Aufgabe_ADS_Completion_BlockA_Solution

May 15, 2025

1 Aufgabe 1 NER

1.1 EDA der gelabelten Mail-Daten

Zuerst lesen wir die Daten ein und erstellen eine kleine explorative Datenanalyse zum Datensatz.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

[2]:
```
# Erst Blick auf die Daten
data = pd.read_csv("labeled_mails_teil_a_1.txt", encoding="utf8")
print(data.tail(10))
```

```
          Mail          Word           Tag
94761  mail982         Grüße             O
94762  mail982             ,             O
94763  mail982     Anneliese        B-NAME
94764  mail982       Lehmann        I-NAME
94765  mail982  Geburtsdatum             O
94766  mail982            12  B-BIRTH_DATE
94767  mail982             .  I-BIRTH_DATE
94768  mail982            07  I-BIRTH_DATE
94769  mail982             .  I-BIRTH_DATE
94770  mail982          1958  I-BIRTH_DATE
```

[3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 94771 entries, 0 to 94770
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Mail    94771 non-null  object
 1   Word    94771 non-null  object
 2   Tag     94771 non-null  object
```

```
dtypes: object(3)
memory usage: 2.2+ MB
```

[4]:
```
data["Tag"].value_counts()
```

[4]:
```
Tag
O                     75865
I-CLAIM_AMOUNT         3395
I-BIRTH_DATE           2600
I-CLAIM_DATE           2447
I-NAME                 1547
B-NAME                 1434
B-CLAIM_LOCATION       1067
B-CLAIM_ITEM            985
B-INSURANCE_NUMBER      982
B-CLAIM_AMOUNT          971
B-CLAIM_TYPE            836
B-CLAIM_DATE            830
B-BIRTH_DATE            822
B-CLAIM_CAUSE           689
I-CLAIM_CAUSE           218
I-CLAIM_TYPE             52
I-CLAIM_ITEM             31
Name: count, dtype: int64
```

Es scheint sich um einen Datensatz im conll-Format zu handeln. Das bedeutet, dass jede Zeile ein Token (also ein einzelnes Wort oder Satzzeichen) repräsentiert, und dieses Token ist typischerweise mit zusätzlichen Annotationen versehen – in diesem Fall durch die Spalten Mail, Word und Tag.

Es scheinen 94771 Wörter im Datensatz enthalten zu sein. Diese sind auf 982 Mails verteilt. Und es sind 17 verschiedene Tags im Datensatz, wobei es eigentlich nur um die elf aus der Aufgabenstellung handelt, da "B-" und "I-" jeweils für gleiche Tags vorkommen können.

Die Anzahl unterschiedlicher Wörter ergibt sich als:

[5]:
```
print("Unique words in corpus:", data['Word'].nunique())
```

```
Unique words in corpus: 4407
```

1.1.1 Fazit EDA

Wir können versuchen auf diesem Datensatz ein BI-LSTM-Modell zu trainieren. Die Datenmenge scheint allerdings etwas klein zu sein und die Verteilung der Tags zwischen `O` (also Wort nicht relevant) und den interessanten Tags scheint darauf zu deuten, dass es schwierig werden könnte, alle relevanten Daten aus den Emails zu extrahieren.

1.2 Implementierung BI-LST-Modell

Um ein Modell zu fitten, müssen wir die Daten erst einmal in das richtige Format bringen, einen Wort2Index und einen Tag2Index aufbauen, und anschließend ein Modell definieren und fitten.

1.2.1 Daten formatieren

[6]:
```
# Mails in eine Liste packen
def mail_integrate(data):
  agg_func = lambda s: [(w,t) for w, t in zip(s["Word"].values.tolist(),␣
 ↪s["Tag"].values.tolist())]
  return data.groupby('Mail').apply(agg_func).tolist()
mails=mail_integrate(data)
len_mails =[len(mail) for mail in mails]
print(f"""Die längste Mail enthält {max(len_mails)} Wörter.
Die kürzeste Mail enthält {min(len_mails)} Wörter.
Durchschnittliche Länge der Mails: {np.mean(len_mails)} Wörter.
      """)
```

```
Die längste Mail enthält 158 Wörter.
Die kürzeste Mail enthält 40 Wörter.
Durchschnittliche Länge der Mails: 96.40996948118006 Wörter.
```

Schauen wir uns einmal zwei Beispielsmails an:

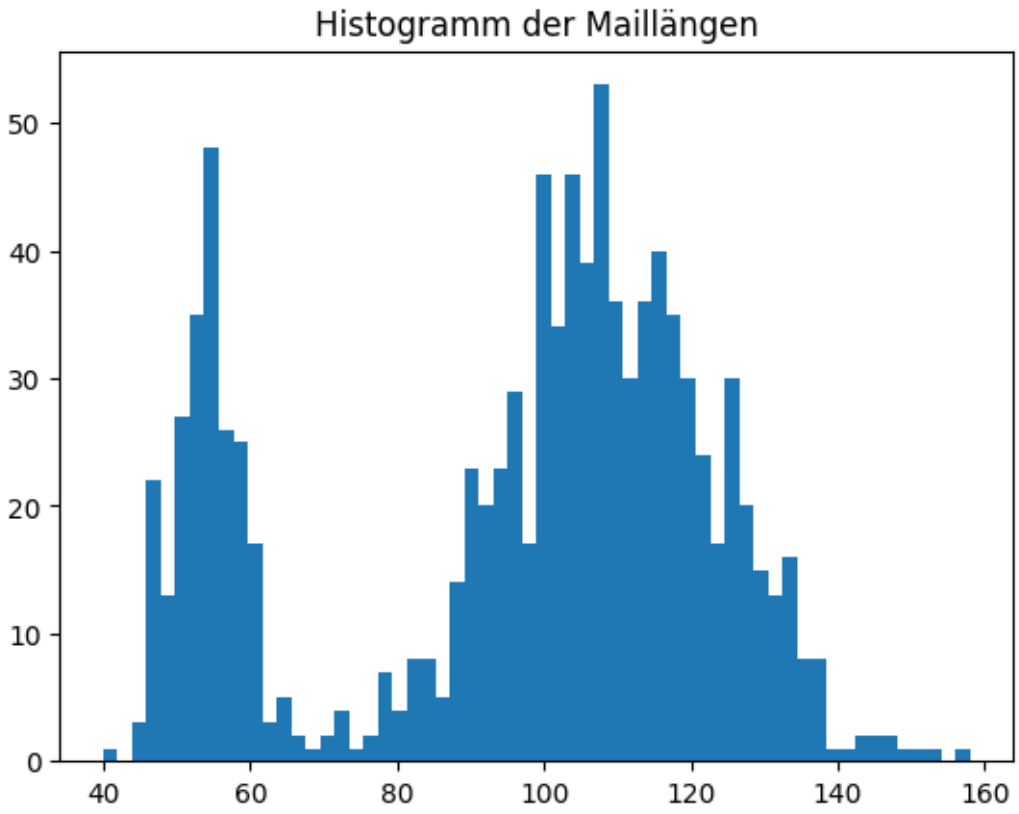
[17]:
```
for mail in mails[0:2]:
    for tup in mail:
        print(tup[0], end=" ")
    print()
```

```
Sehr geehrte Damen und Herren , hiermit möchte ich einen Schadensfall melden ,
der sich am 8 . Januar 2025 in meinem Haus ereignet hat . Die Details des
Schadens sind wie folgt Name Fredo Walter Geburtsdatum 19 . April 1990
Versicherungsnummer 3342331444 Schadensart Einbruch Schadensort Haus
Schadensursache Manipulierter Schließzylinder Beschädigtes Objekt Laptop
Geschätzter Schaden 807 , 94 EUR Ich bitte um eine zügige Bearbeitung meines
Anliegens und stehe für Rückfragen jederzeit zur Verfügung . Vielen Dank im
Voraus für Ihre Unterstützung . Mit freundlichen Grüßen , Fredo Walter
Hallo liebes Schadenservice Team , ich hoffe , es geht euch gut ! Ich muss euch
leider von einem kleinen Missgeschick berichten . Am 19 . April 2024 hat es bei
mir im Garten ganz schön gewittert , und ein Blitz hat direkt eingeschlagen .
Dabei hat es meinen Computer erwischt total kaputt , wirklich ärgerlich ! Der
geschätzte Schaden beläuft sich auf etwa 486 , 88 EUR . Meine
Versicherungsnummer ist die 1939042955 , falls ihr da nachschauen wollt . Ich
wäre super dankbar , wenn ihr mir helfen könntet , die Sache zu klären . Vielen
Dank schon mal im Voraus ! Liebe Grüße , Sigfried Beckmann Geburtsdatum 16 . 08
. 1950
```

Und ein Histogram über die Längen der einzelnen Mails:

[20]:
```
# Histogramm der Maillängen
plt.hist(len_mails, bins=60)
plt.title("Histogramm der Maillängen")
```

[20]: Text(0.5, 1.0, 'Histogramm der Maillängen')



[5]:
```
# Tags und Wörter extrahieren
tags = list(set(data["Tag"].values))
words = list(set(data["Word"].values))
words.append("ENDPAD")
```

[6]:
```
# Vokabular und Index erstellen
word2idx = {w: i + 1 for i, w in enumerate(words)}
tag2idx = {t: i for i, t in enumerate(tags)}
```

jetzt setzen wir noch ein padding, wobei wir die Padding-Größe auf 100 Zeichen setzen, das liegt (siehe oben) nahe am Mittelwert.

[7]:
```
max_len = 100

X = [[word2idx[w[0]] for w in m] for m in mails]
X = pad_sequences(maxlen=max_len, sequences=X, padding="post",␣
 ↪value=len(words)-1)

y = [[tag2idx[w[1]] for w in m] for m in mails]
```

```
y = pad_sequences(maxlen=max_len, sequences=y, padding="post",␣
 ↪value=tag2idx["O"])

print(f"Groesse der Eingabedaten: {X.shape}")
print(f"Laenge der Labels: {len(y)}")
```

```
Groesse der Eingabedaten: (983, 100)
Laenge der Labels: 983
```

Jetzt teilen wir die Daten noch in Trainings- und Testdaten auf:

[8]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=1)
```

Prinzipiell sind wir jetzt in der Lage ein Modell zu definieren und zu trainieren. Die Daten scheinen allerdings etwas zu klein (geringe Anzahl an Wörtern) zu sein. Ebenso wird die Verteilung der Tags, also der Klassen in der Klassifikation, sehr unausgewogen zu sein. Der "unwichtige" Tag "O" kommt deutlich häufiger vor, als die relevanteren Tags (bspw. "CLAIM-CAUSE"). Damit sollten die Erwartungen für die Ergebnisse nicht zu hoch gesteckt werden.

Ebenso wurden keine Stopwords entfernt.

1.3 Modell definieren und trainieren

[9]:
```
from tensorflow.keras import Model, Input
from tensorflow.keras.layers import LSTM, Embedding, Dense
from tensorflow.keras.layers import InputLayer, TimeDistributed,␣
 ↪SpatialDropout1D, Bidirectional
from tensorflow import keras
```

[10]:
```
model = keras.Sequential()
model.add(InputLayer((max_len)))
model.add(Embedding(input_dim=len(words), output_dim=max_len,␣
 ↪input_length=max_len))
model.add(SpatialDropout1D(0.1))
model.add( Bidirectional(LSTM(units=100, return_sequences=True,␣
 ↪recurrent_dropout=0.1)))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 100)          440800

 spatial_dropout1d (SpatialD  (None, 100, 100)         0
 ropout1D)
```

```
 bidirectional (Bidirectiona  (None, 100, 200)         160800
 l)

=================================================================
Total params: 601,600
Trainable params: 601,600
Non-trainable params: 0
_________________________________________________________________
```

[11]:
```
# Modell zusammenstellen
model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
```

[12]:
```
# Training des Modells

chkpt = ModelCheckpoint("model_weights.h5", monitor='val_loss',verbose=1,␣
 ↪save_best_only=True, save_weights_only=True, mode='min')

early_stopping = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=1,␣
 ↪verbose=0, mode='max', baseline=None, restore_best_weights=False)

history = model.fit(
    x=x_train,
    y=y_train,
    validation_data=(x_test,y_test),
    batch_size=32,
    epochs=3,
    verbose=1

)
```

```
Epoch 1/3
25/25 [==============================] - 6s 117ms/step - loss: 3.2330 -
accuracy: 0.7152 - val_loss: 1.8046 - val_accuracy: 0.8127
Epoch 2/3
25/25 [==============================] - 3s 132ms/step - loss: 1.3028 -
accuracy: 0.8127 - val_loss: 0.7990 - val_accuracy: 0.8138
Epoch 3/3
25/25 [==============================] - 5s 193ms/step - loss: 0.7128 -
accuracy: 0.8142 - val_loss: 0.5629 - val_accuracy: 0.8194
```

[13]:
```
print("Evaluate on test data")
results = model.evaluate(x_test, y_test, batch_size=128)
print("test loss: {} ".format(results[0]))
print("test accuracy: {} ".format(results[1]))
```

```
Evaluate on test data
```

```
2/2 [==============================] - 0s 50ms/step - loss: 0.5629 - accuracy:
0.8194
test loss: 0.5628989934921265
test accuracy: 0.8194416165351868
```

[17]:
```python
from sklearn.metrics import classification_report, f1_score
# Vorhersagen holen
y_pred = model.predict(x_test)
y_pred_labels = np.argmax(y_pred, axis=-1)
y_true_labels = np.squeeze(y_test)

# Padding ignorieren
mask = (y_true_labels != 0)
y_pred_filtered = y_pred_labels[mask]
y_true_filtered = y_true_labels[mask]

# F1-Score berechnen
f1 = f1_score(y_true_filtered, y_pred_filtered, average='macro')
print("F1 Score (macro):", f1)
```

```
7/7 [==============================] - 0s 24ms/step
F1 Score (macro): 0.06082099335830319
```

1.4 Fazit Training

Die Accuracy ist auf den Testdaten, wie erwartet nicht gerade berauschend und der f1-score ist so schlecht, dass man eigentlich davon ausgehen muss, dass das Modell nur rät.

2 Aufgabe 2 Vergleich

Zum Vergleich wählen wir die NER-Funktionalitäten von `spacy`. Für diese benötigen wir das den zweiten Datensatz mit gelabelten Mails. Dieser befindet sich schon im von spacy benötigten Format.

[26]:
```python
import spacy
from spacy.training import Example
from spacy import displacy
from spacy.scorer import Scorer
import ast
```

[27]:
```python
# Datensatz laden
with open('labeled_mails_teil_a_2.txt', 'r', encoding="UTF-8") as f:
  data_spacy = ast.literal_eval(f.read())
```

[28]:
```python
# Labels extrahieren
labels = set()
for mail in data_spacy:
    for entity in mail[1]["entities"]:
        labels.add(entity[2])
```

```
labels = list(labels)
print(labels)
```

```
['CLAIM_DATE', 'CLAIM_TYPE', 'CLAIM_CAUSE', 'BIRTH_DATE', 'CLAIM_ITEM',
'CLAIM_AMOUNT', 'INSURANCE_NUMBER', 'CLAIM_LOCATION', 'NAME']
```

[29]:
```
# auch hier wieder die Daten aufteilen
# 80% Training, 20% Test
train_data, test_data = train_test_split(data_spacy, test_size=0.2,␣
 ↪random_state=42)

print(f"Train-Daten: {len(train_data)}")
print(f"Test-Daten: {len(test_data)}")
```

```
Train-Daten: 800
Test-Daten: 200
```

Für die NER verwenden wir ein vortrainiertes Modell, das `de_core_news_md`Modell.

[30]:
```
# Vortrainiertes Modell laden
nlp_pretrained = spacy.load("de_core_news_md")
ner_pretrained = nlp_pretrained.get_pipe("ner")  # Bestehendes NER verwenden
```

[31]:
```
# Entitätslabels registrieren
for label in labels:
    ner_pretrained.add_label(label)
```

[32]:
```
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
optimizer = nlp_pretrained.create_optimizer();
losses = {}
for epoch in range(2):  # Trainingsdurchläufe
  for text, annotations in train_data:
    example = Example.from_dict(nlp_pretrained.make_doc(text), annotations)
    _ = nlp_pretrained.update([example], drop=0.1, losses=losses);
```

[33]:
```
def evaluate_ner(nlp, test_data):
    scorer = Scorer()  # Scoring-Objekt von Spacy
    examples = []

    for text, annotations in test_data:
        doc = nlp(text)  # Modell auf Test-Daten anwenden
        example = Example.from_dict(doc, annotations)  # Test-Beispiel erstellen
        examples.append(example)

    scores = scorer.score(examples);  # Berechne Precision, Recall & F1
    return scores

# Test-Daten evaluieren
```

```
scores_pretrained = evaluate_ner(nlp_pretrained, test_data)
```

[34]:
```
# Ausgabe der Scores

print("\nPretrained Modell (nlp_pretrained):")
print(f"Precision: {scores_pretrained['ents_p']:.3f}")
print(f"Recall: {scores_pretrained['ents_r']:.3f}")
print(f"F1-Score: {scores_pretrained['ents_f']:.3f}")
```

```
Pretrained Modell (nlp_pretrained):
Precision: 0.906
Recall: 0.972
F1-Score: 0.937
```

2.1 Fazit Vortrainiertes Modell

Das vortrainierte Modell von `spacy` liefert deutlich bessere Ergebnisse als das selbst entwickelte.

3 Aufgabe 3 Kostenvorhersage

Jetzt laden wir die Daten für die Kostenvorhersage und modellieren ein Random-Forest Modell. Da in der Aufgabenstellung keinerlei Vorgaben für das Modell gegeben sind, ist hier eine freie Wahl erlaubt.

