced_Accuracy = metrics_xgb_optimized$Balanced_Accuracy * 100,
    Macro_F1_Score = metrics_xgb_optimized$Macro_F1_Score * 100
  )

  # Combine both dataframes into one
  performance_comparison <- rbind(multinom_metrics, xgb_metrics)

  return(performance_comparison)
}

# Assuming metrics_multinom_optimized and metrics_xgb_optimized are already defined
model_comparison <- compare_model_performance(metrics_multinom_optimized, metrics_xgb_optimized)
model_comparison
```

Model <chr>	Accuracy <dbl>	Balanced_Accuracy <dbl>	Macro_F1_Score <dbl>
Multinomial	56.54121	45.29933	45.2973
XGBoost	57.81801	46.37658	46.3860

2 rows

To compare the confusion matrices, we visualize the results of both models side by side.

```
multiplot(cm_multinom, cm_xgb, cols = 2)
```



Based on the model performance and the analysis of the confusion matrix, the XGBoost model can be identified as better suited for the given problem.

First, all three metrics — Accuracy, Balanced Accuracy, and Macro-F1-Score — show higher values for the XGBoost model compared to the multinomial model. This indicates that the XGBoost model demonstrates overall better performance by making correct predictions more frequently and providing a more balanced performance across the different classes.

Second, the confusion matrix of the XGBoost model exhibits a more favorable behavior since misclassifications tend to be closer to the actual classes. This means that when the model makes incorrect predictions, it tends to select a class similar to the true class rather than one that is far off. For example, if the true outcome is `<30`, it is more common with XGBoost that the model incorrectly predicts the class `>30` instead of `NO`, which is closer to the correct class. This is particularly important when the cost of misclassification varies between classes.

d) One-vs-One: Another way to evaluate the model is by calculating the AUC score per class, using a pairwise (One-vs-One) approach. Perform a One-vs-One analysis for the Multinomial Logistic Regression model from subtask R6b), where each possible pair of classes in `TARGET` is treated as a separate binary classification problem, and compute the AUC score for each pair using the test data. Compare and interpret the One-vs-One results, both among themselves and in relation to the results from the Logistic Regression model from Task R4 (binary classification: `<30` vs. `NO`).

```
# Select all class pairs
class_pairs <- combn(levels(test_ter$TARGET), 2, simplify = FALSE)

# Initialize lists for confusion matrices, AUC values, and additional info
conf_matrices <- list()
ovo_auc <- numeric(length(class_pairs))

# Iterate over each class pair and calculate AUC
for (i in seq_along(class_pairs)) {
  pair <- class_pairs[[i]]

  # Filter data for the current class pair
  pair_data <- test_ter[test_ter$TARGET %in% pair, ]

  # Predict probabilities for the current class pair
  pair_probs <- predict(multinom_model, newdata = pair_data, type = "probs")

  # Binarize the target variable: first class = 1, second class = 0
  binary_target <- ifelse(pair_data$TARGET == pair[1], 1, 0)

  # Calculate AUC using the calculate_auc function
  auc_result <- calculate_auc(binary_target, pair_probs[, pair[1]], verbose = FALSE)
  ovo_auc[i] <- auc_result$auc
}

# Output AUC values for each class pair
cat("\nAUC scores of One-vs-One analysis for Multinomial Logistic Regression (ternary):\n")
```

```
##
## AUC scores of One-vs-One analysis for Multinomial Logistic Regression (ternary):
```

```
for (i in seq_along(class_pairs)) {
  pair <- class_pairs[[i]]
  cat("Class pair", pair[1], "vs", pair[2], ":", round(ovo_auc[i], 4), "\n")
}
```

```
## Class pair <30 vs >30 : 0.6145
## Class pair <30 vs NO : 0.6876
## Class pair >30 vs NO : 0.6542
```

```
# Compare with binary classification from Task R4
cat("\nComparison with Task R4, Binary Classification: Logistic Regression (via nnet) for '<30 vs. NO'\n")
```

```
##
## Comparison with Task R4, Binary Classification: Logistic Regression (via nnet) for '<30 vs. NO'
```

```
print(auc_lr_nnet$auc)
```

```
## Area under the curve: 0.6892
```

The AUC score for the most distinct class pair `<30` vs. `NO` (readmission within 30 days vs. no readmission) is significantly higher than that of the other, more similar class pairs. Especially the separation between `<30` and `>30` (i.e., early vs. late readmission) performs the worst — the model struggles the most to distinguish between these classes.

Compared to the result of the binary Logistic Regression (via nnet) in Task R4, the AUC for `<30` vs. `NO` using the Multinomial Logistic Regression is slightly lower. However, considering the large AUC differences in the ternary setting, this difference may be regarded as minor.

e) Conclusion: Summarize your findings, addressing the following questions: What conclusions can be drawn from the results? What added value does ternary classification provide? How should the difference between the data basis for binary and ternary classification be assessed? Which models would you recommend in conclusion?

A conclusion from part d) is that, at least in this case, the Multinomial Logistic Regression can replace the simpler binary Logistic Regression from the previous tasks without loss in classification performance.

A possible added value of ternary classification for predicting readmissions lies in the ability to capture the “gray area” of late readmissions after 30 days. This group was excluded in the binary classification setup, making the data incomplete in terms of a binary decision.

In addition to the LASSO model recommended in Task R5 for binary classification — which is interpretable and parameter-efficient — the XGBoost model can be recommended for its high predictive power. It delivers by far the most accurate predictions for both binary and ternary classification. This is supported not only by its high AUC scores but also by the confusion matrices and evaluation metrics such as Accuracy, Balanced Accuracy, and Macro-F1 score, all of which are best for this model.

Exam Immersion April 13th to May 13th, 2025, Part II (Python)
"Diabetes Hospital Readmission"
Focus: Neural Networks and Embeddings

IMPORTANT NOTES FOR TASK COMPLETION

The second part of the immersion exam is to be implemented in Python. A pre-existing and pre-selected Jupyter Notebook will serve as a template and needs to be adapted to new data and modified questions.

The tasks PT1, PT2, PT3, and PT7 b) need to follow the notebook for Case Study 4 of DAV (see DAV Forecasting Rare Events - Credit Scoring) and must be adjusted according to the task description (i.e., modify, add, insert, remove, or comment text, code, and notebook cells accordingly). The notebook serving as a template is included in the exam files under the name `template2-credit-scoring.ipynb` and will hereafter be abbreviated as CSN.

The tasks PT0, PT4 through PT7 a) cover new topics not included in the template notebook. For these tasks, sufficient notebook cells need to be added and the notebook must be extended according to the task description.

For this work, the datasets `diabetic_data_bin.csv` and `icd9_data.csv`, generated in Part I (R Notebook), are required.

While editing, make sure to retain the section numbering of the source notebook CSN (in tasks PT1 through PT3). The corresponding headings may be adjusted if necessary. The texts taken from the template must be translated into English (e.g., using DeepL) and specifically adapted to the changed circumstances (e.g., data and results). Comments in code cells do not need to be translated. Unnecessary text can be deleted.

Task PT0: Setting up a Development Environment [Learning Goal 5.1; 1 Point]

For setting up the development environment, Python version >= 3.10.11 is recommended. A `requirements.txt` file has been added to the provided materials. You can use this file to install the required packages. The Python version used must be displayed. Additionally, the installed packages should be listed in an organized manner (five packages per row, with version numbers).

Suggested Solution:

In [4]:
```
# !pip3 install -r /teamspace/studios/this_studio/PK_VADS_IMMERSION/2025/1_Erster_Ansatz/requirements.txt
```

In [5]:
```
# check python version
from platform import python_version

print(python_version())
```

```
3.10.11
```

In [6]:
```
import pkg_resources
installed_packages = list(pkg_resources.working_set)

# Anzahl der Pakete pro Zeile
packages_per_line = 5

# Ausgabe der Pakete in Gruppen
for i in range(0, len(installed_packages), packages_per_line):
    print(' | '.join([f"{pkg.project_name}=={pkg.version}".center(30) for pkg in installed_packages[i:i+packages_per_line]]))
```

```
       Markdown==3.7          |       MarkupSafe==3.0.2       |        PySocks==1.7.1         |        PyYAML==6.0.2          |       Send
2Trash==1.8.3
       absl-py==2.1.0         |      accelerate==1.9.0        |        adagio==0.2.6          |    aiohappyeyeballs==2.6.1    |        aio
http==3.12.14
     aiohttp-cors==0.8.1      |       aiosignal==1.4.0        |        alembic==1.16.4        |     annotated-types==0.7.0    | antlr4-pyt
hon3-runtime==4.9.3
        anyio==4.9.0          |        appdirs==1.4.4         |      argon2-cffi==23.1.0      | argon2-cffi-bindings==21.2.0  |          a
rrow==1.3.0
       asttokens==3.0.0       |      astunparse==1.6.3        |        async-lru==2.0.5       |      async-timeout==5.0.1     |         at
trs==25.3.0
       autogluon==1.3.1       |   autogluon.common==1.3.1     |     autogluon.core==1.3.1     |   autogluon.features==1.3.1   |  autogluo
n.multimodal==1.3.1
   autogluon.tabular==1.3.1   | autogluon.timeseries==1.3.1   |        babel==2.17.0          |       beartype==0.21.0        |     beauti
fulsoup4==4.13.3
       bleach==6.2.0          |         blis==1.2.1           |        boto3==1.39.14         |      botocore==1.39.14        |       cach
etools==5.5.2
      catalogue==2.0.10       |       catboost==1.2.7         |      certifi==2024.12.14      |          cffi==1.17.1         |   charset-
normalizer==3.4.0
        click==8.2.1          |     cloudpathlib==0.21.1      |       cloudpickle==3.1.1      |        colorama==0.4.6        |        col
orful==0.5.7
       colorlog==6.9.0        |         comm==0.2.2           |       confection==0.1.5       |        contourpy==1.3.1       |      coref
orecast==0.0.15
       cycler==0.12.1         |        cymem==2.0.11          |       datasets==4.0.0         |        debugpy==1.8.11        |        dec
orator==5.1.1
      defusedxml==0.7.1       |         dill==0.3.8           |        distlib==0.4.0         |         einops==0.8.1         |        eva
luate==0.4.5
    exceptiongroup==1.2.2     |       executing==2.1.0        |        fastai==2.8.2          |       fastcore==1.8.7         |      fastd
ownload==0.0.7
    fastjsonschema==2.21.1    |     fastprogress==1.0.3       |     fasttransform==0.0.2      |       filelock==3.18.0        |      flatb
uffers==24.3.25
      fonttools==4.55.3       |         fqdn==1.5.1           |      frozenlist==1.7.0        |          fs==2.4.16           |        fss
pec==2025.3.0
        fugue==0.9.1          |        future==1.0.0          |         gast==0.6.0           |         gdown==5.2.0          |        glu
onts==0.16.2
   google-api-core==2.25.1    |     google-auth==2.40.3       |     google-pasta==0.2.0       | googleapis-common-protos==1.70.0 |         g
raphviz==0.20.3
       greenlet==3.2.3        |        grpcio==1.68.1         |         h11==0.14.0           |         h5py==3.12.1          |        htt
pcore==1.0.7
        httpx==0.28.1         |   huggingface-hub==0.34.1     |       hyperopt==0.2.7         |          idna==3.10           |        ima
geio==2.37.0
  importlib-metadata==8.5.0   |      ipykernel==6.29.5        |       ipython==8.31.0         |      ipywidgets==8.1.7        |      isodu
ration==20.11.0
        jedi==0.19.2          |        jinja2==3.1.5          |       jmespath==1.0.1         |        joblib==1.4.2          |         js
on5==0.10.0
      jsonpointer==3.0.0      |      jsonschema==4.23.0       | jsonschema-specifications==2024.10.1 |    jupyter-client==8.6.3      |
jupyter-core==5.7.2
    jupyter-events==0.12.0    |      jupyter-lsp==2.2.5       |    jupyter-server==2.15.0     | jupyter-server-terminals==0.5.3 |       jup
yterlab==4.3.6
  jupyterlab-pygments==0.3.0  |  jupyterlab-server==2.27.3    |  jupyterlab-widgets==3.0.15   |         keras==3.7.0          |       kiwi
solver==1.4.7
       langcodes==3.5.0       |     language-data==1.3.0      |       lazy-loader==0.4        |       libclang==18.1.1        |        lig
htgbm==4.5.0
       lightning==2.5.2       | lightning-utilities==0.15.0   |       llvmlite==0.44.0        |         mako==1.3.10          |       mari
sa-trie==1.2.1
    markdown-it-py==3.0.0     |      matplotlib==3.10.0       |   matplotlib-inline==0.1.7    |          mdurl==0.1.2         |         mi
stune==3.1.3
       ml-dtypes==0.4.1       |      mlforecast==0.13.6       |     model-index==0.1.11       |        mpmath==1.3.0          |        ms
gpack==1.1.1
       multidict==6.6.3       |    multiprocess==0.70.16      |      murmurhash==1.0.13       |          namex==0.0.8         |        nbc
lient==0.10.2
      nbconvert==7.16.6       |       nbformat==5.10.4        |      nest-asyncio==1.6.0      |        networkx==3.4.2        |         nl
paug==1.1.11
        nltk==3.8.1           |       notebook==7.3.3         |     notebook-shim==0.2.4      |         numba==0.61.0         |         nu
mpy==1.26.4
    nvidia-ml-py3==7.352.0    |       omegaconf==2.3.0        |      opencensus==0.11.4       |  opencensus-context==0.1.3    |      opend
atalab==0.0.10
       openmim==0.3.9         |       openxlab==0.0.11        |      opt-einsum==3.4.0        |         optree==0.13.1        |         op
tuna==4.4.0
      ordered-set==4.1.0      |        orjson==3.11.1         |       overrides==7.7.0        |        packaging==24.2        |         pa
ndas==2.2.3
     pandocfilters==1.5.1     |         parso==0.8.4          |         patsy==1.0.1          |       pdf2image==1.17.0       |         pe
xpect==4.9.0
      pickleshare==0.7.5      |        pillow==11.0.0         |          pip==24.3.1          |     platformdirs==4.3.6       |         pl
otly==5.24.1
     plum-dispatch==2.5.7     |       preshed==3.0.10         |  prometheus-client==0.21.1    |    prompt-toolkit==3.0.48     |         pro
pcache==0.3.2
      proto-plus==1.26.1      |       protobuf==5.29.2        |        psutil==6.1.0          |       ptyprocess==0.7.0       |         pur
e-eval==0.2.3
        py4j==0.10.9.9        |        py-spy==0.4.0          |       pyarrow==21.0.0         |         pyasn1==0.6.1         |     pyasn1
-modules==0.4.2
       pycparser==2.22        |     pycryptodome==3.23.0      |       pydantic==2.11.7        |    pydantic-core==2.33.2      |         pyg
ments==2.18.0
       pyparsing==3.2.0       |     pytesseract==0.3.13       | python-dateutil==2.9.0.post0  |   python-json-logger==3.3.0   |    pytorch
-lightning==2.5.2
pytorch-metric-learning==2.8.1 |         pytz==2024.2          |         pywin32==310          |        pywinpty==2.0.15       |         py
zmq==26.2.0
         ray==2.44.1          |     referencing==0.36.2       |       regex==2024.11.6        |       requests==2.32.3        |    rfc3339
-validator==0.1.4
   rfc3986-validator==0.1.1   |         rich==13.9.4          |       rpds-py==0.23.1         |          rsa==4.9.1           |       s3tr
ansfer==0.13.1
      safetensors==0.5.3      |       scikeras==0.13.0        |     scikit-image==0.25.2      |     scikit-learn==1.5.2       |         sc
ipy==1.14.1
       seaborn==0.13.2        |     sentencepiece==0.2.0      |        seqeval==1.2.2         |       setuptools==75.6.0      |          s
hap==0.47.1
```

```
      shellingham==1.5.4       |          six==1.17.0          |         slicer==0.0.8         |    smart-open==7.3.0.post1     |          sn
iffio==1.3.1
       soupsieve==2.6          |          spacy==3.8.7         |      spacy-legacy==3.0.12     |      spacy-loggers==1.0.5      |       sqla
lchemy==2.0.41
        srsly==2.5.1           |       stack-data==0.6.3       |      statsforecast==2.0.1     |      statsmodels==0.14.4       |          sy
mpy==1.13.1
       tabulate==0.9.0         |        tenacity==9.0.0        |      tensorboard==2.18.0      | tensorboard-data-server==0.7.2 |      tenso
rboardx==2.6.4
      tensorflow==2.18.0       |    tensorflow-intel==2.18.0   | tensorflow-io-gcs-filesystem==0.31.0 |        termcolor==2.5.0        |
terminado==0.18.1
     text-unidecode==1.3       |          thinc==8.3.4         |      threadpoolctl==3.5.0     |      tifffile==2025.5.10       |          t
imm==1.0.3
       tinycss2==1.4.0         |       tokenizers==0.21.2      |          tomli==2.2.1         |         toolz==0.12.1          |          t
orch==2.6.0
     torchmetrics==1.7.4       |      torchvision==0.21.0      |         tornado==6.4.2        |          tqdm==4.67.1          |       trai
tlets==5.14.3
     transformers==4.49.0      |          triad==0.9.8         |         typer==0.16.0         | types-python-dateutil==2.9.0.20241206 |   t
yping-extensions==4.12.2
   typing-inspection==0.4.1    |         tzdata==2024.2        |      uri-template==1.3.0      |         urllib3==2.3.0         |     utilsf
orecast==0.2.10
     virtualenv==20.32.0       |         wasabi==1.1.3         |        wcwidth==0.2.13        |         weasel==0.4.1          |       webc
olors==24.11.1
     webencodings==0.5.1       |    websocket-client==1.8.0    |        werkzeug==3.1.3        |         wheel==0.45.1          |   widgetsn
bextension==4.0.14
      window-ops==0.0.15       |         wrapt==1.17.0         |         xgboost==2.1.3        |         xxhash==3.5.0          |          y
arl==1.20.1
         zipp==3.21.0          |       autocommand==2.2.2      |    backports.tarfile==1.2.0   |         inflect==7.3.1         |   jaraco.c
ollections==5.1.0
    jaraco.context==5.3.0      |    jaraco.functools==4.0.1    |      jaraco.text==3.12.1      |     more-itertools==10.3.0     |        typ
eguard==4.3.0
```

Task PT1: Creating a Simple Yet Good Benchmark Model [Learning Goal 5.1; 4 Points]

Based on the CSN notebook (Part A), make the following adjustments:

- Remove the cells preceding the text "Part A: Quick & Easy" that relate to the original topic of Credit Scoring.
- In "Part A: Quick & Easy":
  - Add all packages used in this notebook to Section 1.1 and remove unnecessary imports. Use a `RANDOM_SEED` of 42.
  - Replace the dataset `../input/home-credit-default-risk/application_train.csv` used in Section 1.2 with the dataset `diabetic_data_bin.csv` generated in the R-part. What steps are necessary to correctly load this dataset?
  - Set the data type of all features whose names end with `_id` to `object`. What is the benefit of this approach?
  - For Part A, no dataset modifications other than those already present in the notebook should be made. If specific code blocks are unnecessary, comment them out with appropriate notes. Reminder: Follow the general instruction throughout the notebook to translate texts and adjust them to the dataset used.

Suggested Solution:

Part A: Quick and Easy

1. Development of a Baseline Model.

In this initial section, we will quickly create a basic machine learning model for the binary classification task of predicting hospital readmissions for diabetes patients. This baseline model will serve as a benchmark for all subsequent machine learning models.

1.1 Setting up the Modeling Environment

First, we will import all necessary libraries, adjust visualization parameters, and set a random seed to ensure the reproducibility of results.

In [ ]:
```
%%sql
```

In [7]:
```
import os
# Set environment variable to ignore all warnings
os.environ['PYTHONWARNINGS'] = 'ignore'

# Standard Python libraries for data analysis, scientific computing, and plotting
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
import re
from scipy.stats import uniform, loguniform

# Setting display options for pandas
pd.set_option('display.max_columns', None)
COLOR_LIGHT, COLOR_DARK = '#849CBE', '#00548A'

# Scikit-learn modules for preprocessing and machine learning models
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import roc_auc_score
```

```
# Gradient boosting frameworks and logistic regression
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
import statsmodels.formula.api as smf

# Tensorflow, Keras
import tensorflow as tf
from keras import Input
from keras import Model
from keras import regularizers
from keras.api.layers import Embedding, Flatten, concatenate, Dense
from keras.src.utils import set_random_seed

# Ensure reproducibility
RANDOM_SEED = 42
set_random_seed(RANDOM_SEED)
```

In [8]:
```
# Ergänzung: Zeitmessung Notebook
start_time_nb = time.time()
```

To optimize resource usage, we adjust TensorFlow GPU memory settings, enabling efficient computation throughout the entire modeling workflow.

In [9]:
```
# Prevent TensorFlow From Fully Allocating GPU Memory
# Ref: https://www.tensorflow.org/guide/gpu#limiting_gpu_memory_growth
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
    # Currently, memory growth needs to be the same across GPU
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        logical_gpus = tf.config.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
    except RuntimeError as e:
        # Memory growth must be set before GPUs have been initialized
        print(e)
else:
    print("No GPU(s) found")
```

```
No GPU(s) found
```

1.2 Loading and Inspecting Input Data

Next, we will load the mentioned dataset and display its dimensions by printing the number of rows and columns. We will also display some randomly chosen rows to illustrate the content and structure of the data.

Addition: While loading, ensure that the "None" values in the dataset are not interpreted as missing values (`na_values=[""], keep_default_na=False`).

In [10]:
```
# Load the dataset into a pandas DataFrame
#df_raw = pd.read_csv("diabetic_data_bin.csv", na_values=[""], keep_default_na=False)
#df_raw = pd.read_csv("./PK_VADS_IMMERSION/2025/1_Erster_Ansatz/diabetic_data_bin.csv", na_values=[""], keep_default_na=False)
df_raw = pd.read_csv("../1_Erster_Ansatz/diabetic_data_bin.csv", na_values=[""], keep_default_na=False)

# Display the dimensions of the dataset (rows, columns)
print(f"Input Dataset dimensions (rows, columns): {df_raw.shape}")

# Display a random sample of 10 rows from the dataset to inspect data
df_raw.sample(n=7, random_state=RANDOM_SEED)
```

```
Input Dataset dimensions (rows, columns): (47751, 51)
```

Out[10]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_speci
47523	Asian	Female	[50-60)	?	3	6	1	1	?	Orthope
8582	Caucasian	Male	[40-50)	?	2	6	4	8	?	
42024	Caucasian	Female	[70-80)	?	1	3	7	3	MC	
15219	AfricanAmerican	Female	[20-30)	?	1	1	7	2	?	InternalMedi
45649	Caucasian	Female	[60-70)	?	1	2	7	1	?	
23448	Caucasian	Male	[80-90)	?	3	28	1	1	SP	
39945	Caucasian	Male	[40-50)	?	1	1	7	4	?	

In [11]:
```
# Aktuelle Datentypen
pd.DataFrame(df_raw.dtypes).T
```

Out[11]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_specialty	num_la
0	object	object	object	object	int64	int64	int64	int64	object	object	

In [12]:
```
# Step 1: Set the type of columns ending with "_id" to "object"
df_raw = df_raw.astype({col: 'object' for col in df_raw.columns if col.endswith('_id')})
```

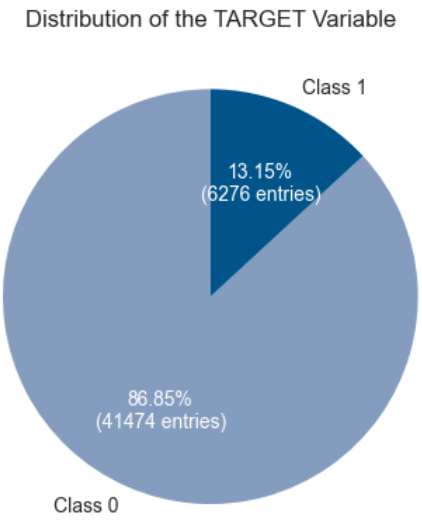
In [13]:
```
# Kontrolle der Datentypen
pd.DataFrame(df_raw.dtypes).T
```

Out[13]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_specialty	num_la
0	object	object	object	object	object	object	object	int64	object	object	

The affected features (e.g., `discharge_disposition_id`) contain numerical values but are, in fact, categorical data types. This is made explicit by converting them to the `object` data type.

In [14]:
```
# Count the occurrences of each unique value in the 'TARGET' column
target_counts = df_raw['TARGET'].value_counts()

# Calculate the total number of entries
total_entries = len(df_raw['TARGET'])

# Define a custom formatting function for the pie chart labels
def custom_autopct(pct):
    absolute = int(pct/100.*total_entries)
    return "{:.2f}%\n({:d} entries)".format(pct, absolute)

# Create a pie chart to visualize the distribution of classes in the 'TARGET' column
pie_chart, _, autotexts = plt.pie(target_counts, labels=target_counts.index.map({0: 'Class 0', 1: 'Class 1'}),
                                  autopct=custom_autopct, startangle=90,
                                  colors=[COLOR_LIGHT, COLOR_DARK], wedgeprops={'edgecolor': 'none'})

# Set the color of the autopct texts to white
for autotext in autotexts:
    autotext.set_color('white')

# Plot the pie chart
plt.title('Distribution of the TARGET Variable')
plt.show()
```



The pie chart clearly shows an imbalance in the distribution of the dependent variable 'TARGET'. Class '1' appears significantly less frequently than class '0'. In such cases, accuracy is not a suitable metric, but the area under the ROC (Receiver Operating Characteristic) curve is recommended.

1.3 Replacing Missing Categorical Values

While many machine learning (ML) tools require the more complex data preparation steps described in Part B, for CatBoost, our ML tool of choice, only the minimal data preparation described below is required, where missing categorical values are replaced with a predefined value.

In [15]:
```
# Identify categorical features by data type 'object'
categorical_features = list(df_raw.select_dtypes(include=['object']).columns)

# Compute the number of unique values for each categorical feature and reset the index to make 'Feature Names' a column
categorical_feature_counts = df_raw[categorical_features].nunique().reset_index()

# Set the column names to 'Feature' and 'Number of Unique Values'
categorical_feature_counts.columns = ['Categorical Feature', 'Number of Unique Values']

# Sort the DataFrame by 'Number of Unique Values' in descending order
categorical_feature_counts.sort_values(by='Number of Unique Values', ascending=False, inplace=True)
```

