VADS_Immersion_2021_Lösungsvorschlag_final

March 9, 2021

0.1 Prüfung im Vertiefungswissen

1 Actuarial Data Science Immersion

1.1 gemäß der Prüfungsordnung 4

1.2 der Deutschen Aktuarvereinigung e.V.

1.3 Zeitraum: 15.04.2021 bis 15.05.2021

Hinweise: - Die Gesamtpunktzahl beträgt 180 Punkte. Die Prüfung ist bestanden, wenn mindestens 90 (50 %) Punkte erreicht werden. - Bitte prüfen Sie das Ihnen vorliegende Dokument auf Vollständigkeit. Insgesamt sind 31 Aufgaben enthalten. - Alle Antworten sind zu begründen und Lösungswege müssen nachvollziehbar sein. - Mit der Einreichung eines Ergebnisnotebooks erklärt die zu prüfende Person, dass das vorliegende Ergebnisbericht inklusive Code- und Beschreibungsanteilen eigenständig erzeugt worden ist. Verstöße gegen diese Grundlagen können vom Prüfungsausschuss sanktioniert werden und zum Ausschluss von der Prüfung führen. - Die Struktur des Notebooks muss beibehalten werden. Nutzen Sie für Ihre Antworten die freien Zelle(n) die einem Frageblock folgen.

2 0. Einführende Informationen

2.0.1 Erläuterungen zur Datengrundlage:

Zusätzlich zu diesem Dokument sollten Ihnen zwei Datensätze vorliegen: cs_train_pk1.csv und cs_test_pk1.csv. Wie die Namen bereits andeuten, soll der erste Datensatz zum Training und der zweite Datensatz zum Testen verwendet werden.

Die Datensätze enthalten verschiedene Schadenmeldungen. Dabei beinhalten die verschiedenen Features (z.B. cat1, cat2, ...) Informationen zum jeweiligen Schaden. Die Zielvariable ist loss.

2.0.2 Inhaltsangabe:

Mit den Aufgaben innerhalb der folgenden Abschnitte sollen die Daten analysiert werden:

1. Allgemeines und Datenimport
   1.1. Allgemeine Fragen
   1.2. Trainings-Daten einlesen und einen ersten Überblick erhalten
2. Explorative Datenanalyse

2.1. Explorative Datenanalyse der numerischen Merkmale

2.2. Explorative Datenanalyse der kategoriellen Merkmale

2.3. Daten für die Modellierung vorbereiten

3. Modelle

3.1. Entscheidungsbaumbasierte Modellensembles

`3.1.1. Random-Forest-Verfahren`

`3.1.2. Gradient-Boosting-Verfahren`

3.2. Regularisierte lineare Modelle

3.3. Vorwärtsgerichtetes Neuronales Netz mit Keras/TensorFlow

4. Blending und Scoring

2.0.3 Anmerkungen zur Lösung:

Hierzu soll ein *Jupyter*-Notebook oder ein *rmarkdown*-Notebook entsprechend erweitert werden. Die zu verwendenden Programmiersprachen sind *Python* oder *R*. Dabei muss sichergestellt werden, dass aktuelle Software-Versionen verwendet werden (Python 3.7 oder höher, R 3.6 oder höher).

Die Aufgabentexte sollen in das Notebook übernommen und dort in der vorgegebenen Reihenfolge beantwortet werden. Ab Frage 4 sind Codes anzugeben, fehlerfrei auszuführen und die Aufgabenerledigung ist folgendermaßen zu strukturieren: 1. Markdown: Aufgabentext 2. Markdown: Kurze Beschreibung dessen, was im Code gemacht wird. 3. Der entsprechende, #-kommentierte R- oder Python-Code samt Output, ggf. mehrfach. 4. Markdown: Erläuterung, Interpretation und Kommentierung gemäß Aufgabenstellung, Beantwortung der Fragen.

Zusätzlich soll das Notebook sinnvoll strukturiert werden, z.B. durch Kapitelüberschriften, Abschnitte und ein Inhaltsverzeichnis. Alle genannten Verfahren sind konkret und sachgerecht zusammen mit der ggf. erforderlichen Datenaufbereitung (z.B. Kodierungen, Skalierung, Transformation) auf den Datensatz anzuwenden.

Das vorgegeben Gütemaß ist der mittlere absolute Fehler ("mean absolute error", MAE). Die Güte jedes Modells soll auf Basis einer für alle Modelle einheitlichen Validierungsstichprobe ermittelt, angezeigt und kommentiert werden.

Die Verwendung von Code-Auszügen aus online verfügbaren Notebooks, die für andere Datensätze entwickelt wurden, ist unter Angabe der jeweiligen Quelle zulässig.

Die vorliegende Lösung wurde in Python erstellt. Im folgenden Lösungsvorschlag verwendete Quellen: 1. Santhosh Sharma Anantharamu, https://www.kaggle.com/sharmasanthosh/exploratory-study-on-ml-algorithms/data 2. nminus1, https://www.kaggle.com/nminus1/allstate-eda-python 3. Jared Turkewitz, https://www.kaggle.com/jturkewitz/allstate-eda-cont-features 4. Achal, https://www.kaggle.com/achalshah/allstate-feature-analysis-python 5. Faron, https://www.kaggle.com/mmueller/yet-another-xgb-starter 6. https://datascience.stackexchange.com und https://stats.stackexchange.com/

3 1. Allgemeines und Datenimport

3.1 1.1 Allgemeine Fragen

3.1.1 A-01: Welche grundsätzliche gute Praxis sollte beim Arbeiten mit Jupyter Notebooks beachtet werden? (Lernziele 3.2.3, 3.2.4) - [2 Punkte]

- Beim Arbeiten mit Jupyter Notebooks sollte mit einem Programm zur Versionsverwaltung gearbeitet werden.
- Python Code Konventionen sollten eingehalten werden (z.B. PEP8). Daneben kann es projektspezifische Konventionen geben.

3.1.2 A-02: Wie kann erreicht werden, dass das eingereichte Notebook reproduzierbar lauffähig ist? (Lernziele 3.2.3, 3.2.4, 3.4.5) - [3 Punkte]

- Versionskontrolle: Data Science-Umgebungen erlauben es üblicherweise, sog. Environments zu erstellen, die konkret miteinander kompatible Pakete enthält. Diese können in der Praxis bspw. als *Docker-Container* oder *virtuelle Maschine* versioniert werden.
- Beim Arbeiten mit einem Versionsverwaltungsprogramm (z.B. Git) sollte im Entwicklungsprozess eine Text-Datei requirement.txt mit versioniert werden, die alle verwendeten Pakete der Environment auflistet und vom Paketmanager automatisiert verarbeitet werden kann.
- Reproduzierbarkeit: Sobald Zufallsdaten zum Einsatz kommen, ist konsequentes Seeding erforderlich. In Python empfiehlt es sich, mit numpy.random.RandomGenerator zu arbeiten, um auch Instanz-übergreifend (und ggf. verteilt) Modellierungen reproduzieren zu können.
- Dokumentation: Es ist gute Praxis, auch verworfene Modellierungen im Notebook zu lassen, mindestens um sich zu vergewissern, warum manch eine Vorgehensweise nicht tauglich ist. Darüber hinaus gilt dies auch, wenn ein RandomState-Objekt verwendet wird, es gehört zur Reproduzierbarkeit dazu, auch untaugliche Berechnungen zu reproduzieren. Überdies ist ein ausführlicher Kommentar für den Transfer der Erkenntnisse unabkömmlich.

3.1.3 A-03: Bei den zugrunde liegenden Daten handelt es sich um einen synthetisierten Datensatz aus dem Bereich der Versicherungsbranche. Er wurde in Anlehnung an öffentlich verfügbare Datensätze, zu denen online zahlreiche Notebooks verfügbar sind, erzeugt. Für die Bearbeitung der Aufgaben ist die Sparte und der konkrete Verwendungszweck im Folgenden irrelevant, das letztliche "Ziel" der Aufgaben besteht darin, die enthaltene Zielgröße möglichst präzise zu prognostizieren. Der Datensatz ist somit im Hinblick auf Einhaltung der kartellrechtlichen Grundlagen unverdächtig. Welche Rechtsnormen oder Vorgaben finden bei echten Daten Anwendung, bevor die Entscheidung getroffen werden kann, dass diese für eine Modellierung herangezogen werden können? (Lernziele 1.1.1, 1.1.2) - [5 Punkte]

- Datenschutz: DSGVO falls Personenbezug vorliegt
  - Einwilligung oder sonstiger Grund (lt. DSGVO bspw. "berechtigtes Interesse" oder "Erfordernis der Vertragserfüllung") muss vorliegen.
  - Datensatz muss fehlerfrei sein, also keine offensichtlichen Mängel enthalten.
- Kartellrecht: Datensatz darf keine Rückschlüsse auf Marktentscheidungen einzelner Versicherungsgesellschaften erlauben, z.B. falls ein Benchmark-Portfolio vorliegt.

- Berufsständische Grundsätze: Im Wesentlichen Replikation der Datenschutzrichtlinie. Zusätzlich: Keine unethische Verwendung, selbst falls sie legal wäre.

3.2 1.2. Trainingsdaten einlesen und einen ersten Überblick erhalten

3.2.1 A-04: Bevor die eigentliche Verarbeitung beginnt, sollen die notwendigen Programmbibliotheken aufgerufen/importiert sowie der Datensatz aus der Datei `cs_train.csv` eingelesen und die ersten Datensätze (vollständig) angezeigt werden. (Lernziel 3.4.5) - [3 Punkte]

[7]:
```
# Import nötiger Pakete
import sys
print(sys.version)
import time

# Unterdrücke unnötige Warnungen
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np

# Plotting Bibliotheken
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 12})
from matplotlib.ticker import MaxNLocator
import pylab as p
import seaborn as sns

# Datenaufbereitung
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

# Modellierungsbilbiotheken
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor
from sklearn.linear_model import Lasso, ElasticNet
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from tensorflow.keras.callbacks import EarlyStopping
from keras.wrappers.scikit_learn import KerasRegressor
```

```
3.7.9 (default, Aug 31 2020, 17:10:11) [MSC v.1916 64 bit (AMD64)]
Using TensorFlow backend.
```

[8]:
```
# Einlesen der Trainingsdaten
df = pd.read_csv("../input/claimseverity/cs_train_pk1.csv")

# Zeige alle Spalten. Verberge keine
pd.set_option('display.max_columns', None)

# Zeige die ersten 5 Zeilen der Daten um einen Überblick zu gewinnen
df.head(5)
```

[8]:

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	cat10	cat11	cat12	cat13
0	1.0	A	A	A	A	A	A	A	A	B	A	A	B	A
1	2.0	A	A	B	A	A	A	A	A	B	A	A	A	A
2	3.0	A	A	A	A	A	A	A	A	B	B	A	A	A
3	5.0	A	A	A	A	B	A	A	A	B	A	A	B	B
4	7.0	A	A	A	A	A	A	A	A	B	A	A	A	A

	cat14	cat15	cat16	cat17	cat18	cat19	cat20	cat21	cat22	cat23	cat24	cat25
0	A	A	A	A	A	A	A	A	B	A	A	A
1	A	A	B	A	A	A	A	A	B	A	A	A
2	A	A	A	A	A	A	A	A	A	A	A	A
3	A	A	A	A	A	A	A	A	A	A	A	A
4	A	A	A	A	A	A	A	A	A	A	A	A

	cat26	cat27	cat28	cat29	cat30	cat31	cat32	cat33	cat34	cat35	cat36	cat37
0	A	A	A	A	A	A	A	A	A	A	A	A
1	A	A	A	A	A	A	A	A	A	A	A	A
2	A	A	A	A	A	A	A	A	A	A	A	A
3	A	A	A	A	A	A	A	A	A	A	B	A
4	A	A	A	A	A	A	A	A	A	A	A	A

	cat38	cat39	cat40	cat41	cat42	cat43	cat44	cat45	cat46	cat47	cat48	cat49
0	A	A	A	A	A	A	A	H	C	A	B	B
1	A	A	A	A	A	A	B	A	C	B	B	C
2	A	A	A	A	A	A	A	A	C	A	B	B
3	A	A	A	A	A	A	A	A	C	B	B	B
4	A	A	A	A	A	A	A	A	C	A	B	C

	cat50	cat51	cat52	cat53	cat54	cat55	cat56	cat57	cat58	cat59	cat60	cat61
0	A	B	A	B	A	A	D	B	D	D	B	A
1	A	C	C	C	B	A	C	D	D	D	D	C
2	A	B	C	B	A	A	A	B	D	C	B	B
3	A	B	C	B	B	A	A	D	D	D	B	B
4	A	B	C	C	A	A	A	B	D	D	A	B

```
  cat62 cat63 cat64 cat65 cat66 cat67 cat68 cat69 cat70 cat71 cat72 cat73  \
0     A     A     B     A     B     I     A     B     E     G     D     F
1     A     A     G     A     C     C     E     A     E     C     D     G
2     A     A     B     A     C     C     B     B     A     B     C     C
3     A     A     B     B     B     D     D     B     A     H     A     F
4     A     A     F     A     B     B     A     A     H     E     B     A

  cat74 cat75     cont1     cont2     cont3     cont4     cont5     cont6  \
0     B    AM  0.308443  0.489699  0.637503  0.836977  0.475581  0.226992
1     B     U  0.567984  0.330670  0.331552  0.707498  0.695888  0.389243
2    AG    AT  0.529221  0.491316  0.211282  0.578662  0.292594  0.377739
3     B     H  0.478143  0.467261  0.194507  0.709301  0.217608  0.679414
4    AG     Y  0.727565  0.594485  0.787480  0.794125  0.150764  0.550874

      cont7     cont8     cont9    cont10         loss
0  0.378503  0.622195  0.408625  0.463387  3095.350932
1  0.381750  0.816750  0.476925  0.273270  7715.360600
2  0.582979  0.569231  0.443441  0.464276  5753.431020
3  0.581600  0.422483  0.344825  0.541706  5738.695439
4  0.535751  0.553705  0.729840  0.540263  7744.579266
```

3.2.2 A-05: Wie viele Schadenfälle und Merkmale hat der Datensatz? (Lernziele 3.3.1, 3.4.1) - [1 Punkt]

[9]:
```
# Größe des DataFrames ermitteln
print(df.shape)
```

```
(70908, 87)
```

70907 Schadenfälle mit 87 Merkmalen

3.2.3 A-06: Das Merkmal 'id' soll entfernt werden. Darüber hinaus ist zu prüfen, ob und wenn ja, wie viele Werte fehlen. Welche Schlussfolgerung ergibt sich daraus? (Lernziele 3.3.2, 3.4.4) - [1 Punkt]

[4]:
```
# Spalte 'id' entfernen
df = df.iloc[:,1:]
# Anzahl fehlender Werte ermitteln
df.isna().sum().sum()
```

[4]:
```
0
```

Der Datensatz ist vollständig. Er kann ohne den Einsatz eines Imputationsverfahrens für die statistische Auswertung und ML-Verfahren verwendet werden.

4 2. Explorative Datenanalyse

4.1 2.1. Explorative Datenanalyse der numerischen Merkmale

4.1.1 A-07: Für numerischen Merkmalen des Datensatzes sind die Wertebereiche festzustellen. Existiert eine Verteilungsschiefe und wie sieht diese, falls vorhanden, aus? (Lernziele 3.3.2, 3.4.4) - [5 Punkte]

[11]:
```
# Statistische Beschreibung
print(df.describe())
```

```
                 id         cont1         cont2         cont3         cont4  \
count  70907.000000  70907.000000  70907.000000  70907.000000  70907.000000
mean   50552.640261      0.450281      0.459453      0.446264      0.487557
std    29203.838170      0.206353      0.151869      0.163795      0.243236
min        1.000000      0.036459      0.012442      0.034804      0.067914
25%    25258.500000      0.303715      0.353774      0.325756      0.269121
50%    50592.000000      0.405016      0.445328      0.446823      0.402488
75%    75953.500000      0.587363      0.553483      0.567774      0.738359
max   101078.000000      0.992755      0.986016      0.987106      0.949230

              cont5         cont6         cont7         cont8         cont9  \
count  70907.000000  70907.000000  70907.000000  70907.000000  70907.000000
mean       0.474755      0.466176      0.474317      0.465831      0.426053
std        0.156561      0.146700      0.130670      0.128576      0.105481
min        0.007233      0.044013      0.016279      0.061179      0.030567
25%        0.358909      0.360336      0.387646      0.378453      0.354258
50%        0.467841      0.474298      0.483670      0.474885      0.422406
75%        0.583034      0.563413      0.563585      0.556459      0.493927
max        0.978350      0.984207      0.999306      0.889366      1.119588

             cont10          loss
count  70907.000000  70907.000000
mean       0.456186   3530.340246
std        0.075859   3062.750087
min        0.030757     62.928335
25%        0.412944   1501.351317
50%        0.451459   2598.337084
75%        0.498148   4556.283617
max        0.918834  64742.003307
```

[12]:
```
# Verteilungsschiefe
print(df.skew())
```

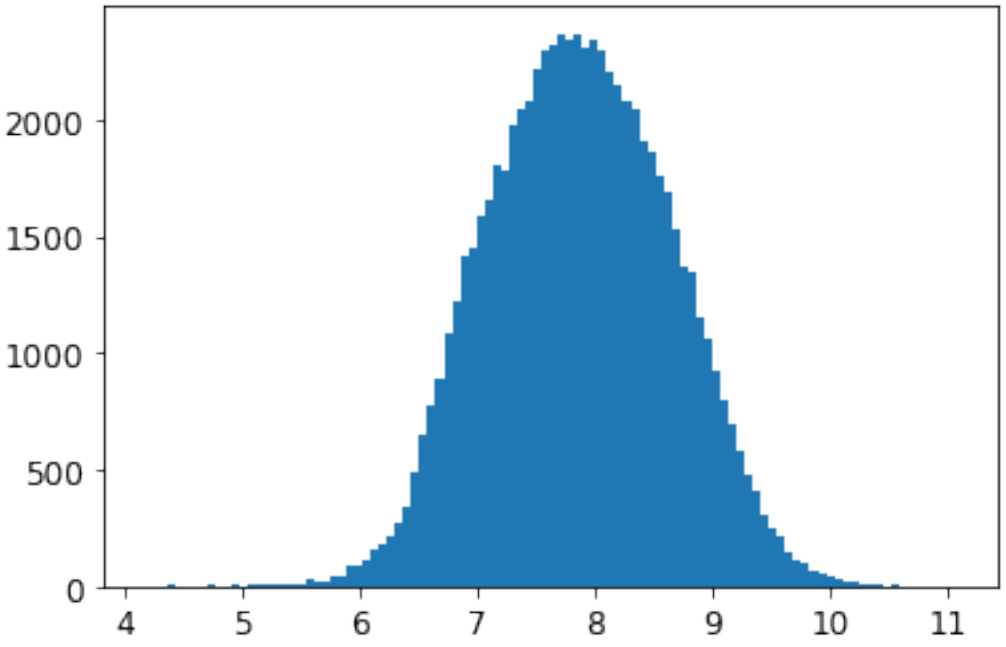
```
id       -0.003078
cont1     0.555085
cont2     0.334507
cont3    -0.002753
cont4     0.277332
```

```
cont5     0.212498
cont6    -0.030149
cont7    -0.085000
cont8    -0.160175
cont9     0.246779
cont10    0.383475
loss      2.753682
dtype: float64
```

Alle numerischen Merkmale weisen eine asymmetrische Verteilung auf. Bei Merkmal "cont3" ist diese am geringsten ausgeprägt, sie ist annähernd symmetrisch. Die Verteilungsschiefe ist bei dem Merkmal "loss" am größten ausgeprägt. Die Verteilung ist stark rechtsschief.

4.1.2 A-08: Die Zielgröße soll geeignet logarithmisch transformiert werden. Wieso ist das nötig, bzw. sinnvoll? Welche Folgen kann diese Vorgehensweise auf die prognostizierten Werte haben? (Lernziele 3.3.2, 3.4.4) - [5 Punkte]

[13]:
```
# Die log1p Funktione wendet log(1+x) auf alle Elemente einer Spalte an
df["loss"] = np.log1p(df["loss"])

# Visualisiere die transformierte Spalte
plt.hist(df.loss, bins=100)
plt.show()
```



Die logarithmische Transformation bietet sich für "fat-tailed" Verteilungen an. Sie sorgt zum einen dafür, dass die Merkmalsausprägungen sich in den selben Größenordnung bewegen. Viele ML-Methoden sind nicht skaleninvariant (ein Gegenbeispiel sind Entscheidungsbäume) und Lern- und

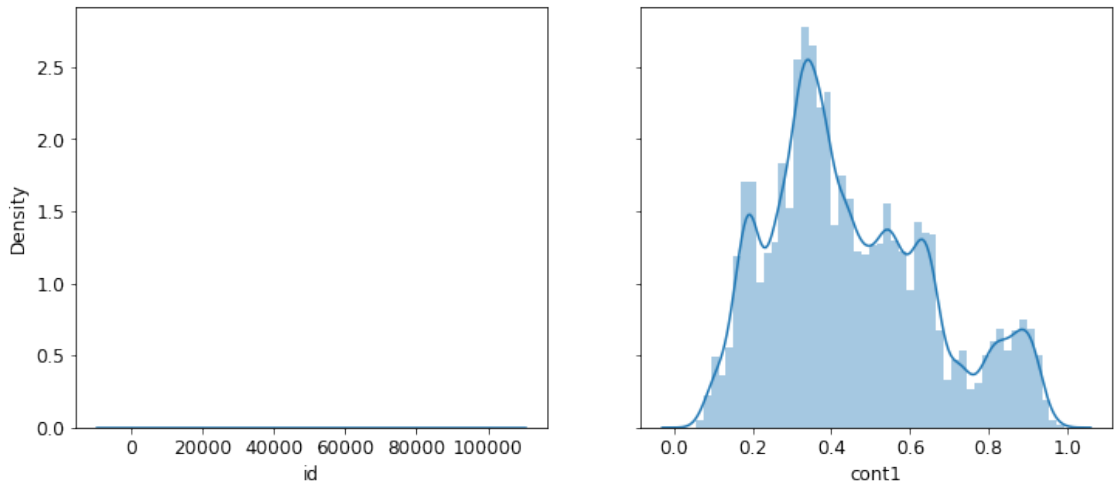
Optimierungsalgorithmen, so auch das Gradientenabstiegsverfahren, performen erheblich besser bei Merkmalsausprägungen derselben Größenordnung. Zum anderen kann sich die stark schiefe Verteilung durch die Transformation einer "Normalverteilung" annähern. Dies ermöglicht gegebenenfalls die Verwendung statistischer Methoden für die Normalverteilung bei der weiteren Datenanalyse.

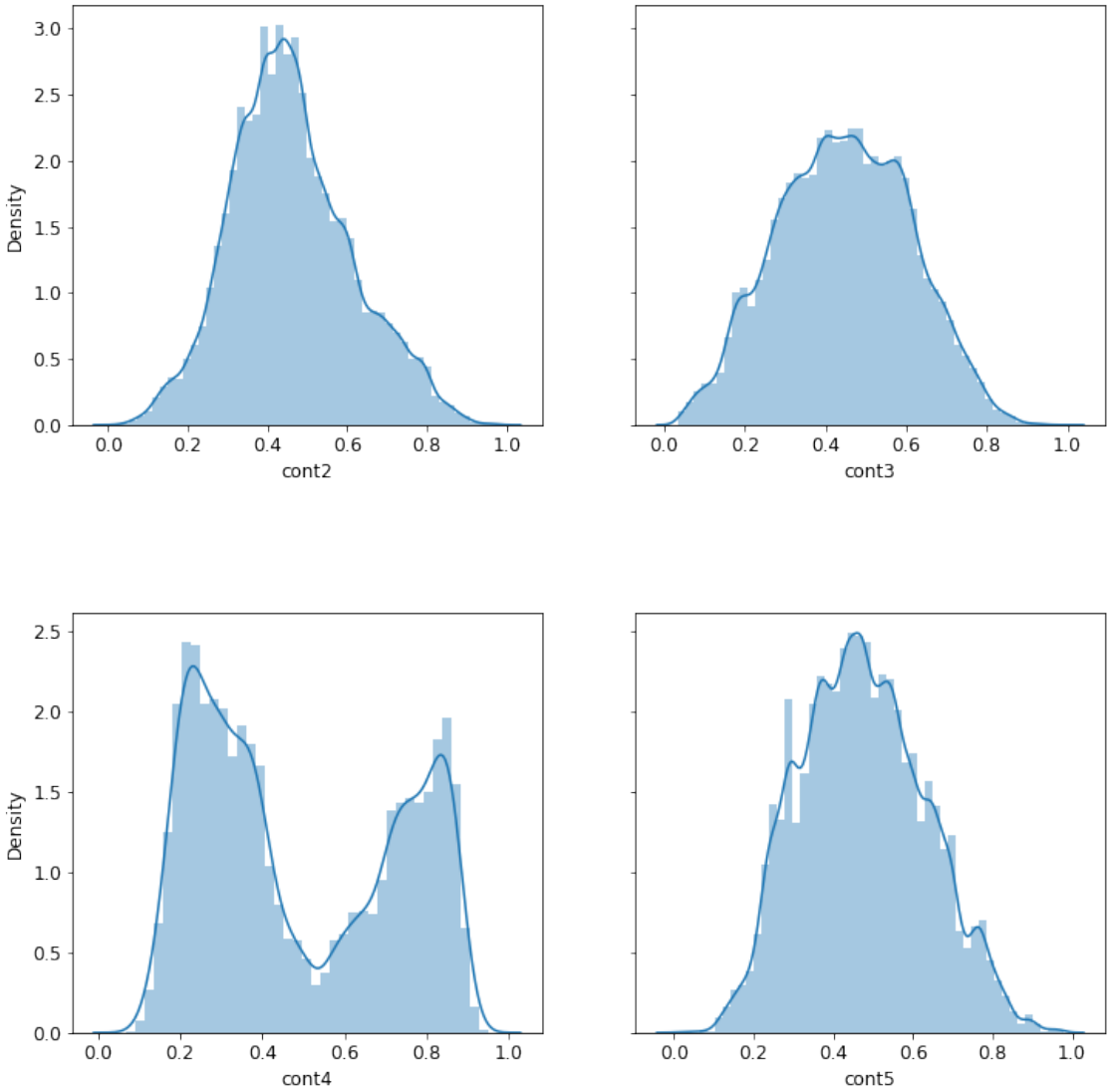
4.1.3 A-09: Es soll im Folgenden die Verteilungsdichte der numerischen Merkmale visualisiert werden. Welche konzeptionellen Möglichkeiten dazu gibt es und wie unterscheiden sich diese? (Lernziele 3.3.2, 3.4.4) - [5 Punkte]

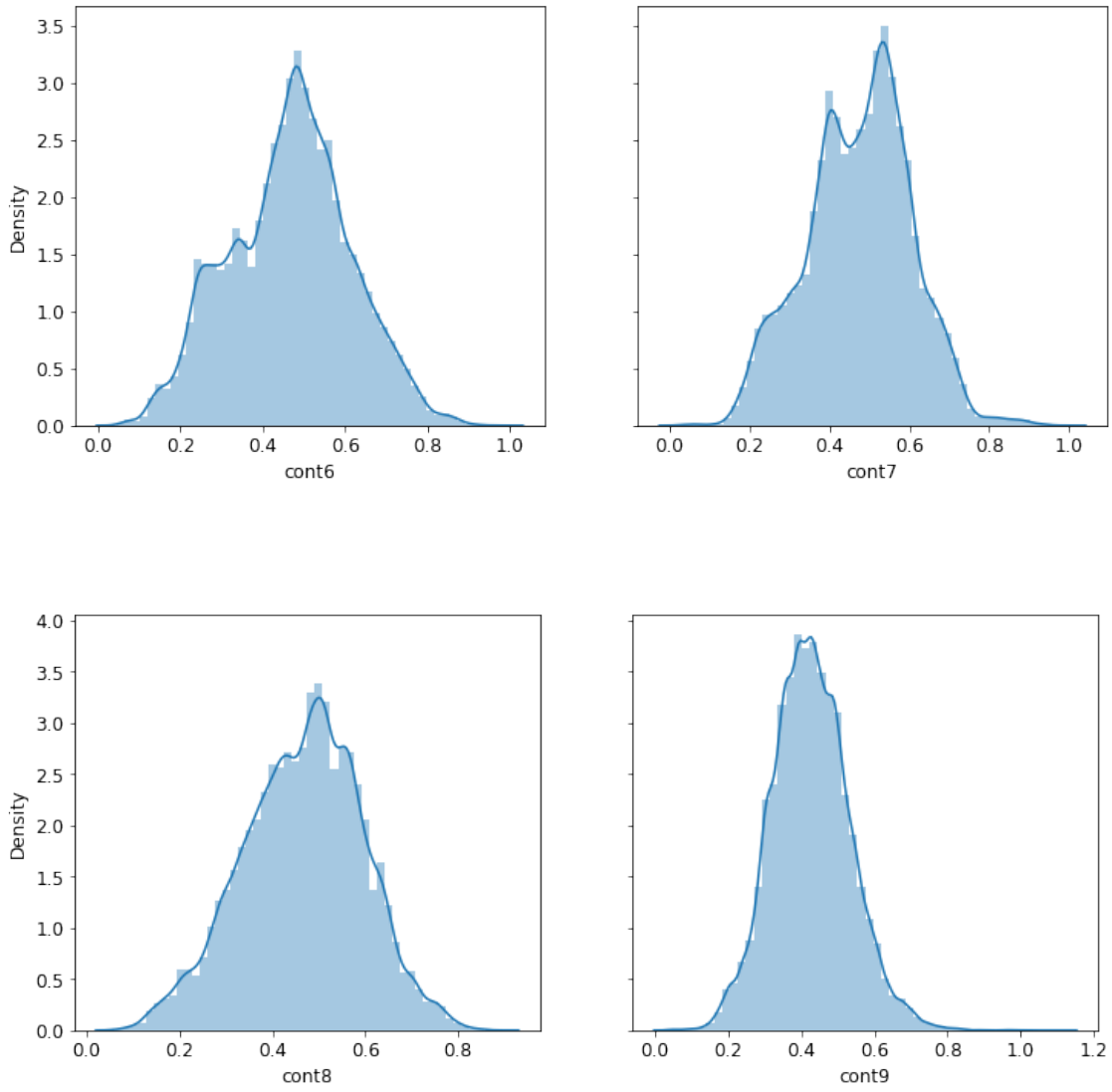
Neben einer Kernel-Schätzung (z.B. mittels `seaborn`) könnte man auch einfach das Histogramm (oder Kombinationen z.B. aus Violinplots oder Boxplots) der Daten plotten und eine Low-Level-Regression der Bins durchführen. Das ist aber nur zielführend, wenn die Verteilung nicht besonders komplex ist.