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

[2]:
```
df = pd.read_csv('historic_claims_data_teil_a.csv')
df.head()
```

[2]:

	name	email	birth_date	insurance_number	claim_type	claim_date	claim_location	claim_cause	claim_item	expected_claim_amount	actual_claim_amount
0	Felicitas Dowerg	felicitas.dowerg@aol.de	1994-09-23	2181241943	Gerätedefekt	2021-11-24	Wohnzimmer	Kurzschluss	Monitor	377.69	399.68
1	Fredo Walter	fredo.walter@aol.de	1990-04-25	3342331444	Einbruch	2025-01-08	Haus	Eingeschlagenes Fenster			
2	Sigfried Beckmann	sigfried.beckmann@aol.de	1950-08-22	1939042955	Gerätedefekt	2024-04-19	Keller	Kurzschluss			
3	Dorothea Bauer	dorothea.bauer@hotmail.de	1957-10-18	2801823908	Gerätedefekt	2023-10-27	Keller	Wasserschaden			
4	Damaris Holsten	damaris.holsten@yahoo.de	2003-06-03	4460967357	Diebstahl	2023-08-06	Straße	Aufgebrochenes Schloss			

```
1         Bargeld               2705.62              8921.75
2  Heimkinoanlage               1234.07              1288.72
3       Fernseher                684.54               646.92
4         Fahrrad               1102.54              1040.26
```

3.1 Kleine EDA

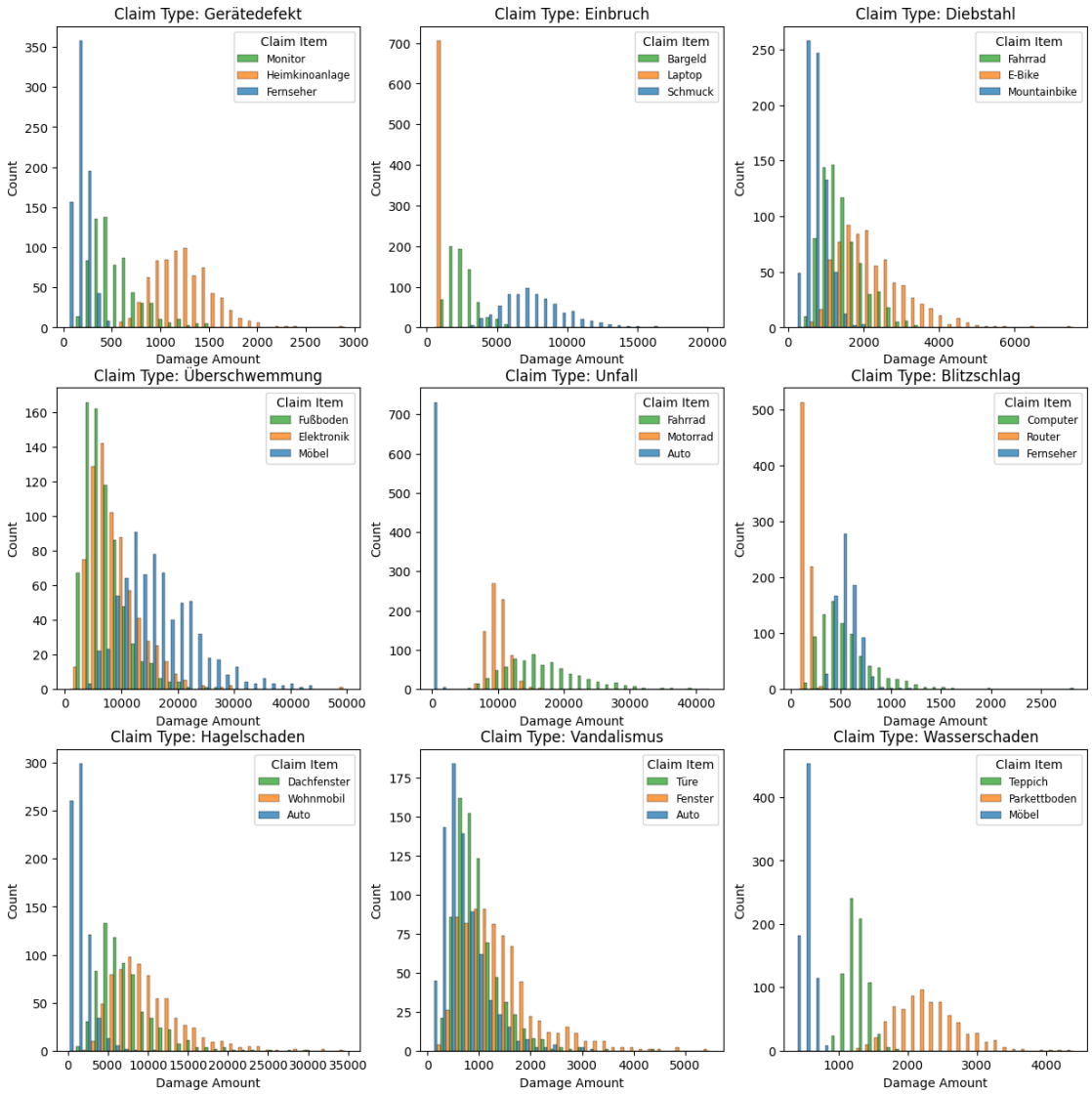
[3]:
```
print(df.info())
print(df.describe())
print(df.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 11 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   name                   20000 non-null  object
 1   email                  20000 non-null  object
 2   birth_date             20000 non-null  object
 3   insurance_number       20000 non-null  int64
 4   claim_type             20000 non-null  object
 5   claim_date             20000 non-null  object
 6   claim_location         20000 non-null  object
 7   claim_cause            20000 non-null  object
 8   claim_item             20000 non-null  object
 9   expected_claim_amount  20000 non-null  float64
 10  actual_claim_amount    20000 non-null  float64
dtypes: float64(2), int64(1), object(8)
memory usage: 1.7+ MB
None
       insurance_number  expected_claim_amount  actual_claim_amount
count      2.000000e+04           20000.000000         20000.000000
mean       5.521359e+09            3821.592291          3810.137099
std        2.602186e+09            5560.732726          5354.077535
min        1.000368e+09              34.300000            70.730000
25%        3.257666e+09             632.470000           651.527500
50%        5.541998e+09            1259.130000          1262.460000
75%        7.790060e+09            4868.692500          5102.065000
max        9.999393e+09           61942.350000         49791.350000
name                    0
email                   0
birth_date              0
insurance_number        0
claim_type              0
claim_date              0
claim_location          0
claim_cause             0
claim_item              0
```

```
expected_claim_amount    0
actual_claim_amount      0
dtype: int64
```

Ein zusätzlicher Blick auf die Schadensverteilung:

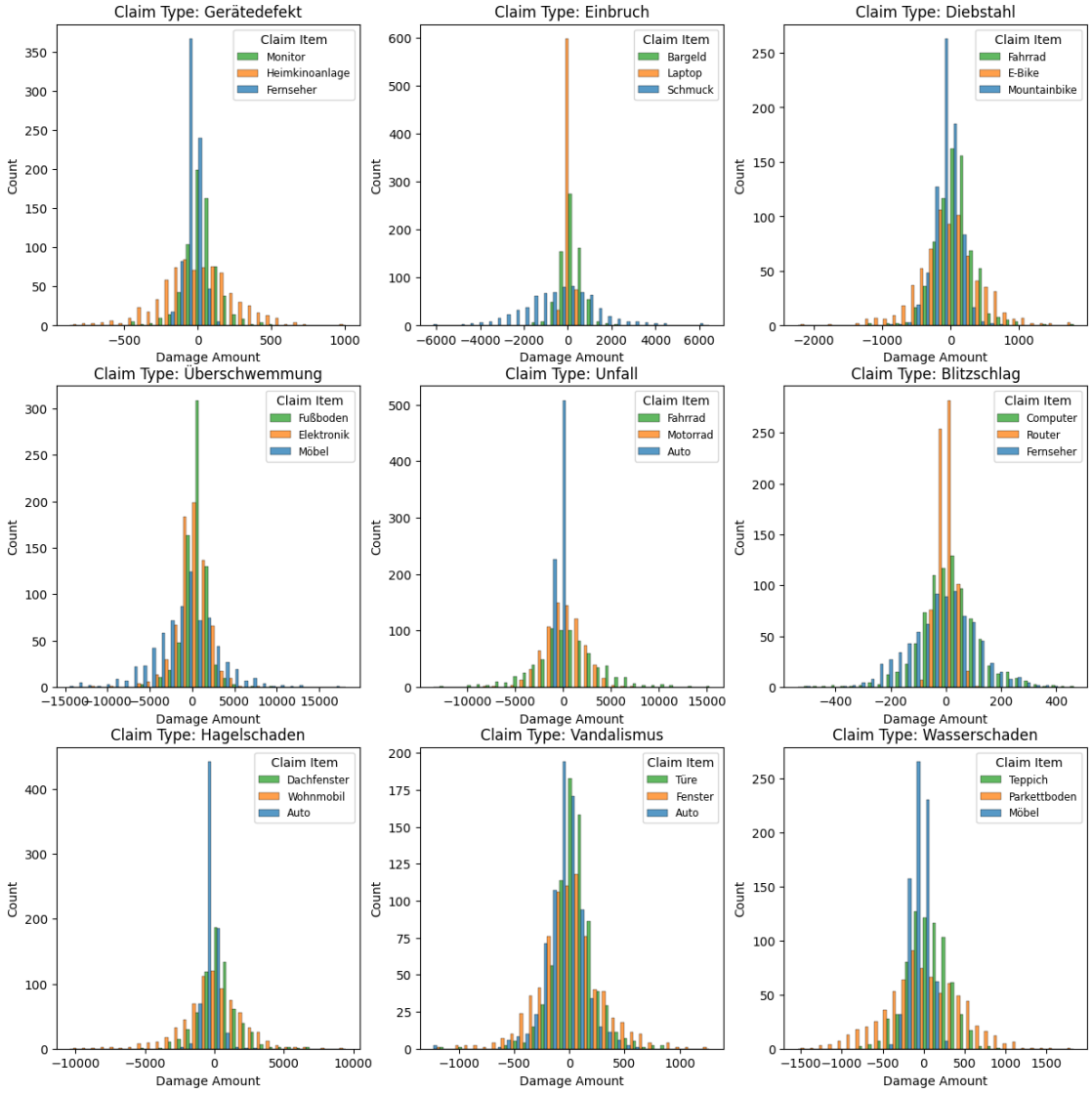
[20]:
```
# Alle claim_types durchlaufen
claim_types = df["claim_type"].unique()
fig, axes = plt.subplots(3, 3, figsize=(15, 15))
axes = axes.flatten()
for i, ctype in enumerate(claim_types):
    ax = axes[i]
    subset = df[df["claim_type"] == ctype]

    # Plot
    sns.histplot(data=subset, x="actual_claim_amount", ax=ax, hue="claim_item",␣
 ↪bins=30, multiple="dodge", legend=True)
    ax.set_title(f"Claim Type: {ctype}")
    ax.set_xlabel("Damage Amount")
    ax.set_ylabel("Count")
    ax.legend(labels=subset["claim_item"].unique(), title="Claim Item",␣
 ↪fontsize="small")


plt.show()
```



Und ebenso interessant, wie weichen gemeldeter und dann wirklicher Schadensbetrag von einander ab:

[22]:
```
# Alle claim_types durchlaufen
df['prediction_error'] = df['actual_claim_amount'] - df['expected_claim_amount']
claim_types = df["claim_type"].unique()
fig, axes = plt.subplots(3, 3, figsize=(15, 15))
axes = axes.flatten()
for i, ctype in enumerate(claim_types):
    ax = axes[i]
    subset = df[df["claim_type"] == ctype]

    # Plot
```

```
    sns.histplot(data=subset, x="prediction_error", ax=ax, hue="claim_item",␣
 ↪bins=30, multiple="dodge", legend=True)
    ax.set_title(f"Claim Type: {ctype}")
    ax.set_xlabel("Damage Amount")
    ax.set_ylabel("Count")
    ax.legend(labels=subset["claim_item"].unique(), title="Claim Item",␣
 ↪fontsize="small")


plt.show()
```



Es sieht so aus, als ob die Schäden einer Log-Normalverteilung nach `claim_types` und `claim_items` folgen. Der Unterschied zwischen vom Kunden gemeldeten Schadenswert `expected_claim_amount`

und dem dann wirklich eingetretenen Schaden `actual_claim_amount` hingegen, scheint einer Normalverteilung zu folgen.

[ ]:
```
# Feature Engineering
# Geburtsjahr aus birth_date extrahieren
df['birth_year'] = pd.to_datetime(df['birth_date']).dt.year

# Schadenjahr aus claim_date extrahieren
df['claim_year'] = pd.to_datetime(df['claim_date']).dt.year
```

[39]:
```
# Zielvariable
y = df['actual_claim_amount']
```

[40]:
```
# Drop nicht benötigte oder identifizierende Spalten
df_features = df.drop(columns=[
    'name', 'email', 'birth_date', 'claim_date',
    'insurance_number', 'actual_claim_amount'
])
```

[54]:
```
# One-Hot-Encoding für kategoriale Variablen
X = pd.get_dummies(df_features)
```

[42]:
```
# Schritt 5: Train-Test-Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

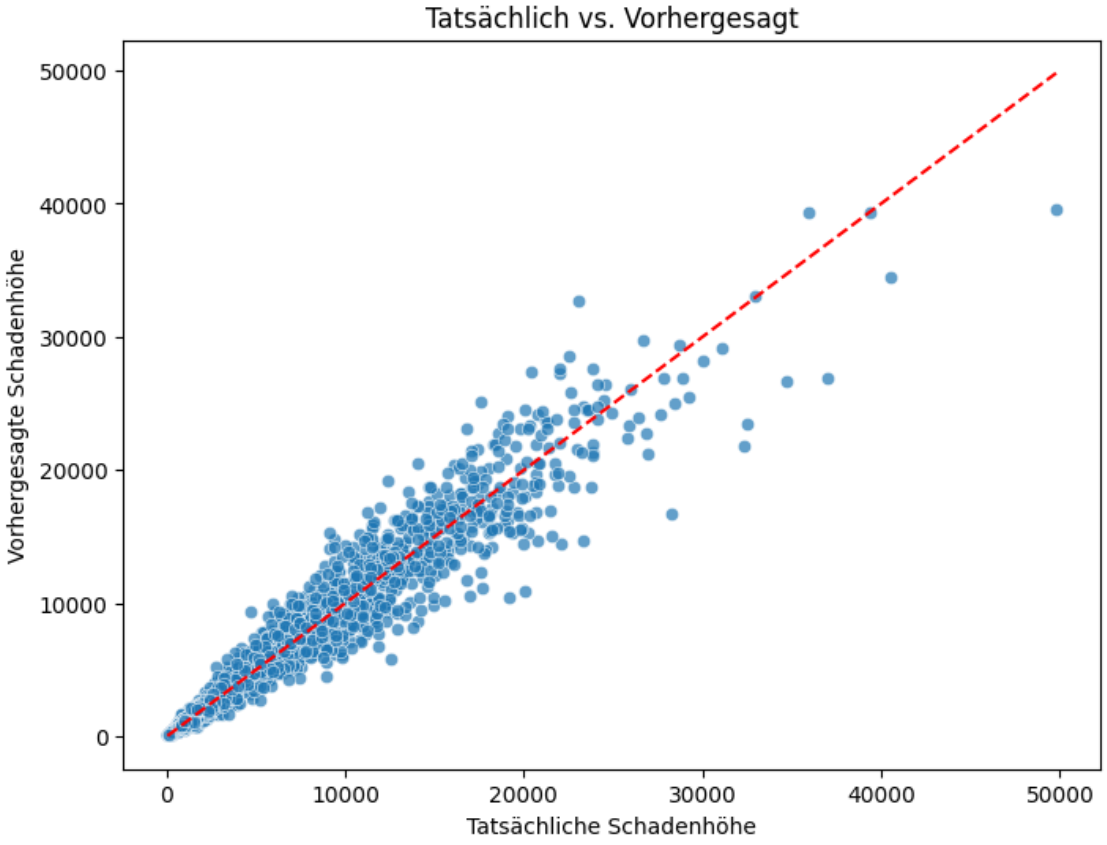
[43]:
```
# Schritt 6: Modelltraining (Random Forest)
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

[43]:
```
RandomForestRegressor(random_state=42)
```

[ ]:
```
# Vorhersage und Bewertung
y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R² Score: {r2:.3f}")
```

```
Mean Absolute Error (MAE): 562.51
Root Mean Squared Error (RMSE): 1184.20
R² Score: 0.952
```

```
[ ]: # Scatterplot tatsächliche vs. vorhergesagte Werte
plt.figure(figsize=(8,6))
sns.scatterplot(x=y_test, y=y_pred, alpha=0.7)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],␣
  ↪color='red', linestyle='--')
plt.xlabel("Tatsächliche Schadenhöhe")
plt.ylabel("Vorhergesagte Schadenhöhe")
plt.title("Tatsächlich vs. Vorhergesagt")
plt.show()
```



3.2 Fazit Kostenvorhersage

Das RandomForest Modell liefert brauchbare Vorhersagen auf den vorgegebenen Schadendaten.

4 Aufgabe 4 Pipeline erstellen

Jetzt erstellen wir eine Pipeline (eine Funktion) die eine unstrukturierte Email erhält und dann mittels unserem `spacy`-Modell die Named-Entities herausliest und diese wiederum in das Kostenvorhersagemodell übergibt.

