BIG DATA COURSEWORK-GROUP ASSIGNMENT

GROUP NO 6

CONTRIBUTIONS: Every group member contributed in all the parts of the assignment. Each one of us worked independently and then picked up the best part from everyone's work.

PREDICTIVE MODELLING TO ASSESS THE SEVERITY OF ACCIDENTS

1. INTRODUCTION : BUSINESS OBJECTIVE AND ITS CONTEXT

Emerging risks are harming businesses more than ever in a world, that is changing quickly. When it comes to managing risks, organisations need to take a comprehensive strategy since it may help them achieve long-term success. **Utilizing data and technology** to bring risk to the forefront of decision-making processes is essential.

Car insurance businesses work in the risk industry. Insurance claims are significant for the insurance company. For insurers, high-risk drivers pose the greatest financial hazards. The risks are collectively determined by experience, age,vehicle age, regulations, etc. **Accident Severity** is one of the main factors in determining the risks.

ML is a key tool in loss prediction and risk management because it can quickly identify possibly anomalous or unexpected activities using data and algorithms.

This project's aim is to create a prediction model that a car insurance provider may use to assess the severity of accidents that a car driver who is seeking for insurance is likely to be

involved in. The auto insurance provider will then use that risk to calculate the premium and coverages for that particular driver based on his driving patterns and behaviours.

2. APPROACH/STRATEGY

The Target Feature in our framework is " **ACCIDENT SEVERITY**" which is a **multi-class variable** classified into three main categories; **"Fatal, Serious and Slight"**. Thus, we will be building a **CLASSIFICATION MODEL** which will predict the class of any new instance based on a set of predictors/features.

This project will be completed in 2 main parts:

A. Data Understanding and Data PreprocessingB. Model Building and Evaluating the Performance

Here, we will be carrying out the initial part of the project , in which we will be selecting the main predictor variables, prepare the dataset by merging the data from the available files ,evaluate the quality of the data, clean the raw data, split the data into training and testing set and perform Data Exploration to gain insights about the data.

3. DATA SOURCE

The data has been taken from [ROAD SAFETY DATA]() provided by TRANSPORT DEPARTMENT, UK. The data for the year 2021 has been considered, and features from all 3 types of data files;"**Accidents, Casualties, and Vehicles**"have been chosen following preliminary research.

4. DATA EXTRACTION AND VALIDATION

Here, we will prepare the environment by importing the required libraries and prepare the dataset.

4.1 IMPORTING LIBRARIES

```
#Importing libraries for data loading and data manipulation
# Preparing the environment


#Base Libraries for work with dataframes, importing files and calculations
import re
import time
import warnings
import numpy as np
```

```
import pandas as pd

#Library for Google Colab
from google.colab import drive
#Libraries for plotting graps, bars and plots
import seaborn as sns
sns.set(style="darkgrid")
import matplotlib.pyplot as plt
%matplotlib inline

#Library for calculating statistics
import statistics as stats
from scipy.stats import chi2_contingency

#Libraries for Data Preprocessing and Cleaning
from sklearn.exceptions import ConvergenceWarning
from sklearn.model_selection import train_test_split
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer, SimpleImputer
```

4.2 LOADING DATASET

We will be loading the data into two dataframes from the two files ; Accident and Vehicles **where we will only include the relevant columns.**

```
# loading the data from the "Accidents" file.
accidents = pd.read_csv("https://data.dft.gov.uk/road-accidents-safety-data/dft-road-casua

# validating the shape of the dataframe
accidents.shape
```

```
<ipython-input-2-07f0b2dce23d>:2: DtypeWarning: Columns (0) have mixed types. Specif
  accidents = pd.read_csv("https://data.dft.gov.uk/road-accidents-safety-data/dft-ro
(101087, 3)
```

We have 101,087 rows and 3 columns in accident dataframe where one of the columns is "**ACCIDENT SEVERITY**", which is our **Target** Variable.

```
# validating the accidents dataframe
accidents.head(20)
```

	accident_index	accident_severity	urban_or_rural_area
0	2021010287148	3	1
1	2021010287149	2	1
2	2021010287151	2	1
3	2021010287155	1	1
4	2021010287157	3	1
5	2021010287163	2	1
6	2021010287167	3	1
7	2021010287168	3	1
8	2021010287185	2	1
9	2021010287189	3	1
10	2021010287201	3	1
11	2021010287204	3	1
12	2021010287210	3	1
13	2021010287223	3	1
14	2021010287227	3	1
15	2021010287232	3	1

```
# loading the data from the "Vehicle" file.
vehicles = pd.read_csv('https://data.dft.gov.uk/road-accidents-safety-data/dft-road-casual
                       sep=",", header=0, na_values= -1,usecols=['accident_index', 'vehicl
                       'vehicle_left_hand_drive', 'sex_of_driver', 'age_of_driver',
                       'engine_capacity_cc', 'age_of_vehicle','propulsion_code','driver_im
                       'driver_home_area_type'])

# validating the shape of the dataframe
vehicles.shape
```

```
<ipython-input-4-28549f5e6362>:2: DtypeWarning: Columns (0) have mixed types. Specif
  vehicles = pd.read_csv('https://data.dft.gov.uk/road-accidents-safety-data/dft-roa
(186443, 10)
```

We have 186,443 rows and 10 columns in vehicle dataframe.

```
# validating the vehicle dataframe
vehicles.head(20)
```

	accident_index	vehicle_type	vehicle_left_hand_drive	sex_of_driver	age_of_dri
0	2021010287148	9.0	1.0	1.0	1
1	2021010287148	9.0	1.0	3.0	N
2	2021010287148	9.0	1.0	3.0	N
3	2021010287149	9.0	1.0	1.0	3
4	2021010287149	9.0	1.0	1.0	2
5	2021010287151	9.0	1.0	1.0	2
6	2021010287151	9.0	1.0	1.0	2
7	2021010287155	9.0	1.0	3.0	N
8	2021010287157	9.0	1.0	1.0	3
9	2021010287157	9.0	1.0	3.0	N
10	2021010287157	9.0	1.0	3.0	N
11	2021010287157	9.0	1.0	3.0	N
12	2021010287163	9.0	1.0	1.0	4
13	2021010287163	19.0	1.0	3.0	N
14	2021010287167	9.0	1.0	1.0	2
15	2021010287167	9.0	1.0	3.0	N
16	2021010287168	9.0	1.0	2.0	3
17	2021010287168	11.0	1.0	1.0	5
18	2021010287185	9.0	1.0	1.0	4

4.3 MERGING DATAFRAMES

Here, we will merge the two dataframes using "**accident_index**" as the key.

```
# accidents and vehicles dataframes are merged on "accident_index"
# Inner join is used to ensure that only accidents that have their "accident index" in all

df_merged = accidents.merge(vehicles,on='accident_index', how="inner")
df_merged.shape
```

```
(157146, 12)
```

The joined dataframe has 157,146 rows and 12 columns. It should be noted here that the dataset is large so it may require long time for the prediction model to run through full dataset.

```
# gaining the information about the different types of the variables
df_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 157146 entries, 0 to 157145
Data columns (total 12 columns):
 #   Column                   Non-Null Count   Dtype  
---  ------                   --------------   -----  
 0   accident_index           157146 non-null  object 
 1   accident_severity        157146 non-null  int64  
 2   urban_or_rural_area      157146 non-null  int64  
 3   vehicle_type             157102 non-null  float64
 4   vehicle_left_hand_drive  157139 non-null  float64
 5   sex_of_driver            157144 non-null  float64
 6   age_of_driver            134143 non-null  float64
 7   engine_capacity_cc       101706 non-null  float64
 8   propulsion_code          102747 non-null  float64
 9   age_of_vehicle           102702 non-null  float64
 10  driver_imd_decile        125653 non-null  float64
 11  driver_home_area_type    126131 non-null  float64
dtypes: float64(9), int64(2), object(1)
memory usage: 15.6+ MB
```

Here, we can observe that we have null values in all the columns except accident_index, accident_severity and urban_or_rural area. The **missing values** will be dealt to remove the **noise** in the dataset.

4.4 SELECTING THE RECORDS CORRESPONDING TO "CAR ACCIDENTS"

The predictive model is being built for a Car Insurance Company so it will be logical to study and analyse the data which corresponds to **Car Accidents** only.

```
# counting the unique values for the vehicle type
print(df_merged.vehicle_type.value_counts())
```

```
9.0     106059
1.0      14805
19.0     10264
3.0       7690
5.0       3473
8.0       2492
11.0      2350
21.0      2260
90.0      2065
4.0       1584
2.0       1101
98.0       781
20.0       627
97.0       461
17.0       321
22.0       264
10.0       223
```

```
23.0      194
16.0       66
18.0       22
Name: vehicle_type, dtype: int64
```

It can be observed that there are around 20 different types of vehicles which are involved in accidents where **9 refers to Car** , **8 refers to taxi/hire car** and **19 refers to van**. **It is worthwhile to note that the maximum number of accidents are CAR ACCIDENTS .**

```
# selecting the records corresponding to Car Accidents only i.e. with vehicle type as 8,9

df_merged=df_merged.loc[df_merged['vehicle_type'].isin([8, 9, 19])]
```

Now, the dataframe "df_merged" contains the data corresponding only to car accidents.

```
# validating the dataframe after extracting the relevant dataset.
print(df_merged.vehicle_type.value_counts())
```

```
9.0     106059
19.0     10264
8.0       2492
Name: vehicle_type, dtype: int64
```

```
# checking the shape of the dataframe
df_merged.shape
```

```
(118815, 12)
```

The dataframe now has 118,815 rows and 12 columns.

5. DATA PREPARATION

This part will be carried out in 4 steps; **Data Cleaning, Data Transformation , Sampling and Splitting**. However, it is important to note that **Data Cleaning and Data Transformation is an **iterative process** and there is a possibility of **repeating these steps based on the EDA.**

5.1 DATA CLEANING

" If the input data is seriously flawed, no amount of statistical massaging will produce a meaningful result".(Guttag,John.V, 2017).

This relates to the concept of **GIGO; "GARBAGE IN GARBAGE OUT"**. The overriding objective of minimizing GIGO can be achieved by **Data Cleaning** which involves with dealing of **duplicate records, missing values and outliers.**

5.1.1 DELETING COLUMNS "VEHICLE TYPE" AND "ACCIDENT INDEX"

The "vehicle_type" column was used to the extract the data related to passenger cars which was important for our business objective. Now, as the data extraction has been done, this column will be of no use for creating the model. The "acccident_index" was used to merge datasets as the key index, and it is not important for future analysis. These 2 columns will be dropped from the data frame.

```
# deleting the vehicle_type column
del df_merged['vehicle_type']

# deleting the accident_index column
del df_merged['accident_index']

# checking the dataframe
df_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 118815 entries, 0 to 157145
Data columns (total 10 columns):
 #   Column                   Non-Null Count   Dtype  
---  ------                   --------------   -----  
 0   accident_severity        118815 non-null  int64  
 1   urban_or_rural_area      118815 non-null  int64  
 2   vehicle_left_hand_drive  118808 non-null  float64
 3   sex_of_driver            118813 non-null  float64
 4   age_of_driver            98838 non-null   float64
 5   engine_capacity_cc       87799 non-null   float64
 6   propulsion_code          88576 non-null   float64
 7   age_of_vehicle           88553 non-null   float64
 8   driver_imd_decile        93003 non-null   float64
 9   driver_home_area_type    93368 non-null   float64
dtypes: float64(8), int64(2)
memory usage: 10.0 MB
```

5.1.2 HANDLING DUPLICATE RECORDS

Duplicate records lead to an overweighting of the data values in those records and creates **BIAS**. Thus, they needs to be dropped from the dataset.

```
# checking the duplication of the records
print(df_merged[df_merged.duplicated()])
```

```
        accident_severity  urban_or_rural_area  vehicle_left_hand_drive  \
30                      3                    1                      1.0   
34                      3                    1                      1.0   
79                      3                    1                      1.0   
110                     3                    1                      1.0   
116                     3                    1                      1.0   
...                   ...                  ...                      ...   
157072                  3                    1                      1.0   
```

```
157078              3                    2                        1.0
157087              2                    1                        1.0
157128              2                    1                        1.0
157141              3                    1                        1.0

        sex_of_driver  age_of_driver  engine_capacity_cc  propulsion_code  \
30                3.0            NaN                 NaN              NaN
34                3.0            NaN                 NaN              NaN
79                3.0            NaN                 NaN              NaN
110               3.0            NaN                 NaN              NaN
116               3.0            NaN                 NaN              NaN
...               ...            ...                 ...              ...
157072            2.0           55.0                 NaN              NaN
157078            1.0           27.0                 NaN              NaN
157087            1.0           45.0              1598.0              2.0
157128            1.0           33.0                 NaN              NaN
157141            1.0           32.0              1798.0              8.0

        age_of_vehicle  driver_imd_decile  driver_home_area_type
30                 NaN                NaN                    NaN
34                 NaN                NaN                    NaN
79                 NaN                NaN                    NaN
110                NaN                NaN                    NaN
116                NaN                NaN                    NaN
...                ...                ...                    ...
157072             NaN                5.0                    1.0
157078             NaN                7.0                    1.0
157087             3.0                4.0                    1.0
157128             NaN                7.0                    1.0
157141             4.0                1.0                    1.0

[28277 rows x 10 columns]
```

There are **28277 duplicate records** which needs to be dropped.

```
# dropping the duplicate records and keeping the first
df_merged.drop_duplicates(keep = 'first', inplace = True)

# validating the dataframe
df_merged.shape
```

```
(90538, 10)
```

5.1.3 HANDLING THE MISSING VALUES

```
# checking the number of missing values in all the columns
df_merged.isna().sum()
```

```
accident_severity             0
urban_or_rural_area           0
vehicle_left_hand_drive       7
sex_of_driver                 2
age_of_driver              7182
engine_capacity_cc         8504
```

```
propulsion_code            7788
age_of_vehicle             7811
driver_imd_decile         10240
driver_home_area_type      9880
dtype: int64
```

The percentage of missing values will give us a better understanding and will help us deciding the srategy to deal with them.

```
# calculating the percentage of missing values
df_merged.isnull().sum() * 100 / len(df_merged)
```

```
accident_severity           0.000000
urban_or_rural_area         0.000000
vehicle_left_hand_drive     0.007732
sex_of_driver               0.002209
age_of_driver               7.932581
engine_capacity_cc          9.392741
propulsion_code             8.601913
age_of_vehicle              8.627317
driver_imd_decile          11.310168
driver_home_area_type      10.912545
dtype: float64
```

The target variable and urban-or-rural area column doesnt have any missing values. All the other columns have less than or around 10% of missing values.

We have a large dataset at our disposition and since, we have to take a sample of only 10k records, we can **drop the missing values** without any **loss of information**. **Else we would have imputed the missing values using Mean/Median or Mode.**

```
# dropping the missing values
df_merged = df_merged.dropna()
```

```
# validating the dataframe
df_merged.shape
```

```
(70100, 10)
```

Now, the dataframe contains 70,100 rows and 10 columns.

