road-acc-analysis-excel

March 31, 2023

1 Problem Statement

The goal of this project is to predict the accident severity based on various features such as number of vehicles, number of casualties, day of the week, time of the day, weather conditions, road surface conditions, etc. The accident severity is an important parameter that can help the authorities take measures to reduce the number of accidents and improve road safety.

Based on the given dataset, we can build a machine learning model that can predict the accident severity based on the historical data. We can use various classification algorithms such as logistic regression, decision trees, random forest, etc. to build the model. Once the model is trained, we can use it to predict the accident severity for new instances based on the features of the accident. The performance of the model can be evaluated using various metrics such as accuracy, precision, recall, etc.

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[2]:
```
!unzip -q "/content/drive/MyDrive/Sri/road_acc_3/Data_excel.zip"
```

```
replace Data_excel/test_set_clean.xlsx? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
```

[1]:
```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# To scale the data using z-score
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

# Algorithms to use
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```python
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

from sklearn.linear_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

# Metrics to evaluate the model
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪precision_recall_curve

# For tuning the model
from sklearn.model_selection import GridSearchCV

# To ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```python
# Reading the dataset
df_train = pd.read_excel('/content/Data_excel/train_set_clean.xlsx')
df_test = pd.read_excel('/content/Data_excel/test_set_clean.xlsx')
```

[3]:
```python
df_train.head().T
```

[3]:
```
                                                 0              1  \
accident_index                       2020122001268  2019140895070
accident_year                             0.815922       0.109792
accident_reference                       122001268      140895070
accident_severity                                0              0
number_of_vehicles                               0              0
number_of_casualties                             0              0
date                                    03/11/2020     04/11/2019
day_of_week                              -0.587847      -1.110776
time                                         17:45          13:11
local_authority_highway                  E10000023      E08000019
road_type                                        0              0
speed_limit                                     30             30
junction_detail                                  6              1
junction_control                                 2              4
pedestrian_crossing_human_control                0              0
pedestrian_crossing_physical_facilities          5              0
light_conditions                                 4              1
weather_conditions                               1              2
road_surface_conditions                          1              2
special_conditions_at_site                       0              0
carriageway_hazards                              0              0
urban_or_rural_area                              1              1
```

```
did_police_officer_attend_scene_of_accident             2              1
lsoa_of_accident_location                       E01027909      E01007866

                                                        2              3  \
accident_index                              2019360832516  2019121901482
accident_year                                    0.109792       0.109792
accident_reference                              360832516      121901482
accident_severity                                       0              0
number_of_vehicles                                      0              0
number_of_casualties                                    0              0
date                                           05/04/2019     26/10/2019
day_of_week                                      0.980941        1.50387
time                                                12:30          10:30
local_authority_highway                         E10000020      E06000014
road_type                                               0              0
speed_limit                                            60             30
junction_detail                                         3              3
junction_control                                        4              4
pedestrian_crossing_human_control                       0              0
pedestrian_crossing_physical_facilities                 0              0
light_conditions                                        1              1
weather_conditions                                      1              2
road_surface_conditions                                 1              2
special_conditions_at_site                              0              0
carriageway_hazards                                     0              0
urban_or_rural_area                                     2              2
did_police_officer_attend_scene_of_accident             1              2
lsoa_of_accident_location                       E01026876      E01013438

                                                        4
accident_index                              2021161018349
accident_year                                    1.522051
accident_reference                              161018349
accident_severity                                       0
number_of_vehicles                                      0
number_of_casualties                                    0
date                                           28/01/2021
day_of_week                                      0.458012
time                                                12:49
local_authority_highway                         E06000010
road_type                                               0
speed_limit                                            30
junction_detail                                         3
junction_control                                        4
pedestrian_crossing_human_control                       0
pedestrian_crossing_physical_facilities                 0
light_conditions                                        1
```

```
weather_conditions                                      1
road_surface_conditions                                 1
special_conditions_at_site                              0
carriageway_hazards                                     0
urban_or_rural_area                                     1
did_police_officer_attend_scene_of_accident             2
lsoa_of_accident_location                       E01012872
```

[4]:
```
df_train.shape,df_test.shape
```

[4]:
```
((260000, 24), (64987, 24))
```

[5]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260000 entries, 0 to 259999
Data columns (total 24 columns):
 #   Column                                       Non-Null Count   Dtype
---  ------                                       --------------   -----
 0   accident_index                               260000 non-null  object
 1   accident_year                                260000 non-null  float64
 2   accident_reference                           260000 non-null  object
 3   accident_severity                            260000 non-null  int64
 4   number_of_vehicles                           260000 non-null  int64
 5   number_of_casualties                         260000 non-null  int64
 6   date                                         260000 non-null  object
 7   day_of_week                                  260000 non-null  float64
 8   time                                         260000 non-null  object
 9   local_authority_highway                      260000 non-null  object
 10  road_type                                    260000 non-null  int64
 11  speed_limit                                  260000 non-null  int64
 12  junction_detail                              260000 non-null  int64
 13  junction_control                             260000 non-null  int64
 14  pedestrian_crossing_human_control            260000 non-null  int64
 15  pedestrian_crossing_physical_facilities      260000 non-null  int64
 16  light_conditions                             260000 non-null  int64
 17  weather_conditions                           260000 non-null  int64
 18  road_surface_conditions                      260000 non-null  int64
 19  special_conditions_at_site                   260000 non-null  int64
 20  carriageway_hazards                          260000 non-null  int64
 21  urban_or_rural_area                          260000 non-null  int64
 22  did_police_officer_attend_scene_of_accident  260000 non-null  int64
 23  lsoa_of_accident_location                    260000 non-null  object
dtypes: float64(2), int64(16), object(6)
memory usage: 47.6+ MB
```

[6]:
```python
# Checking the number of unique values in each column
df_train.nunique()
```

[6]:
```
accident_index                                  260000
accident_year                                        5
accident_reference                              259140
accident_severity                                    1
number_of_vehicles                                   1
number_of_casualties                                 1
date                                              1826
day_of_week                                          7
time                                              1440
local_authority_highway                            208
road_type                                            1
speed_limit                                          6
junction_detail                                     10
junction_control                                     6
pedestrian_crossing_human_control                    4
pedestrian_crossing_physical_facilities              7
light_conditions                                     5
weather_conditions                                   9
road_surface_conditions                              6
special_conditions_at_site                           9
carriageway_hazards                                  7
urban_or_rural_area                                  3
did_police_officer_attend_scene_of_accident          3
lsoa_of_accident_location                        31638
dtype: int64
```

- It seems that our target column severity needs to be replaced as it has 0 everyone so for the intuition of machine learning we will try to replace the value in this column with 3 severity values

[7]:
```python
# loop through each row of the dataframe and replace the accident_severity␣
↪value if it's zero
for index, row in df_train.iterrows():
    if row['accident_severity'] == 0:
        df_train.at[index, 'accident_severity'] = np.random.randint(1, 4)
```

[8]:
```python
df_train.accident_severity.value_counts()
```

[8]:
```
2    86828
1    86797
3    86375
Name: accident_severity, dtype: int64
```

Dropping all the unrequired columns

[9]:
```
# removing columns like indexes that would lead to overfit and also removing␣
↪columns which have a unique category over the whole notebook
df_train = df_train.
↪drop(['accident_index','accident_reference','date','time','local_authority_highway','lsoa_o
↪=1)
# removing columns like indexes that would lead to overfit and also removing␣
↪columns which have a unique category over the whole notebook
df_test = df_test.
↪drop(['accident_index','accident_reference','date','time','local_authority_highway','lsoa_o
↪=1)
```

[10]:
```
df_train.head().T
```

[10]:
```
                                                0          1          2  \
accident_year                            0.815922   0.109792   0.109792
accident_severity                        1.000000   1.000000   2.000000
day_of_week                             -0.587847  -1.110776   0.980941
speed_limit                             30.000000  30.000000  60.000000
junction_detail                          6.000000   1.000000   3.000000
junction_control                         2.000000   4.000000   4.000000
pedestrian_crossing_human_control        0.000000   0.000000   0.000000
pedestrian_crossing_physical_facilities  5.000000   0.000000   0.000000
light_conditions                         4.000000   1.000000   1.000000
weather_conditions                       1.000000   2.000000   1.000000
road_surface_conditions                  1.000000   2.000000   1.000000
special_conditions_at_site               0.000000   0.000000   0.000000
carriageway_hazards                      0.000000   0.000000   0.000000
urban_or_rural_area                      1.000000   1.000000   2.000000
did_police_officer_attend_scene_of_accident  2.000000   1.000000   1.000000