[73]:
```
#json-Datei mit Emails einlesen
import json
with open('customer_emails_teil_a.json', 'r', encoding='utf-8') as f:
    emails = json.load(f)
```

[74]:
```
# Funktion zur Extraktion von benannten Entitäten
def extract_named_entities(email):
    entities = []
    # Verarbeite E-Mail mit Spacy
    doc = nlp_pretrained(email)

    # Extrahiere benannte Entitäten (wie Personen, Organisationen, Orte)
    for ent in doc.ents:
        entities.append((ent.text, ent.label_))

    return entities
```

[75]:
```
# Funktion zum Mappen der Entitäten auf das Feature-Template
def map_entities_to_features(entities, features_template):
    # Initialisiere eine Zeile mit Nullen für das Feature-Template
    feature_row = {col: 0 for col in features_template.columns}

    # Mappings für Entitäten
    for value, entity in entities:
        if entity == 'NAME':
            pass # Namen werden nicht gemappt
        elif entity == 'EMAIL':
            pass # E-Mail-Adressen werden nicht gemappt
        elif entity == 'CLAIM_CAUSE':
            # Mappen der Schadensursache zu den jeweiligen claim_cause_* Spalten
            feature_row[f'claim_cause_{value}'] = 1
        elif entity == 'CLAIM_LOCATION':
            # Mappen der Schadenorte zu den jeweiligen claim_location_* Spalten
            feature_row[f'claim_location_{value}'] = 1
        elif entity == 'CLAIM_ITEM':
            # Mappen der Schadenobjekte zu den jeweiligen claim_item_* Spalten
            feature_row[f'claim_item_{value}'] = 1
        elif entity == 'CLAIM_AMOUNT':
            # Umwandeln des Betrags in eine Zahl
            claim_amount = float(value.replace('EUR', '').replace(',', '.').
 ↪strip())
            feature_row['expected_claim_amount'] = claim_amount
        elif entity == 'CLAIM_TYPE':
            # Mappen des Schadenstypen zu den jeweiligen claim_type_* Spalten
            feature_row[f'claim_type_{value}'] = 1
        elif entity == 'CLAIM_DATE':
            # Umwandeln des Datums, um nur das Jahr zu extrahieren
```

```
            try:
                year = pd.to_datetime(value).year
                feature_row['claim_year'] = year
            except:
                pass
        elif entity == 'BIRTH_DATE':
            # Umwandeln des Geburtsdatums, um nur das Jahr zu extrahieren
            year = pd.to_datetime(value).year
            feature_row['birth_year'] = year
        elif entity == 'INSURANCE_NUMBER':
            pass # wird nicht gemappt

    # Rückgabe des gemappten Feature-Vektors
    return pd.Series(feature_row)
```

[84]:
```
for i, mail in enumerate(emails):
    # Extrahiere benannte Entitäten aus der E-Mail
    entities = extract_named_entities(mail['body'])
    # Mappen der Entitäten auf das Feature-Template
    feature_row = map_entities_to_features(entities, X)
    # Erstelle eine Kostenvorhersage für die E-Mail
    feature_row['expected_claim_amount'] = model.predict(feature_row.values.
 ↪reshape(1, -1))[0]
    print(f"""
Vorhersage für E-Mail {i + 1}:
Erwartete Kosten: {feature_row['expected_claim_amount']:.2f} EUR
Vom Kunden angegebene Entitäten:""")
    for entity in entities:
        print(f"- {entity[0]} ({entity[1]})")
```

```
Vorhersage für E-Mail 1:
Erwartete Kosten: 747.01 EUR
Vom Kunden angegebene Entitäten:
- Stella Zahn (NAME)
- Auffahrunfall (CLAIM_CAUSE)
- Schnellstrasse (CLAIM_LOCATION)
- Fahrrad (CLAIM_ITEM)
- 671,34 EUR (CLAIM_AMOUNT)

Vorhersage für E-Mail 2:
Erwartete Kosten: 164.58 EUR
Vom Kunden angegebene Entitäten:
- Blitzschlag (CLAIM_TYPE)
- Haus (CLAIM_LOCATION)
- Router (CLAIM_ITEM)
- 164,76 EUR (CLAIM_AMOUNT)
- 7571677966 (INSURANCE_NUMBER)
```

```
Vorhersage für E-Mail 3:
Erwartete Kosten: 139.85 EUR
Vom Kunden angegebene Entitäten:
- 14. November (CLAIM_DATE)
- Einbruch (CLAIM_TYPE)
- Wohnung (CLAIM_LOCATION)
- Laptop (CLAIM_ITEM)
- Ivana Gröttner (NAME)
- 09.07.1974 (BIRTH_DATE)

Vorhersage für E-Mail 4:
Erwartete Kosten: 139.60 EUR
Vom Kunden angegebene Entitäten:
- Sabine Barth (NAME)
- 07.02.1950 (BIRTH_DATE)
- 7434895569 (INSURANCE_NUMBER)
- Vandalismus (CLAIM_TYPE)
- 01.03.2020 (CLAIM_DATE)
- Parkplatz (CLAIM_LOCATION)
- Graffiti (CLAIM_CAUSE)
- Auto (CLAIM_ITEM)
- Sabine Barth (NAME)

Vorhersage für E-Mail 5:
Erwartete Kosten: 384.98 EUR
Vom Kunden angegebene Entitäten:
- 1247194832 (INSURANCE_NUMBER)
- Blitzschlag (CLAIM_TYPE)
- 2020-04-19 (CLAIM_DATE)
- Garten (CLAIM_LOCATION)
- Elektrische Störung (CLAIM_CAUSE)
- Fernseher (CLAIM_ITEM)
- 333,24 EUR (CLAIM_AMOUNT)
- Angelique Schottin (NAME)
- 1973-08-13 (BIRTH_DATE)
```

4.1 Fazit

Die Pipeline zeigt vielversprechende Ergebnisse und die Kostenerwartungen für die ausgewählten Beispiel-E-Mails liegen in einem realistischen Bereich. Dennoch ist für einen produktiven Einsatz eine deutlich umfassendere und kontinuierliche Evaluation unabdingbar.

Insbesondere sollte im Kontext regulatorischer Anforderungen wie dem AI Act sichergestellt werden, dass ein angemessenes Human Oversight etabliert ist, um Entscheidungen nachvollziehbar und verantwortungsvoll zu überwachen. Die langfristige Wirksamkeit der Lösung erfordert zudem die Beachtung des gesamten Life Cycle, inklusive Maßnahmen gegen Model Drift und Anpassungen an sich verändernde Daten- und Schadensmuster.

Zudem ist es notwendig, die Validität der Zeitreihen-Daten regelmäßig zu prüfen oder mit einem Sicherheitsaufschlag zu versehen, um Risiken durch Datenfehler oder -veränderungen abzufedern. Dies entspricht auch den geltenden Standesregeln für Versicherungsmathematiker*innen, die eine belastbare und nachvollziehbare Risikobewertung fordern.

In der Praxis bedeutet dies, dass neben der initialen Modellvalidierung eine kontinuierliche Überwachung und regelmäßige Nachjustierung in der Produktion implementiert werden müssen. Für die Beispiel-E-Mails könnte bereits ein vorsichtiger Sicherheitsaufschlag in der Kostenschätzung berücksichtigt werden, um Unsicherheiten abzumildern.

5 Aufgabe 5 Management Summary

Im Rahmen des Projekts bei SecureEverything wurde eine KI-gestützte Pipeline entwickelt, die unstrukturierte Kunden-E-Mails automatisch analysiert, um Schadensinformationen zu extrahieren und darauf basierend Kostenvorhersagen zu generieren. Ziel ist es, den zeitaufwendigen manuellen Prozess der Schadensbearbeitung effizienter zu gestalten.

5.1 Projektergebnisse

- **Explorative Datenanalyse (EDA):**
  Die vorliegenden Daten sind relativ klein und weisen eine starke Klassenungleichheit auf. Insbesondere der „O“-Tag (irrelevante Wörter) dominiert, was die präzise Extraktion relevanter Informationen erschwert. Die begrenzte Datenmenge schränkt die Modellierungsmöglichkeiten ein.
- **Modellentwicklung und Training:**
  Ein BiLSTM-basiertes Named Entity Recognition (NER)-Modell wurde trainiert. Die Modellgenauigkeit liegt jedoch nahe am Zufallsniveau, was auf die schwierige Datenstruktur und unausgewogene Klassenverteilung zurückzuführen ist. Die Verwendung eines freien vortrainierten Modells aus dem Python-Paket `spacy` brachte bessere Extraktionsergebnisse.
- **Kostenvorhersage:**
  Für die Vorhersage der Schadenskosten wurde ein RandomForest-Modell auf historischen Schadensdaten trainiert. Dieses Modell liefert brauchbare und realistische Kostenschätzungen für die geprüften Beispiel-E-Mails.

5.2 Fazit und Handlungsempfehlungen

Die entwickelte Pipeline zeigt grundsätzlich vielversprechende Ansätze und realistische Ergebnisse für ausgewählte Beispiel-E-Mails. Ein produktiver Einsatz erfordert jedoch eine deutlich umfangreichere und kontinuierliche Evaluierung sowie Anpassungen.

Wichtige Aspekte, die im Kontext regulatorischer Vorgaben wie dem AI Act berücksichtigt werden müssen:

- **Human Oversight:**
  Es muss ein angemessenes menschliches Überwachungssystem etabliert werden, um Entscheidungen nachvollziehbar und verantwortungsvoll zu kontrollieren.
- **Life Cycle Management:**
  Die Langzeitwirkung der Modelle muss überwacht werden, um Model Drift zu erkennen und

Anpassungen an veränderte Daten- und Schadensmuster vorzunehmen.

- **Datenvalidität und Sicherheitsaufschläge:**
  Die Qualität der Zeitreihen-Daten sollte regelmäßig geprüft oder durch Sicherheitsaufschläge in den Kostenschätzungen kompensiert werden. Dies entspricht den Standesregeln für Versicherungsmathematiker*innen und gewährleistet eine belastbare Risikobewertung.

Für einen erfolgreichen produktiven Einsatz sind neben der initialen Modellvalidierung insbesondere kontinuierliche Überwachung, regelmäßige Nachjustierung und eine Absicherung gegen Unsicherheiten durch Sicherheitsfaktoren essenziell.

Empfehlungen für SecureEverything:
- Ausbau und Erweiterung der Trainingsdatenbasis
- Verbesserung der Vorverarbeitung und Modellarchitektur
- Implementierung eines umfassenden Monitorings und Reporting-Systems
- Sicherstellung der Einhaltung regulatorischer Anforderungen
- Einbindung von Fachexpert*innen zur humanen Kontrolle und Nachvollziehbarkeit

Mit diesen Maßnahmen kann die Automatisierung der Schadensbearbeitung nachhaltig und effizient umgesetzt werden.

ADS_Klausur_2025

ADS Completion

2025-05-07

Aufgabe B: Kundennutzen POG

Sie arbeiten bei einem Lebensversicherungsunternehmen. Im Rahmen des POG Prozesses (vgl. https://www.bafin.de/SharedDocs/Veroeffentlichungen/DE/Konsultation/2022/kon_08_22_Konsultation_wohlverhaltensaufsichliche_Aspekte_lv_produkte.html) ist es ihre Aufgabe den Kundennutzens ihrer Produkte nachzuweisen, in diesem Beispiel für ein rein fondsgebundenes Versicherungsprodukt. Der Nachweis ist je Zielgruppe zum Ende der Vertragsdauer und zum Zeitpunkt, zu dem 50% des Bestands storniert haben nachzuweisen.

Aufgabe 1

Im ersten Schritt erstellen Sie daher ein Modell, um vertragsindividuell diese beiden Zeitpunkte ermitteln zu können. Es soll ein Modell zur Vorhersage der Größe "Anteil Vertrag" eines Lebensversicherungsvertrages erstellt werden. Diese Größe enthält den Quotienten aus tatsächlicher Dauer und Vertragsdauer. D.h. wenn der Vertrag storniert hat, hat dieses Vertragsfeld einen Wert <1 und falls der Vertrag nicht storniert hat dieses Vertragsfeld den Wert 1. Die Daten sind in der Datei `vertragsdaten.csv` gespeichert.

Aufgabe 1.1.: Lesen Sie die Daten ein und verschaffen Sie sich einen Überblick über die Variablen und die Zielgröße. Entwickeln Sie eine Gradient Boosting Machine (GBM) zur Vorhersage der Zielgröße. Welche Hyperparameter sind wichtig? Erklären Sie, welche drei Gütemaße gut bzw. welche drei Gütemaße weniger gut geeignet sind, um das Modell für die Anwendung zu bewerten.

Im Vertrieb des Produkts wird die Zielgruppe „Versicherungssumme" verwendet. Schätzen Sie für jede Zielgruppe (vier Zielgruppen) den Zeitpunkt, zu dem 50% der Verträge storniert haben (als Prozentzahl der gesamten Versicherungsdauer). Verwenden Sie dabei Ihre Prognose des GBM.

Musterlösung:

```
library(gbm)
```

```
## Loaded gbm 2.2.2
```

```
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.co
```

```
library(caret)
```

```
## Lade nötiges Paket: ggplot2
```

```
## Lade nötiges Paket: lattice
```

```
library(rpart)
library(rpart.plot)
library(glmnet)
```

```
## Lade nötiges Paket: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
library(cluster)
library(clustMixType)
library(knitr)
library(data.table)
library(magrittr)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v lubridate 1.9.3     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::between()     masks data.table::between()
## x tidyr::expand()      masks Matrix::expand()
## x tidyr::extract()     masks magrittr::extract()
## x dplyr::filter()      masks stats::filter()
## x dplyr::first()       masks data.table::first()
## x lubridate::hour()    masks data.table::hour()
## x lubridate::isoweek() masks data.table::isoweek()
## x dplyr::lag()         masks stats::lag()
## x dplyr::last()        masks data.table::last()
## x purrr::lift()        masks caret::lift()
## x lubridate::mday()    masks data.table::mday()
## x lubridate::minute()  masks data.table::minute()
## x lubridate::month()   masks data.table::month()
## x tidyr::pack()        masks Matrix::pack()
## x lubridate::quarter() masks data.table::quarter()
## x lubridate::second()  masks data.table::second()
## x purrr::set_names()   masks magrittr::set_names()
## x purrr::transpose()   masks data.table::transpose()
## x tidyr::unpack()      masks Matrix::unpack()
## x lubridate::wday()    masks data.table::wday()
## x lubridate::week()    masks data.table::week()
## x lubridate::yday()    masks data.table::yday()
## x lubridate::year()    masks data.table::year()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
df = fread("M:/DAA/2025-03-ADS Completion/vertragsdaten.csv")
df %<>% mutate(Alter = factor(Alter,
                              levels = c("<18","18-30","31-40","41-50","51-60",">60")),
               Einkommen = as.factor(Einkommen),
               Haushalt = as.factor(Haushalt),
               Eigentum = as.factor(Eigentum),
               Bildung = as.factor(Bildung),
               Versicherungssumme = factor(Versicherungssumme,
                                           levels = c("0-50k","51k-100k","101k-150k","151k+")),
               Geschlecht = as.factor(Geschlecht),
               Fonds = as.factor(Fonds),
               Vertrieb = as.factor(Vertrieb),
               Zahlweise = as.factor(Zahlweise),
```

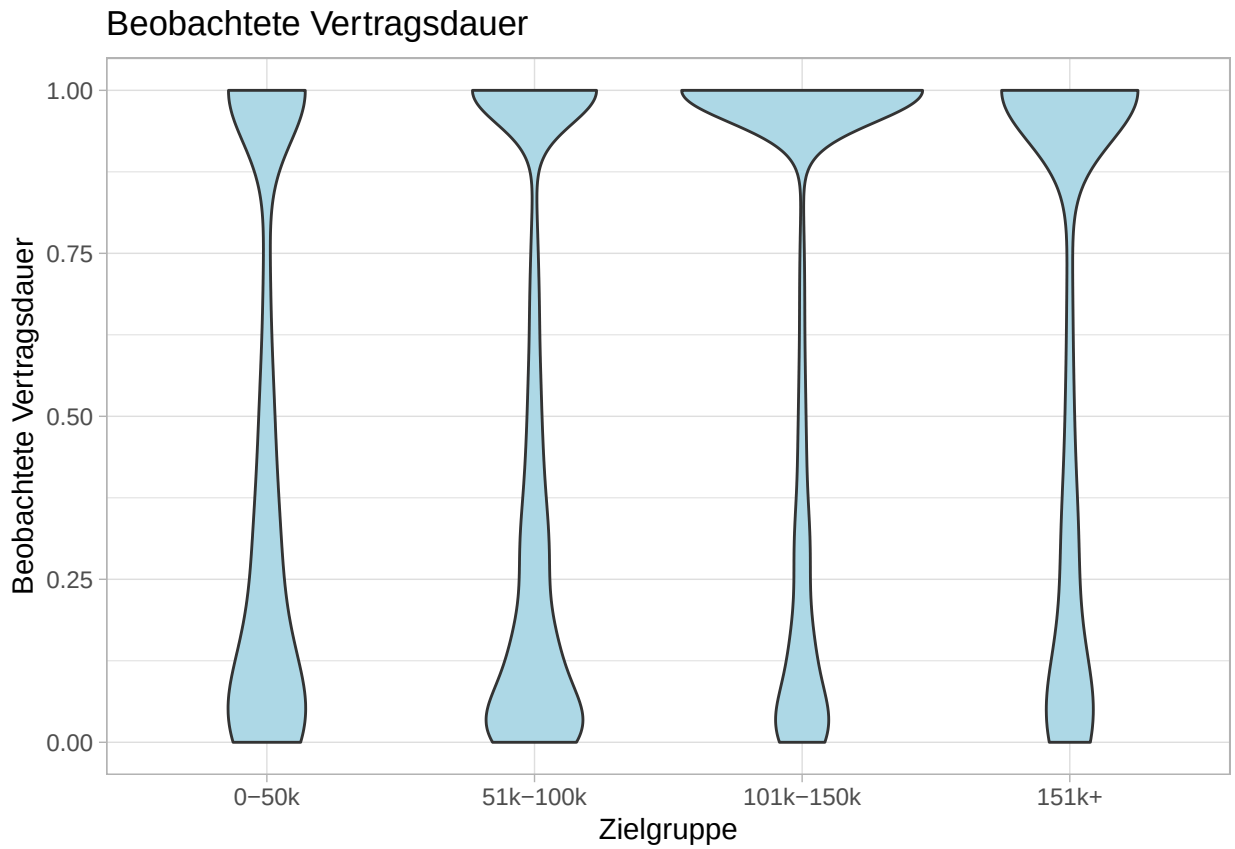
```
               Land = as.factor(Land))

summary(df)
```

```
##    Alter         Einkommen        Haushalt         Eigentum        Bildung
##  <18  :  674   0-20k  : 2371   2      :18123   Eigentum:10526   hoch   :13362
##  18-30: 4978   21k-40k:15985   3      :15491   Miete   :39474   mittel :23466
##  31-40:12154   41k-60k:23624   1      : 8762                    niedrig:13172
##  41-50:15939   61k-80k: 7563   4      : 5097
##  51-60:11275   81k+   :  457   0      : 1827
##  >60  : 4980                   5      :  666
##                                (Other):   34
##  Vertragsdauer   Versicherungssumme Geschlecht Fonds       Vertrieb
##  Min.   :12.00   0-50k    : 2399    M:24916    A:20529   Direkt:48541
##  1st Qu.:19.00   51k-100k :22654    W:25084    B:29471   Makler: 1459
##  Median :20.00   101k-150k:22525
##  Mean   :19.99   151k+    : 2422
##  3rd Qu.:21.00
##  Max.   :28.00
##
##         Zahlweise     Land        Vertragsziel
##  Halbjährlich: 1758   AT: 2442   Min.   :0.0000
##  Jährlich    :48242   DE:47558   1st Qu.:0.1207
##                                  Median :0.9031
##                                  Mean   :0.5990
##                                  3rd Qu.:1.0000
##                                  Max.   :1.0000
##
```