```
# Display the resulting DataFrame
display(categorical_feature_counts.style.hide(axis='index'))
```

Categorical Feature	Number of Unique Values
diag_3	724
diag_2	678
diag_1	661
medical_specialty	45
payer_code	17
discharge_disposition_id	15
admission_source_id	11
group_diag_3	10
group_diag_1	10
age	10
group_diag_2	10
weight	8
admission_type_id	7
race	6
metformin	4
pioglitazone	4
max_glu_serum	4
A1Cresult	4
insulin	4
repaglinide	4
rosiglitazone	4
glyburide	4
glimepiride	4
glipizide	4
acarbose	3
nateglinide	3
tolbutamide	2
chlorpropamide	2
diabetesMed	2
change	2
gender	2
glyburide.metformin	2
miglitol	2
tolazamide	2
citoglipton	1
glipizide.metformin	1
glimepiride.pioglitazone	1
metformin.rosiglitazone	1
metformin.pioglitazone	1
examide	1
acetohexamide	1
troglitazone	1

Next, we identify the categorical features that contain missing values and calculate the percentage of missing values.

In [16]:
```
# Define a function that returns a DataFrame with features containing missing values
# and the percentage of missing values for each of these features
def missing_values_percentage(data):
    # Calculate percentages and create a DataFrame
    nan_percentages = data.isna().sum() * 100 / len(data)
    missing_values_df = pd.DataFrame({
        'Categorical Feature': nan_percentages.index,
        'Missing Percent': nan_percentages.values
    })
    # Format the 'Missing Percent' column as a percentage string with two decimal places
    missing_values_df['Missing Percent'] = missing_values_df['Missing Percent'].apply(lambda x: f'{x:.2f}%')
    # Filter out features without missing values and sort by percentage
```

```
    missing_values_df = missing_values_df[missing_values_df['Missing Percent'] != '0.00%']
    missing_values_df.sort_values(by='Missing Percent', ascending=False, inplace=True)
    return missing_values_df

# Calculate the proportion of missing values for the categorical features
missing_values_df = missing_values_percentage(df_raw[categorical_features])

# Display the resulting DataFrame without the index using the new recommended Styler.hide method
missing_values_df_styled = missing_values_df.style.hide(axis='index')
display(missing_values_df_styled)
```

Categorical Feature	Missing Percent

In [ ]:

To account for missing values in categorical columns without imputing data or deleting datasets, we assign instances of missing data to their own category, `__missingValue__`. This strategy allows our machine learning algorithms to process missing values as a standalone informative feature, preserving the full scope of the dataset and the integrity of the input for modeling.

Note: Since the dataset does not contain missing values, the next cell is unnecessary and will be commented out.

In [17]:
```
# Replace missing values with a new category '__missingValue__' in all categorical columns
# df_raw[categorical_features] = df_raw[categorical_features].fillna('__missingValue__')
```

1.4 Splitting the data into training, validation, and test subsets

When training a machine learning classifier, we do not only want it to learn to model the training data or simply memorize it (ML models can be large and complex enough to do this). We want the model to generalize to data it has never seen before. Therefore, the model's performance is typically measured on a test set consisting of examples that have never been seen before. For consistency with future model optimization strategies, we randomly split our dataset into three parts:

- Training data: 70% of the data for model training
- Validation data: 15% to validate our models after training and potentially decide on changes
- Test data: 15% for the final measurement of generalization performance across all relevant models

This split enables effective model training, hyperparameter tuning to prevent overfitting, and an unbiased evaluation of performance on new, unseen data.

In [18]:
```
# Constants for data split ratios
TRAIN_RATIO = 0.7
VAL_TEST_RATIO = 0.5  # Splitting the remaining 30% equally into validation and test

# Separate the features (X) and the target (y)
X_raw = df_raw.drop(columns=['TARGET'], axis=1)
y = df_raw['TARGET']

# Split the dataset into a training set and a combined validation and test set
X_raw_train, X_raw_val_test, y_train, y_val_test = train_test_split(X_raw, y, train_size=TRAIN_RATIO, random_state=RANDOM_SEED)

# Further split the combined validation and test set into separate validation and test sets
X_raw_val, X_raw_test, y_val, y_test = train_test_split(X_raw_val_test, y_val_test, test_size=VAL_TEST_RATIO, random_state=RANDOM_SEED)

# Verify the dimensions of the training, validation, and test sets by displaying (rows, columns)
print(f"Training set dimensions (rows, columns):   {X_raw_train.shape}")
print(f"Validation set dimensions (rows, columns):  {X_raw_val.shape}")
print(f"Test set dimensions (rows, columns):        {X_raw_test.shape}")
```

```
Training set dimensions (rows, columns):   (33425, 50)
Validation set dimensions (rows, columns):  (7163, 50)
Test set dimensions (rows, columns):        (7163, 50)
```

Check the means and standard deviations of the variable TARGET `y` for the three samples:

In [19]:
```
# Calculate TARGET mean and std for each set
stats = pd.DataFrame({
    'Mean': [y_train.mean(), y_val.mean(), y_test.mean()],
    'Standard Deviation': [y_train.std(), y_val.std(), y_test.std()]
}, index=['Train', 'Validation', 'Test']).round(4)

print(stats)
```

```
             Mean  Standard Deviation
Train       0.1314              0.3378
Validation  0.1312              0.3377
Test        0.1319              0.3384
```

These randomly generated differences in the means of the TARGET variable may initially seem small, but they can impact the final evaluation.

1.5 Training a standard CatBoost classifier (baseline model)

In [20]:
```
# Prepare data for AutoGluon (PT7)
Xy_raw_train = X_raw_train.copy()
Xy_raw_val = X_raw_val.copy()
Xy_raw_test = X_raw_test.copy()
Xy_raw_train['TARGET'] = y_train
Xy_raw_val['TARGET'] = y_val
Xy_raw_test['TARGET'] = y_test
```

We proceed by applying the CatBoost model to our prepared dataset, using the default settings for a straightforward evaluation without hyperparameter tuning.

In [21]:
```
# Start timer to calculate the running time of training the CatBoost model
start_time = time.time()

# Fit the CatBoostClassifier on the training data
CB1 = CatBoostClassifier(eval_metric='AUC', random_seed=RANDOM_SEED)
CB1.fit(X_raw_train, y_train, cat_features=categorical_features, logging_level='Silent')

# Calculate and print the running time in seconds
# The time taken by CatBoost is saved for later use with AutoGluon.
elapsed_time_CB1 = time.time() - start_time
print(f"Elapsed time (sec): {elapsed_time_CB1:.2f}")
```

```
Elapsed time (sec): 103.05
```

To demonstrate how extensive the customization options for refining the CatBoost model are, we present a complete list of hyperparameters and their default settings.

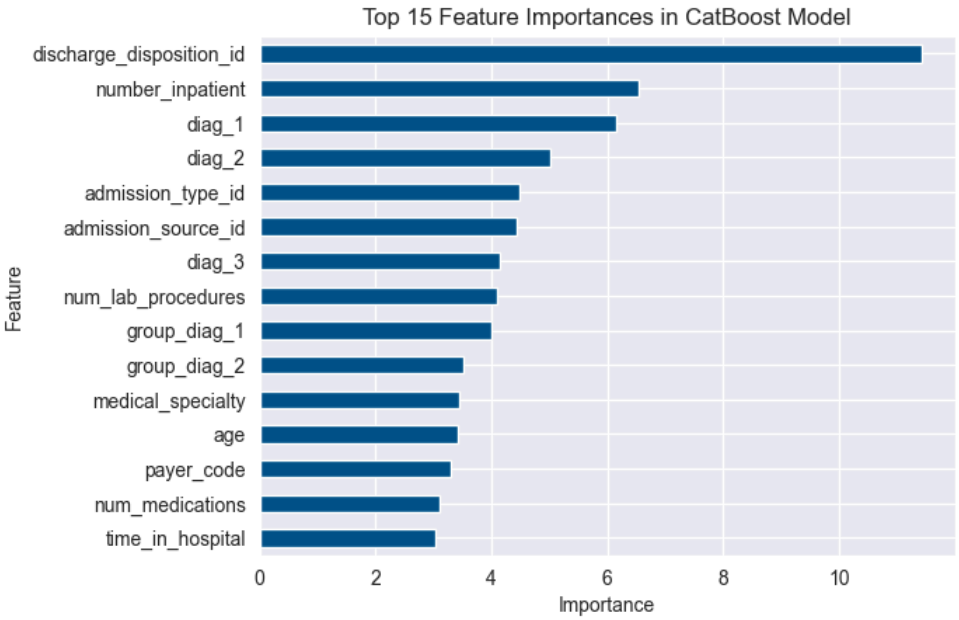
In [22]:
```
# Display all model hyperparameters in a DataFrame
hyperparams_list = [(k, v) for k, v in CB1.get_all_params().items()]
hyperparams_df = pd.DataFrame(hyperparams_list, columns=['Hyperparameter', 'Value'])
display(hyperparams_df.style.hide(axis='index'))
```

Hyperparameter	Value
nan_mode	Min
eval_metric	AUC
combinations_ctr	['Borders:CtrBorderCount=15:CtrBorderType=Uniform:TargetBorderCount=1:TargetBorderType=MinEntropy:Prior=0/1:Prior=0.5/1:Prior=1/1', 'Counter:CtrBorderCount=15:CtrBorderType=Uniform:Prior=0/1']
iterations	1000
sampling_frequency	PerTree
fold_permutation_block	0
leaf_estimation_method	Newton
random_score_type	NormalWithModelSizeDecrease
counter_calc_method	SkipTest
grow_policy	SymmetricTree
penalties_coefficient	1
boosting_type	Plain
model_shrink_mode	Constant
feature_border_type	GreedyLogSum
ctr_leaf_count_limit	18446744073709551615
bayesian_matrix_reg	0.100000
one_hot_max_size	2
eval_fraction	0
force_unit_auto_pair_weights	False
l2_leaf_reg	3
random_strength	1
rsm	1
boost_from_average	False
max_ctr_complexity	4
model_size_reg	0.500000
simple_ctr	['Borders:CtrBorderCount=15:CtrBorderType=Uniform:TargetBorderCount=1:TargetBorderType=MinEntropy:Prior=0/1:Prior=0.5/1:Prior=1/1', 'Counter:CtrBorderCount=15:CtrBorderType=Uniform:Prior=0/1']
pool_metainfo_options	{'tags': {}}
subsample	0.800000
use_best_model	False
class_names	[0, 1]
random_seed	42
depth	6
ctr_target_border_count	1
posterior_sampling	False
has_time	False
store_all_simple_ctr	False
border_count	254
classes_count	0
auto_class_weights	None
sparse_features_conflict_fraction	0
leaf_estimation_backtracking	AnyImprovement
best_model_min_trees	1
model_shrink_rate	0
min_data_in_leaf	1
loss_function	Logloss
learning_rate	0.046101
score_function	Cosine
task_type	CPU
leaf_estimation_iterations	10
bootstrap_type	MVS
max_leaves	64

Hyperparameter	Value
permutation_count	4

To gain initial insights into the behavior of the trained model, we visualize the built-in feature importances, which highlight the most influential features for the predictions.

In [23]:
```
# Extract the 15 most important features and create a Series for plotting
feature_importances = pd.Series(CB1.feature_importances_, index=X_raw_train.columns).nlargest(15)

# Plot the feature importances
feature_importances.plot(kind='barh', color=COLOR_DARK)
plt.gca().invert_yaxis()
plt.title('Top 15 Feature Importances in CatBoost Model')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



Addition:

The above representation of the feature importance plot shows that the most influential predictors are `discharge_disposition_id` and `number_inpatient`. These are followed by the three diagnosis details `diag_1`, `diag_2`, and `diag_3`. `discharge_disposition_id` contains information about where patients are discharged to. For example, IDs 1, 6, and 8 represent discharge to home (as well as the type of care provided there). The second most important feature is `number_inpatient`, which indicates the number of inpatient visits in the last year.

Next, the performance of our CatBoost model will be evaluated using the validation set by employing the Area Under the Receiver Operating Characteristic Curve (AUC) as our chosen evaluation metric (see above).

In [24]:
```
# Calculate the AUC score using the model's prediction probabilities for the positive class
auc_CB1 = roc_auc_score(y_val, CB1.predict_proba(X_raw_val)[:, 1])

# Print the validation AUC
print(f'The validation AUC of the CatBoost model with standard hyperparameters is: {auc_CB1:.6f}')
```

```
The validation AUC of the CatBoost model with standard hyperparameters is: 0.708208
```

In summary, Part A has created a solid foundation for our binary classification task using a CatBoost model with minimal data preparation and without hyperparameter tuning. This is already a complex model, yet it is very simple to build and known for its high predictive power with default parameters. This benchmark serves as a reference point to evaluate the value of the more complex models developed in the following sections.

Task PT2: Logistic Regression and Feature Analysis [Learning Objective 5.1 / 2.2; 10 Points]

Based on the CSN Notebook (Part B), the following adjustments need to be made:

- a) In Section 2.1, only necessary operations are to be performed. What can be (justifiably) commented out?
- b) For the logistic regression in Section 2.2, the following approach should be taken: Refer to the feature importance list from CatBoost and construct a model with at least five features. Pay attention to the scale of the features. Where do problems arise, and how can they be explained?
- c) In Section 2.3, only relevant components are to be worked on. What can be (justifiably) commented out?
- d) The analysis of the categorical features in Section 3.1 is to be limited to `group_diag_1`, `group_diag_2`, `group_diag_3`, `age`, `medical_specialty`, `payer_code`, and `discharge_disposition_id`.
- e) When analyzing the numerical features in Section 3.2, consider `number_outpatient`, `number_inpatient`, `num_procedures`, `num_lab_procedures`, `num_medications`, `time_in_hospital`, `number_emergency`, and `number_diagnoses`.

- Section 3.3 and Section 4 (Enhancing CatBoost's Performance with New Features) are not considered at this point.
- Section 5 is also not considered (it will be addressed later as part of PT7).

- f) Section 6 is to be executed:
  - In Section 6.3, the `non_event_factor` is to be adjusted with justification.
  - After Section 6.4, a new Section 6.5 is to be added, where the previous results of the models are to be compared. Refer to Section 4.2 of the CSN Notebook for guidance.

Proposed Solution:

Part B: Gaining Insights from Data and Modeling

Part B focuses on refining our machine learning strategy to gain valuable insights. We begin with a thorough examination of logistic regression, a cornerstone of actuarial analytics, in its classic, unregularized form. We then proceed to targeted exploratory data analysis and feature engineering to improve the predictive power of our models. Another emphasis is placed on the explainability of models. To lay the groundwork for advanced modeling techniques in Part C, we conclude with essential preprocessing activities: encoding, scaling, and subsampling.

2. Logistic Regression: A Classical Approach to Classification

In this section, we focus on logistic regression, a classical and fundamental technique for binary classification tasks. Since this method cannot handle missing feature values, we start by imputing missing numerical values with median values. Next, we implement and evaluate a classical logistic regression model with low parameters. Subsequently, we fit the CatBoost model based on the modified datasets, compare the performance of the new model with that of the benchmark model, and interpret the results.

2.1 Data Preparation: Imputation of Missing Numbers with Median

Addition: As shown above, there are no missing values in the numerical variables. Therefore, the corresponding section is commented out.

In this subsection, we address the challenge of missing data by imputing these missing values. This is an essential step because logistic regression requires complete datasets to compute accurate predictions and model parameters. Missing values for categorical features with a unique category were already imputed in Subsection 1.3. For numerical features, we choose the median, as it is robust to outliers and effectively preserves the distribution of skewed datasets. Although median imputation reduces data variability and may introduce biases if the missing data is not random, its simplicity provides a practical approach for building a baseline model in binary classification tasks. The following code snippet demonstrates how we impute missing values in numerical features using the median calculated from the training data to avoid data loss.

In [25]:
```
# Identify numerical features
numerical_features = list(X_raw_train.select_dtypes(exclude=['object']).columns)

# Copy the Input Data
df_pre = df_raw.copy()

num_missing = df_pre.isnull().sum().sum()
print(f"\nNumber of missing values before imputation: {num_missing}")

# Replace missing values for numeric features by median value from the training data
#for col in numerical_features:
#    median_value = X_raw_train[col].median()
#    df_pre[col] = df_pre[col].fillna(median_value)

# Check for remaining missing values:
# num_missing = df_pre.isnull().sum().sum()
# print(f"\nNumber of missing values after imputation: {num_missing}")
```

```
Number of missing values before imputation: 0
```

While our CatBoost model had no issues with very rare feature values, this could cause problems for the upcoming logistic regression. Therefore, we replace the extremely rare values with unknown gender (Unknown/Invalid) with the value "Female."

Addition: A replacement has already been performed in the R-Notebook, so the corresponding part is commented out.

In [26]:
```
# print("\nGender_Frequency: ", df_pre.CODE_GENDER.value_counts())
# df_pre.loc[df_pre["CODE_GENDER"] != "M", "CODE_GENDER"] = "F"
```

As already described in Subsection 1.4, we will split our preprocessed dataset into separate training, validation, and test datasets. This crucial step, which is essential for training, fine-tuning, and evaluating our machine learning models, ensures that our model can be validated and tested on data it has not seen during the training process.

In [27]:
```
# Constants for data split ratios
TRAIN_RATIO = 0.7
VAL_TEST_RATIO = 0.5  # Splitting the remaining 30% equally into validation and test

# Separate the features (X) and the target (y) after preprocessing
X_pre = df_pre.drop(['TARGET'], axis=1)
y_pre = df_pre['TARGET']

# Split the preprocessed dataset into a training set and a combined validation and test set
X_pre_train, X_pre_val_test, y_pre_train, y_pre_val_test = train_test_split(X_pre, y_pre, train_size=TRAIN_RATIO, random_state=RANDOM_SEED)

# Further split the combined validation and test set into separate validation and test sets
X_pre_val, X_pre_test, y_pre_val, y_pre_test = train_test_split(X_pre_val_test, y_pre_val_test, test_size=VAL_TEST_RATIO, random_state=RANDOM

# Verify the dimensions of the training, validation, and test sets by displaying (rows, columns)
```

```
print(f"Preprocessed training set dimensions (rows, columns):   {X_pre_train.shape}")
print(f"Preprocessed validation set dimensions (rows, columns): {X_pre_val.shape}")
print(f"Preprocessed test set dimensions (rows, columns):       {X_pre_test.shape}")
```

```
Preprocessed training set dimensions (rows, columns):   (33425, 50)
Preprocessed validation set dimensions (rows, columns): (7163, 50)
Preprocessed test set dimensions (rows, columns):       (7163, 50)
```

2.2 Implementation of Logistic Regression: Fitting and Performance Evaluation

To prepare for a logistic regression analysis that reflects the formula-based modeling style of R, the features and the target variable are combined into a single data frame. This setup facilitates the use of the formula API from statsmodels to define and fit the logistic regression model in a concise manner. With an eye on potential overfitting and convergence issues, we gradually select features from the initial CatBoost list. Convergence problems already arise if "diag_1" - "diag_3" are included in the list. The reason for this is the many classes present in these features, some of which have very low frequencies. These features will therefore be ignored for further processing. A similar problem occurs if "medical_specialty" is used (see the evaluation of the number of categories in Part 1). Therefore, additional variable inclusion is stopped at this point. In total, the model includes **five** variables. Once the model has been fitted, we provide a comprehensive overview of the logistic regression results to facilitate interpretation and analysis.

In [28]:
```
# Combine features and target variable into a single dataframe for formula-based modeling
Xy_pre_train = X_pre_train.copy()
Xy_pre_train['TARGET'] = y_pre_train

# Start timer to calculate the running time of training the logistic regression model
start_time = time.time()

#Define the logistic regression model using statsmodels' formula API and fit it to the training data
# LR1 = smf.logit(formula="TARGET ~  C(discharge_disposition_id) + C(admission_type_id ) + number_inpatient + num_medications + time_in_hospi
LR1 = smf.logit(formula="TARGET ~  C(discharge_disposition_id) + number_inpatient + C(admission_source_id ) + num_lab_procedures + C(admissio
# Calculate and print the running time in seconds
elapsed_time_LR1 = time.time() - start_time
print(f"Elapsed time (sec): {elapsed_time_LR1:.2f}")

# Display a summary of the logistic regression model results
print(LR1.summary())
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.367773
         Iterations: 100
Elapsed time (sec): 4.02
                           Logit Regression Results
==============================================================================
Dep. Variable:                 TARGET   No. Observations:                33425
Model:                          Logit   Df Residuals:                    33392
Method:                           MLE   Df Model:                           32
Date:                Mon, 28 Jul 2025   Pseudo R-squ.:                0.05466
Time:                        08:22:06   Log-Likelihood:                -12293.
converged:                      False   LL-Null:                       -13004.
Covariance Type:            nonrobust   LLR p-value:                9.991e-279
=============================================================================================
                                      coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------------
Intercept                          -2.3953      0.075    -31.915      0.000      -2.542      -2.248
C(discharge_disposition_id)[T.2]    0.6584      0.096      6.878      0.000       0.471       0.846
C(discharge_disposition_id)[T.3]    0.6928      0.047     14.721      0.000       0.601       0.785
C(discharge_disposition_id)[T.4]    0.6174      0.170      3.634      0.000       0.284       0.950
C(discharge_disposition_id)[T.5]    1.2349      0.110     11.215      0.000       1.019       1.451
C(discharge_disposition_id)[T.6]    0.3661      0.053      6.854      0.000       0.261       0.471
C(discharge_disposition_id)[T.7]    0.0611      0.225      0.271      0.786      -0.380       0.502
C(discharge_disposition_id)[T.8]    0.4333      0.487      0.890      0.373      -0.521       1.388
C(discharge_disposition_id)[T.15]   2.8934      0.499      5.800      0.000       1.916       3.871
C(discharge_disposition_id)[T.18]   0.3374      0.087      3.869      0.000       0.166       0.508
C(discharge_disposition_id)[T.22]   1.5649      0.080     19.558      0.000       1.408       1.722
C(discharge_disposition_id)[T.23]  -1.0820      0.459     -2.359      0.018      -1.981      -0.183
C(discharge_disposition_id)[T.24]  -0.4830      1.038     -0.465      0.642      -2.517       1.551
C(discharge_disposition_id)[T.25]  -0.2688      0.193     -1.392      0.164      -0.647       0.110
C(discharge_disposition_id)[T.28]   1.9307      0.288      6.699      0.000       1.366       2.496
C(admission_source_id)[T.2]        -0.2535      0.159     -1.590      0.112      -0.566       0.059
C(admission_source_id)[T.3]         0.6065      0.301      2.018      0.044       0.017       1.196
C(admission_source_id)[T.4]        -0.3454      0.096     -3.601      0.000      -0.533      -0.157
C(admission_source_id)[T.5]        -0.5875      0.207     -2.840      0.005      -0.993      -0.182
C(admission_source_id)[T.6]        -0.3602      0.123     -2.922      0.003      -0.602      -0.119
C(admission_source_id)[T.7]        -0.0090      0.061     -0.147      0.883      -0.129       0.111
C(admission_source_id)[T.8]        -0.3973      1.090     -0.364      0.716      -2.535       1.740
C(admission_source_id)[T.9]        -0.5037      0.473     -1.066      0.287      -1.430       0.423
C(admission_source_id)[T.17]       -0.2122      0.099     -2.154      0.031      -0.405      -0.019
C(admission_source_id)[T.20]        0.7764      0.307      2.525      0.012       0.174       1.379
C(admission_type_id)[T.2]           0.0498      0.060      0.836      0.403      -0.067       0.167
C(admission_type_id)[T.3]          -0.1179      0.070     -1.675      0.094      -0.256       0.020
C(admission_type_id)[T.5]           0.0928      0.107      0.863      0.388      -0.118       0.303
C(admission_type_id)[T.6]           0.3578      0.092      3.887      0.000       0.177       0.538
C(admission_type_id)[T.7]         -24.3872    1.2e+05     -0.000      1.000   -2.35e+05    2.35e+05
C(admission_type_id)[T.8]          -0.3404      0.289     -1.178      0.239      -0.907       0.226
number_inpatient                    0.5897      0.026     23.034      0.000       0.540       0.640
num_lab_procedures                  0.0037      0.001      4.082      0.000       0.002       0.006
=============================================================================================
C:\Users\danie\Projekte\PK_VADS_IMMERSION\2025\.venv3\lib\site-packages\statsmodels\base\model.py:607: ConvergenceWarning: Maximum Likelihood
optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

Five variables were selected from the feature importance list. When specifying the features (following the R style), it is important to note that categorical variables should be indicated with the prefix `C`.

Since our logistic regression model is now trained, we evaluate its performance on the validation set by calculating the AUC, which provides us with a single metric summarizing the model's ability to distinguish between the positive and negative classes.

In [29]:
```
# Evaluate the performance of the logistic regression model on the validation data
# and calculate the Area Under the Curve (AUC) for the ROC
auc_LR1 = roc_auc_score(y_pre_val, LR1.predict(X_pre_val))

# Print the validation AUC
print(f'The validation AUC of the logistic regression model is: {auc_LR1:.6f}')
```

```
The validation AUC of the logistic regression model is: 0.661211
```

2.3 Data Imputation Impact: Re-evaluation of CatBoost with Median Imputation

Since median imputation has not taken place, this section is not considered and is commented out.

In [30]:
```
# # Start timer to calculate the running time of training the CatBoost model with median replacement
# start_time = time.time()

# # Initialize a CatBoostClassifier and fit it to the preprocessed training data
# CB2 = CatBoostClassifier(eval_metric='AUC', random_seed=RANDOM_SEED)
# CB2.fit(X_pre_train, y_train, cat_features=categorical_features, logging_level='Silent')