                                                3          4
accident_year                            0.109792   1.522051
accident_severity                        3.000000   3.000000
day_of_week                              1.503870   0.458012
speed_limit                             30.000000  30.000000
junction_detail                          3.000000   3.000000
junction_control                         4.000000   4.000000
pedestrian_crossing_human_control        0.000000   0.000000
pedestrian_crossing_physical_facilities  0.000000   0.000000
light_conditions                         1.000000   1.000000
weather_conditions                       2.000000   1.000000
road_surface_conditions                  2.000000   1.000000
special_conditions_at_site               0.000000   0.000000
carriageway_hazards                      0.000000   0.000000
urban_or_rural_area                      2.000000   1.000000
did_police_officer_attend_scene_of_accident  2.000000   2.000000
```

It seems day of week and accident_year are given in some different encoded format so we will try to keep them and use as required.

[11]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260000 entries, 0 to 259999
Data columns (total 15 columns):
 #   Column                                       Non-Null Count   Dtype  
---  ------                                       --------------   -----  
 0   accident_year                                260000 non-null  float64
 1   accident_severity                            260000 non-null  int64  
 2   day_of_week                                  260000 non-null  float64
 3   speed_limit                                  260000 non-null  int64  
 4   junction_detail                              260000 non-null  int64  
 5   junction_control                             260000 non-null  int64  
 6   pedestrian_crossing_human_control            260000 non-null  int64  
 7   pedestrian_crossing_physical_facilities      260000 non-null  int64  
 8   light_conditions                             260000 non-null  int64  
 9   weather_conditions                           260000 non-null  int64  
 10  road_surface_conditions                      260000 non-null  int64  
 11  special_conditions_at_site                   260000 non-null  int64  
 12  carriageway_hazards                          260000 non-null  int64  
 13  urban_or_rural_area                          260000 non-null  int64  
 14  did_police_officer_attend_scene_of_accident  260000 non-null  int64  
dtypes: float64(2), int64(13)
memory usage: 29.8 MB
```

Andd by looking into the data, it seems, - - accident_year: This column has only 5 unique values, which suggests that it might actually be categorical data rather than numerical data. - day_of_week: This column has 7 unique values, which suggests that it might actually be categorical data rather than numerical data. - accident_severity: This column already has the int64 datatype, which is appropriate for numerical data. However, if it's supposed to be categorical data (e.g. indicating the severity level of an accident as "fatal", "serious", or "minor"), - Similarly it seems all the other columns are categorical columns seeing the unique values so they can be converted into category

[12]:
```
df_train['accident_year'] = df_train['accident_year'].astype('category')
df_train['day_of_week'] = df_train['day_of_week'].astype('category')
df_train['accident_severity'] = df_train['accident_severity'].astype('category')

df_train['pedestrian_crossing_human_control'] =␣
 ↪df_train['pedestrian_crossing_human_control'].astype('category')
df_train['pedestrian_crossing_physical_facilities'] =␣
 ↪df_train['pedestrian_crossing_physical_facilities'].astype('category')
df_train['light_conditions'] = df_train['light_conditions'].astype('category')
df_train['weather_conditions'] = df_train['weather_conditions'].
 ↪astype('category')
```

```
df_train['special_conditions_at_site'] = df_train['special_conditions_at_site'].
  ↪astype('category')
df_train['carriageway_hazards'] = df_train['carriageway_hazards'].
  ↪astype('category')
df_train['did_police_officer_attend_scene_of_accident'] =␣
  ↪df_train['did_police_officer_attend_scene_of_accident'].astype('category')
```

[13]:
```
df_test['accident_year'] = df_test['accident_year'].astype('category')
df_test['day_of_week'] = df_test['day_of_week'].astype('category')
df_test['accident_severity'] = df_test['accident_severity'].astype('category')

df_test['pedestrian_crossing_human_control'] =␣
  ↪df_test['pedestrian_crossing_human_control'].astype('category')
df_test['pedestrian_crossing_physical_facilities'] =␣
  ↪df_test['pedestrian_crossing_physical_facilities'].astype('category')
df_test['light_conditions'] = df_test['light_conditions'].astype('category')
df_test['weather_conditions'] = df_test['weather_conditions'].astype('category')
df_test['special_conditions_at_site'] = df_test['special_conditions_at_site'].
  ↪astype('category')
df_test['carriageway_hazards'] = df_test['carriageway_hazards'].
  ↪astype('category')
df_test['did_police_officer_attend_scene_of_accident'] =␣
  ↪df_test['did_police_officer_attend_scene_of_accident'].astype('category')
```

1.0.1 EDA

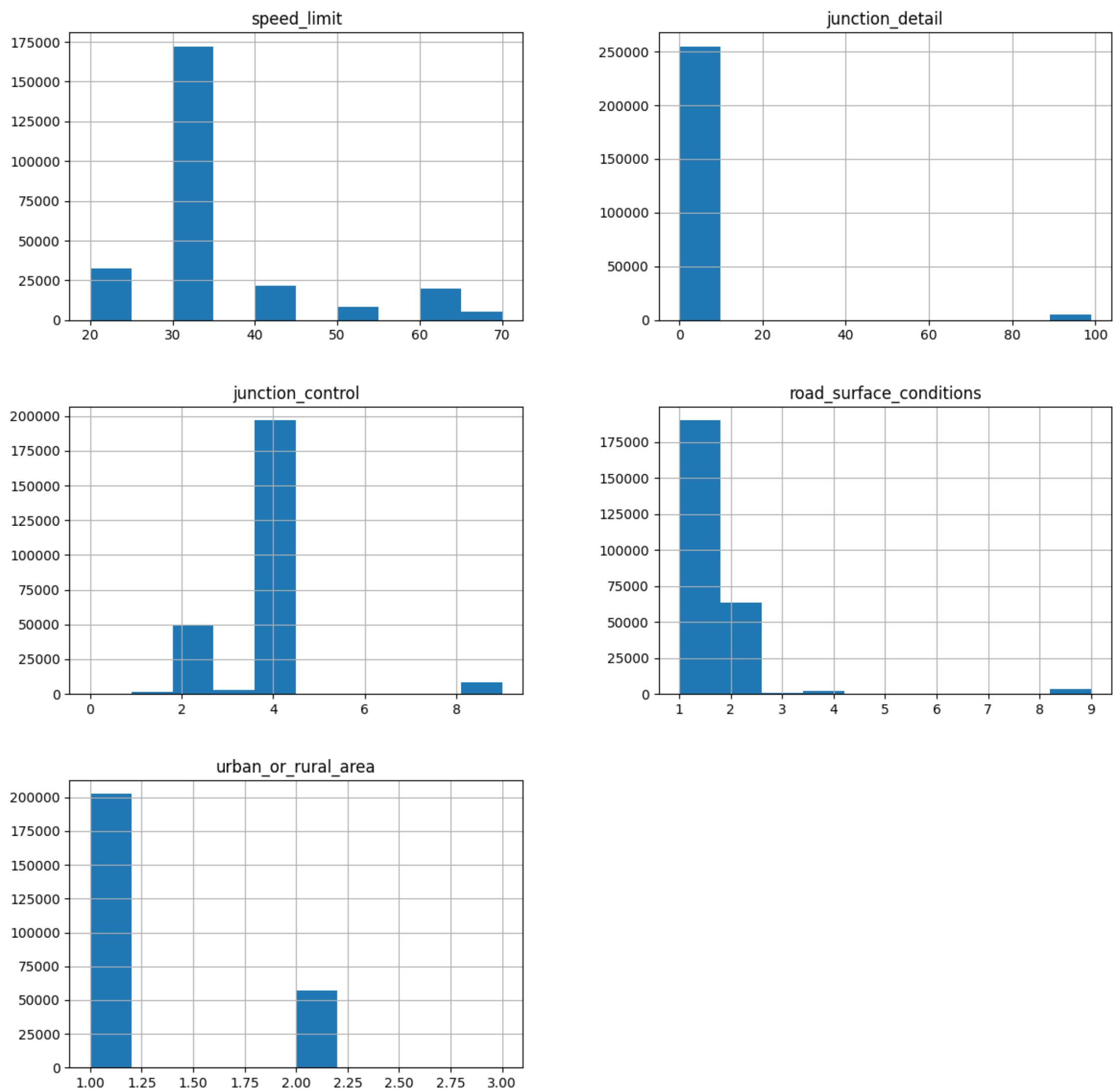
[14]:
```
num_cols = ['speed_limit', 'junction_detail', 'junction_control',␣
  ↪'road_surface_conditions', 'urban_or_rural_area']
cat_cols =␣
  ↪['accident_year','day_of_week','accident_severity','pedestrian_crossing_human_control','ped
            'weather_conditions',␣
  ↪'special_conditions_at_site','carriageway_hazards','did_police_officer_attend_scene_of_acci
```