[14]:
```
# Liste der numerischen Merkmale erstellen und Hilfsdatei erstellen
num_list = [c for c in df.columns if(df[c].dtype != np.object)]
df_num = df[num_list]
```

[15]:
```
# Verteilungsdichte berechnen und graphisch anzeigen
ncol = 2
nrow = 5
numcols = num_list[:-1]

# Quelle: https://www.kaggle.com/nminus1/allstate-eda-python
for i in range(nrow):
    fig,axs = plt.subplots(nrows=1,ncols=ncol,sharey=True,figsize=(12, 5))
    cols = numcols[i*ncol:ncol*(i+1)]
    for i in range(len(axs)):
        sns.distplot(df_num[cols[i]], ax=axs[i], hist=True)
        xlabel=cols[i]
        axs[i].set(xlabel=xlabel, ylabel='Density')
```

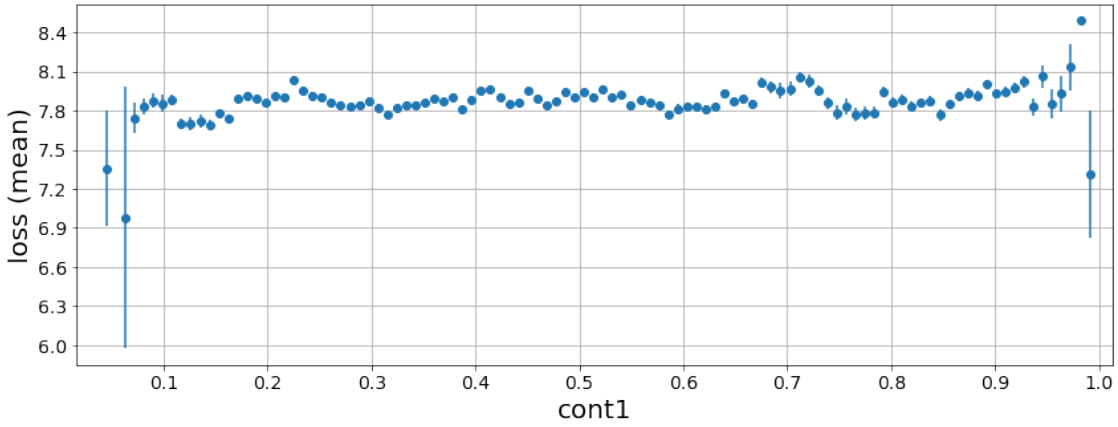


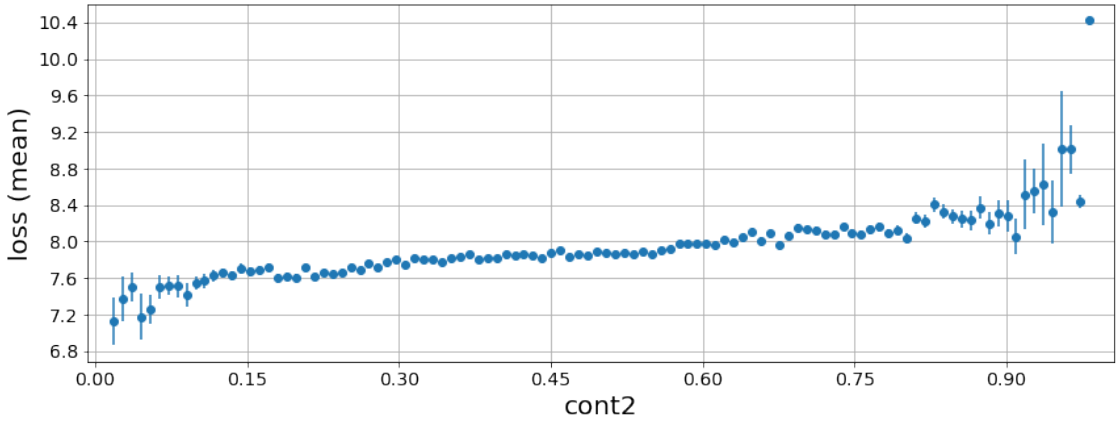

Density
cont2
cont3
Density
cont4
cont5

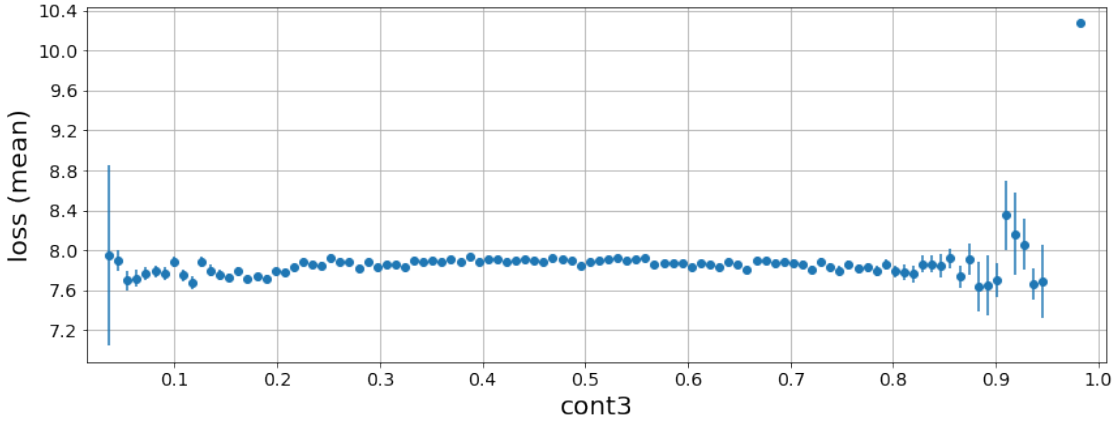


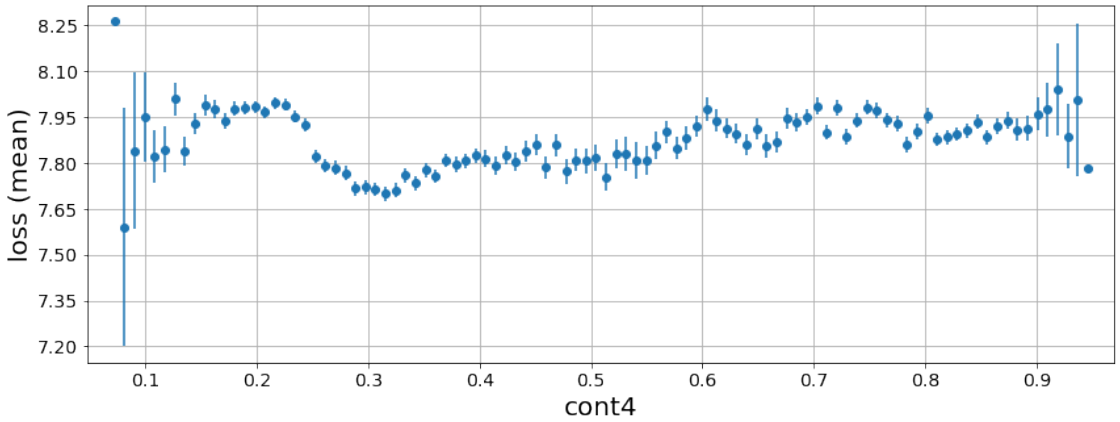
[16]:
```
# Numerische Mermale in Wertbereiche (Bins) einteilen und Schadendurchschnitt␣
 ↪für
# diese Wertebereiche berechnen und anzeigen
# Quelle: https://www.kaggle.com/jturkewitz/allstate-eda-cont-features␣
 ↪(modifiziert)
def plot_feature_loss(input_df,feature_name = 'cont1',num_bins = 111):
    feature_name_binned = feature_name + '_binned'
    bins = np.linspace(0,1.0,num_bins)
    df_num[feature_name_binned] = np.
 ↪digitize(df_num[feature_name],bins=bins,right=True)
    df_num[feature_name_binned] = df_num[feature_name_binned] / num_bins
    cont_14_dict = df_num.groupby(feature_name_binned)['loss'].mean().to_dict()
```

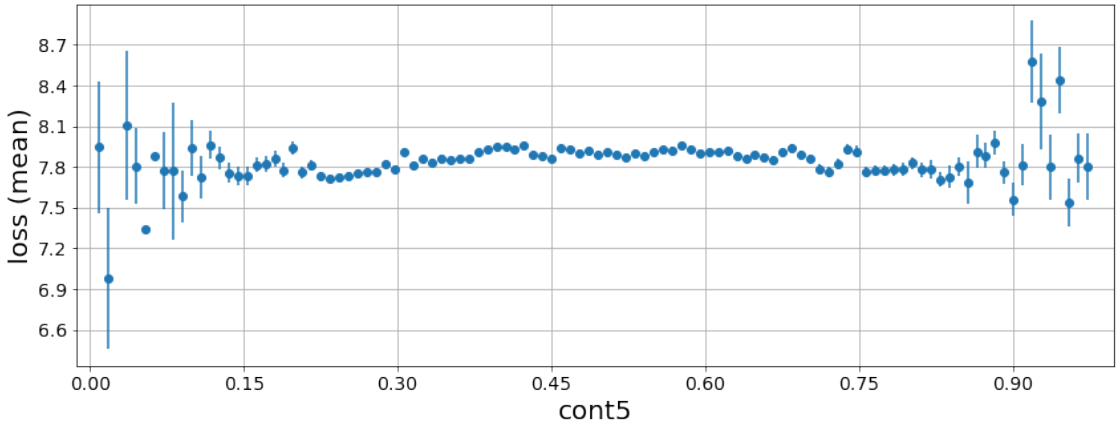
```
    cont_14_err_dict = df_num.groupby(feature_name_binned)['loss'].sem().
↪to_dict()
    lists = sorted(cont_14_dict.items())
    x, y = zip(*lists)
    lists_err = sorted(cont_14_err_dict.items())
    x_err, y_error = zip(*lists_err)

    p.figure(figsize=(14, 5))
    plt.errorbar(x,y,fmt = 'o',yerr = y_error,label = feature_name)
    p.xlabel(feature_name,fontsize=20)
    p.ylabel('loss (mean)',fontsize=20)
    plt.tick_params(axis='both', which='major', labelsize=14)
    p.xlim([df_num[feature_name].min() - 0.02, df_num[feature_name].max() + 0.
↪02 ])
    plt.grid()
    ax = plt.gca()

    plt.tick_params(axis='both', which='major', labelsize=14)
    ax.yaxis.set_major_locator(MaxNLocator(prune='lower'))
    ax.xaxis.set_major_locator(MaxNLocator(prune='lower'))
    ax.ticklabel_format(axis='y', style='sci', scilimits=(-2,2))

for name in df_num.columns:
    if name.startswith('cont'):
        plot_feature_loss(df_num,feature_name = name)

del df_num
```

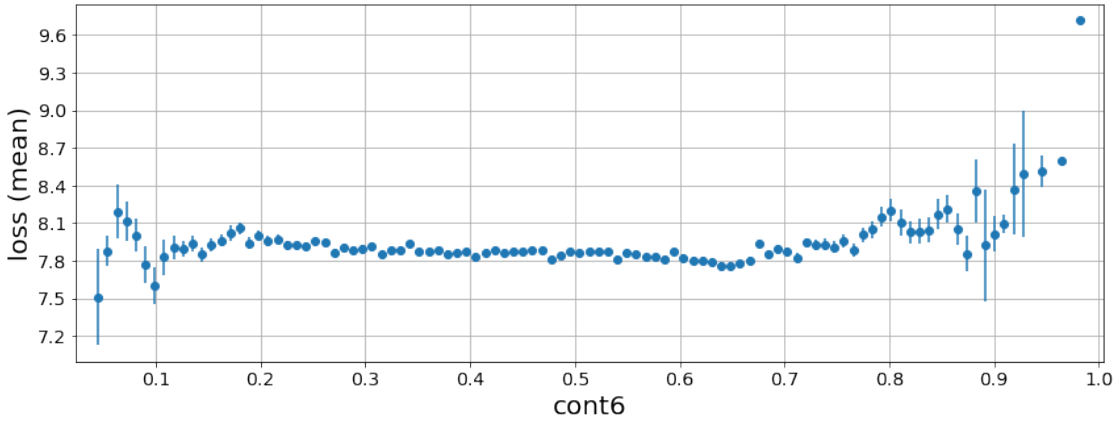


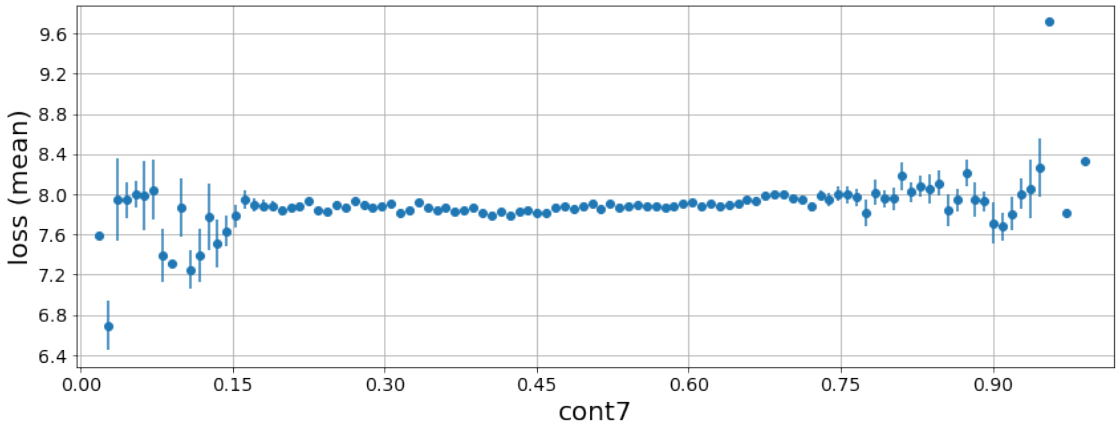

loss (mean)
10.4
10.0
9.6
9.2
8.8
8.4
8.0
7.6
7.2
6.8
0.00
0.15
0.30
0.45
0.60
0.75
0.90
cont2

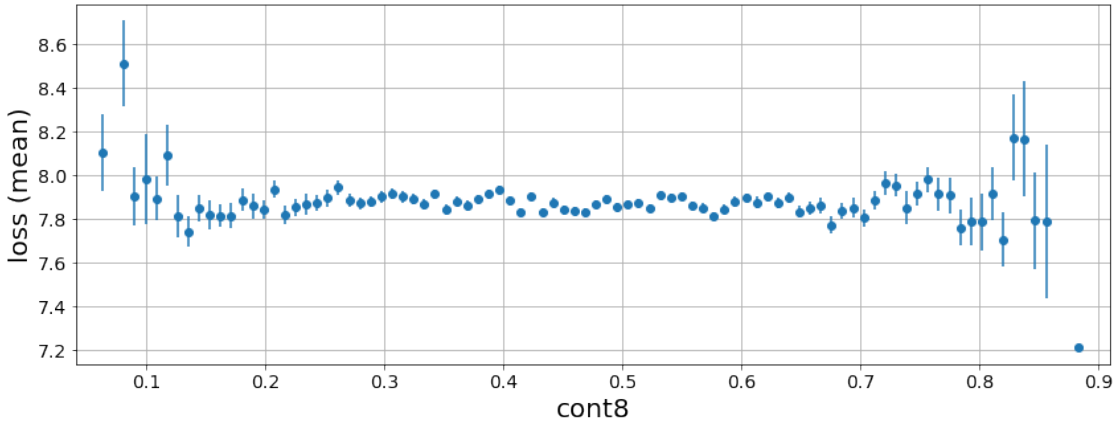

loss (mean)
10.4
10.0
9.6
9.2
8.8
8.4
8.0
7.6
7.2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
cont3

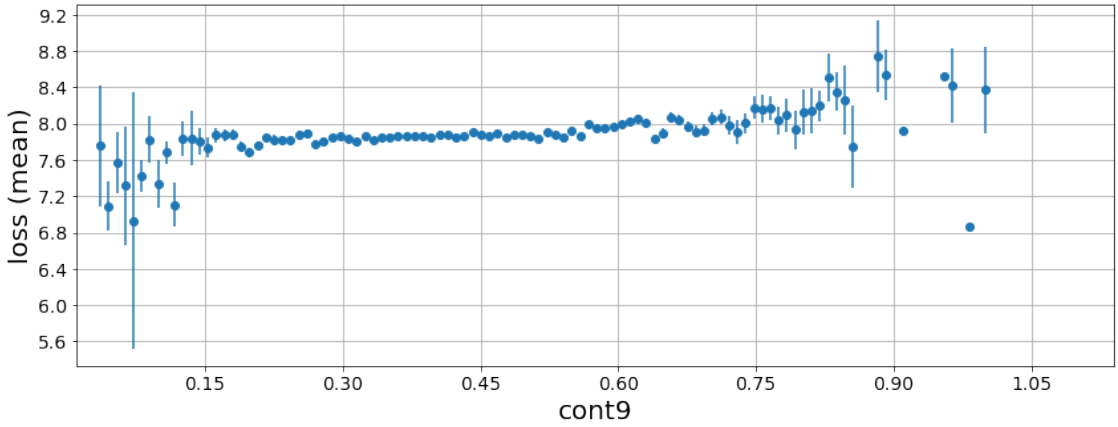

loss (mean)
8.25
8.10
7.95
7.80
7.65
7.50
7.35
7.20
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
cont4

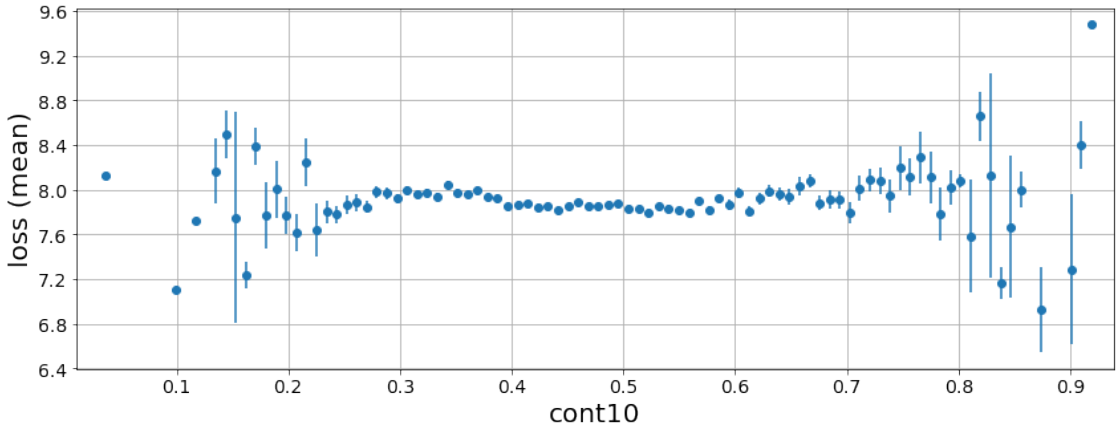

loss (mean)
8.7
8.4
8.1
7.8
7.5
7.2
6.9
6.6
0.00
0.15
0.30
0.45
0.60
0.75
0.90
cont5


loss (mean)
9.6
9.3
9.0
8.7
8.4
8.1
7.8
7.5
7.2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
cont6


loss (mean)
9.6
9.2
8.8
8.4
8.0
7.6
7.2
6.8
6.4
0.00
0.15
0.30
0.45
0.60
0.75
0.90
cont7


loss (mean)
8.6
8.4
8.2
8.0
7.8
7.6
7.4
7.2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
cont8


loss (mean)
9.2
8.8
8.4
8.0
7.6
7.2
6.8
6.4
6.0
5.6
0.15
0.30
0.45
0.60
0.75
0.90
1.05
cont9


loss (mean)
9.6
9.2
8.8
8.4
8.0
7.6
7.2
6.8
6.4
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
cont10

4.1.4 A-10: Im Weiteren soll die Korrelation zwischen den numerischen Merkmalen einschließlich 'loss' bestimmt werden. Dies ist auch in einer geeigneten Graphik darzustellen. Wieso ist die grafische Darstellung sinnvoll? Was sind die auffälligsten Zusammenhänge? (Lernziel 5.2.3) - [5 Punkte]

[11]:
```
# Quelle: https://www.kaggle.com/achalshah/allstate-feature-analysis-python

corrmat = df.corr()

# Parametrisiere matplotlib figure
f, ax = plt.subplots(figsize=(12, 9))

# Erzeuge heatmap mit seaborn
sns.heatmap(corrmat, annot=True, square=True, cmap='rainbow')
plt.show()
```

Die Merkmale "cont1" bis "cont10" sind auffällig unkorreliert. Dies könnte darauf zurück zu führen sein, dass der Datensatz Ergebnis einer vorangegangenen Dimensionsreduktion mittels PCA oder

eines Autoencoders ist. Lediglich mit dem Merkmal “loss” zeigen die Merkmale “cont1” bis “cont10” eine sehr schwache, meist positive Korrelation. Die Merkmale “cont6”, “cont8” und “cont10” korrelieren negativ mit dem Merkmal “loss”. Die Korrelation ist am stärksten ausgeprägt für “cont2”, am schwächsten für “cont8”.