5.2 SELECTING A SAMPLE OF 20,000 RECORDS

A sample of 20,000 records will be selected . More data can be roped in at any point of time as the data cleaning has already been done.

```
# selecting a sample of 20,000 records.
df_merged=df_merged.sample(n=20000, random_state=7)

# validating the dataframe
df_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20000 entries, 97658 to 115857
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   accident_severity       20000 non-null  int64
 1   urban_or_rural_area     20000 non-null  int64
 2   vehicle_left_hand_drive 20000 non-null  float64
 3   sex_of_driver           20000 non-null  float64
 4   age_of_driver           20000 non-null  float64
 5   engine_capacity_cc      20000 non-null  float64
 6   propulsion_code         20000 non-null  float64
 7   age_of_vehicle          20000 non-null  float64
 8   driver_imd_decile       20000 non-null  float64
 9   driver_home_area_type   20000 non-null  float64
dtypes: float64(8), int64(2)
memory usage: 1.7 MB
```

5.3 MAPPING OF THE LABELS TO THE CATEGORIES NAMES

Most of the categorical variables have been labelled as Numbers and their description has been given in an excel file. For better clarity and comprehension, all the labels needs to be converted to their proper categorical names.

```
# loading the file having the categories names
url = ('https://data.dft.gov.uk/road-accidents-safety-data/Road-Safety-Open-Dataset-Data-G

# printing the heads of the road safety guide document to see first 20 rows
RS_guide = pd.read_excel(url, header=0)
RS_guide.head(20)
```

	table	field name	code/format	label	note
0	Accident	accident_index	NaN	NaN	unique value for each accident. The accident_i...
1	Accident	accident_year	NaN	NaN	NaN
2	Accident	accident_reference	NaN	NaN	In year id used by the police to reference a c...
3	Accident	location_easting_osgr	NaN	NaN	Null if not known
4	Accident	location_northing_osgr	NaN	NaN	Null if not known
5	Accident	longitude	NaN	NaN	Null if not known
6	Accident	Latitude	NaN	NaN	Null if not known
7	Accident	police_force	1	Metropolitan Police	NaN
8	Accident	police_force	3	Cumbria	NaN

```
#Grouping data using the column field name
Map = RS_guide.groupby(RS_guide['field name'])
```

```
#defining a function to create individual dictionaries for each categorical variable
def getfunc(object, column):
  x = object.get_group(column).drop(columns=['table', 'field name'])
  x = x.set_index('code/format')['label'].to_dict()
  return x
```

```
#Mapping with Accidents dataset
df_merged['accident_severity'] = df_merged['accident_severity'].map(getfunc(Map,'accident_
df_merged['urban_or_rural_area'] = df_merged['urban_or_rural_area'].map(getfunc(Map,'urbar
df_merged['vehicle_left_hand_drive'] = df_merged['vehicle_left_hand_drive'].map(getfunc(Ma
df_merged['sex_of_driver'] = df_merged['sex_of_driver'].map(getfunc(Map,'sex_of_driver'))
df_merged['driver_imd_decile'] = df_merged['driver_imd_decile'].map(getfunc(Map,'driver_im
df_merged['driver_home_area_type'] = df_merged['driver_home_area_type'].map(getfunc(Map,'c
df_merged['propulsion_code'] = df_merged['propulsion_code'].map(getfunc(Map,'propulsion_cc
```

```
/usr/local/lib/python3.9/dist-packages/pandas/core/indexes/base.py:6999: FutureWarni
  return Index(sequences[0], name=names)
/usr/local/lib/python3.9/dist-packages/pandas/core/indexes/base.py:6999: FutureWarni
  return Index(sequences[0], name=names)
/usr/local/lib/python3.9/dist-packages/pandas/core/indexes/base.py:6999: FutureWarni
  return Index(sequences[0], name=names)
/usr/local/lib/python3.9/dist-packages/pandas/core/indexes/base.py:6999: FutureWarni
  return Index(sequences[0], name=names)
/usr/local/lib/python3.9/dist-packages/pandas/core/indexes/base.py:6999: FutureWarni
  return Index(sequences[0], name=names)
/usr/local/lib/python3.9/dist-packages/pandas/core/indexes/base.py:6999: FutureWarni
  return Index(sequences[0], name=names)
/usr/local/lib/python3.9/dist-packages/pandas/core/indexes/base.py:6999: FutureWarni
  return Index(sequences[0], name=names)
```

```
# validating the dataframe
df_merged.head()
```

	accident_severity	urban_or_rural_area	vehicle_left_hand_drive	sex_of_driv
97658	Slight	Rural	No	Fem
36474	Slight	Urban	No	Fem
123951	Serious	Urban	No	M
24623	Slight	Urban	No	Fem
104031	Slight	Urban	No	M

```
# checking the data type of the variables
df_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20000 entries, 97658 to 115857
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   accident_severity        20000 non-null  object 
 1   urban_or_rural_area      20000 non-null  object 
 2   vehicle_left_hand_drive  20000 non-null  object 
 3   sex_of_driver            20000 non-null  object 
 4   age_of_driver            20000 non-null  float64
 5   engine_capacity_cc       20000 non-null  float64
 6   propulsion_code          20000 non-null  object 
 7   age_of_vehicle           20000 non-null  float64
 8   driver_imd_decile        20000 non-null  object 
 9   driver_home_area_type    20000 non-null  object 
dtypes: float64(3), object(7)
memory usage: 1.7+ MB
```

The data is now ready for splitting.

5.4 TRAIN TEST SPLIT

The dataset will be divided into training and testing sets where the model will be **created on the training set** and the model's performance will be **evaluated on the testing set.** The data will be splitted in 80-20 ratio.

5.4.1 RANDOM SAMPLING

As the dataset is **large**, we can do random sampling **without worrying about the sampling error.**

```
# performing random train-test split
rand_train_set, rand_test_set = train_test_split(df_merged, test_size=0.2, random_state=7)
print(f"There are {rand_train_set.shape[0]} instances for training and {rand_test_set.shap
```

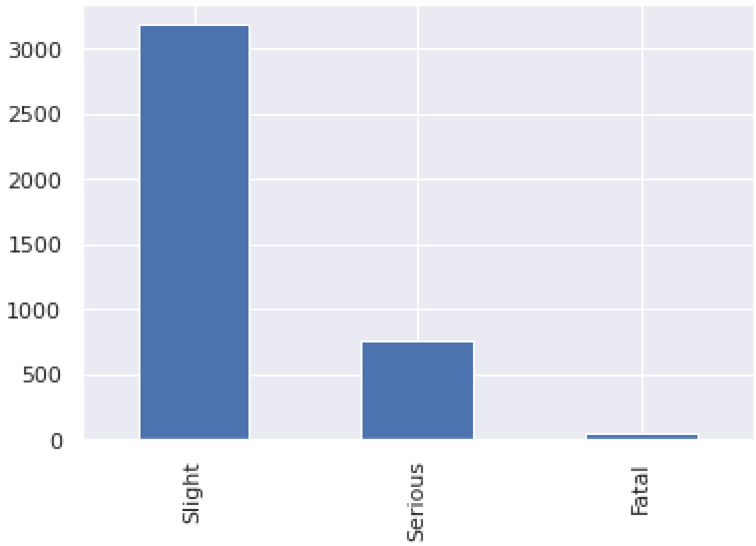
```
There are 16000 instances for training and 4000 instances for testing
```

5.4.1.1 CHECKING FOR CLASS IMBALANCE

Here, our target variable is " Accident Severity", which is a categorical variable , so it is very important to check and ensure that all the **categories of the class are equally represented in the TEST SET**

```
# drawing a bargraph to understand the distribution of categories in the test set.
rand_test_set['accident_severity'].value_counts().plot(kind='bar')
```

```
<Axes: >
```



Here, we can observe that there is a class imbalance and the category "Slight" is overrepresented . Thus, we will try to split the dataset using Stratified Sampling .

5.4.2 STRATIFIED SAMPLING

Stratified Sampling ensure that the frequencies of each group is sufficiently large and the same proportion of group sizes are achieved after the train-test split.

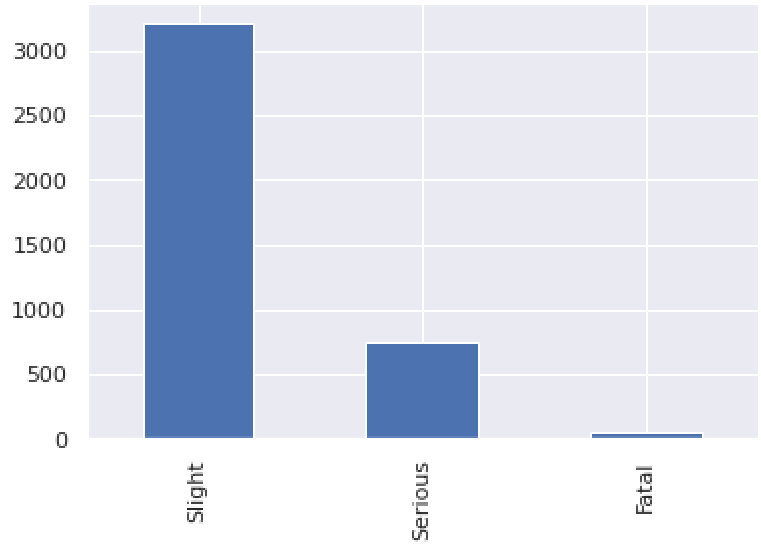
```
# performing Stratified Splitting , using the variable "accident _severity" for stratifica
strat_train_set, strat_test_set = train_test_split(df_merged, test_size=0.2, random_state=
                                                    stratify=df_merged["accident_severity"]
print(f"There are {strat_train_set.shape[0]} instances for training and {strat_test_set.sh
```

```
There are 16000 instances for training and 4000 instances for testing
```

5.4.3 **COMPARING THE PROPORTIONS OF DIFFERENT CATEGORIES IN THE TESTING DATA FRAME**

```
# drawing a bargraph to understand the distribution of categories in the test set.
strat_test_set['accident_severity'].value_counts().plot(kind='bar')
```

```
<Axes: >
```




```
# checking the accident severity category proportions in the full dataset, in the test set
def acc_ser(data):
    return data["accident_severity"].value_counts() / len(data)

# create a temporary dataframe for easy visualization
df_tmp = pd.DataFrame({
    "Overall": acc_ser(df_merged),
    "Random test set": acc_ser(rand_test_set),
    "Stratified test set": acc_ser(strat_test_set),
}).sort_index()

# add two columns for the percent of the difference to the overall proportion
df_tmp["Rand. %error"] = 100 * df_tmp["Random test set"] / df_tmp["Overall"] - 100
df_tmp["Strat. %error"] = 100 * df_tmp["Stratified test set"] / df_tmp["Overall"] - 100

df_tmp
```

	Overall	Random test set	Stratified test set	Rand. %error	Strat. %error
Fatal	0.01215	0.01275	0.01225	4.938272	0.823045
Serious	0.18535	0.19000	0.18525	2.508767	-0.053952
Slight	0.80250	0.79725	0.80250	-0.654206	0.000000

So random splitting produces a test set where "Fatal" category and "Serious" category is over-represented by 5% and almost 3% respectively, as compared to its proportion in the overall dataset, and "Slight" category is under-represented by 1%.

Stratification sampling resulted in under- or over-representations of categories of no more than 1%.

So we will proceed ahead with Stratified Sampling. However, the BAR GRAPH is still showing the "CLASS IMBALANCE" which will be dealt with at a later stage.

```
# storing the splitted dataframes into 2 new dataframes.
trainset = strat_train_set
testset = strat_test_set
print(f"There are {trainset.shape[0]} train and {testset.shape[0]} test instances")
```

```
There are 16000 train and 4000 test instances
```

6.EXPLORATORY DATA ANALYSIS

EDA will be done on training data set to gain more insights about the data. There are 2 types of variables in the dataframe; Numerical and Categorical where,

Numerical(Continuous) variables: age_of_driver, engine_capacity_cc, age_of_vehicle.

Categorical variables: accident_severity, urban_or_rural_area, vehicle_left_hand_drive, sex_of_driver, propulsion_code, driver_imd_decile, driver_home_area_type.

6.1 DESCRIPTIVE STATISTICS

It is used to describe the basic features of the data in a summary form.

```
# finding the summary statistics for the numerical variables.
trainset.describe()
```

	age_of_driver	engine_capacity_cc	age_of_vehicle
count	16000.000000	16000.000000	16000.000000
mean	41.698375	1698.308438	8.524688
std	16.646752	605.126357	5.373646
min	13.000000	505.000000	0.000000
25%	28.000000	1364.000000	4.000000
50%	39.000000	1598.000000	8.000000
75%	53.000000	1987.000000	12.000000
max	100.000000	29980.000000	89.000000

Observations:

1. The mean age of driver is 40 years , with not that high standard deviation. It means that the values are clustered around the mean only.
2. The min and max age of the driver is 13 years and 100 years respectively, where 13 years seems to be a wrong input. It will either be removes as an outlier or will have to be dealt with.
3. Again, the maximum value of engine capacity is starkingly high, which again can be a wrong input and will be dealt with.
4. 25% of cars involved in accident have age more than 12 years which highlights the fact that this can be an important factor **in deciding the insurance.**

Statistical measures can not computed for categorical variables so we use the mode to explore this data.

- Mode - the value that appears most frequently in the dataset.

```
#calculating the mode for categorical variables
mode_accident_severity = trainset.accident_severity.mode()[0]
mode_urban_or_rural_area = trainset.urban_or_rural_area.mode()[0]
mode_vehicle_left_hand_drive = trainset.vehicle_left_hand_drive.mode()[0]
mode_sex_of_driver = trainset.sex_of_driver.mode()[0]
mode_propulsion_code = trainset.propulsion_code.mode()[0]
mode_driver_imd_decile = trainset.driver_imd_decile.mode()[0]
mode_driver_home_area_type = trainset.driver_home_area_type.mode()[0]
#Printing the results
print(f"Mode of accident_severity = {mode_accident_severity}")
print(f"Mode of urban_or_rural_area = {mode_urban_or_rural_area}")
print(f"Mode of vehicle_left_hand_drive = {mode_vehicle_left_hand_drive}")
print(f"Mode of sex_of_driver = {mode_sex_of_driver}")
print(f"Mode of propulsion_code = {mode_propulsion_code}")
print(f"Mode of driver_imd_decile = {mode_driver_imd_decile}")
print(f"Mode of driver_home_area_type = {mode_driver_home_area_type}")
```

```
Mode of accident_severity = Slight
Mode of urban_or_rural_area = Urban
Mode of vehicle_left_hand_drive = No
Mode of sex_of_driver = Male
Mode of propulsion_code = Petrol
Mode of driver_imd_decile = More deprived 10-20%
Mode of driver_home_area_type = Urban area
```

Observations:

1. The majority of the accidents happened in **Urban Area** and **Petrol Cars** were a major cause of accidents.
2. Most of the drivers were from highly deprived class and again **it can be one of the most important factors for the insurance companies.**

6.2 UNIVARIATE VISUALISATION

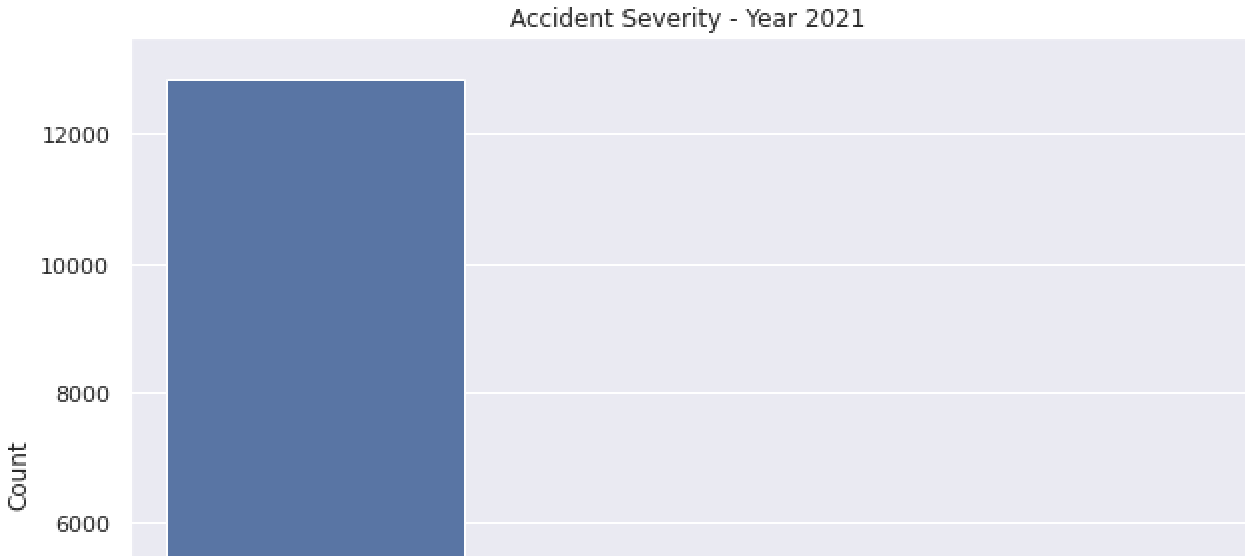
Univariate Visualisation and Analysis of the trainset will not only help us in understanding the distribution of the variables but will also help us in uncovering the hidden trends, patterns, and anomalies.

6.2.1 TARGET VARIABLE

```
#getting the value counts of the accident_severity variable
values = pd.DataFrame(trainset.loc[:,'accident_severity'].value_counts())
values.columns = ['Accident Severity Count']

# computing the percentage of each category using the normalize attribute in the value_cou
percentages = pd.DataFrame(round(trainset.loc[:,'accident_severity'].value_counts(normaliz
percentages.columns = ['accident_severity %']
values.join(percentages)
```

	Accident Severity Count	accident_severity %
Slight	12840	80.25%
Serious	2966	18.54%
Fatal	194	1.21%

```
# Plotting a BARGRAPH to analyse the variable
import seaborn as sns
#Define plot object, setting figure size and colouring scheme
sns.set_theme(palette="Set3")
sns.set(rc={'figure.figsize':(10,8)})
count = sns.countplot(x = trainset['accident_severity'])
#Setting graph title
count.set_title('Accident Severity - Year 2021')
count.set(xlabel = 'Accident Severity', ylabel = 'Count')
plt.xticks(rotation=0)
#Showing the plot
plt.show()
```



It can be observed that the majority of accidents happening are slight in nature.