# # Calculate and print the running time in seconds
# elapsed_time_CB2 = time.time() - start_time
# print(f"Elapsed time (sec): {elapsed_time_CB2:.2f}")
```

In [31]:
```
# # Evaluate the performance of the CatBoost model with median replacement on the
# # validation data and calculate the Area Under the Curve (AUC) for the ROC
# auc_CB2 = roc_auc_score(y_val, CB2.predict_proba(X_pre_val)[:,1])
# print(f'The validation AUC of the CatBoost model with median replacement is:   {auc_CB2:.6f}')
# print(f'The validation AUC of the CatBoost model with minimum replacement was: {auc_CB1:.6f}')
```

3. Exploratory Data Analysis and Feature Engineering

In Section 3, we conduct a targeted exploratory data analysis (EDA), where we analyze the most important categorical and numerical variables of the dataset, and then explore feature engineering to develop new attributes that have the potential to improve the predictive capabilities of our models.

In doing so, we will focus on a selection of key numerical features identified in Subsection 1.5, as well as categorical features of particular interest.

3.1 Analysis of Categorical Features

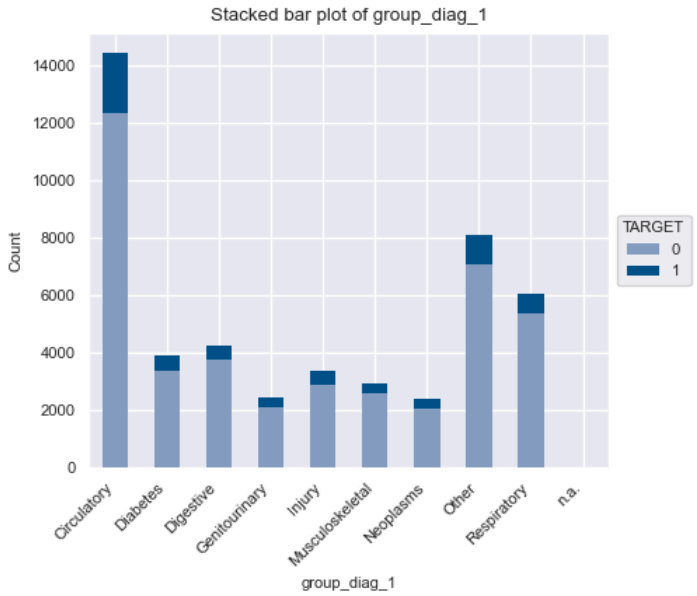
First, we examine seven selected categorical variables - `group_diag_1`, `group_diag_2`, `group_diag_3`, `age`, `medical_specialty`, `payer_code`, and `discharge_disposition_id` - to determine their influence on readmissions. Our analysis includes:

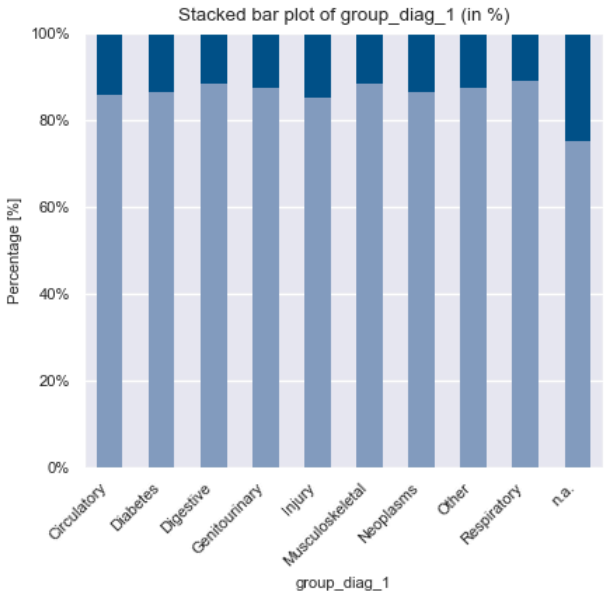
1. Creating stacked bar charts to compare the number of individuals and the percentage distribution across the various categories with respect to the target outcome.
2. Comparing the raw number of readmissions with the proportion within each category to uncover potential predictive patterns.

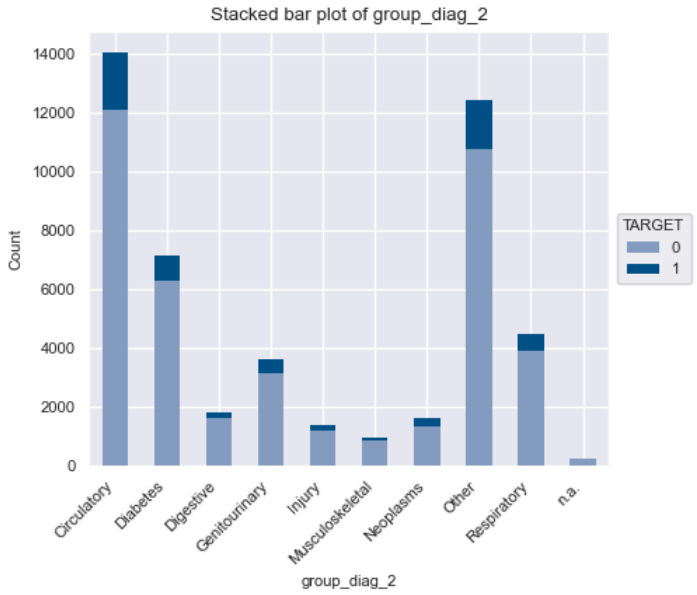
By examining both the counts and percentage distributions side by side, we can better understand how each categorical feature correlates with the likelihood of a person experiencing a readmission.

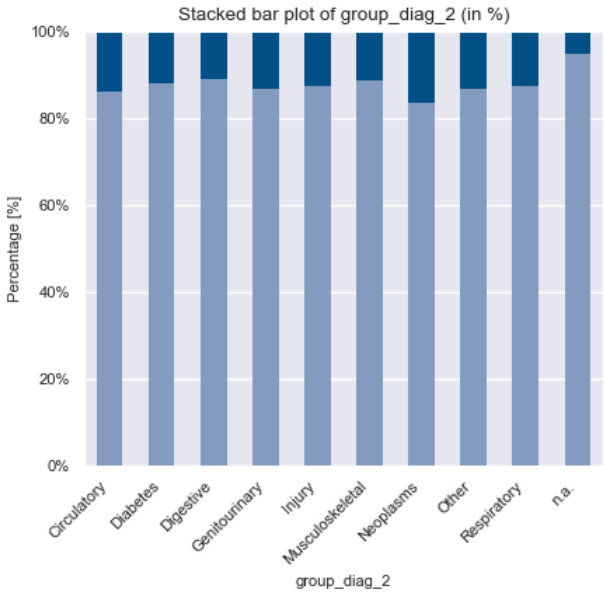
In [32]:
```
# will be needed in Task PT4d

# List of selected categorical features to analyze
selected_categorical_features = ['group_diag_1', 'group_diag_2', 'group_diag_3', 'age', 'discharge_disposition_id', 'medical_specialty', 'pay

# Number of rows/columns for the subplot grid
n_cols = 2  # doubled to fit count and percentage plots side by side
n_rows = len(selected_categorical_features)  # one row for each feature

# Set up the matplotlib figure
plt.figure(figsize=(n_cols * 5, n_rows * 5))

# Correct font size
plt.rcParams.update({'font.size': 8})

# Loop through the number of categorical features
for idx, feature in enumerate(selected_categorical_features):
    # Create a crosstab for stacked bar plot structure
    ctab = pd.crosstab(df_pre[feature], df_pre['TARGET'])

    # (1st column) add a new subplot iteratively for count values
    ax1 = plt.subplot(n_rows, n_cols, idx * n_cols + 1)

    # Create a stacked bar plot for count values
    ctab.plot(kind="bar", stacked=True, color=[COLOR_LIGHT, COLOR_DARK], edgecolor="none", ax=ax1)

    # Additional plot settings for count subplot
    ax1.set_title(f'Stacked bar plot of {feature}')
    ax1.set_xlabel(feature)
    ax1.set_ylabel('Count')
    ax1.legend(title='TARGET', loc='center left', bbox_to_anchor=(1, 0.5))
    plt.xticks(rotation=45, ha='right')

    # (2nd column) add a new subplot iteratively for percentage values
```

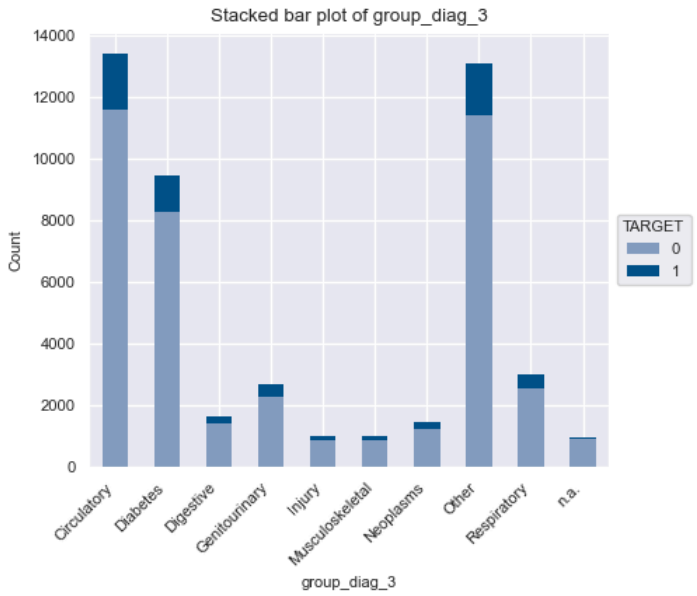
```
    ax2 = plt.subplot(n_rows, n_cols, idx * n_cols + 2)

    # Normalize the crosstab by row and multiply by 100 to convert to percentages
    ctab_normalized = ctab.div(ctab.sum(axis=1), axis=0) * 100

    # Create a stacked bar plot for percentage values
    ctab_normalized.plot(kind="bar", stacked=True, color=[COLOR_LIGHT, COLOR_DARK], edgecolor="none", ax=ax2)

    # Additional plot settings for percentage subplot
    ax2.set_title(f'Stacked bar plot of {feature} (in %)')
    ax2.set_xlabel(feature)
    ax2.set_ylabel('Percentage [%]')
    ax2.legend().remove()
    plt.xticks(rotation=45, ha='right')

    # Make yticks be in percentages for the percentage subplot
    ax2.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, loc: "{:.0f}%".format(int(x))))
    ax2.set_ylim(0, 100)

# Preventing subplots from being too close to each other and display the final plot
plt.tight_layout()
plt.show()
```

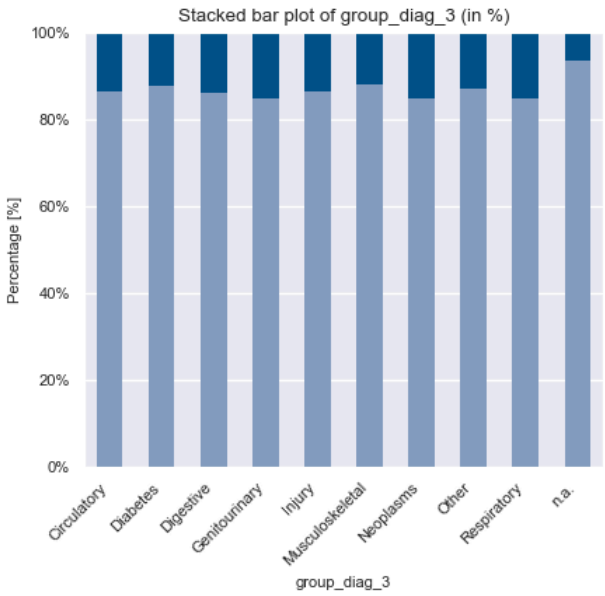

Stacked bar plot of group_diag_1
Count
TARGET
0
1
Circulatory
Diabetes
Digestive
Genitourinary
Injury
Musculoskeletal
Neoplasms
Other
Respiratory
n.a.
group_diag_1


Stacked bar plot of group_diag_1 (in %)
Percentage [%]
Circulatory
Diabetes
Digestive
Genitourinary
Injury
Musculoskeletal
Neoplasms
Other
Respiratory
n.a.
group_diag_1


Stacked bar plot of group_diag_2
Count
TARGET
0
1
Circulatory
Diabetes
Digestive
Genitourinary
Injury
Musculoskeletal
Neoplasms
Other
Respiratory
n.a.
group_diag_2

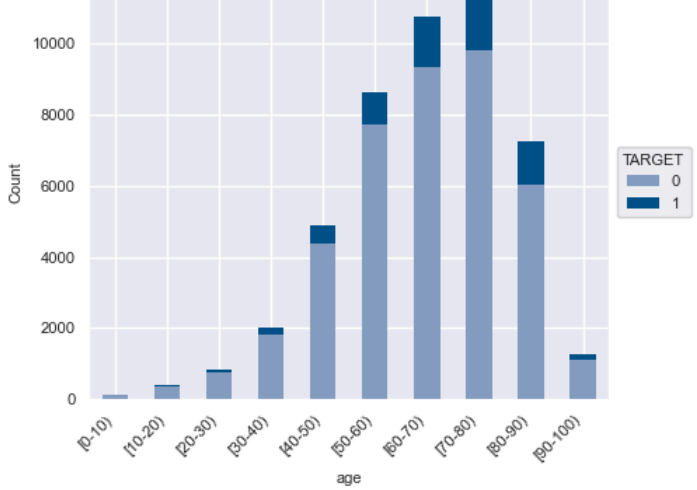

Stacked bar plot of group_diag_2 (in %)
Percentage [%]
Circulatory
Diabetes
Digestive
Genitourinary
Injury
Musculoskeletal
Neoplasms
Other
Respiratory
n.a.
group_diag_2

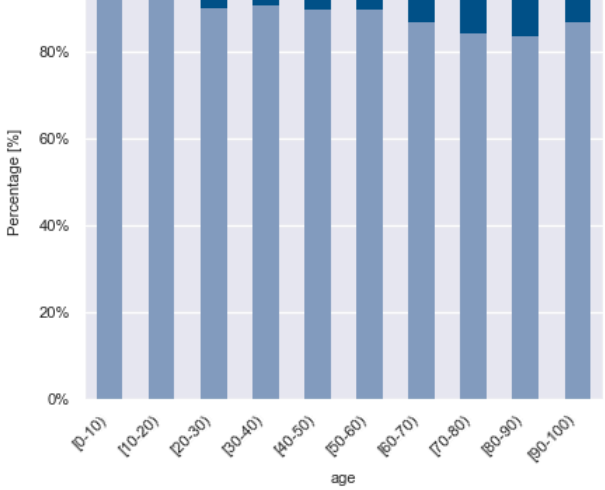

Stacked bar plot of group_diag_3
Count
TARGET
0
1
Circulatory
Diabetes
Digestive
Genitourinary
Injury
Musculoskeletal
Neoplasms
Other
Respiratory
n.a.
group_diag_3

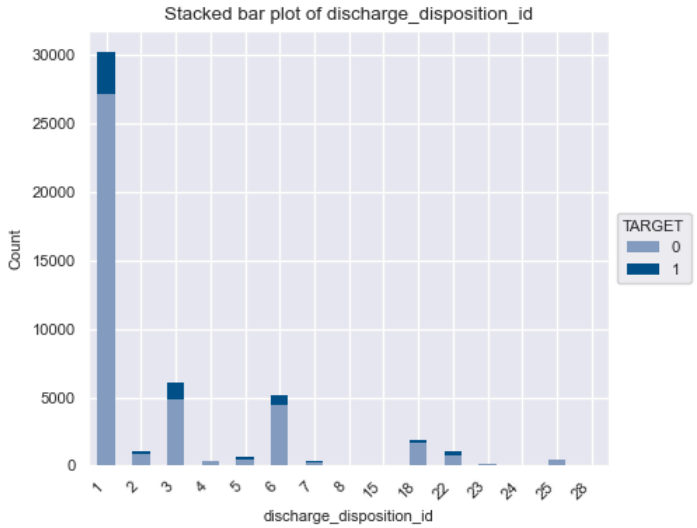

Stacked bar plot of group_diag_3 (in %)
Percentage [%]
Circulatory
Diabetes
Digestive
Genitourinary
Injury
Musculoskeletal
Neoplasms
Other
Respiratory
n.a.
group_diag_3

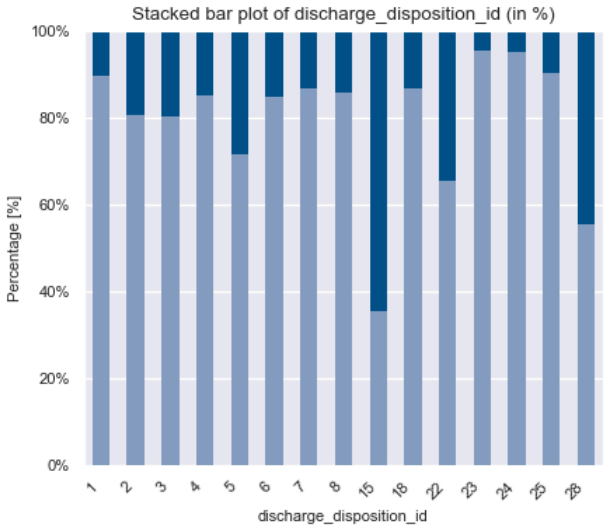

Stacked bar plot of age

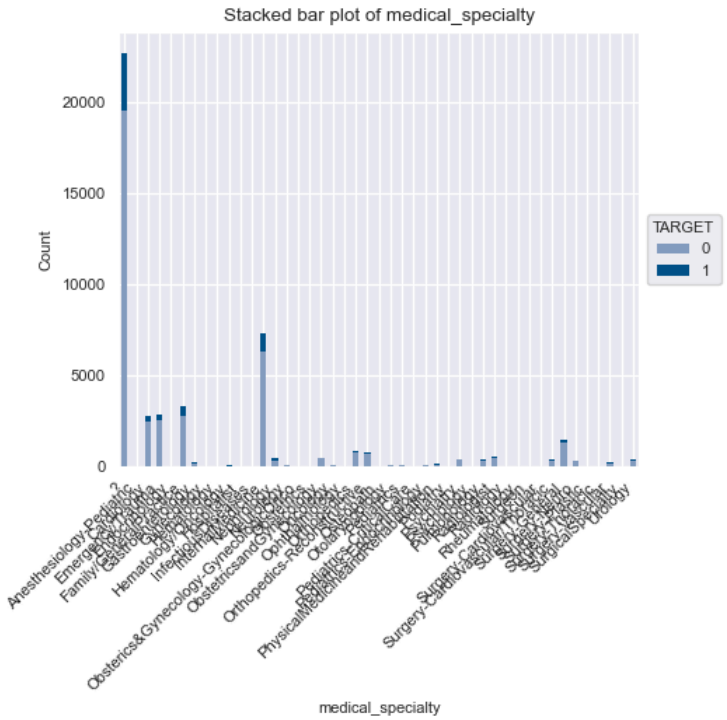

Stacked bar plot of age (in %)

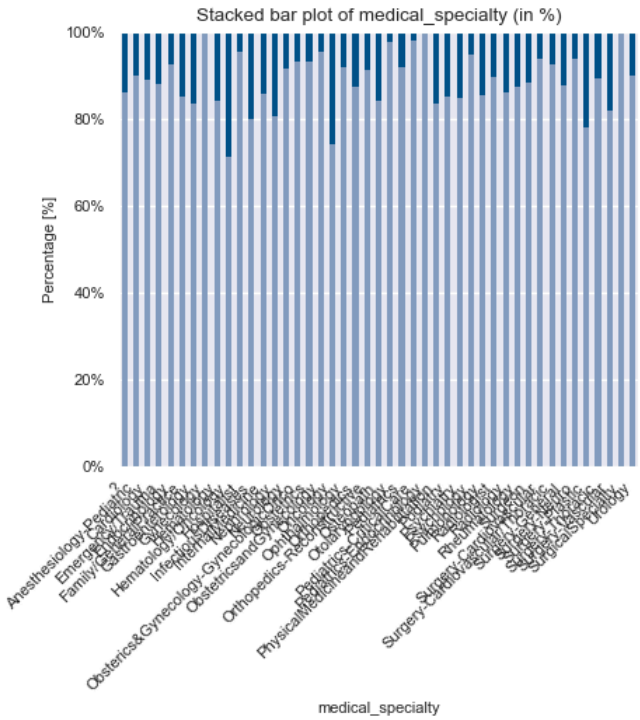

Count
age
TARGET
0
1
[0-10)
[10-20)
[20-30)
[30-40)
[40-50)
[50-60)
[60-70)
[70-80)
[80-90)
[90-100)

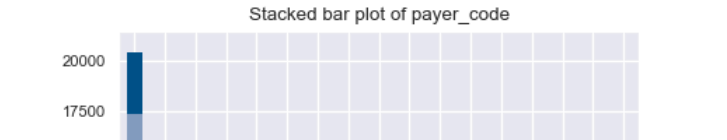

Percentage [%]
age
[0-10)
[10-20)
[20-30)
[30-40)
[40-50)
[50-60)
[60-70)
[70-80)
[80-90)
[90-100)

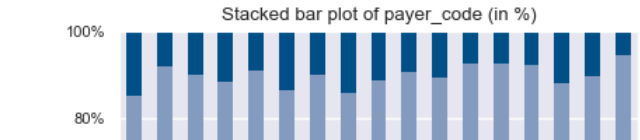

Stacked bar plot of discharge_disposition_id
Count
discharge_disposition_id
TARGET
0
1

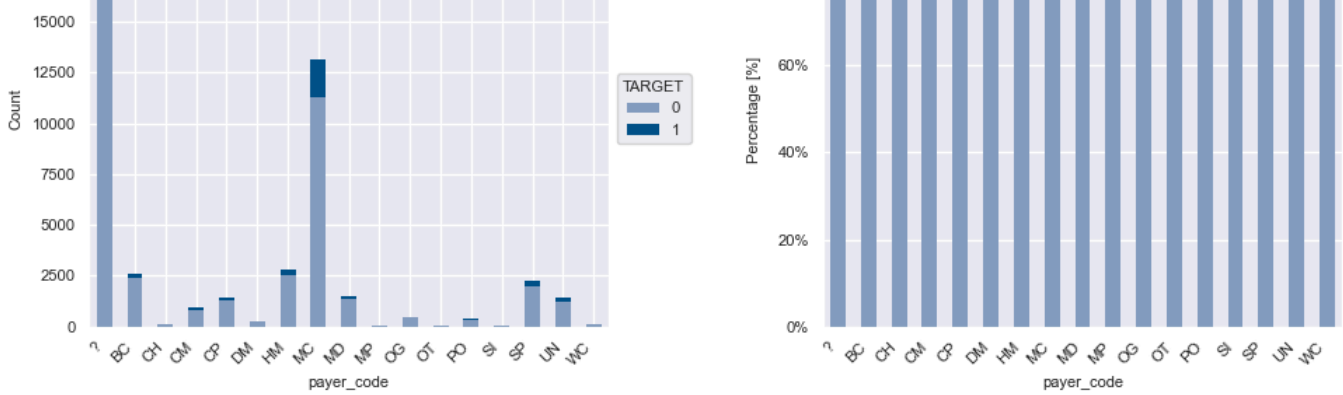

Stacked bar plot of discharge_disposition_id (in %)
Percentage [%]
discharge_disposition_id


Stacked bar plot of medical_specialty
Count
medical_specialty
TARGET
0
1


Stacked bar plot of medical_specialty (in %)
Percentage [%]
medical_specialty


Stacked bar plot of payer_code


Stacked bar plot of payer_code (in %)

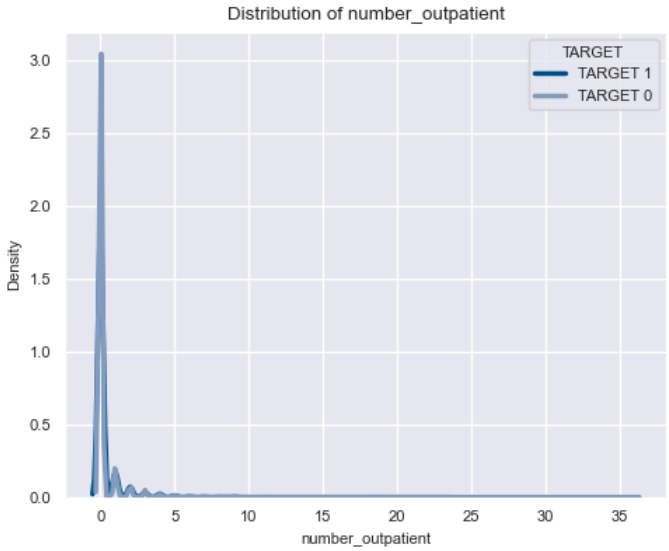


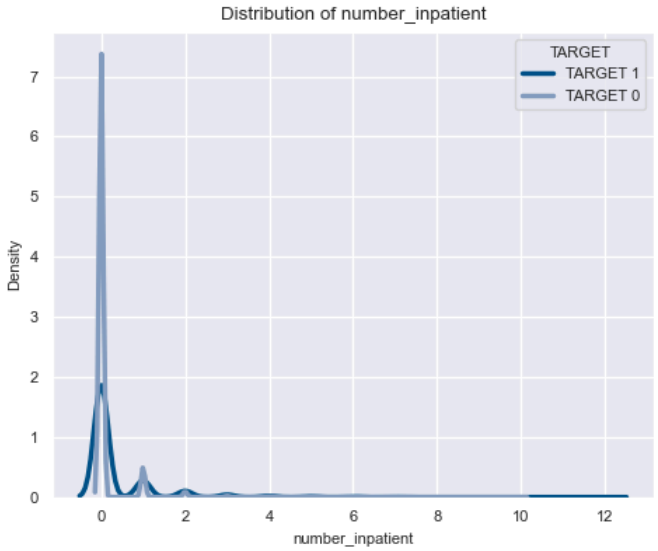
The corresponding interpretations have already been carried out in the R notebook and will not be provided here again.

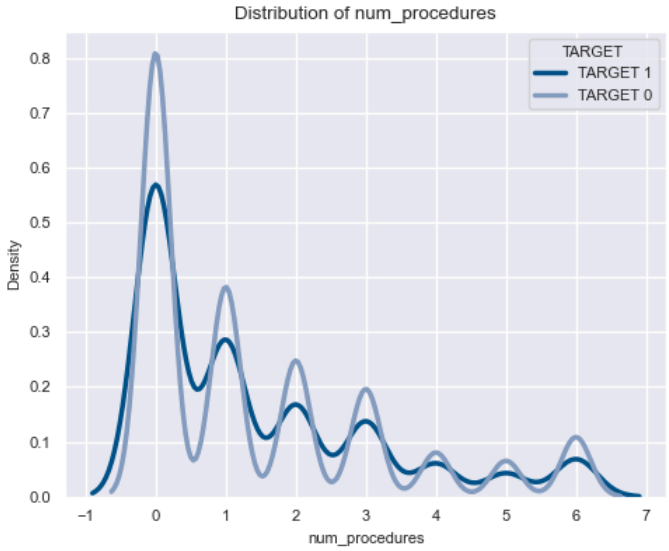
3.2 Analysis of Numerical Features

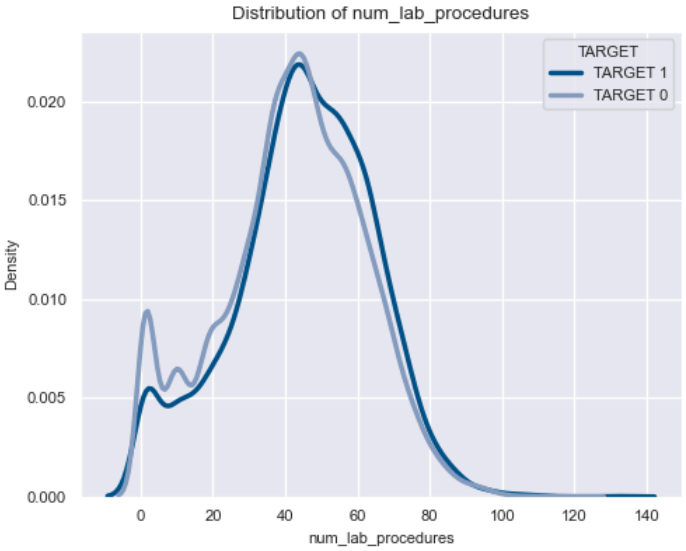
Next, we analyze key numerical variables to identify patterns related to readmissions. Using Kernel Density Estimation (KDE) plots, we observe the density distribution of selected numerical features. By comparing these distributions, we can determine which numerical predictors might have significant predictive power — a crucial step for effective feature engineering and subsequent model development in binary classification tasks.

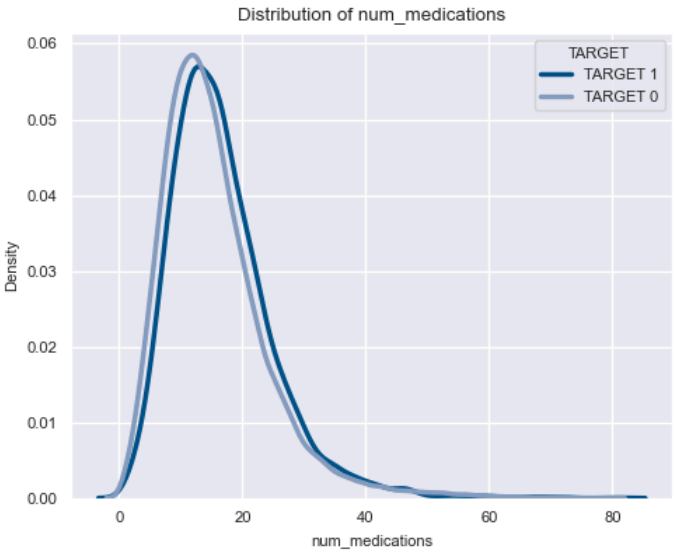
In [33]:
```
# Will be needed in task PT4d.