4.2 2.2. Explorative Datenanalyse der kategoriellen Merkmale

4.2.1 A-11: Die Häufigkeitsverteilungen der kategoriellen Merkmale sind als Säulengraphiken darzustellen. Welche Schlussfolgerungen ergeben sich? (Lernziele 5.2.1, 6.1.2) - [5 Punkte]

[12]:
```
# Überblick über die Anzahl der verschiedenen Ausprägungen und die häufigste␣
↪Ausprägung
df.describe(include=['object'])
```

[12]:

	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	cat10	\
count	70907	70907	70907	70907	70907	70907	70907	70907	70907	70907	
unique	2	2	2	2	2	2	2	2	2	2	
top	A	A	A	A	A	A	A	A	B	A	
freq	62888	52363	67570	69280	47048	49186	69495	67073	70338	61185	

	cat11	cat12	cat13	cat14	cat15	cat16	cat17	cat18	cat19	cat20	\
count	70907	70907	70907	70907	70907	70907	70907	70907	70907	70907	
unique	2	2	2	2	2	2	2	2	2	2	
top	A	A	A	A	A	A	A	A	A	A	
freq	70528	61030	64137	70143	67662	68929	68897	70390	70253	70770	

	cat21	cat22	cat23	cat24	cat25	cat26	cat27	cat28	cat29	cat30	\
count	70907	70907	70907	70907	70907	70907	70907	70907	70907	70907	
unique	2	2	2	2	2	2	2	2	2	2	
top	A	A	A	A	A	A	A	A	A	A	
freq	68751	45470	67221	67989	64226	66751	51037	68437	69685	69635	

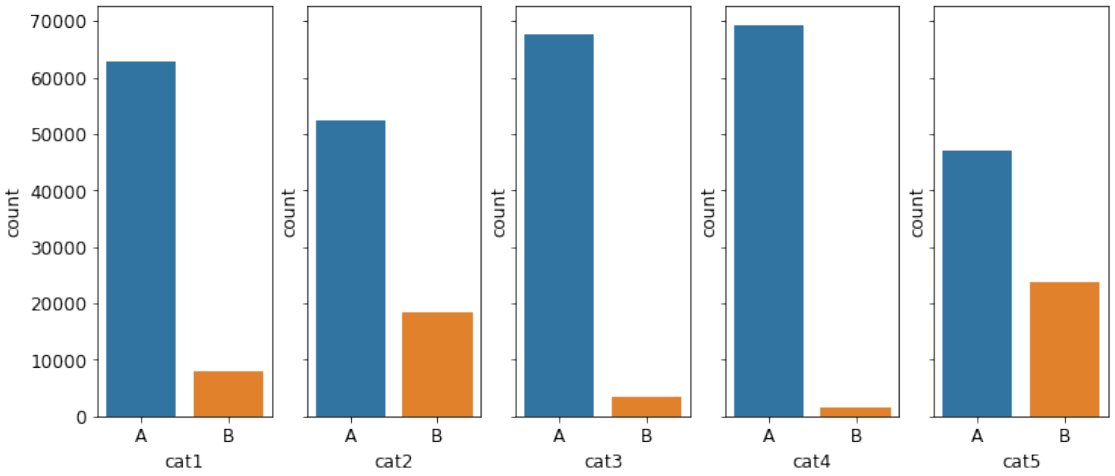
	cat31	cat32	cat33	cat34	cat35	cat36	cat37	cat38	cat39	cat40	\
count	70907	70907	70907	70907	70907	70907	70907	70907	70907	70907	
unique	2	2	2	2	2	2	2	2	2	2	
top	A	A	A	A	A	A	A	A	A	A	
freq	68932	70152	65071	70062	65263	59383	51427	69210	69090	68110	

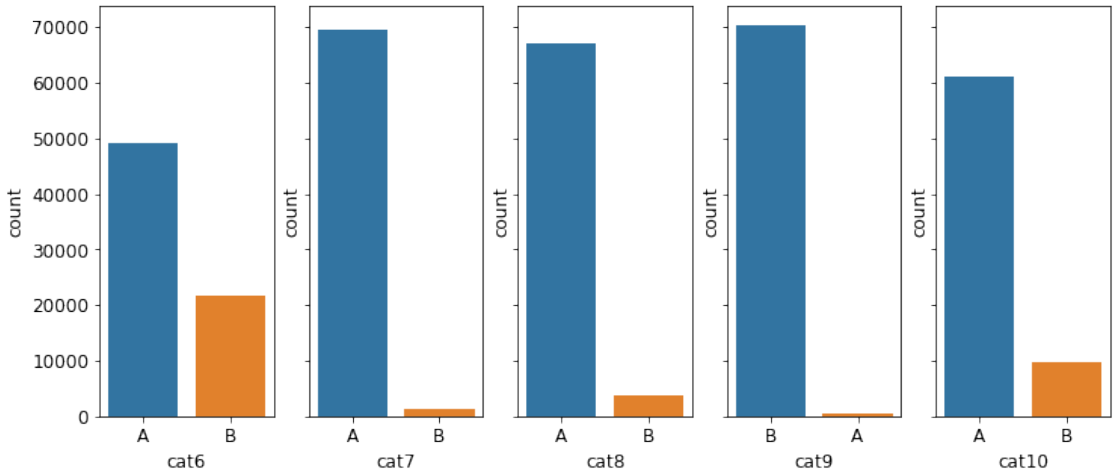
	cat41	cat42	cat43	cat44	cat45	cat46	cat47	cat48	cat49	cat50	\
count	70907	70907	70907	70907	70907	70907	70907	70907	70907	70907	
unique	2	2	2	3	3	3	3	3	3	3	
top	A	A	A	A	A	C	A	B	C	A	
freq	68471	69422	70770	67570	47433	43387	55075	70267	32368	69613	

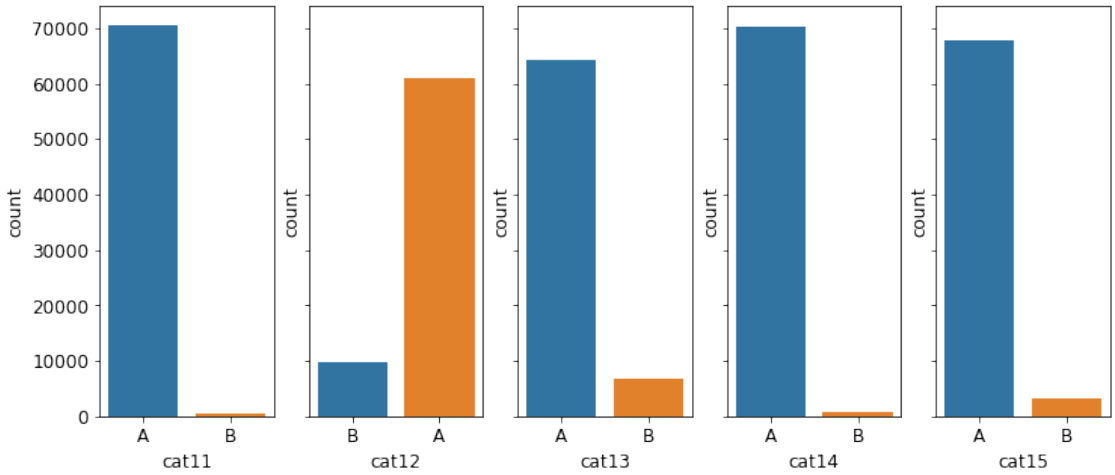
	cat51	cat52	cat53	cat54	cat55	cat56	cat57	cat58	cat59	cat60	\
count	70907	70907	70907	70907	70907	70907	70907	70907	70907	70907	
unique	3	3	3	3	3	4	4	4	4	4	

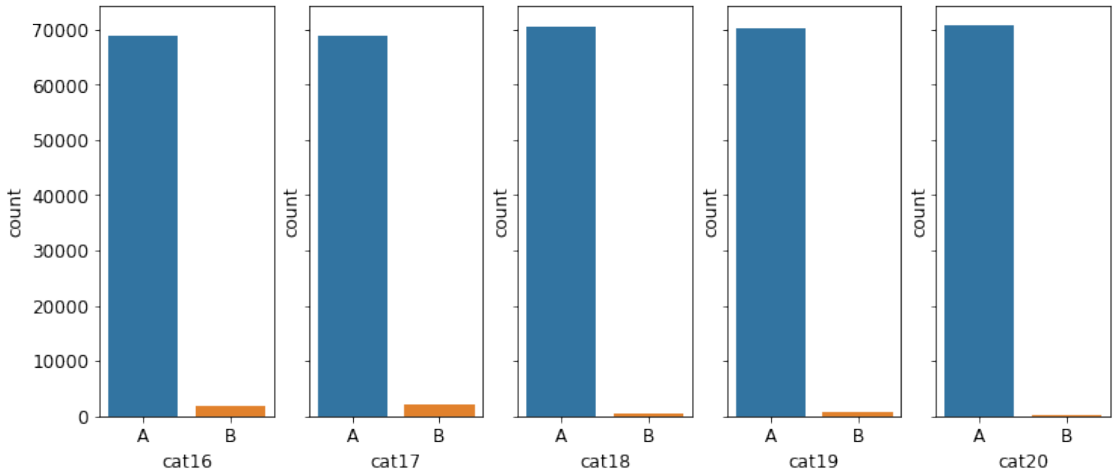
```
top          B      C      B      A      A      A      B      D      D      B
freq     64151  59071  39634  59575  69497  42338  59438  45789  56626  40238

         cat61  cat62  cat63  cat64  cat65  cat66  cat67  cat68  cat69  cat70  \
count    70907  70907  70907  70907  70907  70907  70907  70907  70907  70907
unique       4      5      6      8     10     10     11     12     12     12
top          A      A      A      B      A      B      E      A      B      E
freq     35868  65833  67073  24715  47048  48854  16039  28583  30323  16081

         cat71  cat72  cat73  cat74  cat75
count    70907  70907  70907  70907  70907
unique      12     16     17     43     51
top          A      A      F     AG     AW
freq     41062  17867  17955  36452   9119
```

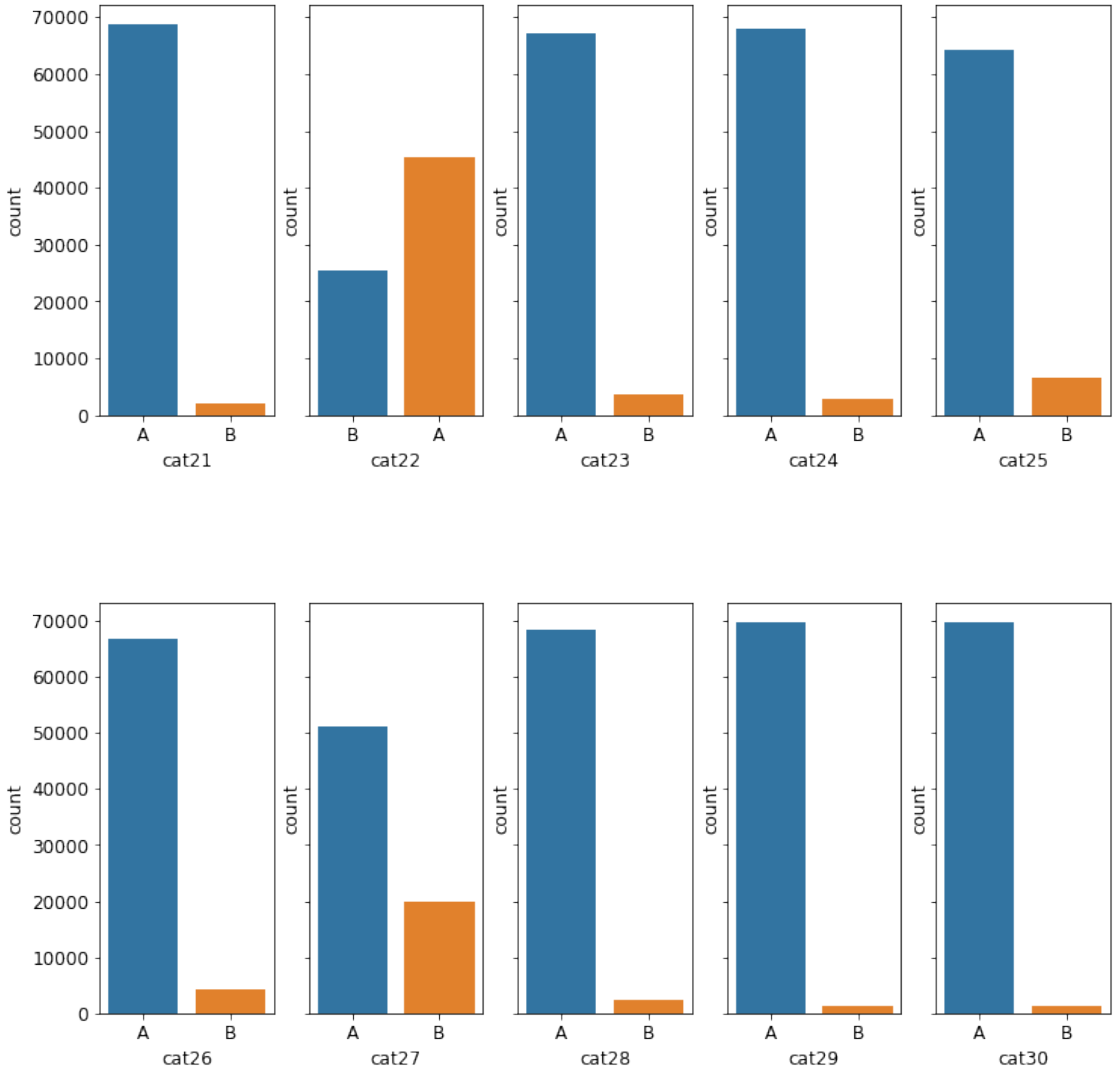
[13]:
```
# Namen aller Spalten
cols = df.columns

# Plot Anzahl für alle Merkmale in einem 12x5 Gitter
n_cols = 5
n_rows = 12
for i in range(n_rows):
    fg,ax = plt.subplots(nrows=1,ncols=n_cols,sharey=True,figsize=(12, 5))
    for j in range(n_cols):
        sns.countplot(x=cols[i*n_cols+j], data=df, ax=ax[j])
```

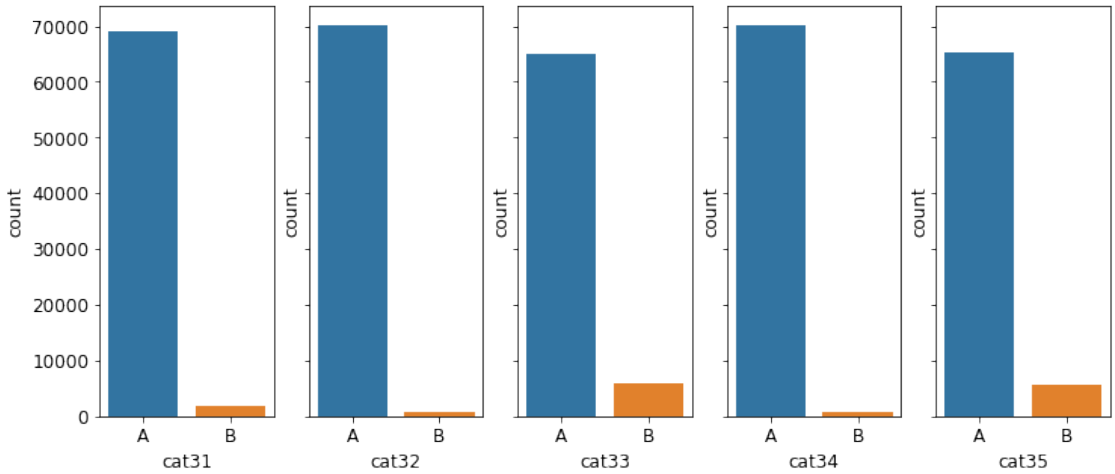


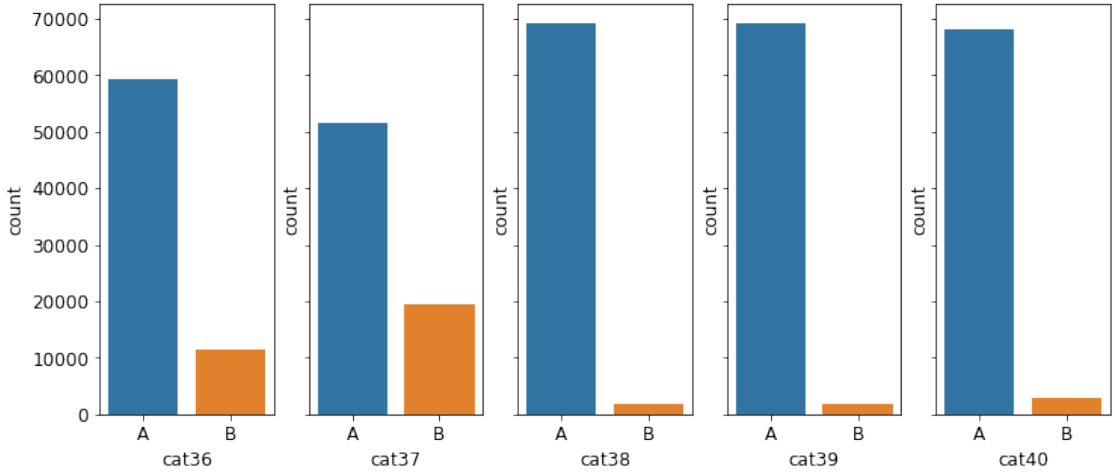

count
70000
60000
50000
40000
30000
20000
10000
0
A
B
cat6
cat7
cat8
B
A
cat9
cat10

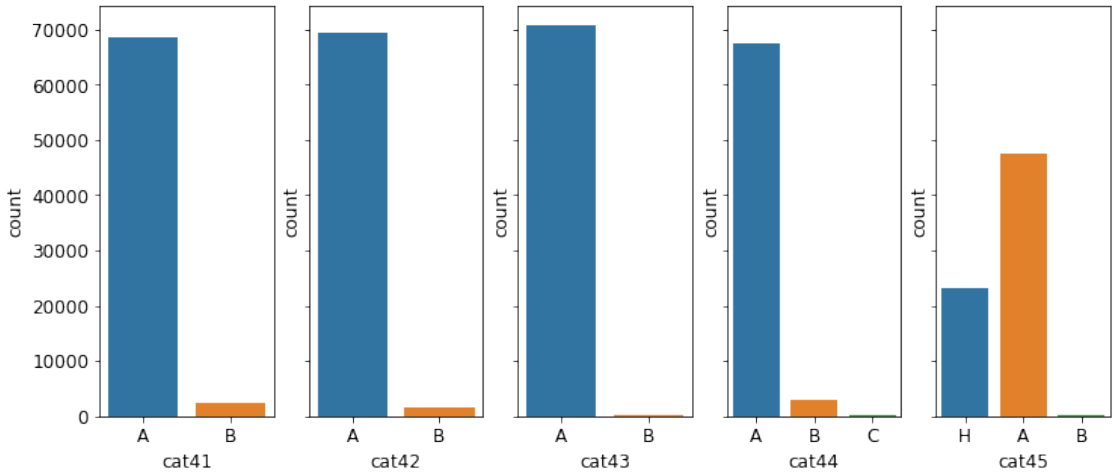

count
70000
60000
50000
40000
30000
20000
10000
0
A
B
cat11
B
A
cat12
cat13
cat14
cat15

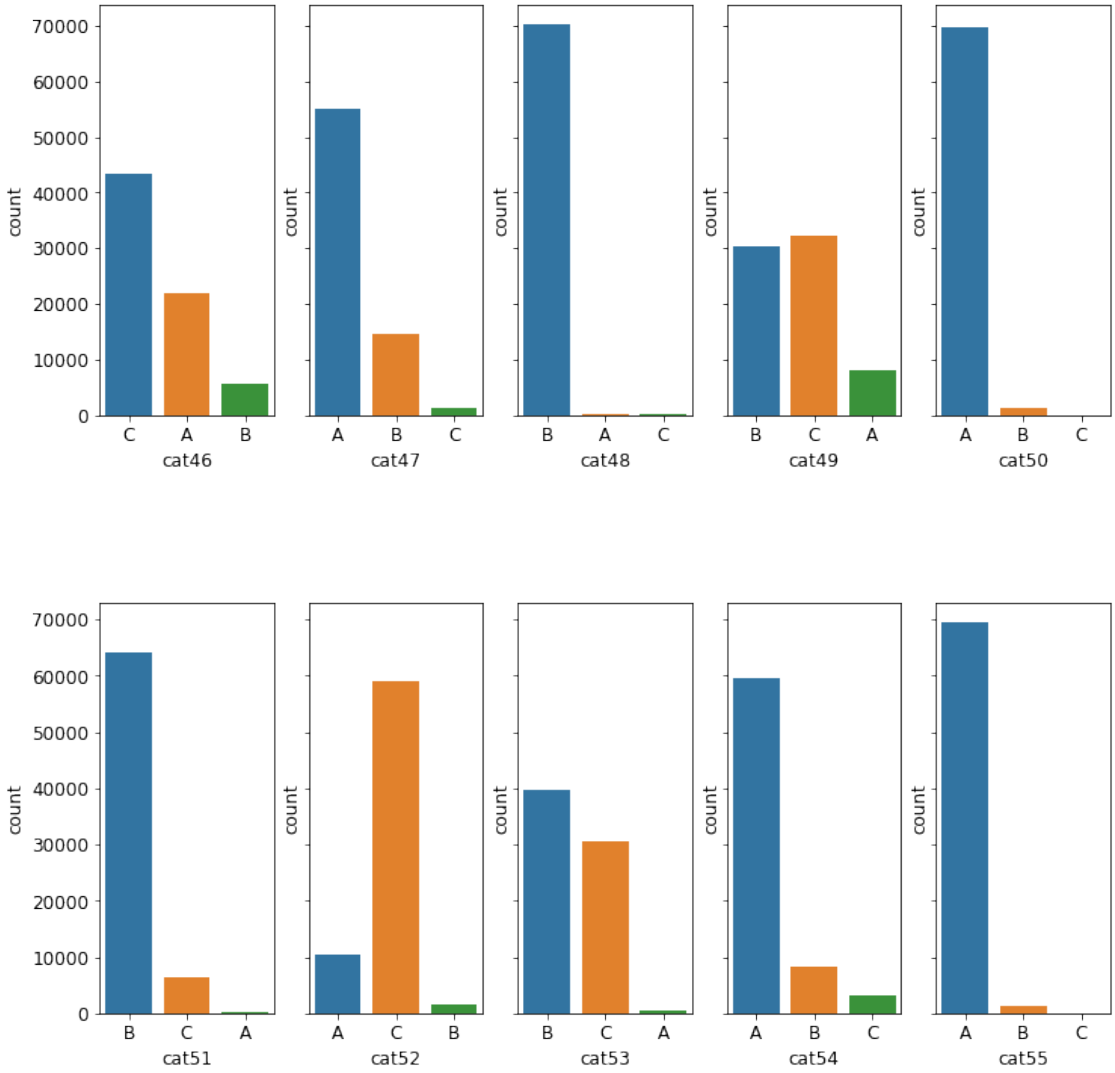

count
70000
60000
50000
40000
30000
20000
10000
0
A
B
cat16
cat17
cat18
cat19
cat20

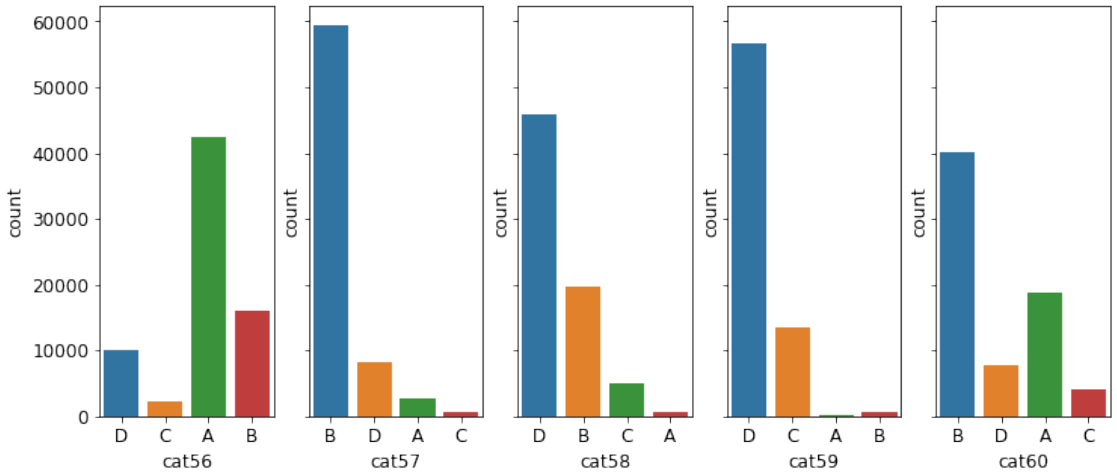

70000
60000
50000
40000
30000
20000
10000
0
count
A
B
cat21
B
A
cat22
A
B
cat23
A
B
cat24
A
B
cat25
A
B
cat26
A
B
cat27
A
B
cat28
A
B
cat29
A
B
cat30


count
70000
60000
50000
40000
30000
20000
10000
0
A
B
cat31
cat32
cat33
cat34
cat35

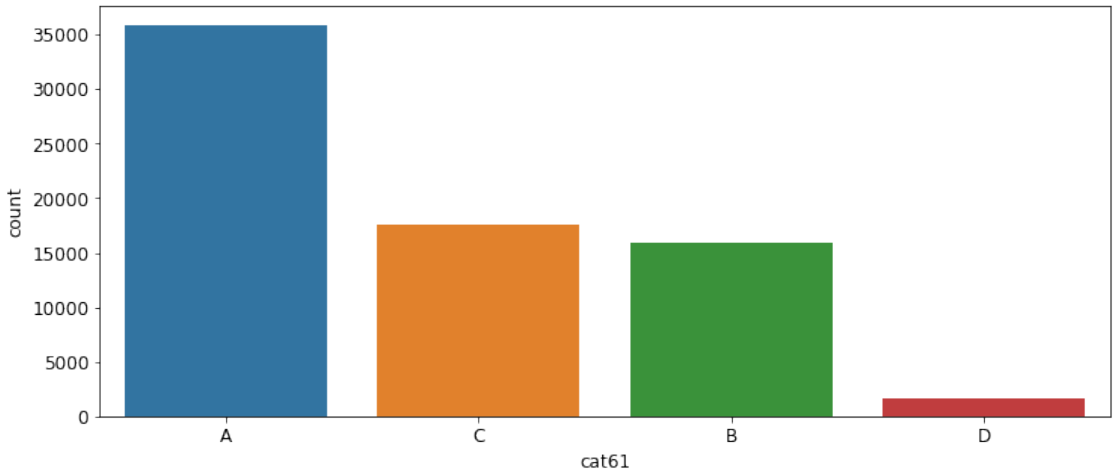

count
70000
60000
50000
40000
30000
20000
10000
0
A
B
cat36
cat37
cat38
cat39
cat40

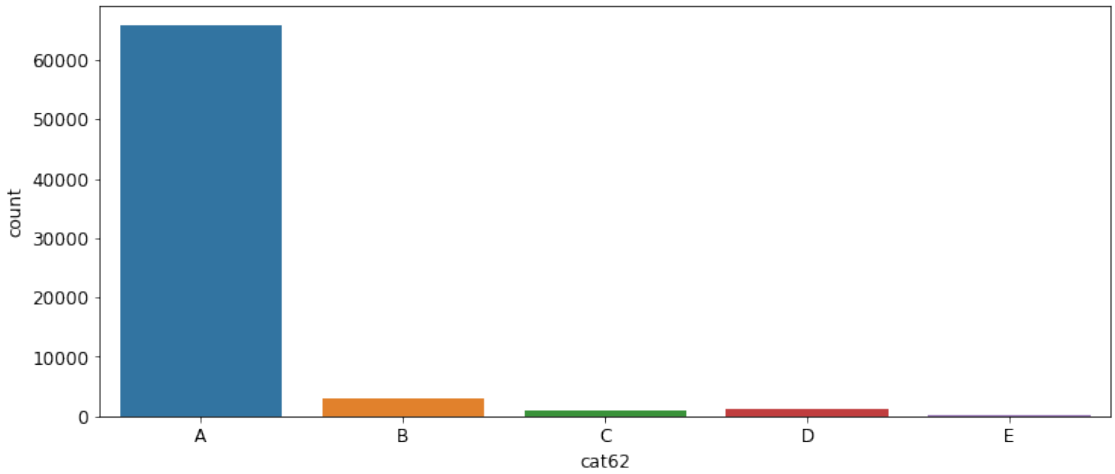

count
70000
60000
50000
40000
30000
20000
10000
0
A
B
C
H
cat41
cat42
cat43
cat44
cat45

