992667_Individual_BIG_DATA

April 3, 2024

1 Big Data Coursework - Road Safety Data 2022

Individual Assignment

ASTON UNIVERSITY

Candidate Number - 992667

This notebook serves as a comprehensive Feature Engineering and Predictive Modeling Report based on the 2022 UK Road Safety dataset.

I will continue to refine the dataset through cleaning and transformation processes. Subsequently, I'll develop multiple models to accurately predict accident severity for insurance applicants, utilizing selected independent variables. These steps aim to enhance the car insurance company's risk assessment and premium customization capabilities.

Table of contents

1. Importing Libraries and Preparing Environment
2. Business Context
3. Data loading
4. Data Preparation
5. Data Transformation
6. Encoding
7. Feature Engineering
8. Model developement
9. Evaluation of the models
10. Conclusion Possible future improvements
11. References

2 Importing Libraries and Preparing Environment

[1]:
```
# Importing the requiiste Library

#Basic Libraries
import re
import time
import warnings
import pandas as pd
import numpy as np
```

```
#Plot Libraries
import matplotlib.pyplot as plt
import seaborn as sns

#Data Pre-processing Libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, OrdinalEncoder,␣
↪LabelEncoder
from category_encoders import TargetEncoder
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import FunctionTransformer
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import VarianceThreshold
```

3 Business Context

A car insurance company aims to enhance its risk assessment process by more accurately predicting the severity of accidents involving new applicants. This initiative seeks to refine premium setting, ensuring it aligns more closely with individual risk profiles. The motivation stems from a need to mitigate financial losses associated with high-severity accidents by adjusting premiums based on predictive insights into an applicant's potential risk factors.

3.0.1 Objective

The primary objective is to develop a predictive model capable of forecasting accident severity levels—categorized into 'Slight', 'Serious', and 'Fatal'—based on a comprehensive analysis of driver demographics, vehicle characteristics, and historical accident data. Insights derived from this model will inform tailored insurance premium strategies, aiming to balance risk with profitability and enhance customer segmentation.

3.0.2 Modeling Task

The task is framed as a multi-class classification problem, where the model will be trained on historical accident data, including:

- Continuous Variables: Age of driver, engine capacity (cc), age of vehicle.
- Categorical Variables: Vehicle type, towing and articulation, journey purpose, sex of driver, vehicle's drive orientation, propulsion type, vehicle make and model, alongside ordinal variables like age band of driver, socio-economic decile, and urban-rural classification of driver's residence. (identified in the previous notebook)

The outcome will directly influence the company's approach to setting premiums by providing a nuanced understanding of risk associated with new applicants, thereby facilitating more informed and equitable insurance pricing strategies.

4 Data loading

[2]:
```
# Since I wanted to do further indepth cleaning I am loading the train_set and␣
↪test_set created in the previous notebook.

train_set = pd.read_csv('train_set.csv', index_col=0)
test_set = pd.read_csv('test_set.csv', index_col=0)

# Data Guide
guide = pd.
↪read_excel("dft-road-casualty-statistics-road-safety-open-dataset-data-guide-2023.
↪xlsx")
```

[3]:
```
train_set.head()
```

[3]:
```
        accident_index  accident_year accident_reference  vehicle_reference  \
19397    2022010383195           2022          010383195                  1
21539    2022010386334           2022          010386334                  3
149854   2022451257055           2022          451257055                  2
176779   2022547895122           2022          547895122                  1
60657    2022070774180           2022          070774180                  3

        vehicle_type  towing_and_articulation  journey_purpose_of_driver  \
19397            9.0                      0.0                          6
21539           19.0                      0.0                          1
149854           9.0                      0.0                          6
176779           9.0                      0.0                          2
60657            9.0                      0.0                          1

        sex_of_driver  vehicle_left_hand_drive  age_of_driver  \
19397               3                        1           38.0
21539               1                        1           29.0
149854              3                        1           38.0
176779              1                        1           35.0
60657               1                        1           30.0

        age_band_of_driver  engine_capacity_cc  propulsion_code  \
19397                  7.0              1242.0              1.0
21539                  6.0              1598.0              2.0
149854                 7.0              1242.0              1.0
176779                 6.0              1598.0              2.0
60657                  6.0              1968.0              2.0

        age_of_vehicle generic_make_model  driver_imd_decile  \
19397              6.0            FORD KA                5.0
21539              1.0         FIAT DOBLO                7.0
149854             5.0           FIAT 500                5.0
```

```
176779            10.0    VOLKSWAGEN GOLF               8.0
60657              1.0            AUDI A3               3.0

        driver_home_area_type  accident_severity
19397                     1.0                  3
21539                     1.0                  3
149854                    1.0                  3
176779                    2.0                  2
60657                     1.0                  2
```

[4]:
```
train_set.dtypes
```

[4]:
```
accident_index                object
accident_year                  int64
accident_reference            object
vehicle_reference              int64
vehicle_type                 float64
towing_and_articulation      float64
journey_purpose_of_driver      int64
sex_of_driver                  int64
vehicle_left_hand_drive        int64
age_of_driver                float64
age_band_of_driver           float64
engine_capacity_cc           float64
propulsion_code              float64
age_of_vehicle               float64
generic_make_model            object
driver_imd_decile            float64
driver_home_area_type        float64
accident_severity              int64
dtype: object
```

5 Data Preparation

[5]:
```
# Decoding the data with the help of data guide.

def decode_dataframe_fields(df, encoded_fields):
    for ef in encoded_fields:
        if ef in df.columns:  # Ensure the column exists in the DataFrame
            # Create a mapping dictionary for the current field
            ef_guide = guide[guide['field name'] == ef]
            code_to_label_mapping = ef_guide.set_index('code/format')['label'].
 ↪to_dict()

            # Use map to apply the mapping
            df[ef] = df[ef].map(code_to_label_mapping).fillna(df[ef])
```

```
        else:
            print(f"Column {ef} not found in DataFrame.")
```

[6]:
```
decoded_set = ['vehicle_type', 'towing_and_articulation',␣
↪'journey_purpose_of_driver', 'sex_of_driver',
               'vehicle_left_hand_drive', 'age_of_driver', 'age_band_of_driver',␣
↪'engine_capacity_cc',
               'propulsion_code', 'age_of_vehicle', 'generic_make_model',␣
↪'driver_imd_decile', 'driver_home_area_type',
               'accident_severity']

decode_dataframe_fields(train_set, decoded_set)
decode_dataframe_fields(test_set, decoded_set)
```

[7]:
```
train_set.head()
```

[7]:
```
       accident_index  accident_year accident_reference  vehicle_reference  \
19397   2022010383195           2022          010383195                  1
21539   2022010386334           2022          010386334                  3
149854  2022451257055           2022          451257055                  2
176779  2022547895122           2022          547895122                  1
60657   2022070774180           2022          070774180                  3

                             vehicle_type towing_and_articulation  \
19397                                 Car     No tow/articulation
21539   Van / Goods 3.5 tonnes mgw or under     No tow/articulation
149854                                Car     No tow/articulation
176779                                Car     No tow/articulation
60657                                 Car     No tow/articulation

       journey_purpose_of_driver sex_of_driver vehicle_left_hand_drive  \
19397                  Not known     Not known                      No
21539     Journey as part of work          Male                      No
149854                 Not known     Not known                      No
176779     Commuting to/from work          Male                      No
60657     Journey as part of work          Male                      No

        age_of_driver age_band_of_driver  engine_capacity_cc propulsion_code  \
19397            38.0            36 - 45              1242.0          Petrol
21539            29.0            26 - 35              1598.0       Heavy oil
149854           38.0            36 - 45              1242.0          Petrol
176779           35.0            26 - 35              1598.0       Heavy oil
60657            30.0            26 - 35              1968.0       Heavy oil

        age_of_vehicle generic_make_model     driver_imd_decile  \
19397              6.0            FORD KA  More deprived 40-50%
21539              1.0         FIAT DOBLO  Less deprived 30-40%
```

```
149854               5.0          FIAT 500  More deprived 40-50%
176779              10.0   VOLKSWAGEN GOLF  Less deprived 20-30%
60657                1.0           AUDI A3  More deprived 20-30%

       driver_home_area_type accident_severity
19397             Urban area            Slight
21539             Urban area            Slight
149854            Urban area            Slight
176779            Small town           Serious
60657             Urban area           Serious
```

[8]:
```python
train_set.shape
```

[8]:
```
(7200, 18)
```

vehicle_type

[9]:
```python
train_set.value_counts('vehicle_type')
```

[9]:
```
vehicle_type
Car                                     4904
Pedal cycle                              610
Van / Goods 3.5 tonnes mgw or under      458
Motorcycle 125cc and under               355
Motorcycle over 500cc                    185
Bus or coach (17 or more pass seats)     117
Goods 7.5 tonnes mgw and over            115
Other vehicle                            109
Taxi/Private hire car                     97
Motorcycle over 125cc and up to 500cc     93
Motorcycle 50cc and under                 33
Goods vehicle - unknown weight            28
Goods over 3.5t. and under 7.5t           27
Motorcycle - unknown cc                   15
Minibus (8 - 16 passenger seats)          12
Mobility scooter                          12
Electric motorcycle                       12
Agricultural vehicle                      11
Ridden horse                               5
Unknown vehicle type (self rep only)       2
Name: count, dtype: int64
```

We can see that there are certain values like `Other vehicle`, `Motorcycle - unknown cc`, `Unknown vehicle type (self rep only)` which are uninformative for a car insurance company, as accurately classifying the driver's and vehicle's risk profile requires precise information about the vehicle type. Also insurance provider won't provide insurance for a `Ridden horse`.

[10]:
```
vehicle_type_percentages = (train_set['vehicle_type'].value_counts() / train_set.
↪shape[0]) * 100
vehicle_type_percentages
```

[10]:
```
vehicle_type
Car                                      68.111111
Pedal cycle                               8.472222
Van / Goods 3.5 tonnes mgw or under       6.361111
Motorcycle 125cc and under                4.930556
Motorcycle over 500cc                     2.569444
Bus or coach (17 or more pass seats)      1.625000
Goods 7.5 tonnes mgw and over             1.597222
Other vehicle                             1.513889
Taxi/Private hire car                     1.347222
Motorcycle over 125cc and up to 500cc     1.291667
Motorcycle 50cc and under                 0.458333
Goods vehicle - unknown weight            0.388889
Goods over 3.5t. and under 7.5t           0.375000
Motorcycle - unknown cc                   0.208333
Electric motorcycle                       0.166667
Minibus (8 - 16 passenger seats)          0.166667
Mobility scooter                          0.166667
Agricultural vehicle                      0.152778
Ridden horse                              0.069444
Unknown vehicle type (self rep only)      0.027778
Name: count, dtype: float64
```

While the low percentages of certain vehicle types like `Other vehicle`, `Motorcycle - unknown cc`, and `Unknown vehicle type (self rep only)` `Goods vehicle - unknown weight` might seem insignificant, retaining these uninformative values could potentially introduce noise and adversely impact the accuracy of risk profile classification for a car insurance company. Since while applying for insurance the apllicant should have the particular vehicle type information.