# List of selected numerical features to analyze
selected_numerical_features = ['number_outpatient', 'number_inpatient', 'num_procedures',
                               'num_lab_procedures','num_medications', 'time_in_hospital',
                               'number_emergency', 'number_diagnoses']

# number of plots, set up the subplot grid
n_plots = len(selected_numerical_features)
n_cols = 2  # adjust the number of columns based on your preference
n_rows = (n_plots + n_cols - 1) // n_cols

# set up the matplotlib figure
plt.figure(figsize=(n_cols * 5, n_rows * 4))

# loop through the list of numerical features
for idx, feature in enumerate(selected_numerical_features):
    ax = plt.subplot(n_rows, n_cols, idx + 1)

    # Create a color mapping based on TARGET values assuming values are 0 and 1
    color_mapping = {0: COLOR_LIGHT, 1: COLOR_DARK}

    class_vals = df_pre['TARGET'].unique()
    for val in class_vals:
        subset = df_pre[df_pre['TARGET'] == val]

        # Draw the KDE for each class
        sns.kdeplot(subset[feature], ax=ax, label=f'TARGET {val}', color=color_mapping[val], linewidth=2.5)

    # additional plot settings
    ax.set_title(f'Distribution of {feature}')
    ax.set_xlabel(f'{feature}')
    ax.set_ylabel('Density')
    ax.legend(title='TARGET')

# Preventing subplots from being too close to each other and display the final plot
plt.tight_layout()
plt.show()
```

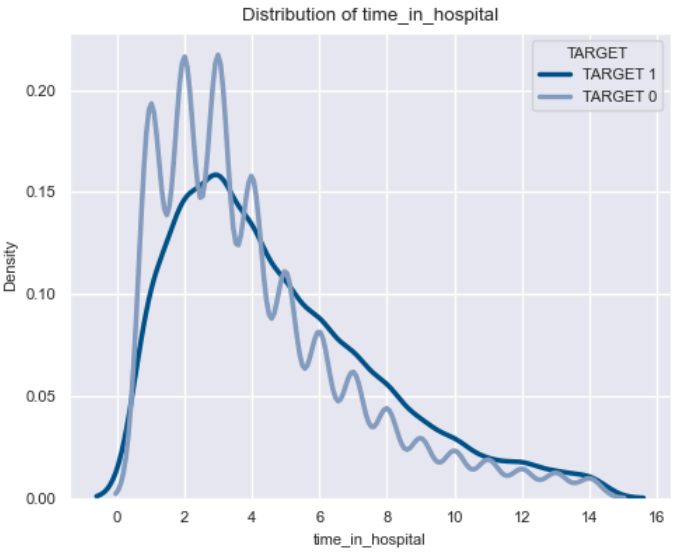

Distribution of number_outpatient
TARGET
TARGET 1
TARGET 0
Density
number_outpatient

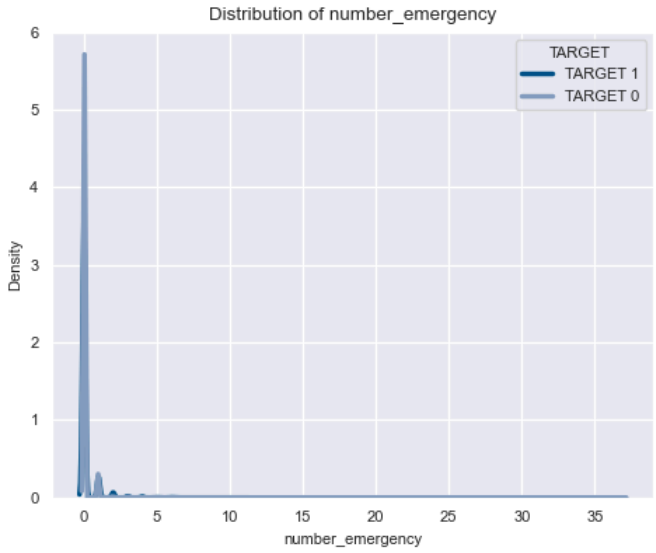

Distribution of number_inpatient
TARGET
TARGET 1
TARGET 0
Density
number_inpatient

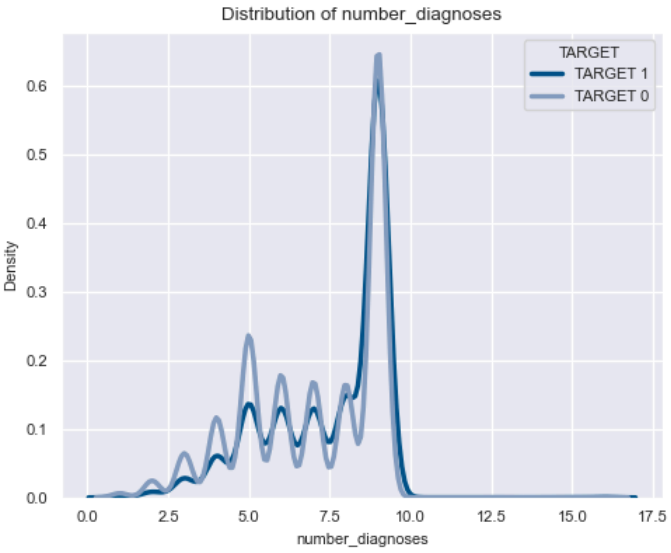

Distribution of num_procedures
TARGET
TARGET 1
TARGET 0
Density
num_procedures


Distribution of num_lab_procedures
TARGET
TARGET 1
TARGET 0
Density
num_lab_procedures


Distribution of num_medications
TARGET
TARGET 1
TARGET 0
Density
num_medications


Distribution of time_in_hospital
TARGET
TARGET 1
TARGET 0
Density
time_in_hospital


Distribution of number_emergency
TARGET
TARGET 1
TARGET 0
Density
number_emergency


Distribution of number_diagnoses
TARGET
TARGET 1
TARGET 0
Density
number_diagnoses

Corresponding interpretations have already been carried out in the R notebook and will not be provided again here.

3.3 Feature Engineering: Creating New Features (not considered)

4. Improving the Performance of CatBoost with New Features (not considered)

5. Interpreting Models for Better Understanding (not considered)

6. Data Preprocessing: Encoding, Scaling, and Subsampling

In this section, we focus on two essential preprocessing steps: encoding, to make categorical data machine-readable, which is crucial for many machine learning algorithms, and scaling, to normalize the values of numerical features, which supports the efficiency of the algorithms. Additionally, we address class imbalance by using subsampling methods, creating a more balanced training dataset and improving the computational efficiency of our machine learning models.

6.1 Encoding Categorical Data and Scaling Numerical Values

In general, machine learning models require input data in numerical format to perform mathematical calculations. However, real-world datasets – like the one considered here – often contain categorical or text data. Therefore, encoding is necessary to transform this non-numerical data into a numerical form that can be understood and processed by machine learning algorithms.

The two most commonly used encoding methods are as follows:

- Label Encoding: Converts each value in a column into a number. This is suitable for ordinal data, i.e., when categories have an inherent order.
- One-Hot Encoding: Creates a binary column for each category/label in the column. This is useful for nominal variables, where the categories have no specific order or priority.

In the following, we will apply One-Hot Encoding to our dataset to prepare it for machine learning methods such as Artificial Neural Networks or XGBoost.

In [34]:
```
# Output the dimensions of the dataset prior to dummy encoding
print("Shape of the dataset before dummy encoding: ", df_pre.shape)

# Perform dummy encoding on categorical features, removing the first level to avoid dummy variable trap
df = pd.get_dummies(df_pre, drop_first=True)

# Output the dimensions of the dataset after dummy encoding to show the change
print("Shape of the dataset after dummy encoding: ", df.shape)
```

```
Shape of the dataset before dummy encoding:  (47751, 51)
Shape of the dataset after dummy encoding:  (47751, 2249)
```

In [35]:
```
# Change columns names (LightGBM does not support special JSON characters in feature names)
# Needed to adjust age groups, for example. Example: 'age_[50-60)': 'age_5060'
new_names = {col: re.sub(r'[^A-Za-z0-9_]+', '', col) for col in df.columns}
new_n_list = list(new_names.values())
new_names = {col: f'{new_col}_{i}' if new_col in new_n_list[:i] else new_col for i, (col, new_col) in enumerate(new_names.items())}
df = df.rename(columns=new_names)
df.head(1)
```

Out[35]:

	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpatient	number_emergency	number_inpatient	number_diagnoses
0	8	77	6	33	0	0	0	8

In [36]:
```
# Generate the samples once again

# Split dataset in 70% training, 15% validation and 15% test and divide into feature matrix x and label y
X = df.drop(['TARGET'], axis=1)
y = df.TARGET
X_train, X_valtest, y_train, y_valtest = train_test_split(X, y, train_size=0.70, random_state=RANDOM_SEED)
X_val, X_test, y_val, y_test = train_test_split(X_valtest, y_valtest, test_size=0.50, random_state=RANDOM_SEED)
print('Sample sizes for training, validation and test: ', X_train.shape, X_val.shape, X_test.shape)
```

```
Sample sizes for training, validation and test:  (33425, 2248) (7163, 2248) (7163, 2248)
```

Scaling is crucial in machine learning for consistency, improved performance, and meeting algorithm requirements:

1. Uniformity: Different features can have different scales, such as age ranging from 0 to 100 and income ranging from thousands to millions. This can mislead algorithms into overestimating features with larger ranges.
2. Efficiency: Algorithms like Gradient Descent and K-Nearest Neighbors (KNN) benefit from features on a similar scale, leading to better performance and faster convergence. For example, KNN is highly influenced by features with larger values.
3. Normalization Requirements: Certain machine learning algorithms, such as neural networks, require input scaling to ensure effective model training.

Typical scaling techniques include Min-Max Scaling, which adjusts features to a range of 0 to 1, and Standard Scaling, which scales features to a mean of zero and unit variance. The appropriate scaling approach depends on the algorithm used and the data characteristics. In our analysis, we will use Standard Scaling.

In [37]:
```
# Generate list: feature names
feature_names=list(X.columns)
```

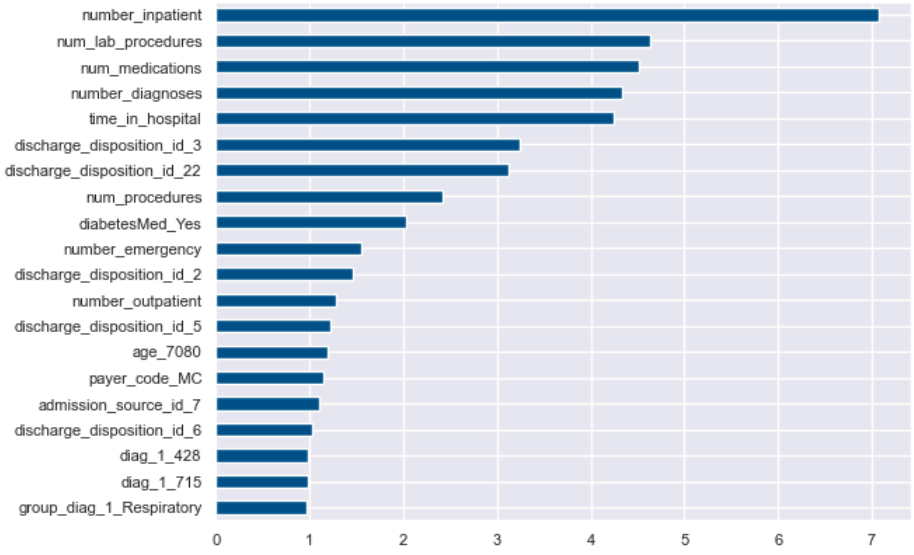
```
# Feature scaling using StandardScaler based on training data distributions
scaler = StandardScaler()
scaler.fit(X_train)
X_train = pd.DataFrame(scaler.transform(X_train), columns = feature_names)
X_val = pd.DataFrame(scaler.transform(X_val), columns = feature_names)

# scaling (previously part of PT7)
X_test = pd.DataFrame(scaler.transform(X_test), columns = feature_names)
```

6.2 Do Encoding and Scaling Affect the Benchmark Model?

In [38]:
```
# Repeat CB with one-hot-encoded and scaled data
start_time = time.time()

CB4 = CatBoostClassifier(eval_metric='AUC', random_seed=RANDOM_SEED)
CB4.fit(X_train, y_train, logging_level='Silent')

# Calculate and print the running time in seconds
elapsed_time_CB4 = time.time() - start_time
print(f"Elapsed time (sec): {elapsed_time_CB4:.2f}")
```

```
Elapsed time (sec): 27.46
```

In [39]:
```
# CatBoost: Plot most the top 20 most important features
pd.Series(CB4.feature_importances_, index=X_train.columns).nlargest(20).plot(kind='barh', color=COLOR_DARK).invert_yaxis()
plt.show()
```



In comparison to the Feature Importance List of the "quick" model, the feature `discharge_disposition_id` (the most important feature in the "quick" list) is represented in the list with several variations. The feature `number_inpatient` remains important. Categorical features such as age are only represented in the list with the relevant classes (in this case, the class `7080`). It should be noted that there are significantly more features in the dataset being considered (an increase from approximately 50 features to approximately 2250 features, i.e., a factor of 45!). Changes in the ranking can be explained, among other things, by this.

In [40]:
```
# Calculate and dispay the model AUC
auc_CB4 = roc_auc_score(y_val, CB4.predict_proba(X_val)[:,1])
print('The validation AUC of the CatBoost model after encoding and scaling is:   {:.6f}'.format(auc_CB4))
#print('The validation AUC of the CatBoost model before encoding and scaling was: {:.6f}'.format(auc_CB3))
```

```
The validation AUC of the CatBoost model after encoding and scaling is:   0.702311
```

In [41]:
```
print(f"Elapsed time of the CatBoost model after encoding and scaling (sec):  {elapsed_time_CB4:.2f}")
#print(f"Elapsed time of the CatBoost model before encoding and scaling (sec): {elapsed_time_CB3:.2f}")
```

```
Elapsed time of the CatBoost model after encoding and scaling (sec):  27.46
```

Summary and Interpretation:

The encoding of categorical data has significantly increased the number of features (as previously mentioned). The AUC value has only slightly decreased (from approximately 0.707 to 0.702). The runtime is significantly lower (from about 57 to 7 seconds). Apparently, the direct use of categorical features within CatBoost leads to a greatly increased computational effort, which, however, results in only minimal improvement in prediction quality.

6.3 Subsampling of Training Data

Similar to the analysis in CSN, readmissions are relatively rare events, meaning that the dataset is imbalanced with regard to `TARGET`. On average, for every readmission, there are approximately 6.5 cases without a readmission in the dataset. Similar to CSN, we examine here whether the imbalance can be reduced to a ratio of 1:3 without degrading the resulting model quality and whether the reduced data volume can accelerate model training. Therefore, the `non_event_factor` is adjusted to 2.

In [42]:
```
non_event_factor = 2 # thus the event rate should be 26% (instead of 8%)

# Save complete training data
X_train_all = X_train.copy(deep=True)
y_train_all = y_train.copy(deep=True)
```

In [43]:
```
# Subsampling: enhance event rate to 1:nov_event_factor
Xs = pd.concat([X_train.reset_index(drop=True), y_train.reset_index(drop=True)], axis=1)
Xs = pd.concat([Xs[Xs['TARGET']==1], Xs[Xs['TARGET']==0].sample(
           n = non_event_factor * len(Xs[Xs['TARGET']==1] ), random_state=RANDOM_SEED)],axis=0).sort_index(ascending=True)

# Over-write complete training data with subsample
X_train = Xs.drop(columns=["TARGET"])
y_train = Xs["TARGET"]

print("\nSize of training data set after subsampling: ", X_train.shape)
print("\nTARGET-distribution after subsampling:\n", y_train.value_counts())
print("\nReduction of data volume (%): ", round((len(y_train)/len(y_train_all)-1)*100))
```

```
Size of training data set after subsampling:  (13176, 2248)

TARGET-distribution after subsampling:
 TARGET
0    8784
1    4392
Name: count, dtype: int64

Reduction of data volume (%):  -61
```

6.4 Does Subsampling Affect the Benchmark Model?

In [44]:
```
# Repeat CB with subsampled data
start_time = time.time()

CB5 = CatBoostClassifier(eval_metric='AUC', random_seed=RANDOM_SEED)
CB5.fit(X_train, y_train, logging_level='Silent')

# Calculate and print the running time in seconds
elapsed_time_CB5 = time.time() - start_time
print(f"Elapsed time (sec): {elapsed_time_CB5:.2f}")
```

```
Elapsed time (sec): 22.83
```

In [45]:
```
# Calculate and dispay the model AUC
auc_CB5 = roc_auc_score(y_val, CB5.predict_proba(X_val)[:,1])
print('The validation AUC of the CatBoost model with subsampled data is: {:.6f}'.format(auc_CB5))
```

```
The validation AUC of the CatBoost model with subsampled data is: 0.699061
```

Summary and Conclusion:

The smaller amount of training data further reduces CatBoost's training time. However, the model performance is not improved. Thus, subsampling of training data can be a key component for accelerating model training and, in particular, hyperparameter tuning. In the following, we focus on this subsampled dataset.

6.5 Model Evaluation and Comparison (new section based on Section 4.2 of the CSN notebook)

Prepare model comparison:

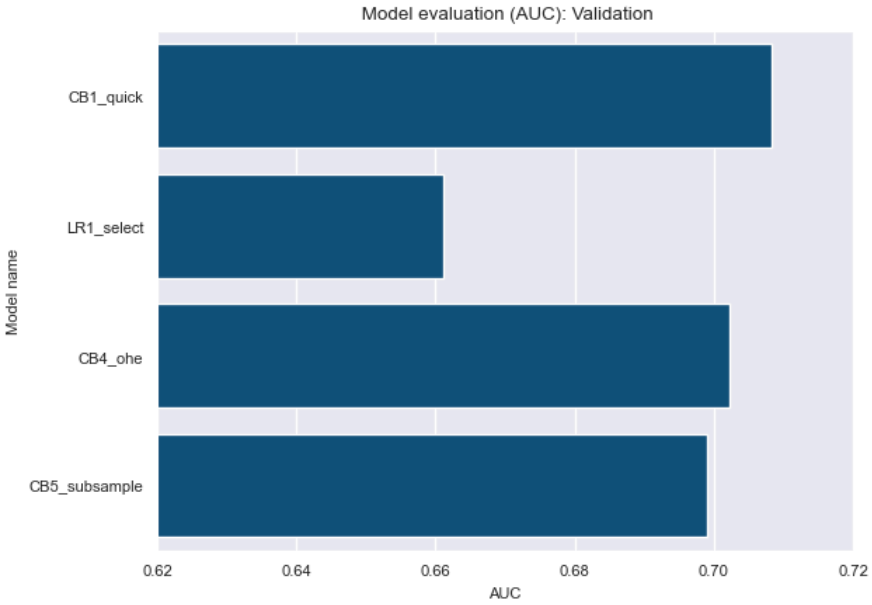
In [46]:
```
# Data structures for model names and AUC
mname = []
mauc = []
dict = {'Model name': mname,'AUC': mauc, }
```

In [47]:
```
# Store name and AUC of prevoisly fitted models
mname.append("CB1_quick")
mauc.append(auc_CB1)
mname.append("LR1_select")
mauc.append(auc_LR1)
mname.append("CB4_ohe")
mauc.append(auc_CB4)
mname.append("CB5_subsample")
mauc.append(auc_CB5)
```

In [48]:
```
def plot_auc(d,x1,x2,t):
    df_eval = pd.DataFrame(d)
    #sns.set_style('darkgrid')
    plt.title(" Model evaluation (AUC): " + t)
    sns.barplot(data = df_eval, x = "AUC", y = "Model name", color = COLOR_DARK)
    plt.xlim(x1, x2)
    plt.show()
```

Compare the model performance (AUC - higher is better):

In [49]:
```
plot_auc(dict,0.62,0.72, "Validation")
```



The first CatBoost model (CB1_quick) is the best model in terms of AUC in the comparison, while logistic regression performs significantly worse. As previously mentioned, scaling and subsampling accelerate the training of the models, but the model quality does not improve as a result.

Task PT3: Adjust and Extend Model Optimization [Learning Goal 5.1; 3 points]

- a) Sections 7 and 8 are not to be processed and should be removed.
- b) Sections 9 and 10 are to be processed and adjusted. Make appropriate settings according to the hardware used (CPU/GPU). What changes need to be made to the HP grids? Interpret the results.
- c) Section 11 is to be skipped initially.

Proposed Solution:

Part C: Tuning and Applying Machine Learning Models

Part C focuses on utilizing the full potential of our models by fine-tuning the hyperparameters for regularized logistic regression, artificial neural networks, and the gradient tree boosting models CatBoost, LightGBM, and XGBoost. We conduct a detailed comparative evaluation using validation and test datasets and examine the application of these models in high-risk areas. Finally, we summarize the key insights gained from our exploration of machine learning. In conclusion, we reflect on the most important findings from our journey into machine learning.

7. Regularized Logistic Regression to Avoid Overfitting (not considered)

8. Optimizing a Feed-Forward Neural Network (not considered)

9. Hyperparameter Optimization in CatBoost

Now we want to investigate whether we can optimize the learning rate and tree depth of CatBoost by performing an expensive cross-validated randomized search (fitting 40 models).

In the first code block, we set up the conditions for optimizing the hyperparameters of the CatBoostClassifier model. We define a parameter grid that specifies the range of values to be explored for the learning rate and tree depth. Here, "loguniform" is used for the learning rate to select values from a logarithmic distribution and optimize the search across different scales. Similarly, a list of integers represents possible values for the tree depth.

In [50]:
```
# Define the hyperparameters grid to search
param_grid_CB ={'learning_rate': loguniform(0.015, 0.06), # loguniform(0.015, 0.06),
                'depth':         [5,6,7,8,9]}

# Define variables for results reports
sel_params_CB = ['param_learning_rate','param_depth','mean_test_score','rank_test_score']
```

After the initial setup, our next code block implements the actual hyperparameter optimization using RandomizedSearchCV, a strategy that randomly samples the parameter space and performs a four-fold cross-validation. We select a reasonable number of iterations (`n_iter=10`) to balance computation time and the thoroughness of the search. The results are recorded, including the runtime and the best-found parameters.

In [51]:
```
# CatBoostClassifier: Hyper parameter optimization with RandomizedSearchCV
tic = time.time()

CB_rs = RandomizedSearchCV(CatBoostClassifier(iterations=1000, eval_metric='AUC'), param_grid_CB,
                           cv=4, n_iter=10, random_state=RANDOM_SEED, scoring='roc_auc') # n_jobs=-1,

CB_rs.fit(X_train, y_train, logging_level='Silent')
```

```
# Print the runtime as well as the five best hyper parameter constellations
print("time (sec):" + "%6.0f" % (time.time() - tic))
print("Best hyper parameters:",CB_rs.best_params_)
pd.DataFrame(CB_rs.cv_results_)[sel_params_CB].sort_values("rank_test_score").head()
```

```
time (sec):  1033
Best hyper parameters: {'depth': 7, 'learning_rate': 0.017229876161789517}
```

Out[51]:

	param_learning_rate	param_depth	mean_test_score	rank_test_score
3	0.017230	7	0.693159	1
5	0.018285	8	0.692142	2
8	0.019300	6	0.691830	3
7	0.040811	6	0.691635	4
6	0.015434	7	0.691492	5

After determining the best hyperparameters, we use them to create a new CatBoostClassifier model with the entire training dataset. This approach utilizes the optimized parameters, which should theoretically result in a more accurate model.

In [52]:
```
# CatBoostClassifier: Fit new model on all folds (with best parameters)
tic = time.time()
CB6 = CatBoostClassifier(**CB_rs.best_params_, iterations=1000, eval_metric='AUC')
CB6.fit(X_train, y_train, logging_level='Silent')
print("time (sec):" + "%6.0f" % (time.time() - tic))
```

```
time (sec):    29
```

Finally, we evaluate the performance of the tuned model by calculating the AUC score using the validation dataset.

In [53]:
```
# Calculate and dispay the model AUC
auc_CB6 = roc_auc_score(y_val, CB6.predict_proba(X_val)[:,1])
print('The validation AUC of the CatBoost model with tuned hyperparameters is: {:.6f}'.format(auc_CB6))
print('Comparison: The val. AUC of the CatBoost model with subsampled data is: {:.6f}'.format(auc_CB5))
```

```
The validation AUC of the CatBoost model with tuned hyperparameters is: 0.700055
Comparison: The val. AUC of the CatBoost model with subsampled data is: 0.699061
```

The training of the model takes significantly more time. However, the resulting model is slightly better.

10. HP-Tuning LightGBM and XGBoost

In this section, we focus our attention on the main competitors of CatBoost: LightGBM and XGBoost. By leveraging the power of GPUs to accelerate computations and conducting a more extensive process for hyperparameter tuning, we significantly intensify our efforts in model optimization by doubling the number of evaluated models. This rigorous comparison aims to assess the performance of these leading algorithms under a similar optimization regime.

10.1 LightGBM: HP-Tuning and Evaluation

In [54]:
```
# Parameter search range for LightGBM
param_grid_LGB = {'learning_rate':    loguniform(0.01, 0.05),
                  'num_leaves':       [32,40,45,50,55],
                  'subsample':        uniform(0,1),
                  'colsample_bytree': uniform(0,1),
                  'verbose':          [-1]}