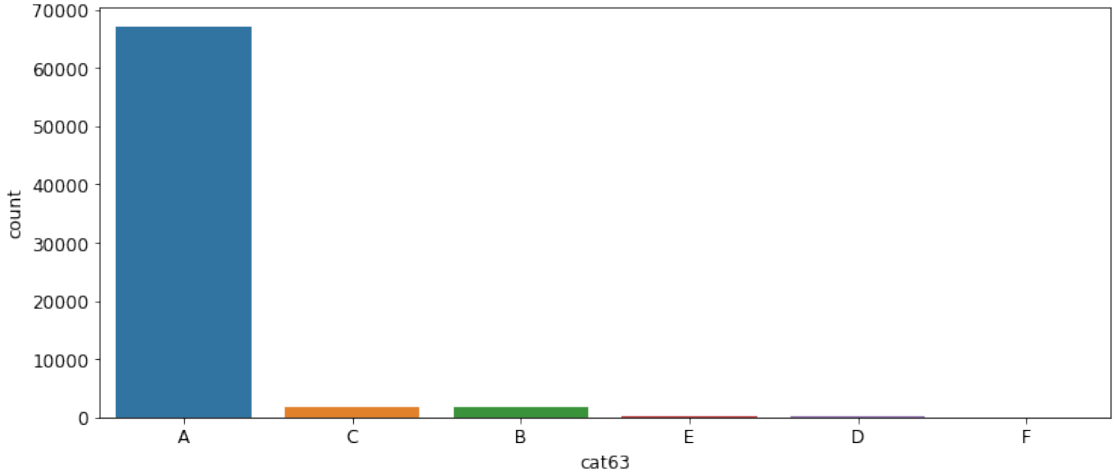

count
70000
60000
50000
40000
30000
20000
10000
0
C A B
cat46
A B C
cat47
B A C
cat48
B C A
cat49
A B C
cat50
B C A
cat51
A C B
cat52
B C A
cat53
A B C
cat54
A B C
cat55

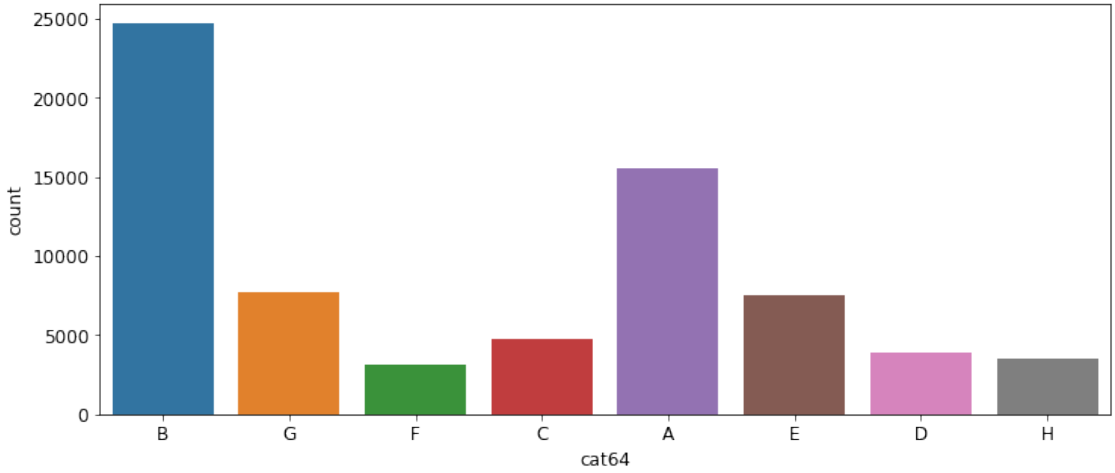


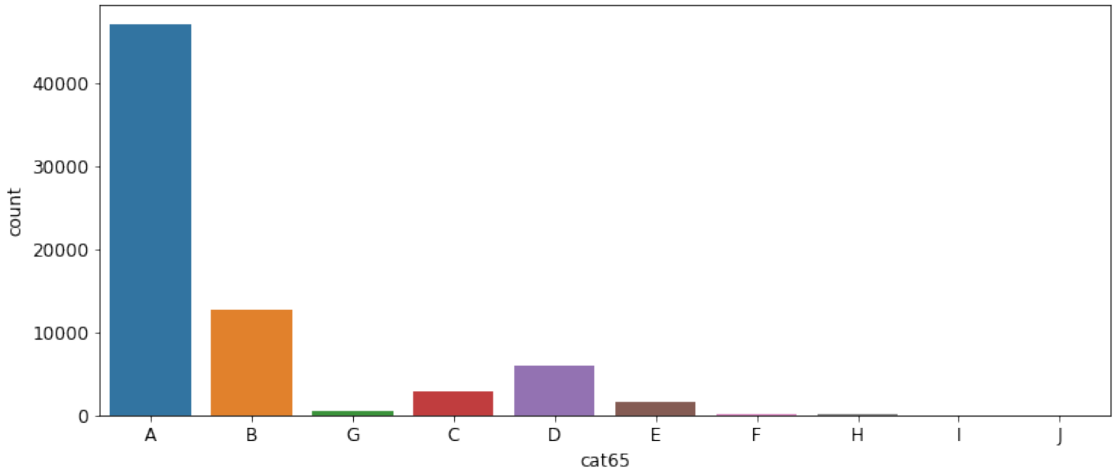
[14]:
```
# Die weiteren Plots einzeln anzeigen
n_rows = 15
for i in range(n_rows):
    fg,ax = plt.subplots(figsize=(12, 5))
    sns.countplot(x=cols[i+60], data=df)
```

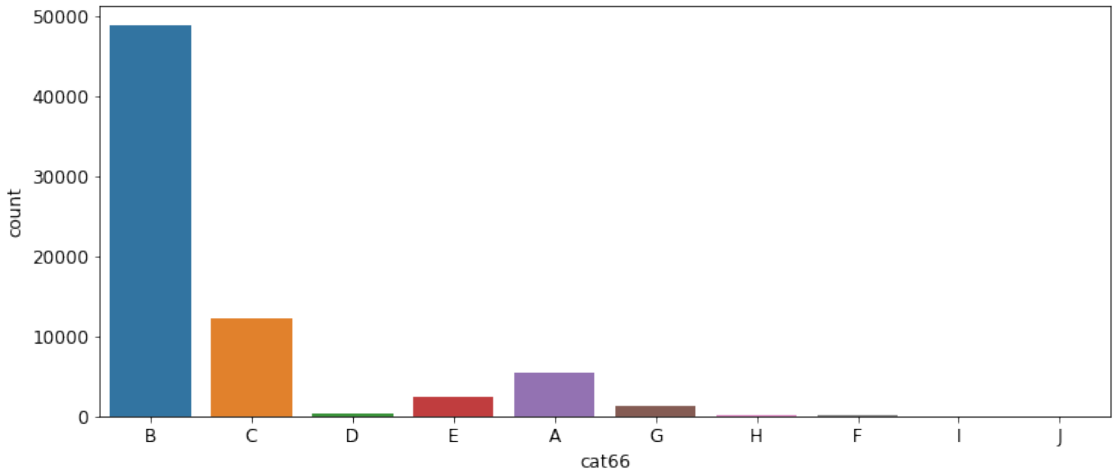


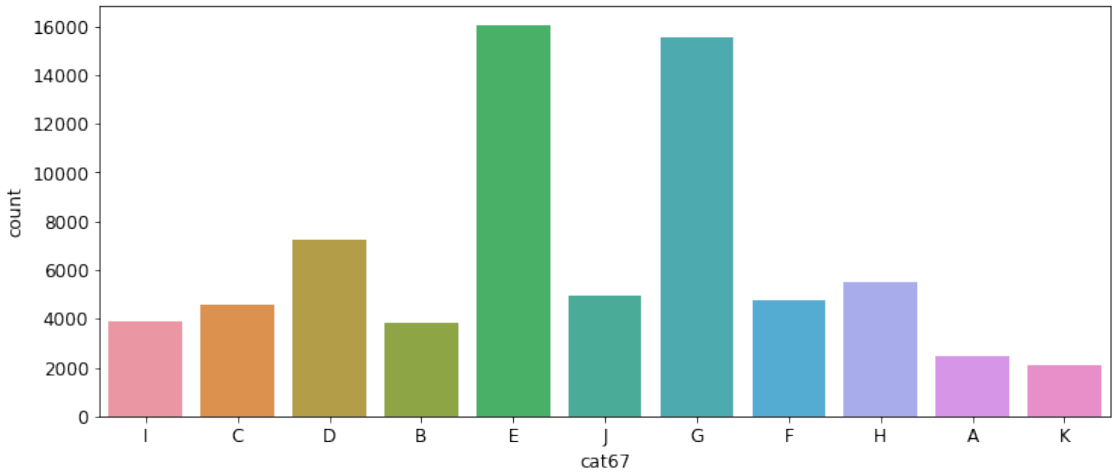

60000
50000
40000
30000
20000
10000
0
count
A
B
C
D
E
cat62

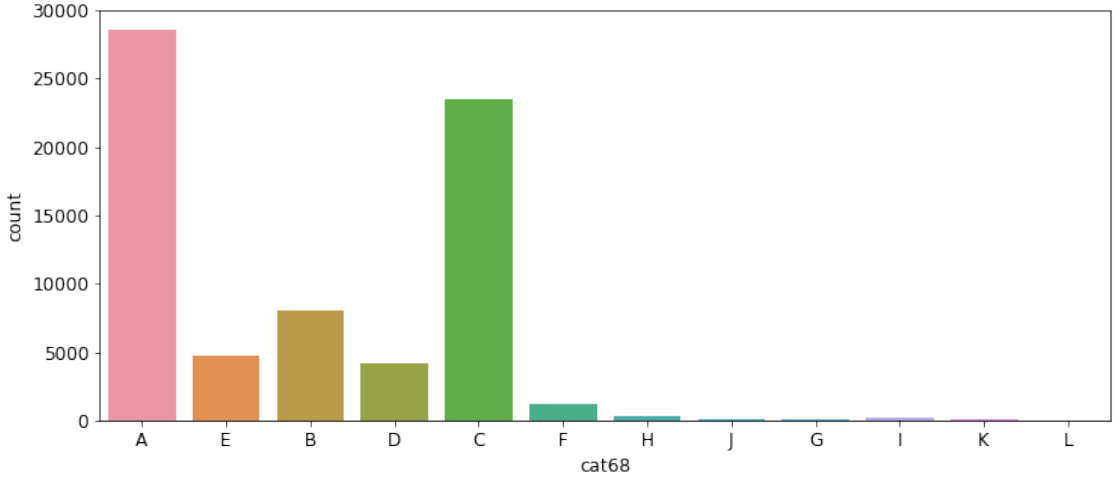

70000
60000
50000
40000
30000
20000
10000
0
count
A
C
B
E
D
F
cat63

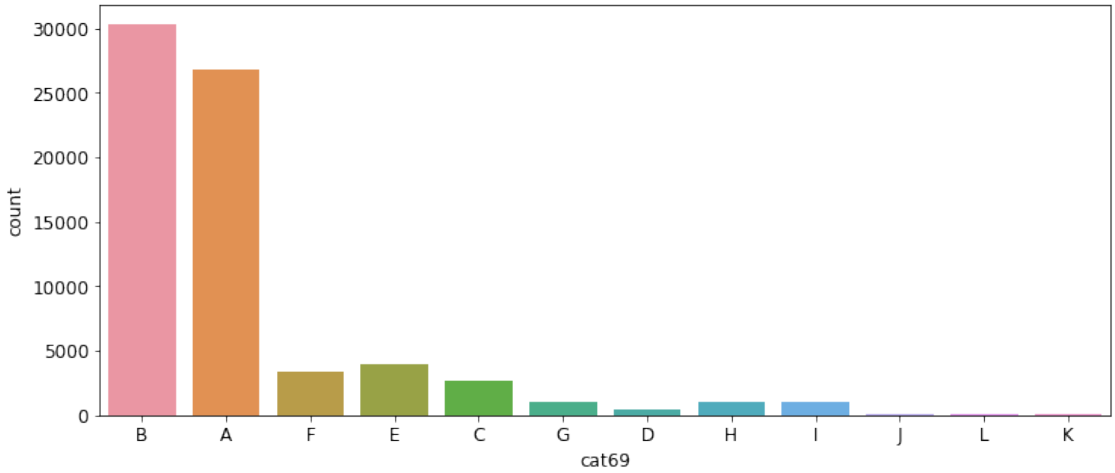

25000
20000
15000
10000
5000
0
count
B
G
F
C
A
E
D
H
cat64

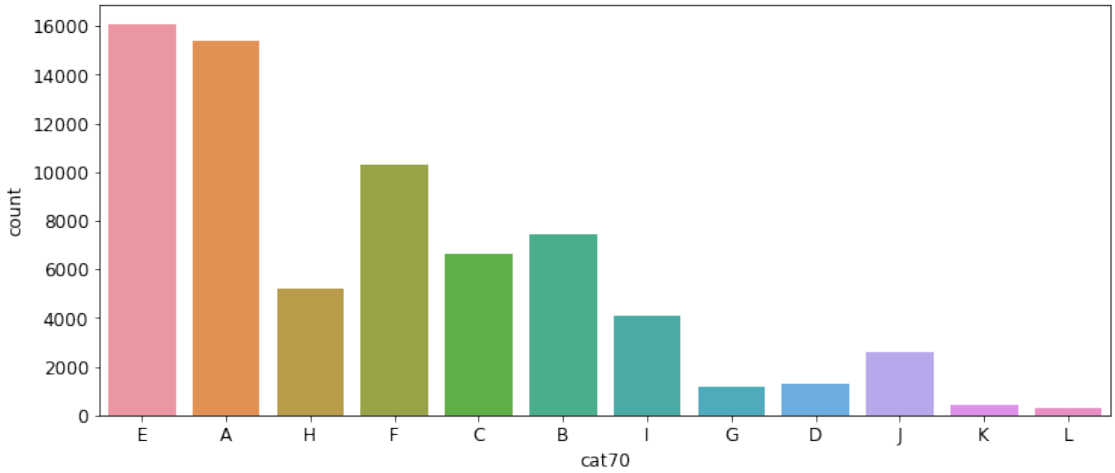

40000
30000
20000
10000
0
count
A
B
G
C
D
E
F
H
I
J
cat65

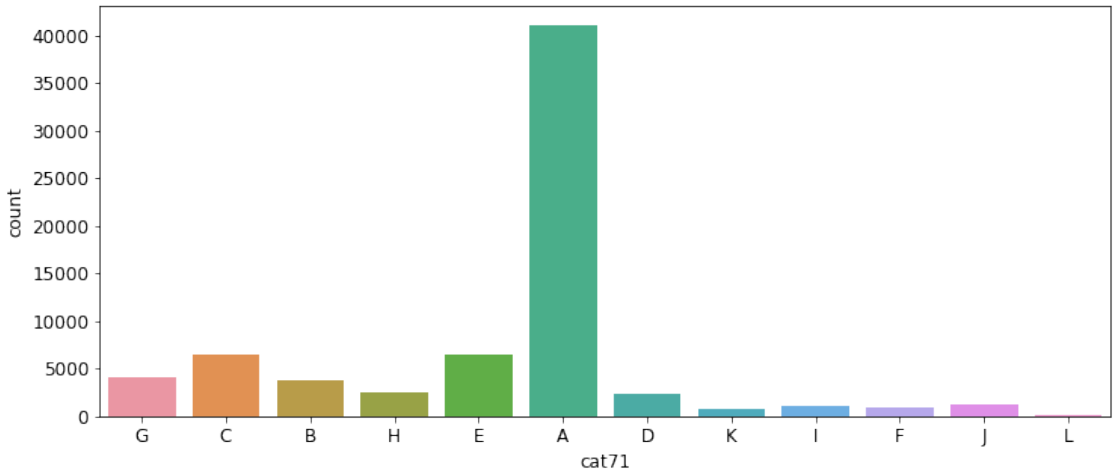

50000
40000
30000
20000
10000
0
count
B
C
D
E
A
G
H
F
I
J
cat66

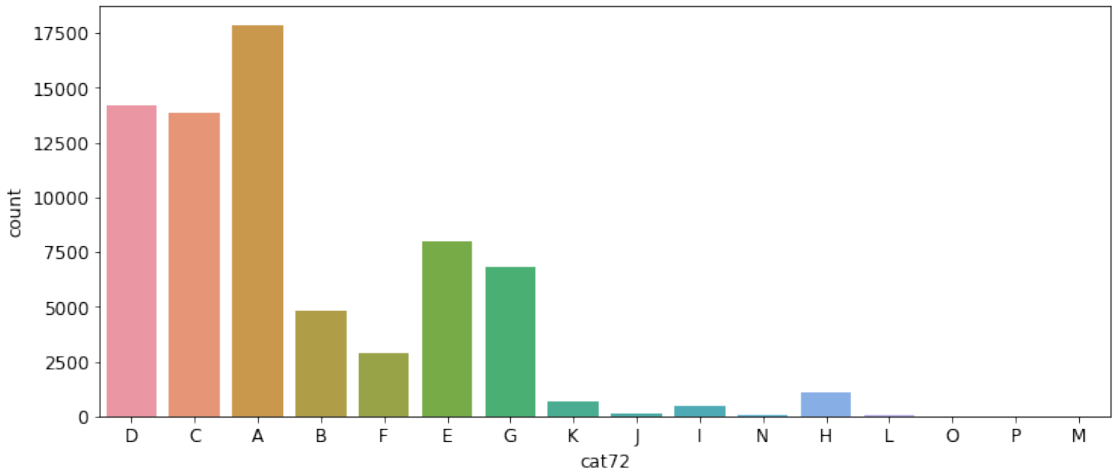

16000
14000
12000
10000
8000
6000
4000
2000
0
count
I
C
D
B
E
J
G
F
H
A
K
cat67

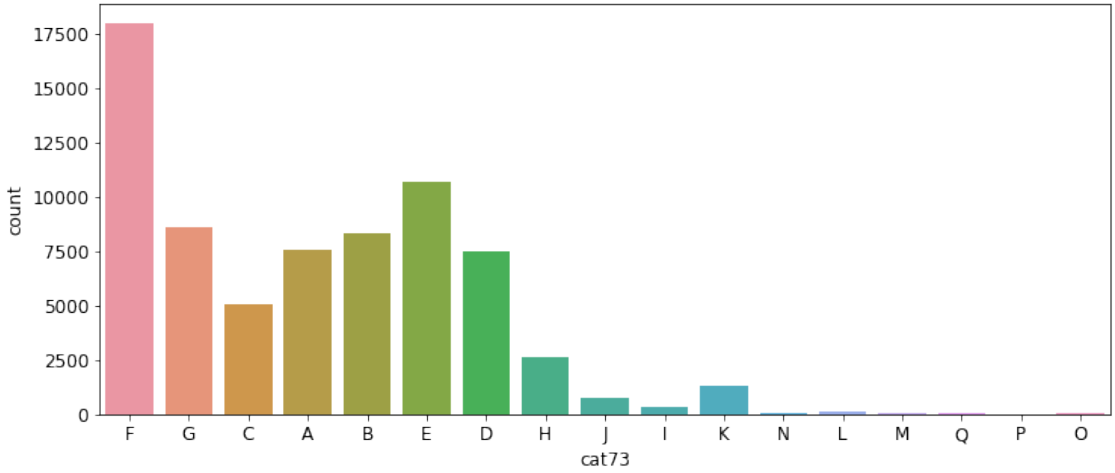

30000
25000
20000
15000
10000
5000
0
count
A
E
B
D
C
F
H
J
G
I
K
L
cat68

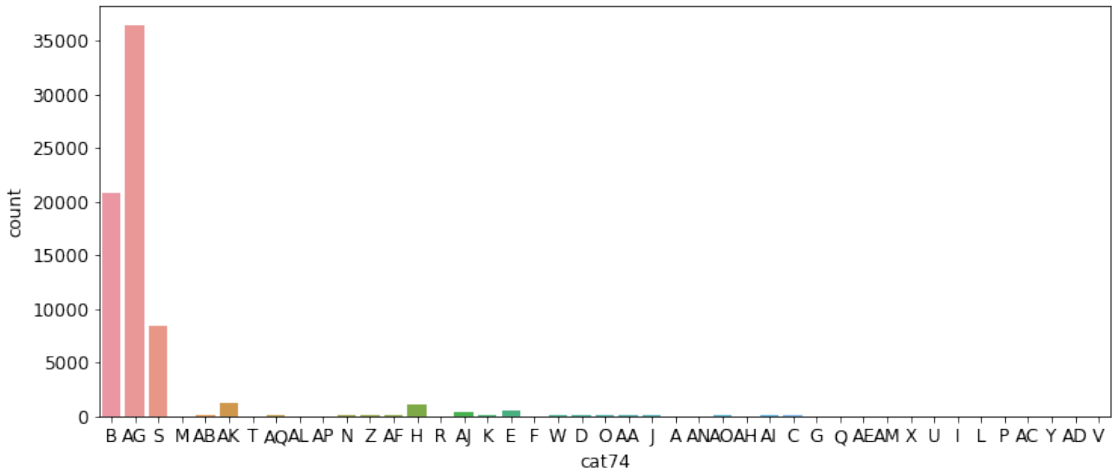

30000
25000
20000
15000
10000
5000
0
count
B
A
F
E
C
G
D
H
I
J
L
K
cat69

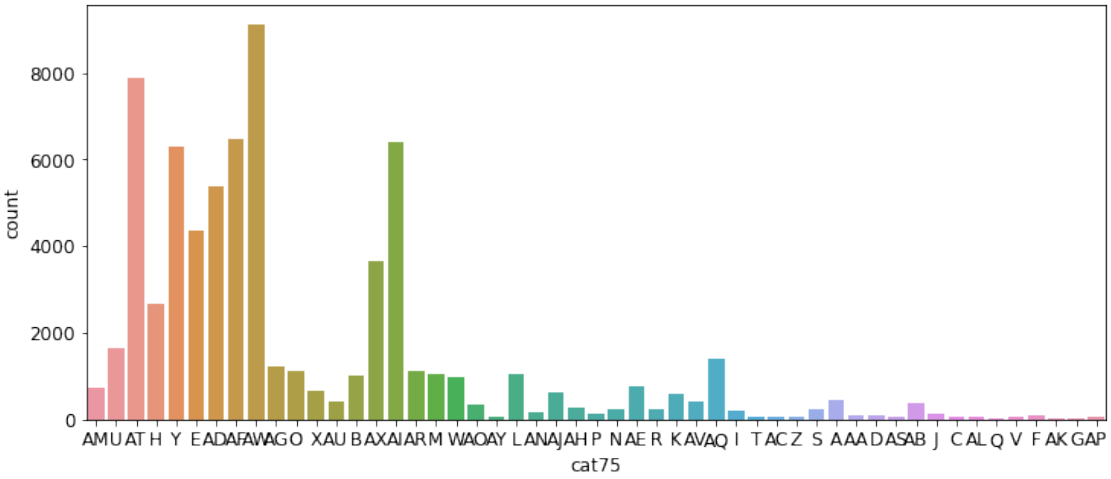

16000
14000
12000
10000
8000
6000
4000
2000
0
count
E
A
H
F
C
B
I
G
D
J
K
L
cat70


40000
35000
30000
25000
20000
15000
10000
5000
0
count
G
C
B
H
E
A
D
K
I
F
J
L
cat71


count
17500
15000
12500
10000
7500
5000
2500
0
D C A B F E G K J I N H L O P M
cat72


count
17500
15000
12500
10000
7500
5000
2500
0
F G C A B E D H J I K N L M Q P O
cat73

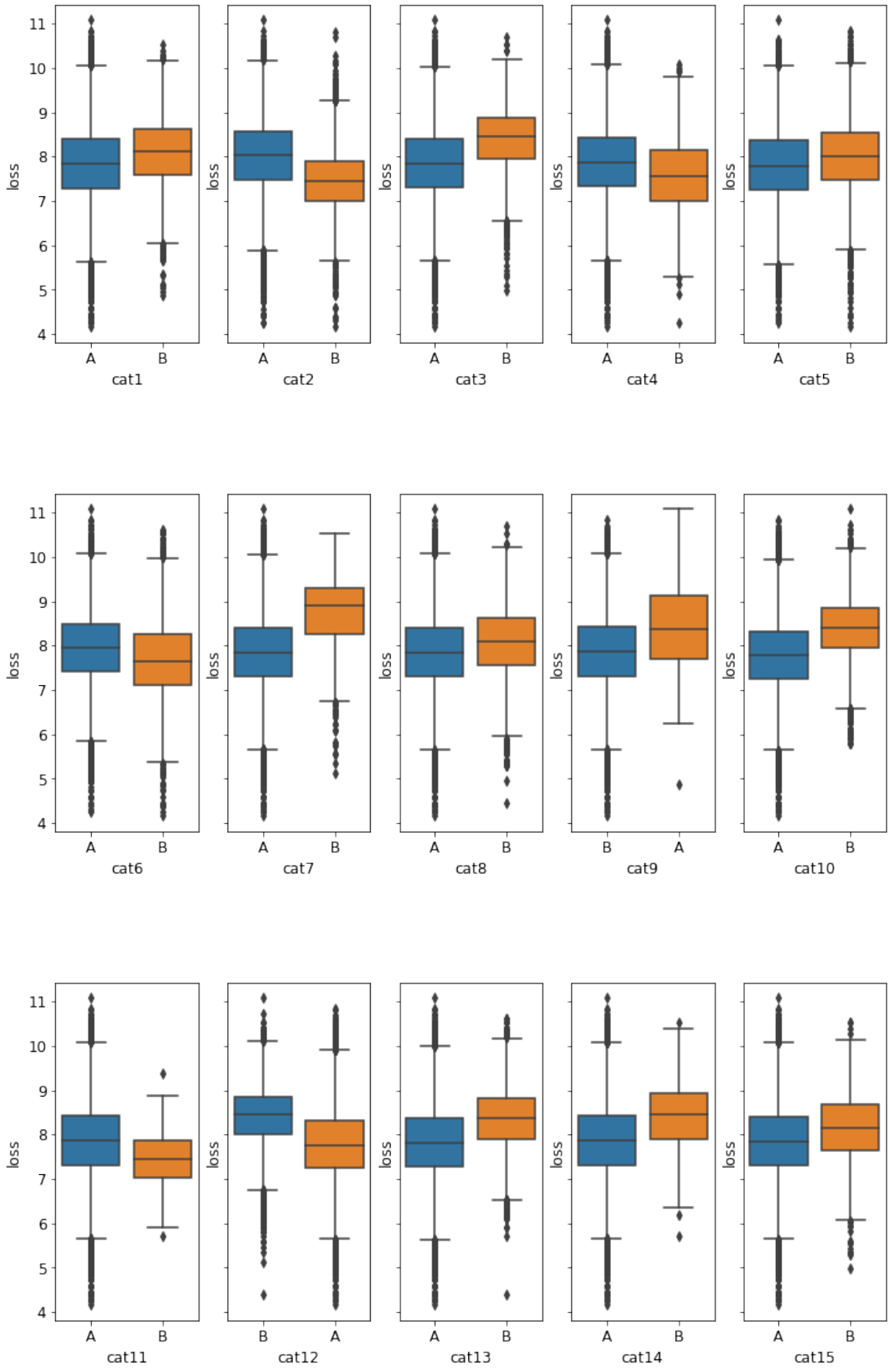

count
35000
30000
25000
20000
15000
10000
5000
0
B AG S M AB AK T AQ AL AP N Z AF H R AJ K E F W D O AA J A AN AO AH AI C G Q AE AM X U I L P AC Y AD V
cat74

