modeling

April 15, 2025

1 Modeling

The goal is to predict the expected value of the total claim amount per exposure unit (year).

- Model the number of claims with a Poisson distribution, and the average claim amount per claim, with a Gamma distribution.

[1]:
```
import sys
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import PoissonRegressor, GammaRegressor
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import (
    FunctionTransformer,
    OneHotEncoder,
    StandardScaler,
    KBinsDiscretizer,
)

# Add parent directory to sys.path
parent_dir = Path().resolve().parent
sys.path.append(str(parent_dir))

from src.utils import replace_birthdate_with_age, load_data, plot_boxplots
from src.metrics import score_estimator
```

1.1 Data load and pre-processing

- load and preprocess data

[2]:
```
# load feature data
file_path = parent_dir / 'features.parquet'
df_feat = load_data(file_path)
```

```
# preprocess feature data
df_feat = df_feat[df_feat['Exposure'] > 0.2 ]
#df_feat = df_feat[df_feat['ClaimNb'] < 5]

df_feat = replace_birthdate_with_age(df_feat, 'BirthD',␣
↪reference_date='2023-01-01')
df_feat['VehGas'] = df_feat['VehGas'].fillna('G3')
df_feat["Exposure"] = df_feat["Exposure"].clip(0.1, 1)
df_feat["ClaimNb"] = df_feat["ClaimNb"].clip(upper=4)
df_feat["DriverAge"] = df_feat["DriverAge"].clip(19, 85)
df_feat['VehAge'] = df_feat['VehAge'].clip(0, 20)
df_feat['BonusMalus'] = df_feat['BonusMalus'].clip(0, 100)

# load target data
file_path = parent_dir / 'target.parquet'
df_target = load_data(file_path)

# preprocess target data
df_target = df_target.groupby('IDpol', as_index=False).agg({'ClaimAmount':␣
↪'sum'})
df_target['ClaimAmount'] = df_target['ClaimAmount'].clip(0, 100000)

# merge feature and target data
df_feat["IDpol"] = df_feat["IDpol"].astype(int)
df_feat.set_index("IDpol", inplace=True)
df = pd.merge(df_feat, df_target, on='IDpol', how='left')

#df = df[(df['IDpol'] > 4000000) & (df['IDpol'] < 5000000)]
#df = df[df['ClaimNb'] > 0]
```

```
DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 678013 entries, 0 to 678012
Data columns (total 12 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   IDpol       678013 non-null  float64
 1   ClaimNb     678013 non-null  float64
 2   Exposure    678013 non-null  float64
 3   Area        678013 non-null  object
 4   VehPower    678013 non-null  float64
 5   VehAge      678013 non-null  float64
 6   BonusMalus  678013 non-null  float64
 7   VehBrand    678013 non-null  object
 8   VehGas      644112 non-null  object
 9   Density     678013 non-null  float64
```

```
 10  Region      678013 non-null  object
 11  BirthD      678013 non-null  object
dtypes: float64(7), object(5)
memory usage: 62.1+ MB
None


##########################################################################
First 5 Rows:
   IDpol  ClaimNb  Exposure Area  VehPower  VehAge  BonusMalus VehBrand  \
0    1.0      1.0      0.10  'D'       5.0     0.0        50.0    'B12'
1    3.0      1.0      0.77  'D'       5.0     0.0        50.0    'B12'
2    5.0      1.0      0.75  'B'       6.0     2.0        50.0    'B12'
3   10.0      1.0      0.09  'B'       7.0     0.0        50.0    'B12'
4   11.0      1.0      0.84  'B'       7.0     0.0        50.0    'B12'

    VehGas  Density Region      BirthD
0     None   1217.0  'R82'  1967-05-08
1  Regular   1217.0  'R82'  1967-12-28