# Create a list of variables for displaying the cross-validation result
sel_params_LGB = ['param_learning_rate', 'param_num_leaves', 'param_subsample',
                  'param_colsample_bytree', 'mean_test_score', 'rank_test_score']
```

In [55]:
```
# LGBMClassifier: Hyperparameter optimization with RandomizedSearchCV
tic = time.time()

LGB_rs = RandomizedSearchCV(LGBMClassifier(device='cpu', n_estimators=1000),
                            param_grid_LGB, cv=4, n_iter=20, scoring='roc_auc',
                            random_state=RANDOM_SEED) # n_jobs=-1,
LGB_rs.fit(X_train, y_train)

# Output the runtime and best parameters (as well as the runner-ups)
print("Elapsed time (sec):" + "%6.0f" % (time.time() - tic))
print("Best Parameters:", LGB_rs.best_params_)
pd.DataFrame(LGB_rs.cv_results_)[sel_params_LGB].sort_values("rank_test_score").head()
```

```
Elapsed time (sec):   229
Best Parameters: {'colsample_bytree': 0.33370861113902184, 'learning_rate': 0.012585185474052117, 'num_leaves': 45, 'subsample': 0.020584499295802447, 'verbose': -1}
```

Out[55]:

	param_learning_rate	param_num_leaves	param_subsample	param_colsample_bytree	mean_test_score	rank_test_score
2	0.012585	45	0.020584	0.333709	0.688414	1
8	0.011104	50	0.942202	0.170524	0.688042	2
4	0.016318	55	0.431945	0.183405	0.684956	3
15	0.018693	40	0.539692	0.325330	0.684851	4
9	0.018595	40	0.097672	0.563288	0.684122	5

In [56]:
```
# LGBMClassifier: Create a new model on all folds using the best parameters
tic = time.time()
LGB = LGBMClassifier(**LGB_rs.best_params_, device='cpu', n_estimators=1000)
LGB.fit(X_train, y_train)
print("Elapsed time (sec):" + "%6.0f" % (time.time() - tic))
```

```
Elapsed time (sec):     4
```

In [57]:
```
# Calculate and dispay the model AUC
auc_LGB = roc_auc_score(y_val, LGB.predict_proba(X_val)[:,1])
print('The validation AUC of the LightGBM model with tuned hyperparameters is: {:.6f}'.format(auc_LGB))
```

```
The validation AUC of the LightGBM model with tuned hyperparameters is: 0.699489
```

10.2 XGBoost: HP-tuning and evaluation

In [58]:
```
# Parameter search range for XGBoost
param_grid_XGB = {'learning_rate':    loguniform(0.01, 0.05),
                  'max_depth':        [5,6,7,8,9],
                  'subsample':        uniform(0,1),
                  'colsample_bytree': uniform(0,1)}

# Create a list of variables for displaying the cross-validation result
sel_params_XGB = ['param_learning_rate', 'param_max_depth', 'param_subsample',
                  'param_colsample_bytree', 'mean_test_score', 'rank_test_score']
```

In [59]:
```
# XGBClassifier: Hyperparameter optimization with RandomizedSearchCV
tic = time.time()

XGB_rs = RandomizedSearchCV(XGBClassifier(n_estimators=1000, tree_method="hist", device="cpu"),
                            param_grid_XGB, cv=4, n_iter=20, scoring="roc_auc",
                            n_jobs=-1, random_state=RANDOM_SEED)
XGB_rs.fit(X_train, y_train)

# Output the runtime and best parameters (as well as the runner-ups)
print("Elapsed time (sec):" + "%6.0f" % (time.time() - tic))
print("Best Parameters:", XGB_rs.best_params_)
pd.DataFrame(XGB_rs.cv_results_)[sel_params_XGB].sort_values("rank_test_score").head()
```

```
Elapsed time (sec):   677
Best Parameters: {'colsample_bytree': 0.32533033076326434, 'learning_rate': 0.018692774498209535, 'max_depth': 6, 'subsample': 0.5396921323890
798}
```

Out[59]:

	param_learning_rate	param_max_depth	param_subsample	param_colsample_bytree	mean_test_score	rank_test_score
15	0.018693	6	0.539692	0.325330	0.691951	1
12	0.019823	6	0.567700	0.755361	0.690907	2
1	0.020494	7	0.459249	0.596850	0.690617	3
8	0.011104	8	0.942202	0.170524	0.689871	4
6	0.014545	7	0.382462	0.973756	0.689156	5

In [60]:
```
# XGBClassifier: Create a new model on all folds using the best parameters
tic = time.time()
XGB = XGBClassifier(**XGB_rs.best_params_, n_estimators=1000, tree_method="hist", eval_metric="auc")
XGB.fit(X_train, y_train)
print("Elapsed time (sec):" + "%6.0f" % (time.time() - tic))
```

```
Elapsed time (sec):    24
```

In [61]:
```
# Calculate and display the model AUC
auc_XGB = roc_auc_score(y_val, XGB.predict_proba(X_val)[:,1])
print('The validation AUC of the XGBoost model with tuned hyperparameters is: {:.6f}'.format(auc_XGB))
```

```
The validation AUC of the XGBoost model with tuned hyperparameters is: 0.700406
```

For the HP grids, it is important to ensure that the parameters used are not located at the outer edges of the specified value range. If this is the case, it may indicate the need for further optimization. The value ranges should be adjusted. Compared to the CSN notebook, this has been done for the HP tuning. The calculated AUC values fall behind the original CatBoost model. In the next section, these models will be compared overall, including with the following models.

This marks the end of the modeling approaches from the CSN notebook for now.

Task PT4: Forming DXG Diagnosis Groups and Preparing Embedding Creation

Task PT4-1: [Learning Objective 5.1; 5 Points]

In preparation for sections PT4 and PT5, adjustments are made for categorical features to create embeddings.

Note: This section of the task steps outside the scope of the CSN notebook. The following tasks must be completed in the given sequence.

a) Read the dataset `icd9_data.csv` provided in the materials and display the first three rows.

b) Merge the original dataset (from task section PT1 1.2) with the dataset from part a). Only the columns `group`, `L2code`, and `category` from the dataset in part a) should be retained. Afterward, display three rows of the new dataset.

c) Based on the dataset from PT4-1 b): Which categories have the highest absolute proportions with regard to readmission? Display the top three categories.

d) The feature `category` has values that are sparsely represented in some cases. To create stable embeddings for this feature, proceed as follows:

- Add the column `sum_target_cg` to the dataset from part b), which is to be populated using the result from part c). This should assign the corresponding absolute proportions to the values of `category`.
- Finally, add the column `hdiag`: If the category of a row appears more than 20 times in the dataset, the value of `category` is to be retained. Otherwise, the value should be the combination of the string "o." and the value of the feature `group`. Shorten the content of the new column to a maximum of 30 characters. Verify the results using two arbitrary rows as examples.

Suggested Solution:

a):

In [62]:
```
# Read diagnosis grouping data
# icd9_data = pd.read_csv("C:/ml/data/DiabetesHospital/icd9_data.csv", sep=";")
# icd9_data = pd.read_csv("../input/diabeteshospital/icd9_data.csv", sep=";")
icd9_data = pd.read_csv("../2_Datensätze/icd9_data.csv", sep=";")
# icd9_data = pd.read_csv("./PK_VADS_IMMERSION/2025//2_Datensätze/icd9_data.csv", sep=";")
icd9_data.head(3)
```

Out[62]:

	category_id	code	group	chapter	chapter_id	L2code	category	X
0	0	?	n.a.	n.a	0	n.a.	Not available	NaN
1	135	1	Digestive	Diseases of the digestive system	9	c9.1	Intestinal infection	NaN
2	1	10	Other	Infectious and parasitic diseases	1	c1.1	Tuberculosis	NaN

b):

In [63]:
```
df_diag = pd.merge(df_raw, icd9_data, how='left', left_on= 'diag_1', right_on= 'code')
df_diag = df_diag.drop(columns=['category_id', 'code', 'chapter','chapter_id', 'X'], axis=1)
df_diag.head(3)
```

Out[63]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_specialty	num_
0	Caucasian	Female	[50-60)	?	2	1	1	8	?	Cardiology	
1	Caucasian	Female	[50-60)	?	3	1	1	2	?	Surgery-Neuro	
2	Caucasian	Female	[80-90)	?	1	3	7	4	MC	InternalMedicine	

c):

In [64]:
```
# Step 1: Group by "category" and sum up the variable TARGET
df_sum = df_diag.groupby('category', as_index=False)['TARGET'].sum().rename(columns={'TARGET': 'sum_target_cg'})
df_sum.sort_values(by='sum_target_cg', ascending=False).head(3)
```

Out[64]:

	category	sum_target_cg
60	Congestive heart failure; nonhypertensive	443
62	Coronary atherosclerosis and other heart disease	429
5	Acute cerebrovascular disease	324

d):

In [65]:
```
# Step 2: Assign the result by "category" to the DataFrame df_diag (merge)
df_diag = df_diag.merge(df_sum, on='category', how='left')

min_TARGET = 20

# Step 3: Calculate new field "h_diag": If TARGET is too rare, then use the broad disease group and mark it with 'o.' (as it does not include
df_diag['hdiag'] = df_diag.apply(
    lambda row: 'o.' + row['group'] if row['sum_target_cg'] < min_TARGET else row['category'], axis=1
)

# Shorten length
df_diag['hdiag'] = df_diag['hdiag'].str[:30]

# Count results
df_diag['hdiag'].value_counts()
```

Out[65]:
```
hdiag
Coronary atherosclerosis and o    3835
Congestive heart failure; nonh    2219
Other lower respiratory diseas    2170
Diabetes mellitus without comp    2007
Acute myocardial infarction       2006
                                  ...
Acute posthemorrhagic anemia       135
Chronic ulcer of skin               98
Maintenance chemotherapy; radi      83
Other liver diseases                72
o.n.a.                               8
Name: count, Length: 72, dtype: int64
```

In [66]:
```
# Example with category >= 20
df_diag[["sum_target_cg", "category", "group", "hdiag"]].iloc[0]
```

Out[66]:
```
sum_target_cg                       22
category         Essential hypertension
group                      Circulatory
hdiag            Essential hypertension
Name: 0, dtype: object
```

In [67]:
```
# Example with category < 20
df_diag[["sum_target_cg", "category", "group", "hdiag"]].iloc[3]
```

Out[67]:
```
sum_target_cg                                    6
category         Gout and other crystal arthropathies
group                                        Other
hdiag                                      o.Other
Name: 3, dtype: object
```

Task PT4-2: [Learning Goal 5.1 / 2.2; 10 Points]

a) After adjusting the feature `category` in the previous task, other relevant categorical features will be adjusted further. Proceed as follows:

If the number of feature values is less than the threshold of 20, then:

- for the feature `discharge_disposition_id`, the value should be replaced with `99`,
- for the feature `medical_specialty`, the value should be replaced with `ZZ`,
- for the feature `payer_code`, the value should be replaced with `ZZ`,
- for the feature `admission_type_id`, the value should be replaced with `99`,
- for the feature `age`, the value should be replaced with `[0,20)`.

After the replacement, the feature values and their respective frequencies should be displayed.

b) Visualize the features created under a) as well as in PT4-1 appropriately. At least one standout observation should be analyzed in detail.

Proposed Solution:

a)

In [68]:
```
# discharge_disposition:

# Step 1: Group by FEATURE and sum the variable TARGET
df_sum = df_diag.groupby('discharge_disposition_id', as_index=False)['TARGET'].sum().rename(columns={'TARGET': 'sum_target_dd'})

# Step 2: Assign the result by FEATURE to the DataFrame df_diag (Merge)
df_diag = df_diag.merge(df_sum, on='discharge_disposition_id', how='left')

# Step 3: Calculate a new field: If TARGET is too rare, combine into a "remainder group"
df_diag['ddisp'] = df_diag.apply(
    lambda row: 99 if row['sum_target_dd'] < min_TARGET else row['discharge_disposition_id'], axis=1)

# Count the result
df_diag['ddisp'].value_counts()
```

Out[68]:
```
ddisp
1     30258
3      6029
6      5189
18     1921
2      1101
22     1073
5       664
25      494
4       375
7       296
99      279
28       72
Name: count, dtype: int64
```

In [69]:
```
# medical_specialty:

# Step 1: Group by FEATURE and sum the variable TARGET
df_sum = df_diag.groupby('medical_specialty', as_index=False)['TARGET'].sum().rename(columns={'TARGET': 'sum_target_ms'})

# Step 2: Assign the result to the DataFrame df_diag via FEATURE (Merge)
df_diag = df_diag.merge(df_sum, on='medical_specialty', how='left')

# Step 3: Calculate a new field: If TARGET is too rare, group it into a rest category
```

```
df_diag['mspec'] = df_diag.apply(
    lambda row: 'ZZ' if row['sum_target_ms'] < min_TARGET else row['medical_specialty'], axis=1)

# Count the result
df_diag['mspec'].value_counts()
```

Out[69]:

```
mspec
?                                 22642
InternalMedicine                   7341
Family/GeneralPractice             3312
Emergency/Trauma                   2875
Cardiology                         2811
Surgery-General                    1528
ZZ                                 1126
Orthopedics                         891
Orthopedics-Reconstructive          806
Radiologist                         576
ObstetricsandGynecology             506
Nephrology                          460
Psychiatry                          450
Pulmonology                         415
Urology                             409
Surgery-Cardiovascular/Thoracic     402
Surgery-Neuro                       340
Gastroenterology                    246
Surgery-Vascular                    241
PhysicalMedicineandRehabilitation   153
Oncology                            144
Hematology/Oncology                  77
Name: count, dtype: int64
```

In [70]:

```
# payer_code:

# Step 1: Group by FEATURE and sum the variable TARGET
df_sum = df_diag.groupby('payer_code', as_index=False)['TARGET'].sum().rename(columns={'TARGET': 'sum_target_pc'})

# Step 2: Assign the result to the DataFrame df_diag via FEATURE (Merge)
df_diag = df_diag.merge(df_sum, on='payer_code', how='left')

# Step 3: Calculate a new field: If TARGET is too rare, group it into a rest category
df_diag['pcode'] = df_diag.apply(
    lambda row: 'ZZ' if row['sum_target_pc'] < min_TARGET else row['payer_code'], axis=1)

# Count the result
df_diag['pcode'].value_counts()
```

Out[70]:

```
pcode
?     20378
MC    13115
HM     2821
BC     2601
SP     2259
MD     1522
CP     1391
UN     1385
CM      913
OG      475
PO      357
ZZ      278
DM      256
Name: count, dtype: int64
```

In [71]:

```
# admission_type:

# Step 1: Group by FEATURE and sum the variable TARGET
df_sum = df_diag.groupby('admission_type_id', as_index=False)['TARGET'].sum().rename(columns={'TARGET': 'sum_target_at'})

# Step 2: Assign the result to the DataFrame df_diag via FEATURE (Merge)
df_diag = df_diag.merge(df_sum, on='admission_type_id', how='left')

# Step 3: Calculate a new field: If TARGET is too rare, group it into a rest category
df_diag['atype'] = df_diag.apply(
    lambda row: 99 if row['sum_target_at'] < min_TARGET else row['admission_type_id'], axis=1)

# Count the result
df_diag['atype'].value_counts()
```

Out[71]:

```
atype
1     24036
3     10056
2      8731
6      2641
5      2047
8       222
99       18
Name: count, dtype: int64
```

In [72]:

```
# age:

# Step 1: Group by FEATURE and sum the variable TARGET
df_sum = df_diag.groupby('age', as_index=False)['TARGET'].sum().rename(columns={'TARGET': 'sum_target_age'})

# Step 2: Assign the result to the DataFrame df_diag via FEATURE (Merge)
df_diag = df_diag.merge(df_sum, on='age', how='left')
```

```
# Step 3: Calculate a new field: If TARGET is too rare, group it into a rest category
df_diag['agegr'] = df_diag.apply(
    lambda row: '[0-20)' if row['sum_target_age'] < min_TARGET else row['age'], axis=1)

# Count the result
df_diag['agegr'].value_counts()
```

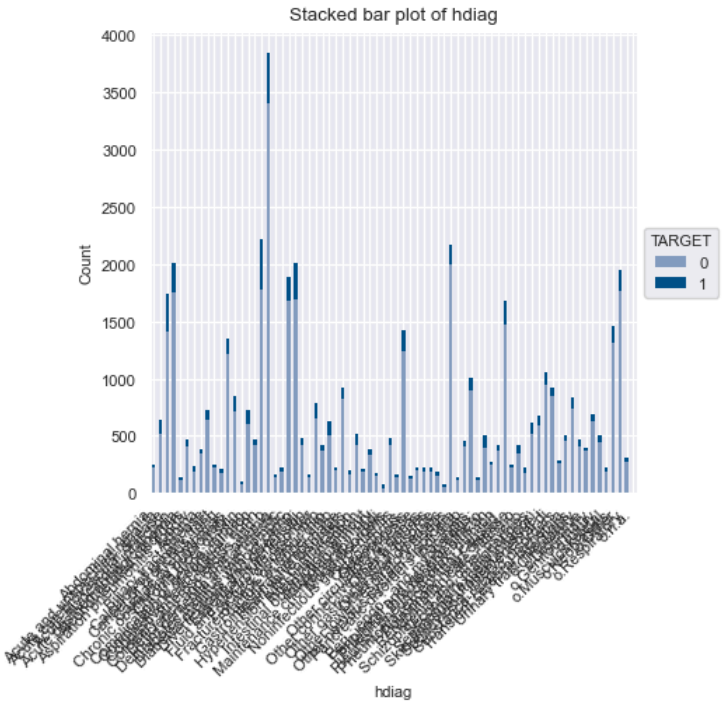
Out[72]:

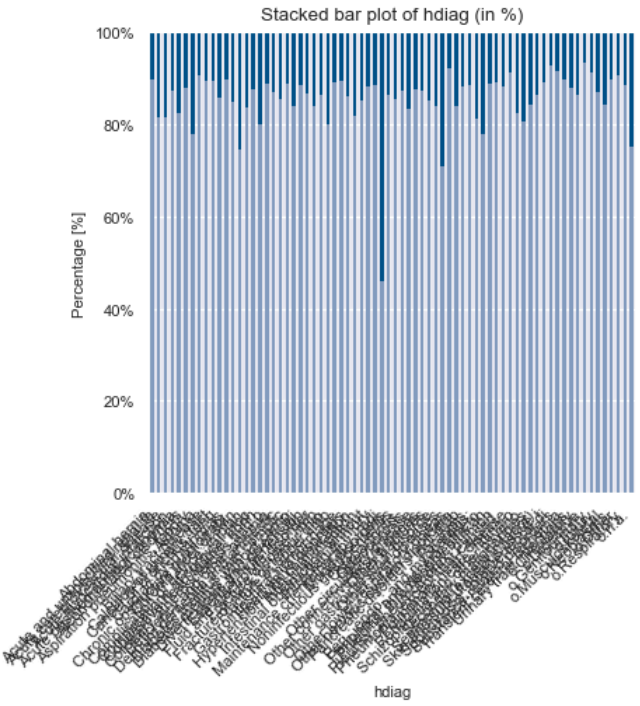
```
agegr
[70-80)     11623
[60-70)     10736
[50-60)      8615
[80-90)      7242
[40-50)      4885
[30-40)      2003
[90-100)     1282
[20-30)       844
[10-20)       392
[0-20)        129
Name: count, dtype: int64
```

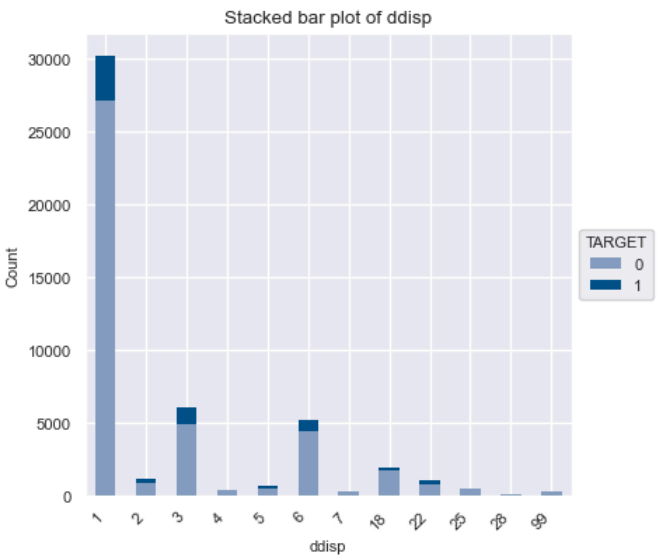
Zu b)

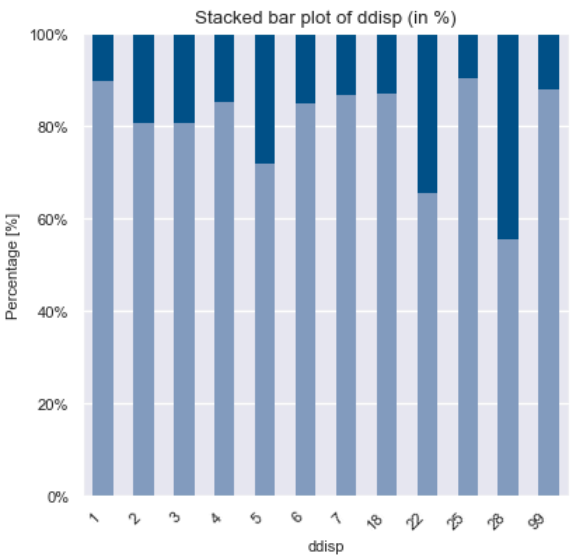
In [73]:

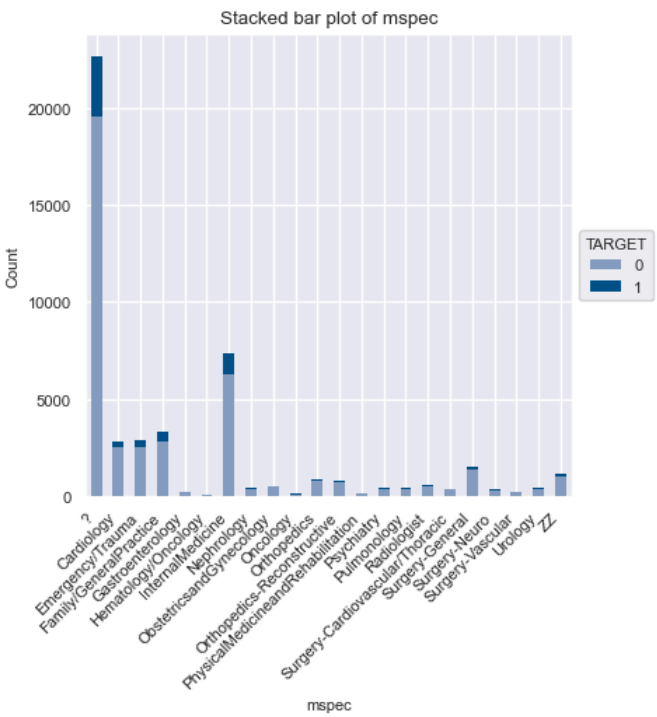
```
# List of selected categorical features to analyze
selected_categorical_features = ['hdiag', 'ddisp','mspec', 'pcode', 'atype', 'agegr']

# Number of rows/columns for the subplot grid
n_cols = 2  # doubled to fit count and percentage plots side by side
n_rows = len(selected_categorical_features)  # one row for each feature

# Set up the matplotlib figure
plt.figure(figsize=(n_cols * 5, n_rows * 5))

# Correct font size
plt.rcParams.update({'font.size': 8})

# Loop through the number of categorical features
for idx, feature in enumerate(selected_categorical_features):
    # Create a crosstab for stacked bar plot structure
    ctab = pd.crosstab(df_diag[feature], df_diag['TARGET'])

    # (1st column) add a new subplot iteratively for count values
    ax1 = plt.subplot(n_rows, n_cols, idx * n_cols + 1)

    # Create a stacked bar plot for count values
    ctab.plot(kind="bar", stacked=True, color=[COLOR_LIGHT, COLOR_DARK], edgecolor="none", ax=ax1)

    # Additional plot settings for count subplot
    ax1.set_title(f'Stacked bar plot of {feature}')
    ax1.set_xlabel(feature)
    ax1.set_ylabel('Count')
    ax1.legend(title='TARGET', loc='center left', bbox_to_anchor=(1, 0.5))
    plt.xticks(rotation=45, ha='right')

    # (2nd column) add a new subplot iteratively for percentage values
    ax2 = plt.subplot(n_rows, n_cols, idx * n_cols + 2)

    # Normalize the crosstab by row and multiply by 100 to convert to percentages
    ctab_normalized = ctab.div(ctab.sum(axis=1), axis=0) * 100

    # Create a stacked bar plot for percentage values
    ctab_normalized.plot(kind="bar", stacked=True, color=[COLOR_LIGHT, COLOR_DARK], edgecolor="none", ax=ax2)

    # Additional plot settings for percentage subplot
    ax2.set_title(f'Stacked bar plot of {feature} (in %)')
    ax2.set_xlabel(feature)
    ax2.set_ylabel('Percentage [%]')
    ax2.legend().remove()
    plt.xticks(rotation=45, ha='right')

    # Make yticks be in percentages for the percentage subplot
    ax2.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, loc: "{:.0f}%".format(int(x))))
    ax2.set_ylim(0, 100)