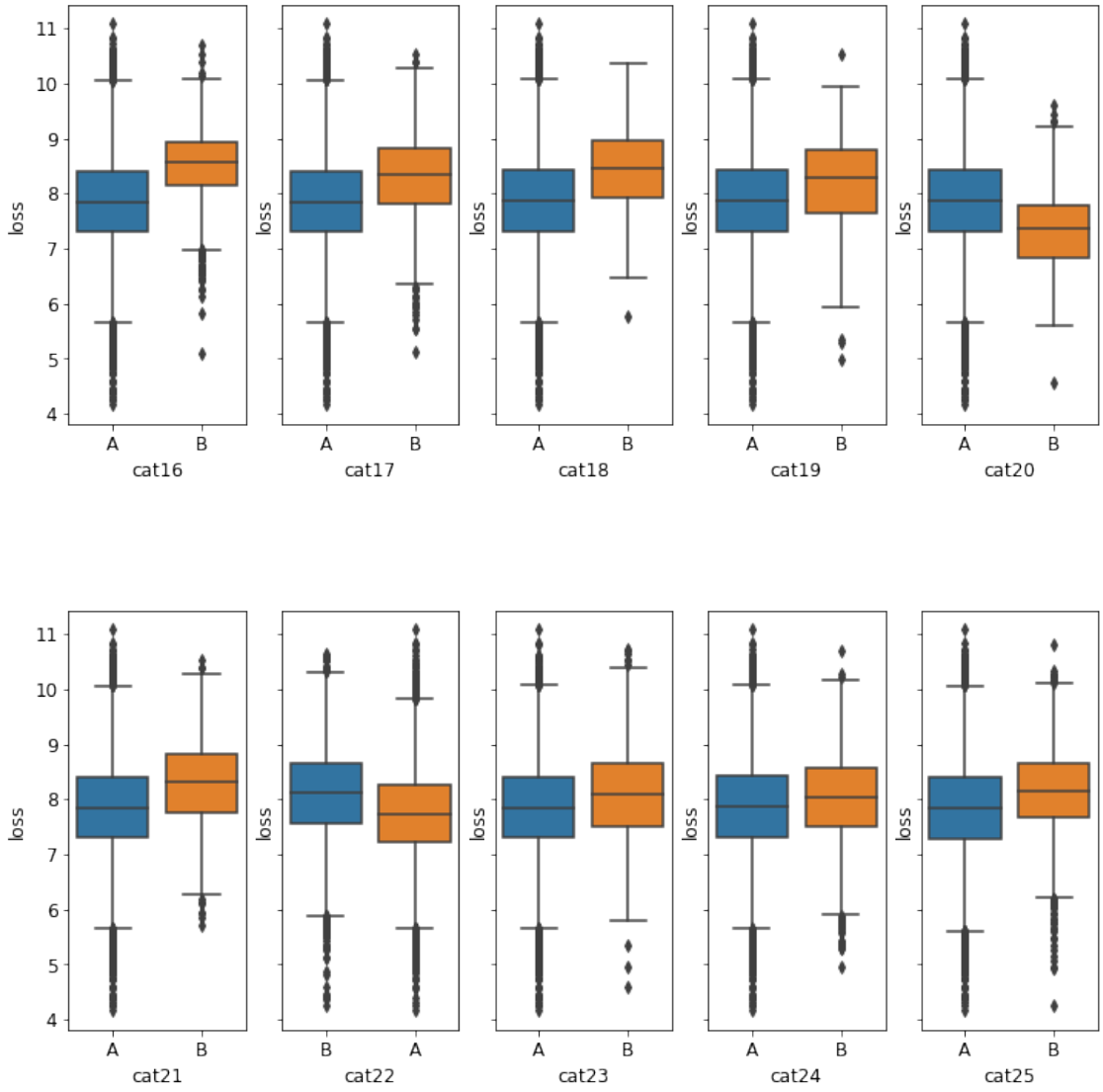


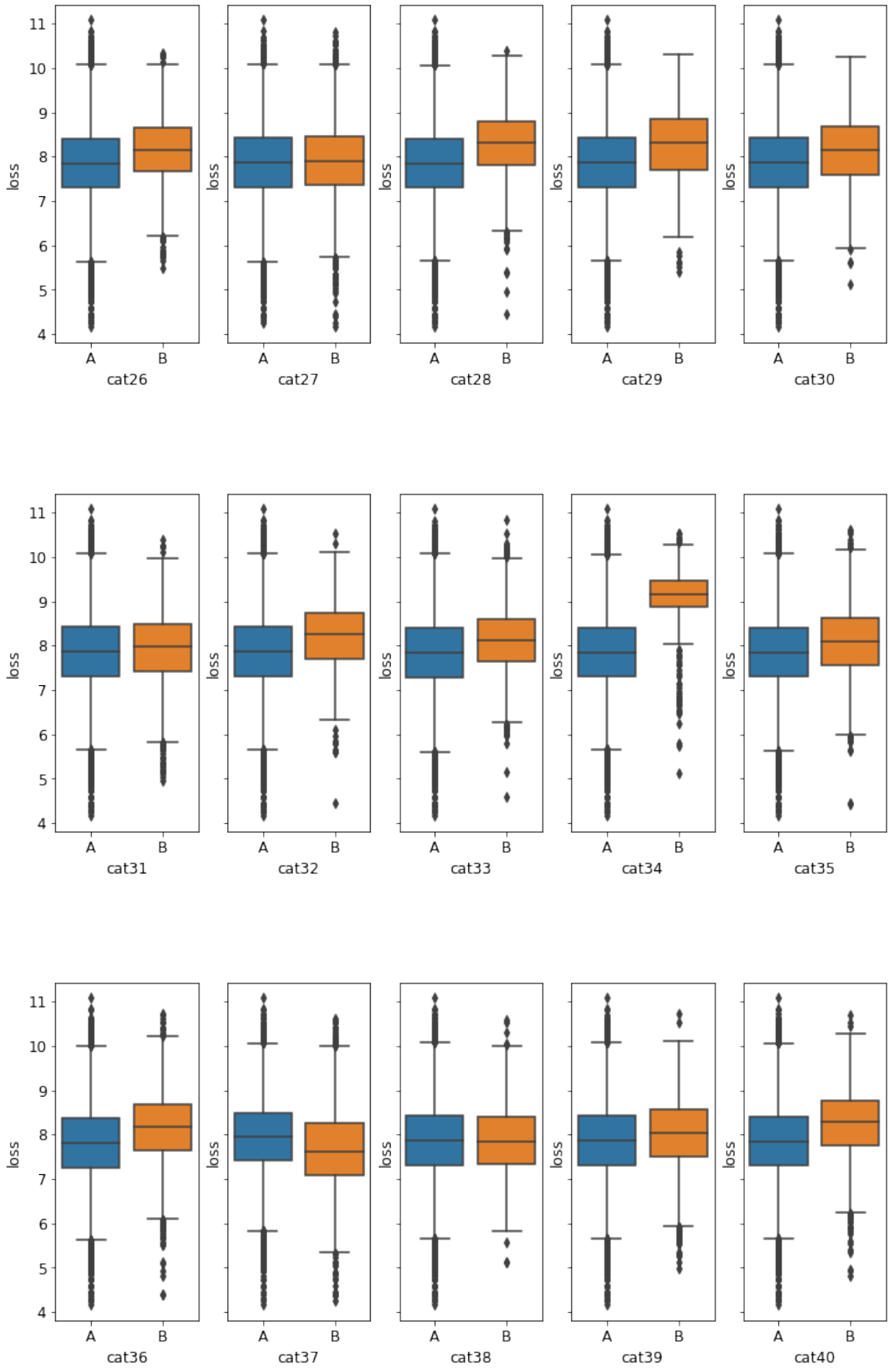
Der größte Teil der Merkmale (75 von 86) sind nominale Merkmale. Davon haben 43 Merkmale zwei Merkmalsausprägungen ("cat1" bis "cat43"), 12 Merkmale haben 3 Ausprägungen, 6 Merkmale haben 4 Ausprägungen, 14 Merkmale haben 5 Ausprägungen oder mehr bis hin zu 51 Ausprägungen. Jeder diese nominalen Merkmale muss für die Auswertung mit einem ML-Verfahren codiert werden.

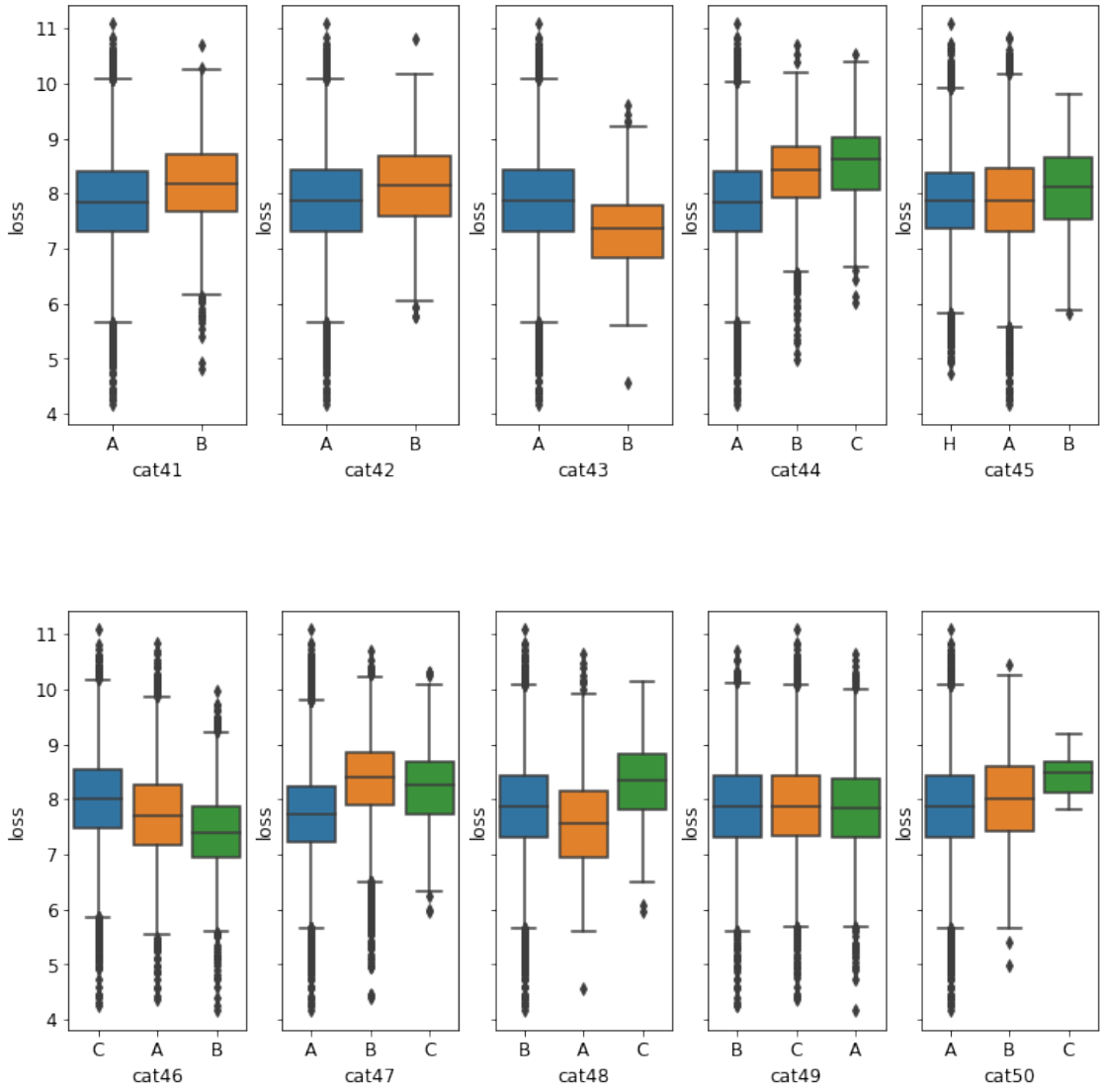
4.2.2 A-12: Welche Erkenntnisse können aus dem logarithmierten Schadenaufwand gezogen werden? Zudem sollen Boxplot-Diagramme für jedes Attribut der kategoriellen Merkmale dargestellt werden. Was ist daraus abzuleiten? (Lernziele 5.2.1, 6.1.2) - [5 Punkte]

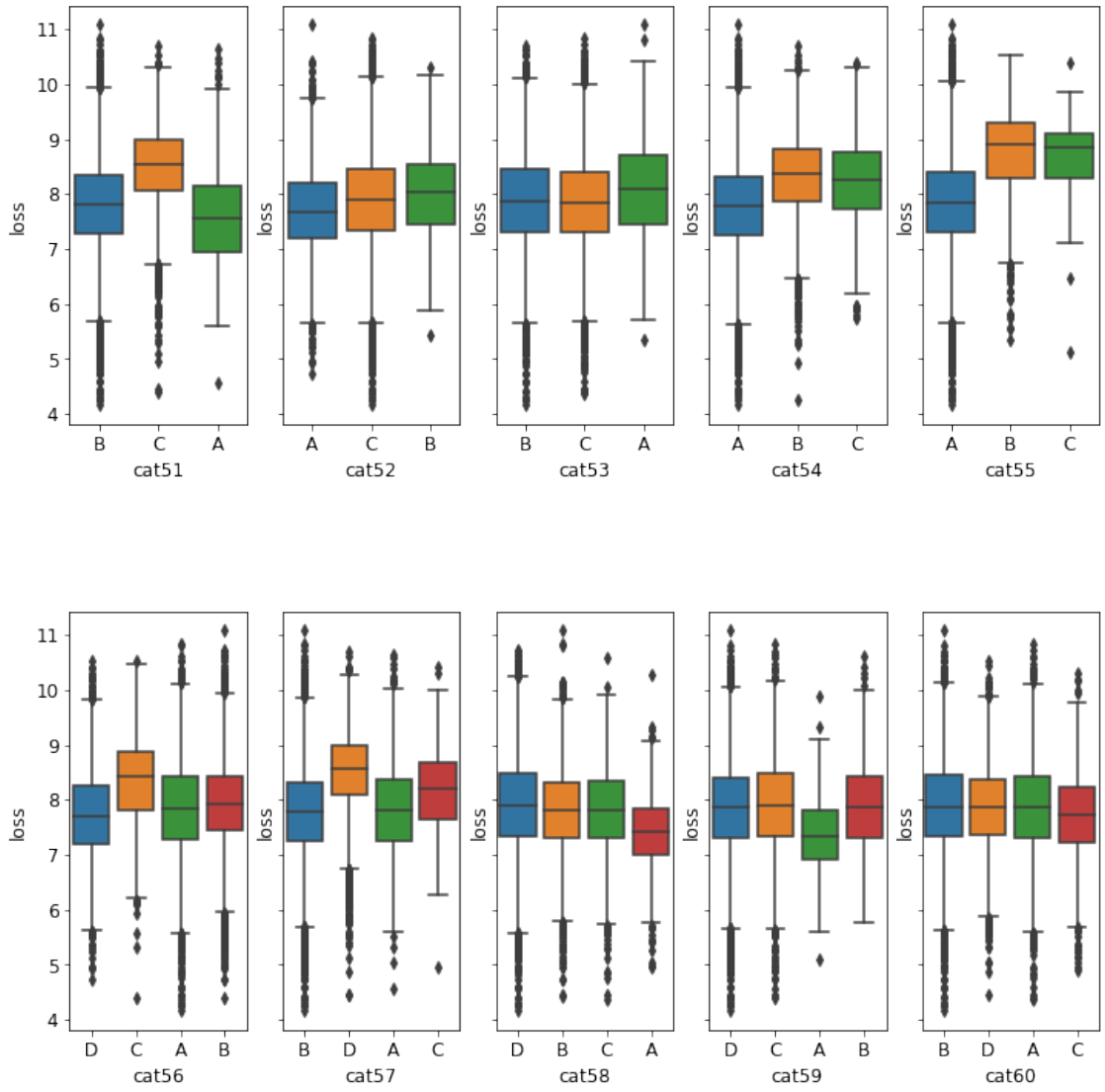
[15]:
```
# Namen aller Spalten
cols = df.columns

# Boxplot für alle Merkmale in einem 12x5 Gitter
n_cols = 5
n_rows = 12
for i in range(n_rows):
    fg,ax = plt.subplots(nrows=1,ncols=n_cols,sharey=True,figsize=(12, 5))
    for j in range(n_cols):
        sns.boxplot(x=cols[i*n_cols+j], y="loss", data=df, ax=ax[j])
```


loss
cat1
cat2
cat3
cat4
cat5
cat6
cat7
cat8
cat9
cat10
cat11
cat12
cat13
cat14
cat15
A
B

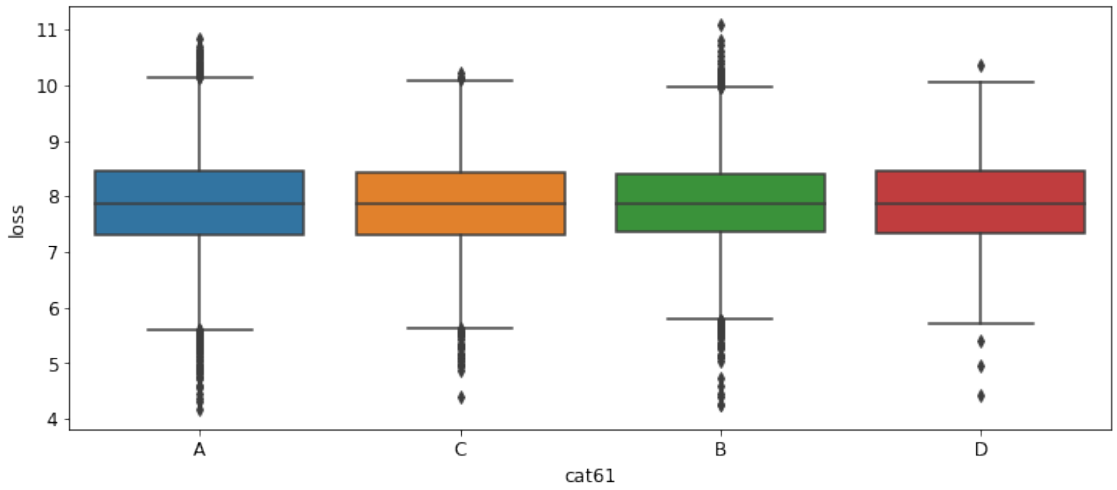

loss
11
10
9
8
7
6
5
4
A
B
cat16
cat17
cat18
cat19
cat20
cat21
cat22
cat23
cat24
cat25

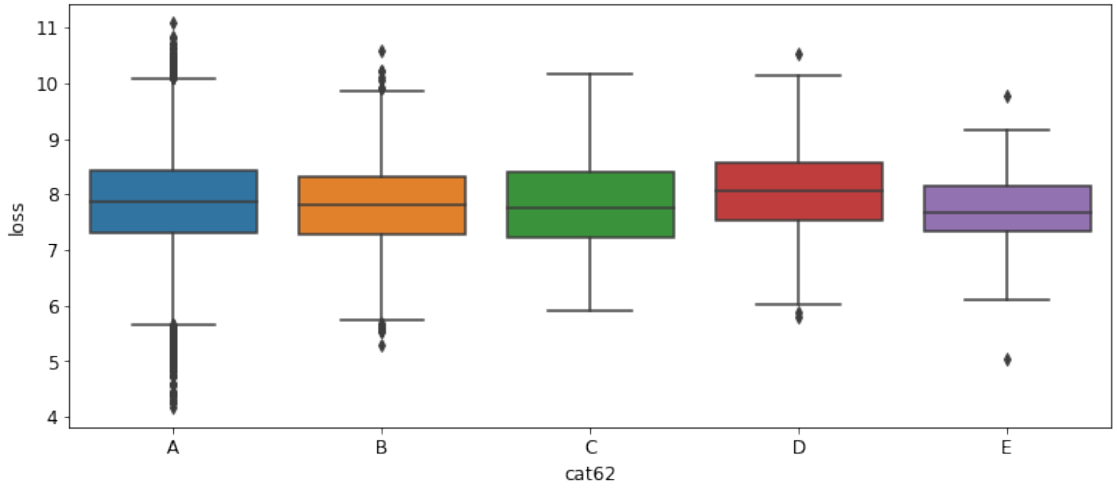

loss
11
10
9
8
7
6
5
4
A
B
cat26
cat27
cat28
cat29
cat30
cat31
cat32
cat33
cat34
cat35
cat36
cat37
cat38
cat39
cat40

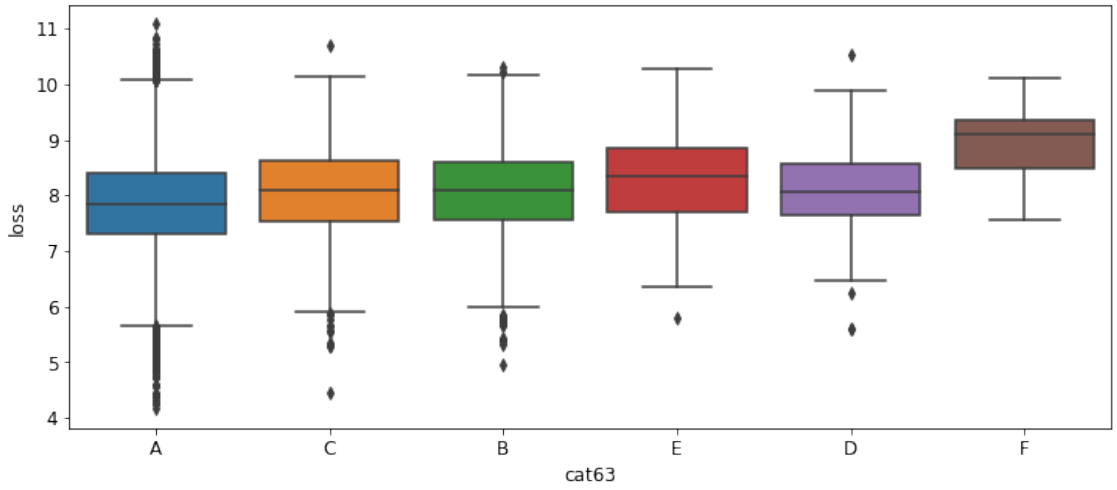

loss
11
10
9
8
7
6
5
4
A
B
cat41
A
B
cat42
A
B
cat43
A
B
C
cat44
H
A
B
cat45
C
A
B
cat46
A
B
C
cat47
B
A
C
cat48
B
C
A
cat49
A
B
C
cat50

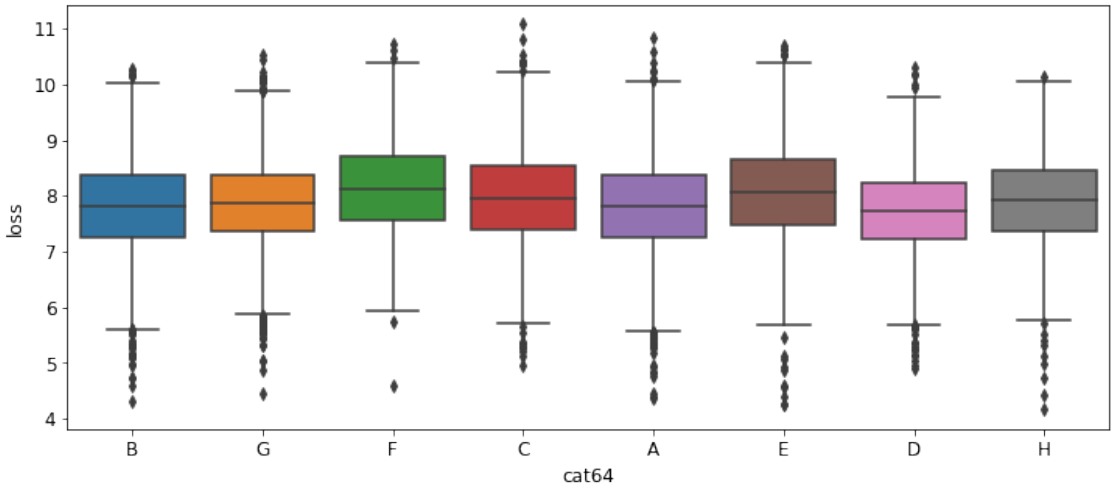


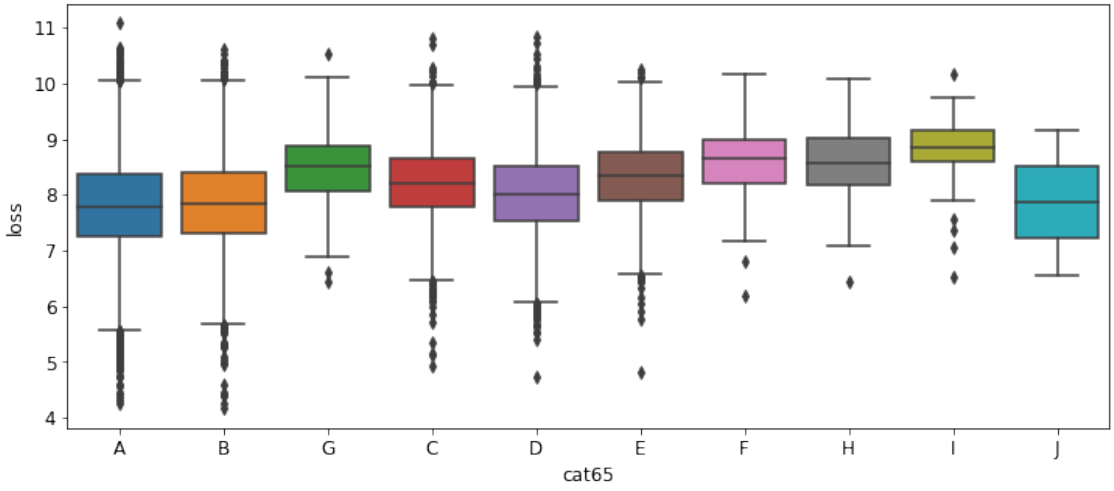
[16]:
```
# Die weiteren Plots einzeln anzeigen
n_rows = 15
for i in range(n_rows):
    fg,ax = plt.subplots(figsize=(12, 5))
    sns.boxplot(x=cols[i+60], y="loss", data=df)
```