2   Diesel     54.0  'R22'  1970-08-13
3   Diesel     76.0  'R72'  1976-12-05
4   Diesel     76.0  'R72'  1976-02-29


##########################################################################
Summary Statistics:
              IDpol        ClaimNb       Exposure       VehPower  \
count  6.780130e+05  678013.000000  678013.000000  678013.000000
mean   2.621857e+06       0.053247       0.528750       6.454631
std    1.641783e+06       0.240117       0.364442       2.050906
min    1.000000e+00       0.000000       0.002732       4.000000
25%    1.157951e+06       0.000000       0.180000       5.000000
50%    2.272152e+06       0.000000       0.490000       6.000000
75%    4.046274e+06       0.000000       0.990000       7.000000
max    6.114330e+06      16.000000       2.010000      15.000000

              VehAge     BonusMalus        Density
count  678013.000000  678013.000000  678013.000000
mean        7.044265      59.761502    1792.422405
std         5.666232      15.636658    3958.646564
min         0.000000      50.000000       1.000000
25%         2.000000      50.000000      92.000000
50%         6.000000      50.000000     393.000000
75%        11.000000      64.000000    1658.000000
max       100.000000     230.000000   27000.000000


##########################################################################
```

```
Unique Values Per Column:
IDpol         678013
ClaimNb           11
Exposure         181
Area               6
VehPower          12
VehAge            78
BonusMalus       115
VehBrand          11
VehGas             2
Density         1607
Region            22
BirthD         25775
dtype: int64


##########################################################################
Total Missing Values in DataFrame:
IDpol             0
ClaimNb           0
Exposure          0
Area              0
VehPower          0
VehAge            0
BonusMalus        0
VehBrand          0
VehGas        33901
Density           0
Region            0
BirthD            0
dtype: int64


##########################################################################
DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26639 entries, 0 to 26638
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   IDpol        26639 non-null  float64
 1   ClaimAmount  26639 non-null  float64
dtypes: float64(2)
memory usage: 416.4 KB
None


##########################################################################
```

```
First 5 Rows:
       IDpol  ClaimAmount
0     1552.0       995.20
1  1010996.0      1128.12
2  4024277.0      1851.11
3  4007252.0      1204.00
4  4046424.0      1204.00


###########################################################################
Summary Statistics:
              IDpol   ClaimAmount
count  2.663900e+04  2.663900e+04
mean   2.279864e+06  2.278536e+03
std    1.577202e+06  2.929748e+04
min    1.390000e+02  1.000000e+00
25%    1.087642e+06  6.868100e+02
50%    2.137413e+06  1.172000e+03
75%    3.180162e+06  1.228080e+03
max    6.113971e+06  4.075401e+06


###########################################################################
Unique Values Per Column:
IDpol          24950
ClaimAmount    12369
dtype: int64


###########################################################################
Total Missing Values in DataFrame:
IDpol          0
ClaimAmount    0
dtype: int64


###########################################################################
```