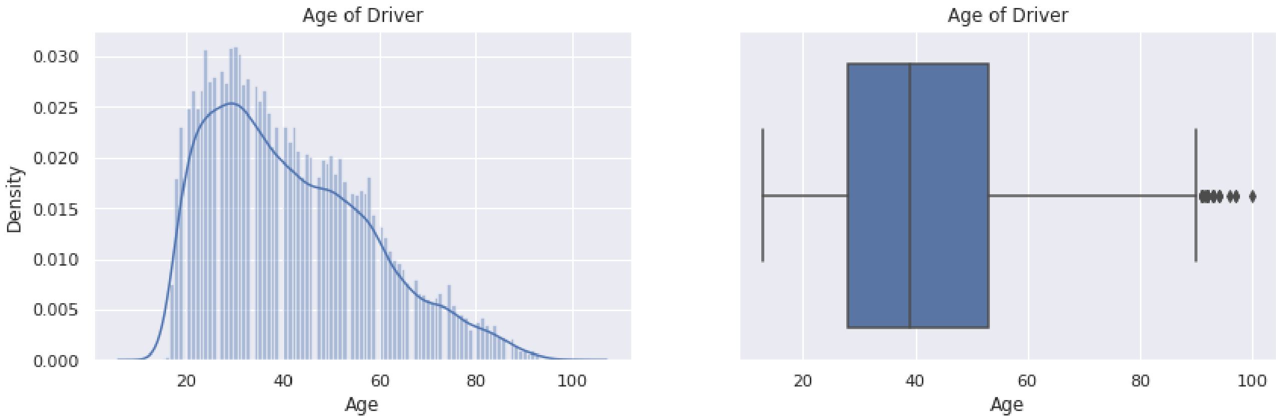
6.2.2 NUMERICAL PREDICTORS

Histograms will be plotted to trace the skewness of the variables and BoxPlots will be plotted for the detection of outliers.

6.2.2.1 AGE OF DRIVER

```
# plotting a histogram
fig = plt.figure(figsize = (14,4))
#adding the histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(trainset.age_of_driver, bins = 100)
#Setting graph title
hist.set_title('Age of Driver')
hist.set(xlabel = 'Age')

#plotting a boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x= trainset['age_of_driver'])
#Setting graph title
box.set_title('Age of Driver')
box.set(xlabel = 'Age')
#Showing the plot
plt.show()
```

```
<ipython-input-37-f0a10292bbea>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  hist = sns.distplot(trainset.age_of_driver, bins = 100)
```

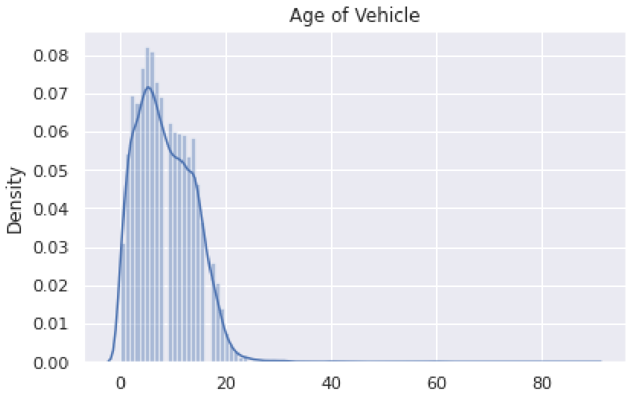


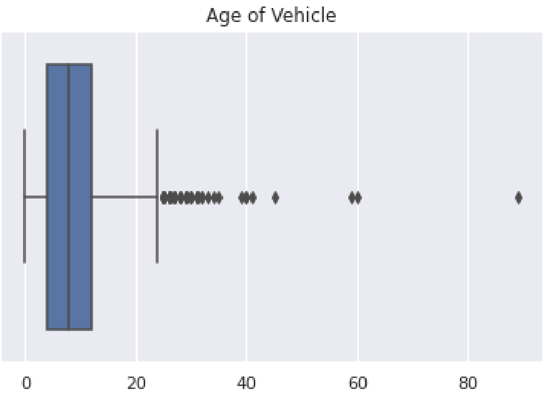
The distribution for the age of driver is **highly positively skewed** *. ***Apart from this, the box plot highlights the*** ***presence of few outliers** in the training set.

6.2.2.2 AGE OF VEHICLE

```
# plotting a histogram
fig = plt.figure(figsize = (14,4))
#adding histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(trainset.age_of_vehicle, bins = 100)
#Setting graph title
hist.set_title('Age of Vehicle')
hist.set(xlabel = 'Age')

# plotting a boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x = trainset['age_of_vehicle'])
#Setting graph title
box.set_title('Age of Vehicle')
box.set(xlabel = 'Age')
#Showing the plot
plt.show()
```

```
<ipython-input-38-b64316465555>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  hist = sns.distplot(trainset.age_of_vehicle, bins = 100)
```



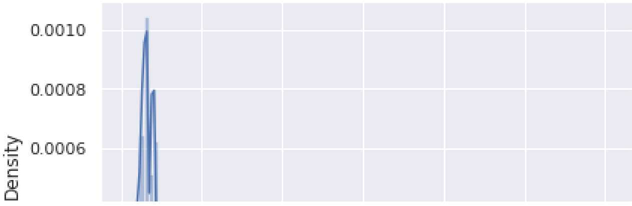


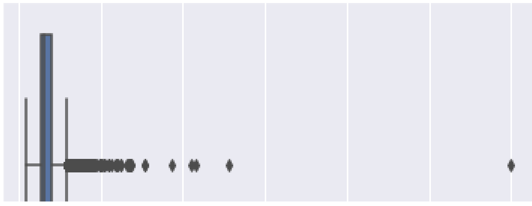
Here also the distribution is positively skewed with presence of few outliers.

6.2.2.3 ENGINE CAPACITY

```
# plotting the histogram
fig = plt.figure(figsize = (14,4))
#adding histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(trainset.engine_capacity_cc, bins = 100)
#Setting graph title
hist.set(xlabel = 'Engine capacity')

#plotting a boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x = trainset['engine_capacity_cc'])
#Setting graph title
box.set(xlabel = 'Engine Capacity')
#Showing the plot
plt.show()
```

```
<ipython-input-39-89ba17eda953>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  hist = sns.distplot(trainset.engine_capacity_cc, bins = 100)
```





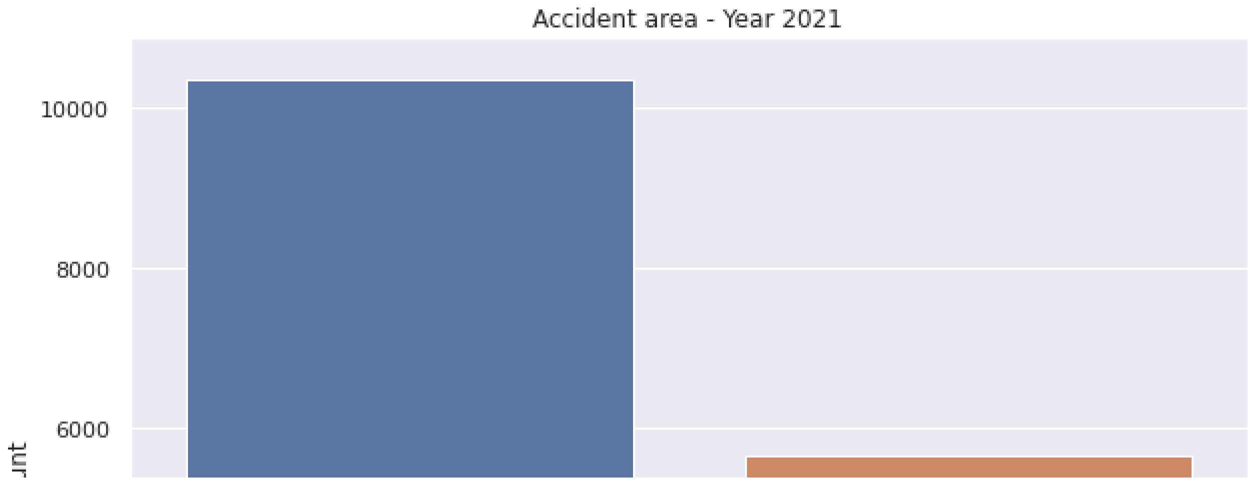
All the three numerical predictors are positively skewed and have outliers as well. Also, the scale for engine capacity is different as compared to the other 2 variables.

All the corrective measures will be taken for the above mentioned points.

6.2.3 CATEGORICAL PREDICTORS

6.2.3.1 ACCIDENT AREA

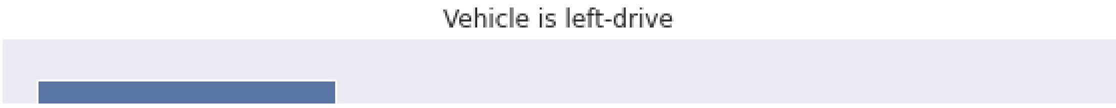
```
#Define plot object, setting figure size and colouring scheme
sns.set_theme(palette="Set3")
sns.set(rc={'figure.figsize':(10,8)})
count = sns.countplot( x = trainset['urban_or_rural_area'])
#Setting graph title
count.set_title('Accident area - Year 2021')
count.set(xlabel = 'Accident area', ylabel = 'Count')
plt.xticks(rotation=0)
#Showing the plot
plt.show()
```



Urban areas are more prone to accidents as compared to rural areas.

6.2.3.2 VEHICLE LEFT HAND DRIVE

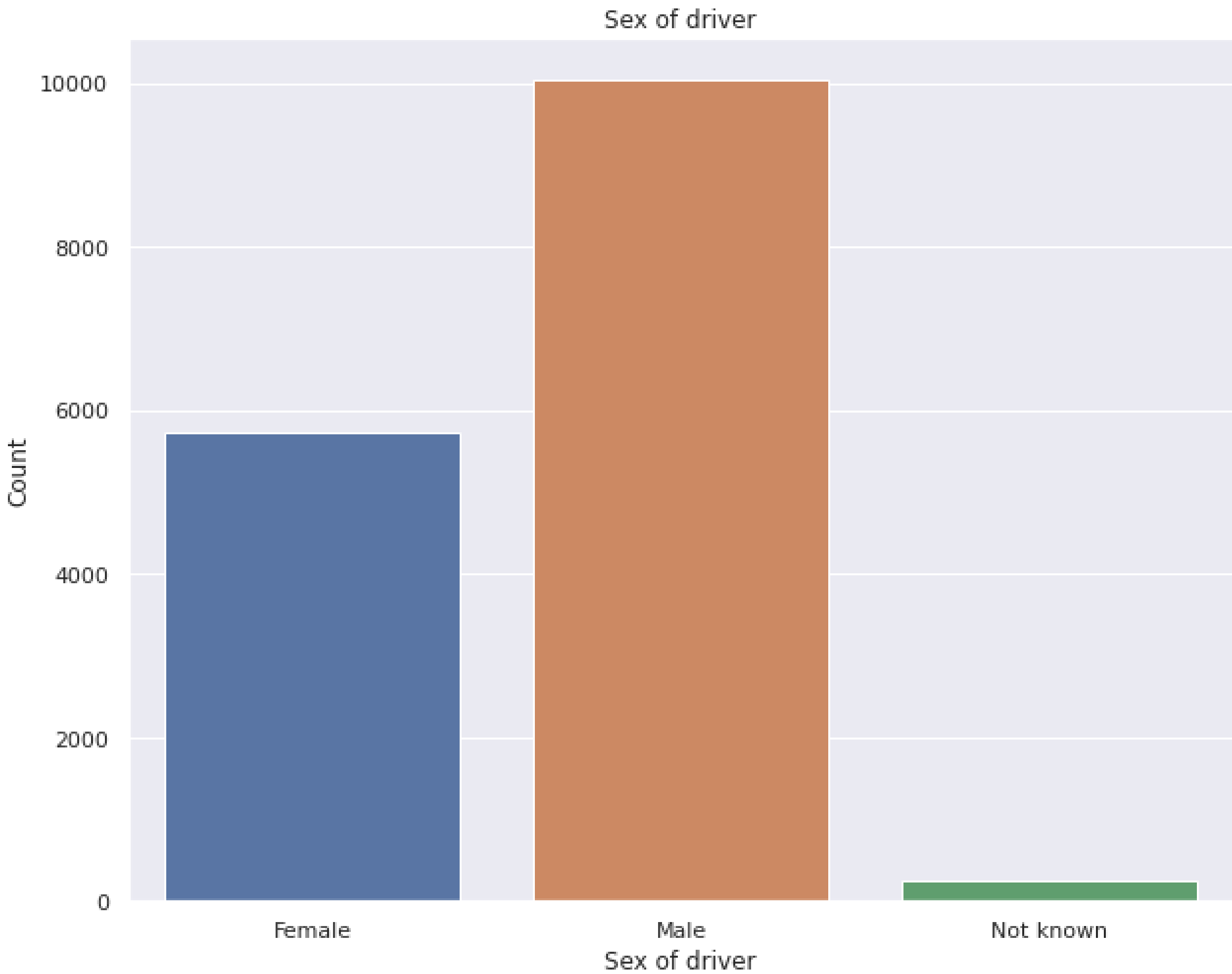
```
#Define plot object, setting figure size and colouring scheme
sns.set_theme(palette="Set3")
sns.set(rc={'figure.figsize':(10,8)})
count = sns.countplot(x=trainset['vehicle_left_hand_drive'])
#Setting graph title
count.set_title('Vehicle is left-drive')
count.set(xlabel = 'Vehicle is left-drive', ylabel = 'Count')
plt.xticks(rotation=0)
#Showing the plot
plt.show()
```



Majority of the cars are not Left handed which makes sense because the UK uses a right hand drive approach.

6.2.3.3 SEX OF DRIVER

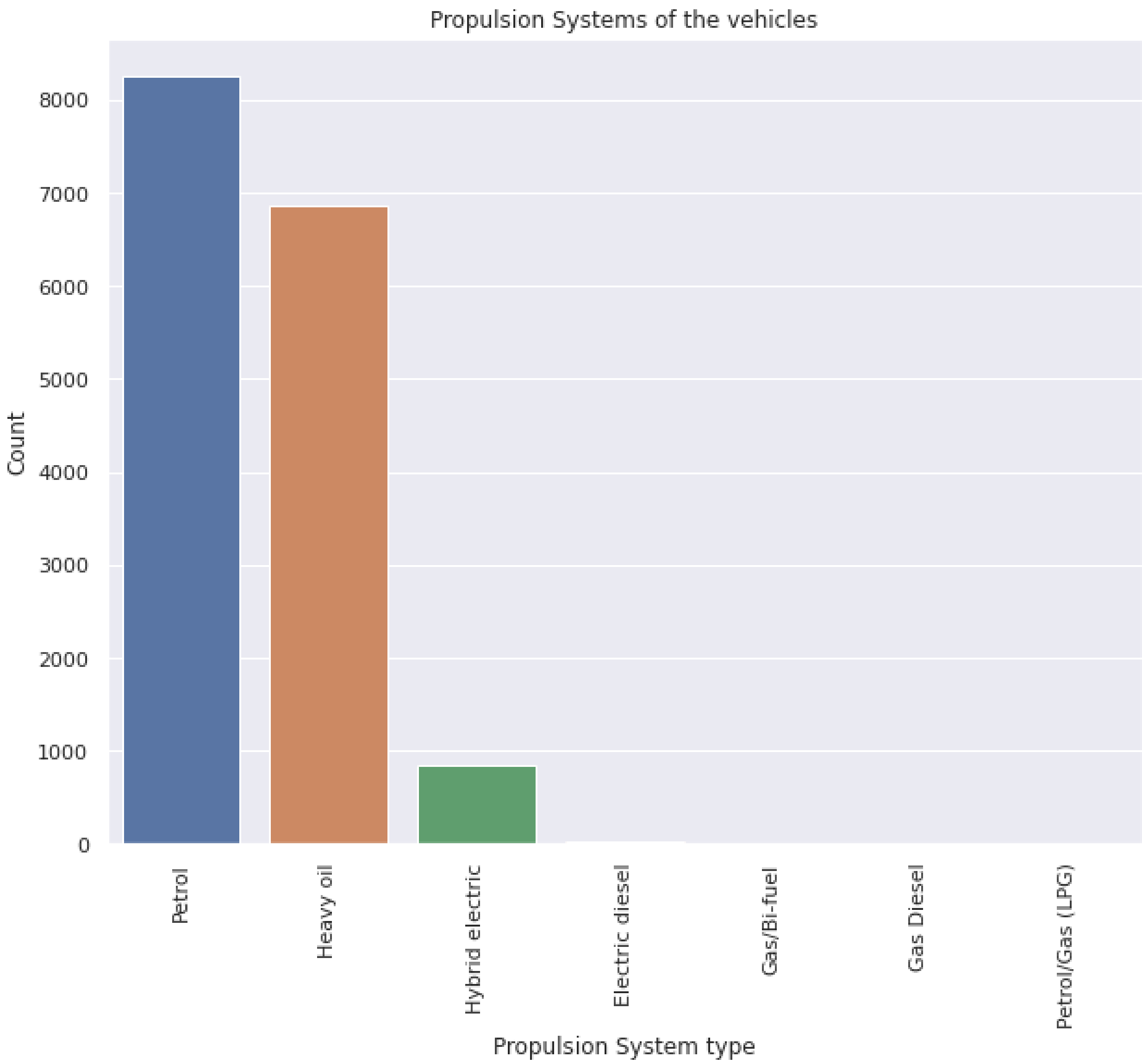
```
#Define plot object, setting figure size and colouring scheme
sns.set_theme(palette="Set3")
sns.set(rc={'figure.figsize':(10,8)})
count = sns.countplot(x = trainset['sex_of_driver'])
#Setting graph title
count.set_title('Sex of driver')
count.set(xlabel = 'Sex of driver', ylabel = 'Count')
plt.xticks(rotation=0)
#Showing the plot
plt.show()
```



The ratio of males and females involved in accidents is 2:1 i.e. males are 2 times more accounatble for the accidents occurence than females.