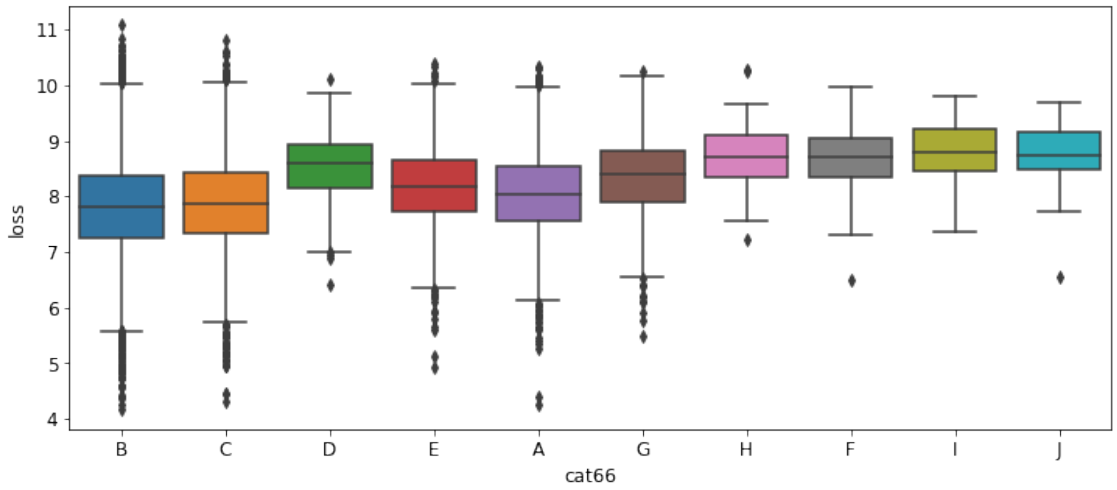

loss
11
10
9
8
7
6
5
4
A
C
B
D
cat61

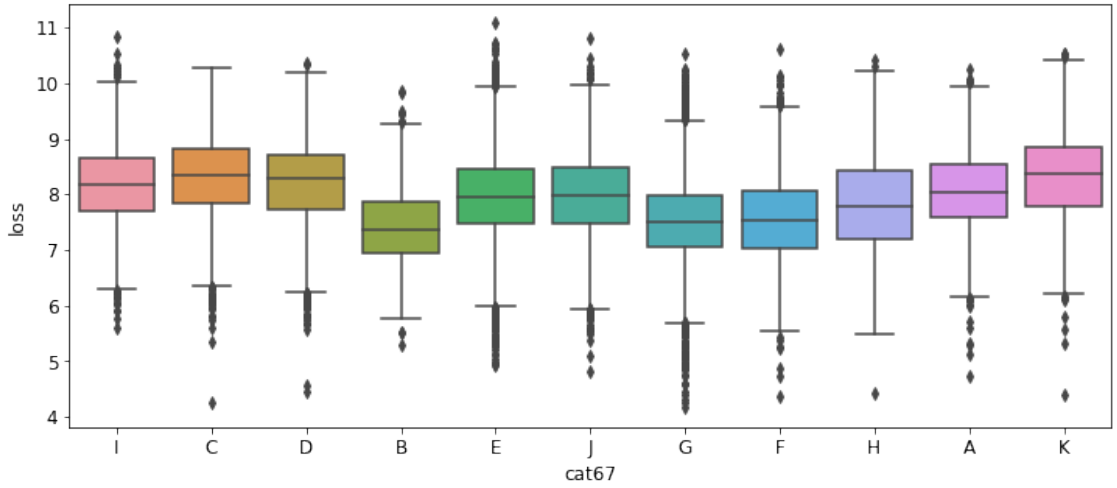

loss
11
10
9
8
7
6
5
4
A
B
C
D
E
cat62

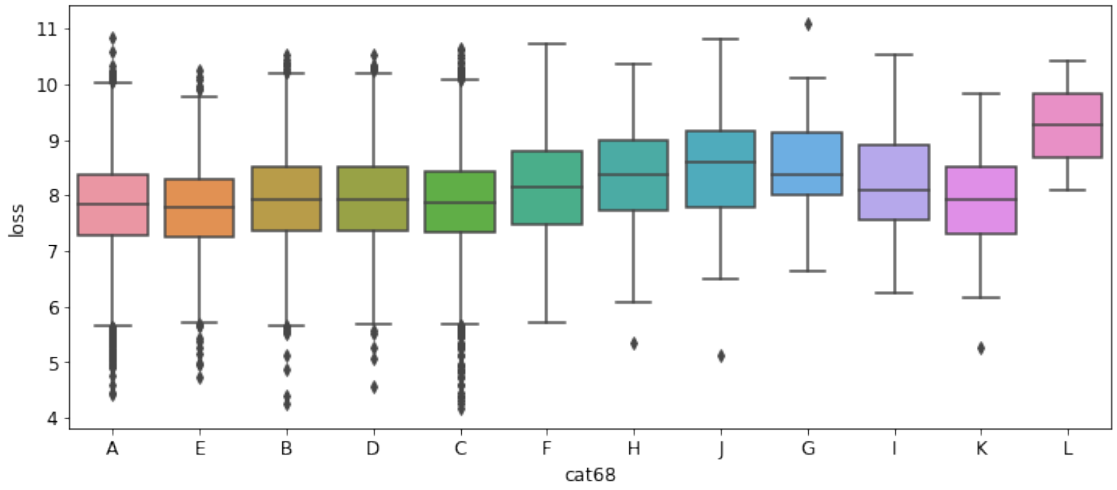

loss
11
10
9
8
7
6
5
4
A
C
B
E
D
F
cat63

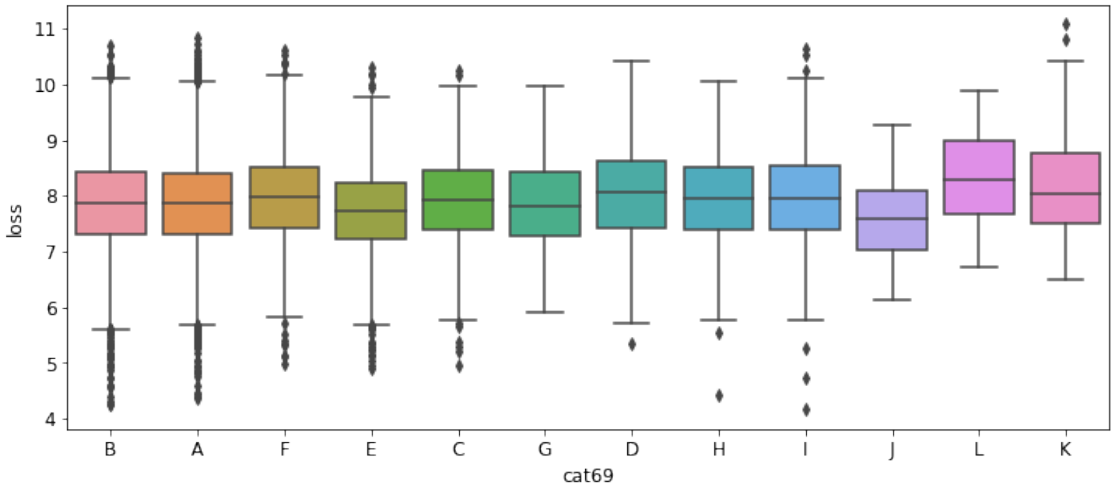

loss
11
10
9
8
7
6
5
4
B
G
F
C
A
E
D
H
cat64

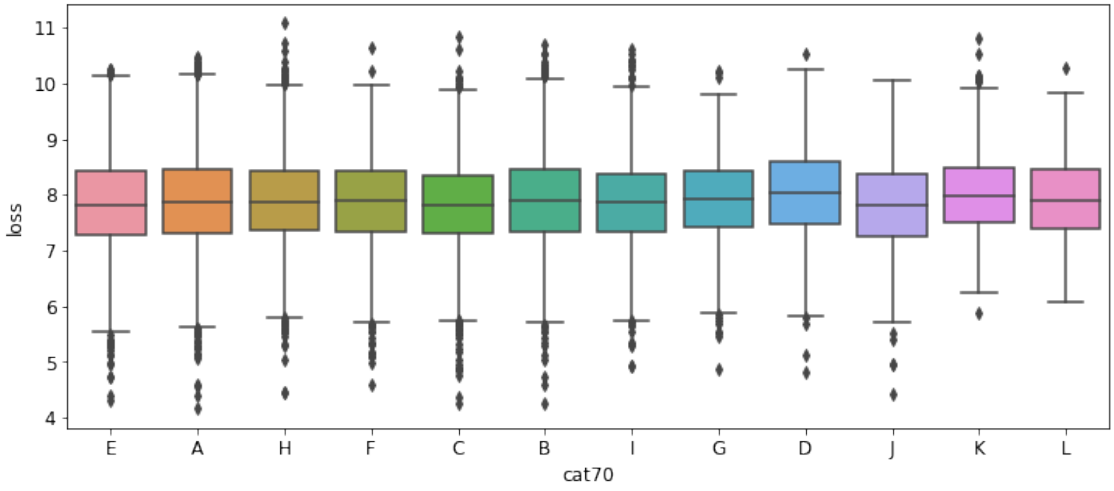

loss
11
10
9
8
7
6
5
4
A
B
G
C
D
E
F
H
I
J
cat65

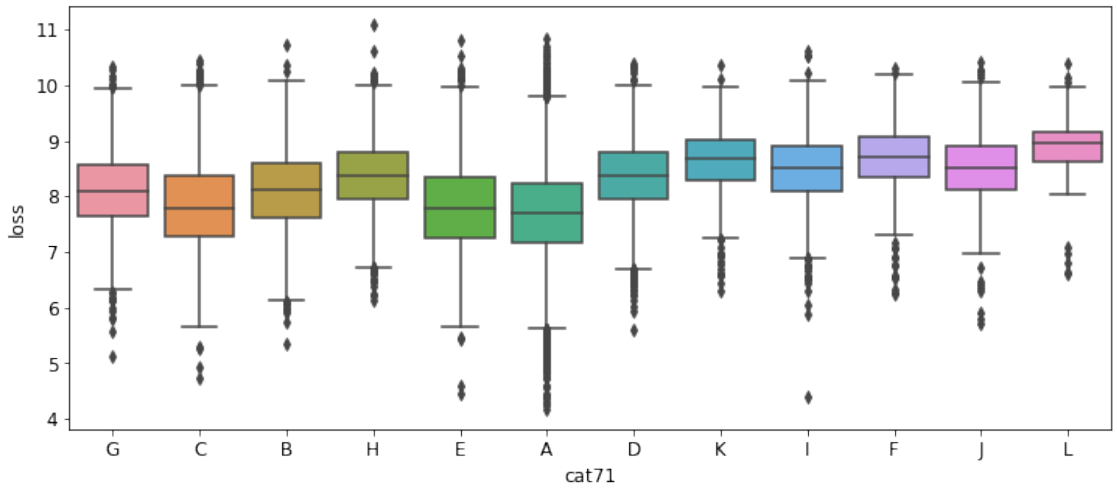

loss
11
10
9
8
7
6
5
4
B
C
D
E
A
G
H
F
I
J
cat66

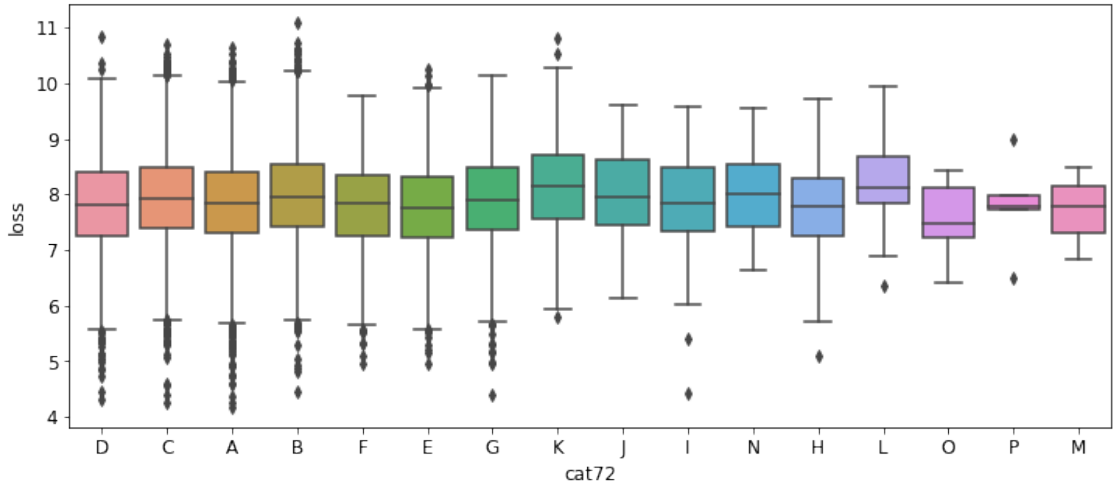

loss
11
10
9
8
7
6
5
4
I
C
D
B
E
J
G
F
H
A
K
cat67

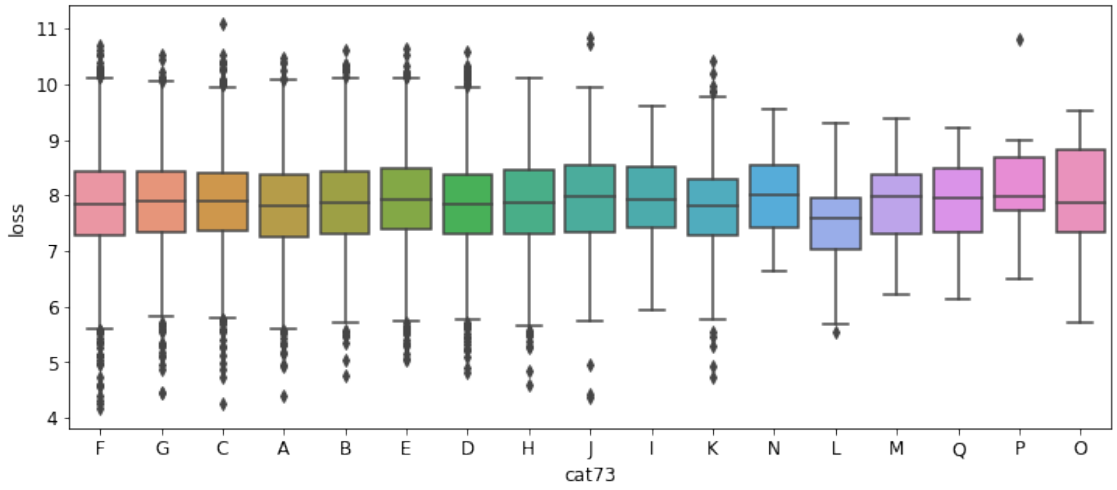

loss
11
10
9
8
7
6
5
4
A
E
B
D
C
F
H
J
G
I
K
L
cat68

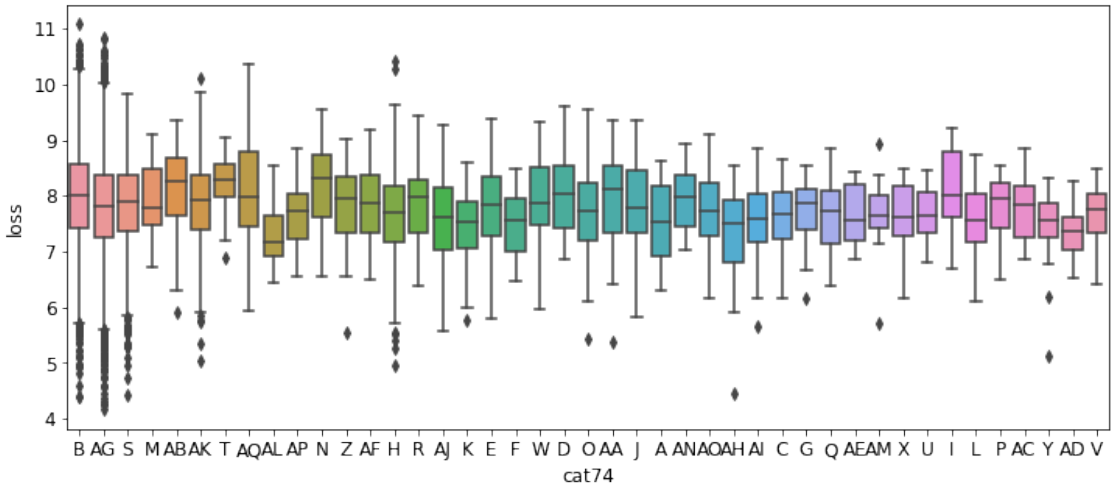

loss
11
10
9
8
7
6
5
4
B
A
F
E
C
G
D
H
I
J
L
K
cat69

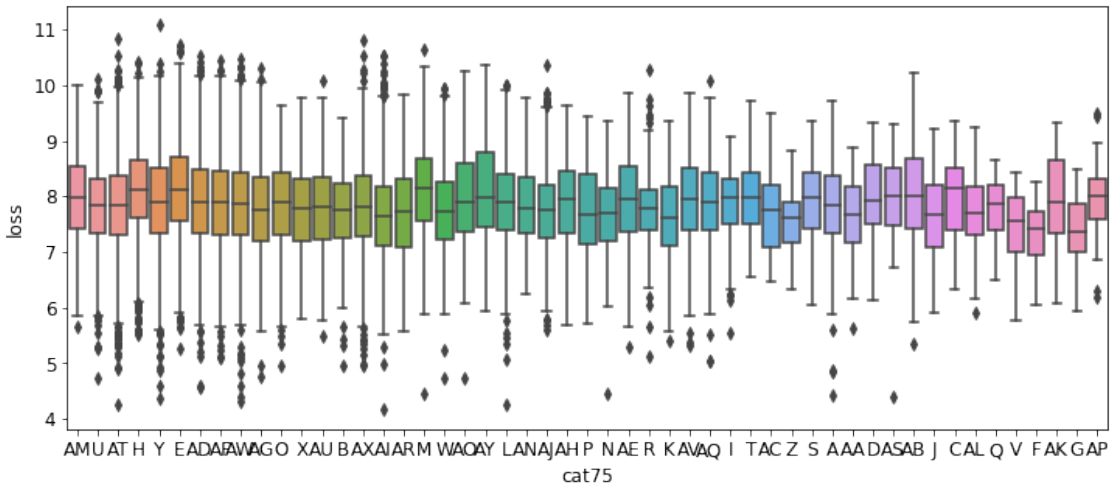

loss
11
10
9
8
7
6
5
4
E
A
H
F
C
B
I
G
D
J
K
L
cat70


loss
11
10
9
8
7
6
5
4
G C B H E A D K I F J L
cat71


loss
11
10
9
8
7
6
5
4
D C A B F E G K J I N H L O P M
cat72


loss
11
10
9
8
7
6
5
4
F G C A B E D H J I K N L M Q P O
cat73





Die Boxplot-Diagramme visualisieren, wie der logarithmische Schadenaufwand bzgl. der einzelnen Merkmalsausprägungen der kategoriellen Variablen differenziert. Insbesondere sind dem Boxplot-Diagramme der Median, die oberen und unteren Quantile und die "Whisker" zu entnehmen. Es wird auch deutlich, für welche Merkmalsausprägungen der logarithmische Schadenaufwand Ausreißer besitzt.

4.3 2.3. Daten für die Modellierung vorbereiten

4.3.1 A-13: Der Zufallsgenerator soll sinnvoll und zielführend eingestellt werden. Welche Möglichkeiten zur Kontrolle des "Random State" gibt es dazu und wie wirken sie sich auf die Reproduzierbarkeit aus? (Lernziel 3.3.1, 3.4.1) - [2 Punkte]

4.3.2 Wenden Sie diesen im weiteren Verlauf zur Erzeugung reproduzierbare Ergebnisse an. Setzen Sie nach Vollendung Ihrer Analyse einen neuen Startwert und überprüfen Sie die Stabilität Ihrer Ergebnisse.

[17]:
```
seed = 123
```

Generell ist es gute Praxis, den Seed für die Erzeugung von Zufallszahlen zu fixieren, um Berechnungsergebnisse reproduzierbar zu machen. Dies ist zum Beispiel mittels Zuweisung eines Seed an numpy.random.seed möglich. Damit können (Pseudo-) Zufallszahlen verschiedener Bibliotheken erzeugt werden. Sofern ein Seed-Objekt existiert, sollte es verwendet werden, wenn es sich für alle eingesetzten Algorithmen eignet. Im Zweifelsfall ist es deshalb besser, vor jedem einzelnen Modellrun einen (dokumentierten) Seed zu verwenden. Auch verworfenen Modelle sollen reproduziert werden können.

Besondere Umsicht ist hier bei der Verwendung von Multiprozessen (z.B. Modul multiprocessing) geboten. Der Seed muss hier explizit gesetzt werden, anderenfalls wird ein Seed anhand der Systemzeit erzeugt. Dieser ist dann nicht nur nicht reproduzierbar, die Seeds können unter Umständen auch übereinstimmen, wenn die Subprozesse gleichzeitig erzeugt werden.

Als Argument kann mithin oft einer von vielen Zufallszahlengeneratoren (bit-Generator) und ein Seed übergeben werden. Besonders geeignet für parallele Anwendungen ist die Klasse der sogenannten "counter-based" Pseudo-Zufallszahlengeneratoren, die auch bei hochzählenden Seeds ausreichend unabhängige Zufallszahlenfolgen liefert.

4.3.3 A-14: Zur sinnvollen Verwendung der kategoriellen Merkmale in Regressionsmodellen müssen diese (häufig) enkodiert werden. Es soll ein "One-Hot-Encoding" auf die kategoriellen Merkmale angewendet werden. Welche anderen Möglichkeiten des Encodings gibt es und wie unterscheiden sie sich bezüglich Praktikabilität und Performance? Welche möglichen Unterschiede zwischen den Trainings- und Testdaten sind hierbei zu beachten? (Lernziel 3.3.1, 3.4.1) - [6 Punkte]

[18]:
```
#Einlesen der Testdaten
df_test = pd.read_csv("../input/claimseverity/cs_test_pk1.csv")
print(df_test.shape)
df_test.tail()
```

```
(30172, 87)
```

[18]:

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	cat10	cat11	cat12	\
30167	101052	A	A	A	B	A	B	A	A	B	A	A	A	
30168	101057	A	B	A	A	B	B	A	A	B	A	A	A	
30169	101058	A	A	A	A	A	B	A	A	B	A	A	A	

```
30170  101059    A    A    A    A    B    A    A    B    B     A     A     A
30171  101079    A    A    A    B    A    B    A    A    B     A     A     A

      cat13 cat14 cat15 cat16 cat17 cat18 cat19 cat20 cat21 cat22 cat23 cat24  \
30167     A     A     A     A     A     A     A     A     A     B     A     A
30168     A     A     A     A     A     A     A     A     A     A     A     A
30169     A     A     A     A     A     A     A     A     A     B     A     A
30170     A     B     A     A     A     A     A     A     B     A     A     B
30171     A     A     A     A     A     A     A     A     A     A     A     A

      cat25 cat26 cat27 cat28 cat29 cat30 cat31 cat32 cat33 cat34 cat35 cat36  \
30167     A     A     A     A     A     A     A     A     A     A     B     A
30168     A     A     B     A     A     A     A     A     A     A     B     A
30169     A     A     A     A     A     A     A     A     A     A     A     A
30170     A     A     A     A     A     A     A     B     A     A     A     B
30171     B     B     B     A     A     A     A     A     A     A     A     A

      cat37 cat38 cat39 cat40 cat41 cat42 cat43 cat44 cat45 cat46 cat47 cat48  \
30167     B     A     A     A     A     A     A     A     A     A     A     B
30168     B     A     A     A     A     A     A     A     H     A     A     B
30169     B     B     A     A     A     A     A     A     A     A     A     B
30170     A     A     A     A     A     A     A     A     H     C     B     B
30171     B     A     A     A     A     A     A     A     H     C     A     B

      cat49 cat50 cat51 cat52 cat53 cat54 cat55 cat56 cat57 cat58 cat59 cat60  \
30167     C     A     B     C     C     A     A     A     B     D     C     A
30168     C     A     B     C     C     A     A     B     B     C     D     D
30169     B     A     B     C     B     A     A     A     B     D     D     B
30170     B     A     C     A     B     C     A     B     B     D     D     B
30171     B     A     B     A     B     A     A     D     B     D     D     B

      cat61 cat62 cat63 cat64 cat65 cat66 cat67 cat68 cat69 cat70 cat71 cat72  \
30167     A     A     A     A     A     B     H     C     A     C     A     C
30168     B     A     A     G     B     B     E     C     A     H     A     C
30169     A     A     A     B     A     B     H     C     B     E     A     D
30170     D     A     E     B     B     B     E     A     B     J     G     I
30171     A     A     A     B     A     E     E     D     B     A     A     D

      cat73 cat74 cat75     cont1     cont2     cont3     cont4     cont5  \
30167     D    AG     Y  0.659441  0.464374  0.491183  0.247520  0.467797
30168     G    AG     L  0.865913  0.299718  0.500715  0.335910  0.538319
30169     E    AG     Y  0.293503  0.694709  0.529271  0.394592  0.509058
30170     D     B    AQ  0.156619  0.358369  0.139691  0.798304  0.615124
30171     F     B    AI  0.387685  0.604411  0.230108  0.339674  0.151509

          cont6     cont7     cont8     cont9    cont10          loss
30167  0.550505  0.232383  0.594124  0.435485  0.561000   8127.221924
```

```
30168  0.512945  0.467870  0.610032  0.295336  0.506225   2207.423078
30169  0.490603  0.473228  0.556024  0.358398  0.387013  15486.423301
30170  0.384784  0.643973  0.356930  0.491521  0.436456   6862.352181
30171  0.319026  0.438558  0.495021  0.382283  0.558410   3110.558275
```

[19]:
```
#Speichere id's für das submission file
ID = df_test['id']
#Unnötige Spalten entfernen
df_test.drop('id',axis=1,inplace=True)
```

Beim Encoding ist darauf zu achten, dass Trainings- und Testdaten gemeinsam encodiert werden, um sicherzustellen dass in beiden Datensätze das Auftreten der selben Merkmalsausprägungen durch das selbe Encoding repräsentiert wird. Das gilt insbesondere, falls nicht alle Merkmalsausprägungen in einem der beiden Datensets vorhanden sein sollen.

Alternative Encoding-Methode: Target-Encoding (Mean Encoding). Das kategorielle Merkmal wird durch den Durchschnittswert eines Target-Merkmals für diese Merkmalsausprägung ersetzt. Vorteil: Keine Erhöhung der Dimension, keine dünn besetzten Input-Matrizen führen zu einer höheren Performance. Nachteil: Kein direkter Zusammenhang zwischen kategoriellem Merkmal und Vorhersage, nur zwischen dem Verhältnis vom kategoriellen Merkmal und Target-Merkmal.

[20]:
```
# Dummy-Variablen ("One-Hot-Encoding") für alle Merkmalskategorien in der␣
 ↪Trainings-
# und Testdatei einheitlich anlegen:

# "combine"
df_all = pd.concat([df.assign(role="train"), df_test.assign(role="test")])
df_all = pd.get_dummies(df_all)
# "seperate"
df_test_dummies, df_dummies = df_all[df_all["role_test"].eq(1)],␣
 ↪df_all[df_all["role_train"].eq(1)]
df_dummies.drop(['role_test','role_train'],axis=1,inplace=True)
df_test_dummies.drop(['role_test','role_train','loss'],axis=1,inplace=True)
# Quelle "combine-seperate": https://datascience.stackexchange.com/questions/
 ↪81617/how-to-combine-and-separate-test-and-train-data-for-data-cleaning

df_dummies.shape
```

[20]:
```
(70907, 382)
```

4.3.4 A-15: Das Ergebnis des "One-Hot-Encoding" ist zu erläutern. Was sind die zu erwartenden Folgen? Treten hier Probleme hinsichtlich des "Curse of Dimensionality" auf? Welcher Art sind diese und welcher Umgang sollte damit getroffen werden? (Lernziel 5.1.1) - [5 Punkte]

Für das "One-Hot-Encoding" werden für jede mögliche Merkmalsausprägung eines Merkmals sogenannte Dummy-Merkmale eingeführt und mit 1 belegt, falls das Merkmal die entsprechende Ausprägung hat, sonst mit 0. Nach dem "One-Hot-Encoding" weist der Datensatz 382 Merk-

male auf, die Methode führt also zu einer erheblichen Dimensionssteigerung. Dies kann zu einem erheblichen Performance-Verlust beim Training führen. Insbesondere bei der hohen Anzahl der kategoriellen Merkmale mit nur zwei Ausprägungen ist darauf zu achten, dass gegebenenfalls ein Dummy-Merkmal ausreicht, um das kategorielle Merkmal ohne Informationsverlust zu codieren. Ein weiteres Performance-Problem kann auftreten, weil der Optimierungsalgorithmus bei dünn besetzten Input Matrizen keine guten Ergebnisse liefert, dies gilt insbesondere für neuronale Netze.