[11]:
```
uninformative_vehicle_types = ['Other vehicle', 'Motorcycle - unknown cc',␣
↪'Unknown vehicle type (self rep only)',
                               'Ridden horse', 'Goods vehicle - unknown weight']
```

[12]:
```
train_set['vehicle_type'].isnull().mean() * 100
```

[12]: 0.0

[13]:
```
train_set.loc[train_set['vehicle_type'].isin(uninformative_vehicle_types),␣
↪'vehicle_type'] = np.nan
```

[14]:
```
train_set['vehicle_type'].isnull().mean() * 100
```

[14]: 2.2083333333333335

[15]:
```
# Doing the same for test_set
test_set.loc[test_set['vehicle_type'].isin(uninformative_vehicle_types),␣
 ↪'vehicle_type'] = np.nan
```

[16]:
```
test_set['vehicle_type'].isnull().mean() * 100
```

[16]:
```
2.2777777777777777
```

This null values will be imputed in subsequent process.

towing_and_articulation

[17]:
```
train_set.value_counts('towing_and_articulation')
```

[17]:
```
towing_and_articulation
No tow/articulation        6946
unknown (self reported)     169
Articulated vehicle          46
Single trailer               23
Other tow                    10
Caravan                       6
Name: count, dtype: int64
```

[18]:
```
towing_and_articulation_percentages = (train_set['towing_and_articulation'].
 ↪value_counts() / train_set.shape[0]) * 100
towing_and_articulation_percentages
```

[18]:
```
towing_and_articulation
No tow/articulation        96.472222
unknown (self reported)     2.347222
Articulated vehicle         0.638889
Single trailer              0.319444
Other tow                   0.138889
Caravan                     0.083333
Name: count, dtype: float64
```

[19]:
```
uninformative_towing_and_articulation = ['unknown (self reported)', 'Other tow']
```

[20]:
```
train_set['towing_and_articulation'].isnull().mean() * 100
```

[20]:
```
0.0
```

[21]:
```
train_set.loc[train_set['towing_and_articulation'].
 ↪isin(uninformative_towing_and_articulation), 'towing_and_articulation'] = np.
 ↪nan
```

[22]:
```
train_set['towing_and_articulation'].isnull().mean() * 100
```

[22]:
```
2.486111111111111
```

[23]:
```
# Doing the same for test_set
test_set.loc[test_set['towing_and_articulation'].
↪isin(uninformative_towing_and_articulation), 'towing_and_articulation'] = np.
↪nan
```

[24]:
```
test_set['towing_and_articulation'].isnull().mean() * 100
```

[24]:
```
2.111111111111111
```

journey_purpose_of_driver

[25]:
```
train_set['journey_purpose_of_driver'].isnull().mean() * 100
```

[25]:
```
0.0
```

[26]:
```
train_set.value_counts('journey_purpose_of_driver')
```

[26]:
```
journey_purpose_of_driver
Not known                         4255
Other                             1134
Journey as part of work            978
Commuting to/from work             722
Taking pupil to/from school         75
Pupil riding to/from school         36
Name: count, dtype: int64
```

Combining `Not known` and `Other` into `Leisure`, `Commuting to/from work` and `Journey as part of work` into `Work related travel`, `Taking pupil to/from school` and `Pupil riding to/from school` into `School related travel` , seems the most straightforward and informative approach.

[27]:
```
# Creating a dictionary to map the old categories to the new ones
category_mapping = {
    'Not known': 'Leisure',
    'Other': 'Leisure',
    'Commuting to/from work': 'Work related travel',
    'Journey as part of work': 'Work related travel',
    'Taking pupil to/from school': 'School related travel',
    'Pupil riding to/from school': 'School related travel'
}

# Replacing the old categories with the new ones
train_set['journey_purpose_of_driver'] = train_set['journey_purpose_of_driver'].
↪replace(category_mapping)
```

[28]:
```
train_set.value_counts('journey_purpose_of_driver')
```

[28]:
```
journey_purpose_of_driver
Leisure                   5389
Work related travel       1700
```

```
School related travel     111
Name: count, dtype: int64
```

[29]:
```
# Doing same for test_set
test_set['journey_purpose_of_driver'] = test_set['journey_purpose_of_driver'].
↪replace(category_mapping)
```

vehicle_left_hand_drive

[30]:
```
train_set.value_counts('vehicle_left_hand_drive')
```

[30]:
```
vehicle_left_hand_drive
No         6775
Unknown     359
Yes          66
Name: count, dtype: int64
```

The vehicle has to be either left hand drive or right hand drive it can't be unknown.

[31]:
```
uninformative_vehicle_left_hand_drive = ['Unknown']
```

[32]:
```
vehicle_left_hand_drive_percentages = (train_set['vehicle_left_hand_drive'].
↪value_counts() / train_set.shape[0]) * 100
vehicle_left_hand_drive_percentages
```

[32]:
```
vehicle_left_hand_drive
No         94.097222
Unknown     4.986111
Yes         0.916667
Name: count, dtype: float64
```

[33]:
```
train_set.loc[train_set['vehicle_left_hand_drive'].
↪isin(uninformative_vehicle_left_hand_drive), 'vehicle_left_hand_drive'] = np.
↪nan
```

[34]:
```
train_set['vehicle_left_hand_drive'].isnull().mean() * 100
```

[34]:
```
4.986111111111112
```

[35]:
```
# Doing same for test set
test_set.loc[test_set['vehicle_left_hand_drive'].
↪isin(uninformative_vehicle_left_hand_drive), 'vehicle_left_hand_drive'] = np.
↪nan
```

[36]:
```
train_set['vehicle_left_hand_drive'].isnull().mean() * 100
```

[36]:
```
4.986111111111112
```

[37]:
```
train_set.shape
```

[37]: (7200, 18)

`age_band_of_driver` Since the objective is to provide tailorised risk premiums for risk the applicant inhibits according to their sepcific age, `age_band_of_driver` is not needed.

[38]:
```
# Droppping age_band_of_driver
train_set.drop('age_band_of_driver', axis=1, inplace=True)
test_set.drop('age_band_of_driver', axis=1, inplace=True)
```

[39]:
```
print(train_set.shape)
print(test_set.shape)
```

```
(7200, 17)
(1800, 17)
```

`propulsion_code`

[40]:
```
train_set.value_counts('propulsion_code')
```

[40]:
```
propulsion_code
Petrol             4572
Heavy oil          2254
Hybrid electric     260
Electric             92
Electric diesel      18
Gas                   2
Gas/Bi-fuel           2
Name: count, dtype: int64
```

No need to modify `propulsion_code`

`driver_imd_decile`

[41]:
```
train_set.value_counts('driver_imd_decile')
```

[41]:
```
driver_imd_decile
More deprived 40-50%    2089
More deprived 10-20%     719
More deprived 20-30%     677
More deprived 30-40%     613
Most deprived 10%        613
Less deprived 40-50%     589
Less deprived 30-40%     537
Less deprived 20-30%     517
Less deprived 10-20%     480
Least deprived 10%       366
Name: count, dtype: int64
```

No need to modify `driver_imd_decile`

`driver_home_area_type`

[42]:
```
train_set.value_counts('driver_home_area_type')
```

[42]:
```
driver_home_area_type
Urban area    6023
Rural          702
Small town     475
Name: count, dtype: int64
```

No need to modify `driver_home_area_type`

6 Data transformation

Handling null values

[43]:
```
from sklearn.impute import SimpleImputer

cat_cols = ['vehicle_type', 'towing_and_articulation',␣
 ↪'journey_purpose_of_driver', 'vehicle_left_hand_drive']

cat_imputer = SimpleImputer(strategy='most_frequent')

train_set[cat_cols] = cat_imputer.fit_transform(train_set[cat_cols])
test_set[cat_cols] = cat_imputer.transform(test_set[cat_cols])
```

[44]:
```
train_set[['vehicle_type', 'towing_and_articulation',␣
 ↪'journey_purpose_of_driver', 'vehicle_left_hand_drive']].isnull().mean() * 100
```

[44]:
```
vehicle_type                 0.0
towing_and_articulation      0.0
journey_purpose_of_driver    0.0
vehicle_left_hand_drive      0.0
dtype: float64
```

[45]:
```
test_set[['vehicle_type', 'towing_and_articulation',␣
 ↪'journey_purpose_of_driver', 'vehicle_left_hand_drive']].isnull().mean() * 100
```

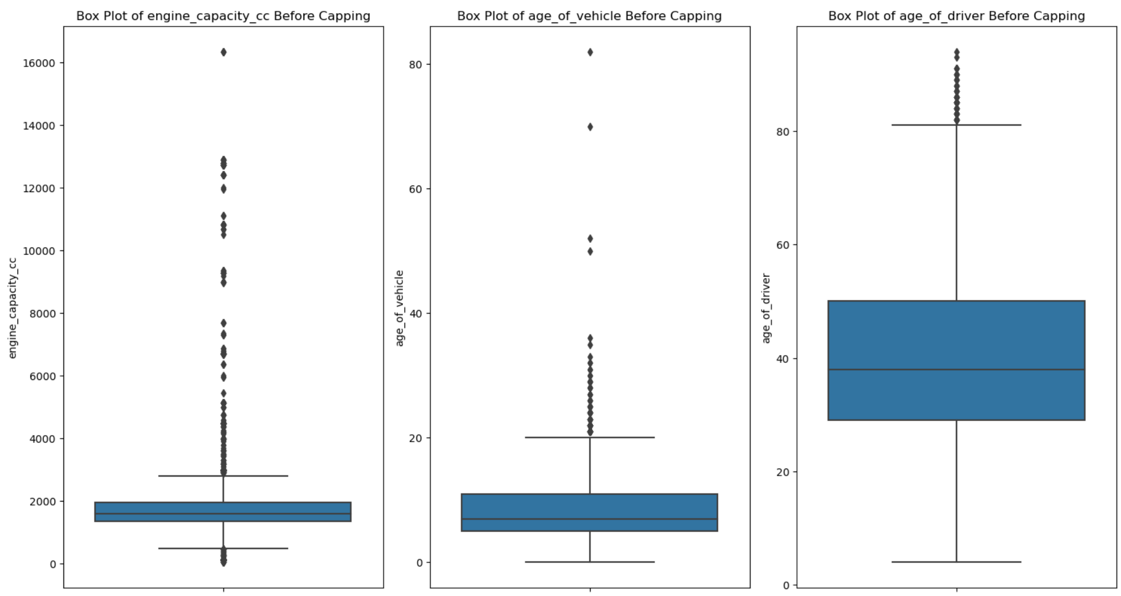
[45]:
```
vehicle_type                 0.0
towing_and_articulation      0.0
journey_purpose_of_driver    0.0
vehicle_left_hand_drive      0.0
dtype: float64
```

Null values have been imputed succesfully from both train_set and test_set.