# Preventing subplots from being too close to each other and display the final plot
plt.tight_layout()
plt.show()
```

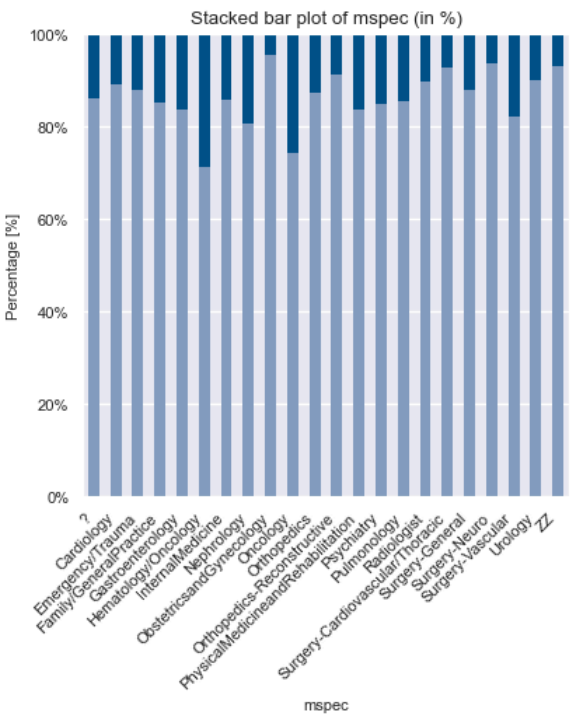

Stacked bar plot of hdiag
Count
TARGET
0
1
hdiag

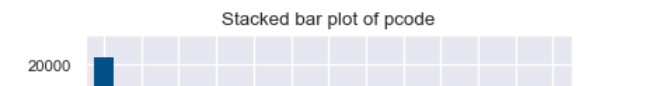

Stacked bar plot of hdiag (in %)
Percentage [%]
hdiag

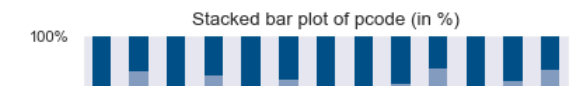

Stacked bar plot of ddisp
Count
TARGET
0
1
ddisp

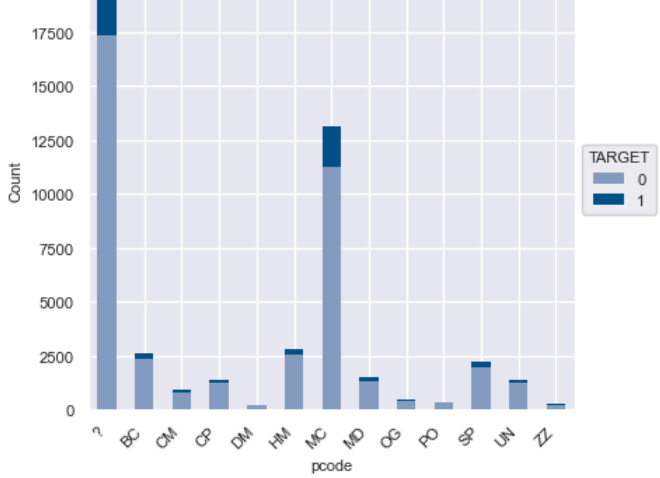

Stacked bar plot of ddisp (in %)
Percentage [%]
ddisp

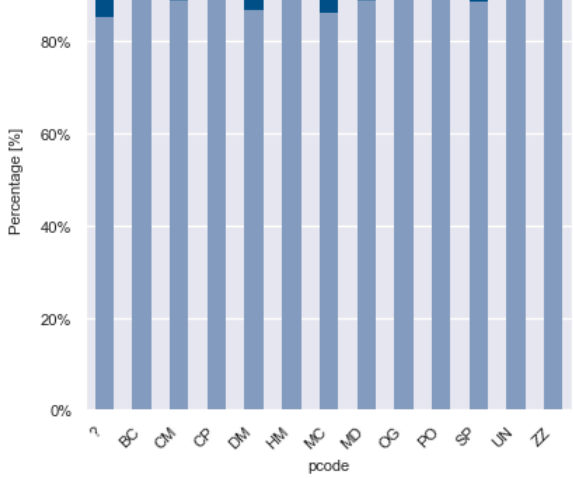

Stacked bar plot of mspec
Count
TARGET
0
1
mspec

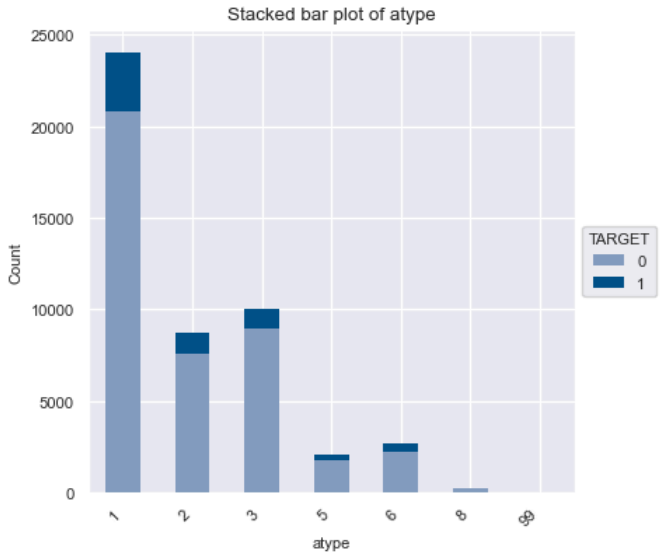

Stacked bar plot of mspec (in %)
Percentage [%]
mspec

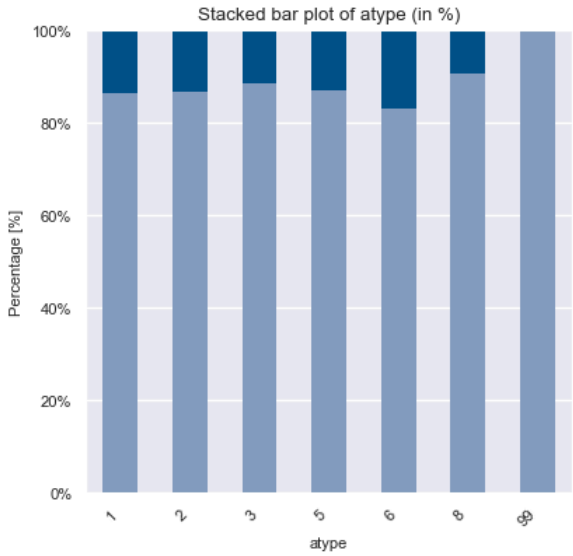

Stacked bar plot of pcode

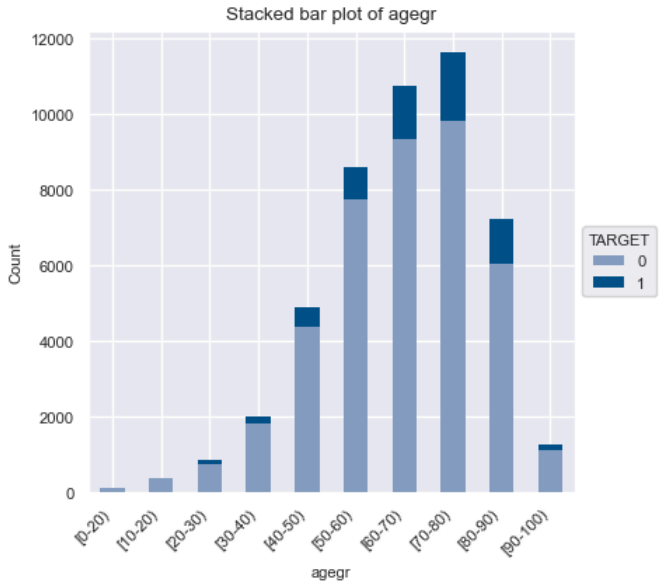

Stacked bar plot of pcode (in %)

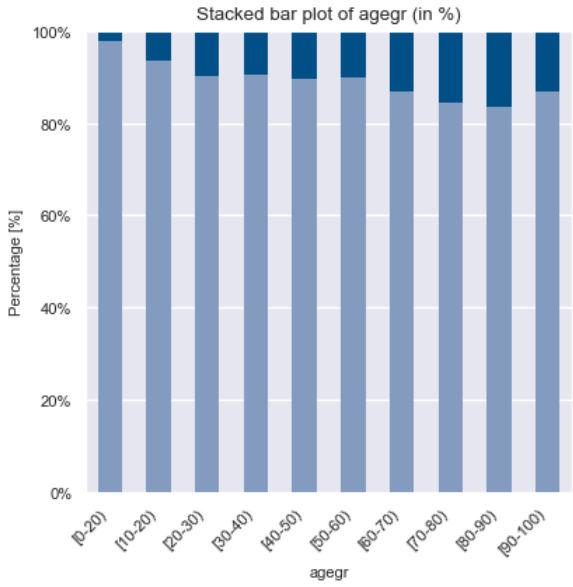












The graphic for `hdiag` (consolidated feature `category`) still shows many values (resulting in barely readable text on the horizontal axis). Upon closer inspection, a deep blue bar can be seen in the right graphic, indicating a noticeable value with a high probability of readmission. To determine this value, the readmission probabilities are calculated and sorted in descending order:

In [74]:
```
# Check hdiag: Diseases with the highest readmission probabilities
df_diag.groupby('hdiag', as_index=False)['TARGET'].mean().sort_values(by='TARGET', ascending=False).head()
```

Out[74]:

	hdiag	TARGET
34	Maintenance chemotherapy; radi	0.542169
43	Other liver diseases	0.291667
13	Chronic ulcer of skin	0.255102
71	o.n.a.	0.250000
6	Aspiration pneumonitis; food/v	0.223629

From this list, it can be seen that this feature refers to chemotherapy. Increased hospital visits are understandable and explainable.

For the feature `mspec`, a noticeable blue bar can also be observed. Since the consolidation of sparsely populated feature values has provided a well-readable representation, the feature can be directly identified: Hematology/Oncology. Determination of readmission probabilities by `mspec`:

In [75]:
```
# Check mspec: Medical Specialty with the highest readmission probabilities
df_diag.groupby('mspec', as_index=False)['TARGET'].mean().sort_values(by='TARGET', ascending=False).head()
```

Out[75]:

	mspec	TARGET
5	Hematology/Oncology	0.285714
9	Oncology	0.256944
7	Nephrology	0.191304
19	Surgery-Vascular	0.178423
12	PhysicalMedicineandRehabilitation	0.163399

Task PT5: Creation of Stable Diagnosis Embeddings with a High-Performance Neural Network

Task PT5-1: [Learning Objective 5.1; 6 points]

a) Perform label encoding for the categorical variables created in task parts PT4-1 and PT4-2. Display three random rows from the new dataset.

b) Scale the numerical features `number_inpatient`, `num_lab_procedures`, `number_diagnoses`, and `num_medications`. Demonstrate the effect of the scaling.

c) To train a neural network, create a dataset consisting of the categorical features from PT4-2 a) and the numerical features from part b) of this task. Split this dataset into proportions of 30%/15%/15% (training set, validation set, test set).

Proposed Solution:

a):

In [76]:
```
# Create a list of nominal features for embeddings
emb_features = ['hdiag', 'ddisp', 'mspec', 'pcode', 'atype', 'agegr']
num_emb_features = []

# "Factorize" nominal features (label encoding), 0,1,2,...
for i in emb_features:
    df_diag[f'num_{i}'] = df_diag[f'{i}'].factorize()[0]
    num_emb_features.append(f'num_{i}')
num_emb_features
```

Out[76]:
```
['num_hdiag', 'num_ddisp', 'num_mspec', 'num_pcode', 'num_atype', 'num_agegr']
```

In [77]:
```
df_diag.sample(n=3)
```

Out[77]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_specialty
47523	Asian	Female	[50-60)	?	3	6	1	1	?	Orthopedics
8582	Caucasian	Male	[40-50)	?	2	6	4	8	?	?
42024	Caucasian	Female	[70-80)	?	1	3	7	3	MC	?

b):

In [78]:
```
# Scaling of numerical data using StandardScaler
num_features = ['number_inpatient', 'num_lab_procedures', 'number_diagnoses', 'num_medications']
scaler = StandardScaler()
df_scaled = df_diag.copy(deep=True)
df_scaled[num_features] = scaler.fit_transform(df_diag[num_features])
```

```
# Display summary statistics
df_scaled.describe()
```

Out[78]:

	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpatient	number_emergency	number_inpatient	number_diagn
count	47751.000000	4.775100e+04	47751.000000	4.775100e+04	47751.000000	47751.000000	4.775100e+04	4.775100e
mean	4.205106	5.356859e-17	1.458043	-4.850934e-17	0.232979	0.081569	4.761653e-18	-1.047564
std	2.914298	1.000010e+00	1.764084	1.000010e+00	0.927995	0.445226	1.000010e+00	1.000010e
min	1.000000	-2.087767e+00	0.000000	-1.709703e+00	0.000000	0.000000	-2.595729e-01	-2.981599e
25%	2.000000	-6.237921e-01	0.000000	-6.528603e-01	0.000000	0.000000	-2.595729e-01	-1.027122e
50%	3.000000	8.295457e-02	1.000000	-1.831526e-01	0.000000	0.000000	-2.595729e-01	4.387350
75%	6.000000	6.887375e-01	2.000000	5.214089e-01	0.000000	0.000000	-2.595729e-01	9.273541
max	14.000000	4.525362e+00	6.000000	7.684451e+00	36.000000	37.000000	2.226972e+01	4.347688e

In [79]:
```
df_scaled.head(3)
```

Out[79]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_specialty	num_
0	Caucasian	Female	[50-60)	?	2	1	1	8	?	Cardiology	
1	Caucasian	Female	[50-60)	?	3	1	1	2	?	Surgery-Neuro	
2	Caucasian	Female	[80-90)	?	1	3	7	4	MC	InternalMedicine	

c):

In [80]:
```
# Generate the samples once again: X-y-split
feature_list = num_features + num_emb_features
xn = df_scaled[feature_list]
yn = df_scaled.TARGET
xn.head(3)
```

Out[80]:

	number_inpatient	num_lab_procedures	number_diagnoses	num_medications	num_hdiag	num_ddisp	num_mspec	num_pcode	num_atype	num_agegr
0	-0.259573	1.748858	0.438735	2.047959	0	0	0	0	0	0
1	-0.259573	0.335364	-2.004361	-0.535433	1	0	1	0	1	0
2	-0.259573	1.294520	0.927354	0.873690	2	1	2	1	2	1

In [81]:
```
# Split dataset in 70% training, 15% validation and 15% test and devide into feature matrix x and label y
xn_train, xn_valtest, yn_train, yn_valtest = train_test_split(xn, yn, train_size=0.70, random_state=RANDOM_SEED)
xn_val, xn_test, yn_val, yn_test = train_test_split(xn_valtest, yn_valtest, test_size=0.50, random_state=RANDOM_SEED)
print('Sample sizes for training, validation and test: ', xn_train.shape, xn_val.shape, xn_test.shape)
```

```
Sample sizes for training, validation and test:  (33425, 10) (7163, 10) (7163, 10)
```

Task PT5-2: [Learning Objective 5.1; 20 points]

a) A neural network is to be created for the embeddings of the categorical features. The following should be considered:

- The input to the network is the dataset from PT5-1 c).
- For each categorical feature, an embedding with two dimensions should be created.
- The number of hidden layers can be chosen freely. The final decision must be justified.

b) Fit the network defined in a). Plot the Loss and Validation Loss over the training epochs. Comment on the trend. Then calculate the Validation Loss.

c) To enable an evaluation of the computed embeddings in the next section, the network from part a) should be cloned and then fitted again. The Validation Loss should be calculated and displayed.

d) Extract the embeddings from both networks and plot them side by side in pairs. What anomalies are observed? How can the embeddings be assessed in terms of stability?

Proposed Solution:

a):

In [82]:
```
# Features for the embeddings
print(emb_features)
```

```
['hdiag', 'ddisp', 'mspec', 'pcode', 'atype', 'agegr']
```

In [83]:
```
# Count the number of unique values of the embedding features and save as len_name
for i in emb_features:
    # Create the variable name dynamically and assign the length
    var_name = f"len_{i}"
    globals()[var_name] = len(xn_train[f'num_{i}'].unique())
    print(var_name)
    print(globals()[var_name])
```

```
len_hdiag
72
len_ddisp
12
len_mspec
22
len_pcode
13
len_atype
7
len_agegr
10
```

Das folgende Netzwerk berücksichtigt die (skalierten) numerischen Variablen sowie die per label Encoding modifizierten kategoriellen Variablen. Nach der Angabe der Embedding Layers werden diese über Concatenate mit den numerischen Merkmalen zusammengeführt. Es werden insgesamt zwei hidden Layers berücksichtigt, da damit bereits eine schnelle Konvergenz erreicht werden kann.

In [84]:
```
# Numerical input
in_num = Input(shape=(4,), name='in_num')

# Define input dimension dictionary for each feature
input_dims = {'hdiag': len_hdiag, 'ddisp': len_ddisp, 'mspec': len_mspec, 'pcode': len_pcode, 'atype': len_atype, 'agegr': len_agegr}

# Dictionary to store the input and embedding layers
input_layers = {}
embedding_layers = {}

# Loop through each feature to create Input, Embedding, and Flatten layers
for feature in emb_features:
    # Create Input layer
    input_layers[feature] = Input(shape=(1,), name=f'in_{feature}')

    # Create Embedding and Flatten layers
    embedding = Embedding(input_dim=input_dims[feature], output_dim=2, name=f'emb_{feature}')(input_layers[feature])
    embedding_layers[feature] = Flatten()(embedding)

# Concatenate the numerical input and all embedding layers
concatenated_input = concatenate([in_num] + list(embedding_layers.values()), name='concate')

# Define hidden layers
hidden1 = Dense(64, kernel_regularizer=regularizers.l2(0.0001), activation='relu')(concatenated_input)
hidden2 = Dense(32, kernel_regularizer=regularizers.l2(0.0001), activation='relu')(hidden1)
hidden3 = Dense(16, kernel_regularizer=regularizers.l2(0.0001), activation='relu')(hidden2)

# Output layer
output = Dense(1, activation='sigmoid')(hidden3)
#output = Dense(1, activation='sigmoid')(hidden2)

# Instantiate and compile the model
NNemb = Model(inputs=[in_num] + list(input_layers.values()), outputs=output)
NNemb.compile(optimizer='adam', loss='binary_crossentropy', metrics=['AUC'])

# Summary
NNemb.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
in_hdiag (InputLayer)	(None, 1)	0	-
in_ddisp (InputLayer)	(None, 1)	0	-
in_mspec (InputLayer)	(None, 1)	0	-
in_pcode (InputLayer)	(None, 1)	0	-
in_atype (InputLayer)	(None, 1)	0	-
in_agegr (InputLayer)	(None, 1)	0	-
emb_hdiag (Embedding)	(None, 1, 2)	144	in_hdiag[0][0]
emb_ddisp (Embedding)	(None, 1, 2)	24	in_ddisp[0][0]
emb_mspec (Embedding)	(None, 1, 2)	44	in_mspec[0][0]
emb_pcode (Embedding)	(None, 1, 2)	26	in_pcode[0][0]
emb_atype (Embedding)	(None, 1, 2)	14	in_atype[0][0]
emb_agegr (Embedding)	(None, 1, 2)	20	in_agegr[0][0]
in_num (InputLayer)	(None, 4)	0	-
flatten (Flatten)	(None, 2)	0	emb_hdiag[0][0]
flatten_1 (Flatten)	(None, 2)	0	emb_ddisp[0][0]
flatten_2 (Flatten)	(None, 2)	0	emb_mspec[0][0]
flatten_3 (Flatten)	(None, 2)	0	emb_pcode[0][0]
flatten_4 (Flatten)	(None, 2)	0	emb_atype[0][0]
flatten_5 (Flatten)	(None, 2)	0	emb_agegr[0][0]
concate (Concatenate)	(None, 16)	0	in_num[0][0], flatten[0][0], flatten_1[0][0], flatten_2[0][0], flatten_3[0][0], flatten_4[0][0], flatten_5[0][0]
dense (Dense)	(None, 64)	1,088	concate[0][0]
dense_1 (Dense)	(None, 32)	2,080	dense[0][0]
dense_2 (Dense)	(None, 16)	528	dense_1[0][0]
dense_3 (Dense)	(None, 1)	17	dense_2[0][0]

Total params: 3,985 (15.57 KB)

Trainable params: 3,985 (15.57 KB)

Non-trainable params: 0 (0.00 B)

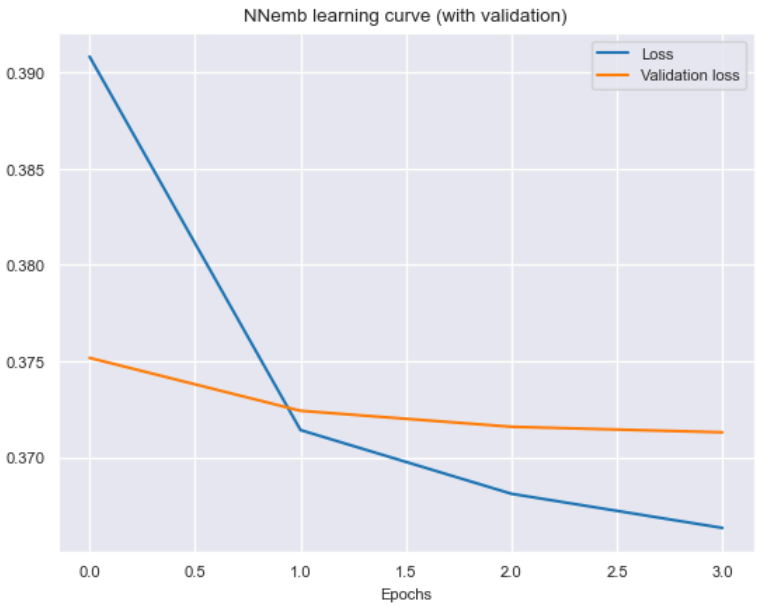
b):

In [85]:
```
# Fitting
tic = time.time()

# Prepare the input data using a list comprehension
input_data = [xn_train[num_features]] + [np.asarray(xn_train[feature]) for feature in num_emb_features]

# Fit the model
fit = NNemb.fit(input_data, np.asarray(yn_train), epochs=4, verbose=0, validation_split=0.2)

runtime_nnemb = time.time() - tic

print("Runtime NNemb (sec):" + "%6.0f" % (runtime_nnemb))
```

```
Runtime NNemb (sec):      8
```

In [86]:
```
try:
    plot = pd.DataFrame.from_dict({"Loss": fit.history["loss"],
```

```
                                    "Validation loss": fit.history["val_loss"],
                                   })
    plot.index.name = "Epochs"
    plot = plot.plot(title="NNemb learning curve (with validation)")
except Exception:
    pass
```



Interpretation:

The graph shows that the network converges after a short amount of time. Therefore, it can be assumed that the architecture used is suitable for the problem.

In [87]:
```
# Prepare the input data using a list comprehension
input_val_data = [xn_val[num_features]] + [np.asarray(xn_val[feature]) for feature in num_emb_features]

yn_val_pred = NNemb.predict(input_val_data)[:,0]
auc_NNemb = roc_auc_score(yn_val, yn_val_pred)

mname.append("NNemb")
mauc.append(auc_NNemb)

print('The validation AUC is: {:.6f}'.format(auc_NNemb))

# Prepare test data
input_test_data = [xn_test[num_features]] + [np.asarray(xn_test[feature]) for feature in num_emb_features]
yn_test_pred = NNemb.predict(input_test_data)[:,0]
```

```
224/224 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
The validation AUC is: 0.684252
224/224 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

c):

In [88]:
```
# Create and compile the model again

# Create a copy of the test model (with freshly initialized weights) and compile.
NNembC = tf.keras.models.clone_model(NNemb)
NNembC.compile(optimizer='adam', loss='binary_crossentropy', metrics=['AUC'])

# Fitting
fitC = NNembC.fit(input_data, np.asarray(yn_train), epochs=4, verbose=0, validation_split=0.2)

yn_val_predC = NNembC.predict(input_val_data)[:,0]
auc_NNembC = roc_auc_score(yn_val, yn_val_predC)

mname.append("NNembC")
mauc.append(auc_NNembC)

print('The validation AUC is: {:.6f}'.format(auc_NNembC))
```

```
224/224 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
The validation AUC is: 0.683469
```

d):

In [89]:
```
# Define a function that writes the 2-dimensional embedding weights of a specific layer with labels into a DataFrame
def weights_emb(prefix, feature, model, layer):
    name = f"wgt_{prefix}_{feature}"
    # Retrieve the specified layer from the model and create a DataFrame from its weights
    globals()[name] = model.layers[layer]
    globals()[name] = pd.DataFrame(globals()[name].get_weights()[0], columns=["dim_0", "dim_1"])
    # Group by `num_<feature>`, take the last value of `<feature>`, and assign it to the DataFrame
    globals()[name][feature] = df_diag.groupby(f"num_{feature}")[feature].last().values
    globals()[name] = globals()[name].set_index(feature)
```

In [90]:
```
# Apply the function to all embedding features (including the clone)
for idx, feature in enumerate(emb_features, start=6):
```

```
    weights_emb('NNemb', feature, NNemb, idx)
    weights_emb('NNembC', feature, NNembC, idx)