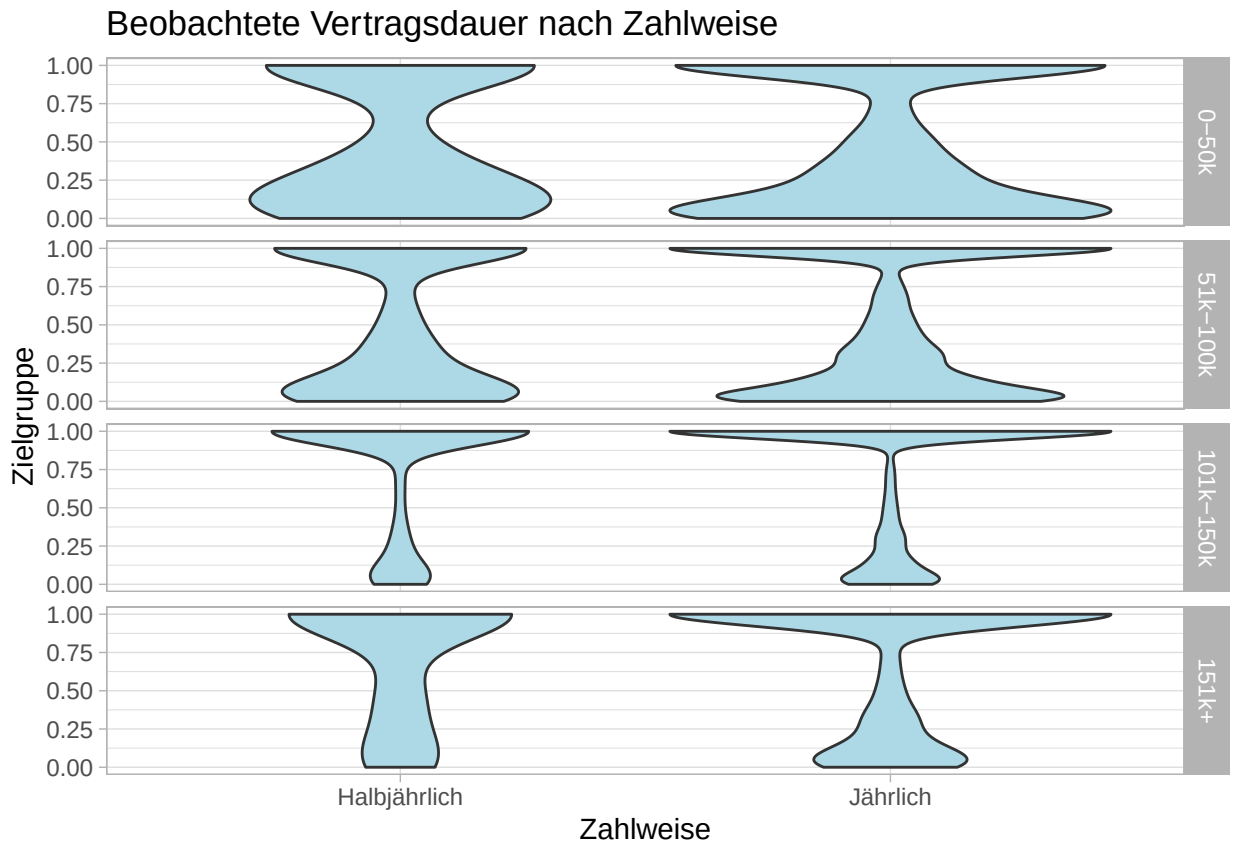
```
#relevel order in Targetvariable:

set.seed(1234)
train_index <- createDataPartition(df$Vertragsziel, p = 0.8, list = FALSE)
train_data <- df[train_index, ]
test_data  <- df[-train_index, ]
```

Da die Daten bimodal bezüglich der Zielgröße sind, bieten sich violin-Plots an. Dies liegt in der Natur der Daten. Es ist Frühstorno beobachtbar und dass eben doch viele Verträge bis zum Vertragsende bestehen bleiben.

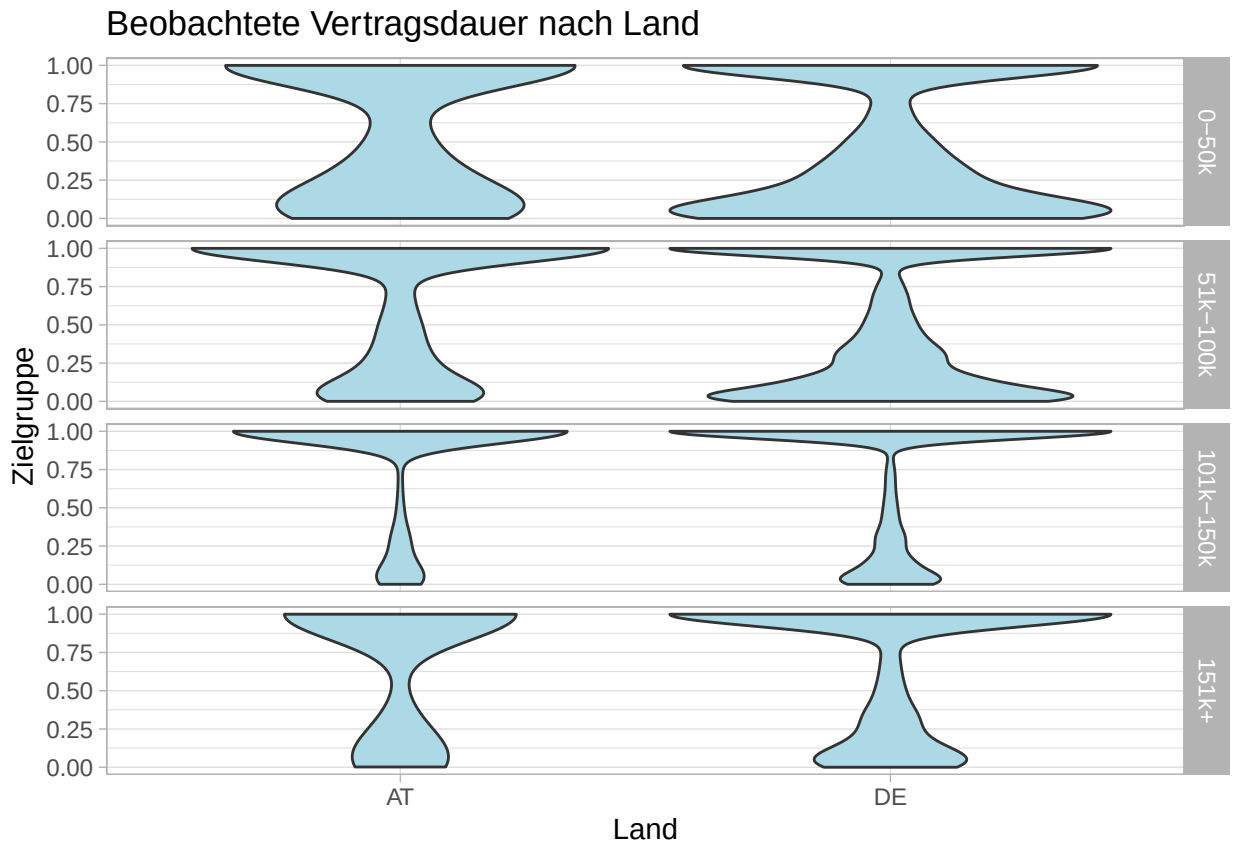
```
pic = ggplot(df, aes(x = Versicherungssumme, y = Vertragsziel)) +
  geom_violin(fill = "lightblue") +
  labs(title = "Beobachtete Vertragsdauer",
       x = "Zielgruppe",
       y = "Beobachtete Vertragsdauer") +
  theme_light() +
  scale_color_brewer(palette = "Set1")
pic
```



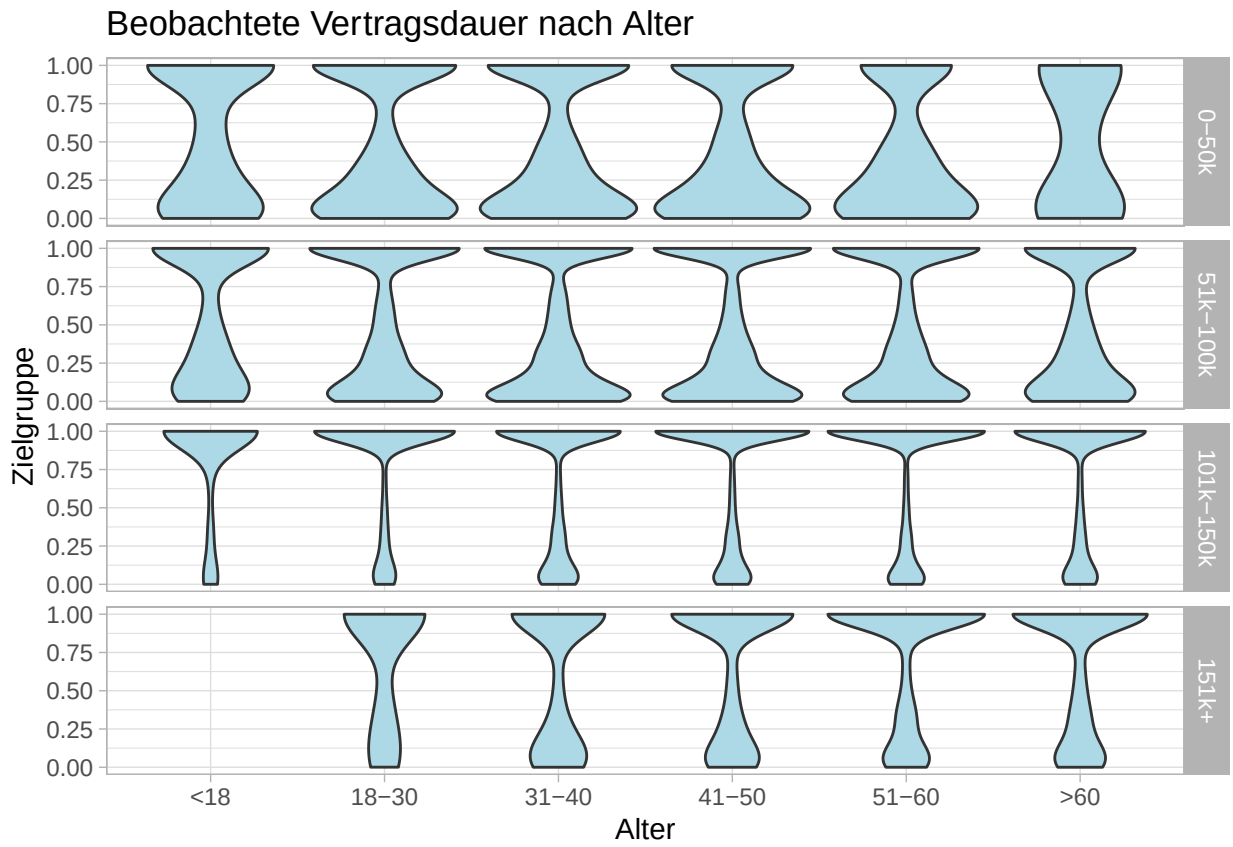

```
pic = ggplot(df, aes(x = Zahlweise, y = Vertragsziel)) +
  geom_violin(fill = "lightblue") +
  labs(title = "Beobachtete Vertragsdauer nach Zahlweise",
       x = "Zahlweise",
       y = "Zielgruppe") +
  theme_light() +
  scale_color_brewer(palette = "Set1") +
  facet_grid(rows = vars(Versicherungssumme))
pic
```



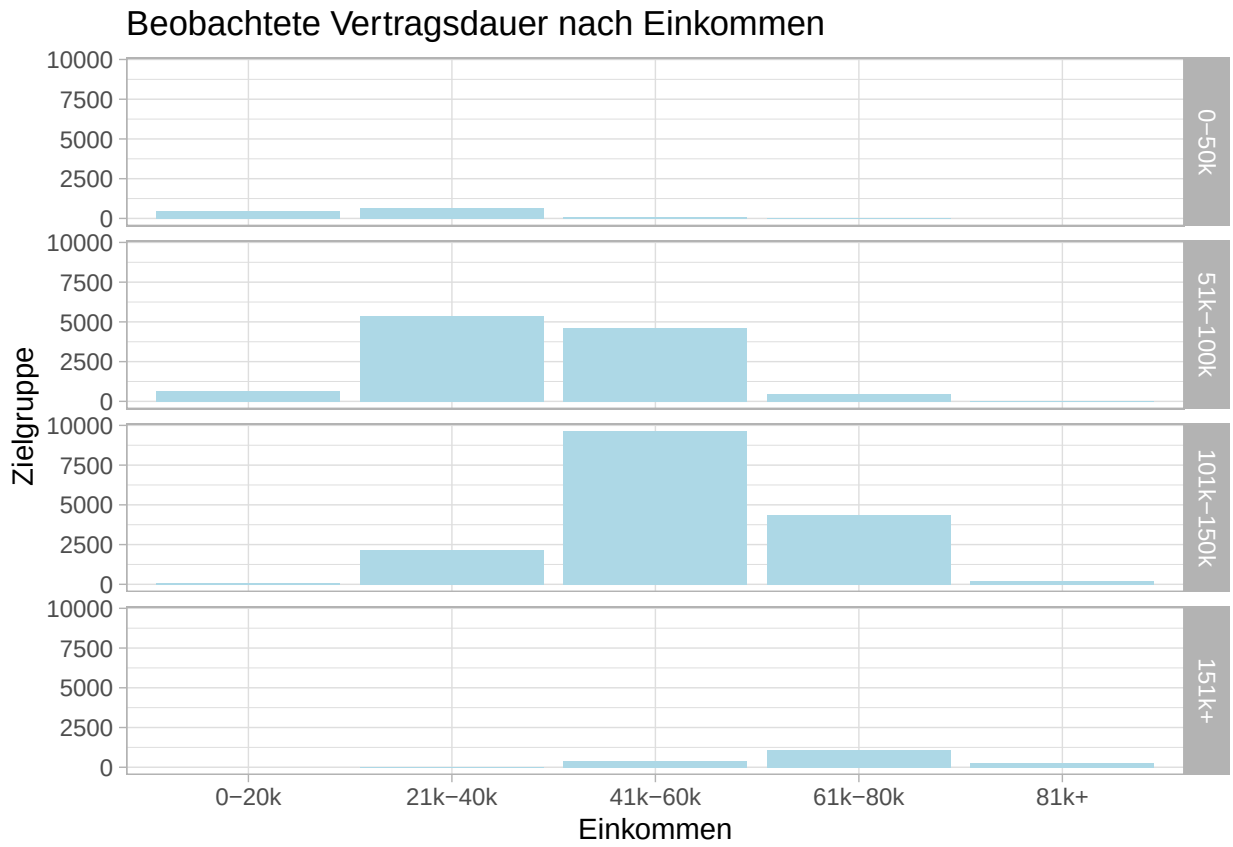

```
pic = ggplot(df, aes(x = Land, y = Vertragsziel)) +
  geom_violin(fill = "lightblue") +
  labs(title = "Beobachtete Vertragsdauer nach Land",
       x = "Land",
       y = "Zielgruppe") +
  theme_light() +
  scale_color_brewer(palette = "Set1") +
  facet_grid(rows = vars(Versicherungssumme))
pic
```