1.2 feature and target definitions

- transform features

[3]:
```
# log-transform the target variable
log_scale_transformer = make_pipeline(
    FunctionTransformer(func=np.log), StandardScaler()
)

# create a column transformer for preprocessing
```

```
column_trans = ColumnTransformer(
    [
        ("onehot_categorical", OneHotEncoder(), ["VehBrand", "VehPower",␣
 ↪"VehGas", "Region", "Area"]),
        ("standardized_numeric", StandardScaler(), ["VehAge", "DriverAge",␣
 ↪"BonusMalus"]),
        ("log_scaled_numeric", log_scale_transformer, ["Density"]),
    ],
    remainder="drop",
)
transmored_features = column_trans.fit_transform(df)
########################################################

df["Claim_freq"] = df["ClaimNb"] / df["Exposure"]
df["Avg_claim_amount"] = df["ClaimAmount"] / np.fmax(df["ClaimNb"], 1)
```

1.3 Model Claim frequency model

- The number of claims (`ClaimNb`) is a positive integer (0 included).
- discrete events occurring in a given time interval (`Exposure`) independent from each other.
- model the Claim frequency `ClaimNb / Exposure` and use `Exposure` as offset.

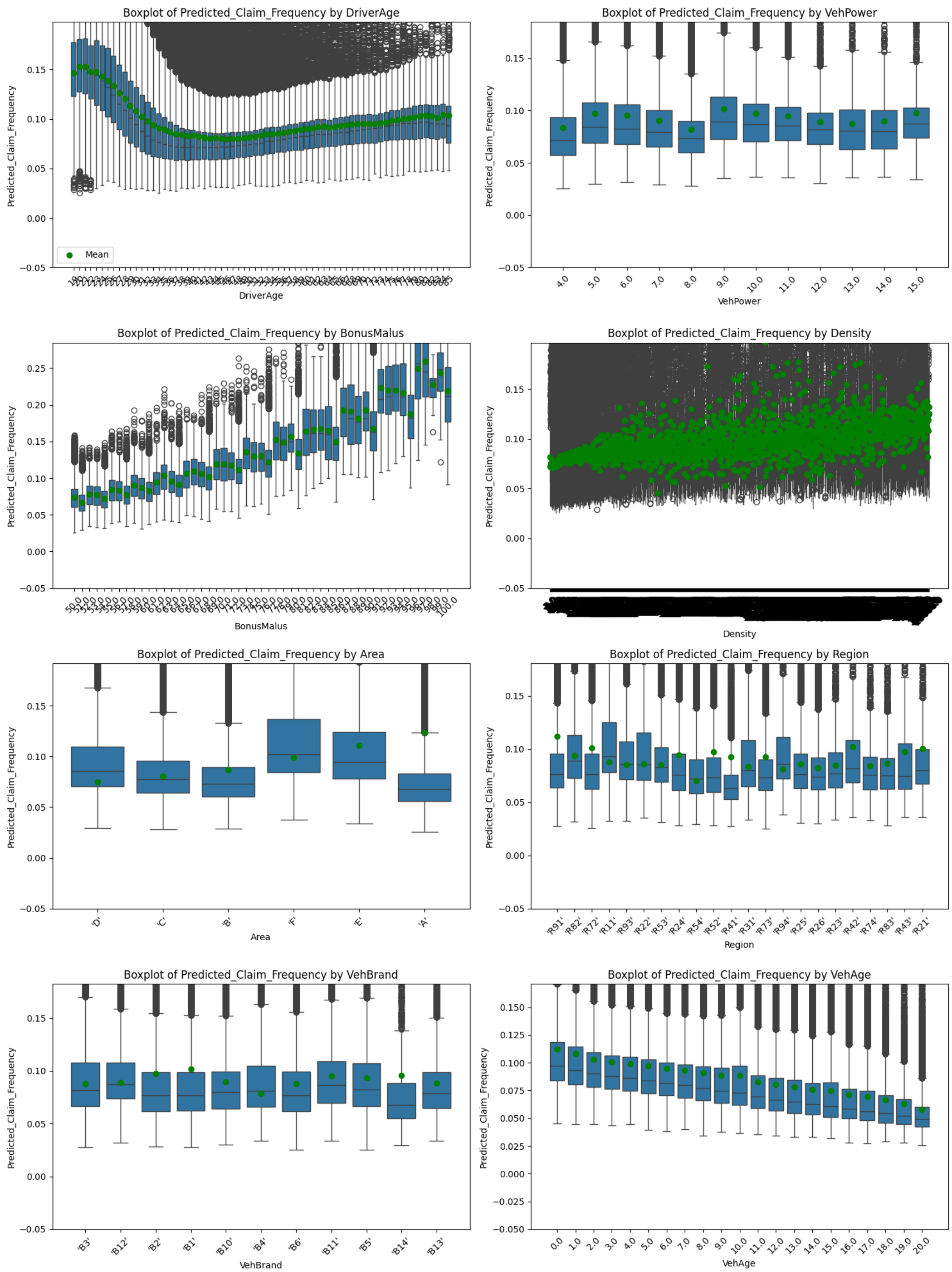
[4]:
```
df_train, df_test, X_train, X_test = train_test_split(df, transmored_features,␣
 ↪random_state=0)

# Fit a Poisson regression model for claim frequency
glm_freq = PoissonRegressor(alpha=1e-3, max_iter=1000)
glm_freq.fit(X_train, df_train["Claim_freq"],␣
 ↪sample_weight=df_train["Exposure"])

scores = score_estimator(glm_freq, X_train, X_test, df_train, df_test,␣
 ↪target="Claim_freq", weights="Exposure")
print("Evaluation of PoissonRegressor on target Claim_freq")
print(scores)
```

```
Evaluation of PoissonRegressor on target Claim_freq
subset               train    test
metric
R-squared score     0.0111  0.0092
mean abs. error     0.1646  0.1647
mean squared error  0.1540  0.1532
```