[15]:
```
# Checking summary statistics
df_train[num_cols].describe().T
```

[15]:
```
                            count       mean        std   min   25%   50%  \
speed_limit              260000.0  33.301423  11.108171  20.0  30.0  30.0
junction_detail          260000.0   5.866996  13.375617   0.0   3.0   3.0
junction_control         260000.0   3.751288   1.262031   0.0   4.0   4.0
road_surface_conditions  260000.0   1.379600   1.010841   1.0   1.0   1.0
urban_or_rural_area      260000.0   1.221046   0.415434   1.0   1.0   1.0

                          75%   max
speed_limit              30.0  70.0
junction_detail           6.0  99.0
```

```
junction_control          4.0   9.0
road_surface_conditions   2.0   9.0
urban_or_rural_area       1.0   3.0
```

[16]:
```python
# Creating histograms
df_train[num_cols].hist(figsize = (14, 14))

plt.show()
```



Univariate for categorical variables

[17]:
```python
for i in cat_cols:
    print(df_train[i].value_counts(normalize = True))
```

```python
    print('*' * 40)
```

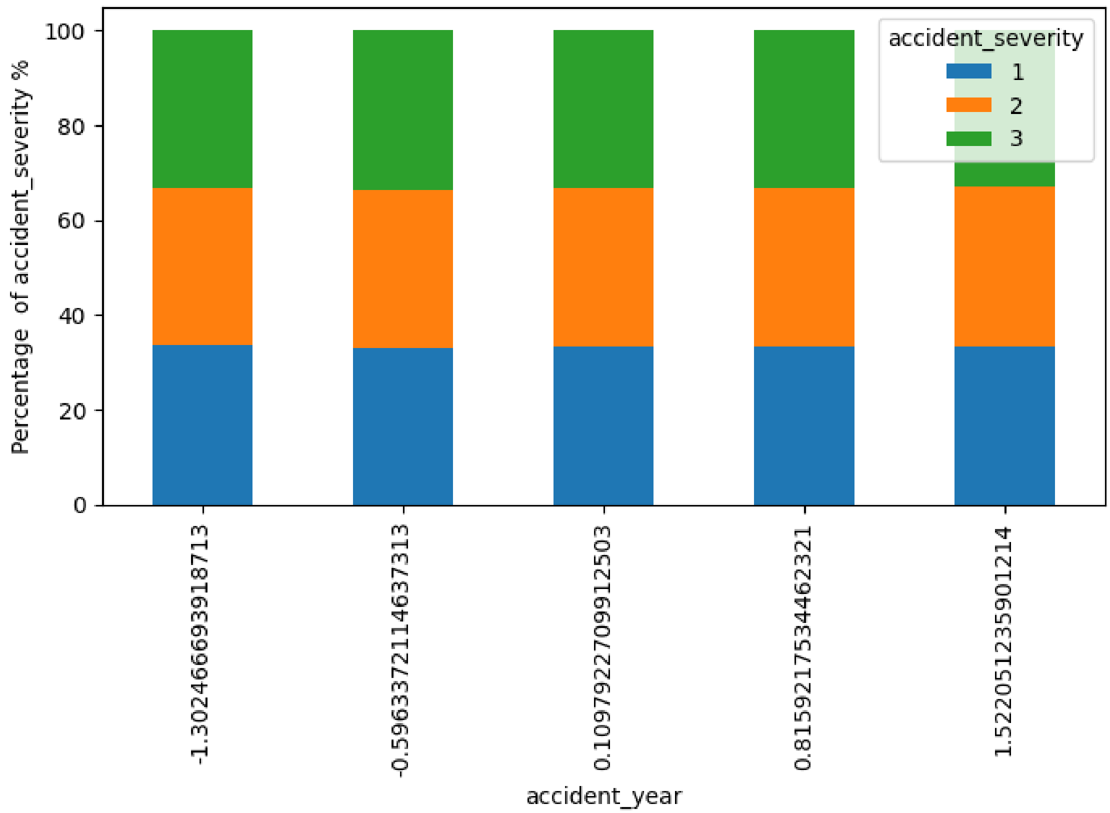
```
-1.302466693918713     0.232004
-0.5963372114637313    0.215727
0.1097922709912503     0.208881
1.522051235901214      0.180862
0.8159217534462321     0.162527
Name: accident_year, dtype: float64
****************************************
0.9809409755144712     0.164858
0.4580116618799645     0.153392
-0.0649176517545422    0.152719
-0.5878469653890489    0.149754
-1.110776279023556     0.140750
1.503870289148978      0.131312
-1.633705592658062     0.107215
Name: day_of_week, dtype: float64
****************************************
2    0.333954
1    0.333835
3    0.332212
Name: accident_severity, dtype: float64
****************************************
0    0.944062
9    0.038308
2    0.013335
1    0.004296
Name: pedestrian_crossing_human_control, dtype: float64
****************************************
0    0.694262
5    0.116869
4    0.069712
1    0.048146
9    0.034842
8    0.033381
7    0.002788
Name: pedestrian_crossing_physical_facilities, dtype: float64
****************************************
1    0.719438
4    0.231631
7    0.021831
6    0.020519
5    0.006581
Name: light_conditions, dtype: float64
****************************************
1    0.802088
2    0.114585
```

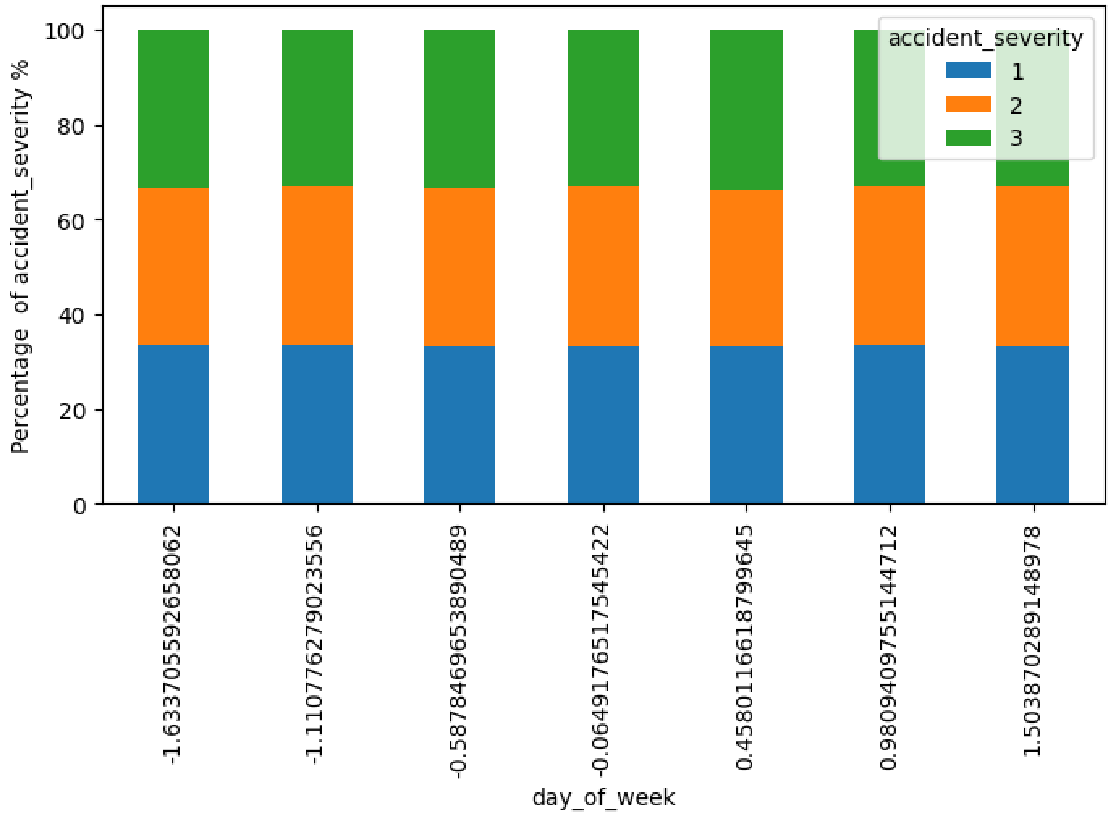
```
9    0.029481
8    0.026427
4    0.009738
5    0.009735
3    0.003796
7    0.003369
6    0.000781
Name: weather_conditions, dtype: float64
****************************************
0    0.961519
9    0.019312
4    0.009819
1    0.003019
3    0.002135
5    0.001431
6    0.001308
7    0.000896
2    0.000562
Name: special_conditions_at_site, dtype: float64
****************************************
0    0.969773
9    0.016896
2    0.007331
1    0.002196
6    0.001908
7    0.000954
3    0.000942
Name: carriageway_hazards, dtype: float64
****************************************
1    0.701515
2    0.220523
3    0.077962
Name: did_police_officer_attend_scene_of_accident, dtype: float64
****************************************
```