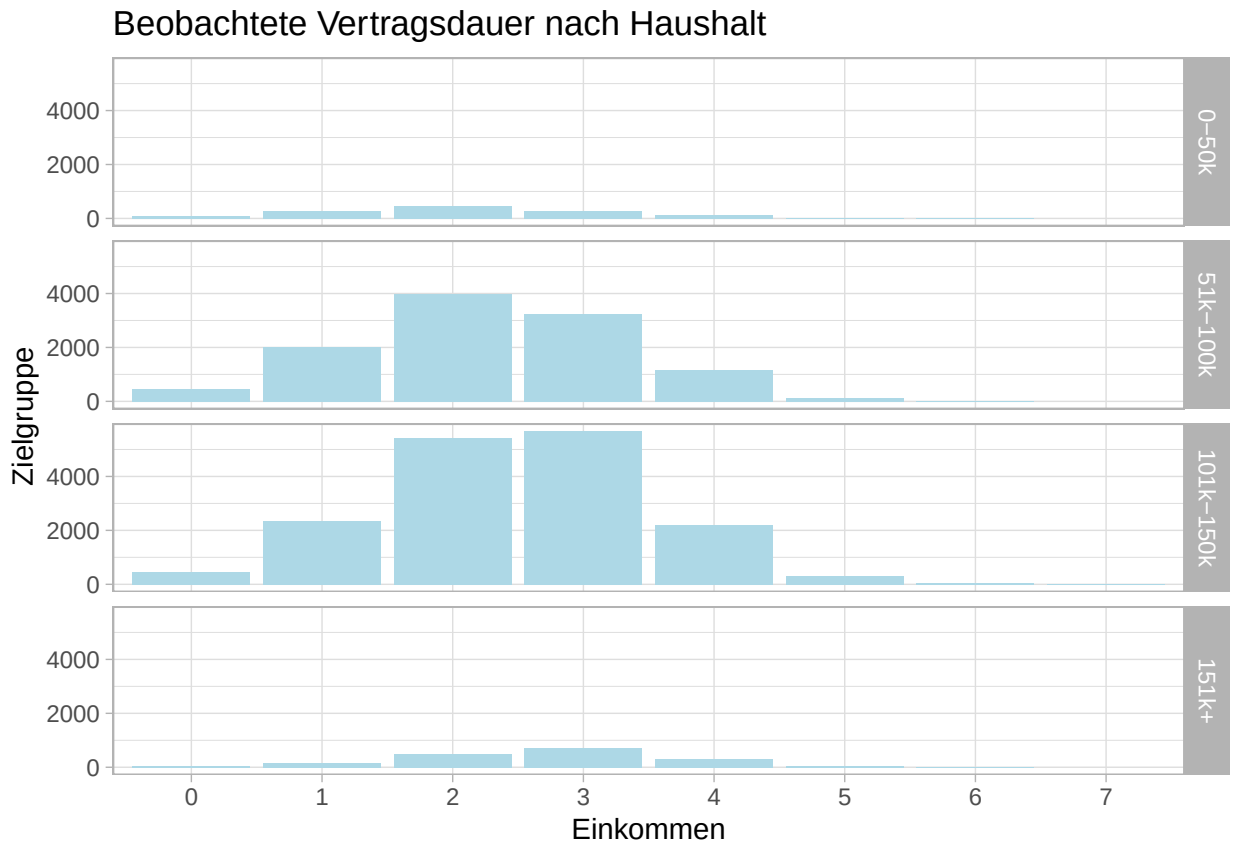

```
pic = ggplot(df, aes(x = Alter, y = Vertragsziel)) +
  geom_violin(fill = "lightblue") +
  labs(title = "Beobachtete Vertragsdauer nach Alter",
       x = "Alter",
       y = "Zielgruppe") +
  theme_light() +
  scale_color_brewer(palette = "Set1") +
  facet_grid(rows = vars(Versicherungssumme))
pic
```



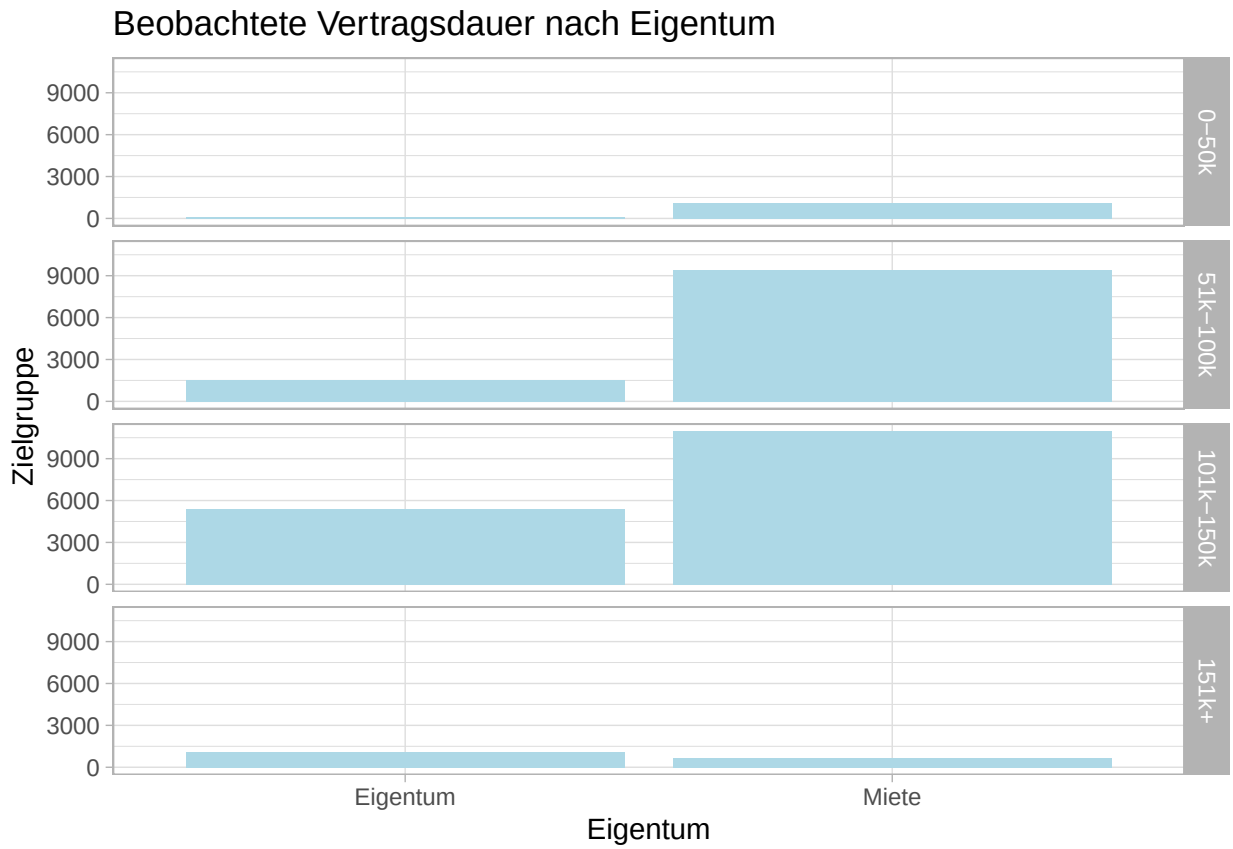

```
pic = ggplot(df, aes(x = Einkommen, y = Vertragsziel)) +
  geom_col(fill = "lightblue") +
  labs(title = "Beobachtete Vertragsdauer nach Einkommen",
       x = "Einkommen",
       y = "Zielgruppe") +
  theme_light() +
  scale_color_brewer(palette = "Set1") +
  facet_grid(rows = vars(Versicherungssumme))
pic
```



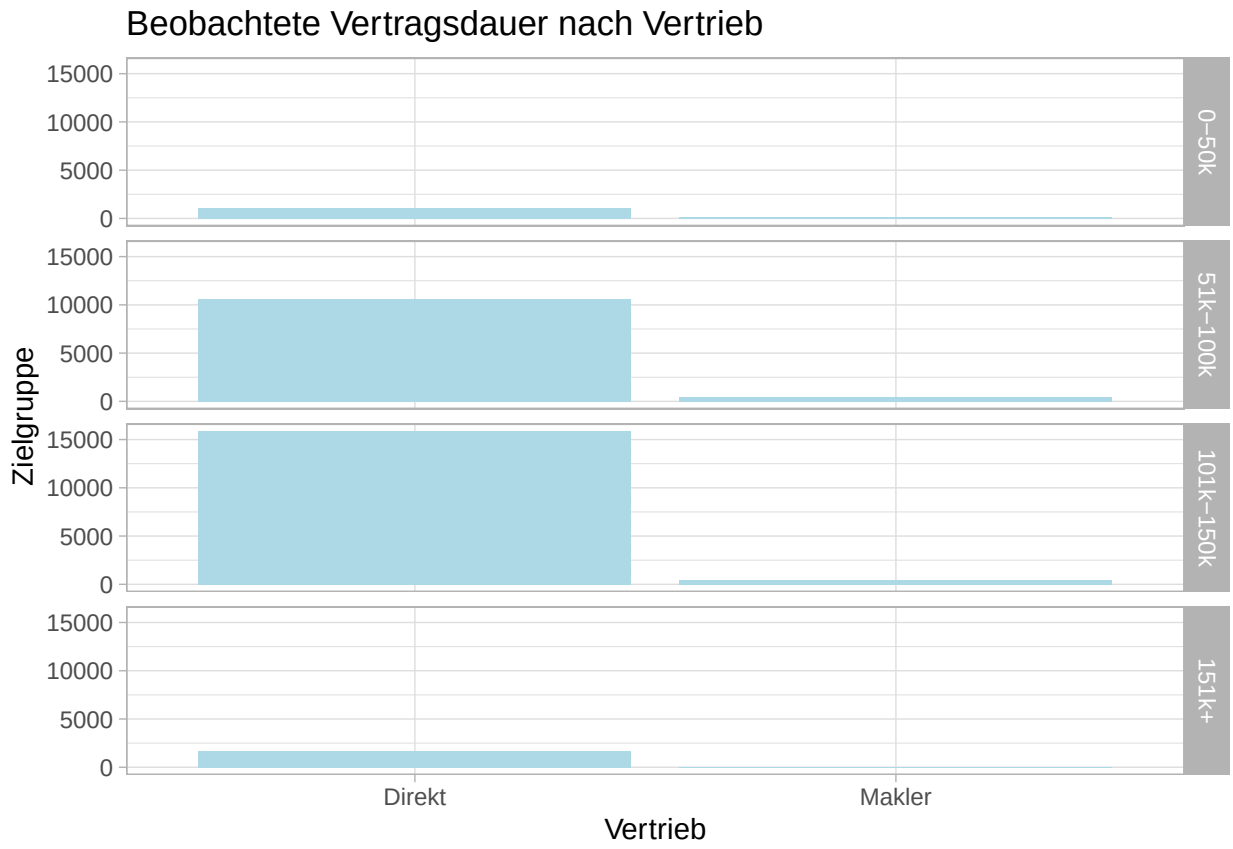

```
pic = ggplot(df, aes(x = Haushalt, y = Vertragsziel)) +
  geom_col(fill = "lightblue") +
  labs(title = "Beobachtete Vertragsdauer nach Haushalt",
       x = "Einkommen",
       y = "Zielgruppe") +
  theme_light() +
  scale_color_brewer(palette = "Set1") +
  facet_grid(rows = vars(Versicherungssumme))
pic
```



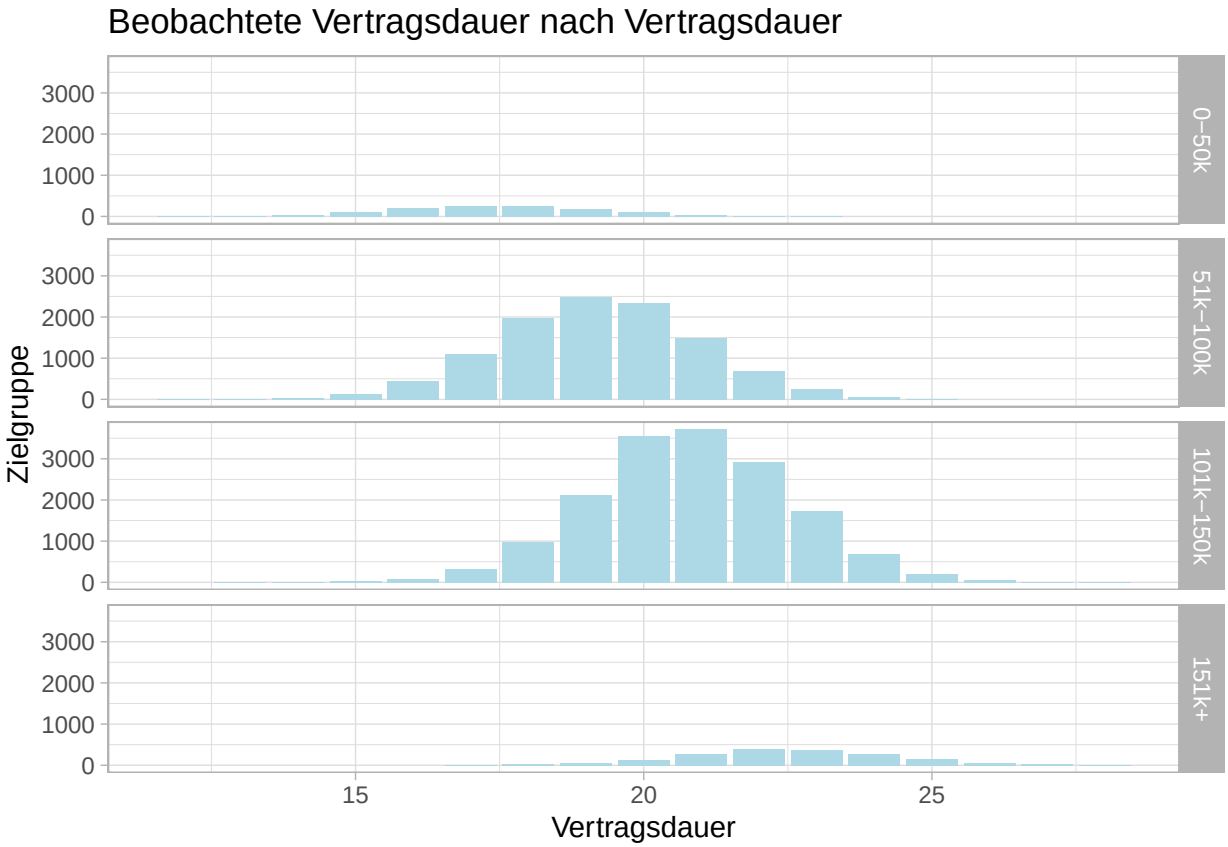

```
pic = ggplot(df, aes(x = Eigentum, y = Vertragsziel)) +
  geom_col(fill = "lightblue") +
  labs(title = "Beobachtete Vertragsdauer nach Eigentum",
       x = "Eigentum",
       y = "Zielgruppe") +
  theme_light() +
  scale_color_brewer(palette = "Set1") +
  facet_grid(rows = vars(Versicherungssumme))
pic
```




```
pic = ggplot(df, aes(x = Vertrieb, y = Vertragsziel)) +
  geom_col(fill = "lightblue") +
  labs(title = "Beobachtete Vertragsdauer nach Vertrieb",
       x = "Vertrieb",
       y = "Zielgruppe") +
  theme_light() +
  scale_color_brewer(palette = "Set1") +
  facet_grid(rows = vars(Versicherungssumme))
pic
```




```
pic = ggplot(df, aes(x = Vertragsdauer, y = Vertragsziel)) +
  geom_col(fill = "lightblue") +
  labs(title = "Beobachtete Vertragsdauer nach Vertragsdauer",
       x = "Vertragsdauer",
       y = "Zielgruppe") +
  theme_light() +
  scale_color_brewer(palette = "Set1") +
  facet_grid(rows = vars(Versicherungssumme))
pic
```




```
kable(df %>% group_by(Versicherungssumme) %>%
  summarise(Freqmean = mean(Vertragsziel),
            Vertragsdauer_mean = mean(Vertragsdauer),
            Anzahl = n(),
            ),
  caption = "Empirische Verteilung der Zielgruppen")
```

Table 1: Empirische Verteilung der Zielgruppen

Versicherungssumme	Freqmean	Vertragsdauer_mean	Anzahl
0-50k	0.4659327	17.46478	2399
51k-100k	0.4810076	19.20464	22654
101k-150k	0.7233411	20.78650	22525
151k+	0.6782211	22.52436	2422

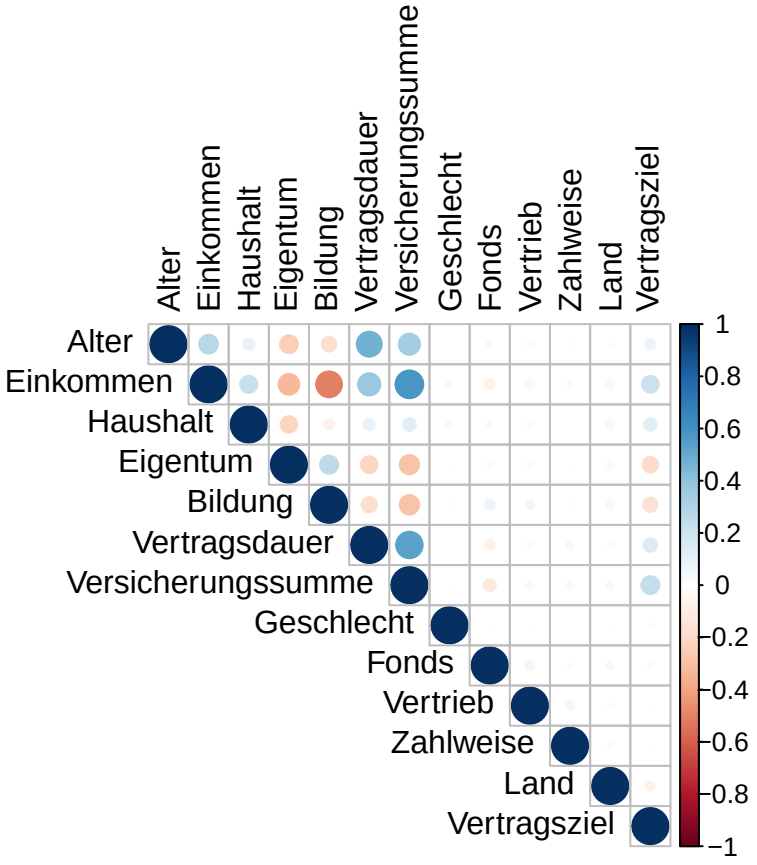
Die meisten Merkmale haben in der Univariaten Sicht einen Einfluss auf die Zielgröße. Deshalb erscheint es lohnenswert ein multivariates Modell zu verwenden.

```
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```
#convert categorical variables to numeric:
df_num = df %>% mutate_if(is.character, as.numeric) %>%
  mutate_if(is.factor, as.numeric)
```

```r
corrplot(cor(df_num), method = "circle", type = "upper", tl.col = "black")
```