6.2.3.4 PROPULSION CODE

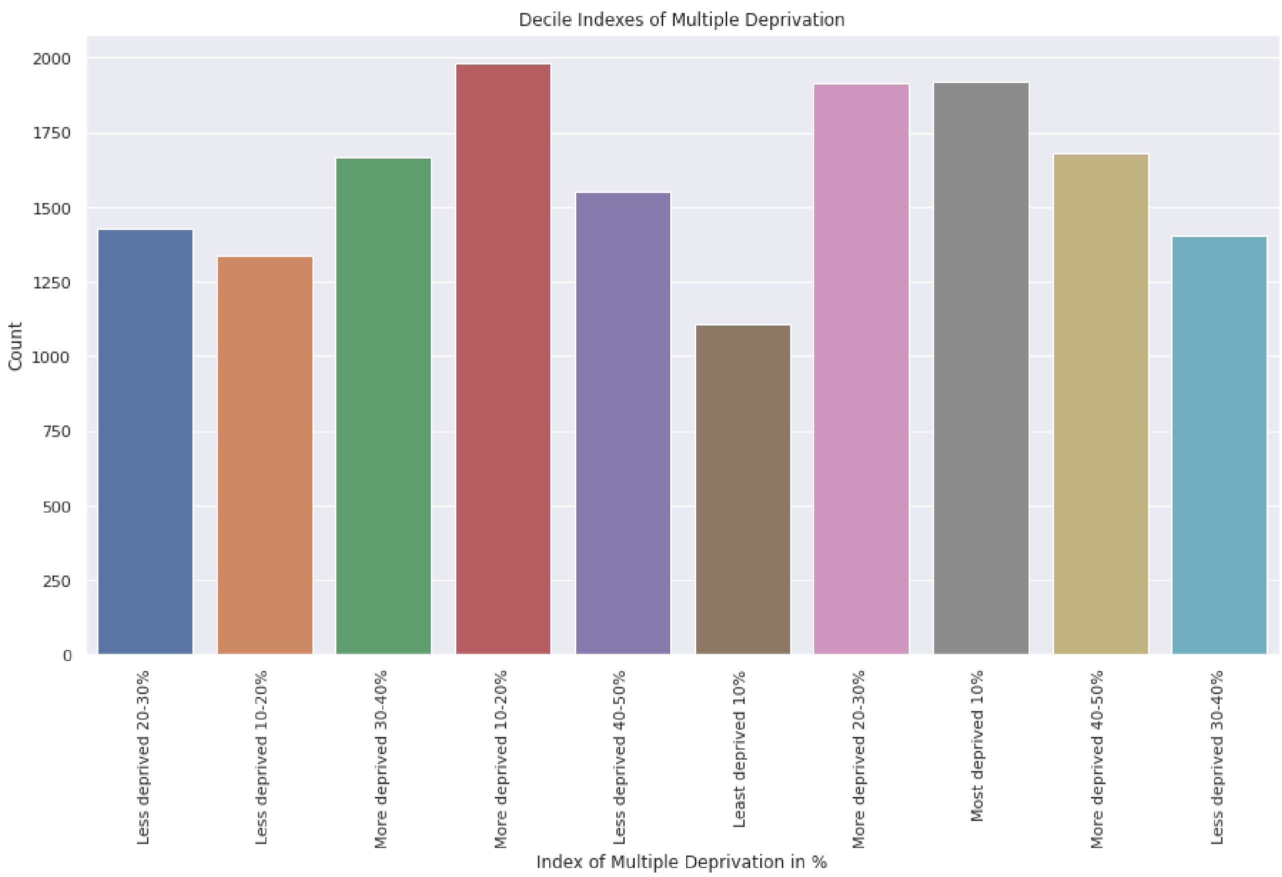
```
#Define plot object, setting figure size and colouring scheme
sns.set_theme(palette="Set3")
sns.set(rc={'figure.figsize':(10,8)})
count = sns.countplot(x=trainset['propulsion_code'])
#Setting graph title
count.set_title('Propulsion Systems of the vehicles')
count.set(xlabel = 'Propulsion System type', ylabel = 'Count')
plt.xticks(rotation=90)
#Showing the plot
plt.show()
```



Petrol and heavy oil cars have a very high count as compared to Hybrid electric cars.

6.2.3.5 DRIVER IMD DECILE

```
#Define plot object, setting figure size and colouring scheme
sns.set_theme(palette="Set3")
sns.set(rc={'figure.figsize':(15,8)})
count = sns.countplot(x= trainset['driver_imd_decile'])
#Setting graph title
count.set_title('Decile Indexes of Multiple Deprivation')
count.set(xlabel = 'Index of Multiple Deprivation in %', ylabel = 'Count')
plt.xticks(rotation=90)
#Showing the plot
plt.show()
```

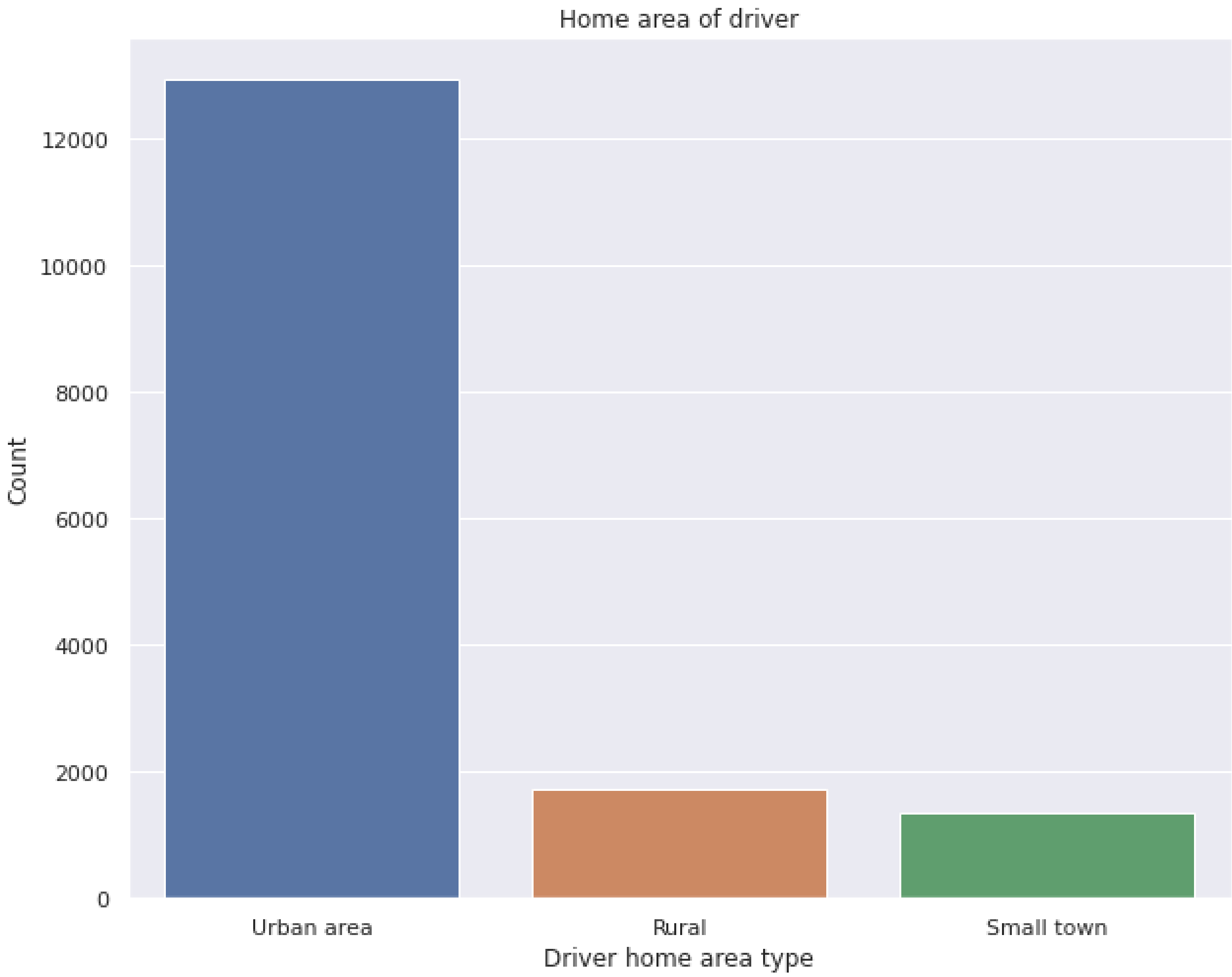


The drivers from MORE DEPRIVED AREA have a higher chance of being involved in accidents as compared to the drivers from the least deprived areas.

6.2.3.6 DRIVER HOME AREA TYPE

```
#Define plot object, setting figure size and colouring scheme
sns.set_theme(palette="Set3")
```

```
sns.set(rc={'figure.figsize':(10,8)})
count = sns.countplot(x = trainset['driver_home_area_type'])
#Setting graph title
count.set_title('Home area of driver')
count.set(xlabel = 'Driver home area type', ylabel = 'Count')
plt.xticks(rotation=0)
#Showing the plot
plt.show()
```



The bar plot clearly highlights that most of the drivers have their homes in URBAN AREAS.In addition, it seems no big difference in classes "rural area" and "small town" so we can merge these 2 categories into one called "Rural" in the 7. Feature transformation.

6.3 BIVARIATE ANALYSIS

Bivariate Analysis is done to observe patterns, trends, or empirical relationships between the 2 variables.

Here, the analysis will be done to understand the predictive power of the independent variables.

It will again be done in 2 parts:

1. Target variable vs Numerical Predictors

2. Target Variable Vs Categorical Predictors

6.3.1 TARGET VARIABLES VS NUMERICAL PREDICTORS

Box plots will be plotted to analyse the relationship between the ACCIDENT SEVERITY and the NUMERICAL PREDICTORS

6.3.1.1 ACCIDENT SEVERITY AND AGE OF DRIVER

```
#plotting boxplot to visualize the relationship between the accident_severity and age_of_c
#Setting figure size
plt.figure(figsize=(12,10), dpi=70)

#Plotting boxplot using seaborn
ax1=sns.boxplot(x='age_of_driver', y='accident_severity', data=trainset )
ax1.set_title('Accident severity and Age of driver relation')
ax1.set_xlabel('Age of driver')
ax1.set_ylabel('Accident severity')

plt.show()
```

The median age of the drivers in all the categories are nearly equal. However, it can be observed that there is a higher spread in the ages of the drivers involved in Fatal accidents. The graph also indicates the presece of some outliers.

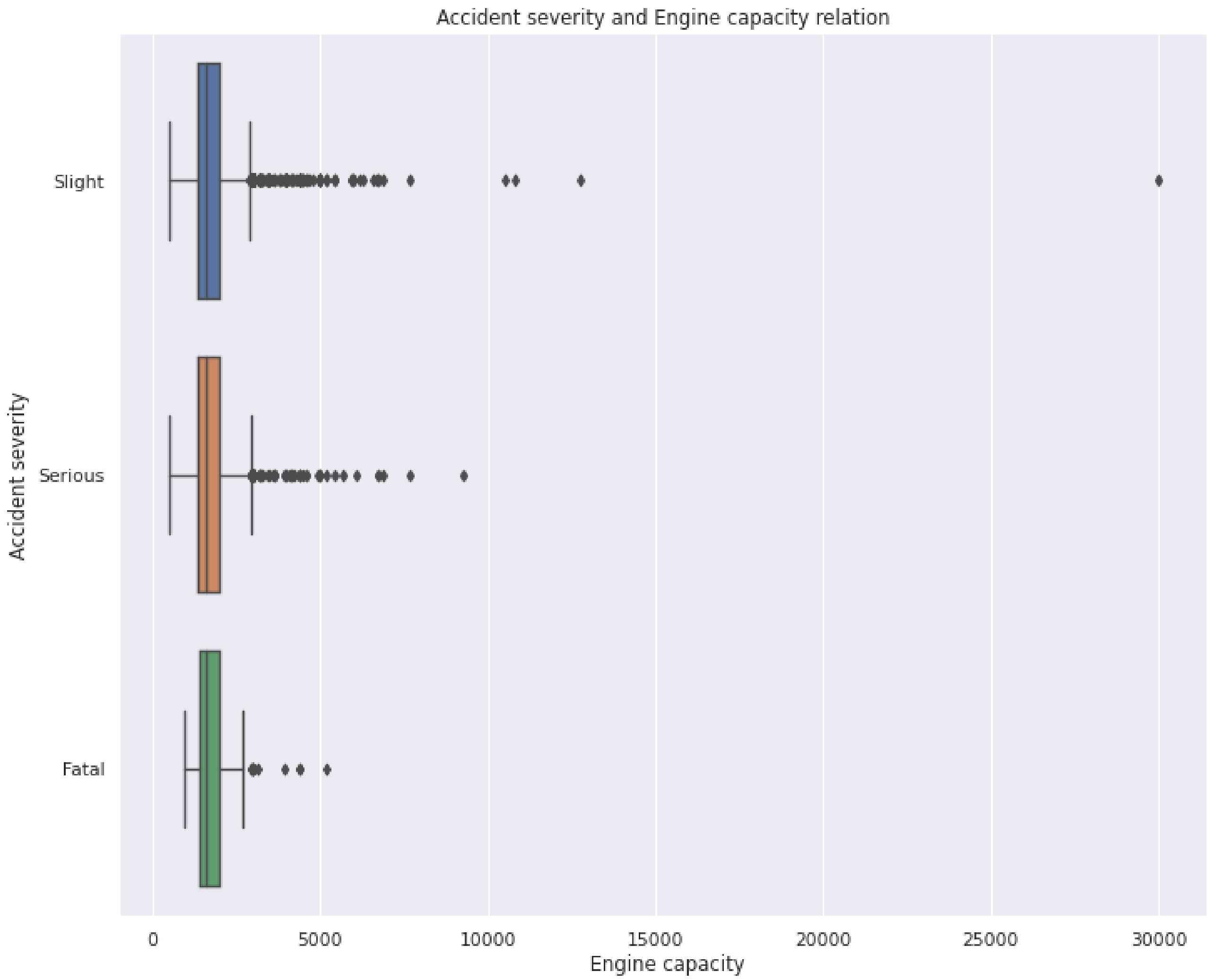
6.3.1.2 ACCIDENT SEVERITY AND AGE OF VEHICLE

```
# plotting box plot to visualise the relationship
#Setting figure size
plt.figure(figsize=(12,10), dpi=70)

#Plotting boxplot using seaborn
ax1=sns.boxplot(x='age_of_vehicle', y='accident_severity', data=trainset)
ax1.set_title('Accident severity and Age of vehicle relation')
ax1.set_xlabel('Age of vehicle')
ax1.set_ylabel('Accident severity')

plt.show()
```

The Median Age of vehicles indicates that older vehicles are more likely to be involved in fatal accidents . The graph also indicates the presece of some outliers.