`engine_capacity_cc`, `age_of_vehicle`, `age_of_driver` As we observed outliers in the previous notebook in this section I am going to handle them according to the business needs.

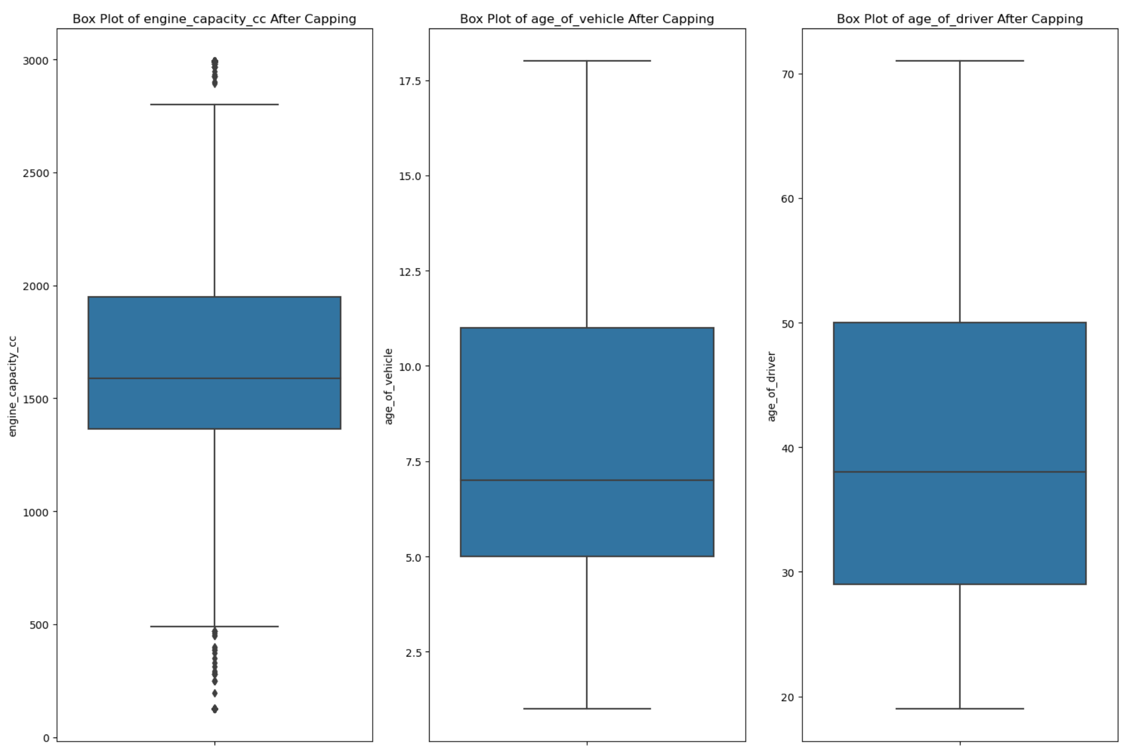
[46]:
```

```

```
features_to_visualize = ['engine_capacity_cc', 'age_of_vehicle', 'age_of_driver'␣
↪]

# Visualizing the selected features before capping
plt.figure(figsize=(15, 8))

for i, feature in enumerate(features_to_visualize, 1):
    plt.subplot(1, 3, i)
    sns.boxplot(y=train_set[feature])
    plt.title(f'Box Plot of {feature} Before Capping')

plt.tight_layout()
plt.show()
```



[47]:
```
def cap_outliers(series, lower_percentile=0.05, upper_percentile=0.95):
    """
    Caps the outliers in a pandas series based on the specified lower and upper␣
↪percentiles.
    """
    lower_limit = series.quantile(lower_percentile)
    upper_limit = series.quantile(upper_percentile)
    return series.clip(lower=lower_limit, upper=upper_limit)

# Applying capping to the selected features
train_set['engine_capacity_cc'] = cap_outliers(train_set['engine_capacity_cc'],␣
↪0.05, 0.95)
```

```python
train_set['age_of_vehicle'] = cap_outliers(train_set['age_of_vehicle'], 0.05, 0.
    ↪95)
train_set['age_of_driver'] = cap_outliers(train_set['age_of_driver'], 0.05, 0.95)

# Features to visualize after capping
features_to_visualize = ['engine_capacity_cc', 'age_of_vehicle', 'age_of_driver'␣
    ↪]

# Visualizing the features after capping to confirm the adjustment
plt.figure(figsize=(15, 10))

for i, feature in enumerate(features_to_visualize, 1):
    plt.subplot(1, 3, i)
    sns.boxplot(y=train_set[feature])
    plt.title(f'Box Plot of {feature} After Capping')

plt.tight_layout()
plt.show()
```



The box plots above visualize the `engine_capacity_cc`, `age_of_vehicle`, and `age_of_driver` features after applying the capping based on the **5th and 95th percentiles**. This adjustment effectively limits the influence of extreme values on these features. As a result, the range of values in each box plot is more constrained, and outliers beyond the specified percentiles have been adjusted to the nearest threshold within the capped range.

This method helps ensure that our model is not unduly influenced by extreme cases while still retaining the overall structure and variability of the data. It's a balanced approach that prepares the dataset for the development of a predictive model, aimed at accurately assessing the risk profiles of new insurance applicants.

[48]:
```
train_set.shape
```

[48]:
```
(7200, 17)
```

[49]:
```
# Doing same for test_set
test_set['engine_capacity_cc'] = cap_outliers(test_set['engine_capacity_cc'], 0.
 ↪05, 0.95)
test_set['age_of_vehicle'] = cap_outliers(test_set['age_of_vehicle'], 0.05, 0.95)
test_set['age_of_driver'] = cap_outliers(test_set['age_of_driver'], 0.05, 0.95)
```

[50]:
```
# Dropping the identifiers
train_set = train_set.drop(['accident_index', 'accident_reference',␣
 ↪'accident_year', 'vehicle_reference'], axis=1)
test_set = test_set.drop(['accident_index', 'accident_reference',␣
 ↪'accident_year', 'vehicle_reference'], axis=1)
```

7 Encoding

[51]:
```
train_set.dtypes
```

[51]:
```
vehicle_type                  object
towing_and_articulation       object
journey_purpose_of_driver     object
sex_of_driver                 object
vehicle_left_hand_drive       object
age_of_driver                float64
engine_capacity_cc           float64
propulsion_code               object
age_of_vehicle               float64
generic_make_model            object
driver_imd_decile             object
driver_home_area_type         object
accident_severity             object
dtype: object
```

[52]:
```
one_hot_encoder = OneHotEncoder(drop="first", sparse_output=False)

# categorical columns to transform
cat_cols = ["vehicle_type", "towing_and_articulation",␣
 ↪"journey_purpose_of_driver",
```

```python
            "sex_of_driver", "vehicle_left_hand_drive", "propulsion_code",
            "driver_home_area_type"]

# fitting encoder and transforming the **trainset**
cat_vals = train_set[cat_cols].to_numpy()
transformed = one_hot_encoder.fit_transform(cat_vals)

new_col_names = one_hot_encoder.get_feature_names_out(cat_cols)

# putting the transformed data as columns in the trainset dataframe
for i, new_col_name in enumerate(new_col_names):
    train_set[new_col_name] = transformed[:,i]

# checkking if the dummies are produced correctly in the trainset
train_set.head()
```

[52]:

```
                               vehicle_type towing_and_articulation  \
19397                                   Car     No tow/articulation
21539   Van / Goods 3.5 tonnes mgw or under     No tow/articulation
149854                                  Car     No tow/articulation
176779                                  Car     No tow/articulation
60657                                   Car     No tow/articulation

       journey_purpose_of_driver sex_of_driver vehicle_left_hand_drive  \
19397                    Leisure     Not known                      No
21539        Work related travel          Male                      No
149854                   Leisure     Not known                      No
176779       Work related travel          Male                      No
60657        Work related travel          Male                      No

        age_of_driver  engine_capacity_cc propulsion_code  age_of_vehicle  \
19397            38.0              1242.0          Petrol             6.0
21539            29.0              1598.0       Heavy oil             1.0
149854           38.0              1242.0          Petrol             5.0
176779           35.0              1598.0       Heavy oil            10.0
60657            30.0              1968.0       Heavy oil             1.0

       generic_make_model  ... sex_of_driver_Not known  \
19397             FORD KA  ...                     1.0
21539          FIAT DOBLO  ...                     0.0
149854           FIAT 500  ...                     1.0
176779     VOLKSWAGEN GOLF  ...                     0.0
60657             AUDI A3  ...                     0.0

       vehicle_left_hand_drive_Yes propulsion_code_Electric diesel  \
19397                          0.0                             0.0
21539                          0.0                             0.0
```

```
149854                        0.0                             0.0
176779                        0.0                             0.0
60657                         0.0                             0.0

        propulsion_code_Gas  propulsion_code_Gas/Bi-fuel  \
19397                  0.0                          0.0
21539                  0.0                          0.0
149854                 0.0                          0.0
176779                 0.0                          0.0
60657                  0.0                          0.0

        propulsion_code_Heavy oil  propulsion_code_Hybrid electric  \
19397                        0.0                              0.0
21539                        1.0                              0.0
149854                       0.0                              0.0
176779                       1.0                              0.0
60657                        1.0                              0.0

        propulsion_code_Petrol  driver_home_area_type_Small town  \
19397                      1.0                               0.0
21539                      0.0                               0.0
149854                     1.0                               0.0
176779                     0.0                               1.0
60657                      0.0                               0.0

        driver_home_area_type_Urban area
19397                                1.0
21539                                1.0
149854                               1.0
176779                               0.0
60657                                1.0

[5 rows x 43 columns]
```