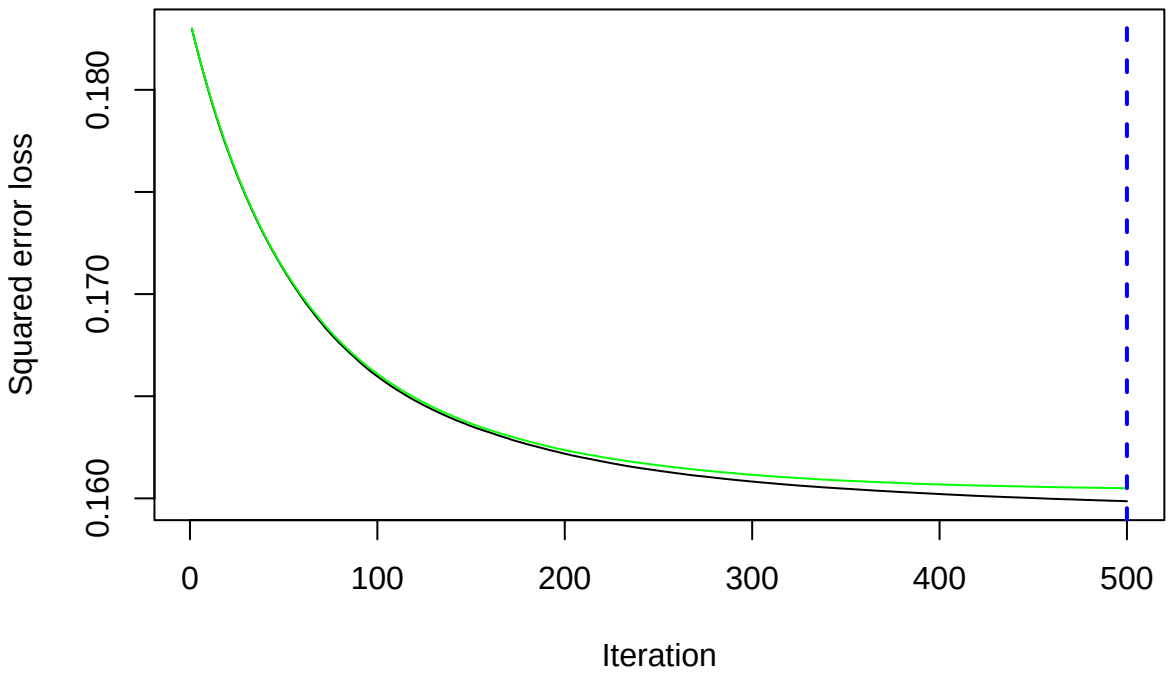

```r
grid = expand.grid(n.trees = c(100, 500),
                   interaction.depth = c(1, 3, 5),
                   shrinkage = c(0.01, 0.1),
                   n.minobsinnode = c(5, 10))

log_loss <- function(y_true, y_pred, epsilon = 1e-15) {
    y_pred <- pmin(pmax(y_pred, epsilon), 1 - epsilon)
    -mean(y_true * log(y_pred) + (1 - y_true) * log(1 - y_pred))
}

for(i in 1:nrow(grid)) {
  gbm_model <- gbm(
    formula = Vertragsziel ~ Alter + Einkommen + Haushalt + Eigentum + Bildung +
                             Vertragsdauer + Versicherungssumme + Geschlecht +
                             Fonds + Vertrieb + Zahlweise + Land,  # alle Prädiktoren
    distribution = "gaussian",
    data = train_data,
    n.trees = grid[i, "n.trees"],
    interaction.depth = grid[i, "interaction.depth"],
    shrinkage = grid[i, "shrinkage"],
    n.minobsinnode = grid[i, "n.minobsinnode"],
    cv.folds = 5,  # Kreuzvalidierung
    verbose = FALSE
  )
```

```
  best_iter <- gbm.perf(gbm_model, method = "cv", plot.it = FALSE)

  p_test <- predict(gbm_model, test_data, n.trees = best_iter, type = "response")

  cat("Log-Loss für Modell", i, "mit Parametern \n
       Learner = ", grid[i, "n.trees"],
      "Tiefe = ",  grid[i, "interaction.depth"],
      "shrinkage = ", grid[i, "shrinkage"],
      "Mindestanzahl = ", grid[i, "n.minobsinnode"], ": ",
      log_loss(test_data$Vertragsziel, p_test), "\n")
}
```

```
## Log-Loss für Modell 1 mit Parametern
##
##        Learner =  100 Tiefe =  1 shrinkage =  0.01 Mindestanzahl =  5 :  0.6460367
## Log-Loss für Modell 2 mit Parametern
##
##        Learner =  500 Tiefe =  1 shrinkage =  0.01 Mindestanzahl =  5 :  0.6288432
## Log-Loss für Modell 3 mit Parametern
##
##        Learner =  100 Tiefe =  3 shrinkage =  0.01 Mindestanzahl =  5 :  0.6382455
## Log-Loss für Modell 4 mit Parametern
##
##        Learner =  500 Tiefe =  3 shrinkage =  0.01 Mindestanzahl =  5 :  0.6232536
## Log-Loss für Modell 5 mit Parametern
##
##        Learner =  100 Tiefe =  5 shrinkage =  0.01 Mindestanzahl =  5 :  0.6350871
## Log-Loss für Modell 6 mit Parametern
##
##        Learner =  500 Tiefe =  5 shrinkage =  0.01 Mindestanzahl =  5 :  0.6218096
## Log-Loss für Modell 7 mit Parametern
##
##        Learner =  100 Tiefe =  1 shrinkage =  0.1 Mindestanzahl =  5 :  0.6256173
## Log-Loss für Modell 8 mit Parametern
##
##        Learner =  500 Tiefe =  1 shrinkage =  0.1 Mindestanzahl =  5 :  0.6277228
## Log-Loss für Modell 9 mit Parametern
##
##        Learner =  100 Tiefe =  3 shrinkage =  0.1 Mindestanzahl =  5 :  0.6227017
## Log-Loss für Modell 10 mit Parametern
##
##        Learner =  500 Tiefe =  3 shrinkage =  0.1 Mindestanzahl =  5 :  0.6228045
## Log-Loss für Modell 11 mit Parametern
##
##        Learner =  100 Tiefe =  5 shrinkage =  0.1 Mindestanzahl =  5 :  0.6221598
## Log-Loss für Modell 12 mit Parametern
##
##        Learner =  500 Tiefe =  5 shrinkage =  0.1 Mindestanzahl =  5 :  0.6220235
## Log-Loss für Modell 13 mit Parametern
##
##        Learner =  100 Tiefe =  1 shrinkage =  0.01 Mindestanzahl =  10 :  0.6460057
## Log-Loss für Modell 14 mit Parametern
##
```

```
##        Learner =  500 Tiefe =  1 shrinkage =  0.01 Mindestanzahl =  10 :  0.628824
## Log-Loss für Modell 15 mit Parametern
##
##        Learner =  100 Tiefe =  3 shrinkage =  0.01 Mindestanzahl =  10 :  0.6382654
## Log-Loss für Modell 16 mit Parametern
##
##        Learner =  500 Tiefe =  3 shrinkage =  0.01 Mindestanzahl =  10 :  0.6232513
## Log-Loss für Modell 17 mit Parametern
##
##        Learner =  100 Tiefe =  5 shrinkage =  0.01 Mindestanzahl =  10 :  0.6351091
## Log-Loss für Modell 18 mit Parametern
##
##        Learner =  500 Tiefe =  5 shrinkage =  0.01 Mindestanzahl =  10 :  0.6218902
## Log-Loss für Modell 19 mit Parametern
##
##        Learner =  100 Tiefe =  1 shrinkage =  0.1 Mindestanzahl =  10 :  0.6256671
## Log-Loss für Modell 20 mit Parametern
##
##        Learner =  500 Tiefe =  1 shrinkage =  0.1 Mindestanzahl =  10 :  0.6278512
## Log-Loss für Modell 21 mit Parametern
##
##        Learner =  100 Tiefe =  3 shrinkage =  0.1 Mindestanzahl =  10 :  0.6225024
## Log-Loss für Modell 22 mit Parametern
##
##        Learner =  500 Tiefe =  3 shrinkage =  0.1 Mindestanzahl =  10 :  0.6230319
## Log-Loss für Modell 23 mit Parametern
##
##        Learner =  100 Tiefe =  5 shrinkage =  0.1 Mindestanzahl =  10 :  0.6220396
## Log-Loss für Modell 24 mit Parametern
##
##        Learner =  500 Tiefe =  5 shrinkage =  0.1 Mindestanzahl =  10 :  0.6218033
```

```r
#Optimales GBM gemäß Grid
gbm_model <- gbm(
    formula = Vertragsziel ~ Alter + Einkommen + Haushalt + Eigentum + Bildung +
                             Vertragsdauer + Versicherungssumme + Geschlecht +
                             Fonds + Vertrieb + Zahlweise + Land,  # alle Prädiktoren
    distribution = "gaussian",
    data = train_data,
    n.trees = 500,
    interaction.depth = 5,
    shrinkage = 0.01,
    n.minobsinnode = 10,
    cv.folds = 5,  # Kreuzvalidierung
    verbose = FALSE
  )

  best_iter <- gbm.perf(gbm_model, method = "cv")
```




```
  p_test <- predict(gbm_model, test_data, n.trees = best_iter, type = "response")
```

Der Log-Loss liegt für die betrachteten Hyperparameter im Bereich 0.62 - 0.65 und ist somit für alle Kombinationen relativ ähnlich. Genügend Bäume (n.trees) und eine ausreichend Tiefe (interaction.depth) führen zu etwas besseren Ergebnissen. Die beste Kombination ist n.trees = 500, interaction.depth = 5, shrinkage = 0.01 und n.minobsinnode = 10.

Für die Gütemaße gilt:

- Gut geeignet ist der Log-Loss oder auch RMSE für die numerische Zielgröße (zwischen 0 und 1). Es kann auch der MAE betrachtet werden.
- Reine Klassifikationsgüte-Maße wie Accuracy, AUC oder PRAUC sind weniger geeignet. Auch eine Konfusionsmatirx kann nicht sinnvoll interpretiert werden.

Musterlösung:

```
df_pred = cbind(test_data,p_test)


kable(df_pred %>% group_by(Versicherungssumme) %>%
  summarise(Freq = median(p_test),
            Vertragsdauer_median = median(Vertragsdauer),
            Vertragsdauer_mean = mean(Vertragsdauer),
            Anzahl = n()),
      caption = "Median Dauer Zielgruppen")
```

Table 2: Median Dauer Zielgruppen

Versicherungssumme	Freq	Vertragsdauer_median	Vertragsdauer_mean	Anzahl
0-50k	0.4513918	17	17.37924	472
51k-100k	0.4582727	19	19.21607	4480
101k-150k	0.7234787	21	20.78473	4557
151k+	0.7044675	23	22.60000	490

Der Datensatz enthält 4 Zielgruppen, die sich in ihrem Stornoverhalten signifikant unterscheiden. Die individuelle Stornoentscheidung ist auf Basis der vorliegenden Vertragsdaten nicht sehr gut vorhersagbar. Trotzdem ist es erforderlich ein gewisses Parametertuning vorzusehen, da sonst keine Differenzierung gefunden werden kann. Das ist durchaus üblich, da die Gründe für Storno sehr vielschichtig sind und dem Versicherungsunternehmen in der Regel nicht vorliegen.

Aufgabe 1.2: Das Modell wird an einer aus Sicht des Verantwortlichen Aktuars kritischen Schnittstelle zum Kunden eingesetzt. Sie möchten deshalb die Ergebnisse des GBM möglichst gut verstehen und interpretieren. Nutzen Sie hierfür Local Interpretable Model-Agnostic Explanations (LIME), siehe https://doi.org/10.1145/2939672.2939778. Wählen Sie anschließend einen zufälligen Vertrag aus dem Datensatz zur Erklärung der Prognose mit LIME.

- Nennen Sie die wesentlichen Schritte des LIME. Implementieren Sie eine eigene LIME-Analyse ohne die Verwendung von LIME spezifischen Paketen.
- Implementieren Sie für jeden wesentlichen Schritt eine weitere, signifikant unterschiedliche Alternative und diskutieren Sie die Wahl und die Ergebnisse.
- Diskutieren Sie die Ergebnisse kritisch. Welche Vor- und Nachteile hat der LIME-Ansatz? Wie unterscheiden sich die Prognosen der Modelle?

Musterlösung:

Die wesentlichen Schritte sind:

- vergleichbare / benachbarte / ähnliche Daten anlegen / auswählen und mit Basismodell (GBM) vorhersagen
- Definition eines Abstandsmaß der Daten um den Abstand zum zu erklärenden Fall zu messen. Dieses Maß kann dann anschließend als Gewichtungsmaß verwendet werden.
- lokales / einfaches Modell auf den Daten bauen unter Verwendung des Abstandsmaßes, z.B. als Gewichtung
- lokales / einfaches Modell interpretieren.

```
#Funktion für Vorhersage des GBM Modells
y_hat <- function(newdata) predict(gbm_model, newdata,
                                   n.trees = best_iter,
                                   type = "response")

#Zeile (Vertrag) festlegen
i = 12
x0 <- df %>% slice(i)
```

```
#Schritt 1: Stör-Daten
#Variante 1: neue Daten erstellen mit Hilfe Verteilung des ursprünglichen Datensatzes
generate_perturbations <- function(x0, n = 500) {
  perturbed <- x0[rep(1, n), , drop = FALSE]
```

```r
  for (col in names(x0)) {
    if (is.numeric(x0[[col]])) {
      perturbed[[col]] <- pmax(0, rnorm(n, mean = mean(df[[col]]),
                                        sd = 0.1 * sd(df[[col]], na.rm = TRUE)))
    } else if (is.factor(x0[[col]])) {
      perturbed[[col]] <- factor(sample(levels(df[[col]]), n, replace = TRUE))
    }
  }

  return(perturbed)
}
X_local <- generate_perturbations(x0)

#Vorhersage vom GBM Modell auf Stördaten
y_local <- y_hat(X_local)
```

```r
#Schritt 2: Distanzgewichtung
#Variante 1: Gower-Distanz
dists_matrix <- as.matrix(daisy(rbind(x0, X_local), metric = "gower"))
dists = dists_matrix[1,-1]
#Gewichtung (z.B. durch Gauß-Funktion)
kernel_weights <- exp(- (dists^2) / 0.25)  # 0.25 ist Bandbreite
```

```r
#Schritt 3: Lokales Modell
#Variante 1: Ridge
X_mat <- model.matrix(Vertragsziel ~ Alter + Einkommen +
                        Haushalt + Eigentum +
                        Bildung + Vertragsdauer +
                        Versicherungssumme + Geschlecht +
                        Fonds + Vertrieb +
                        Zahlweise + Land,
                      data = X_local)[, -1]  # ohne Intercept

# Ridge-Regression mit Gewichtung
ridge_model_cv <- cv.glmnet(
  x = X_mat,
  y = y_local,
  alpha = 0,  # Ridge
  weights = kernel_weights,
  nfolds = 5  # lambda mit cv bestimmen
)
```

```r
#Schritt 4: Erklärbarkeit des lokalen Modells
#Variante 1: Blick auf die Koeffizienten
loc_res = as.matrix(coef(ridge_model_cv, s = "lambda.min"))

#x0 vorbereiten
x0_x = model.matrix(Vertragsziel ~ Alter +
                      Einkommen + Haushalt +
                      Eigentum + Bildung +
                      Vertragsdauer + Versicherungssumme +
                      Geschlecht +
                      Fonds + Vertrieb +
                      Zahlweise + Land,
```

```
                    data = x0)
# Variablennamen im Modell
model_vars <- rownames(loc_res)  # ohne Intercept
# Variablennamen der neuen Beobachtung
nonzero_vars <- x0_x[,which(x0_x != 0)]
new_vars <- names(nonzero_vars)

# Finde die Überschneidung der Variablen
relevant_vars <- intersect(model_vars, new_vars)

# Koeffizienten extrahieren, die für die neuen Variablen relevant sind
relevant_coefs <- loc_res[match(relevant_vars, model_vars), , drop = FALSE]
print(relevant_coefs)
```

```
##                                    s1
## (Intercept)                 0.810459301
## Alter31-40                 -0.092700540
## Einkommen21k-40k            0.004379609
## Haushalt3                   0.026131138
## EigentumMiete              -0.070439400
## Bildungniedrig             -0.048148715
## Vertragsdauer              -0.001354919
## Versicherungssumme51k-100k -0.001257431
## FondsB                     -0.002743147
## ZahlweiseJährlich          -0.008027674
```

```
#Was schätzt das Ridge-Modell für x0?
x0_pred_Ridge = as.numeric(predict(ridge_model_cv,
                                   newx = x0_x[, -1],
                                   s = "lambda.min"))
#Was schätzt das ursprüngliche GBM?
x0_pred_GBM = y_hat(x0)
#Plausi
print(x0_pred_Ridge)
```

```
## [1] 0.8305404
```

```
print(x0_pred_GBM)
```

```
## [1] 0.600282
```

Alternative - Schritte:

```
#Schritt 1: Stör-Daten
#Variante 2: Mittels Cluster Algorithmus "ähnliche" Beobachtungen ziehen
set.seed(12)
clust = kproto(df %>% dplyr::select(-Vertragsziel), k = 10)
```