6.3.1.3 ACCIDENT SEVERITY AND ENGINE CAPACITY

```
#Setting figure size
plt.figure(figsize=(12,10), dpi=70)

#Plotting boxplot using seaborn
ax1=sns.boxplot(x='engine_capacity_cc', y='accident_severity', data=trainset, notch=False)
ax1.set_title('Accident severity and Engine capacity relation')
ax1.set_xlabel(' Engine capacity')
ax1.set_ylabel(' Accident severity ')

plt.show()
```



The median Engine Capacity seems to be highly skewed for the "Slight Category".

6.3.2 TARGET VARIABLE VS CATEGORICAL PREDICTORS

Heat Map will be plotted to understand that whether all the categories of the predictor variable are equally distributed across all the classes of the target variable or not.

Also, **CHI-SQUARE TEST OF INDEPENDENCE** will be carried out to understand that whether the target variable is dependent on the categorical predictor or not which will help us in the process of "**FEATURE ENGINEERING**".

The chi square test will be carried out by creating a contingency table, followed by formulating the NULL AND ALTERNATIVE HYPOTHESIS , where we will try to test the Null Hypothesis for the independence of 2 variables against the alternative hypothesis that the 2 variables are dependent.

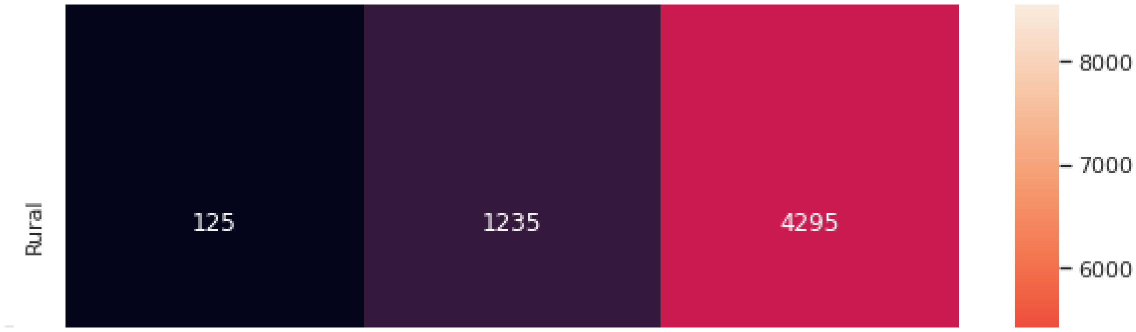
The **LEVEL OF SIGNIFICANCE** is taken as **5%** implying that if $p \leq 0.05$, we will reject the null hypothesis stating that there is association between the target variable and the predictor.

6.3.2.1 ACCIDENT SEVERITY AND ACCIDENT AREA

A heatmap is a graphical representation of data where each value of a matrix is represented as a color.

```
## summarizing the counts into a matrix

ct_counts = trainset.groupby(['accident_severity', 'urban_or_rural_area']).size()
ct_counts = ct_counts.reset_index(name = 'count')
ct_counts = ct_counts.pivot(index = 'urban_or_rural_area', columns = 'accident_severity',
                            values = 'count')


# plotting the heatmap
sns.heatmap(ct_counts, annot=True, fmt = 'd')
```

```
<Axes: xlabel='accident_severity', ylabel='urban_or_rural_area'>
```



Let's compute a frequency table of 2 variables and create a contingency dataframe using crosstab function. For understanding a probability of correlation let's make a c for 2 variables. We want

```
## conducting chi-sq test of independence to check whether the area of accident has any in

#creating a contingency dataframe usins crosstab from pandas for "accident_severity" and '
contingency_1 = pd.crosstab(trainset['accident_severity'], trainset['urban_or_rural_area']
contingency_1

# plotting the above contingency table
axx=contingency_1.plot(kind="bar", stacked = True, rot =0)

# rotating the labels for readability
axx.set_xticklabels(axx.get_xticklabels())
```

```
[Text(0, 0, 'Fatal'), Text(1, 0, 'Serious'), Text(2, 0, 'Slight')]
```

Formulating the hypothesis:

H_0 : Accident severity is independent of area of accident

H_1 : Accident severity is dependent of area of accident

```
import statistics as stats
from scipy.stats import chi2_contingency
```

```
# computing the p value
chi2,p_val,dof,expected = chi2_contingency(contingency_1)
print (f"p-value : {p_val}")
```

```
p-value : 7.214611976896973e-32
```

As p ≤ 0.05 , we reject the H_0 and accept H_1 to conclude that accident_severity is dependent on urban_or_rural_area.

6.3.2.2 ACCIDENT SEVERITY AND VEHICLE LEFT HAND DRIVE

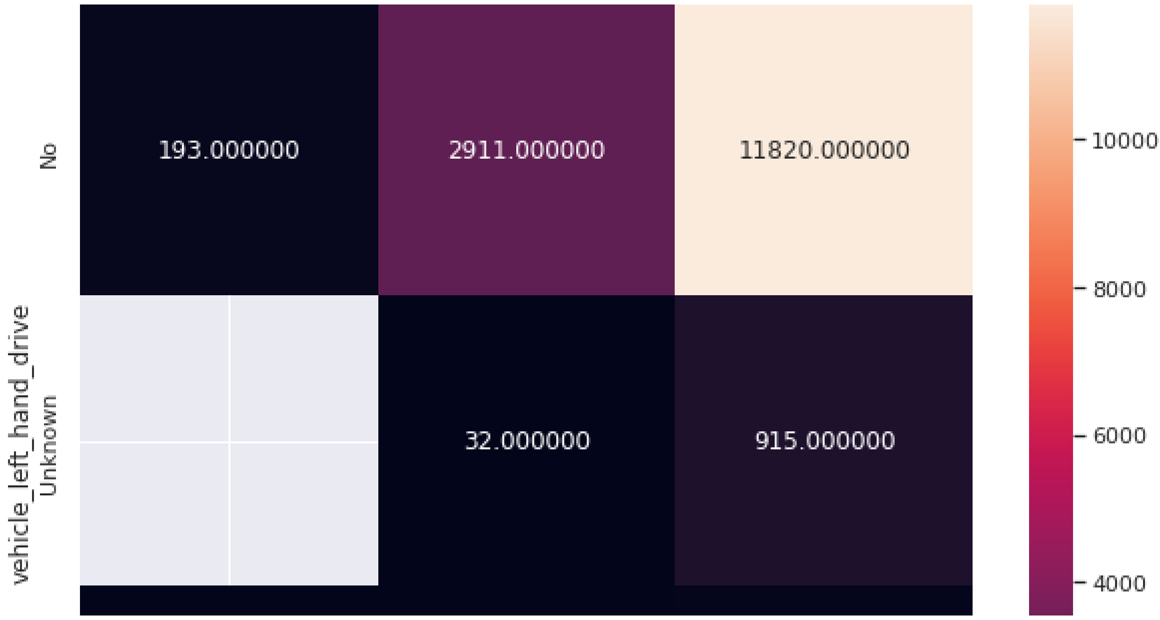
```
ct_counts = trainset.groupby(['accident_severity', 'vehicle_left_hand_drive']).size()
ct_counts = ct_counts.reset_index(name = 'count')
ct_counts = ct_counts.pivot(index = 'vehicle_left_hand_drive', columns = 'accident_severit
                            values = 'count')
```

```
ct_counts
```

accident_severity	Fatal	Serious	Slight
vehicle_left_hand_drive			
No	193.0	2911.0	11820.0
Unknown	NaN	32.0	915.0
Yes	1.0	23.0	105.0

```
sns.heatmap(ct_counts, annot=True, fmt= "f")
```

```
<Axes: xlabel='accident_severity', ylabel='vehicle_left_hand_drive'>
```



Let's compute create a contingency dataframe; for understanding a probability of correlation let's make a chi-sq test of independence for vehicle_left_hand_drive and accident_severity.

```
#create a contingency dataframe usins crosstab from pandas for "accident_severity" and "ve
contingency_2 = pd.crosstab(trainset['accident_severity'], trainset['vehicle_left_hand_dri
contingency_2
# plotting the above contingency table
axx=contingency_2.plot(kind="bar", stacked = True, rot =0)
# rotating the labels for readability
axx.set_xticklabels(axx.get_xticklabels())
```

```
[Text(0, 0, 'Fatal'), Text(1, 0, 'Serious'), Text(2, 0, 'Slight')]
```

vehicle_left_hand_drive

Describing the hypothesis:

H_0: Accident severity is independent of vehicle_left_hand_drive

H_1: Accident severity is dependent of vehicle_left_hand_drive

```
# Computing the p-value of the contingency to check the significance
chi2,p_val,dof,expected = chi2_contingency(contingency_2)
print (f"p-value : {p_val}")
```

```
p-value : 5.946816705902061e-36
```

We can see that p value is significant ($p \leq 0.05$) which means that we reject the H_0 hypothesis of independance and accept H_1 hypothesis and conclude that accident_severity is dependent on vehicle_left_hand_drive.

2000

6.3.2.3 ACCIDENT SEVERITY AND SEX OF DRIVER

0

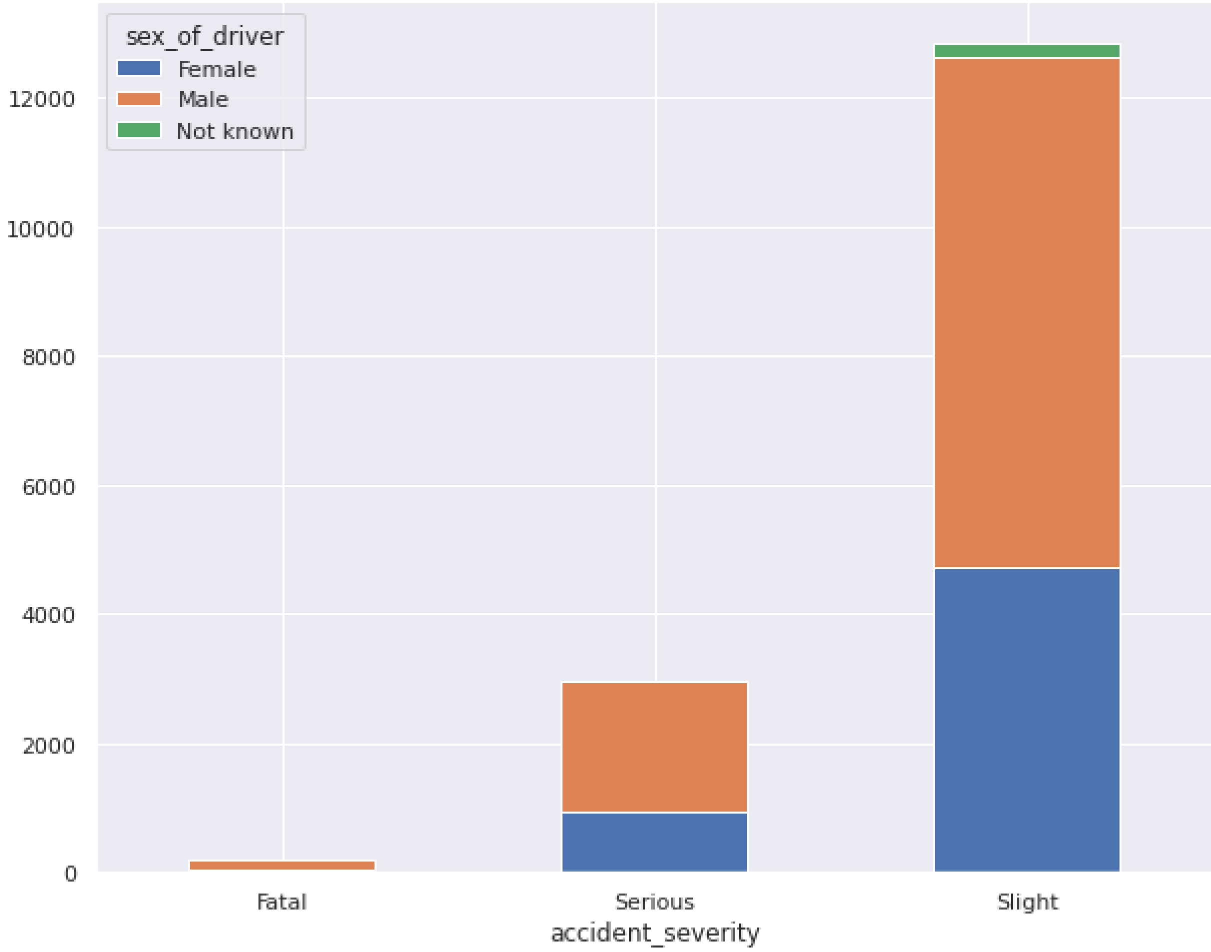
```
ct_counts = trainset.groupby(['accident_severity', 'sex_of_driver']).size()
ct_counts = ct_counts.reset_index(name = 'count')
ct_counts = ct_counts.pivot(index = 'sex_of_driver', columns = 'accident_severity',
                            values = 'count')
sns.heatmap(ct_counts, annot=True, fmt= "d")
```

```
<Axes: xlabel='accident_severity', ylabel='sex_of_driver'>
```

Let's compute create a contingency dataframe; for understanding a probability of correlation let's make a chi-sq test of independence for sex_of_driver and accident_severity.

```
#create a contingency dataframe usins crosstab from pandas for "accident_severity" and "se
contingency_3 = pd.crosstab(trainset['accident_severity'], trainset['sex_of_driver'])
# plotting the above contingency table
axx=contingency_3.plot(kind="bar", stacked = True, rot =0)
# rotating the labels for readability
axx.set_xticklabels(axx.get_xticklabels())
```

```
[Text(0, 0, 'Fatal'), Text(1, 0, 'Serious'), Text(2, 0, 'Slight')]
```



Describing the hypothesis:

H_0: Accident severity is independent of sex_of_driver

H_1: Accident severity is dependent of sex_of_driver

```
# Computing the p-value of the contingency to check the significance
chi2,p_val,dof,expected = chi2_contingency(contingency_3)
print (f"p-value : {p_val}")
```

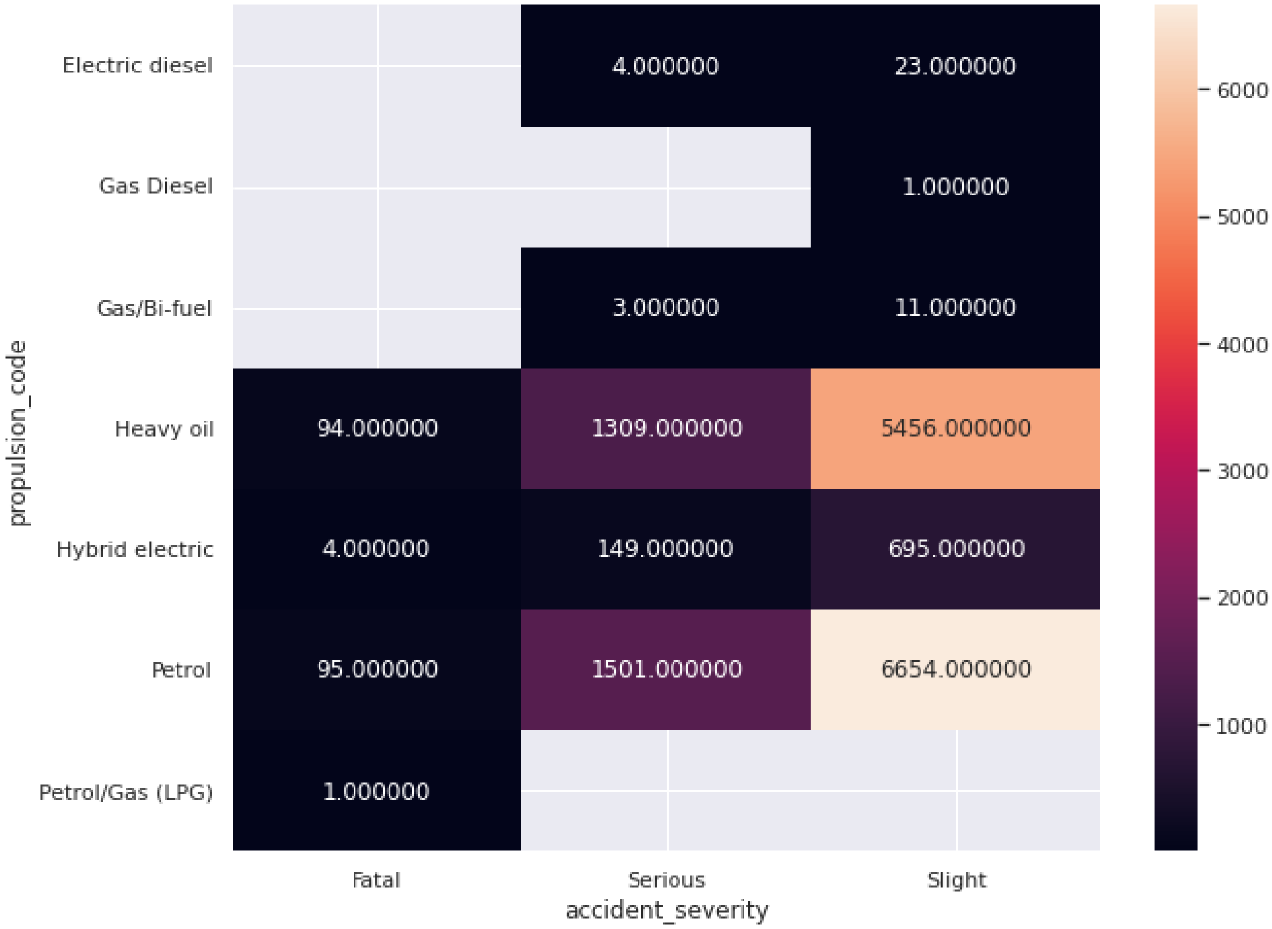
```
p-value : 2.3029944352997045e-11
```

We can see that p value is significant (p ≤ 0.05) which means that we reject the H_0 hypothesis of independance and accept H_1 hypothesis and conclude that accident_severity is dependent on sex_of_driver.