[53]:

```
# transforming the **testset** using the encoder fitted on trainset
cat_vals = test_set[cat_cols].to_numpy()
transformed = one_hot_encoder.transform(cat_vals)

# putting the transformed data as columns in the testset dataframe
for i, new_col_name in enumerate(new_col_names):
    test_set[new_col_name] = transformed[:,i]

# checkking if the dummies are produced correctly in the testset
test_set.head()
```

[53]:

```
                                  vehicle_type towing_and_articulation  \
120363                                     Car      No tow/articulation
```

```
165257                                   Car     No tow/articulation
161317                                   Car     No tow/articulation
27684                                    Car     No tow/articulation
27271   Motorcycle over 125cc and up to 500cc     No tow/articulation

       journey_purpose_of_driver sex_of_driver vehicle_left_hand_drive  \
120363     School related travel          Male                      No
165257                   Leisure     Not known                      No
161317                   Leisure        Female                      No
27684                    Leisure     Not known                      No
27271        Work related travel          Male                      No

        age_of_driver  engine_capacity_cc propulsion_code  age_of_vehicle  \
120363           19.0              1198.0          Petrol            17.0
165257           38.0              1588.5          Petrol             7.0
161317           46.0              1598.0          Petrol             9.0
27684            38.0              1197.0          Petrol            11.0
27271            33.0               125.0          Petrol             3.0

       generic_make_model  ... sex_of_driver_Not known  \
120363    VOLKSWAGEN POLO  ...                     0.0
165257        FORD FIESTA  ...                     1.0
161317       MINI CLUBMAN  ...                     0.0
27684             AUDI A3  ...                     1.0
27271          YAMAHA GPD  ...                     0.0

       vehicle_left_hand_drive_Yes propulsion_code_Electric diesel  \
120363                         0.0                             0.0
165257                         0.0                             0.0
161317                         0.0                             0.0
27684                          0.0                             0.0
27271                          0.0                             0.0

        propulsion_code_Gas  propulsion_code_Gas/Bi-fuel  \
120363                  0.0                          0.0
165257                  0.0                          0.0
161317                  0.0                          0.0
27684                   0.0                          0.0
27271                   0.0                          0.0

        propulsion_code_Heavy oil  propulsion_code_Hybrid electric  \
120363                        0.0                              0.0
165257                        0.0                              0.0
161317                        0.0                              0.0
27684                         0.0                              0.0
27271                         0.0                              0.0
```

```
        propulsion_code_Petrol  driver_home_area_type_Small town  \
120363                     1.0                              0.0
165257                     1.0                              0.0
161317                     1.0                              1.0
27684                      1.0                              0.0
27271                      1.0                              0.0

        driver_home_area_type_Urban area
120363                               0.0
165257                               1.0
161317                               0.0
27684                                1.0
27271                                1.0

[5 rows x 43 columns]
```

[54]:
```python
train_set.drop(columns=cat_cols, inplace=True)
test_set.drop(columns=cat_cols, inplace=True)
```

[55]:
```python
train_set.head()
```

[55]:
```
        age_of_driver  engine_capacity_cc  age_of_vehicle generic_make_model  \
19397            38.0              1242.0             6.0            FORD KA
21539            29.0              1598.0             1.0         FIAT DOBLO
149854           38.0              1242.0             5.0           FIAT 500
176779           35.0              1598.0            10.0    VOLKSWAGEN GOLF
60657            30.0              1968.0             1.0            AUDI A3

           driver_imd_decile accident_severity  \
19397   More deprived 40-50%            Slight
21539   Less deprived 30-40%            Slight
149854  More deprived 40-50%            Slight
176779  Less deprived 20-30%           Serious
60657   More deprived 20-30%           Serious

        vehicle_type_Bus or coach (17 or more pass seats)  vehicle_type_Car  \
19397                                                 0.0               1.0
21539                                                 0.0               0.0
149854                                                0.0               1.0
176779                                                0.0               1.0
60657                                                 0.0               1.0

        vehicle_type_Electric motorcycle  \
19397                                0.0
21539                                0.0
149854                               0.0
176779                               0.0
```

```
60657                               0.0

       vehicle_type_Goods 7.5 tonnes mgw and over  ...  \
19397                                         0.0  ...
21539                                         0.0  ...
149854                                        0.0  ...
176779                                        0.0  ...
60657                                         0.0  ...

       sex_of_driver_Not known  vehicle_left_hand_drive_Yes  \
19397                      1.0                          0.0
21539                      0.0                          0.0
149854                     1.0                          0.0
176779                     0.0                          0.0
60657                      0.0                          0.0

       propulsion_code_Electric diesel  propulsion_code_Gas  \
19397                              0.0                  0.0
21539                              0.0                  0.0
149854                             0.0                  0.0
176779                             0.0                  0.0
60657                              0.0                  0.0

       propulsion_code_Gas/Bi-fuel  propulsion_code_Heavy oil  \
19397                          0.0                        0.0
21539                          0.0                        1.0
149854                         0.0                        0.0
176779                         0.0                        1.0
60657                          0.0                        1.0

       propulsion_code_Hybrid electric  propulsion_code_Petrol  \
19397                              0.0                     1.0
21539                              0.0                     0.0
149854                             0.0                     1.0
176779                             0.0                     0.0
60657                              0.0                     0.0

       driver_home_area_type_Small town  driver_home_area_type_Urban area
19397                               0.0                               1.0
21539                               0.0                               1.0
149854                              0.0                               1.0
176779                              1.0                               0.0
60657                               0.0                               1.0

[5 rows x 36 columns]
```

[56]:
```
# Define a function to apply ordinal encoding
def apply_ordinal_encoding(train_set, test_set, column_name, categories):
    ordinal_encoder = OrdinalEncoder(categories=[categories])
    train_set[column_name] = ordinal_encoder.
 ↪fit_transform(train_set[[column_name]]) + 1
    test_set[column_name] = ordinal_encoder.transform(test_set[[column_name]]) +␣
 ↪1

# Label encoding the target variable 'accident_severity'
label_encoder = LabelEncoder()
train_set['accident_severity'] = label_encoder.
 ↪fit_transform(train_set['accident_severity'])
test_set['accident_severity'] = label_encoder.
 ↪transform(test_set['accident_severity'])

# Defining categories for 'driver_imd_decile'
driver_imd_decile_categories = ['Least deprived 10%', 'Less deprived 10-20%',
                                'Less deprived 20-30%', 'Less deprived 30-40%',
                                'Less deprived 40-50%', 'More deprived 40-50%',
                                'More deprived 30-40%', 'More deprived 20-30%',
                                'More deprived 10-20%', 'Most deprived 10%']

# Applying ordinal encoding to 'driver_imd_decile'
apply_ordinal_encoding(train_set, test_set, 'driver_imd_decile',␣
 ↪driver_imd_decile_categories)
```

Ordinal encoding preserves the order of categories, making it suitable for accident_severity to reflect the severity scale and driver_imd_decile to maintain the socioeconomic rank. For the target variable accident_severity, label encoding can be appropriate if the model interprets the order as an indication of severity progression, enhancing prediction accuracy.

[57]:
```
train_set.head()
```

[57]:
```
        age_of_driver  engine_capacity_cc  age_of_vehicle generic_make_model  \
19397            38.0              1242.0             6.0            FORD KA
21539            29.0              1598.0             1.0         FIAT DOBLO
149854           38.0              1242.0             5.0           FIAT 500
176779           35.0              1598.0            10.0    VOLKSWAGEN GOLF
60657            30.0              1968.0             1.0            AUDI A3

        driver_imd_decile  accident_severity  \
19397                 6.0                  2
21539                 4.0                  2
149854                6.0                  2
176779                3.0                  1
60657                 8.0                  1
```

```
        vehicle_type_Bus or coach (17 or more pass seats)  vehicle_type_Car  \
19397                                                0.0               1.0
21539                                                0.0               0.0
149854                                               0.0               1.0
176779                                               0.0               1.0
60657                                                0.0               1.0

        vehicle_type_Electric motorcycle  \
19397                                0.0
21539                                0.0
149854                               0.0
176779                               0.0
60657                                0.0

        vehicle_type_Goods 7.5 tonnes mgw and over  ...  \
19397                                          0.0  ...
21539                                          0.0  ...
149854                                         0.0  ...
176779                                         0.0  ...
60657                                          0.0  ...

        sex_of_driver_Not known  vehicle_left_hand_drive_Yes  \
19397                       1.0                          0.0
21539                       0.0                          0.0
149854                      1.0                          0.0
176779                      0.0                          0.0
60657                       0.0                          0.0

        propulsion_code_Electric diesel  propulsion_code_Gas  \
19397                               0.0                  0.0
21539                               0.0                  0.0
149854                              0.0                  0.0
176779                              0.0                  0.0
60657                               0.0                  0.0

        propulsion_code_Gas/Bi-fuel  propulsion_code_Heavy oil  \
19397                           0.0                        0.0
21539                           0.0                        1.0
149854                          0.0                        0.0
176779                          0.0                        1.0
60657                           0.0                        1.0

        propulsion_code_Hybrid electric  propulsion_code_Petrol  \
19397                               0.0                     1.0
21539                               0.0                     0.0
149854                              0.0                     1.0
176779                              0.0                     0.0
```

```
60657                                0.0                      0.0

        driver_home_area_type_Small town  driver_home_area_type_Urban area
19397                                0.0                               1.0
21539                                0.0                               1.0
149854                               0.0                               1.0
176779                               1.0                               0.0
60657                                0.0                               1.0

[5 rows x 36 columns]
```

8 Feature Engineering

A simple interaction between the age of the driver and the engine capacity might indicate that younger drivers with powerful vehicles are at a higher risk of severe accidents.

[58]:
```python
train_set['risk_score'] = train_set['age_of_driver'] *␣
↪train_set['engine_capacity_cc']
test_set['risk_score'] = test_set['age_of_driver'] *␣
↪test_set['engine_capacity_cc']
```

This ratio might provide insights into whether having a high-powered vehicle relative to the driver's age impacts accident severity.

[59]:
```python
train_set['age_to_power_ratio'] = train_set['age_of_driver'] /␣
↪(train_set['engine_capacity_cc'] + 1)  # Adding 1 to avoid division by zero
test_set['age_to_power_ratio'] = test_set['age_of_driver'] /␣
↪(test_set['engine_capacity_cc'] + 1)
```

This feature could help understand if the relative age of the vehicle to the driver has any correlation with accident severity, perhaps indicating less experienced drivers with older or potentially less safe vehicles.