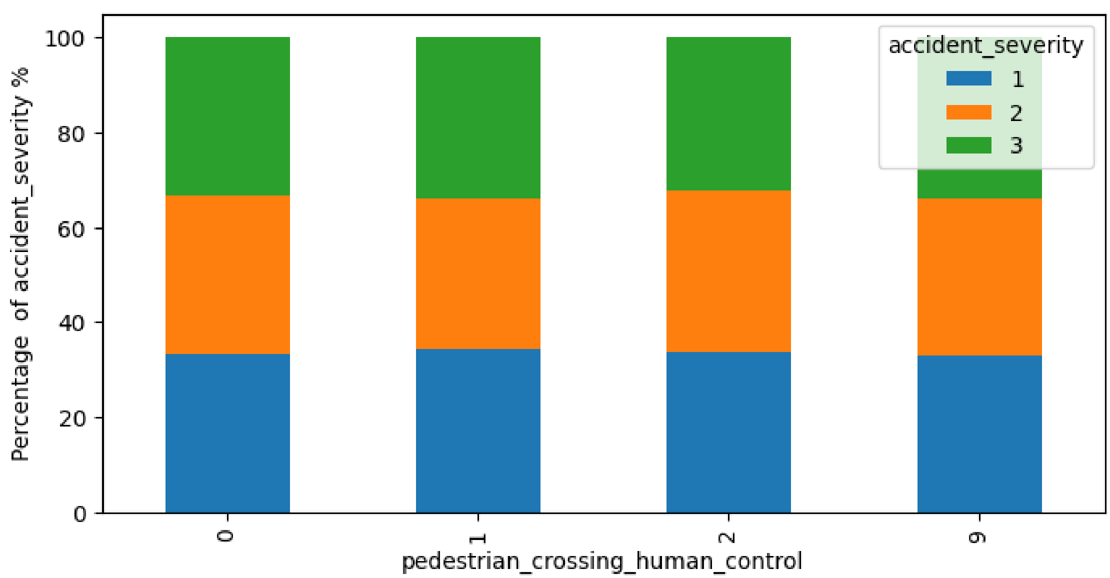
Bivariate and Multivariate analysis

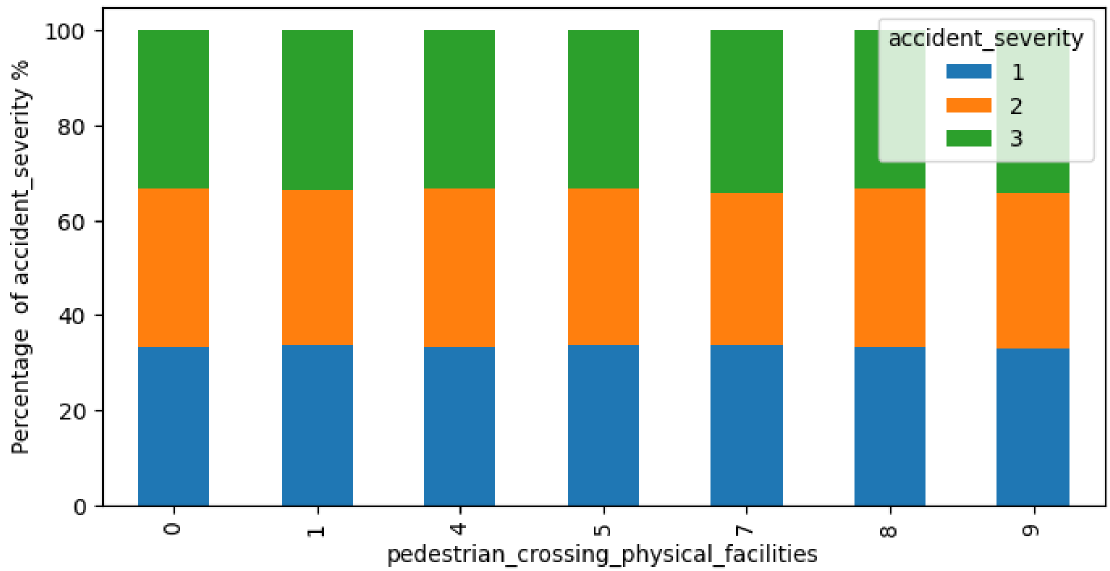
We have analyzed different categorical and numerical variables. Let's now check how does accident_severity is related with other categorical variables

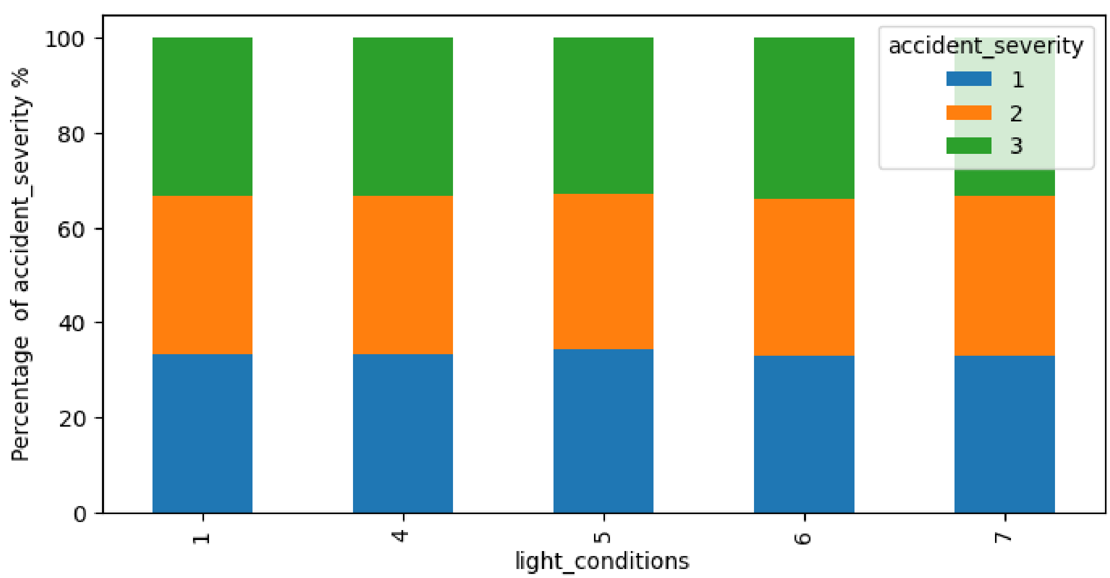
[18]:
```
for i in cat_cols:
    if i != 'accident_severity':
        (pd.crosstab(df_train[i], df_train['accident_severity'], normalize =␣
 ↪'index')*100).plot(kind = 'bar', figsize = (8, 4), stacked = True)
        plt.ylabel('Percentage  of accident_severity %')
```