4.3.5 A-16: Auf Basis der Trainingsdaten sind die üblichen ML-Matrizen X und y für das Modelltraining und für die Modellvalidierung zu erzeugen. (Lernziele 3.3.3, 3.4.5) - [2 Punkte]

[21]:
```
# Daten in internes Training und Validierung (25%) unterteilen

X = df_dummies.drop(['loss'], axis=1)
y = df_dummies['loss']
X_train, X_val, y_train, y_val=train_test_split(X, y, test_size=0.25,␣
 ↪random_state=seed)
```

[22]:
```
# Anzahl der Features ("One-Hot-Encoding")
n_features = X_train.shape[1]
n_features
```

[22]:
```
381
```

[23]:
```
# Anzahl fehlender Werte ermitteln
df.isna().sum().sum()
```

[23]:
```
0
```

5 3. Modellierung

5.0.1 A-17: Es sollen Datenstrukturen vorbereitet werden, die einen komfortablen Vergleich der Modelle und Ergebnisse erleichtern. Worauf gilt es hierbei zu achten und welche Aspekte sind in Bezug auf die Gütemaße bereits vor der Modellierung zu überlegen? Welche Vor- und Nachteile hat das vorgegebene Gütemaß mittlerer absoluter Fehler (MAE) im Vergleich zum mittleren quadratischen Fehler (MSE) im Hinblick auf die Vorhersagegenauigkeit auf individueller Ebene sowie die Erwartungstreue des Schätzers? (Lernziel 4.1.7) - [5 Punkte]

[24]:
```
#Alle Features
X_all = []

#Liste aller Kombinationen
comb = []

#Liste für den MAE aller Algorithmen
```

```
mae = []
```

Generell hängt die Wahl eines geeigneten Güßemaßes davon ab, ob es sich um ein Regressions- oder Klassifikationsproblem handelt. Hier handelt es sich um ein Regressionsproblem, dafür bieten sich

- Mittlerer quadratischer Fehler (MSE)
- Mittlerer absoluter Fehler (MAE)
- Bestimmtheitsmaß R^2

an. Das Gütemaß mittlerer absoluter Fehler liefert bei typischen Schadenverteilungen genauere individuelle Prognosen der Schadenhöhen, ist aber im Gegensatz zum mittleren quadratischen Fehler nicht erwartungstreu. Im vorliegenden Fall einer stark rechtsschiefen Verteilung wird bei Verwendung des MAE die Gesamtschadenhöhe unterschätzt.

5.1 3.1. Entscheidungsbaumbasierte Modellensembles

5.1.1 3.1.1. Random-Forest-Verfahren

5.1.2 A-18: Zunächst soll das Random-Forest-Verfahren verwendet werden. Es sollen die Hyperparameter Baumtiefe und Baumanzahl über eine Gittersuche bestimmt werden. Was sind Hyperparameter, was ist ihre Bedeutung? (Lernziele 4.1.3, 4.1.4) - [10 Punkte]

[25]:
```
# RandomForestRegressor: Hyperparameteroptimierung mit GridSearchCV
tic = time.time()

# Parameter mit Rücksicht auf die Laufzeit angepasst
param_grid ={'max_depth':[35,50],'max_features':[50,75], 'n_estimators':[100]}
RF_grid_search = GridSearchCV(RandomForestRegressor(), param_grid,␣
 ↪scoring='neg_mean_absolute_error', n_jobs=-1, cv=2)
RF_grid_search.fit(X_train,y_train)
print("Beste Parameter:",RF_grid_search.best_params_)

# Erstelle neues Modell auf allen Folds mit den besten Parametern
RF2 = RandomForestRegressor(**RF_grid_search.best_params_)
RF2.fit(X_train, y_train)

print("time (sec):" + "%6.0f" % (time.time() - tic))

# MAE an Validierungsdaten berechnen
result = mean_absolute_error(np.expm1(y_val), np.expm1(RF2.predict(X_val)))
print("\nMAE: " + "%6.2f" % result)

mae.append(result)
comb.append("RandomForest")
```

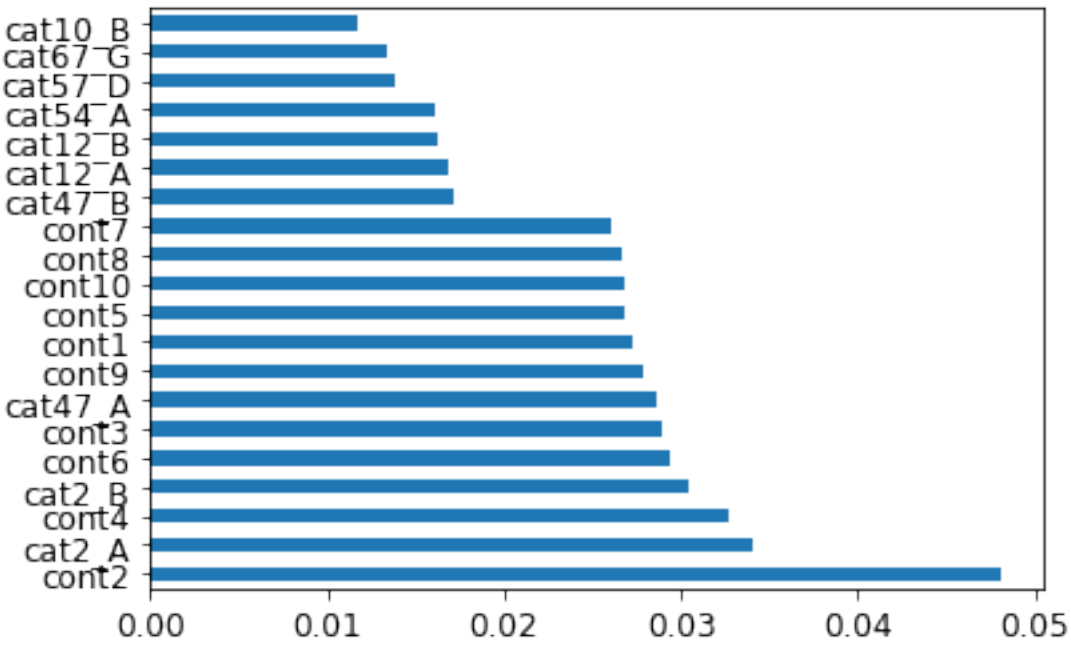
```
Beste Parameter: {'max_depth': 35, 'max_features': 50, 'n_estimators': 100}
time (sec):   252
```

```
MAE: 1467.81
```

Hyperparameter sind Parameter, die gewählt werden können, um den Lernprozess zu steuern. Im Gegensatz dazu werden Modellparameter (z.B. Gewichte) durch das Training gelernt. Die Wahl der Hyperparameter ist daher entscheidend für den Trainingserfolg. Der Erfolg von ML-Anwendungen durch gesteigerter Rechenleistung ist auch auf die Möglichkeit der automatisierten Hyperparameteroptimierung zurückzuführen. Dafür wird das gewählte Modell für eine Vielzahl an Hyperparametern trainiert und die Modellergebnisse bzgl. des gewählten Gütemaßes verglichen.

5.1.3 A-19: Die Variablen-Importance (Z. 4.1.4) soll berechnet werden und Auskunft über die Eigenschaften des trainierten Lerners geben. Welche Möglichkeiten zur Berechnung und Visualisierung am Beispiel des Random-Forest sind hier zielführend? (Lernziel 4.1.4) - [8 Punkte]

[26]:
```
# Random-Forest: Feature-Importance plotten
(pd.Series(RF2.feature_importances_, index=X.columns).nlargest(20).
 ↪plot(kind='barh'))
plt.show()
```



Die Variablen-Importance oder Feature-Importance ist ein Maß, das angibt welchen Beitrag jedes einzelne Feature zum Vorhersageergebnis leistet. Dies ist also ein abstraktes und Modell-abhängiges Konzept. Für lineare Regressionsmodelle lässt sich beispielsweise der Beitrag einzelner Input-Variablen zum Vorhersageergebnis an den Regressionskoeffizienten ablesen. Für Regressions- und Klassifikationsprobleme werden andere Maße verwendet. Diese Werte sind Modell-übergreifend also nur schwer zu vergleichen, können aber wertvolle Informationen zur Feature-Selection oder für das Feature-Engineering liefern.

Beim Random-Forest gibt es zwei wichtige Konzepte für die Variablen-Importance. Zum einen die mittlere Abnahme der Summe der quadratischen Abstände (mean decrease in node impurity bzw. average impurity reduction) und die Permutation-Importance (mean decrease in accuracy). Als Feature-Importance wird bei Random-Forest als Standard die erste Kennzahl ausgegeben.

5.1.4 3.1.2 Gradient-Boosting-Verfahren

5.1.5 A-20: Die hinter den Gradient-Boosting-Tools lightGBM, XGBoost und CatBoost stehende Methodik ist zu beschreiben. Welche Unterschiede, Vor- und Nachteile lassen sich identifizieren? Welches Konzept eignet sich im vorliegenden Fall? (Lernziele 3.2.1, 4.1.7) - [8 Punkte]

XGBoost: Open Source Bibliothek und bekannteste Implementierung der Gradient-Boosting-Methodik. Beim Gradient-Boosting wird ein starkes Prognosemodell durch die Bildung eines Ensembles aus schwachen Entscheidungsbaum-Modellen gebildet. Dabei wird stufenweise ein Ensemble aus M Entscheidungsbäumen gebildet. In jeder Stufe m wird ein weiterer Entscheidungsbaum an die (Pseudo)-Residuen (die Komponenten des negativen Gradienten) der Stufe m angepasst. Der Entscheidungsbaum der Stufe m korrigiert also den Vorhersagefehler des Prediktors der Stufe m-1. Neben dem Gradient-Boosting-Algorithmus implementiert die Bibliothek Erweiterungen und Regularisierungsverfahren.

Vorteil: * Lernalgorithmen mit einer sehr hohen Vorhersagegenauigkeit. * Für C++, Java, Python, R und Julia verfügbar. * Unterstützt paralleles und verteiltes Lernen.

Nachteil: * Gegebenenfalls zeitaufwendiges Training.

lightGBM: Open Source Bibliothek ursprünglich von Microsoft entwickelt. Bei lightGBM werden die Entscheidungsbäume nicht Stufen-weise sondern “Blatt”-weise aufgebaut. Für den Split wird ein hochoptimierter Histogramm-basierter Entscheidungsbaum-Algorithmus angewendet, der sehr effizient und Speicher-schonend ist.

Vorteile: * Schnelles Training und hohe Effizienz. * Höhere Vorhersagegenauigkeit als andere GB-Verfahren. * Speicherschonend. * Unterstützt paralleles und verteiltes Lernen.

CatBoost: Open Source Bibliothek entwickelt von Yandex. Diese implementiert insbesondere den CatBoost-Algorithmus, der zwei Verbesserungen zum ursprünglichen GB-Verfahren liefert. Zum einen wird das “ordered boosting”, eine Permutations-getriebene Alternative zum GB-Algorithmus implementiert. Zum anderen wird ein Algorithmus für die effiziente Verarbeitung kategorieller Features implementiert.

Vorteile: * Besonders Effizient für kategorielle Features.

Nachteile: * Verfügbar für Python und R, trainierte Modelle müssen für andere Sprachen und Formate exportiert werden.

5.1.6 A-21: Es ist eine Hyperparameteroptimierung über eine mehrdimensionale "Randomized Search" für lightGBM durchzuführen. (Lernziele 3.2.1, 4.1.7) - [8 Punkte]

[27]:
```
# LGBMRegressor: Hyperparameteroptimierung mit RandomizedSearchCV
tic = time.time()
# Parameter mit Rücksicht auf die Laufzeit angepasst
param_grid ={'learning_rate': [0.015,0.02], 'num_leaves':␣
 ↪[75,100,125],'n_estimators': [750] }
LGB_random_search = RandomizedSearchCV(LGBMRegressor(),param_grid,␣
 ↪scoring='neg_mean_absolute_error', cv=4,  n_iter=3, random_state=seed)
LGB_random_search.fit(X_train,y_train)
print("Beste Parameter:",LGB_random_search.best_params_)
print("time (sec):" + "%6.0f" % (time.time() - tic))
```

```
Beste Parameter: {'num_leaves': 100, 'n_estimators': 750, 'learning_rate':
0.015}
time (sec):   222
```

[28]:
```
# LGBMRegressor: Erstelle neues Modell auf allen Folds mit den besten Parametern
tic = time.time()
LGB2 = LGBMRegressor(**LGB_random_search.best_params_)
LGB2.fit(X_train, y_train)
print("time (sec):" + "%6.0f" % (time.time() - tic))

# Validierung MAE
result = mean_absolute_error(np.expm1(y_val), np.expm1(LGB2.predict(X_val)))
print("\nMAE: " + "%6.2f" % result)

mae.append(result)
comb.append("lightGBM")
```

```
time (sec):    20

MAE: 1412.72
```

5.1.7 A-22: Es ist eine Hyperparameteroptimierung über eine mehrdimensionale "Randomized Search" für XGBoost durchzuführen. (Lernziele 3.2.1, 4.1.7) - [8 Punkte]

[29]:
```
# XGBRegressor: Hyperparameteroptimierung mit RandomizedSearchCV
tic = time.time()
# Parameter mit Rücksicht auf die Laufzeit angepasst
param_grid ={'learning_rate': [0.03,0.04],  'max_depth':[5,6], 'n_estimators':␣
 ↪[600],'tree_method': ['hist'], 'colsample_bytree':[0.7], 'subsample':[0.7]}
XGB_random_search = RandomizedSearchCV(XGBRegressor(),param_grid,␣
 ↪scoring='neg_mean_absolute_error', cv=4,  n_iter=3, random_state=seed)
```

```
XGB_random_search.fit(X_train,y_train)

print("Beste Parameter:",XGB_random_search.best_params_)
print("time (sec):" + "%6.0f" % (time.time() - tic))
```

```
Beste Parameter: {'tree_method': 'hist', 'subsample': 0.7, 'n_estimators': 600,
'max_depth': 6, 'learning_rate': 0.04, 'colsample_bytree': 0.7}
time (sec):    606
```

[30]:
```
# XGBRegressor: Erstelle neues Modell auf allen Folds mit den besten Parametern
tic = time.time()
GB2 = XGBRegressor(learning_rate=0.04, max_depth=6, n_estimators=600,␣
 ↪tree_method='hist', colsample_bytree=0.7, subsample=0.7)
GB2.fit(X_train, y_train)
print("time (sec):" + "%6.0f" % (time.time() - tic))

# Validierung MAE
result = mean_absolute_error(np.expm1(y_val), np.expm1(GB2.predict(X_val)))
print("\n MAE: " + "%6.2f" % result)

mae.append(result)
comb.append("XGBoost")
```

```
time (sec):    57

 MAE: 1415.18
```

5.1.8 A-23: Es soll Catboost mit Default-Werten angewendet werden. (Lernziele 3.2.1, 4.1.7) - [4 Punkte]

[31]:
```
# CatBoostRegressor: Mit Default-Parametern
tic = time.time()
CGB = CatBoostRegressor(logging_level='Silent')
CGB.fit(X_train, y_train)
print("time (sec):" + "%6.0f" % (time.time() - tic))

# Validierung MAE
result = mean_absolute_error(np.expm1(y_val), np.expm1(CGB.predict(X_val)))
print("\nMAE: " + "%6.2f" % result)

mae.append(result)
comb.append("CatBoost")
```

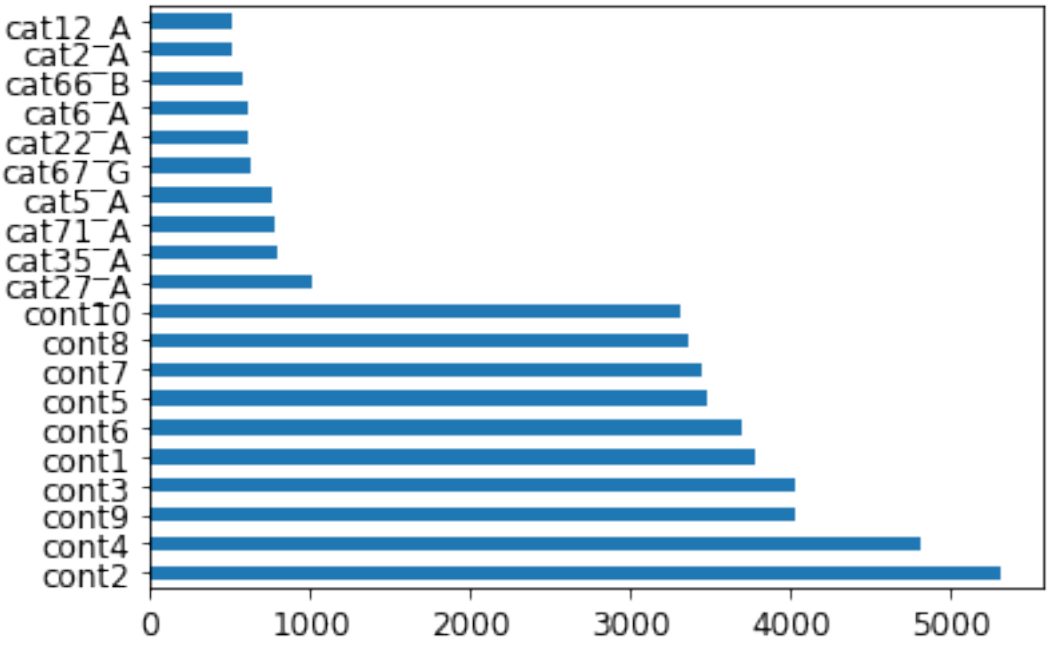
```
time (sec):    25

MAE: 1411.84
```

5.1.9 A-24: Es sind die Ergebnisse und Laufzeiten von A-21 bis A-23 zu vergleichen und zu bewerten. (Lernziele 3.2.1, 4.1.7) - [4 Punkte]

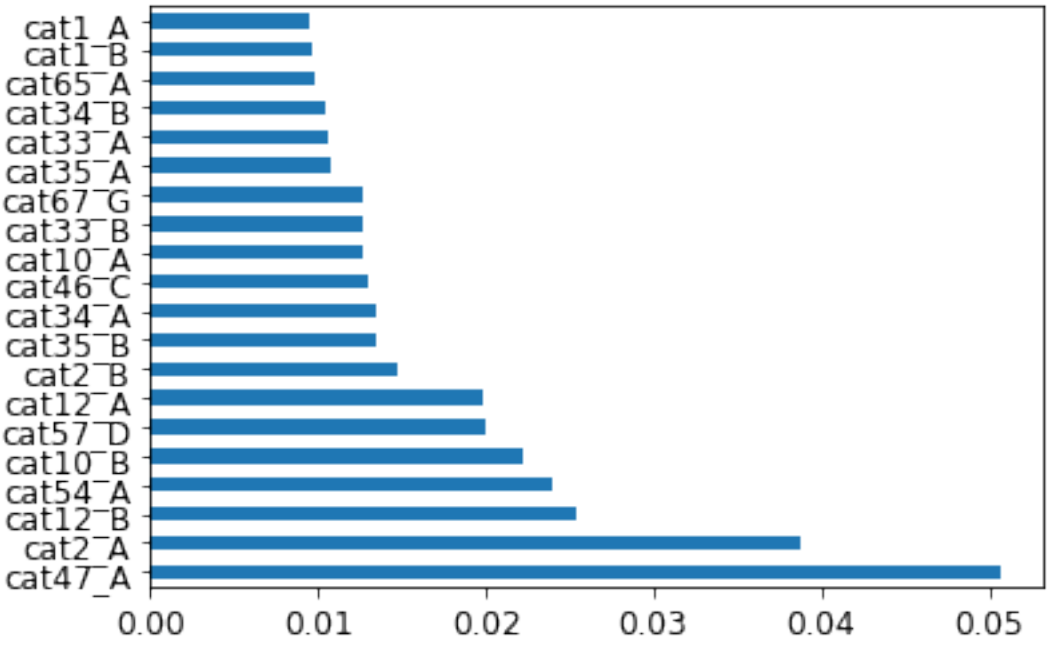
lightGBM hat die kürzeste Trainingszeit und liefert mit MAE 1412.72 eine sehr gute Vorhersagegüte. XGBoost benötigt für das Training fast doppelt so lange, liefert aber immer noch eine sehr gute Vorhersagegüte. CatBoost benötigt etwas länger als lightGBM für das Training und liefert für diesen Datensatz bereits mit Standardeinstellungen die beste Vorhersagegüte mit MAE 1411.84.

5.1.10 A-25: Es soll für die in A-21, A-22 und A-23 trainierten Modelle die "Variable-Importance" berechnet und für die 20 wichtigesten Merkmale visualisiert werden. Welche Unterschiede im Vergleich zum RF-Verfahren ergeben sich und worauf sind diese zurückzuführen? (Lernziel 4.1.4) - [4 Punkte]

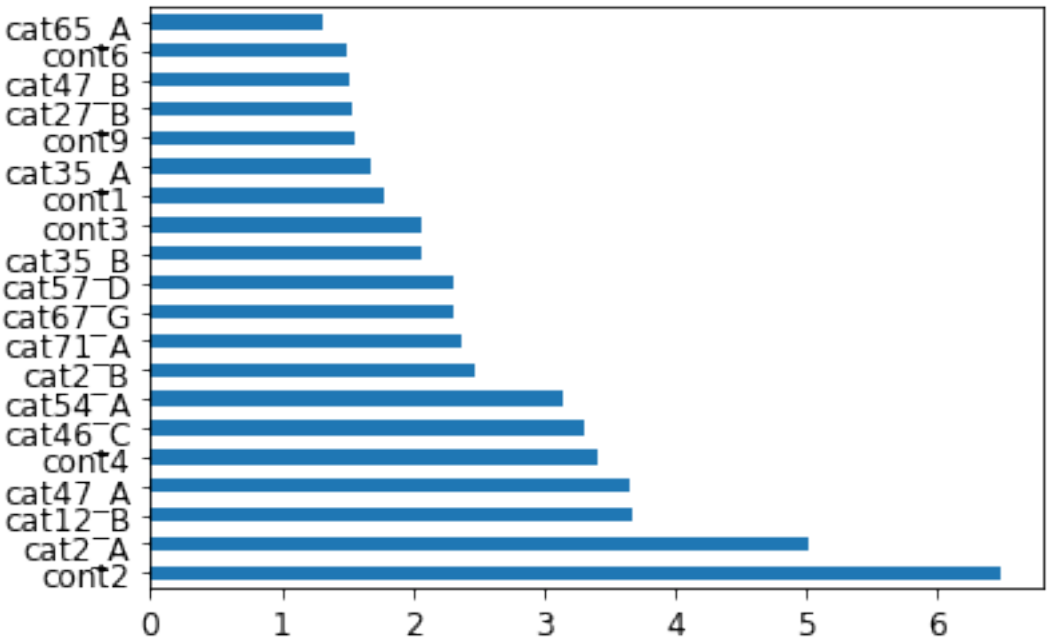
[32]:
```
# lightGBM: Feature-Importance plotten
(pd.Series(LGB2.feature_importances_, index=X.columns).nlargest(20).
 ↪plot(kind='barh'))
plt.show()
```



[33]:
```
# XGBoost: Feature-Importance plotten
(pd.Series(GB2.feature_importances_, index=X.columns).nlargest(20).
 ↪plot(kind='barh'))
plt.show()
```



[34]:
```
# CatBoost: Feature-Importance plotten
(pd.Series(CGB.feature_importances_, index=X.columns).nlargest(20).
 ↪plot(kind='barh'))
plt.show()
```



Allgemein ist festzustellen, dass alle Feature-Importance Visualisierungen eine andere Reihenfolge der Feature-Importance zeigen. Dies ist zum einen auf die unterschiedlichen Definitionen der Feature-Importance zurückzuführen, aber auch auf die Unterschiede in den Optimierungsalgorithmen.

Für GB-Verfahren gibt es unterschiedliche Typen von Feature-Importance. Bei LGBMRegressor und XGRegressor wird als Default die gewichtete Anzahl der Verwendung des Features in den Splits ausgegeben. Bei CatBoost wird als Standard der "Predicted Value Change" ausgegeben, die durchschnittliche Vorhersageänderung relativ zur Änderung des Features. Diese Definitionen weichen von der Feature-Importance des RF Verfahrens ab.