# Display the embedding weights for a feature (transposed)
wgt_NNembC_mspec.T
```

Out[90]:

mspec	Cardiology	Surgery-Neuro	InternalMedicine	Orthopedics-Reconstructive	Surgery-General	Surgery-Cardiovascular/Thoracic	Emergency/Trauma	Nephrology	Family/GeneralPractice
dim_0	0.099369	0.273834	0.088721	0.304053	0.052265	0.178722	0.046021	-0.061752	-0.05452
dim_1	-0.174990	-0.290567	0.023822	-0.305305	-0.041070	-0.146025	0.009892	0.177952	0.07565

In [91]:

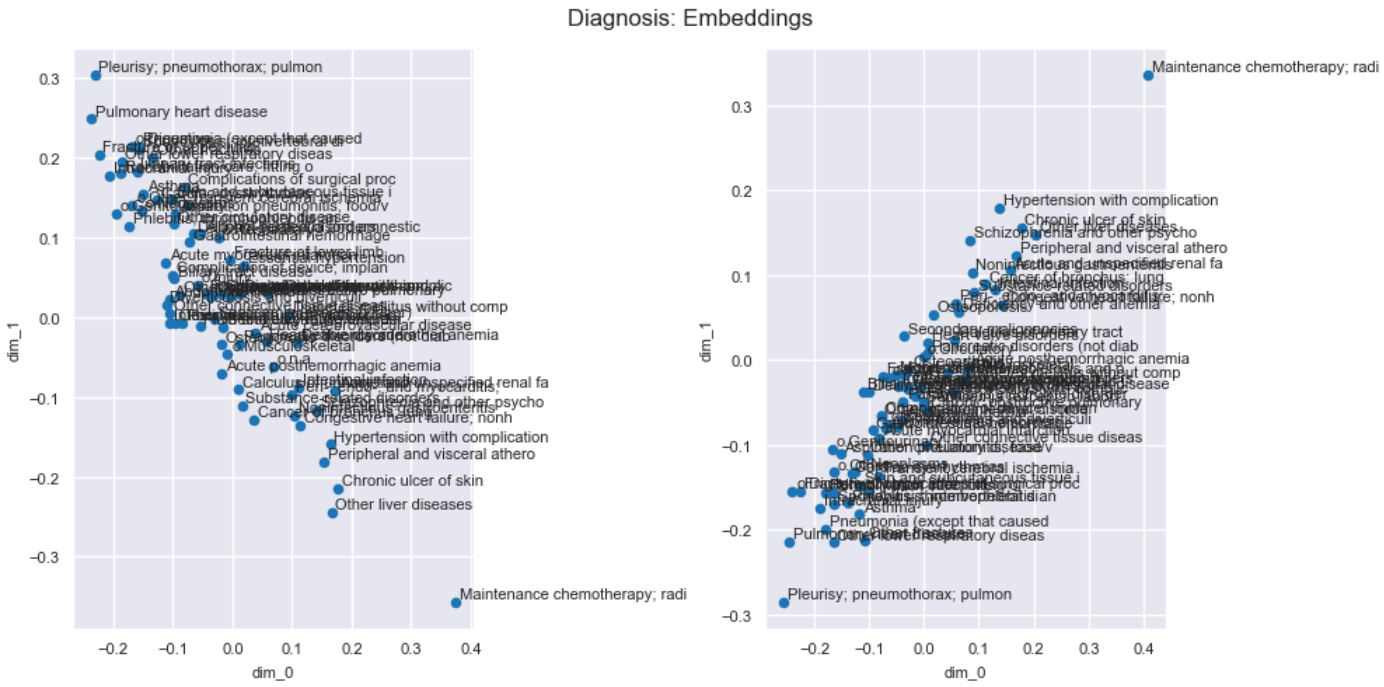
```
def plot_embeddings(df1, df2, text):
    fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
    fig.suptitle(f"{text}: Embeddings", fontsize=12)

    df1.plot(ax=ax1, kind="scatter", x="dim_0", y="dim_1")
    def annotate1(row, ax=ax1):
        ax.annotate(row.name, row.values[:2], xytext=(2, 2),
                    textcoords="offset points", size=8)
    df1.apply(annotate1, axis="columns")

    df2.plot(ax=ax2, kind="scatter", x="dim_0", y="dim_1")
    def annotate2(row, ax=ax2):
        ax.annotate(row.name, row.values[:2], xytext=(2, 2),
                    textcoords="offset points", size=8)
    df2.apply(annotate2, axis="columns")

    plt.tight_layout()
    plt.show()
```

Plot and analyze two-dimensional embeddings:

1. Diagnosis Embedding

In [92]:

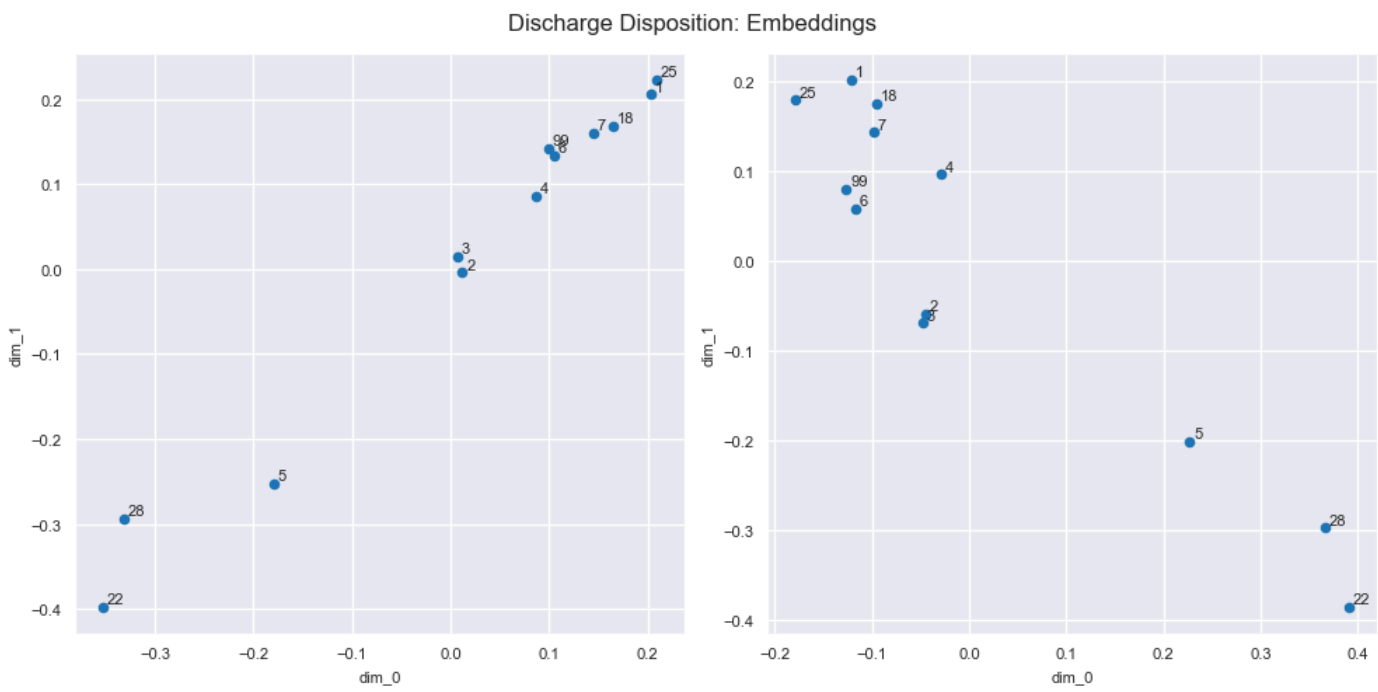
```
plot_embeddings(wgt_NNemb_hdiag, wgt_NNembC_hdiag, "Diagnosis")
```



2. Discharge Disposition

In [93]:

```
plot_embeddings(wgt_NNemb_ddisp,wgt_NNembC_ddisp, "Discharge Disposition")
```

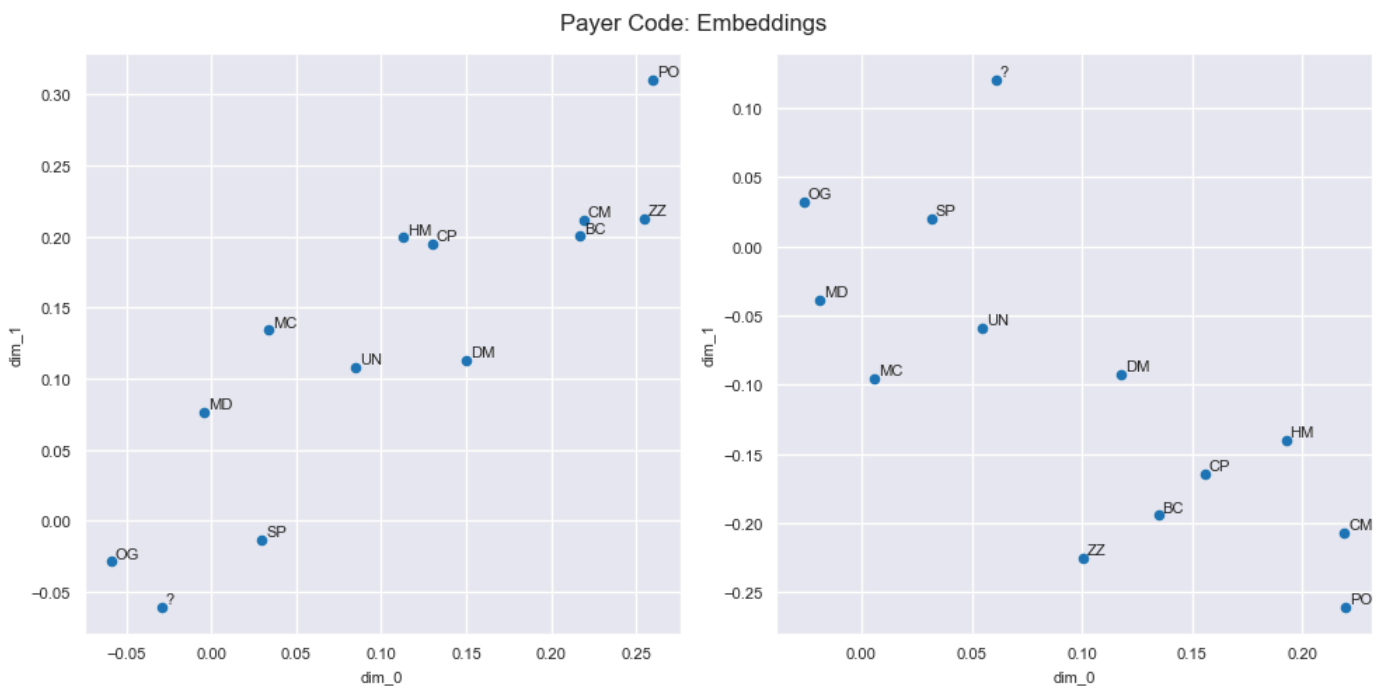


3. Medical Specialty

```
In [94]: plot_embeddings(wgt_NNemb_mspec, wgt_NNembC_mspec, "Medical Specialty")
```

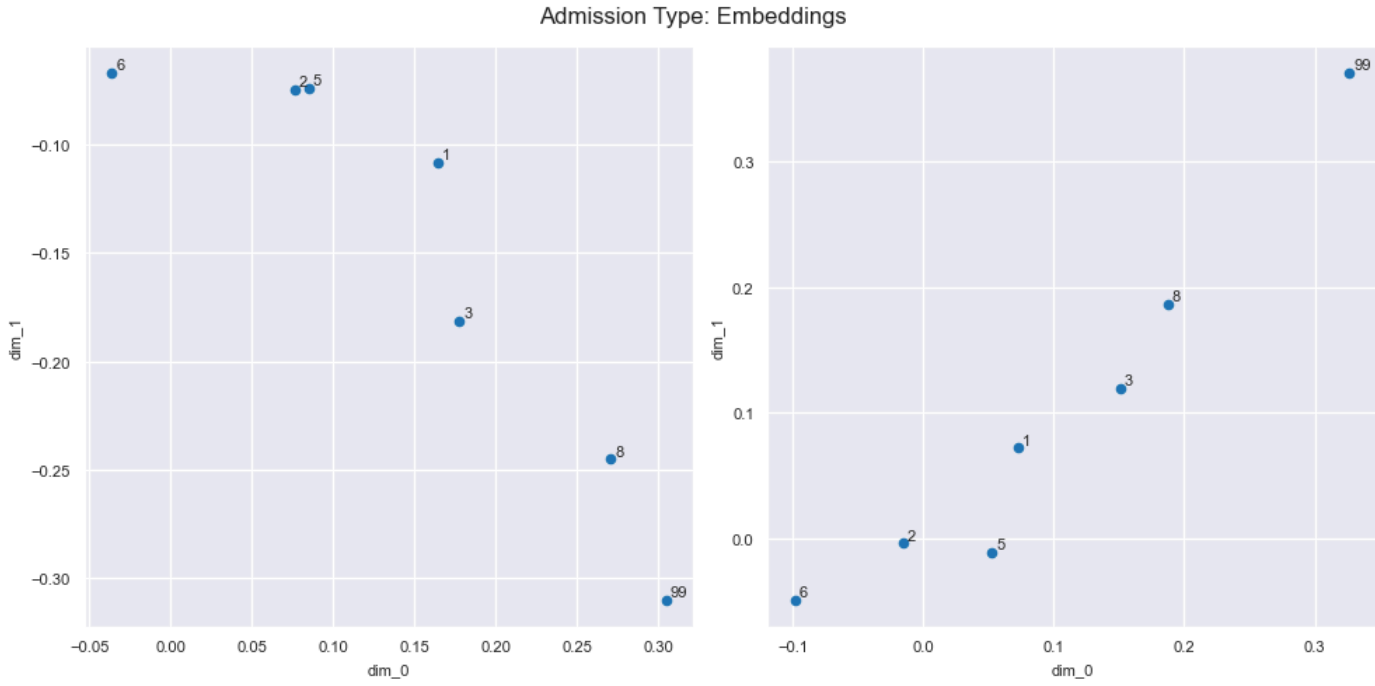
4. Payer Code

```
In [95]: plot_embeddings(wgt_NNemb_pcode, wgt_NNembC_pcode, "Payer Code")
```



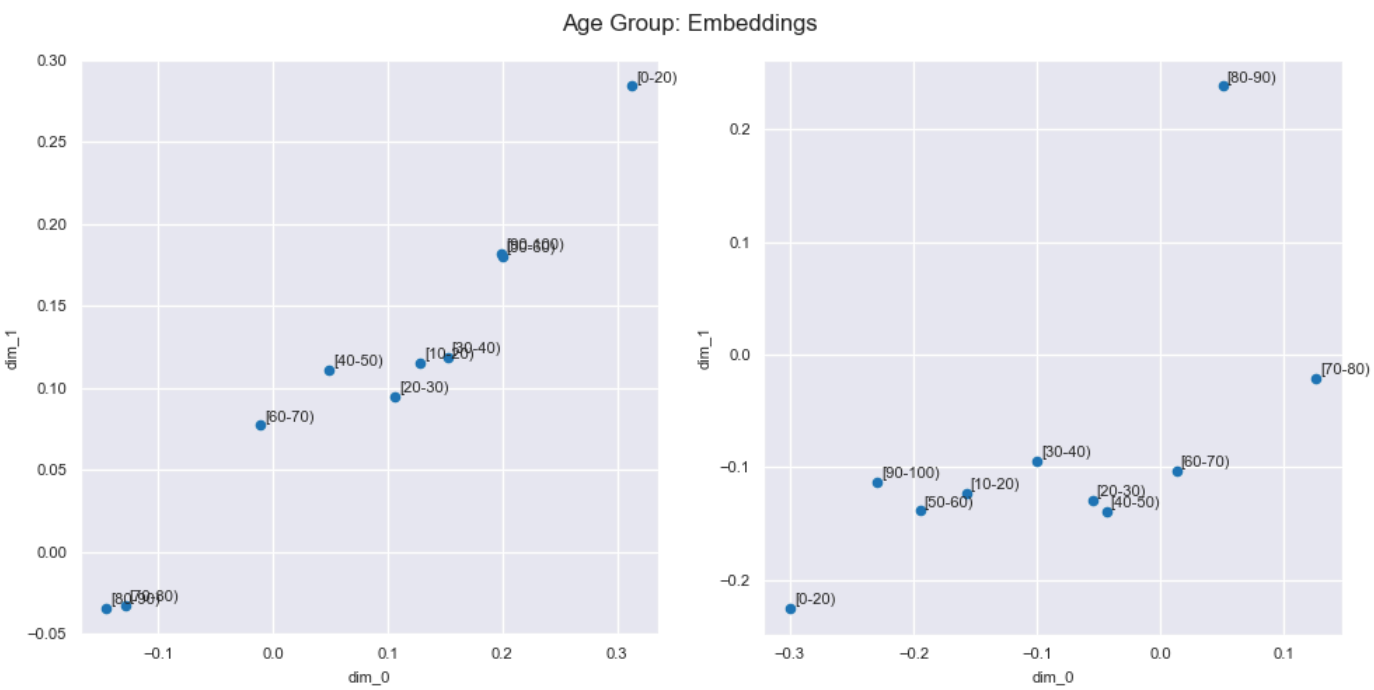
5. Admission Type

In [96]:
```
plot_embeddings(wgt_NNemb_atype, wgt_NNembC_atype, "Admission Type")
```



6. Age Group

In [97]:
```
plot_embeddings(wgt_NNemb_agegr, wgt_NNembC_agegr, "Age Group")
```



To assess the stability of the embeddings, they are compared in pairs. Since the network architecture (due to cloning) has not changed, stability is evaluated as follows: Differences may occur in pairwise comparisons (it cannot be expected that the images are identical). Slight variations, as well as rotations and reflections, are to be expected. Significant deviations from this indicate instability of the embeddings.

1. Diagnosis Embedding (hdiag): As already mentioned in Part PT4-2 b), the embeddings also show a noticeable difference in the feature `Maintenance chemotherapy` in both graphs. Apart from that, both rotation and reflection can be observed.
2. Discharge Disposition Embedding: A clear similarity between the two embedding representations is visible. Roughly three clusters of data points can be identified: the two points 22 and 28, the isolated point 5, and the remaining data points.
3. Medical Specialty Embedding: Clear similarities between the two images can also be observed here. Noteworthy is the feature `Obstetrics and Gynecology`, which stands out with a significant distance from the rest of the data points.

Similar observations apply to the embeddings of `payer code`, `Admission Type`, and `Age Group`. Therefore, the embeddings can be considered stable.

Task PT6: Joining Embeddings and Using Them in Modeling

Task PT6 - 1: [Learning Objective 5.1; 15 Points]

Using the embeddings generated in PT5, investigate to what extent the use of these(instead of the original categorical variables) impacts the modeling results(i.e., the AUC score).

a) Extend the dataset from task part PT5 - 1 a) with the corresponding results of the embeddings.Remove unnecessary features from the dataset.Subsequently, split this dataset into training (70 %), validation (15 %), and test (15 %) sets.

b) Based on the dataset from a): Perform logistic regression considering the embedding features.Use the model from section 2.2 as a reference.What can be said in comparison?

c) Analogous to part b): Perform CatBoost considering the embedding features.What can be said in comparison to the results in section 1.5(Baseline Model)?

d) Finally, evaluate the impact of the embeddings on the considered models.Use a bar chart that shows the previous models(i.e., in particular, the first calculated CatBoost model, logistic regression, as well as the models from the one-hot-encoding and subsampling) with their validation results.The evaluation should also include a statement regarding the usefulness of the embeddings, as well as your own recommendation.

Proposed Solution:

a):

In [98]:
```
# Base dataset for supplementing with embedding features
df_diag.head()
```

Out[98]:

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_specialty
0	Caucasian	Female	[50-60)	?	2	1	1	8	?	Cardiology
1	Caucasian	Female	[50-60)	?	3	1	1	2	?	Surgery-Neuro
2	Caucasian	Female	[80-90)	?	1	3	7	4	MC	InternalMedicine
3	Caucasian	Female	[80-90)	?	1	1	7	3	?	InternalMedicine
4	AfricanAmerican	Female	[30-40)	?	1	1	7	5	?	InternalMedicine

In [99]:
```
# Embedding dim0 and dim1 should be prefixed with feature names (if necessary, implement via a list, see above).
wgt_NNemb_hdiag = wgt_NNemb_hdiag.add_prefix('hdiag_')
wgt_NNemb_ddisp = wgt_NNemb_ddisp.add_prefix('ddisp_')
wgt_NNemb_mspec = wgt_NNemb_mspec.add_prefix('mspec_')
wgt_NNemb_pcode = wgt_NNemb_pcode.add_prefix('pcode_')
wgt_NNemb_atype = wgt_NNemb_atype.add_prefix('atype_')
wgt_NNemb_agegr = wgt_NNemb_agegr.add_prefix('agegr_')
```

In [100…
```
# Appearance of the embedding results (example):
wgt_NNemb_hdiag.head()
```

Out[100…

hdiag	hdiag_dim_0	hdiag_dim_1
Essential hypertension	0.020261	0.064555
Spondylosis; intervertebral di	-0.157591	0.211554
Fracture of neck of femur (hip	-0.044819	0.029408
o.Other	-0.168107	0.141445
Urinary tract infections	-0.160506	0.183175

In [101…
```
# Add embedding values
df_emb = df_diag.merge(wgt_NNemb_hdiag, on='hdiag', how='left')
df_emb = df_emb.merge(wgt_NNemb_ddisp, on='ddisp', how='left')
df_emb = df_emb.merge(wgt_NNemb_mspec, on='mspec', how='left')
df_emb = df_emb.merge(wgt_NNemb_pcode, on='pcode', how='left')
df_emb = df_emb.merge(wgt_NNemb_atype, on='atype', how='left')
df_emb = df_emb.merge(wgt_NNemb_agegr, on='agegr', how='left')
df_emb.head(3)
```

Out[101…

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code	medical_specialty	num_
0	Caucasian	Female	[50-60)	?	2	1	1	8	?	Cardiology	
1	Caucasian	Female	[50-60)	?	3	1	1	2	?	Surgery-Neuro	
2	Caucasian	Female	[80-90)	?	1	3	7	4	MC	InternalMedicine	

In [102…
```
# Removal of unnecessary features
df_emb = df_emb.drop(columns=['admission_type_id', 'discharge_disposition_id', 'payer_code', 'medical_specialty',  'diag_1', 'diag_2', 'diag_
                              'group_diag_1', 'group_diag_2', 'group_diag_3', 'group', 'L2code', 'hdiag', 'category', 'sum_target_cg',
                              'ddisp', 'sum_target_dd', 'mspec', 'sum_target_ms', 'pcode', 'sum_target_pc', 'atype', 'sum_target_at',
                              'agegr', 'sum_target_age', 'num_hdiag', 'num_ddisp', 'num_mspec', 'num_pcode', 'num_atype', 'num_agegr'], axis=

df_emb.head()
```

Out[102…

	race	gender	age	weight	admission_source_id	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpatier
0	Caucasian	Female	[50-60)	?	1	8	77	6	33	
1	Caucasian	Female	[50-60)	?	1	2	49	1	11	
2	Caucasian	Female	[80-90)	?	7	4	68	2	23	
3	Caucasian	Female	[80-90)	?	7	3	46	0	20	
4	AfricanAmerican	Female	[30-40)	?	7	5	49	0	5	

In [103…

```
# Generation of training, validation, and test datasets
xe = df_emb.drop(columns=['TARGET'], axis=1)
ye = df_emb.TARGET
# Split dataset in 70% training, 15% validation and 15% test and devide into feature matrix x and label y
xe_train, xe_valtest, ye_train, ye_valtest = train_test_split(xe, ye, train_size=0.70, random_state=RANDOM_SEED)
xe_val, xe_test, ye_val, ye_test = train_test_split(xe_valtest, ye_valtest, test_size=0.50, random_state=RANDOM_SEED)
print('Sample sizes for training, validation and test: ', xe_train.shape, xe_val.shape, xe_test.shape)
```

```
Sample sizes for training, validation and test:  (33425, 52) (7163, 52) (7163, 52)
```

zu b):

In [104…

```
Xy_emb_train = xe_train.copy()
Xy_emb_train['TARGET'] = ye_train

LR_emb = smf.logit(formula="TARGET ~ number_inpatient + num_lab_procedures + hdiag_dim_0 + hdiag_dim_1 + ddisp_dim_0 + ddisp_dim_1 + mspec_di
                   data=Xy_emb_train).fit()

# Display a summary of the logistic regression model results
print(LR_emb.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.362729
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                 TARGET   No. Observations:                33425
Model:                          Logit   Df Residuals:                    33410
Method:                           MLE   Df Model:                           14
Date:                Mon, 28 Jul 2025   Pseudo R-squ.:                 0.06762
Time:                        08:59:37   Log-Likelihood:                -12124.
converged:                       True   LL-Null:                       -13004.
Covariance Type:            nonrobust   LLR p-value:                     0.000
============================================================================================
                               coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------------------
Intercept                   -1.3711      0.101    -13.629      0.000      -1.568      -1.174
number_inpatient             0.5625      0.026     22.019      0.000       0.512       0.613
num_lab_procedures           0.0030      0.001      3.397      0.001       0.001       0.005
hdiag_dim_0                  0.8396      0.352      2.387      0.017       0.150       1.529
hdiag_dim_1                 -1.6823      0.347     -4.850      0.000      -2.362      -1.002
ddisp_dim_0                 -2.8616      1.171     -2.445      0.014      -5.156      -0.567
ddisp_dim_1                  0.0048      1.079      0.004      0.996      -2.111       2.120
mspec_dim_0                 -1.5122      0.448     -3.375      0.001      -2.390      -0.634
mspec_dim_1                 -0.7797      0.545     -1.431      0.152      -1.847       0.288
pcode_dim_0                 -1.1832      0.464     -2.551      0.011      -2.092      -0.274
pcode_dim_1                 -0.4674      0.302     -1.550      0.121      -1.058       0.124
atype_dim_0                 -0.9342      0.414     -2.259      0.024      -1.745      -0.124
atype_dim_1                  0.0005      0.650      0.001      0.999      -1.274       1.275
agegr_dim_0                 -0.7982      0.665     -1.201      0.230      -2.101       0.505
agegr_dim_1                 -0.0708      1.027     -0.069      0.945      -2.085       1.943
============================================================================================
```

In [105…

```
# Evaluate the performance of the logistic regression model on the validation data
# and calculate the Area Under the Curve (AUC) for the ROC
auc_LR_emb = roc_auc_score(ye_val, LR_emb.predict(xe_val))

mname.append("LR_emb")
mauc.append(auc_LR_emb)

# Print the validation AUC
print(f'The validation AUC of the logistic regression model is: {auc_LR_emb:.6f}')
```

```
The validation AUC of the logistic regression model is: 0.669969
```

In task PT2, logistic regression was performed with features based on the feature importance of CatBoost. Specifically, the five features `discharge_disposition_id`, `number_inpatient`, `admission_source_id`, `num_lab_procedures`, and `admission_type_id` were used. The validation score for this model was approximately 0.6612. In the current model, significantly more features are used due to the embeddings (a total of 14). However, only a very slight increase in the corresponding AUC value can be observed.

c):

In [106…

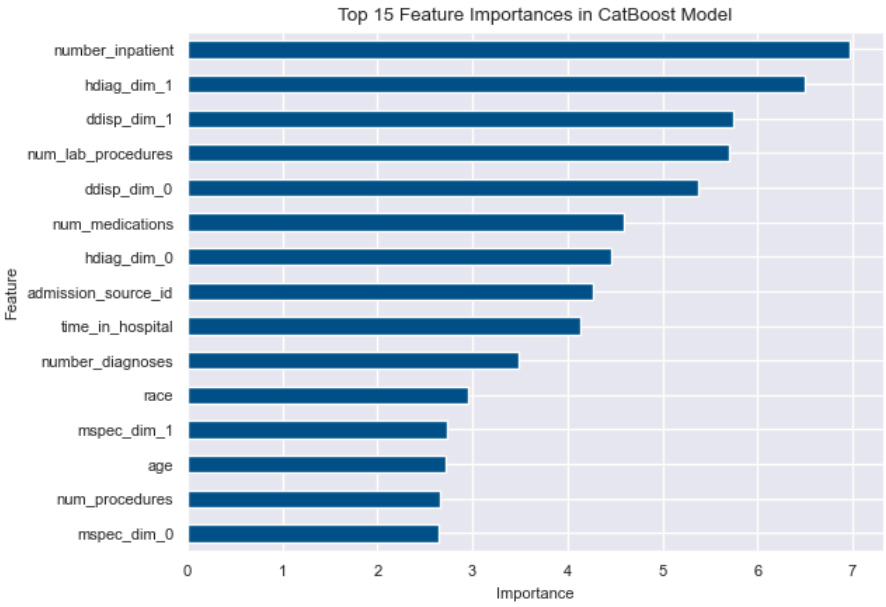
```
# Identify categorical features by data type 'object'
categorical_features_emb = list(xe_train.select_dtypes(include=['object']).columns)
```

```
# Start timer to calculate the running time of training the CatBoost model
start_time = time.time()

# Fit the CatBoostClassifier on the training data
CB_emb = CatBoostClassifier(eval_metric='AUC', random_seed=RANDOM_SEED)
CB_emb.fit(xe_train, ye_train, cat_features=categorical_features_emb, logging_level='Silent')

# Calculate and print the running time in seconds
elapsed_time_CB_emb = time.time() - start_time
print(f"Elapsed time (sec): {elapsed_time_CB_emb:.2f}")
```

```
Elapsed time (sec): 93.24
```

In [107…

```
# Extract the 15 most important features and create a Series for plotting
feature_importances = pd.Series(CB_emb.feature_importances_, index=xe_train.columns).nlargest(15)

# Plot the feature importances
feature_importances.plot(kind='barh', color=COLOR_DARK)
plt.gca().invert_yaxis()
plt.title('Top 15 Feature Importances in CatBoost Model')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



In [108…

```
# Calculate the AUC score using the model's prediction probabilities for the positive class
auc_CB_emb = roc_auc_score(ye_val, CB_emb.predict_proba(xe_val)[:, 1])

mname.append("CB_emb")
mauc.append(auc_CB_emb)

# Print the validation AUC
print(f'The validation AUC of the CatBoost model with standard hyperparameters is: {auc_CB_emb:.6f}')
```

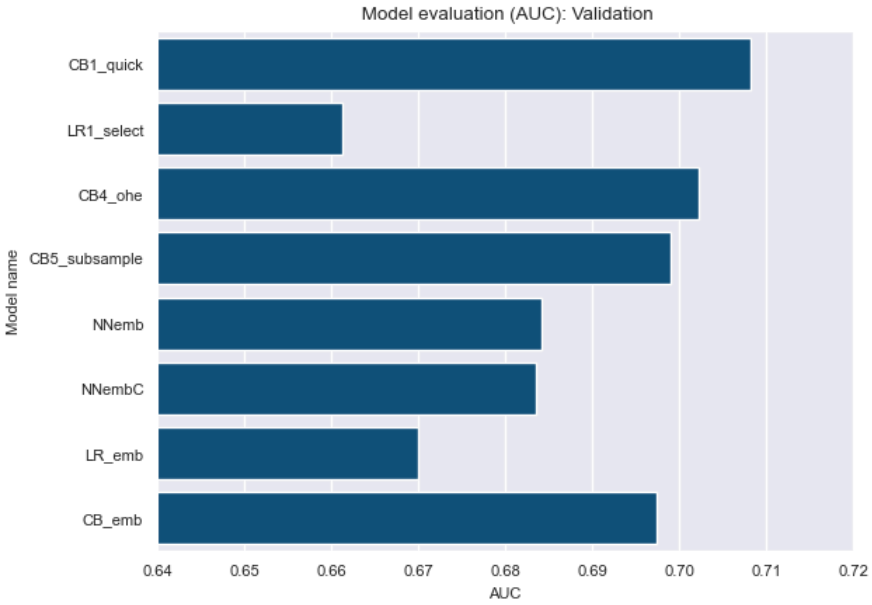
```
The validation AUC of the CatBoost model with standard hyperparameters is: 0.697473
```

The result of CatBoost taking into account the embeddings falls behind the result of the first "CB-quick." The training times are at approximately the same level.

d):

In [109…

```
plot_auc(dict,0.64,0.72, "Validation")
```



The image shows that the first executed CatBoost algorithm could not be surpassed by the models performed in this notebook, measured by AUC. This was not expected, especially since significantly more time (compared to CatBoost's training time) was invested in hyperparameter tuning. However, as seen in the graphic, the differences are minimal. Techniques such as scaling or encoding can significantly reduce the training times for tree-based methods (i.e., to a few seconds), which can be advantageous for larger datasets. Logistic regressions, which have the shortest training times, lag significantly behind. Adding embeddings to the applied CatBoost process yields similarly good results to "CB-Quick" and additionally offers the advantage of gaining new patterns or insights about the data points through embedding visualization.