[60]:
```python
train_set['vehicle_to_driver_age_ratio'] = train_set['age_of_vehicle'] /␣
↪(train_set['age_of_driver'] + 1)  # Adding 1 to avoid division by zero
test_set['vehicle_to_driver_age_ratio'] = test_set['age_of_vehicle'] /␣
↪(test_set['age_of_driver'] + 1)
```

[61]:
```python
y_train = train_set["accident_severity"].copy()
X_train = train_set.drop("accident_severity", axis=1)
y_test = test_set["accident_severity"].copy()
X_test = test_set.drop("accident_severity", axis=1)
```

Separating Predictors

[62]:
```python
import category_encoders as ce

loo_encoder = ce.LeaveOneOutEncoder(cols=['generic_make_model'])
```

```python
# Fitting the encoder using the training data and transform the␣
↪'generic_make_model' column
X_train_loo_encoded = loo_encoder.fit_transform(X_train, y_train)
X_test_loo_encoded = loo_encoder.transform(X_test)

X_train['generic_make_model'] = X_train_loo_encoded['generic_make_model']
X_test['generic_make_model'] = X_test_loo_encoded['generic_make_model']
```

Leave-One-Out Encoding (LOO encoding) is a technique often used to encode high-cardinality categorical features, particularly useful to prevent target leakage when dealing with categorical variables in supervised learning tasks. It's similar to target encoding but leaves out the current row's target when calculating the mean target for a level to reduce overfitting.

[63]:
```python
scaler = StandardScaler()

# Scaling the training predictors
scaled_vals_train = scaler.fit_transform(X_train)
X_train = pd.DataFrame(scaled_vals_train, columns=X_train.columns)

# Scaling the testing predictors (using only transform here to use the same␣
↪scaling as the training set)
scaled_vals_test = scaler.transform(X_test)
X_test = pd.DataFrame(scaled_vals_test, columns=X_test.columns)

X_train.head()
```

[63]:
```
   age_of_driver  engine_capacity_cc  age_of_vehicle  generic_make_model  \
0      -0.159811           -0.585932       -0.428383            0.263249
1      -0.784101            0.017548       -1.507735           -5.457642
2      -0.159811           -0.585932       -0.644253            0.298132
3      -0.367907            0.017548        0.435099            0.024878
4      -0.714735            0.644761       -1.507735            0.058931

   driver_imd_decile  vehicle_type_Bus or coach (17 or more pass seats)  \
0           0.021849                                          -0.128524
1          -0.789044                                          -0.128524
2           0.021849                                          -0.128524
3          -1.194491                                          -0.128524
4           0.832742                                          -0.128524

   vehicle_type_Car  vehicle_type_Electric motorcycle  \
0          0.649678                         -0.040859
1         -1.539224                         -0.040859
2          0.649678                         -0.040859
3          0.649678                         -0.040859
4          0.649678                         -0.040859
```

```
   vehicle_type_Goods 7.5 tonnes mgw and over  \
0                                   -0.127403
1                                   -0.127403
2                                   -0.127403
3                                   -0.127403
4                                   -0.127403

   vehicle_type_Goods over 3.5t. and under 7.5t  ...  propulsion_code_Gas  \
0                                     -0.061352  ...            -0.016669
1                                     -0.061352  ...            -0.016669
2                                     -0.061352  ...            -0.016669
3                                     -0.061352  ...            -0.016669
4                                     -0.061352  ...            -0.016669

   propulsion_code_Gas/Bi-fuel  propulsion_code_Heavy oil  \
0                    -0.016669                  -0.675072
1                    -0.016669                   1.481324
2                    -0.016669                  -0.675072
3                    -0.016669                   1.481324
4                    -0.016669                   1.481324

   propulsion_code_Hybrid electric  propulsion_code_Petrol  \
0                        -0.193556                0.758158
1                        -0.193556               -1.318987
2                        -0.193556                0.758158
3                        -0.193556               -1.318987
4                        -0.193556               -1.318987

   driver_home_area_type_Small town  driver_home_area_type_Urban area  \
0                         -0.265767                          0.442061
1                         -0.265767                          0.442061
2                         -0.265767                          0.442061
3                          3.762698                         -2.262133
4                         -0.265767                          0.442061

   risk_score  age_to_power_ratio  vehicle_to_driver_age_ratio
0   -0.516381           -0.141665                    -0.417339
1   -0.540852           -0.377946                    -1.177758
2   -0.516381           -0.141665                    -0.579130
3   -0.266109           -0.306646                     0.364652
4   -0.176992           -0.433054                    -1.184542

[5 rows x 38 columns]
```

Scaling the datasets

[64]:
```python
# Final shape before model training and evaluation
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

[64]: ((7200, 38), (7200,), (1800, 38), (1800,))

[65]:
```python
y_train.value_counts(normalize=True) * 100
```

[65]:
```
accident_severity
2    77.555556
1    20.972222
0     1.472222
Name: proportion, dtype: float64
```

[66]:
```python
y_test.value_counts(normalize=True) * 100
```

[66]:
```
accident_severity
2    77.555556
1    21.000000
0     1.444444
Name: proportion, dtype: float64
```

9 Model Developement

[67]:
```python
# execution time
from timeit import default_timer as timer
from datetime import timedelta
import os
from joblib import dump
from lightgbm import LGBMClassifier
from sklearn.model_selection import RandomizedSearchCV
from timeit import default_timer as timer
from datetime import timedelta
```

10 Baseline

[68]:
```python
from sklearn.dummy import DummyClassifier
from sklearn.metrics import precision_recall_fscore_support, accuracy_score,␣
↪precision_score, recall_score, f1_score

dummy_clf = DummyClassifier()
dummy_clf.fit(X_train, y_train)

yhat = dummy_clf.predict(X_train)
p, r, f, s = precision_recall_fscore_support(y_train, yhat, average="macro",␣
↪zero_division=0.0)
```

```
print("Baseline:")
print(f"Precision: {p:.3f}")
print(f"Recall: {r:.3f}")
print(f"F score: {f:.3f}")
```

```
Baseline:
Precision: 0.259
Recall: 0.333
F score: 0.291
```

11 RandomForestClassifier without SMOTE

[69]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

start = timer()

# Baseline Random forest without HP tuning.
rf_clf = RandomForestClassifier(random_state=7, max_depth=40,␣
↪min_samples_split=5)
rf_clf.fit(X_train, y_train)

# Making predictions on X_test
y_pred = rf_clf.predict(X_test)

end = timer()

# Calculating evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro', zero_division=0)
recall = recall_score(y_test, y_pred, average='macro', zero_division=0)
f1 = f1_score(y_test, y_pred, average='macro', zero_division=0)

# Displaying the execution time and evaluation metrics
print("Execution time HH:MM:SS:", timedelta(seconds=end - start))
print("Classification Report on Test Data:")
print(classification_report(y_test, y_pred, zero_division=0))
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')
```

```
Execution time HH:MM:SS: 0:00:00.920938
Classification Report on Test Data:
              precision    recall  f1-score   support
```

```
              0       0.00      0.00      0.00        26
              1       0.22      0.32      0.26       378
              2       0.78      0.70      0.74      1396

       accuracy                           0.61      1800
      macro avg       0.33      0.34      0.33      1800
   weighted avg       0.65      0.61      0.63      1800

Accuracy: 0.6122
Precision: 0.3344
Recall: 0.3409
F1 Score: 0.3341
```

This will be considered as groundtruth for random forest classifier

12 Applying SMOTE

[70]:
```python
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X_train, y_train)
y_res.value_counts(), y_res.shape
```

[70]:
```
(accident_severity
 2    5584
 1    5584
 0    5584
 Name: count, dtype: int64,
 (16752,))
```

13 RandomForestClassifier

[71]:
```python
from sklearn.model_selection import RandomizedSearchCV
rf_clf_smote = RandomForestClassifier(random_state=7, max_depth=40,␣
 ↪min_samples_split=5)

hp_grid_smote = {
    'n_estimators': [100, 200, 500],
    'max_features': ["sqrt", 0.5],
    'max_samples': [None, 0.5],
}

# 5-fold cross-validation
random_search1 = RandomizedSearchCV(rf_clf_smote, hp_grid_smote, cv=5,
                            scoring='f1_macro',
                            return_train_score=True, verbose=0)
```

```
random_search1.fit(X_res, y_res)

print("Execution time HH:MM:SS:", timedelta(seconds=timer() - start))
```

```
Execution time HH:MM:SS: 0:07:45.216860
```

[72]:
```
# create a folder where all trained models will be kept
if not os.path.exists("models"):
    os.makedirs("models")

dump(random_search1.best_estimator_, 'models/rf_clf_smote.joblib')
```

[72]:
```
['models/rf_clf_smote.joblib']
```

[73]:
```
random_search1.best_score_
```

[73]:
```
0.9161704205877639
```

14 DecisionTreeClassifier

[74]:
```
start = timer()

from sklearn.tree import DecisionTreeClassifier

dt_clf_smote = DecisionTreeClassifier(random_state=7)

hp_grid_dt = {
    'max_depth': [5, 10, 15, 20, 25, 30, 35, 40],
    'min_samples_split': [5, 10, 15, 20, 25, 30, 35],
}

random_search2 = RandomizedSearchCV(dt_clf_smote, hp_grid_dt, cv=5,
                           scoring='f1_macro',
                           return_train_score=True, verbose=0)

random_search2.fit(X_res, y_res)

print("Execution time HH:MM:SS:", timedelta(seconds=timer() - start))
```

```
Execution time HH:MM:SS: 0:00:08.796655
```

[75]:
```
if not os.path.exists("models"):
    os.makedirs("models")

dump(random_search2.best_estimator_, 'models/dt_clf_smote.joblib')
```

[75]:
```
['models/dt_clf_smote.joblib']
```

[76]:
```python
random_search2.best_score_
```

[76]: 0.8364250844041378

15 Linear SVC

[77]:
```python
start = timer()

from sklearn.svm import LinearSVC

lsvm_clf_smote = LinearSVC(random_state=7, max_iter=5000, dual='auto')

hp_grid_lsvm = {
    'C': [0.001, 0.01, 0.1, 1, 10],
}

random_search3 = RandomizedSearchCV(lsvm_clf_smote, hp_grid_lsvm, cv=5,␣
 ↪scoring='f1_macro',
                               return_train_score=True, n_iter=min(5,␣
 ↪len(hp_grid_lsvm['C'])))
random_search3.fit(X_res, y_res)

print("Execution time HH:MM:SS:", timedelta(seconds=timer() - start))
```

```
Execution time HH:MM:SS: 0:00:12.228110
```

[78]:
```python
if not os.path.exists("models"):
    os.makedirs("models")

dump(random_search3.best_estimator_, 'models/lsvm_clf_smote.joblib')
```