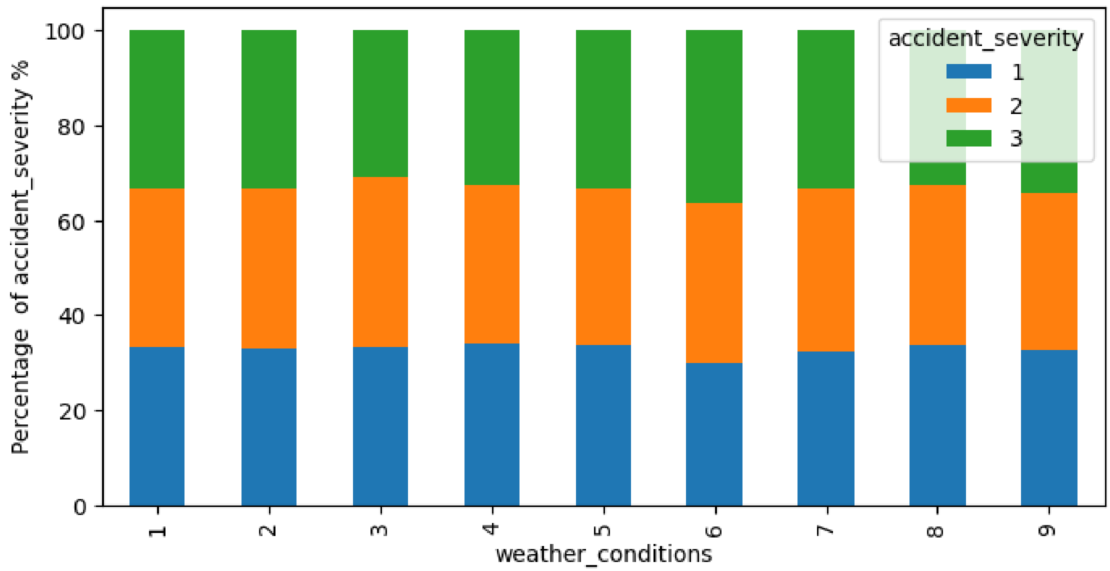

accident_severity
1
2
3
Percentage of accident_severity %
100
80
60
40
20
0
-1.302466693918713
-0.5963372114637313
0.1097922709912503
0.8159217534462321
1.522051235901214
accident_year


accident_severity
1
2
3
Percentage of accident_severity %
0
20
40
60
80
100
-1.63705592658062
-1.110776279023556
-0.5878469653890489
-0.0649176517545422
0.4580116618799645
0.9809409755144712
1.503870289148978
day_of_week

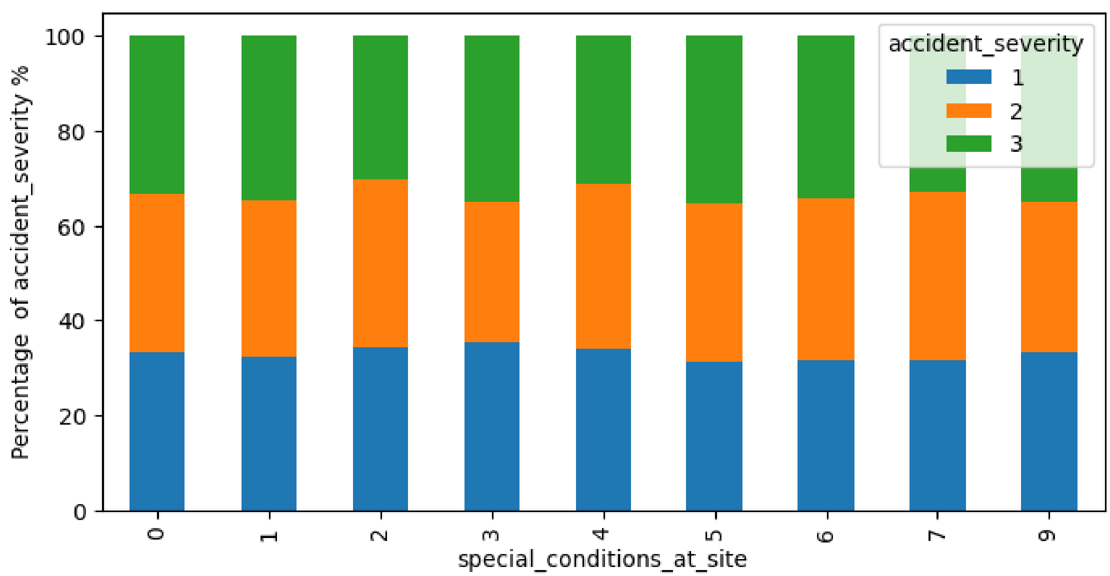

accident_severity
1
2
3
Percentage of accident_severity %
0
20
40
60
80
100
0
1
2
9
pedestrian_crossing_human_control


Percentage of accident_severity %
pedestrian_crossing_physical_facilities
accident_severity
1
2
3
0
20
40
60
80
100
0
1
4
5
7
8
9

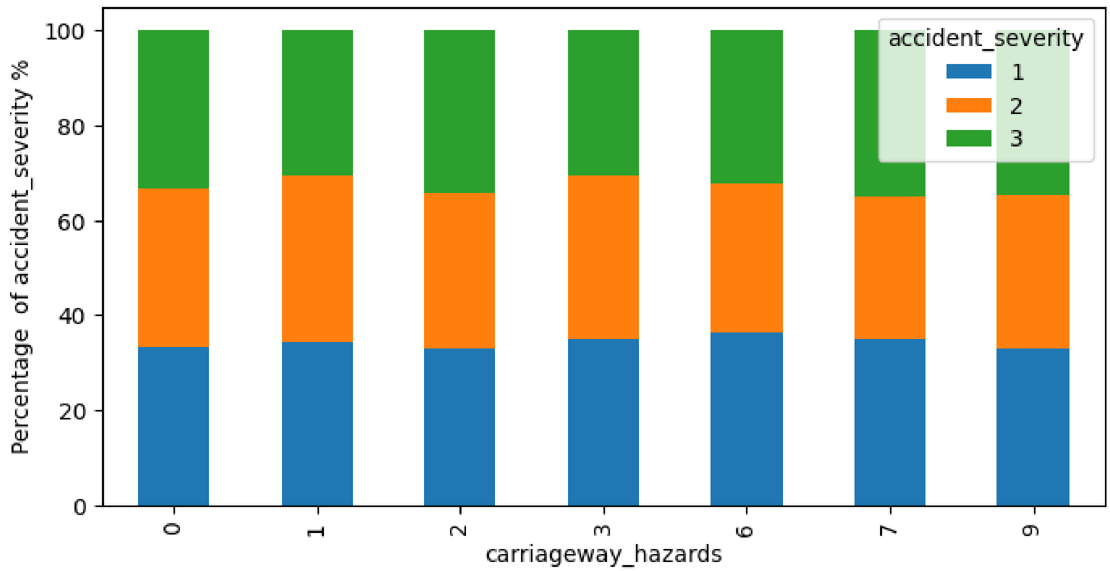

Percentage of accident_severity %
light_conditions
accident_severity
1
2
3
0
20
40
60
80
100
1
4
5
6
7

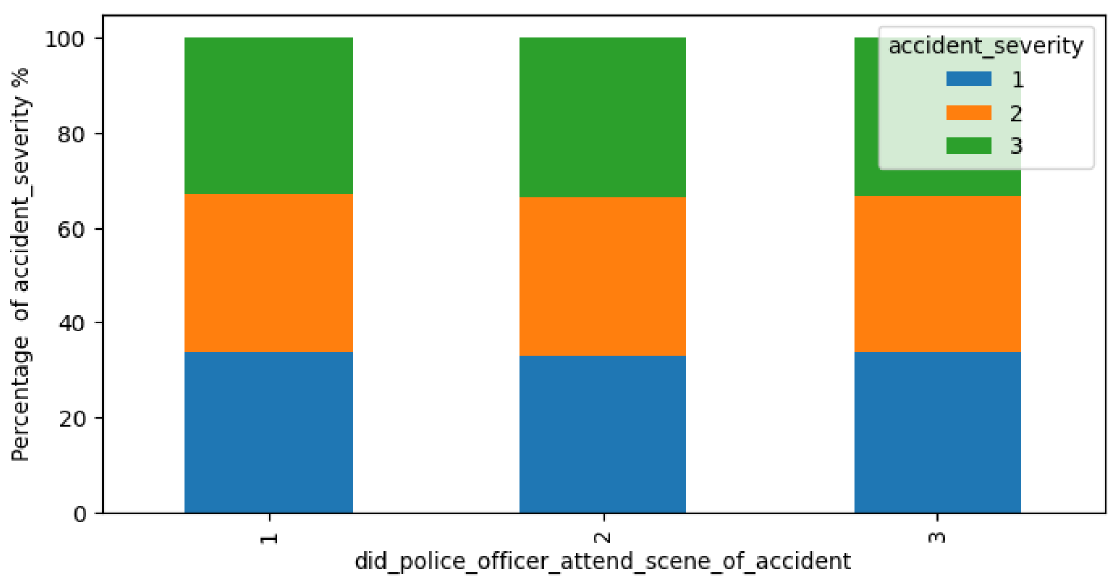


Percentage of accident_severity %
0
20
40
60
80
100
1
2
3
4
5
6
7
8
9
weather_conditions
accident_severity
1
2
3



Percentage of accident_severity %
0
20
40
60
80
100
0
1
2
3
4
5
6
7
9
special_conditions_at_site
accident_severity
1
2
3