6.3.2.4 ACCIDENT SEVERITY AND PROPULSION CODE

```
ct_counts = trainset.groupby(['accident_severity', 'propulsion_code']).size()
ct_counts = ct_counts.reset_index(name = 'count')
ct_counts = ct_counts.pivot(index = 'propulsion_code', columns = 'accident_severity',
                            values = 'count')
sns.heatmap(ct_counts, annot=True, fmt= "f")
```

```
<Axes: xlabel='accident_severity', ylabel='propulsion_code'>
```

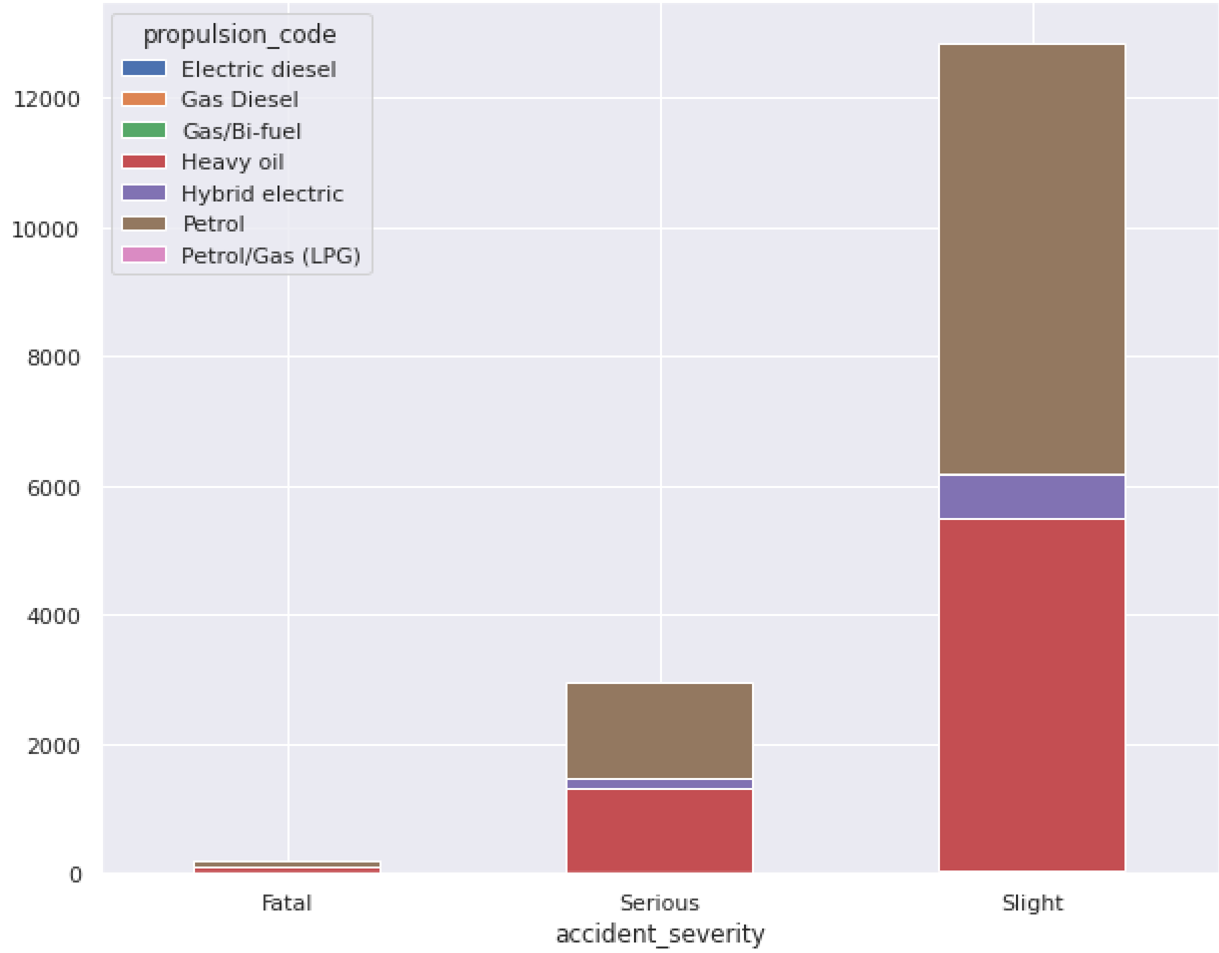


Let's compute create a contingency dataframe; for understanding a probability of correlation let's make a chi-sq test of independence for propulsion_code and accident_severity.

```
#create a contingency dataframe usins crosstab from pandas for "accident_severity" and "pr
contingency_4 = pd.crosstab(trainset['accident_severity'], trainset['propulsion_code'])
# plotting the above contingency table
axx=contingency_4.plot(kind="bar", stacked = True, rot =0)
```

```
# rotating the labels for readability
axx.set_xticklabels(axx.get_xticklabels())
```

```
[Text(0, 0, 'Fatal'), Text(1, 0, 'Serious'), Text(2, 0, 'Slight')]
```



Describing the hypothesis:

H_0: Accident severity is independent of propulsion_code

H_1: Accident severity is dependent of propulsion_code

```
# Computing the p-value of the contingency to check the significance
chi2,p_val,dof,expected = chi2_contingency(contingency_4)
print (f"p-value : {p_val}")
```

```
p-value : 3.115049561097119e-14
```

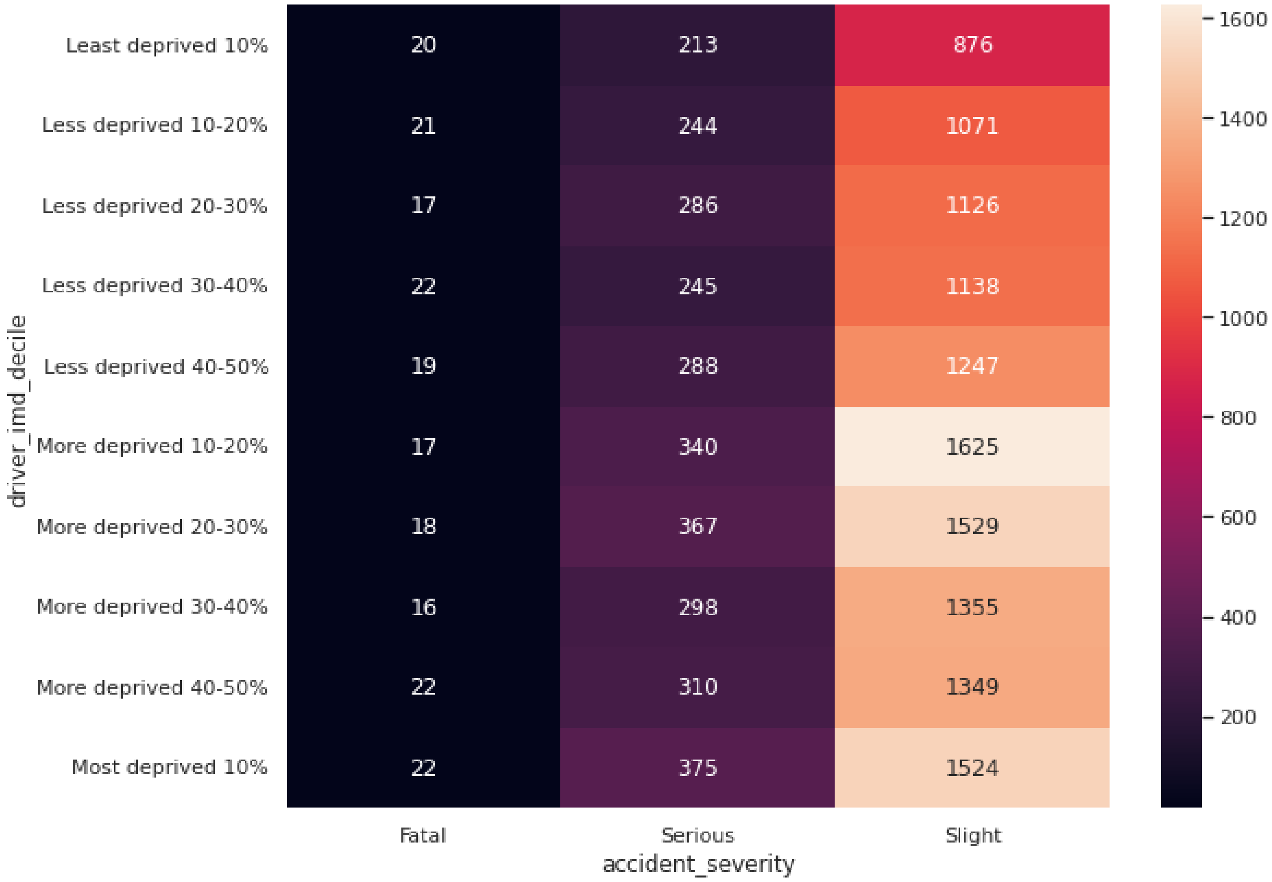
We can see that p value is significant ($p \leq 0.05$) which means that we reject the H_0 hypothesis of independance and accept H_1 hypothesis and conclude that accident_severity is dependent on propulsion_code.

6.3.2.5 ACCIDENT SEVERITY AND DRIVER IMD DECILE

```
ct_counts = trainset.groupby(['accident_severity', 'driver_imd_decile']).size()
ct_counts = ct_counts.reset_index(name = 'count')
```

```
ct_counts = ct_counts.pivot(index = 'driver_imd_decile', columns = 'accident_severity',
                            values = 'count')
sns.heatmap(ct_counts, annot=True, fmt= 'd')
```

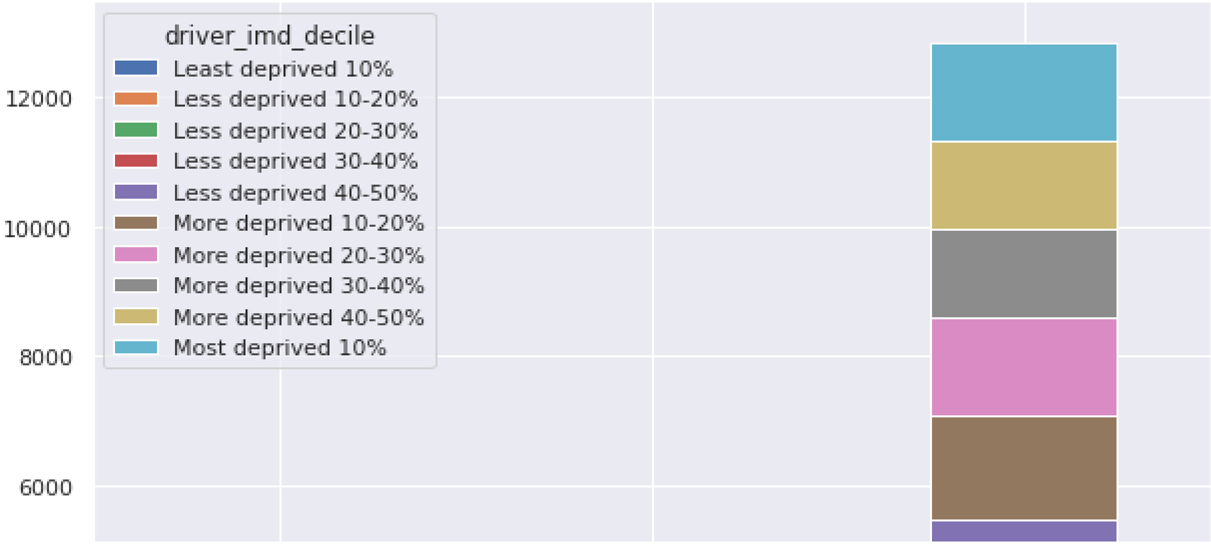
```
<Axes: xlabel='accident_severity', ylabel='driver_imd_decile'>
```



Let's compute create a contingency dataframe; for understanding a probability of correlation let's make a chi-sq test of independence for driver_imd_decile and accident_severity.

```
#create a contingency dataframe usins crosstab from pandas for "accident_severity" and "dr
contingency_5 = pd.crosstab(trainset['accident_severity'], trainset['driver_imd_decile'])
# plotting the above contingency table
axx=contingency_5.plot(kind="bar", stacked = True, rot =0)
# rotating the labels for readability
axx.set_xticklabels(axx.get_xticklabels())
```

```
[Text(0, 0, 'Fatal'), Text(1, 0, 'Serious'), Text(2, 0, 'Slight')]
```



Describing the hypothesis:

H_0: Accident severity is independent of driver_imd_decile

H_1: Accident severity is dependent of driver_imd_decile

```
# Computing the p-value of the contingency to check the significance
chi2,p_val,dof,expected = chi2_contingency(contingency_5)
print (f"p-value : {p_val}")
```

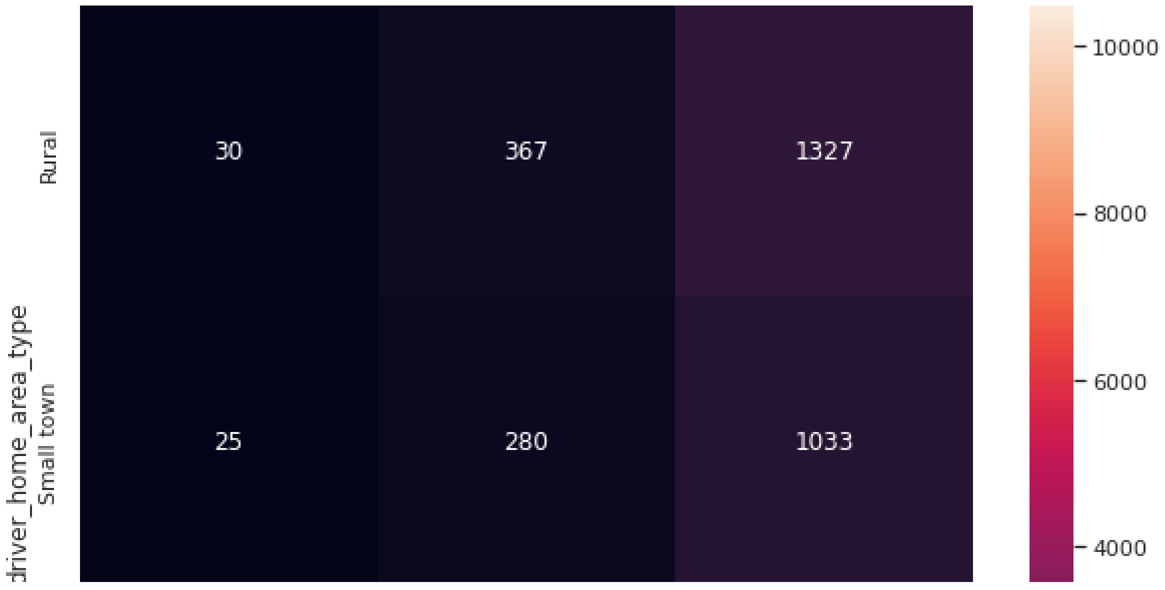
```
p-value : 0.388563761821275
```

We can see that p value is not significant (p > 0.05) which means that the null hypothesis is true. We accept the H_0 hypothesis of independance and conclude that accident_severity is independent on driver_imd_decile.