- visualize the predictes values

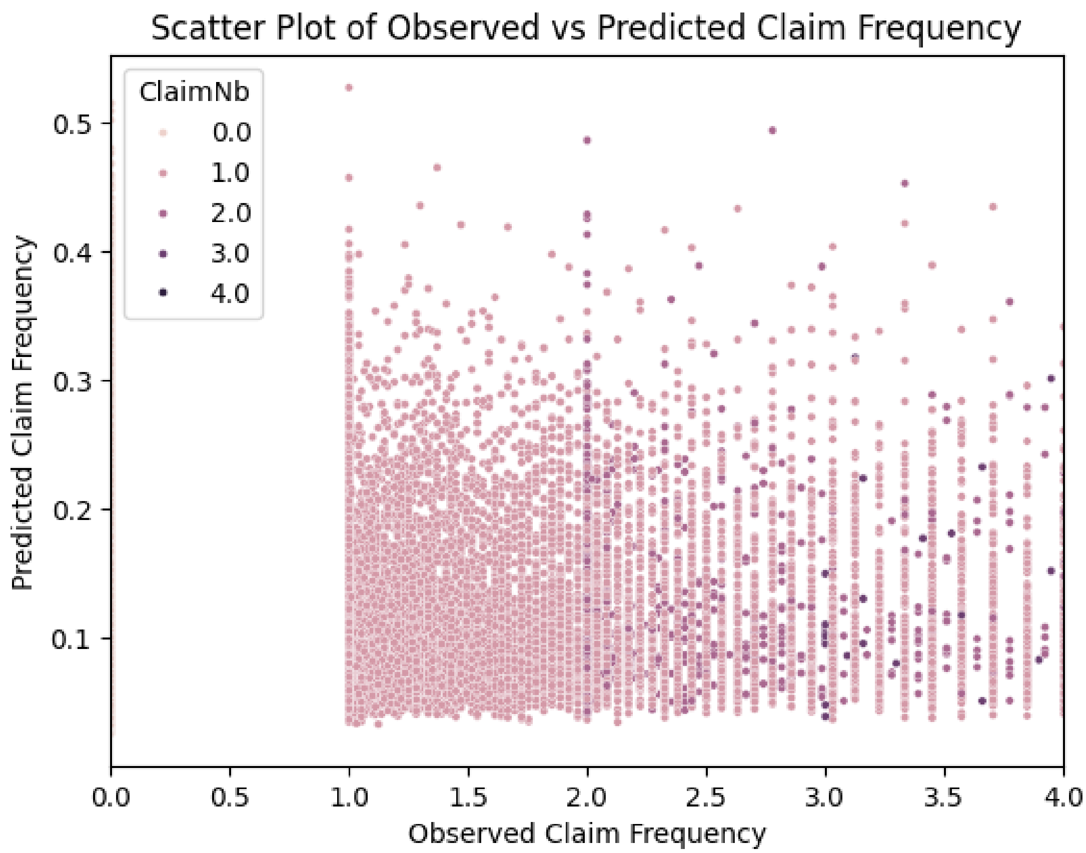
[5]:
```
df_train["Predicted_Claim_Frequency"] = glm_freq.predict(X_train)
#print(df_train.head(10))
```

```python
plot_boxplots(['DriverAge', 'VehPower', 'BonusMalus', 'Density', 'Area',
    'Region', 'VehBrand', 'VehAge'], 'Predicted_Claim_Frequency', df_train)
```



- visulaize the scatter plot of observed and

[6]:
```
sns.scatterplot(
    x=df_train['Claim_freq'],
    y=df_train['Predicted_Claim_Frequency'],
    hue=df_train['ClaimNb'],
    s=10  # Set marker size to be smaller
)
plt.xlabel('Observed Claim Frequency')
plt.ylabel('Predicted Claim Frequency')
plt.title('Scatter Plot of Observed vs Predicted Claim Frequency')
plt.xlim(0, 4)
plt.show()
```



1.4 Model average claim amount (Gamma distribution)

- filter out records with 0 claim amount
- use `ClaimNb` as `sample_weight`

[7]:

```python
mask_train = df_train["ClaimAmount"] > 0
mask_test = df_test["ClaimAmount"] > 0

glm_amount = GammaRegressor(alpha=10.0, max_iter=10000)

glm_amount.fit(X_train[mask_train.values],
               df_train.loc[mask_train, "Avg_claim_amount"],
               sample_weight=df_train.loc[mask_train, "ClaimNb"],
)

scores = score_estimator(
    glm_amount,
    X_train[mask_train.values],
    X_test[mask_test.values],
    df_train[mask_train],
    df_test[mask_test],
    target="Avg_claim_amount",
    weights="ClaimNb",
)
print("Evaluation of GammaRegressor on target AvgClaimAmount")
print(scores)
```

```
Evaluation of GammaRegressor on target AvgClaimAmount
subset                     train          test
metric
R-squared score     1.000000e-04 -1.300000e-03
mean abs. error     1.474881e+03  1.327977e+03
mean squared error  2.590239e+07  1.753600e+07
```

[8]:

```python
print(
    "actual average claim Amount :           %.2f"
    % df_train["Avg_claim_amount"][df_train["Avg_claim_amount"] > 0].mean()
)
print(
    "Predicted average claim Amount:         %.2f"
    % glm_amount.predict(X_train).mean()
)
```

```
actual average claim Amount :           1774.01
Predicted average claim Amount:         1799.16
```