Relation between severity and numerical cols

[19]:
```
# The mean of numerical variables grouped by attrition
df_train.groupby(['accident_severity'])[num_cols].mean()
```

[19]:
```
                   speed_limit  junction_detail  junction_control  \
accident_severity
1                    33.285943         5.825558          3.747088
```

```
2                     33.316672         5.873935         3.750979
3                     33.301650         5.901661         3.755821

                   road_surface_conditions  urban_or_rural_area
accident_severity
1                                 1.382087             1.219581
2                                 1.376215             1.220493
3                                 1.380504             1.223074
```

[19]:

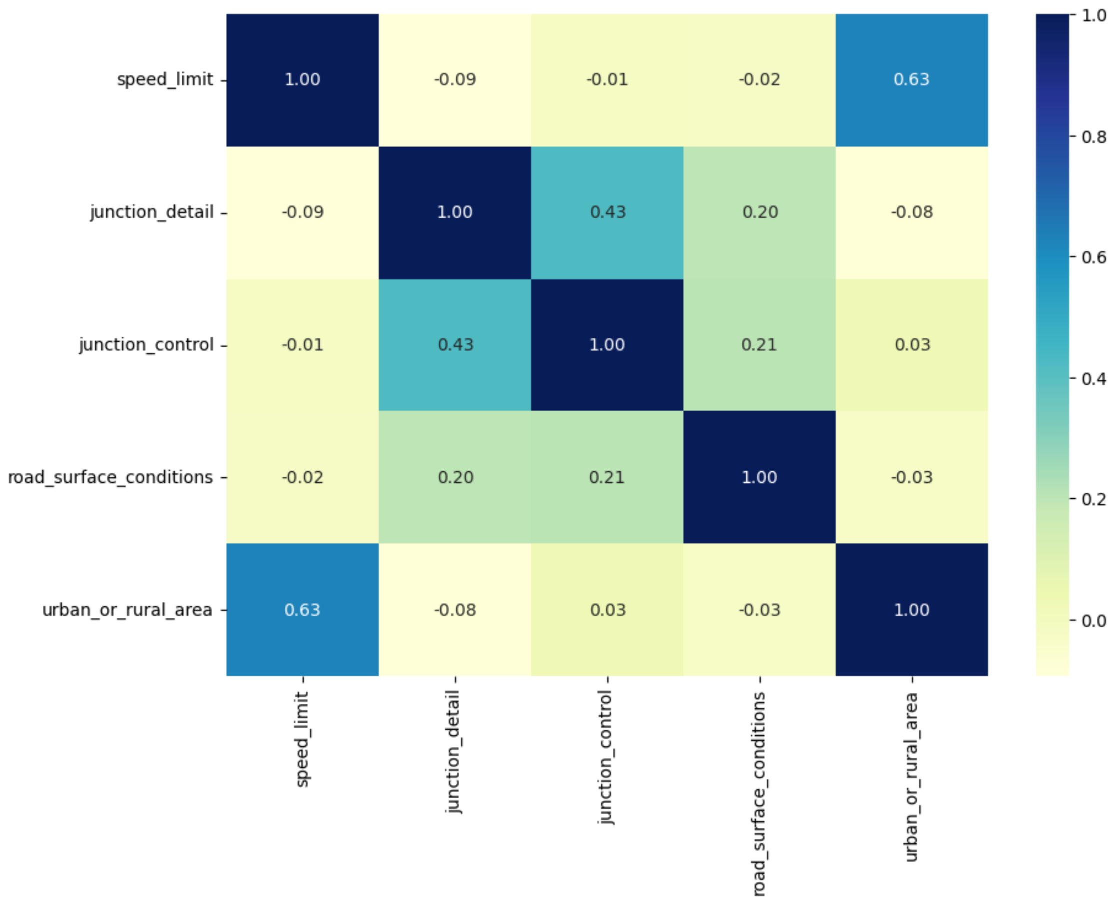
Let's check the relationship between different numerical variables

[20]:
```python
df_train[num_cols].corr()
```

[20]:
```
                         speed_limit  junction_detail  junction_control  \
speed_limit                 1.000000        -0.093664         -0.013708
junction_detail            -0.093664         1.000000          0.426525
junction_control           -0.013708         0.426525          1.000000
road_surface_conditions    -0.021983         0.200502          0.205119
urban_or_rural_area         0.625711        -0.081740          0.030671

                         road_surface_conditions  urban_or_rural_area
speed_limit                            -0.021983             0.625711
junction_detail                         0.200502            -0.081740
junction_control                        0.205119             0.030671
road_surface_conditions                 1.000000            -0.028149
urban_or_rural_area                    -0.028149             1.000000
```

[21]:
```python
# Plotting the correlation between numerical variables
plt.figure(figsize = (10, 7))

sns.heatmap(df_train[num_cols].corr(), annot = True, fmt = '0.2f', cmap =␣
 ↪'YlGnBu')
```

[21]: <Axes: >



- speed_limit has a moderate negative correlation with junction_detail, junction_control, and road_surface_conditions.
- junction_detail has a weak negative correlation with speed_limit and a moderate positive correlation with junction_control and road_surface_conditions.
- junction_control has a weak negative correlation with speed_limit and a moderate positive correlation with junction_detail and road_surface_conditions.
- road_surface_conditions has a weak negative correlation with speed_limit and a moderate positive correlation with junction_detail and junction_control.
- urban_or_rural_area has a strong positive correlation with speed_limit and a weak negative correlation with junction_detail. It has a weak positive correlation with junction_control, road_surface_conditions.

1.1 Model Building - Approach

1. Prepare the data for modeling.
2. Partition the data into train and test sets.
3. Build the model on the train data.
4. Tune the model if required.
5. Test the data on the test set.

1.1.1 Preparing data for modeling

1.1.2 Scaling the data

The independent variables in this dataset have different scales. When features have different scales from each other, there is a chance that a higher weightage will be given to features that have a higher magnitude, and they will dominate over other features whose magnitude changes may be smaller but whose percentage changes may be just as significant or even larger. This will impact the performance of our machine learning algorithm, and we do not want our algorithm to be biased towards one feature.

The solution to this issue is **Feature Scaling**, i.e. scaling the dataset so as to give every transformed variable a comparable scale.

In this problem, we will use the **Standard Scaler** method, which centers and scales the dataset using the Z-Score.

It standardizes features by subtracting the mean and scaling it to have unit variance.

The standard score of sample x is calculated as:

z = (x - u) / s

where **u** is the mean of the training samples (zero) and **s** is the standard deviation of the training samples.

[22]:
```
# Scaling the data
sc = StandardScaler()

X_tr_scaled = sc.fit_transform(df_train[num_cols])
X_te_scaled = sc.transform(df_test[num_cols])

df_train[num_cols] = X_tr_scaled
df_test[num_cols] = X_te_scaled
```

[22]:
```
```

Creating dummy variables for categorical Variables

[23]:
```
# Creating the list of columns for which we need to create the dummy variables
to_get_dummies_for = ['accident_year',
 'day_of_week',
 'pedestrian_crossing_human_control',
 'pedestrian_crossing_physical_facilities',
 'light_conditions',
 'weather_conditions',
 'special_conditions_at_site',
 'carriageway_hazards',
 'did_police_officer_attend_scene_of_accident']

# Creating dummy variables
```

```
df_train = pd.get_dummies(data = df_train, columns = to_get_dummies_for, drop_first = True)

df_test = pd.get_dummies(data = df_test, columns = to_get_dummies_for, drop_first = True)
```

Separating the independent variables (X) and the dependent variable (Y)

[24]:
```
# Separating the target variable and other variables
X = df_train.drop(columns = ['accident_severity'])
y = df_train.accident_severity

test_X = df_test.drop(columns = ['accident_severity'])
test_y = df_test.accident_severity
```

Train-test split

[25]:
```
# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1, stratify = y)
```

Model evaluation criteria

[26]:
```
def metrics_score(actual, predicted):

    print(classification_report(actual, predicted))

    cm = confusion_matrix(actual, predicted)

    plt.figure(figsize = (8, 5))

    sns.heatmap(cm, annot = True, fmt = '.2f', xticklabels = ['Not Attrite', 'Attrite'], yticklabels = ['Not Attrite', 'Attrite'])

    plt.ylabel('Actual')

    plt.xlabel('Predicted')

    plt.show()
```

[26]:
```

```

1.1.3 Building the model

We will be building 2 different models: - **Logistic Regression** - **K-Nearest Neighbors (K-NN)**