[78]: ['models/lsvm_clf_smote.joblib']

[79]:
```python
random_search3.best_score_
```

[79]: 0.4925524340381279

16 SVC-RBF

[80]:
```python
start = timer()
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

rbf_svm_clf_smote = SVC(random_state=7, kernel='rbf')

hp_grid_rbf_svm = {
    'C': [0.01, 0.1, 1, 10, 100],
```

```
}

grid_search = GridSearchCV(rbf_svm_clf_smote, hp_grid_rbf_svm, cv=5,
                           scoring='f1_macro',
                           return_train_score=True, verbose=0)

grid_search.fit(X_res, y_res)

print("Execution time HH:MM:SS:", timedelta(seconds=timer() - start))
```

```
Execution time HH:MM:SS: 0:07:54.611463
```

[81]:
```
if not os.path.exists("models"):
    os.makedirs("models")

dump(grid_search.best_estimator_, 'models/rbf_svm_clf_smote.joblib')
```

[81]:
```
['models/rbf_svm_clf_smote.joblib']
```

[82]:
```
grid_search.best_score_
```

[82]:
```
0.7598294622249778
```

17 GradientBoostingClassifier

[83]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import RandomizedSearchCV
from timeit import default_timer as timer
from datetime import timedelta

start = timer()

# Initialize the Gradient Boosting Classifier
gbc_clf_smote = GradientBoostingClassifier(random_state=7)

# Defining the hyperparameter grid for GBC
hp_grid_gbc = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

random_search_gbc = RandomizedSearchCV(gbc_clf_smote, hp_grid_gbc, cv=5,␣
 ↪scoring='f1_macro',
```

```
                                          n_iter=10, return_train_score=True,␣
 ↪random_state=7)

random_search_gbc.fit(X_res, y_res)

end = timer()

print("Execution time HH:MM:SS:", timedelta(seconds=end - start))
```

Execution time HH:MM:SS: 0:30:09.712501

[84]:
```
if not os.path.exists("models"):
    os.makedirs("models")

dump(random_search_gbc.best_estimator_, 'models/gbc_clf_smote.joblib')
```

[84]: ['models/gbc_clf_smote.joblib']

[85]:
```
random_search_gbc.best_score_
```

[85]: 0.9270472090613386

18 LGBM Classifier

[86]:
```
start = timer()

# Initialize the LGBM Classifier
lgbm_clf_smote = LGBMClassifier(random_state=7,force_row_wise=True, verbosity=-1)

# Defining the hyperparameter grid for LGBM
hp_grid_lgbm = {
    'n_estimators': [100, 200, 300, 400],
    'learning_rate': [0.01, 0.05, 0.1],
    'num_leaves': [31, 50, 70],
    'boosting_type': ['gbdt', 'dart'],
    'max_depth': [3, 5, -1],
    'min_child_samples': [20, 30, 50],
}

# Setup for RandomizedSearchCV for LGBM
random_search_lgbm = RandomizedSearchCV(lgbm_clf_smote, hp_grid_lgbm, cv=5,␣
 ↪scoring='f1_macro',
                                        n_iter=10, return_train_score=True,␣
 ↪random_state=7)

random_search_lgbm.fit(X_res, y_res)
```

```
end = timer()

print("Execution time HH:MM:SS:", timedelta(seconds=end - start))
```

Execution time HH:MM:SS: 0:00:57.999921

[87]:
```
if not os.path.exists("models"):
    os.makedirs("models")

dump(random_search_gbc.best_estimator_, 'models/lgbm_clf_smote.joblib')
```

[87]: ['models/lgbm_clf_smote.joblib']

[88]:
```
random_search_lgbm.best_score_
```

[88]: 0.9324135194266165

19 Evaluation of the Models

[89]:
```
def process_cv_results(search_cv):
    cv_results = pd.DataFrame(search_cv.cv_results_)[['params',␣
 ↪'mean_train_score', 'mean_test_score']]
    cv_results["diff, %"] = 100 * (cv_results["mean_train_score"] -␣
 ↪cv_results["mean_test_score"]) / cv_results["mean_train_score"]
    return cv_results.sort_values('mean_test_score', ascending=False)

pd.set_option('display.max_colwidth', 100)

# Processing the sorted cv results for each search
lgbm_results = process_cv_results(random_search_lgbm)
gbc_results = process_cv_results(random_search_gbc)
search1_results = process_cv_results(random_search1)
search2_results = process_cv_results(random_search2)
grid_search_results = process_cv_results(grid_search)
search3_results = process_cv_results(random_search3)
```

[90]:
```
lgbm_results.head(1)
```

[90]:
```
                 params  \
6  {'num_leaves': 70, 'n_estimators': 300, 'min_child_samples': 30, 'max_depth':
-1, 'learning_rate...

   mean_train_score  mean_test_score   diff, %
6          0.997717         0.932414  6.545255
```

[91]:
```
gbc_results.head(1)
```

[91]:
```
                params  \
5  {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 2,
'max_depth': 7, 'learning_r...

   mean_train_score  mean_test_score  diff, %
5          0.998463         0.927047  7.15256
```

[92]:
```
search1_results.head(1)
```

[92]:
```
                                                                params  \
2  {'n_estimators': 500, 'max_samples': None, 'max_features': 'sqrt'}

   mean_train_score  mean_test_score   diff, %
2          0.999075          0.91617  8.298116
```

[93]:
```
search2_results.head(1)
```

[93]:
```
                                        params  mean_train_score  \
0  {'min_samples_split': 5, 'max_depth': 30}          0.980882

   mean_test_score    diff, %
0         0.836425  14.727269
```

[94]:
```
search3_results.head(1)
```

[94]:
```
     params  mean_train_score  mean_test_score  diff, %
3  {'C': 1}          0.495031         0.492552   0.5007
```

[95]:
```
grid_search_results.head(1)
```

[95]:
```
       params  mean_train_score  mean_test_score   diff, %
4  {'C': 100}          0.812976         0.759829  6.537304
```

Model Name	Training Time	Best Score	diff% between Mean train and test score
LGBM Classifier	01:43.4	0.932413519	6.545255
GradientBoostingClassifier	40:23.5	0.927047209	7.15256
Random Forest Classifier	22:34.0	0.916170421	8.298116
Decision Tree Classifier	00:19.1	0.836425084	14.727269
SVC-RBF	16:50.2	0.759829462	6.537304
Linear SVC	00:22.9	0.492552434	0.5007

Here we can see that LGBM has an efficient time and has a low diff%

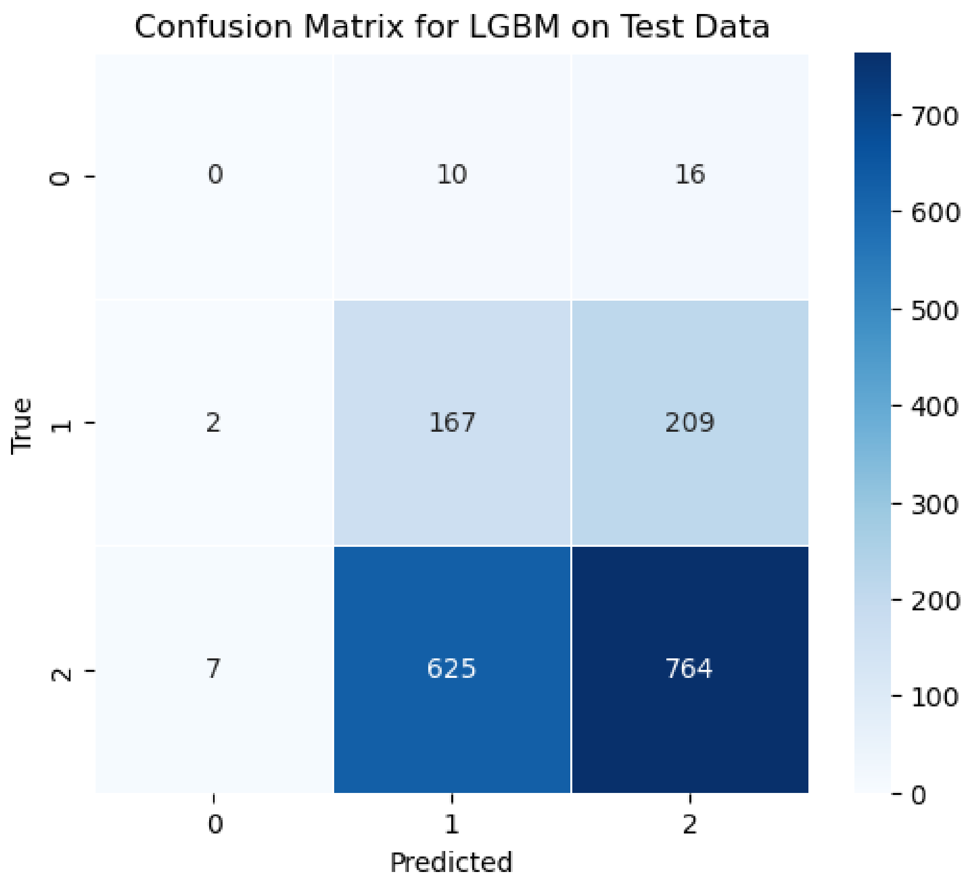
[96]:
```
from joblib import load

best_lgbm = load("models/lgbm_clf_smote.joblib")
best_gbc = load("models/gbc_clf_smote.joblib")
```