6.3.2.6 ACCIDENT SEVERITY AND DRIVER HOME AREA TYPE

```
ct_counts = trainset.groupby(['accident_severity', 'driver_home_area_type']).size()
ct_counts = ct_counts.reset_index(name = 'count')
ct_counts = ct_counts.pivot(index = 'driver_home_area_type', columns = 'accident_severity'
                            values = 'count')
sns.heatmap(ct_counts, annot=True, fmt= "d")
```

```
<Axes: xlabel='accident_severity', ylabel='driver_home_area_type'>
```



Let's compute create a contingency dataframe; for understanding a probability of correlation let's make a chi-sq test of independence for driver_home_area_type and accident_severity.


```
#create a contingency dataframe usins crosstab from pandas for "accident_severity" and "dr
contingency_6 = pd.crosstab(trainset['accident_severity'], trainset['driver_home_area_type
# plotting the above contingency table
axx=contingency_6.plot(kind="bar", stacked = True, rot =0)
# rotating the labels for readability
axx.set_xticklabels(axx.get_xticklabels())
```

```
[Text(0, 0, 'Fatal'), Text(1, 0, 'Serious'), Text(2, 0, 'Slight')]
```

Describing the hypothesis:

H_0: Accident severity is independent of driver_home_area_type

H_1: Accident severity is dependent of driver_home_area_type

```
# Computing the p-value of the contingency to check the significance
chi2,p_val,dof,expected = chi2_contingency(contingency_6)
print (f"p-value : {p_val}")
```

```
p-value : 6.824584518479814e-06
```

We can see that p value is significant ($p \leq 0.05$) which means that we reject the H_0 hypothesis of independance and accept H_1 hypothesis and conclude that accident_severity is dependent on driver_home_area_type.

6.4 OBSERVATIONS

From the above visualisations and the tests of independence , we have observed that Driver_IMD Decile and Accident Severity are not dependent on each other.

Apart from that there are some categories of a specific variable which have very few or no values.

Also, the presence of outliers in some variables were highlighted.

The above mentioned points will be handled in the "FEATURE ENGINEERING ".

7.FEATURE ENGINEERING

The process of extracting features from data and converting them into forms compatible with machine learning algorithms is known as feature engineering.

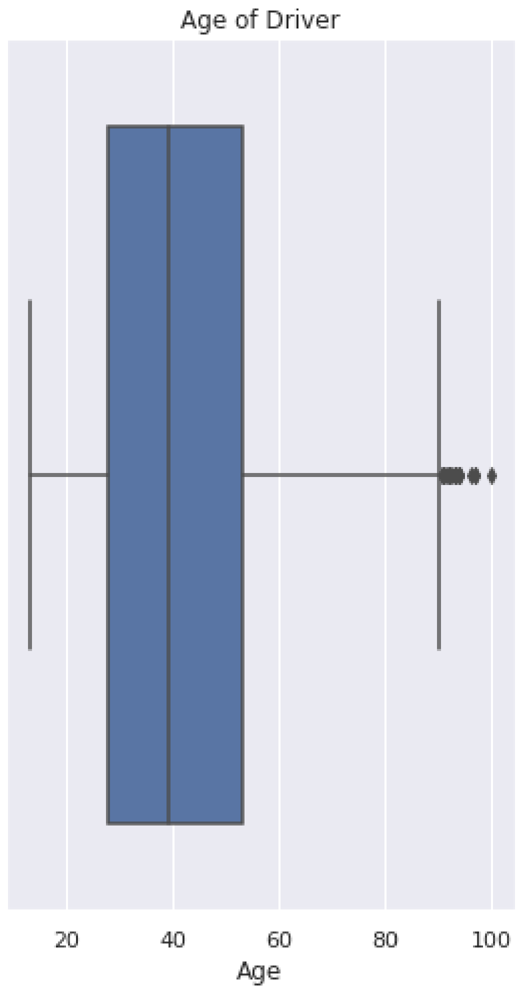
7.1 HANDLING OUTLIERS

The outliers can negatively impact the accuracy of out Machine learing model by skewing the data and leading to incorrect predictions. Hence, we will remove them before training the model.

7.1.1 AGE OF DRIVER

```
#checking the outlier for age of driver
#plotting a boxplot
plt.subplot(1,2,2)
```

```
#Define plot object
box = sns.boxplot(x= trainset['age_of_driver'])
#Setting graph title
box.set_title('Age of Driver')
box.set(xlabel = 'Age')
#Showing the plot
plt.show()
```



The box plot above shows the outliers for Age of driver, it's seen mostly after the age of 80. Let's calculate the count.

```
trainset[trainset['age_of_driver'] > 80].shape[0]
```

```
368
```

```
trainset[trainset['age_of_driver'] > 80].shape[0]* 100 / len(trainset)
```

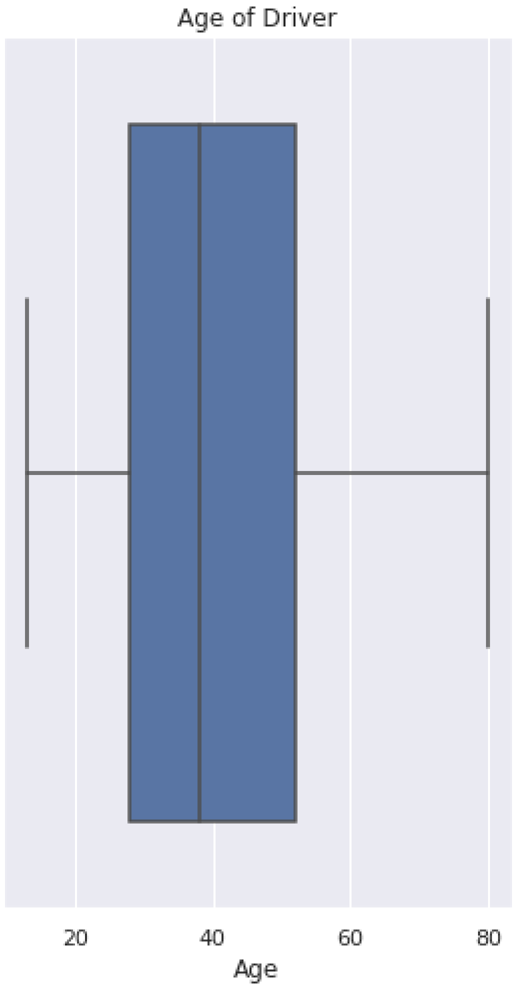
```
2.3
```

2.3% of the data has the outlier, so we will remove it to avoid any skewness in our data.

```
trainset.drop(trainset[trainset['age_of_driver'] > 80].index, inplace = True)
trainset.shape
```

```
(15632, 10)
```

```
#plotting a boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x= trainset['age_of_driver'])
#Setting graph title
box.set_title('Age of Driver')
box.set(xlabel = 'Age')
#Showing the plot
plt.show()
```



We see that the outliers for Age of Driver is handled.

Let's do the same manipulations we did on training set on the training dataset to prevent bias.

```
testset.drop(testset[testset['age_of_driver'] > 80].index, inplace = True)
```

7.1.2 ENGINE CAPACITY

```
#plotting a boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x = trainset['engine_capacity_cc'])
#Setting graph title
box.set(xlabel = 'Engine Capacity')
#Showing the plot
plt.show()
```


```
trainset[trainset['engine_capacity_cc'] > 2500].shape[0]
```

```
869
```

```
trainset[trainset['engine_capacity_cc'] > 2500].shape[0]* 100 / len(trainset)
```

```
5.559109518935517
```

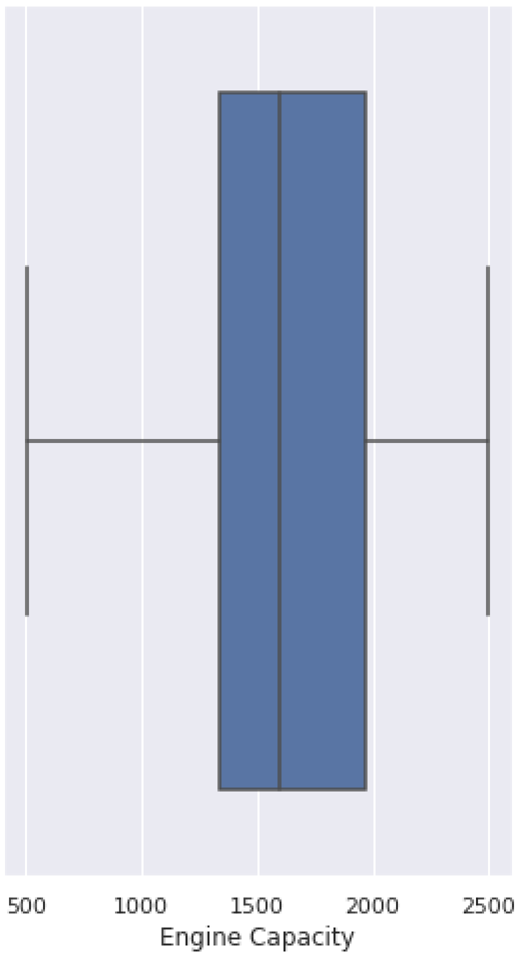
We have 5.559% of outliers, which we will handle by removing.

```
trainset.drop(trainset[trainset['engine_capacity_cc'] > 2500].index, inplace = True)
trainset.shape
```

```
(14763, 10)
```

```
#plotting a boxplot
```

```
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x = trainset['engine_capacity_cc'])
#Setting graph title
box.set(xlabel = 'Engine Capacity')
#Showing the plot
plt.show()
```



We removed Engine Capacity outliers.

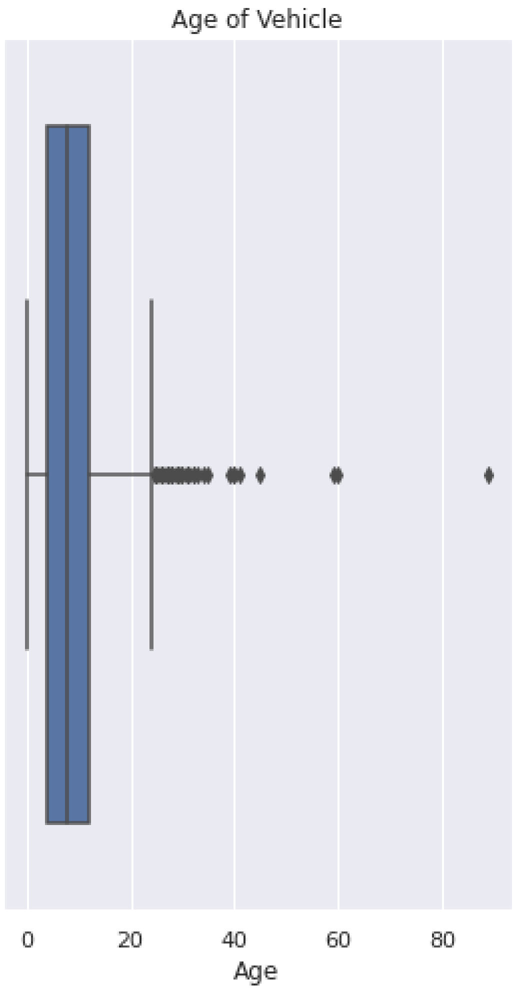
Let's do the same manipulations we did on training set on the training dataset to prevent bias.

```
testset.drop(testset[testset['engine_capacity_cc'] > 2500].index, inplace = True)
```

7.1.3 AGE OF VEHICLE

```
# plotting a boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x = trainset['age_of_vehicle'])
#Setting graph title
box.set_title('Age of Vehicle')
box.set(xlabel = 'Age')
```

```
#Showing the plot
plt.show()
```



The Ageof Vehicle has many outliers over the age 20.

```
trainset[trainset['age_of_vehicle'] > 20].shape[0]
```

```
165
```

```
trainset[trainset['age_of_vehicle'] > 20].shape[0]* 100 / len(trainset)
```

```
1.1176590123958545
```

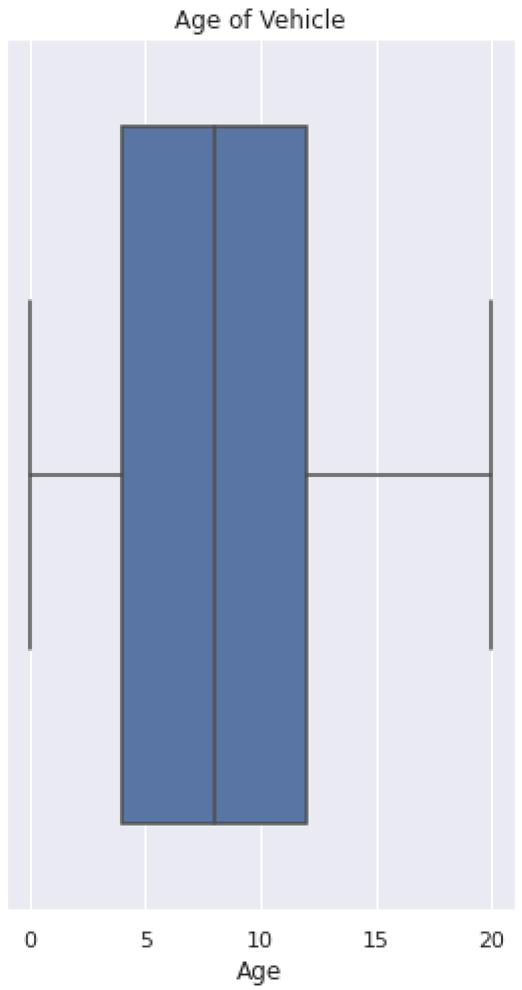
Total of 1.117% of data has age over 20 which are classified as outliers and we will remove them.

```
trainset.drop(trainset[trainset['age_of_vehicle'] > 20].index, inplace = True)
trainset.shape
```

```
(14598, 10)
```

```
# plotting a boxplot
plt.subplot(1,2,2)
```

```
#Define plot object
box = sns.boxplot(x = trainset['age_of_vehicle'])
#Setting graph title
box.set_title('Age of Vehicle')
box.set(xlabel = 'Age')
#Showing the plot
plt.show()
```



Let's do the same manipulations we did on training set on the training dataset to prevent bias.

```
testset.drop(testset[testset['age_of_vehicle'] > 20].index, inplace = True)
```

7.2 DIMENSIONALITY REDUCTION

In order to decrease the number of characteristics (dimensions) in the data with the least amount of important information lost, dimensionality reduction will be used.

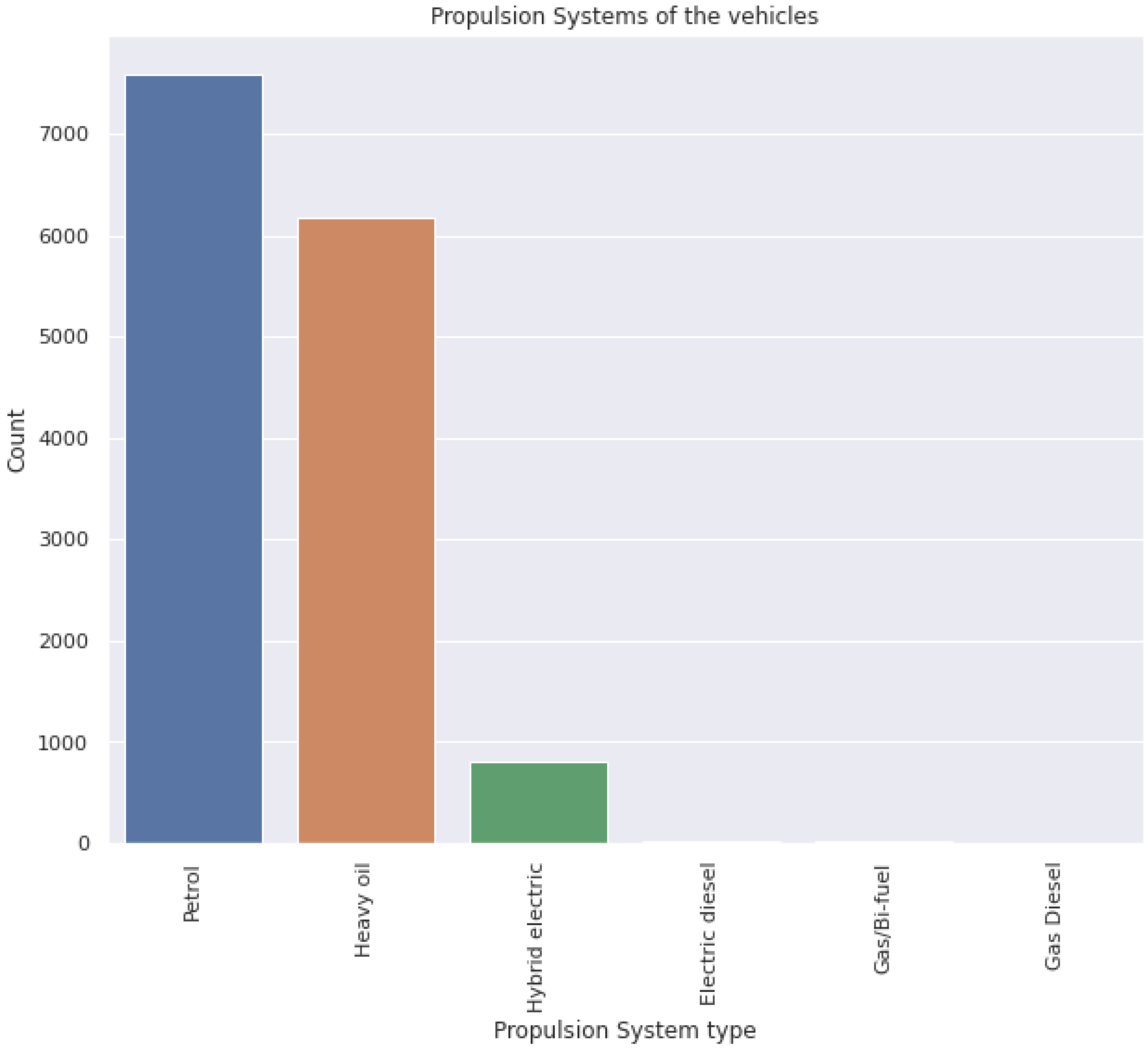
Here we will merge or drop some categories of the variables which have very few/no observations as comapred to the other categories.