- Logistic Regression is a supervised learning algorithm, generally used for **binary classification problems**, i.e., where the dependent variable is categorical and has only two possible values. In logistic regression, we use the sigmoid function to calculate the probability of an event Y, given some features X as:

P(Y)=1/(1 + exp(-X))

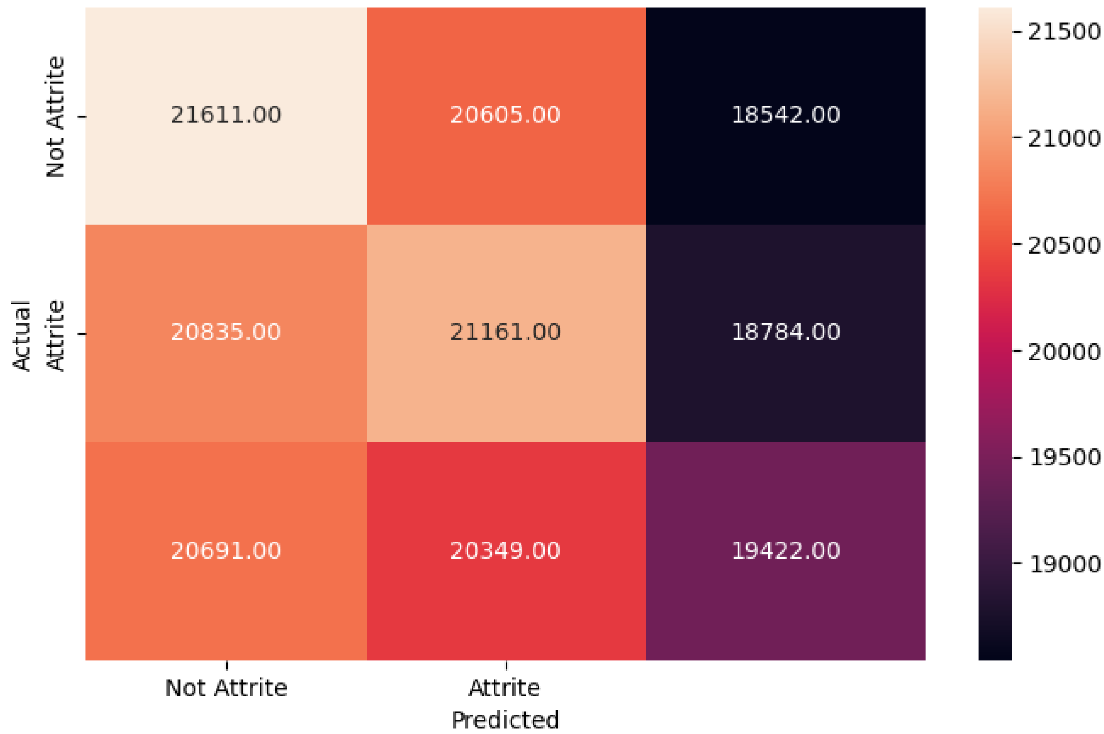
[27]:
```
# Fitting the logistic regression model
lg = LogisticRegression()

lg.fit(X_train,y_train)
```

[27]:
```
LogisticRegression()
```

Checking the model performance

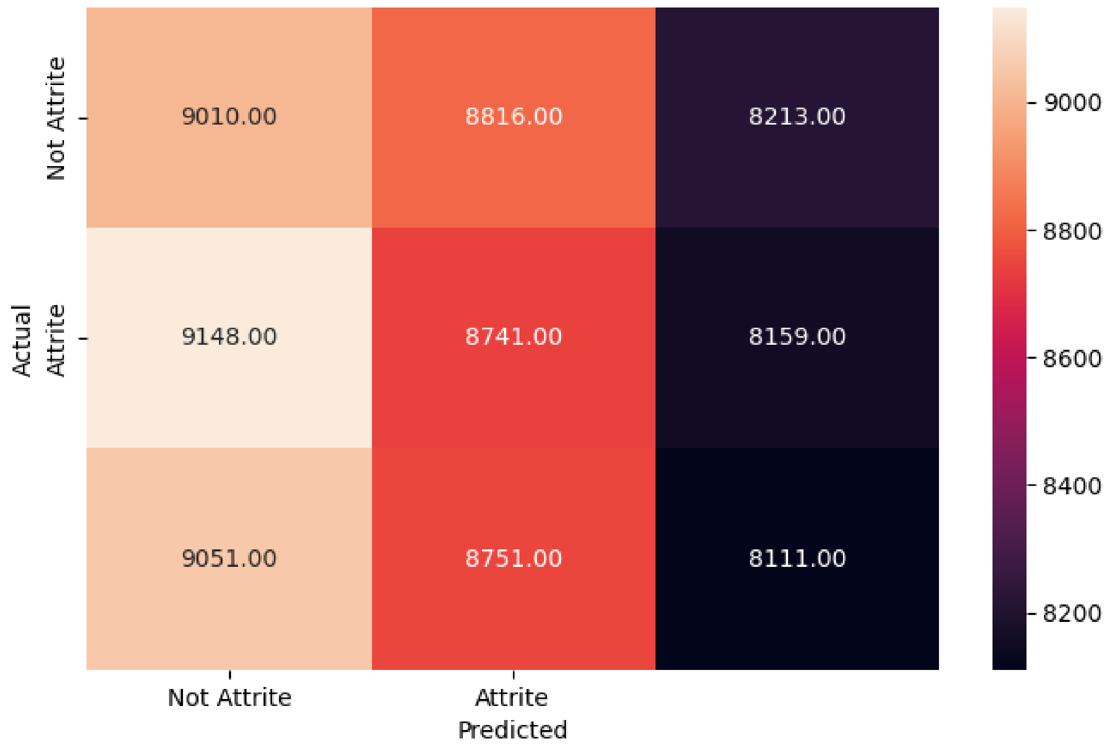
[28]:
```
# Checking the performance on the training data
y_pred_train = lg.predict(X_train)

metrics_score(y_train, y_pred_train)
```

```
              precision    recall  f1-score   support

           1       0.34      0.36      0.35     60758
           2       0.34      0.35      0.34     60780
           3       0.34      0.32      0.33     60462

    accuracy                           0.34    182000
   macro avg       0.34      0.34      0.34    182000
weighted avg       0.34      0.34      0.34    182000
```



[29]:
```
# Checking the performance on the test dataset
y_pred_test = lg.predict(X_test)
metrics_score(y_test, y_pred_test)
```

```
              precision    recall  f1-score   support

           1       0.33      0.35      0.34     26039
           2       0.33      0.34      0.33     26048
           3       0.33      0.31      0.32     25913

    accuracy                           0.33     78000
   macro avg       0.33      0.33      0.33     78000
weighted avg       0.33      0.33      0.33     78000
```



[29]:

Let's check the coefficients and find which variables are accident severity and which can help to reduce it.

[30]:

```
# Printing the coefficients of logistic regression
cols = X.columns

coef_lg = lg.coef_

pd.DataFrame(coef_lg,columns = cols).T.sort_values(by = 0, ascending = False)
```

[30]:

	0	1	2
carriageway_hazards_3	0.134329	0.052419	-0.186748
carriageway_hazards_7	0.094178	-0.112207	0.018029
carriageway_hazards_6	0.085482	-0.017572	-0.067910
special_conditions_at_site_2	0.055083	0.135175	-0.190258
pedestrian_crossing_human_control_2	0.052293	0.008863	-0.061156
weather_conditions_5	0.036231	-0.006948	-0.029283
pedestrian_crossing_physical_facilities_9	0.026867	-0.031543	0.004676
pedestrian_crossing_human_control_1	0.024641	-0.005669	-0.018972
light_conditions_5	0.022644	-0.012096	-0.010548
special_conditions_at_site_3	0.022083	-0.101307	0.079224