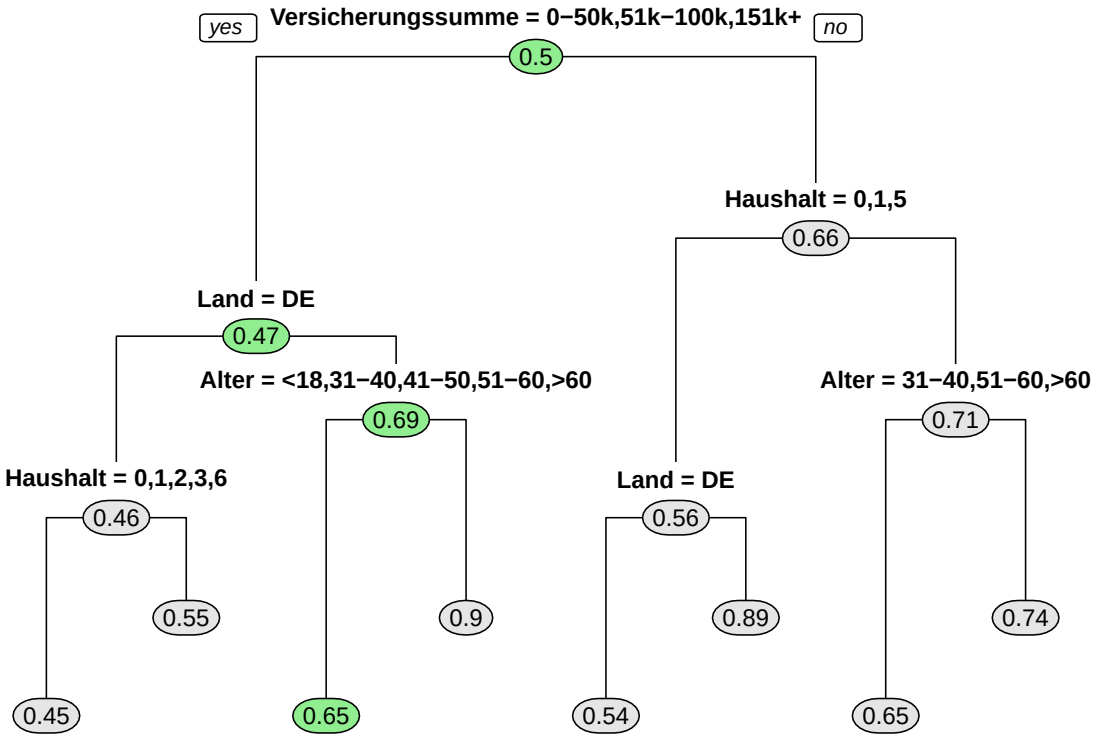
```
## # NAs in variables:
##              Alter          Einkommen           Haushalt           Eigentum
##                  0                  0                  0                  0
##            Bildung      Vertragsdauer Versicherungssumme         Geschlecht
##                  0                  0                  0                  0
##              Fonds           Vertrieb          Zahlweise               Land
##                  0                  0                  0                  0
## 0 observation(s) with NAs.
```

```
##
## Estimated lambda: 9.192363
```

```
X_local = cbind(df, clust$cluster)
#Schauen, welches Cluster x0 hat
x0_cluster = clust$cluster[i]
#Wähle alle Beobachtungen aus dem Cluster
X_local = df[clust$cluster == x0_cluster, ]

#Vorhersage vom GBM Modell auf Stördaten
y_local <- y_hat(X_local)
```

```
#Schritt 2: Distanzgewichtung
#Hinfällig - es wird davon ausgegangen, dass alle Beobachtungen im Cluster ähnlich sind


#Schritt 3: Lokales Modell
#Variante 2: Baum mit rpart
tree_model <- rpart(Vertragsziel ~ Alter + Einkommen + Haushalt + Eigentum + Bildung +
                                   Vertragsdauer + Versicherungssumme + Geschlecht +
                                   Fonds + Vertrieb + Zahlweise + Land,
                    data = X_local,
                    control = rpart.control(cp = 0, minbucket = 5, maxdepth = 3))
```

```
#Schritt 4: Erklärbarkeit des lokalen Modells
#Variante 2: Visualisierung des Baums - farbliche Hervorhebung von x0
all_nodes <- as.numeric(row.names(tree_model$frame))
highlight_nodes = c(1,2,5,10)
node_colors <- ifelse(all_nodes %in% highlight_nodes, "lightgreen", "gray90")
rpart.plot(tree_model,
           type = 1,
           extra = 0,
           fallen.leaves = TRUE,
           box.col = node_colors)
```




```
#Was schätzt der Baum für x0?
x0_pred_Baum = predict(tree_model, newdata = x0)
print(x0_pred_Baum)
```

```
##         1
## 0.648723
```

```
#Was schätzt das ursprüngliche GBM?
x0_pred_GBM = y_hat(x0)
print(x0_pred_GBM)
```

```
## [1] 0.600282
```

Diskussion der Alternativen einfügen

- Schritt 1: Clustering bringt zusätzlich Komplexität. Dafür müssen aber keine Stördaten generiert werden. Die Stördaten sind nicht immer sinnvoll und können zu Verzerrungen führen.
- Schritt 2: Abstandsmaß vor allem bei der ersten Variante wichtig. Hier ist die Gower-Distanz eine Möglichkeit. Bei der zweiten Variante ist es nicht zwingend notwendig, da angenommen wird, dass bei einem (guten) Clustering alle Beobachtungen im Cluster ähnlich sind.
- Schritt 3/4: Ridge-Regression ist ein einfaches Modell, das sich gut für die Interpretation eignet, da einzelne Koeffizienten des Modells ausgelesen und erklärt werden können. Hier sind jedoch (zunächst) keine Interaktionen abgebildet. Der Baum ist ebenfalls einfach zu interpretieren, hat aber etwas mehr Hyperparameter, die man sinnvoll wählen muss. Beispielsweise sollte man darauf achten, dass der Baum nicht zu tief ist, da sonst die Interpretierbarkeit leidet. Ein zentraler Vorteil von Baum-Ansätzen ist die einfache Darstellung von Interaktionseffekten.

Für die von uns gewählte Beobachtungen sind laut Ridge die Features Alter, Eigentum und Bildung wichtig

(absolut gesehen größte Koeffizienten für Beobachtung). Laut dem Baum sind Versicherungssumme, Land und Alter wichtig (Splits für betrachtete Boebachtung).

Ein wesentlicher Vorteil von LIME ist, dass es modellagnostisch arbeitet. Es ist also nicht nur für GBMs anwendbar und dadurch sehr flexibel. Außerdem kann es nützlich sein, wenn spezielle (kritische oder wichtige) Beobachtungen genauer analysiert und erklärt werden sollen. Ein Nachteil ist, dass die Erklärbarkeit eben nur für einzelne Beobachtungen gemacht werden kann. Es kann sein, dass für eine andere Beobachtung ganz andere Features wichtig sind. LIME gibt also kaum eine generelle Aussage über das Modell, sondern nur über die Vorhersage einer einzelnen Beobachtung. Außerdem kann es auch passieren, dass die Vorhersage des lokalen Modells signifikant von der Vorhersage des GBM abweicht (z.B. weil Stördaten nicht sinnvoll sind oder weil das lokale Modell nicht gut ist). Dies schränkt dann natürlich die Aussagekraft der Erklärung ein.

Aufgabe 2

Für ein rein fondsgebundenes Versicherungsprodukt stehen verschiedene Finanzinstrumente aus einem Basket, den ihr Unternehmen anbietet zur Verfügung. Daten aus dem Jahr 2024 sollen repräsentativ verwendet werden und sind im Datensatz "Aktienkurse.csv" enthalten. Dabei entspricht die erste Zeile den Kennungen der Finanzinstrumente (und Datumsspalte). Die zweite Zeile enthält die Kosten der Finanzinstrumente in % pro Jahr, die noch nicht bei der Kursentwicklung berücksichtigt sind. Die weiteren Zeilen enthalten die Kurse und sind jeweils als Opening-Kurse zu verstehen. Die Kurse sind in Euro angegeben.Die Kursverläufe sind zu Beginn auf 100 Euro normiert.

Aufgabe 2.1: Schätzen Sie zu jedem Finanzinstrument eine Rendite und eine Volatilität. Analysieren Sie grafisch, die Kosten / Nutzen / Risikosicht der einzelnen Titel. Finden Sie Anzeichen für gewisse Anlageklassen? Um welche könnte es sich handeln? Gibt es Ausreißer? Schätzen Sie für alle identifizierten Anlageklassen eine durchschnittliche Rendite, Volatilität, und Kosten als Basis für eine Simulation der künftigen Entwicklung.

```
library(tidyr)
library(dplyr)
library(lubridate)
library(stringr)
library(FNN)
library(ggplot2)

Aktienkurse <- read.csv2("M:/DAA/2025-03-ADS Completion/Aktienkurse.csv")
cost = Aktienkurse[1, 2:3001]
Aktienkurse = Aktienkurse[-1, ]
```

Musterlösung:

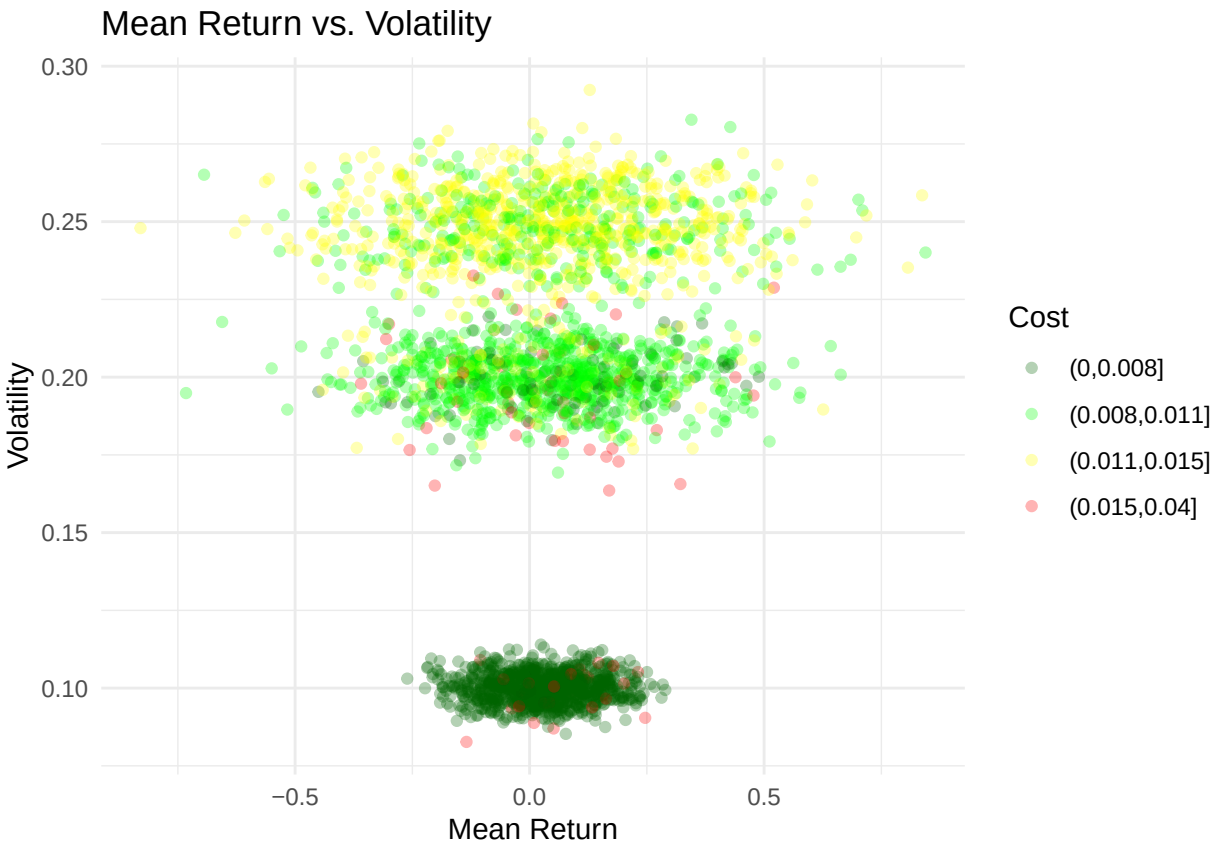
Schätzen der Vola und Rendite

```
estimate_vola_and_mean = function(stock, Datum) {
  df = tibble(stock = stock, Datum = Datum)
  df = df %>% drop_na()
  df = df %>% mutate(Datum = ymd(Datum), year = year(Datum))
  df = df %>% arrange(Datum)
  df = df %>% mutate(daily_return = log(stock) - lag(log(stock)))

  df = df %>% group_by(year) %>% summarise(
    mean_return_log = log(last(stock) / first(stock)),
    sd = sd(daily_return, na.rm = T) * sqrt(n()),
    last_stock = last(stock))
  return(c(mean(df$mean_return_log), mean(df$sd),df$last_stock)) }

# Schätzen der Vola und Rendite
```

```
doall = lapply(2:3001, function(i){
  estimate_vola_and_mean(Aktienkurse[,i],
                         dmy(str_replace_all(Aktienkurse$date,"[[:punct:]]","-"))) })
# Ergebnis aufbereiten


doall = tibble(do.call(rbind,doall))
erg = doall[[1]]
colnames(erg) = c("mean_return_log", "sd","Schlusskurs")
erg = as.data.frame(erg)
```

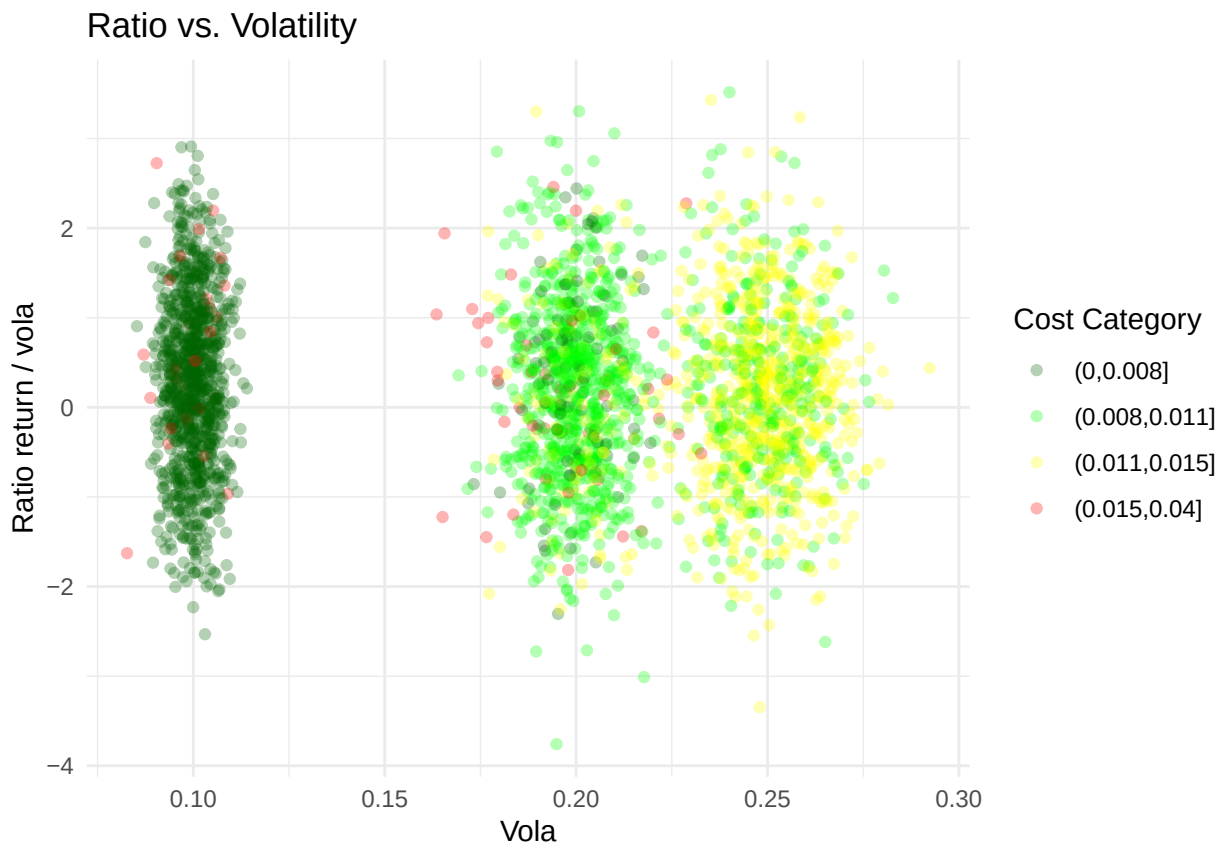
```
erg$cost = unlist(cost)
```

```
#Gruppierung nach Kosten
erg %>% mutate(costcat = cut(cost,breaks = c(0.0,0.008,0.011,0.015,0.04))) %>%
  ggplot(aes(x = log(Schlusskurs/100),y = sd,color = costcat)) +
  geom_point(alpha = 0.3) +
  scale_colour_manual(values = c("darkgreen", "green", "yellow","red"))+
  labs(title = "Mean Return vs. Volatility",
       x = "Mean Return",
       y = "Volatility",
       color = "Cost") +
  theme_minimal()
```

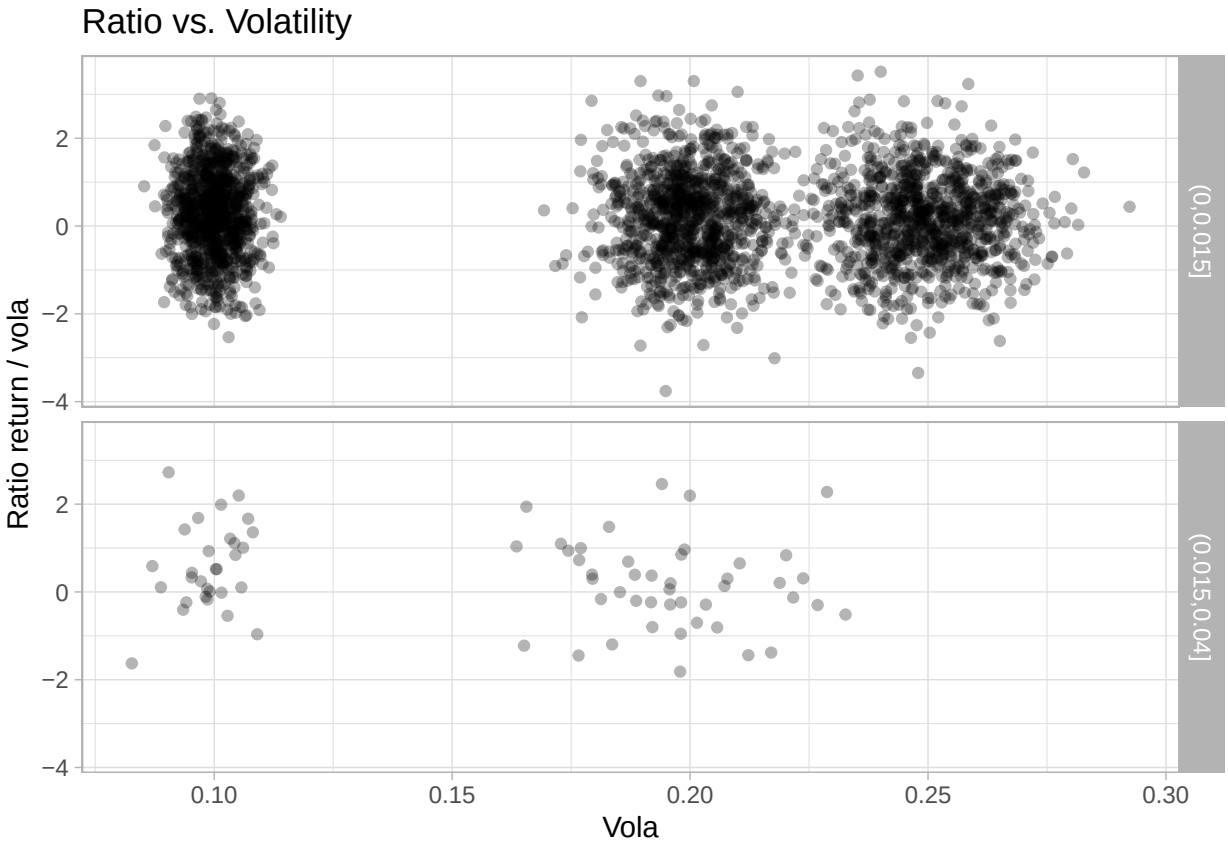



```
#Gruppierung nach Rendite/vola und Kosten
erg %>% mutate(ratio = log(Schlusskurs/100) / sd,
```

```
              costcat = cut(cost,breaks = c(0.0,0.008,0.011,0.015,0.04))) %>%
  ggplot(aes(x = sd,y = ratio,color = costcat)) +
  geom_point(alpha = 0.3) +
  scale_colour_manual(values = c("darkgreen", "green", "yellow","red"))+
  labs(title = "Ratio vs. Volatility",
       x = "Vola",
       y = "Ratio return / vola",
       color = "Cost Category") +
  theme_minimal()
```




```
erg %>% mutate(ratio = log(Schlusskurs/100) / sd,
              costcat = cut(cost,breaks = c(0.0,0.015,0.04))) %>%
  ggplot(aes(x = sd,y = ratio)) +
  geom_point(alpha = 0.3) +
  scale_colour_manual(values = c("darkgreen", "green", "yellow","red"))+
  labs(title = "Ratio vs. Volatility",
       x = "Vola",
       y = "Ratio return / vola",
       color = "Cost Category") +
  theme_light() +
  facet_grid(rows = vars(costcat))
```



Es sind drei große cluster mit einer vola von 10%, 20%, 25% identifizierbar. Dabei handelt es sich vermutlich um ETFs mit unterschiedlichen Volatilitäts-Leveln.