```
best_rf = load("models/rf_clf_smote.joblib")
best_dt = load("models/dt_clf_smote.joblib")
best_lsvm = load("models/lsvm_clf_smote.joblib")
best_rbf = load("models/rbf_svm_clf_smote.joblib")
```

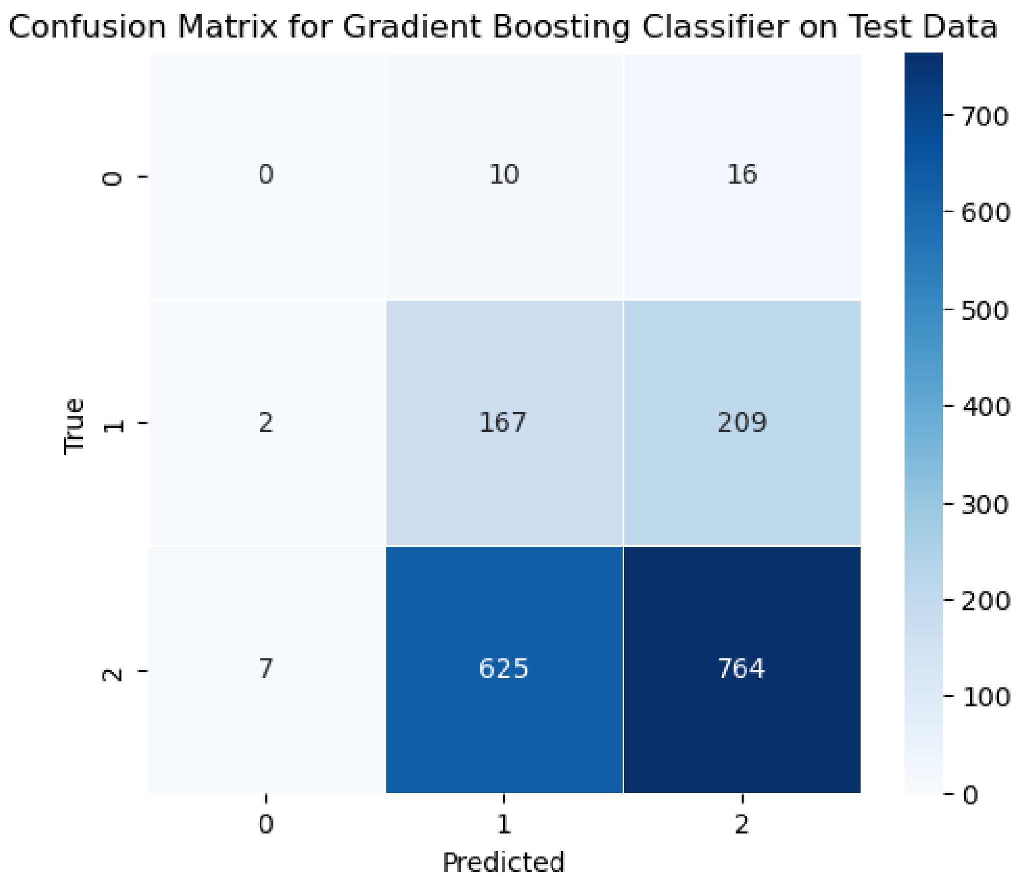
[97]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import precision_recall_fscore_support, confusion_matrix

# Function to evaluate the model
def evaluate_model(model, X_test, y_test, model_name):
    # Predict on the test data
    yhat = model.predict(X_test)

    # Calculating precision, recall, and F-score
    p, r, f, s = precision_recall_fscore_support(y_test, yhat, average="macro")
    print(f"{model_name}:")
    print(f"Precision: {p}")
    print(f"Recall: {r}")
    print(f"F score: {f}")

    # Generating confusion matrix
    cm = confusion_matrix(y_test, yhat)
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", linewidths=0.5)
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.title(f"Confusion Matrix for {model_name} on Test Data")
    plt.show()
```

[98]:
```
evaluate_model(best_lgbm, X_test, y_test, "LGBM")
evaluate_model(best_gbc, X_test, y_test, "Gradient Boosting Classifier")
evaluate_model(best_rf, X_test, y_test, "Random Forest Classifier")
evaluate_model(best_dt, X_test, y_test, "Decision Tree Classifier")
evaluate_model(best_lsvm, X_test, y_test, "Linear SVC")
evaluate_model(best_rbf, X_test, y_test, "SVC-RBF")
```

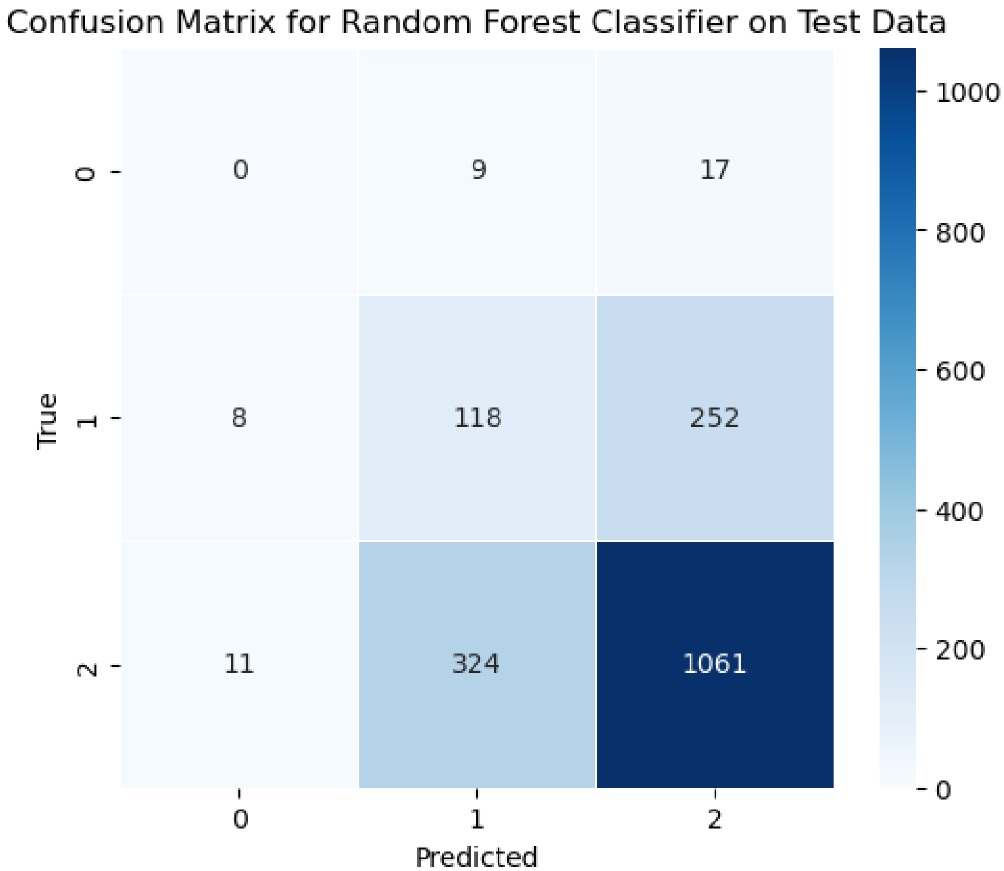
```
LGBM:
Precision: 0.3269089662093502
Recall: 0.3296922929205642
F score: 0.3079072356654704
```



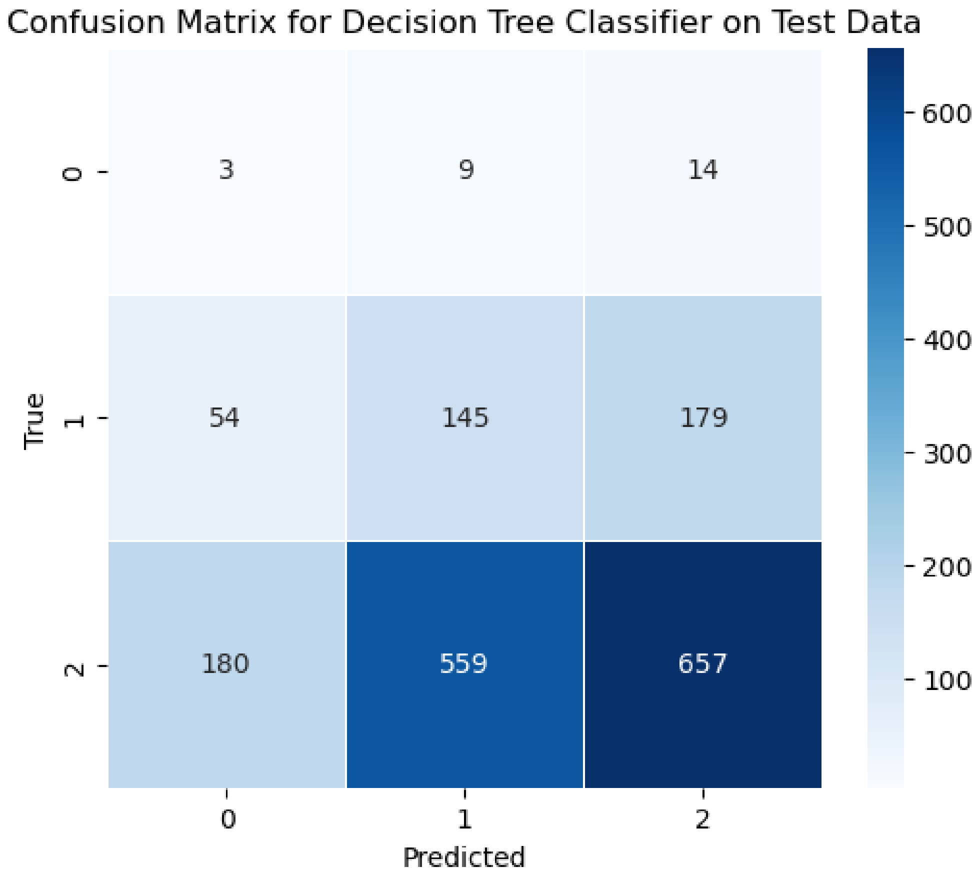

```
Gradient Boosting Classifier:
Precision: 0.3269089662093502
Recall: 0.3296922929205642
F score: 0.3079072356654704
```



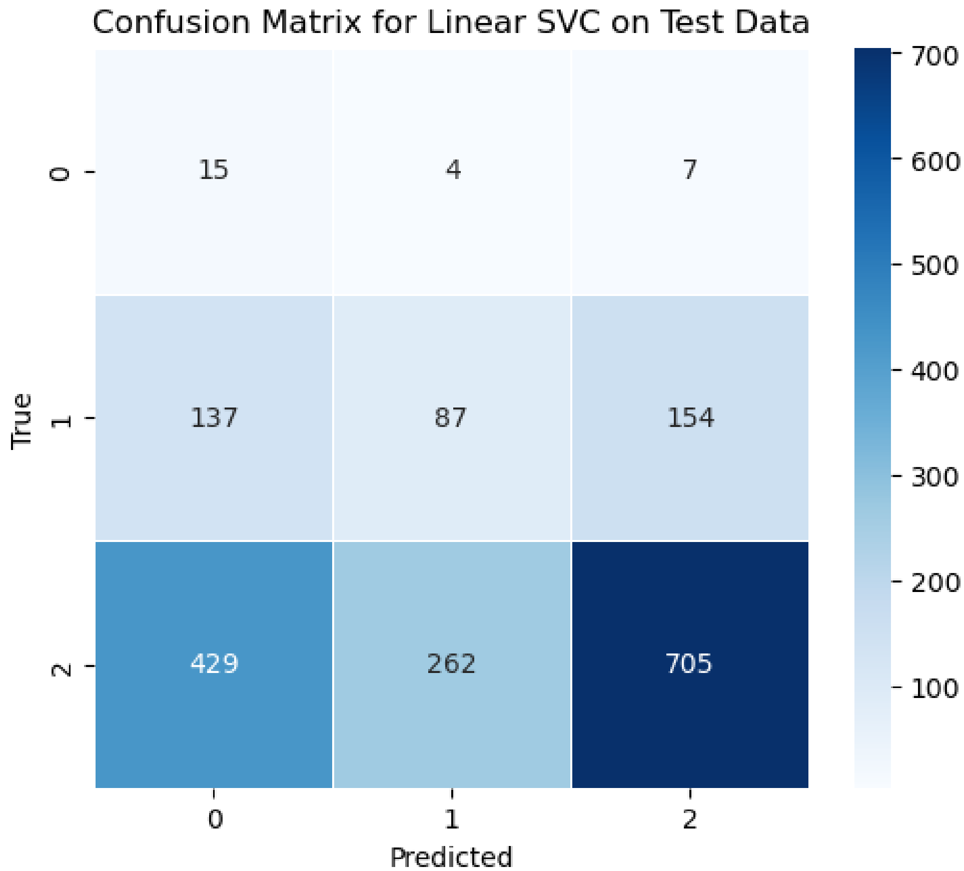

```
Random Forest Classifier:
Precision: 0.3531283863761399
Recall: 0.3573993218214804
F score: 0.3543700905751728
```