5.2 3.2. Regularisierte lineare Modelle

5.2.1 3.2.1 Datenaufbereitung

[35]:
```
# Skalierung der Daten: Min-Max-Scaling
scaler = MinMaxScaler()
Xs = scaler.fit_transform(X)
Xs_train, Xs_val, y_train, y_val = train_test_split(Xs, y, test_size=0.25,␣
 ↪random_state=seed)

# Die ersten 5 Trainingsdatensätze mit skalierten Features anlisten
pd.DataFrame(Xs_train).head(5) # (... und zuvor ndarray in DataFrame umwandeln)
```

[35]:

	0	1	2	3	4	5	6	\
0	0.377569	0.272540	0.505232	0.424899	0.505892	0.662452	0.470157	
1	0.353508	0.441195	0.040810	0.524294	0.520607	0.496243	0.370595	
2	0.663116	0.389215	0.147559	0.210734	0.331765	0.398076	0.377881	
3	0.622625	0.566389	0.179382	0.380713	0.456300	0.556805	0.341770	
4	0.648464	0.444289	0.501221	0.842879	0.790309	0.512023	0.530223	

	7	8	9	10	11	12	13	14	15	16	17	18	\
0	0.107116	0.455499	0.322925	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	
1	0.312473	0.441021	0.448556	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	
2	0.451853	0.387844	0.508115	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	
3	0.538839	0.360505	0.544365	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	
4	0.299681	0.279892	0.596080	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	

	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	\
0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	
1	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	
2	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	
3	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	
4	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	

	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	\
0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	
1	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	

```
2  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0
3  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0
4  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0

    49   50   51   52   53   54   55   56   57   58   59   60   61   62   63  \
0  0.0  1.0  0.0  0.0  1.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0
1  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0
2  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0
3  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0
4  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0

    64   65   66   67   68   69   70   71   72   73   74   75   76   77   78  \
0  1.0  0.0  1.0  0.0  0.0  1.0  0.0  1.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0
1  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0
2  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  0.0
3  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  0.0  1.0  1.0  0.0  1.0
4  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0

    79   80   81   82   83   84   85   86   87   88   89   90   91   92   93  \
0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0
1  0.0  0.0  1.0  1.0  0.0  1.0  0.0  1.0  0.0  0.0  1.0  1.0  0.0  1.0  0.0
2  1.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0
3  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0
4  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0  1.0  0.0

    94   95   96   97   98   99  100  101  102  103  104  105  106  107  108  \
0  1.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0
1  1.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0
2  1.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0  0.0
3  1.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0
4  1.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0

   109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  \
0  1.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0
1  1.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0
2  1.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0
3  1.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0  0.0
4  1.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0

   124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  \
0  0.0  1.0  1.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0
1  0.0  1.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0
2  0.0  1.0  1.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0
3  0.0  1.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  1.0  0.0
4  0.0  1.0  1.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0

   139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  \
```

```
0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0
1  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  1.0  0.0
3  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0  1.0  0.0
4  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0  1.0  0.0

   154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  \
0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
1  1.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
2  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
4  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0

   169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  \
0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0
2  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0
4  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

   184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  \
0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0

   199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  \
0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0
3  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0

   214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  \
0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

   229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  \
0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
1  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
3  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
4  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	\
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	

	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	\
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	

	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	\
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	\
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	\
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	\
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	\
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

```
3  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

   349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  \
0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

   364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  \
0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

   379  380
0  0.0  0.0
1  0.0  0.0
2  0.0  0.0
3  0.0  0.0
4  0.0  0.0
```

[36]:
```
# Plausibilisierung: Datensätze anzeigen und mit der entsprechenden Auflistung
# in Abschnitt 2.4 vergleichen
y_val.head()
```

[36]:
```
16369    7.487953
42332    6.494105
4332     6.380700
12723    6.474871
30405    9.511523
Name: loss, dtype: float64
```

5.2.2 3.2.2 Das Lasso-Verfahren

5.2.3 A-26: Es sind die Gemeinsamkeiten und der Unterschied zwischen den Verfahren Ridge-Regression und Lasso-Regression zu erläutern. Das Lasso-Verfahren soll auf den Daten für verschiedene Komplexitätsvorgaben (via Penalty-Parameter) sachgerecht angewendet und die Ergebnisse interpretiert werden. Es soll gezeigt werden, wie sich in Abhängigkeit vom Penalty-Parameter der Messfehler an der Validierungsstichprobe entwickelt und wie sich dabei die Anzahl der vom Modell nicht berücksichtigten Merkmale (also mit Koeffizient=0) in Abhängigkeit vom Penalty-Parameter verändert. (Lernziel 4.1.2) - [10 Punkte]

[37]:
```
#  Liste alpha-Werte
a_list = np.array([0.01, 0.001, 0.0001])

tic = time.time()

for alpha in a_list:
    model = Lasso(alpha=alpha,random_state=seed)
    algo = "Lasso"
    model.fit(Xs_train,y_train)
    result = mean_absolute_error(np.expm1(y_val), np.expm1(model.
 ↪predict(Xs_val)))
    print(algo + " %8s" % alpha + ", MAE:" + "%8.2f" % result + ",␣
 ↪Coefficients=0:" +  " %s" % len(model.coef_[model.coef_ == 0]))

print("time (sec):" + "%6.0f" % (time.time() - tic))
```

```
Lasso     0.01, MAE: 1601.39, Coefficients=0: 339
Lasso    0.001, MAE: 1515.13, Coefficients=0: 252
Lasso   0.0001, MAE: 1515.46, Coefficients=0: 127
time (sec):    18
```

Bei beiden Verfahren handelt es sich um regularisierte lineare Regressionsverfahren. Im Falle der Ridge-Regression wird die L2-Norm, im Falle des Lasso-Verfahrens die L1-Norm verwendet. Letztere führt dazu, dass eine Merkmalsauswahl stattfindet, indem mit steigender „Penalty“ und der damit bewirkten abnehmenden Modellkomplexität die Anzahl der Koeffizienten mit Wert 0 steigt. Beim oben durchgeführten Verfahren wird beim Penalty-Parameter Alpha=0.001 der geringste Messfehler erreicht. Dabei werden rund 250 der insgesamt 381 Merkmale im Modell nicht berücksichtigt (=0).

5.2.4 3.2.3 Das Elastic-Net-Verfahren

5.2.5 A-27: Als abschließendes lineares Modell soll das Elastic-Net-Verfahren sachgerecht angewendet werden. Wie hängt dieses mit der Ridge-Regression und LASSO-Regression zusammen? Auch hier sollen die Hyperparameter über eine mehrdimensionale "Randomized Search" bestimmt werden. Welche Vor- und Nachteile hat dieses Vorgehen gegenüber anderen Optimierungsverfahren und welche Implikationen ergeben sich für die praktische Anwendbarkeit? (Lernziel 4.1.2) - [10 Punkte]

[38]:
```
# ElasticNet: Hyperparameteroptimierung mit RandomizedSearchCV

tic = time.time()
# Parameter mit Rücksicht auf die Laufzeit angepasst
param_grid ={'alpha': [0.0002,0.0003,0.0004], 'l1_ratio': [0.05,0.10]}
EN_random_search = RandomizedSearchCV(ElasticNet(),param_grid,␣
 ↪scoring='neg_mean_absolute_error', cv=4,  n_iter=3,random_state=seed)
EN_random_search.fit(Xs_train,y_train)
print("Beste Parameter:",EN_random_search.best_params_)

# Erstelle neues Modell auf allen Folds mit den besten Parametern
EN2 = ElasticNet(**EN_random_search.best_params_)
EN2.fit(Xs_train, y_train)

print("time (sec):" + "%6.0f" % (time.time() - tic))

# Validierung MAE
result = mean_absolute_error(np.expm1(y_val), np.expm1(EN2.predict(Xs_val)))
print("\nMAE: " + "%6.2f" % result)

mae.append(result)
comb.append("ElasticNet")
```

```
Beste Parameter: {'l1_ratio': 0.1, 'alpha': 0.0003}
time (sec):   202

MAE: 1515.98
```

Das Elastic-Net-Verfahren ist eine Art gewichtete Mischung aus Ridge-Regression und Lasso, denn es verwendet für die Penalisierung sowohl die L1- als auch die L2-Norm. Es sind also (mindestens) zwei Hyperparameter zu optimieren. In der hier verwendeten „scikit-learn"-Umsetzung sind das der aus Aufgabe 23 bekannte Penalty-Parameter alpha sowie zusätzlich der Lasso-Mischungsanteil l1_ratio. Im vorliegenden Anwendungsfall ist der via „Randomized Search" gefundene optimale Lasso-Anteil mit 10% recht gering. Der Messfehler ist deutlich höher als bei den entscheidungsbaumbasierten Ensemble-Verfahren (Aufgaben 18 bis 23).

Die verbreitetsten Verfahren sind die Gittersuche und das „Randomized Search"-Verfahren. Die Gittersuche hat den Vorteil, dass im definierten Suchraum die beste Lösung gefunden wird, denn es werden alle Kombinationen aller Parameter ausprobiert. Das ist bei Verfahren wie dem LGBM-

Regressor, bei dem über zehn numerische Hyperparameter optimiert werden können, aufgrund der Kombinatorik meist unpraktikabel. Bei der „Randomized Search“ wird daher nur eine meist kleine Zufallsstichprobe der möglichen Kombination verwendet und dadurch die Anzahl der zu berechnenden Varianten stark begrenzt, im vorliegenden Anwendungsfall auf 3 aus 6. Diese Zufallsauswahl führt dazu, dass bei der „Randomized Search“ der Wertebereich jedes Merkmals oft bereits mit wenigen Interaktionen abgedeckt ist. Es ist jedoch nicht gewährleistet, dass die optimale Kombination entdeckt wird.

5.3 3.3. Vorwärtsgerichtetes Neuronales Netz mit Keras/TensorFlow

5.3.1 A-28: Als letztes Modell wird ein vorwärtsgerichtetes künstliches neuronales Netz (MLP) herangezogen, das mit Keras/Tensorflow mit zwei bis drei verborgenen Schichten mit mindestens 35.000 zu trainierenden Gewichten sowie dropout-Regularisierung zu implementieren ist. Die gewählte Startarchitektur, die Optimierungsstrategie sowie der Umgang mit „Over-Fitting“ und schwankenden Prognoseergebnissen sollen erläutert werden.

5.3.2 Die als optimal erachtete finale Netzarchitektur soll insgesamt drei Mal vollständig neu (d.h. Definition, Initialisierung & Zufallszahlen, Kompilierung, Ausführung, Evaluierung) im Notebook nacheinander angewandt und die Validierungsergebnisse angezeigt werden. Die erzielte Modellgüten und deren Varianz sollen insbesondere mit den Ergebnissen der Gradient-Boosting-Verfahren verglichen und unter Einbeziehung des Modellierungsaufwands eine vergleichende Bewertung sowie eine abschließende Modellempfehlung dokumentiert werden. (Lernziele 3.2.2, 4.1.7, 4.3.1) - [20 Punkte]

[39]:
```
def base_model():
    n_dim = n_features
    m     = 50
    model = Sequential()
    model.add(Dense(2*m, input_dim=n_dim, kernel_initializer='normal',␣
 ↪activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(m, kernel_initializer='normal', activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(1, kernel_initializer='normal'))
    # Kompiliere Modell
    model.compile(loss='mean_absolute_error', optimizer='adam')
    print(model.summary())
    return model

estimator = KerasRegressor(build_fn=base_model, epochs=500, batch_size=1000,␣
 ↪verbose=False)
%time estimator.fit(Xs_train, y_train, validation_split=0.2,␣
 ↪callbacks=[EarlyStopping(monitor='val_loss', patience=50)])

# Validierung MAE
```

```
result = mean_absolute_error(np.expm1(y_val), np.expm1(estimator.
 ↪predict(Xs_val)))
print("\nMAE: " + "%6.2f" % result)

mae.append(result)
comb.append("NeuralNet")
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 100)               38200
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 43,301
Trainable params: 43,301
Non-trainable params: 0
_________________________________________________________________
None
CPU times: user 3min 33s, sys: 24.2 s, total: 3min 57s
Wall time: 1min 47s

MAE: 1447.69
```

Bei Datensätzen mit vergleichbarem Umfang und Komplexität sind erfolgreiche MLP-Netze mit zwei verborgenen Schichten und einer Anzahl an „Neuronen“ in Höhe der Hälfte der Anzahl an Eingabeparametern bekannt, hier also rund 200 Neuronen. Da diese gewählte Startarchitektur bereits einigermaßen stabile Ergebnisse erzielte, wurde zur Sicherstellung dessen, dass das Modell über genügend Kapazität für die Komplexität der Daten verfügt, die Modellgröße in Tiefe und alternativ Breite erweitert. Das dabei aufgetretene Over-Fitting wurde zunächst mittels „Dropout“-Regularisierung und dann über eine Reduzierung der Modellgröße eingefangen.
Des Weiteren wurden verschiedene Initialisierung- und Optimierungsverfahren, „Batch-Größen“ und „Epochen“ ausprobiert. Bei Wiederholungsläufen (andere TensorFlow-Zufallszahlen) wurde zum Teil eine hohe Variabilität der Modellgüte festgestellt. Da die Laufzeiten relativ kurz waren konnte für jeden Optimierungsschritt jedes Modell mindestens zwei Mal neu trainiert werden.

[40]:
```
# 2. Vollständiger Durchlauf mit neuer Modelldefinition und TensorFlow-intern␣
 ↪neue erzeugten Zufallszahlen
def fc_model():
    n_dim = n_features
    model = Sequential()
```

```
    model.add(Dense(100, input_dim=n_dim, kernel_initializer='normal',␣
 ↪activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(50, kernel_initializer='normal', activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(1, kernel_initializer='normal'))
    # Kompiliere Modell
    model.compile(loss='mean_absolute_error', optimizer='adam')
    print(model.summary())
    return model

estimator = KerasRegressor(build_fn=fc_model, epochs=500, batch_size=1000,␣
 ↪verbose=False)

%time estimator.fit(Xs_train, y_train, validation_split=0.2,␣
 ↪callbacks=[EarlyStopping(monitor='val_loss', patience=50)])

# Validierung MAE
result = mean_absolute_error(np.expm1(y_val), np.expm1(estimator.
 ↪predict(Xs_val)))
print("\nMAE: " + "%6.2f" % result)
mae.append(result)
comb.append("NeuralNet2")
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 100)               38200
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_4 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_3 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 51
=================================================================
Total params: 43,301
Trainable params: 43,301
Non-trainable params: 0
_________________________________________________________________
None
CPU times: user 2min 51s, sys: 19.5 s, total: 3min 11s
Wall time: 1min 25s

MAE: 1449.49
```

[41]:
```
# 3. Vollständiger Durchlauf mit neuer Modelldefinition und TensorFlow-intern␣
 ↪neue erzeugten Zufallszahlen
def fc_model():
    n_dim = n_features
    model = Sequential()
    model.add(Dense(100, input_dim=n_dim, kernel_initializer='normal',␣
 ↪activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(50, kernel_initializer='normal', activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(1, kernel_initializer='normal'))
    # Kompiliere Modell
    model.compile(loss='mean_absolute_error', optimizer='adam')
    print(model.summary())
    return model

estimator = KerasRegressor(build_fn=fc_model, epochs=500, batch_size=1000,␣
 ↪verbose=False)

%time estimator.fit(Xs_train, y_train, validation_split=0.2,␣
 ↪callbacks=[EarlyStopping(monitor='val_loss', patience=50)])

# Validierung MAE
result = mean_absolute_error(np.expm1(y_val), np.expm1(estimator.
 ↪predict(Xs_val)))
print("\nMAE: " + "%6.2f" % result)
mae.append(result)
comb.append("NeuralNet3")
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 100)               38200
_________________________________________________________________
dropout_4 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_7 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_5 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 51
=================================================================
Total params: 43,301
Trainable params: 43,301
Non-trainable params: 0
_________________________________________________________________
```

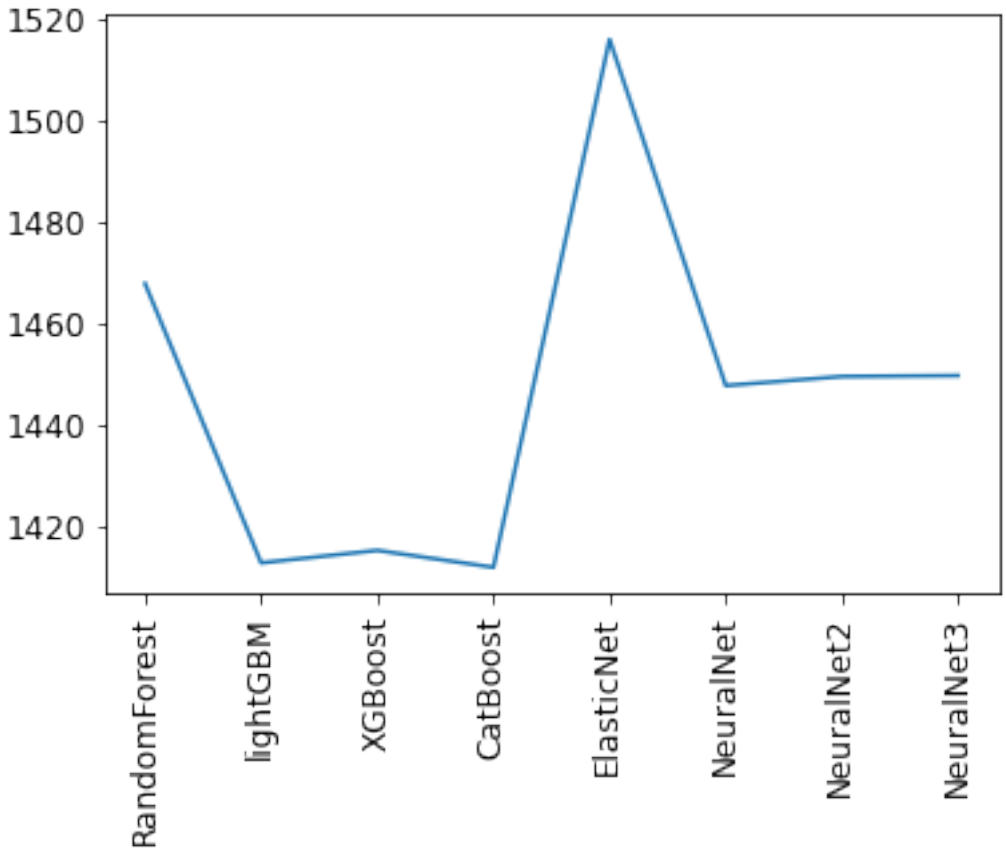
```
None
CPU times: user 3min 14s, sys: 22.3 s, total: 3min 36s
Wall time: 1min 36s

MAE: 1449.66
```

Das Neuronale Netz erzielt recht stabile Validierungsergebnisse und die damit erreichten Modellgüten sind deutlich besser als die mittels des Random-Forest-Verfahrens und insbesondere des Elastic-Net-Verahrens generierten Werte. Allerdings erzielt jedes angewendete Gradient-Boosting-Verfahren auf Basis der Validierungsdaten eine wesentlich bessere Modellgüte, im Falle von CatBoost in wenigen Sekunden ohne jeglichem Anpassungsaufwand. Die Schwankungen zwischen den drei verschiedenen Boosting-Verfahren sind kaum größer als die zwischen den drei Läufen desselben Neuronalen Netzes. Da die um einiges besseren Ergebnisse der Gradient-Boosting-Verfahren zudem mit einem deutlich kleineren bis zu gar keinem Optimierungsaufwand und entsprechend geringem Ressourcenbedarf einhergehen, erwiesen sich diese in allen Belangen als überlegen und sind daher für die vorliegende und ähnliche Fragestellungen bei ähnlichen Daten zu empfehlen. Für die vorliegenden Daten mit zahlreichen kategorialen Merkmalen trifft dies insbesondere für CatBoost zu.

6 4. Blending und Scoring

6.0.1 A-29: Abschließend ist die Prognosegüte (MAE) der Modelle graphisch zu vergleichen. Welche Schlussfolgerung ergibt sich für ein finales Ensemble der Lerner? (Lernziel 4.1.5, 4.1.7) - [5 Punkte]

[42]:
```
# Graphische Darstellung der Prognosegüte
fig, ax = plt.subplots()
plt.plot(mae)
# x-Achse mit den Modellnamen beschriften
ax.set_xticks(range(len(comb)))
ax.set_xticklabels(comb,rotation='vertical')
plt.show()
```



Die Gradient-Boosting-Methoden lightGBM und CatBoost liefern für die Aufgabenstellung bzgl. des Gütemaßes MAE die besten Ergebnisse. Ein finales Ensemble aus diesen beiden Methoden könnte durch die Verringerung der Varianz zu einem Modell mit einer verbesserten Performance führen.

6.0.2 A-30: Es ist ein finales Ensemble aus mindestens zwei Modellklassen zu konstruieren. Welche Modellgüte ergibt sich an den Validierungsdaten und wie ist diese im Vergleich zu den einzelnen Modellen einzuordnen? Welche praktischen Implikationen im Hinblick auf die Nutzbarkeit ergeben sich im Hinblick auf das Life-Cycle-Management von zusammengesetzten Lernern und wie sind diese mit praktischen Anforderungen der Wiederverwendbarkeit (also bspw. Re-Training) und Wartbarkeit des Codes zu vereinbaren? (Lernziel 4.1.7) - [8 Punkte]

[43]:
```
# Validierung MAE (Blend aus CatBoost und LightGBM)
pred_val = 0.5*(np.expm1(LGB2.predict(X_val))+np.expm1(CGB.predict(X_val)))
result = mean_absolute_error(np.expm1(y_val), pred_val)
print("MAE Validierung: " + "%6.2f" % result)

mae.append(result)
```

```
comb.append("Blend C+L")
```

```
MAE Validierung: 1404.10
```

Das Ergebnis des MAE verbessert sich durch das Blending von CatBoost und LightGBM um etwa 0,5%. Man erwartet grundsätzlich keine große Verbesserungen, weil die Modelle für sich genommen ja schon sehr gut sind. Dennoch ist dies eine gute praktische Möglichkeit, durch ein Ensemble aus mehreren sehr guten, möglichst unterschiedlichen Modellen ein besser generalisierendes Gesamtmodell mit geringerer Anfälligkeit für Over-Fitting und einer etwas besseren finalen Performance zu erzeugen.

Mögliche Antworten im Hinblick auf die Anwendbarkeit:

- Oftmals sind gestackte Modelle deswegen problematisch, weil bei einer Änderung die ganze Pipeline neu trainiert werden muss, es eignet sich also nur, wenn Updates selten sind oder es eine klare Update-Strategie gibt, die Re-Training nur einzelner Komponenten erlaubt. Dies ist aber einzeln zu überprüfen, was in jedem Fall zusätzlichen Aufwand bedeutet.
- Die Reproduzierbarkeit von gestackten Modellen erfordert es, streng die gegebene Trainingsreihenfolge einzuhalten und ggf. jedes Teilmodell separat und unabhängig voneinander deterministisch zu seeden. Falls sogar die Input-Transformationen wiederholbar sein müssen, muss noch mehr Sorgfalt aufgewendet werden, was den pflegeaufwand erheblich erhöht.
- Darüber hinaus können mehrfach gestackte Modelle *sehr* komplex werden. Diese Komplexität erlaubt einerseits eine hohe Prognosegüte, bringt aber Nachteile in Form von eingeschränkter Erklärbarkeit mit sich. Auch ist aufgrund dessen das Debugging, sprich die Suche nach Grenzfällen und/oder pathologischen Prognosen sehr schwierig. Dabei ist wichtig zu bedenken, dass somit immer die gesamte Pipeline vom Sampling über die Merkmalsgenerierung und (ggf.) Dimensionreduktion über Training und Stacking bis hin zum Blending in eins betrachtet werden muss. Einzelne Änderungen müssen konsequenterweise immer einen Check der gesamtperformance zur Folge haben, was beim Tuning zu Performanceproblemen und schlicht ineffizienter Zeit- und Energieverwendung führen kann.
- Nicht zuletzt gilt es auch, den Nutzen ökonomisch abzuwägen. Für eine Prognoseverbesserung im Promille-Bereich ist es oftmals nicht angemessen, Trainings- und Rechenzeit von der Größenordnung eines stabilen, simpleren Modells erneut aufzuwenden. Dies muss aber für das Management innerhalb gewisser Rahmenerwägungen transparent gemacht werden (können).
- Geblendete Modelle haben andererseits den Vorteil, dass bei einer Verbesserung des Teilmodells i.d.R. auch das resultierende Modell nicht schlechter werden sollte. Der Nachweis muss aber dann auf dem gleichen Validierungsdatensatz erfolgen. In der Praxis kann dies dazu führen, dass man zwar Daten zum Updaten hat, aber mit “alten” Validierungsdaten Vorlieb nehmen muss. Da aber die Anforderung besteht, prinzipiell jede aktive Tarifgeneration reproduktiv nachberechnen zu können, kann sich hier eine Diskrepanz ergeben, sowohl was das Testen der kompletten Pipeline angeht als auch die Aktualität der Validierung. Er ergibt sich also ein Redundanzproblem, das ggf. nur für neue Tarifgenerationen mit neu auflegbaren Validierungsdaten befriedigend gelöst werden kann.
- Daraus ergibt sich die grundsätzliche Kritik, dass geblendete Modelle in der Praxis bspw. für Stresstests nur bedingt geeignet sind und sich nur schwer (d.h. über zusätzliche Annahmen) Konfidenzintervalle bestimmen lassen. Während dies für bspw. einen Kreditscore nicht

weiter problematisch ist, stellt es für Reservierungs- und Tarifierungszwecke eine ungelöste Problemstellung dar, da nicht bekannt ist, inwiefern der Regulator die aktuelle Praxis zugunsten von besseren Prognosen aber (ggf.) schlechterer Stabilität abzuändern gedenkt. Es gibt jedenfalls keinerlei Einblicke dazu, welche Tendenz hier besteht. Es scheint daher vernünftig, die Limitationen der jeweiligen Modellauswahl transparent zu dokumentieren.

6.0.3 A-31: Das finale Ensemble aus Aufgabe 30 soll auf die Testdaten angewendet werden. Welche Modellgüte ergibt sich? Worauf können Unterschiede zur in Aufgabe 30 ermittelten Modellgüte zurückzuführen sein? Des weiteren soll überprüft werden, ob die an den Testdaten durchgeführte Schadenaufwandsprognose insgesamt erwartungstreu ist und erklärt werden, worauf der beobachtete Unterschied zurückzufügen ist und welche Konsequenzen das für eine Anwendung in der Tarifierung hätte. (Lernziel 4.1.7) - [8 Punkte]

[44]:
```
# Vorhersage mittels blending (CatBoost und LightGBM)
predictions = 0.5*(np.expm1(LGB2.predict(df_test_dummies))+np.expm1(CGB.
 ↪predict(df_test_dummies)))

# DataFrame erzeugen und erste Testdatensätze anzeigen (Spezialfall: hier loss␣
 ↪bekannt)
df_test_pred = pd.DataFrame({'claim number':ID,'loss':df_test.loss, 'predicted':
 ↪predictions})
df_test_pred.head()
```

[44]:

	claim number	loss	predicted
0	4	2974.452295	1834.396487
1	6	1453.687463	2338.573051
2	8	9183.368240	6655.236679
3	12	2695.777301	5529.857045
4	13	1367.790353	1566.759871

[45]:
```
# Finalen Messfehler an den bisher umbesehenen Testdaten ermitteln
result = mean_absolute_error(df_test.loss, predictions)
print("MAE Testdaten: " + "%6.2f" % result)
```

```
MAE Testdaten: 1421.28
```

Bei der Betrachtung der Testdaten zeigt sich, dass die Modellgüte um 1,2% schlechter als bei den Validierungsdaten ist. Das könnte daran liegen, dass im Falle von lightGBM die Modellauswahl auf Basis der Valdierungsdaten durchgeführt wurde und dadurch die Validierungsdaten nicht mehr als gänzlich neutral bei der Ermittlung der Modellgüte angesehen werden können und somit die hier ermittelte Modellgüte an unbesehenen Testdaten als realistischer eingeschätzt werden kann. Eine andere Ursache könnte in einer unterschiedlichen Zusammensetzung der Testdaten liegen. So könnte beispielsweise ein etwas höherer Anteil an schlechter prognostizierbaren Schäden den an den Testdaten ermittelten MAE erhöht haben.

[46]:
```
# Erwartungstreue an den Testdaten überprüfen (und mit Validierungsdatem␣
 ↪vergleichen)
```

```
print("Mittelwert der Schadenhöhen:")

print("\nTestdaten, Prognose: " + "%6.2f" % predictions.mean())
print("Testdaten, Messwert: " + "%6.2f" % df_test.loss.mean())

print("\nValidierungsdaten, Prognose: " + "%6.2f" % pred_val.mean())
print("Validierungsdaten, Messwert: " + "%6.2f" % np.expm1(y_val).mean())
```

```
Mittelwert der Schadenhöhen:

Testdaten, Prognose: 3012.50
Testdaten, Messwert: 3529.49

Validierungsdaten, Prognose: 3034.51
Validierungsdaten, Messwert: 3523.10
```

Durch das gewählte Gütemaß mittlerer **absoluter** Fehler (MAE) wird auf Ebene der einzelnen Schäden zwar eine gute Prognose erzielt, insgesamt wird jedoch der Schadenaufwand um mehr als 10% unterschätzt, da Großschäden einen vergleichsweise geringen Einfluss auf den MAE haben (im Gegensatz zum MSE, siehe Aufgabe 17). Im Falle einer Tarifierung würde auf Basis der vorliegenden Modellierung der gesamte Schadenaufwand deutlich unterschätzt werden.