7.2.1 PROPULSION CODE

Propulsion code has 7 different categories where "Electric diesel", "Gas/Bi-fuel", "Gas Diesel" and "Petrol/Gas (LPG)" has fewer values compared with the remaining 3 categories i.e "Petrol", "Heavy Oil" and "Hybrid Electric".

Although "Hybrid Electric" is underrepresented in comparison to the other categories, we will retain this category as this is a new technology and a future ecological trend so we believe people will tend to buy more hybrid cars in the future. So, we will keep this category to have more insight into this category.

```
#Define plot object, setting figure size and colouring scheme
sns.set_theme(palette="Set3")
sns.set(rc={'figure.figsize':(10,8)})
count = sns.countplot(x=trainset['propulsion_code'])
#Setting graph title
count.set_title('Propulsion Systems of the vehicles')
count.set(xlabel = 'Propulsion System type', ylabel = 'Count')
plt.xticks(rotation=90)
#Showing the plot
plt.show()
```



```
# count of each unique value in the "propulsion_code" column
print(trainset['propulsion_code'].value_counts())
```

```
Petrol             7590
Heavy oil          6169
Hybrid electric     806
Electric diesel      21
Gas/Bi-fuel          11
Gas Diesel            1
Name: propulsion_code, dtype: int64
```

The last three categories will be removed from both training and testing dataset.

```
#Let's delete Electric diesel, Electric diesel, Gas Diesel from train and test sets
outliers=trainset[(trainset['propulsion_code']=='Electric diesel') | (trainset['propulsior
trainset = trainset.drop(outliers, axis=0)   #axis 0 is for dropping rows
outliers1=testset[(testset['propulsion_code']=='Electric diesel') | (testset['propulsion_c
testset = testset.drop(outliers1, axis=0)   #axis 0 is for dropping rows
```

```
# count of each unique value in the "propulsion_code" column in train set
print(trainset['propulsion_code'].value_counts())
```

```
Petrol             7590
Heavy oil          6169
Hybrid electric     806
Gas/Bi-fuel          11
Name: propulsion_code, dtype: int64
```

7.2.2 VEHICLE LEFT HAND DRIVE

As we found out before (in the 6.3.1.2) for the column vehicle_left_hand_drive there is a category where the type of vehicle is not known. We can drop this category because it does not have logical sense for the analysis and we can say that this category is a missing data.

```
# count of each unique value in the "vehicle_left_hand_drive" column
print(trainset['vehicle_left_hand_drive'].value_counts())
```

```
No         13575
Unknown      879
Yes          122
Name: vehicle_left_hand_drive, dtype: int64
```

There are 760 vehicles where the type of vehicle is unknown. Let's delete categoty "Unknown" from training and test sets:

```
outliers2=trainset[(trainset['vehicle_left_hand_drive']=='Unknown')].index
trainset = trainset.drop(outliers2, axis=0)   #axis 0 is for dropping rows
outliers3=testset[(testset['vehicle_left_hand_drive']=='Unknown')].index
testset = testset.drop(outliers3, axis=0)   #axis 0 is for dropping rows
```

Let's check the categories in the train set:

```
# count of each unique value in the "vehicle_left_hand_drive" column
print(trainset['vehicle_left_hand_drive'].value_counts())
```

```
No     13575
Yes      122
Name: vehicle_left_hand_drive, dtype: int64
```

7.2.3 SEX OF DRIVER

As we found out before (in the 6.3.1.3) for the column sex_of_driver there is a category where the age of driver is not known. This category will be dropped as it cant be merged with any other category.

```
# count of each unique value in the "sex_of_driver" column
print(trainset['sex_of_driver'].value_counts())
```

```
Male         8480
Female       5163
Not known      54
Name: sex_of_driver, dtype: int64
```

There are 54 observations where the age is unknown. Let's delete categoty "Not known" from training and test sets:

```
outliers4=trainset[(trainset['sex_of_driver']=='Not known')].index
trainset = trainset.drop(outliers4, axis=0)   #axis 0 is for dropping rows
outliers5=testset[(testset['sex_of_driver']=='Not known')].index
testset = testset.drop(outliers5, axis=0)   #axis 0 is for dropping rows
```

Let's check the categories in the train set where we can see 2 classes:

```
# count of each unique value in the "sex_of_driver" column
print(trainset['sex_of_driver'].value_counts())
```

```
Male      8480
Female    5163
Name: sex_of_driver, dtype: int64
```

7.2.4 DRIVER HOME AREA TYPE

The "Small Town " category will be merged into rural area as there are a very few observation for that category.

```
trainset['driver_home_area_type'] = trainset['driver_home_area_type'].replace({'Small towr
testset['driver_home_area_type'] = testset['driver_home_area_type'].replace({'Small town':
trainset['driver_home_area_type'].value_counts()
```

```
Urban area    10980
Rural          2663
Name: driver_home_area_type, dtype: int64
```

7.3 FEATURE SELECTION

The process of feature selection involves choosing the characteristics that best describe the association between an independent variable and the target variable.

7.3.1 DRIVER IMD DECILE

The chi-squared test highlighted the independence of accident severity and driver_imd_decile but before excluding the column it will be better to check that whether the merging of categories will make any difference in the test or not. All the categories belonging to less or least type will be categorised as less and the rest will be categorised as more.

```
# counting the unique values
trainset['driver_imd_decile'].value_counts()
```

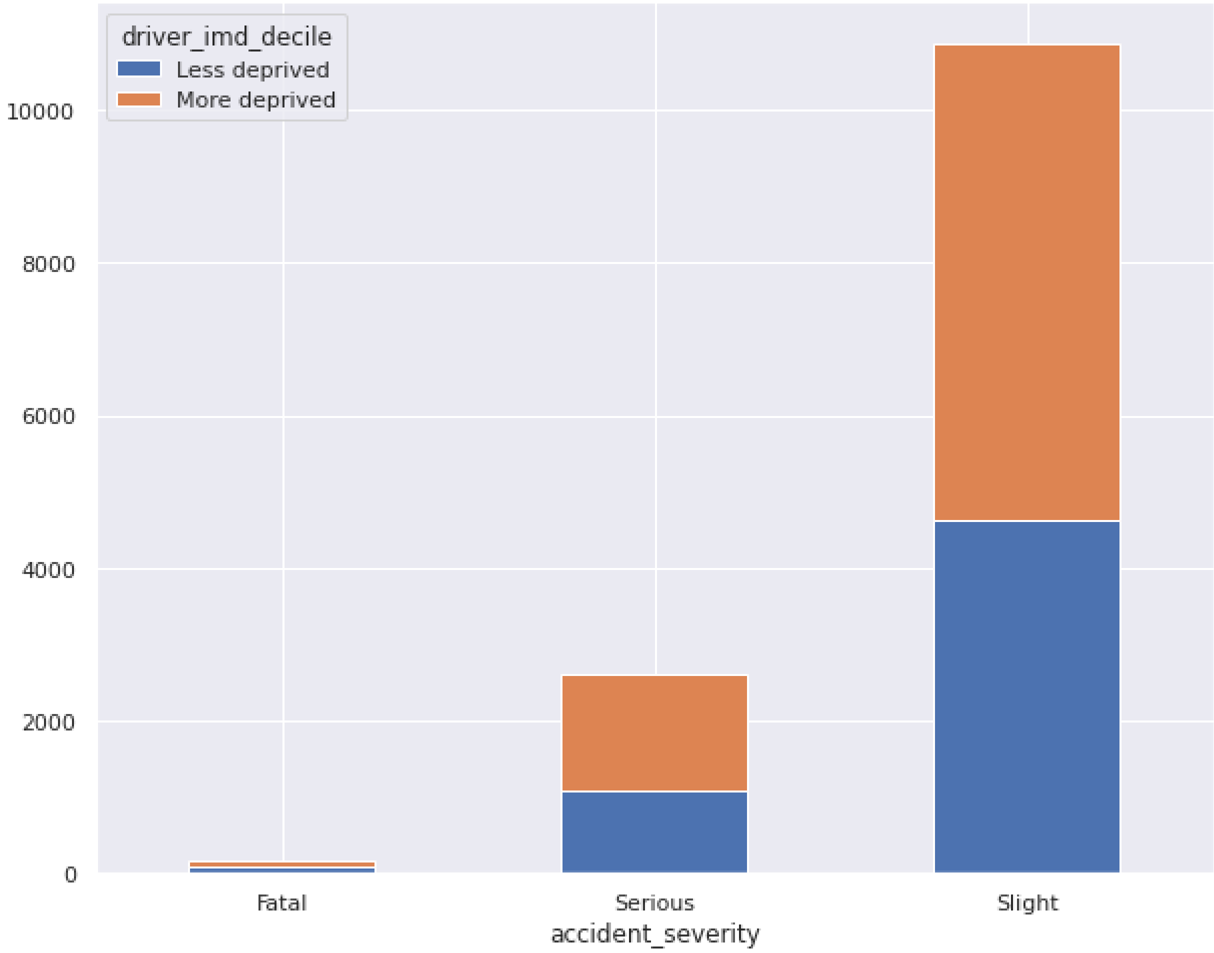
```
Most deprived 10%       1747
More deprived 10-20%    1686
More deprived 20-30%    1601
More deprived 40-50%    1427
More deprived 30-40%    1377
Less deprived 40-50%    1324
Less deprived 20-30%    1223
Less deprived 30-40%    1178
Less deprived 10-20%    1149
Least deprived 10%       931
Name: driver_imd_decile, dtype: int64
```

```
# merging the categories in the training set
#use replace function to change classes in train set
trainset['driver_imd_decile'] = trainset['driver_imd_decile'].replace({'Most deprived 10%'
                                                                       'More deprived 10-2
                                                                       'More deprived 20-3
                                                                       'More deprived 30-4
                                                                       'More deprived 40-5
trainset['driver_imd_decile'] = trainset['driver_imd_decile'].replace({'Least deprived 10%
                                                                       'Less deprived 10-2
                                                                       'Less deprived 20-3
                                                                       'Less deprived 30-4
                                                                       'Less deprived 40-5

#checkig the unique value counts
trainset['driver_imd_decile'].value_counts()
```

```
More deprived    7838
Less deprived    5805
Name: driver_imd_decile, dtype: int64
```

```
# CONDUCTING CHI SQUARE TEST OF INDEPENDENCE ON THE MERGED CATEGORIES.
#create a contingency dataframe using crosstab from pandas for "accident_severity" and "dr
contingency_8 = pd.crosstab(trainset['accident_severity'], trainset['driver_imd_decile'])
# plotting the above contingency table
axx=contingency_8.plot(kind="bar", stacked = True, rot =0)
# rotating the labels for readability
axx.set_xticklabels(axx.get_xticklabels())
```

```
[Text(0, 0, 'Fatal'), Text(1, 0, 'Serious'), Text(2, 0, 'Slight')]
```



In accordance to the barplot we can say that there are balanced distrinbution in beteween target and independent variable. Let's repeat the chi-test with 2 classes in the driver_imd_decile to be sure that it is independent from
our target. Describing the hypothesis:

H_0: Accident severity is independent of driver_imd_decile

H_1: Accident severity is dependent of driver_imd_decile

```
# Computing the p-value of the contingency to check the significance
chi2,p_val,dof,expected = chi2_contingency(contingency_8)
print (f"p-value : {p_val}")
```

```
p-value : 0.11578548480989866
```

As p>0.05, we reject the null hypothesis and conclude that accident_severity is independent of driver_imd_decile. Thus, it will be dropped from the data frame.

```
# dropping the DRIVER IMD DECILE column from test and train set.
del trainset['driver_imd_decile']
del testset['driver_imd_decile']
```

```
#validating the dataframe
trainset.shape
```

```
(13643, 9)
```

7.4 FEATURE TRANSFORMATION

It means transforming our original feature to the functions of original features

A dummy variable is a numerical value used to represent categorical data. It is necessary to make them to incorporate qualitative information into analysis and model building.

We create dummy variables for 5 categorical variables in training dataset and apply the same transformations to test data.

```
from sklearn.preprocessing import OneHotEncoder
one_hot_encoder = OneHotEncoder(drop="first", sparse=False)

# categorical columns to transform
cat_cols = ["urban_or_rural_area","vehicle_left_hand_drive","sex_of_driver","propulsion_cc

# fit an encoder and transform the **trainset**
cat_vals = trainset[cat_cols].to_numpy()
transformed = one_hot_encoder.fit_transform(cat_vals)

# the names of the new columns are the unique values of the categorical variables
new_col_names = one_hot_encoder.get_feature_names_out(cat_cols)

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

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

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868: Futur
  warnings.warn(
```

	accident_severity	urban_or_rural_area	vehicle_left_hand_drive	sex_of_driv
44923	Slight	Urban	No	Fem
110490	Slight	Urban	No	Fem
56293	Slight	Urban	No	M

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

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

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

	accident_severity	urban_or_rural_area	vehicle_left_hand_drive	sex_of_drive
19933	Slight	Urban	No	Ma
51085	Slight	Urban	No	Ma
41521	Slight	Rural	No	Ma
64094	Slight	Rural	No	Ma
84495	Slight	Urban	No	Fema

```
trainset.drop(columns=cat_cols, inplace=True)
testset.drop(columns=cat_cols, inplace=True)
```

We deleted the original columns from both testing and traing set.

```
trainset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13643 entries, 44923 to 75967
Data columns (total 11 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   accident_severity               13643 non-null  object
 1   age_of_driver                   13643 non-null  float64
 2   engine_capacity_cc              13643 non-null  float64
 3   age_of_vehicle                  13643 non-null  float64
 4   urban_or_rural_area_Urban       13643 non-null  float64
 5   vehicle_left_hand_drive_Yes     13643 non-null  float64
 6   sex_of_driver_Male              13643 non-null  float64
```

```
 7   propulsion_code_Heavy oil          13643 non-null  float64
 8   propulsion_code_Hybrid electric    13643 non-null  float64
 9   propulsion_code_Petrol             13643 non-null  float64
 10  driver_home_area_type_Urban area   13643 non-null  float64
dtypes: float64(10), object(1)
memory usage: 1.2+ MB
```

8.EXPORTING THE DATASET

```
#Exporting train and test sets to csv files
trainset.to_csv('/trainset.csv')
testset.to_csv('/testset.csv')


#Check if it was saved - reading files as a dataframe
check1 = pd.read_csv('/trainset.csv')
check2 = pd.read_csv('/testset.csv')

#printing the shape of files and the trainset
print('The shape of train set is ', check1.shape, 'The shape of test set is ', check2.shap
print('The first rows of training set:')
check1.head(5)
```

```
The shape of train set is  (13643, 12) The shape of test set is  (3414, 12)
The first rows of training set:
```

	Unnamed: 0	accident_severity	age_of_driver	engine_capacity_cc	age_of_vehicle	u
0	44923	Slight	42.0	1360.0	10.0	
1	110490	Slight	74.0	1200.0	4.0	
2	56293	Slight	50.0	1560.0	10.0	
3	127729	Slight	18.0	1870.0	18.0	
4	42831	Slight	33.0	1248.0	5.0	

All the preprocessing steps were done in this part . The main takeaway from this part is that there is a very high class imbalance which can alter the model's performance.

```
**REFERENCES:**
1. Pekar, V. (2022). Big Data for Decision M
examples and exercises. (Version 1.0.0). URL
com/vpekar/bd4dm
2.
```

REFERENCES:

1. Pekar, V. (2022). Big Data for Decision Making. Lecture examples and exercises. (Version 1.0.0). URL: https://github.com/vpekar/bd4dm

2.