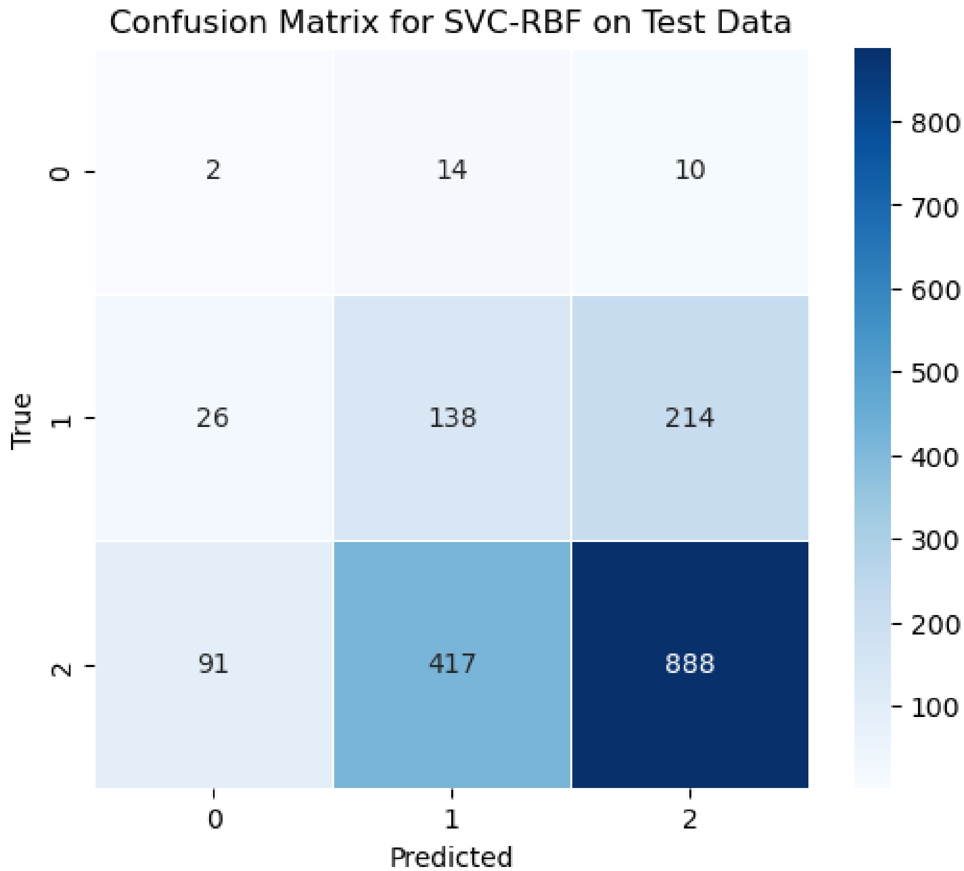

```
Decision Tree Classifier:
Precision: 0.3296551544082401
Recall: 0.32320429049177857
F score: 0.29122164728405014
```




```
Linear SVC:
Precision: 0.36212141308867607
Recall: 0.4373653779097905
F score: 0.30359855485508347
```




```
SVC-RBF:
Precision: 0.352632876493329l4
Recall: 0.359368531288302
F score: 0.3423889506316197
```



LGBM Classifier: - With identical precision, recall, and F-score values to the Gradient Boosting Classifier, the LGBM Classifier shows a lower performance with a precision of 0.327 and recall of 0.330, indicating moderate ability in predicting true positives and relevancy of positive predictions. The F-score of 0.308 suggests a balance between precision and recall but indicates room for improvement in model performance, especially in the context of predicting varying accident severity levels for tailored insurance premiums.

Gradient Boosting Classifier: - Surprisingly The Gradient Boosting Classifier gave same precision and recall (0.327 and 0.330, respectively) as with LGBM, which might indicate similar feature usage or handling by both models or an unknown error. Given the long training time, efficiency improvements are necessary for real-time application in insurance settings.

Random Forest Classifier: - This model outperforms the LGBM and Gradient Boosting models with a precision of 0.353 and recall of 0.357, indicating a slightly better capability in identifying relevant instances and covering the actual positive instances. The F-score of 0.354 implies a better harmonic mean between precision and recall, which could lead to more accurate premium settings based on predicted accident severity.

Decision Tree Classifier: - The Decision Tree shows a precision of 0.330 and a slightly lower recall of 0.323, indicating it may miss some severe accidents while also making some irrelevant predictions as severe. The F-score of 0.291 is the lowest among the models, suggesting it may not be as reliable for setting insurance premiums based on accident severity predictions.

Linear SVC: - The model demonstrates the highest recall of 0.437, suggesting it is the best at identifying severe accidents. However, its F-score of 0.304, despite being higher than the Decision Tree's, still indicates potential misclassification issues. The higher recall is crucial for an insurance company as it may prefer to capture more potential severe accidents, even at the cost of lower precision (0.362).

SVC-RBF: - This model achieves a precision of 0.353 and a recall of 0.359, which are comparable to the Random Forest Classifier, but with a slightly lower F-score of 0.342. This balance makes it a potentially useful model for the insurance company, suggesting a good trade-off between identifying severe cases and avoiding false alarms.

20 Conclusion and Possible future improvements

This project, highlights the performance of the various predictive models in the study, recognizing the challenges presented by imbalanced datasets within which prediction of severity to car accidents is made. The relatively high bias of the 'Slight' severity class in the produced predictions was observed, while the precision and recall scores pertaining to this class remained moderately high. Much lower scores were experienced in cases of 'Fatal' or 'Severe' incidents mostly 0. The model's tendency to over-predict the 'Slight' severity class raises concerns about its practical deployment, particularly when it comes to the accurate prediction of 'Fatal' or 'Severe' incidents. This is indicative of a bias towards the majority class, a common issue in imbalanced datasets, which in this case compromises the model's reliability in identifying high-risk scenarios.

Such an imbalance in prediction could lead to underestimating the risk associated with serious accidents. Therefore, it is important for a car insurance company to employ models that provide balanced and equitable predictions across all severity classes. This is not only crucial for the integrity of the risk assessment process but also for maintaining consumer trust and ensuring that premiums are justly assigned based on the true risk.

To enhance the predictive modeling of accident severity for car insurance premium estimations, a multifaceted approach is necessary, addressing data handling, model optimization, computational constraints, and deployment strategies, all tailored to meet the business objectives.
Outline for future improvements:

Collection of additional data, particularly features that may impact accident severity,like behavioral factors, how many previous accidents was driver invovled in.

Improving data preprocessing to include advanced anomaly detection, error handling, and feature imputation strategies to ensure model inputs are reliable and reflective of real-world scenarios.

Developing new features through domain knowledge exploration, and considering temporal features that could capture changes over time, which is critical for predictive performance.

Experimenting with a variety of machine learning algorithms, including ensemble methods and deep learning, where the complexity of the model is matched to the complexity of the task.

Focusing on models that not only perform well but also provide insights into their predictions, ensuring transparency and facilitating trust with stakeholders.

Applying and experimenting with different resampling strategies to create a balanced dataset that enables the models to learn from all classes equally.

Introduction to custom loss functions in the learning algorithms that penalize misclassifications of the minority class more severely to improve model performance on rare but critical events.

Due to limited computational resources, consideration of algorithms that are more efficient or models that can be updated incrementally can be helpful.

After deploying with a robust pipeline that can handle model versioning and scale according to the data, establishing a monitoring system that tracks the model's performance over time, using real-time data streams to identify and respond to drift, model degradation, or shifts in the underlying data distribution can be helpful.

Modification of models in accordance to Business needs by adjusting models to be particularly sensitive to the business's most critical risks, prioritizing the accurate prediction of severe accidents even if it comes at the cost of predicting less severe accidents with less accuracy.

Developing models that can adjust to changing risk profiles, considering the dynamic nature of risk factors, and ensure models can be updated as new data becomes available.

Precision-Recall Trade-off: Given the high cost associated with severe accidents, prioritizing models with higher recall for severe accidents even if it means accepting lower precision, thereby minimizing the chances of underestimating accident severity.

In summary, by enhancing the data foundation, refining modeling techniques, ensuring computational efficiency, and focusing on business-aligned performance metrics, the predictive modeling process can be significantly improved. These advancements will contribute to a more robust risk assessment framework, allowing the car insurance company to set premiums that are both fair to customers and reflective of actual risk, thus safeguarding the company's financial health.

21 References

Pekar, V. (2024). Big Data for Decision Making. Lecture examples and exercises. (Version 1.0.0). URL: https://github.com/vpekar/bd4dm

[99]:
```python
import nbformat
import re

nb = nbformat.read(open('Individual.ipynb', 'r'), as_version=4)

word_count = 0
heading_pattern = re.compile(r'^#+\s*')  # Pattern to match heading lines

for cell in nb.cells:
    if cell.cell_type == 'markdown':
        text = cell.source
        lines = text.split('\n')
        for line in lines:
            if not heading_pattern.match(line):
                word_count += len(line.split())
```

```python
print(f"Total word count (excluding headings): {word_count}")
```

```
Total word count (excluding headings): 1922
```

[ ]:

```python

```