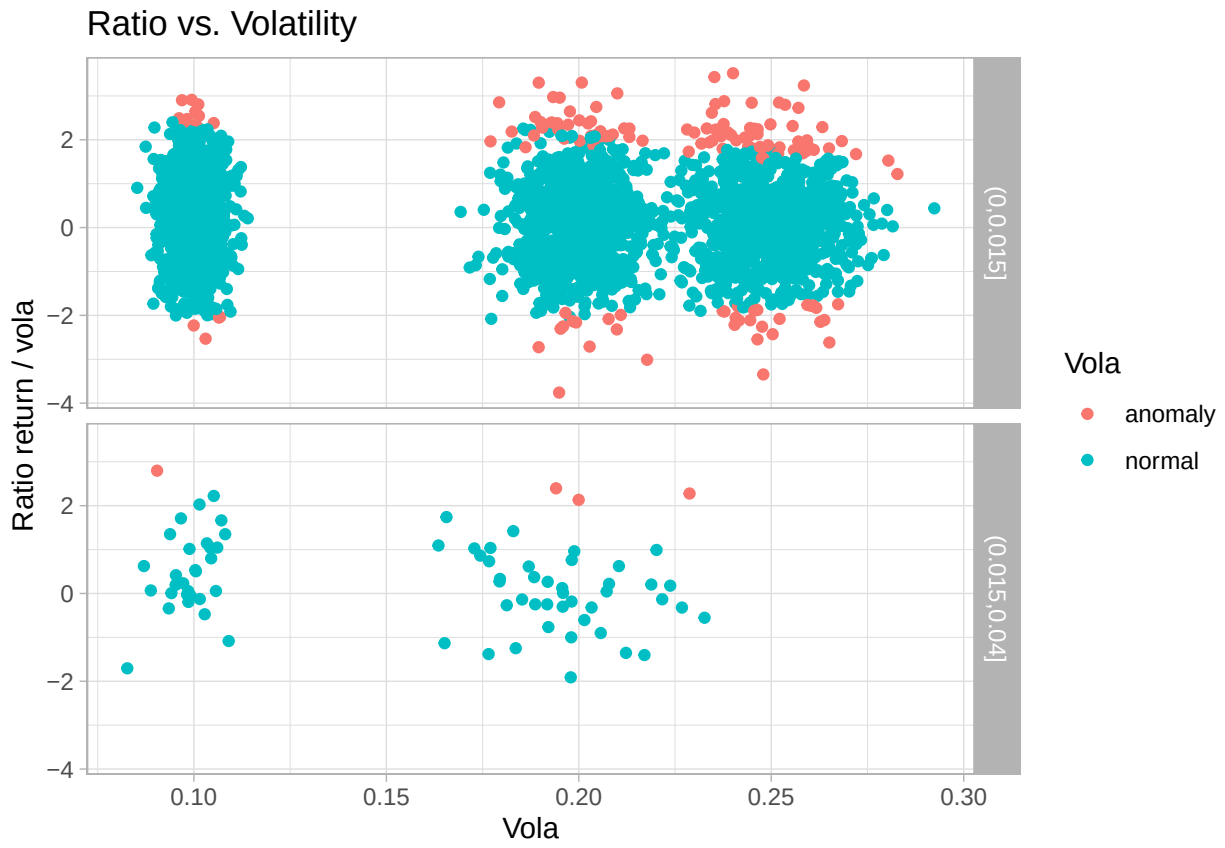
Es sind zwei kleine Cluster mit einer vola von 10%, 20%, relativ hohen Kosten identifizierbar. Dabei handelt es sich vermutlich um aktiv gemanagte Aktienfonds. Das Cluster mit Vola 10% hat auffällig viele fehlende Einträge. Es könnte sich um einen Infrastrukturfonds handeln.

Aufgabe 2.2: Verwenden Sie einen Algorithmus zur Anomalie-Detektion um auffällige Titel zu identifizieren. Werden Auffälligkeiten gefunden und sind diese vergleichbar zu den Anlageklassen aus Aufgabe 2.1? Welches Kontextwissen kann dem Algorithmus helfen, sodass dieser die gleichen Anlageklassen identifiziert?

Musterlösung: Wir verwenden einen Ansatz auf Basis des knn-Algorithmus. Dabei werden Finanzinstrumente identifiziert, die wenige vergleichbare Titel haben.

```
# KNN Distanz berechnen
k <- 20

erg = erg %>% mutate(ratio = log(Schlusskurs/100) / sd)
knn_result <- get.knn(erg, k = k)

# Durchschnittliche Distanz zum k-Nächsten Nachbarn
avg_dist <- rowMeans(knn_result$nn.dist)

# Plot mit Anomalie-Schwelle
threshold <- quantile(avg_dist, 0.95)  # oberste 5 % als Anomalie

erg %>% mutate(ratio = mean_return_log / sd,
               costcat = cut(cost,breaks = c(0.0,0.015,0.04)),
```

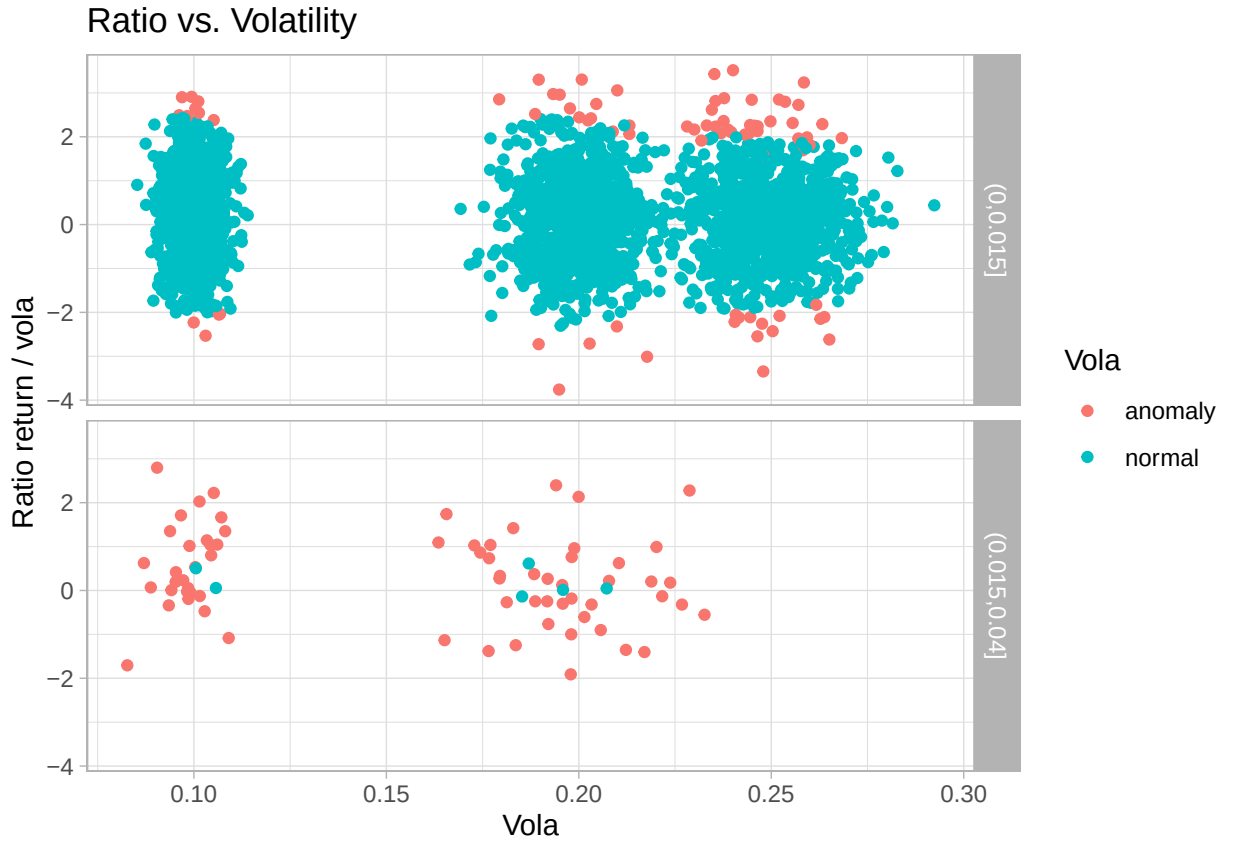
```
              anomalycat = ifelse(avg_dist > threshold,"anomaly","normal")) %>%
  ggplot(aes(x = sd,y = ratio,color = anomalycat)) +
  geom_point() +   labs(title = "Ratio vs. Volatility",
                        x = "Vola",
                        y = "Ratio return / vola",
                        color = "Vola") +
  theme_light() +
  facet_grid(rows = vars(costcat))
```



Es gelingt dem Algorithmus nicht die gleichen auffälligen Cluster zu identifizieren. Dies liegt vermutlich daran, dass die auffälligen Kosten als Prozentzahl skaliert sind. Eine Lösung könnte sein, diese zu skalieren um in der Bewertung stärker ins Gewicht zu fallen.

```
#variante mit höherer Gewichtung der kosten
k <- 10
erg_mod = erg %>% mutate(ratio = log(Schlusskurs/100) / sd,
                         cost = cost * 100)
knn_result <- get.knn(erg_mod, k = k)

## Durchschnittliche Distanz zum k-Nächsten Nachbarn
avg_dist <- rowMeans(knn_result$nn.dist)

# Plot mit Anomalie-Schwelle
threshold <- quantile(avg_dist, 0.95)  # oberste 5 % als Anomalie
erg %>% mutate(ratio = mean_return_log / sd,
               costcat = cut(cost,breaks = c(0.0,0.015,0.04)),
```

```
            anomalycat = ifelse(avg_dist > threshold,"anomaly","normal")) %>%
ggplot(aes(x = sd,y = ratio,color = anomalycat)) +
geom_point() +   labs(title = "Ratio vs. Volatility",
                      x = "Vola",
                      y = "Ratio return / vola",
                      color = "Vola") +
theme_light() +
facet_grid(rows = vars(costcat))
```



Es gelingt der einfachen Anomalie-Detection recht gut die Cluster zu separieren, nachdem alle relevanten Merkmale vergleichbar skaliert werden. Die Versicherung bietet insgesamt 5 Fondspaletten an. Da nur Beobachtungen von einem Jahr vorliegen, ist es nicht sinnvoll, jeden einzelnen Titel separat zu schätzen. Zufällig könnte insbesondere die Rendite in diesem Jahr nicht repräsentativ für die Gesamtentwicklung sein. Bei der Volatilität und den Kosten ist das längst nicht so ausgeprägt sodass die Schätzer hier deutlich robuster sind. Die historische Entwicklung der Kurse ist nur bedingt aussagekräftig für die künftige Entwicklung. Sie dient lediglich dazu, eine Schätzung der erwarteten Rendite und der Volatilität einer Anlagegruppe zu schätzen, die anschließend in einer Simulation verwendet werden kann. Die Simulation zeigt anschließend sehr schön, dass auch für Titel mit eigentlich guter erwarteter Rendite durch die Volatilität auch (sehr) negative Pfade möglich sind. Dies ist bei der Analyse der Zielgruppen zu beachten. Insbesondere bei den Zielgruppen mit höherer Stornoquote, ist es wichtig, dass die Rendite auch in der Simulation nicht zu stark negativ ist und möglichst zu allen Zeitpunkten positiv ist (kleiner Volatilität vorteilhaft). Dies könnte dazu führen, dass diese Zielgruppen von gewissen Titeln ausgeschlossen werden sollte, da ein negativer Kundennutzen vorliegen könnte.

Aufgabe 3

Simulieren Sie eine künftige Entwicklung des rein fondsgebundenen Produkts für alle identifizierten Anlageklassen auf Basis der für diese Anlageklasse geschätzten Rendite, Volatilität und Kosten (z.B. durch eine geometrisch Brownsche Bewegung). Im Rahmen des POG-Prozesses müssen Sie für Ihre vier Zielgruppen aus Aufgabe 1 einen angemessenen Kundennutzen nachweisen zu den Zeitpunkten

- t1 = Zeitpunkt zu dem 50% der Zielgruppe storniert haben
- t2 = Vertragsdauer des Produkts

Betrachten Sie zu diesen Zeitpunkten die Medianrendite der Anlageklassen und vergleichen Sie diese mit der Zielrendite von 2%. Welche Anlageklasse ist aus Sicht des angemessenen Kundennutzens für welche Zielgruppe kritisch zu hinterfragen? Verwenden Sie ein Produkt mit 20 Jahren Laufzeit. Nehmen Sie an, dass auch für das Neugeschäft das Stornoverhalten der Bestandsdaten anwendbar ist.

Musterlösung:

Wir simulieren das Produkt als geometrisch Brownsche Bewegung über 20 Jahre. Also Zeitpunkte werden x / y betrachtet.

```
result = erg %>% mutate(volcat = cut(sd,breaks = c(0,0.15,0.24,0.3)),
                        costcat = cut(cost,breaks = c(0.0,0.015,0.04))) %>%
  group_by(volcat,costcat) %>%
  summarise(mean_return = log(mean(Schlusskurs)/100),
            sd = mean(sd),
            cost = mean(cost),
            Anzahl =n(),
            ratio = mean_return / sd)
```

```
## `summarise()` has grouped output by 'volcat'. You can override using the
## `.groups` argument.
```

```
gbm_vec_with_cost <- function(nsim = 100,cost = 0,
                              t = 25, mu = 0, sigma = 0.1,
                              S0 = 100, dt = 1./255) {
  # matrix of random draws - one for each day for each simulation
  epsilon <- matrix(rnorm((t-1)*nsim), ncol = nsim, nrow = t-1)
  sigma_m <- sigma

  # get GBM and convert to price paths
  gbm <- exp((mu - sigma_m * sigma_m / 2) * dt +
               sigma_m * epsilon * sqrt(dt)) * (1-cost)^(dt)

  gbm <- apply(rbind(rep(S0,nsim), gbm), 2, cumprod)
  return(gbm)
  }

set.seed(10)
t1 = 9 ##ergibt sich aus etwa 45% (Zielgruppe 1/2)
t2 = 14 #ergibt sich aus etwa 70% (Zielgruppe 3/4)
t3 = 20

analyse_cluster = sapply(1:5, function(i) {
  gbm = gbm_vec_with_cost(
    nsim = 10000,
    cost = result$cost[i],
    t = 20 * 255,
```

```
    mu = result$mean_return[i],
    sigma = result$sd[i],
    S0 = 100000,
    dt = 1. / 255
  )
  returnst1 = log((gbm[t1 * 255, ] / 100000)) / t1
  returnst2 = log((gbm[t2 * 255, ] / 100000)) / t2
  returnst3 = log((gbm[t3 * 255, ] / 100000)) / t3

  return(c(median(returnst1), median(returnst2), median(returnst3)))
})
colnames(analyse_cluster) = c("ETF1", "Infrastruktur", "ETF2", "Aktiv", "ETF3")
rownames(analyse_cluster) = c("t1","t2","VD")
kable(analyse_cluster,
      caption = "Median Rendite Zielgruppencluster")
```

Table 3: Median Rendite Zielgruppencluster

	ETF1	Infrastruktur	ETF2	Aktiv	ETF3
t1	0.0238346	0.0295042	0.0263000	0.0032036	0.0196906
t2	0.0234744	0.0290246	0.0255851	0.0028739	0.0200696
VD	0.0226900	0.0292769	0.0263380	0.0033824	0.0197151

Die Anlageklasse 4 der aktiv gemanagten Fonds liegt in der Medianrendite deutlich unterhalb von 2%. Hier ist der Kundennutzen zu hinterfragen. Auch die Gruppe 5 mit hoher Volatilität ist knapp unterhalb von 2% Zielrendite.