Task PT7: AutoML as well as Model Evaluation and Application

Task PT7-1: [Learning Objective 5.1; 16 points]

- a) Using AutoML or any other model or model ensemble not previously used (e.g., Stacking, Blending), an attempt should be made to improve the AUC. An example of AutoML is AutoGluon, see https://auto.gluon.ai/stable/index.html. Similar to the previous approach, a training, validation, and test set should be used, and the model quality should be determined.
- b) At the end of the notebook, section 11 should be fully adjusted, and everything following it (12, Appendix) should be deleted. In section 11.1, all previously created or optimized models (without Search-CV individual models) should be evaluated. In section 11.2, a selection of at least six models should be included in the Lift Chart, and a suitable model should be chosen for subsequent probability and percentile analysis.

Proposed Solution:

a):

In [110…
```
from autogluon.tabular import TabularPredictor
# save_path = 'C:/ml/model_data/AutoGluon'
save_path = '/teamspace/studios/this_studio/PK_VADS_IMMERSION/2025/1_Erster_Ansatz/AutoGluon'
```

In [130…
```
AG1 = TabularPredictor(label='TARGET', eval_metric='roc_auc', path=save_path, verbosity=0).fit(
    Xy_raw_train, presets='best_quality', time_limit=elapsed_time_CB1*5
)

print(AG1.feature_metadata)
```

```
('category', [])  : 26 | ['race', 'age', 'weight', 'admission_type_id', 'discharge_disposition_id', ...]
('int', [])       :  8 | ['time_in_hospital', 'num_lab_procedures', 'num_procedures', 'num_medications', 'number_outpatient', ...]
('int', ['bool']) :  8 | ['gender', 'chlorpropamide', 'tolbutamide', 'miglitol', 'tolazamide', ...]
```

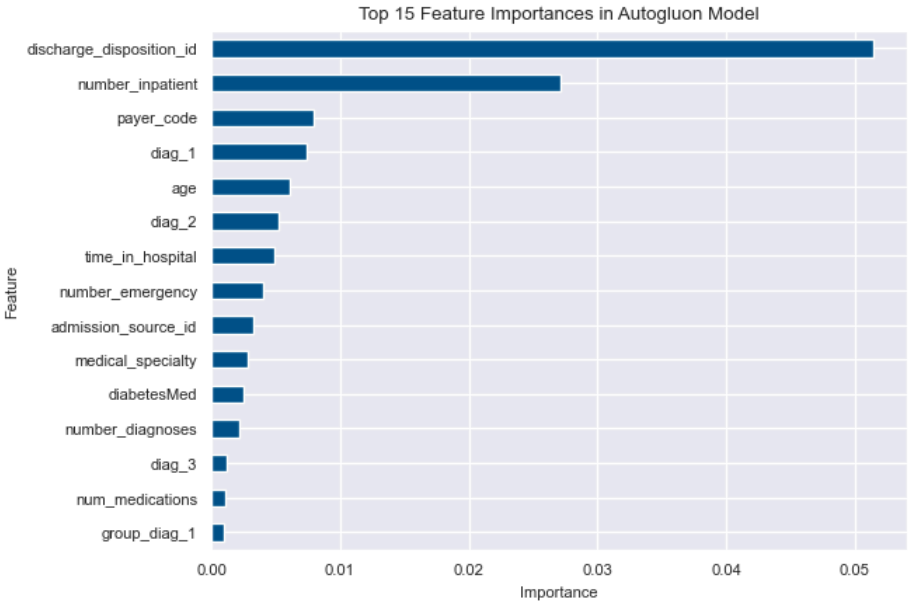
In [112…
```
fi_AG1 = AG1.feature_importance(Xy_raw_val)
fi_AG1.head(15)
```

Out[112…

	importance	stddev	p_value	n	p99_high	p99_low
discharge_disposition_id	0.051470	0.004868	0.000009	5	0.061493	0.041446
number_inpatient	0.027074	0.004722	0.000107	5	0.036796	0.017352
payer_code	0.007961	0.002790	0.001548	5	0.013706	0.002216
diag_1	0.007409	0.001639	0.000269	5	0.010783	0.004035
age	0.006097	0.002637	0.003327	5	0.011527	0.000667
diag_2	0.005207	0.002170	0.002912	5	0.009675	0.000739
time_in_hospital	0.004889	0.000975	0.000180	5	0.006897	0.002881
number_emergency	0.004061	0.001864	0.004105	5	0.007898	0.000223
admission_source_id	0.003240	0.001967	0.010575	5	0.007291	-0.000810
medical_specialty	0.002852	0.003262	0.061105	5	0.009568	-0.003864
diabetesMed	0.002522	0.004564	0.142103	5	0.011921	-0.006876
number_diagnoses	0.002140	0.001433	0.014432	5	0.005090	-0.000811
diag_3	0.001155	0.003284	0.237721	5	0.007917	-0.005606
num_medications	0.001078	0.000646	0.010132	5	0.002408	-0.000252
group_diag_1	0.000941	0.001773	0.150337	5	0.004591	-0.002708

In [113…

```
# Extract the 15 most important features and create a Series for plotting
feature_importances = pd.Series(fi_AG1.importance, index=X_raw_train.columns).nlargest(15)

# Plot the feature importances
feature_importances.plot(kind='barh', color=COLOR_DARK)
plt.gca().invert_yaxis()
plt.title('Top 15 Feature Importances in Autogluon Model')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



In [114…

```
AG1.leaderboard(Xy_raw_val)
```

Out[114…

	model	score_test	score_val	eval_metric	pred_time_test	pred_time_val	fit_time	pred_time_test_marginal	pred_time_val_marginal
0	WeightedEnsemble_L2	0.709335	0.707921	roc_auc	4.305476	6.867017	222.393423	0.005521	0.008013
1	WeightedEnsemble_L3	0.708945	0.708060	roc_auc	6.714903	11.724396	307.247430	0.005001	0.004029
2	CatBoost_BAG_L2	0.708114	0.706709	roc_auc	6.309689	10.239308	303.127975	0.147372	0.256862
3	LightGBMXT_BAG_L2	0.707491	0.702185	roc_auc	6.399297	10.291601	230.242440	0.236979	0.309155
4	CatBoost_BAG_L1	0.705858	0.703841	roc_auc	0.415258	0.473350	182.167246	0.415258	0.473350
5	NeuralNetFastAI_BAG_L2	0.704102	0.680909	roc_auc	8.715873	11.622133	238.015023	2.553556	1.639687
6	ExtraTreesGini_BAG_L2	0.703645	0.688623	roc_auc	6.566218	11.435822	229.789802	0.403901	1.453377
7	LightGBMXT_BAG_L1	0.700023	0.700309	roc_auc	0.663687	0.524139	2.465278	0.663687	0.524139
8	ExtraTreesEntr_BAG_L2	0.699809	0.690666	roc_auc	6.562529	11.463505	229.706552	0.400212	1.481059
9	RandomForestEntr_BAG_L2	0.693994	0.689504	roc_auc	6.427138	11.437454	232.940647	0.264820	1.455008
10	LightGBM_BAG_L2	0.691873	0.689956	roc_auc	6.387864	10.238271	230.233174	0.225546	0.255826
11	RandomForestGini_BAG_L2	0.691653	0.684444	roc_auc	6.488678	11.672378	232.517368	0.326360	1.689932
12	RandomForestGini_BAG_L1	0.689840	0.666698	roc_auc	0.524758	1.506777	2.790008	0.524758	1.506777
13	NeuralNetFastAI_BAG_L1	0.688213	0.649348	roc_auc	1.658042	1.596099	28.994395	1.658042	1.596099
14	RandomForestEntr_BAG_L1	0.687714	0.674098	roc_auc	0.441014	1.268171	2.659870	0.441014	1.268171
15	ExtraTreesGini_BAG_L1	0.684561	0.664138	roc_auc	0.597196	1.490468	2.094584	0.597196	1.490468
16	LightGBM_BAG_L1	0.682677	0.682165	roc_auc	0.438008	0.292747	2.293514	0.438008	0.292747
17	ExtraTreesEntr_BAG_L1	0.679172	0.663207	roc_auc	0.519828	1.434854	1.891471	0.519828	1.434854
18	XGBoost_BAG_L1	0.677629	0.647442	roc_auc	0.584097	0.434981	2.089921	0.584097	0.434981
19	KNeighborsUnif_BAG_L1	0.551008	0.531913	roc_auc	0.149853	0.490913	0.112550	0.149853	0.490913
20	KNeighborsDist_BAG_L1	0.550761	0.534956	roc_auc	0.170576	0.469948	0.113562	0.170576	0.469948

In [115…
```
y_val_AG1 = AG1.predict_proba(X_raw_val).iloc[:, 1]
y_val_AG1.head(5)  # some example predictions
```

Out[115…
```
19120    0.143584
8718     0.147167
36646    0.584534
40896    0.071326
20211    0.227711
Name: 1, dtype: float64
```

In [116…
```
# Calculate the AUC score using the model's prediction probabilities for the positive class
auc_AG1 = roc_auc_score(y_val, y_val_AG1)

# Print the validation AUC
print(f'The validation AUC of AutoGluon is: {auc_AG1:.6f}')
```

```
The validation AUC of AutoGluon is: 0.708945
```

In [117…
```
AG1.evaluate(Xy_raw_val)
```

Out[117…
```
{'roc_auc': 0.7089449400131975,
 'accuracy': 0.8715621946111964,
 'balanced_accuracy': 0.5169602469904028,
 'mcc': 0.1403841008165169,
 'f1': 0.06882591093117409,
 'precision': 0.7083333333333334,
 'recall': 0.036170212765957444}
```

In [118…
```
AG1.evaluate(Xy_raw_test)
```

Out[118…
```
{'roc_auc': 0.7115719680531518,
 'accuracy': 0.8701661315091442,
 'balanced_accuracy': 0.5160129067173133,
 'mcc': 0.12890224179681264,
 'f1': 0.06626506024096386,
 'precision': 0.6470588235294118,
 'recall': 0.03492063492063492}
```

11. Model Evaluation and Application

11.1 Model Evaluation: Validation and Test Set

Now that the modeling is complete, it is time to evaluate our models using test data and compare the results with the validation data and between the various models of interest. We will consider all tuned models.

In [119…
```
# Calulate model AUC based on test data
auc_CB1_test = roc_auc_score(y_test, CB1.predict_proba(X_raw_test)[:, 1])
auc_LR1_test = roc_auc_score(y_test, LR1.predict(X_pre_test))
```

```
# Feature scaling using StandardScaler based on training data distributions (6.1)
# X_test = pd.DataFrame(scaler.transform(X_test), columns = feature_names)
auc_CB4_test = roc_auc_score(y_test, CB4.predict_proba(X_test)[:,1])
auc_CB5_test = roc_auc_score(y_test, CB4.predict_proba(X_test)[:,1])
auc_CB6_test = roc_auc_score(y_test, CB6.predict_proba(X_test)[:,1])
auc_LGB_test = roc_auc_score(y_test, LGB.predict_proba(X_test)[:,1])
auc_XGB_test = roc_auc_score(y_test, XGB.predict_proba(X_test)[:,1])

# NNemb, NNembC
input_test_data = [xn_test[num_features]] + [np.asarray(xn_test[feature]) for feature in num_emb_features]
y_test_NNemb = NNemb.predict(input_test_data)[:,0]
auc_NNemb_test = roc_auc_score(y_test, y_test_NNemb)
y_test_NNembC = NNembC.predict(input_test_data)[:,0]
auc_NNembC_test = roc_auc_score(y_test, y_test_NNembC)

# auc_NNemb_test = roc_auc_score(yn_test, yn_test_pred)
auc_LR_emb_test = roc_auc_score(ye_test, LR_emb.predict(xe_test))
auc_CB_emb_test = roc_auc_score(ye_test, CB_emb.predict_proba(xe_test)[:, 1])
auc_AG1_test = roc_auc_score(y_test, AG1.predict_proba(X_raw_test).iloc[:, 1])
```

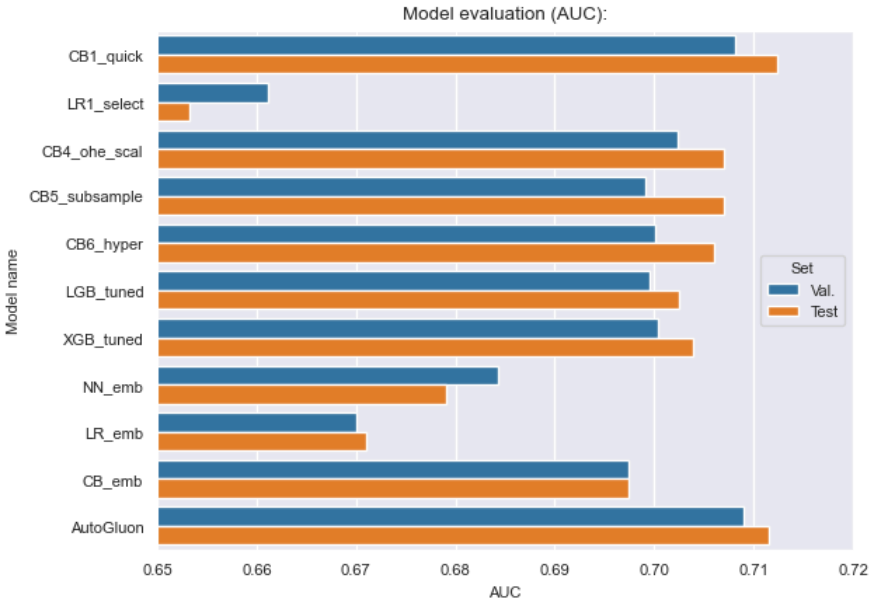
```
224/224 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
224/224 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

In [120…

```
# Data structures for model names, val/test-sample and AUC
mdict = {'Model name': [], 'Set': [], 'AUC': []}

# List of models and their AUC scores on validation and test sets
models_auc = [
    ("CB1_quick",       auc_CB1, auc_CB1_test),
    ("LR1_select",      auc_LR1, auc_LR1_test),
    ("CB4_ohe_scal",    auc_CB4, auc_CB4_test),
    ("CB5_subsample",   auc_CB5, auc_CB5_test),
    ("CB6_hyper",       auc_CB6, auc_CB6_test),
    ("LGB_tuned",       auc_LGB, auc_LGB_test),
    ("XGB_tuned",       auc_XGB, auc_XGB_test),
    ("NN_emb",          auc_NNemb, auc_NNemb_test),
    ("LR_emb",          auc_LR_emb, auc_LR_emb_test),
    ("CB_emb",          auc_CB_emb, auc_CB_emb_test),
    ("AutoGluon",       auc_AG1, auc_AG1_test),
]

# Populate the dictionary by looping over the models and their AUC scores
for model_name, auc_val, auc_test in models_auc:
    mdict['Model name'].extend([model_name] * 2)  # Model name twice for Val. and Test
    mdict['Set'].extend(["Val.", "Test"])
    mdict['AUC'].extend([auc_val, auc_test])
```

In [121…

```
df_eval = pd.DataFrame(mdict)
plt.title("Model evaluation (AUC):")
sns.barplot(data=df_eval, x="AUC", y="Model name",hue="Set")
plt.xlim(0.65, 0.72)
plt.show()
```



In contrast to the CSN notebook, not all models perform better on the test data. It can also be seen here that the "Quick-CB" model already achieves very good AUC scores on the training and test datasets. Other individual approaches, such as XGB or LGMB, come close but cannot reach these values (even with the use of hyperparameter tuning). To achieve a higher AUC score than "Quick-CB," further efforts such as AutoML (in this case: AutoGluon) are required, where not individual models but combinations of models are employed.

11.2 High risk prediction

In subsection 11.1, we identified the model with the highest Area Under the Receiver Operating Characteristic Curve (AUC) as the best model based on all readmissions ranging from low to high risk. Since the high-risk area typically receives the most attention for health-related reasons, the performance of the models in this area is crucial for application. The Lift Chart is suitable for this evaluation. A cumulative Lift Chart illustrates the improvement a model offers compared to a random estimate and measures the change in the form of a Lift Score.

In [122…

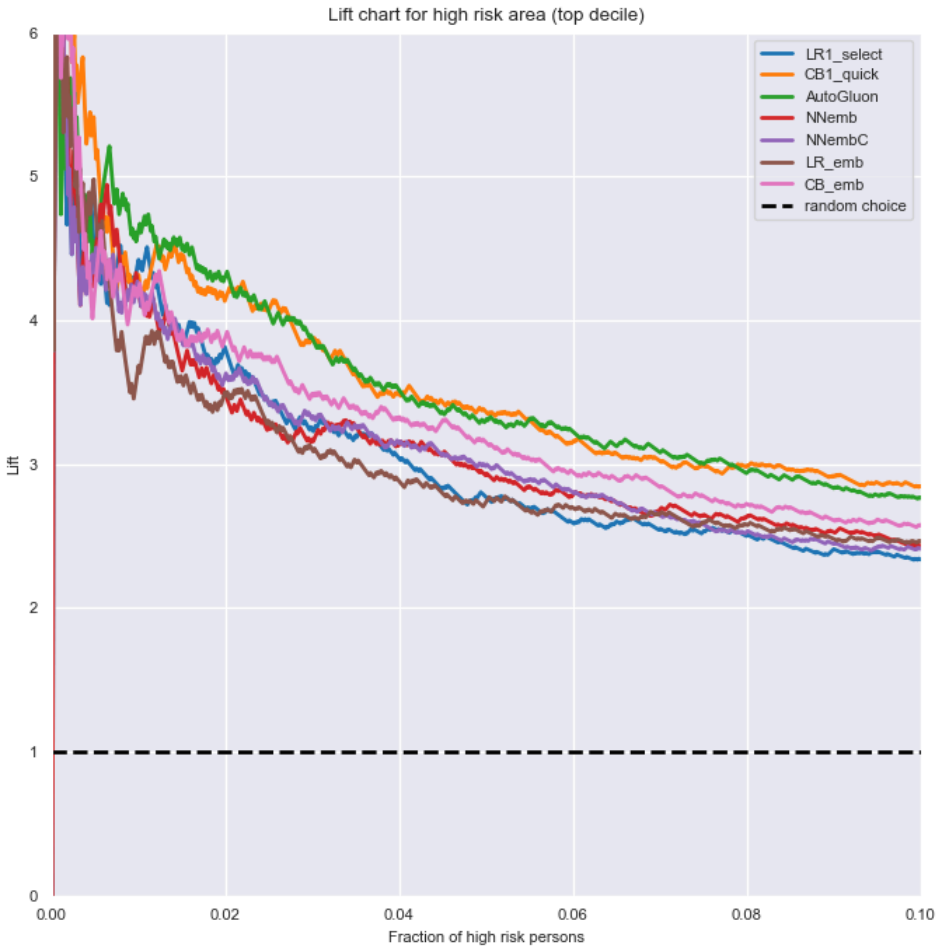
```
# Make preditions for test data
y_test_LR1 = LR1.predict(X_pre_test)
y_test_CB1 = CB1.predict_proba(X_raw_test)[:, 1]
y_test_AG1 = AG1.predict_proba(X_raw_test).iloc[:, 1]

y_test_LR_emb = LR_emb.predict(xe_test)
y_test_CB_emb = CB_emb.predict_proba(xe_test)[:, 1]
```

In [123…

```
# Core of scikit-plot function cumulative_gain_curve (necessary, since scypi release 1.12.0 broke compatibility with scikit-plot version 0.3.
# Source: https://github.com/reiinakano/scikit-plot/blob/26007fbf9f05e915bd0f6acb86850b01b00944cf/scikitplot/helpers.py
def cumulative_gain_curve1(y_true, y_score):
    """This binary classification function generates the points necessary to plot the Cumulative Gain"""
    y_true, y_score = np.asarray(y_true), np.asarray(y_score)
    # make y_true a boolean vector
    y_true = (y_true == 1)
    sorted_indices = np.argsort(y_score)[::-1]
    y_true = y_true[sorted_indices]
    gains = np.cumsum(y_true)
    percentages = np.arange(start=1, stop=len(y_true) + 1)
    gains = gains / float(np.sum(y_true))
    percentages = percentages / float(len(y_true))
    gains = np.insert(gains, 0, [0])
    percentages = np.insert(percentages, 0, [0])
    return percentages, gains

# Calculate cumulative gains for lift chart
def calculate_gains_and_percentages(model_predictions, y_true):
    """Calculate gains and percentages for a given model's predictions."""
#    percentages, gains = cumulative_gain_curve1(y_true, model_predictions[:, 1])
    percentages, gains = cumulative_gain_curve1(y_true, model_predictions)
    gains_adjusted = gains[1:] / percentages[1:]  # Adjust gains starting from the second element
    return percentages[1:], gains_adjusted
```

In [124…

```
# Select models and prepare true and predicted values
y_true = np.array(np.ravel(y_test))
model_predictions = {
    'LR1_select': np.array(y_test_LR1),
    'CB1_quick': np.array(y_test_CB1),
    'AutoGluon': np.array(y_test_AG1),
    'NNemb': np.array(y_test_NNemb),
    'NNembC': np.array(y_test_NNembC),
    'LR_emb': np.array(y_test_LR_emb),
    'CB_emb': np.array(y_test_CB_emb)
}

# Initialize lift chart
fig, ax = plt.subplots(figsize=(8, 8))
ax.set_title("Lift chart for high risk area (top decile)")
ax.set_xlabel('Fraction of high risk persons')
ax.set_ylabel('Lift')
plt.xlim(0, 0.1)
plt.ylim(0, 6.0)

# Calculate and plot lift score for each model
for label, predictions in model_predictions.items():
    percentages, gains_adjusted = calculate_gains_and_percentages(predictions, y_true)
    ax.plot(percentages, gains_adjusted, lw=2, label=label)

# Plot random choice baseline
ax.plot([0, 1], [1, 1], 'k--', lw=2, label='random choice')
ax.legend(loc='upper right')
plt.show()
```



The cumulative Lift Chart shows the factor (Lift Score) by which the model can improve the "hit rate." In our case, there is no clear winner model for the top decile. All Gradient Tree Boosting models have a similar Lift Score. Therefore, we can use our first "quick and simple" baseline model for defining the high-risk area. In the following, we focus on the top decile and create a list of individuals who have a high probability of readmission. Since we know the true value for our test data, we compare it with the predicted values.

In [125…

```
# Merge predicted an true values into a DataFrame
df = pd.DataFrame({
    'PredProbability': y_test_CB1,
    'TrueValue': y_test
})

# Sort by Probability in descending order
df_sorted = df.sort_values(by='PredProbability', ascending=False)
df_sorted.head()
```

Out[125…

	PredProbability	TrueValue
34138	0.889983	1
33042	0.871476	1
33582	0.824957	1
45194	0.813839	1
35906	0.813183	1

In [126…

```
# Creating a 'Percentile' column based on quantile-based discretization into 100 bins
df_sorted['Percentile'] = pd.qcut(df_sorted['PredProbability'], 100, labels=range(1, 101))

# Calculate means of predicted and true values for the top 10 percentiles
percentile_means = df_sorted.groupby('Percentile', observed=True).mean().tail(10)

# Printing the results for the top 10 percentiles
print(percentile_means[::-1])  # This reverses the order to start with the highest probabilities
```

```
            PredProbability  TrueValue
Percentile
100                0.630715   0.569444
99                 0.467155   0.527778
98                 0.410477   0.436620
97                 0.373865   0.305556
96                 0.340553   0.375000
95                 0.314294   0.281690
94                 0.292272   0.291667
93                 0.274749   0.366197
92                 0.257833   0.305556
91                 0.245951   0.291667
```

Although the predicted probabilities here also seem somewhat too high, the ranking of the percentiles works very well and can be used to decide on the actions to be taken.

In [127…

```
# Timing Notebook:
elapsed_time_nb = time.time() - start_time_nb
print(f"Elapsed time entire notebook (min): {elapsed_time_nb/60:.2f}")
```

```
Elapsed time entire notebook (min): 73.15
```