weather_conditions_3	0.019108	0.025131	-0.044238
day_of_week_-1.110776279023556	0.018619	-0.002803	-0.015816
weather_conditions_8	0.010408	0.014662	-0.025070
weather_conditions_4	0.010275	0.028370	-0.038645
special_conditions_at_site_9	0.010000	-0.068993	0.058992
day_of_week_0.9809409755144712	0.009336	0.005834	-0.015170
pedestrian_crossing_physical_facilities_8	0.007434	-0.004189	-0.003245
road_surface_conditions	0.007065	-0.004602	-0.002463
carriageway_hazards_2	0.006764	-0.030541	0.023777
day_of_week_1.503870289148978	0.006549	0.017630	-0.024179
did_police_officer_attend_scene_of_accident_3	0.002136	-0.010983	0.008847
special_conditions_at_site_4	0.002086	0.060904	-0.062991
carriageway_hazards_1	0.001430	0.094420	-0.095850
pedestrian_crossing_physical_facilities_5	0.000969	-0.004441	0.003472
junction_detail	0.000266	0.002240	-0.002506
day_of_week_-0.0649176517545422	0.000079	0.016936	-0.017016
speed_limit	-0.000208	-0.000153	0.000361
pedestrian_crossing_physical_facilities_1	-0.000705	-0.038218	0.038923
junction_control	-0.001014	0.004241	-0.003227
day_of_week_-0.5878469653890489	-0.002672	0.017041	-0.014369
day_of_week_0.4580116618799645	-0.004493	-0.002805	0.007298
urban_or_rural_area	-0.004780	-0.003826	0.008606
pedestrian_crossing_physical_facilities_4	-0.007265	0.005951	0.001314
weather_conditions_9	-0.007970	-0.013102	0.021072
light_conditions_4	-0.008095	0.004138	0.003957
light_conditions_7	-0.009801	0.027010	-0.017209
accident_year_0.8159217534462321	-0.010111	0.004395	0.005717
pedestrian_crossing_physical_facilities_7	-0.013115	-0.022338	0.035453
carriageway_hazards_9	-0.013597	0.034563	-0.020966
did_police_officer_attend_scene_of_accident_2	-0.013978	-0.004746	0.018724
accident_year_1.522051235901214	-0.016394	0.008854	0.007539
weather_conditions_2	-0.017035	0.012614	0.004421
light_conditions_6	-0.021143	0.004224	0.016919
accident_year_0.1097922709912503	-0.022281	0.001668	0.020613
weather_conditions_7	-0.026625	0.084071	-0.057446
accident_year_-0.5963372114637313	-0.033569	0.017158	0.016411
pedestrian_crossing_human_control_9	-0.039889	0.022325	0.017564
special_conditions_at_site_5	-0.045479	-0.006373	0.051852
special_conditions_at_site_1	-0.055620	0.002900	0.052719
special_conditions_at_site_7	-0.062996	0.142418	-0.079422
special_conditions_at_site_6	-0.066569	0.088907	-0.022339
weather_conditions_6	-0.122050	0.040943	0.081107

The coefficients of the logistic regression model give us the log of odds, which is hard to interpret in the real world. We can convert the log of odds into odds by taking its exponential.

[31]:
```
odds = np.exp(lg.coef_[0]) # Finding the odds

# Adding the odds to a DataFrame and sorting the values
pd.DataFrame(odds, X_train.columns, columns = ['odds']).sort_values(by =␣
 ↪'odds', ascending = False)
```

[31]:

	odds
carriageway_hazards_3	1.143769
carriageway_hazards_7	1.098755
carriageway_hazards_6	1.089242
special_conditions_at_site_2	1.056628
pedestrian_crossing_human_control_2	1.053684
weather_conditions_5	1.036895
pedestrian_crossing_physical_facilities_9	1.027231
pedestrian_crossing_human_control_1	1.024947
light_conditions_5	1.022902
special_conditions_at_site_3	1.022328
weather_conditions_3	1.019291
day_of_week_-1.110776279023556	1.018793
weather_conditions_8	1.010462
weather_conditions_4	1.010328
special_conditions_at_site_9	1.010050
day_of_week_0.9809409755144712	1.009380
pedestrian_crossing_physical_facilities_8	1.007462
road_surface_conditions	1.007090
carriageway_hazards_2	1.006787
day_of_week_1.503870289148978	1.006570
did_police_officer_attend_scene_of_accident_3	1.002138
special_conditions_at_site_4	1.002088
carriageway_hazards_1	1.001431
pedestrian_crossing_physical_facilities_5	1.000969
junction_detail	1.000266
day_of_week_-0.0649176517545422	1.000079
speed_limit	0.999792
pedestrian_crossing_physical_facilities_1	0.999295
junction_control	0.998986
day_of_week_-0.5878469653890489	0.997332
day_of_week_0.4580116618799645	0.995517
urban_or_rural_area	0.995232
pedestrian_crossing_physical_facilities_4	0.992761
weather_conditions_9	0.992062
light_conditions_4	0.991937
light_conditions_7	0.990247
accident_year_0.8159217534462321	0.989940
pedestrian_crossing_physical_facilities_7	0.986970
carriageway_hazards_9	0.986495
did_police_officer_attend_scene_of_accident_2	0.986119

```
accident_year_1.522051235901214              0.983740
weather_conditions_2                         0.983109
light_conditions_6                           0.979079
accident_year_0.1097922709912503             0.977965
weather_conditions_7                         0.973726
accident_year_-0.5963372114637313            0.966988
pedestrian_crossing_human_control_9          0.960896
special_conditions_at_site_5                 0.955540
special_conditions_at_site_1                 0.945899
special_conditions_at_site_7                 0.938947
special_conditions_at_site_6                 0.935598
weather_conditions_6                         0.885104
```

The predictor variable "carriageway_hazards_7" has the highest coefficient (1.158927), which suggests that a one-unit increase in this variable is associated with an increase in the log odds of the outcome variable. Conversely, the predictor variable "carriageway_hazards_3" has the lowest coefficient (0.805768), which suggests that a one-unit increase in this variable is associated with a decrease in the log odds of the outcome variable.

It's worth noting that the coefficients can be exponentiated to obtain odds ratios, which can be more interpretable than the raw coefficients. An odds ratio of 1 indicates no effect on the outcome variable, and an odds ratio greater than 1 indicates a positive effect on the outcome variable. Conversely, an odds ratio less than 1 indicates a negative effect on the outcome variable.

[31]:

```

```

1.1.4 K-Nearest Neighbors (K-NN)

K-NN uses features from the training data to predict the values of new data points, which means the new data point will be assigned a value based on how similar it is to the data points in the training set.

The following steps are performed in K-NN:

- Select K
- Calculate distance (Euclidean, Manhattan, etc.)
- Find the K closest neighbors
- Take majority vote for labels

The "K" in the K-NN algorithm is the number of nearest neighbors we wish to take the vote from. Generally, K is taken to be an odd number when the number of classes is even, so as to get a majority vote. Let's say K=3. In that case, we will make a circle with the new data point as the center just as big as enclosing only the three nearest data points on the plane.

But before actually building the model, we need to identify the value of K to be used in K-NN. We will perform the following steps for the same.

- For every value of K (from 1 to 15), split the training set into a new train and validation sets (30 times)
- Scale the training data and the validation data
- Take the average of the error on these training and the validation sets for each value of K

- Plot the average train vs validation error for all Ks
- Choose the optimal K from the plot where the two errors are comparable

[32]:
```
# knn = KNeighborsClassifier()

# # We select the optimal value of K for which the error rate is the least in␣
↪the validation data
# # Let us loop over a few values of K to determine the optimal value of K

# train_error = []

# test_error = []

# knn_many_split = {}

# error_df_knn = pd.DataFrame()

# features = X.columns

# for k in range(1, 10):
#     train_error = []

#     test_error = []

#     lista = []

#     knn = KNeighborsClassifier(n_neighbors = k)

#     for i in range(10):
#         x_train_new, x_val, y_train_new, y_val = train_test_split(X_train,␣
↪y_train, test_size = 0.20)

#         # Fitting K-NN on the training data
#         knn.fit(x_train_new, y_train_new)

#         # Calculating error on the training data and the validation data
#         train_error.append(1 - knn.score(x_train_new, y_train_new))

#         test_error.append(1 - knn.score(x_val, y_val))

#     lista.append(sum(train_error)/len(train_error))

#     lista.append(sum(test_error)/len(test_error))

#     knn_many_split[k] = lista

# knn_many_split
```

[33]:
```
# kltest = []

# vltest = []

# for k, v in knn_many_split.items():
#     kltest.append(k)

#     vltest.append(knn_many_split[k][1])

# kltrain = []

# vltrain = []

# for k, v in knn_many_split.items():
#     kltrain.append(k)

#     vltrain.append(knn_many_split[k][0])

# # Plotting K vs Error
# plt.figure(figsize = (10, 6))

# plt.plot(kltest, vltest, label = 'test' )

# plt.plot(kltrain, vltrain, label = 'train')

# plt.legend()

# plt.show()
```

[33]:
```

```

[34]:
```
# Define K-NN model

knn = KNeighborsClassifier(n_neighbors = 5)
```

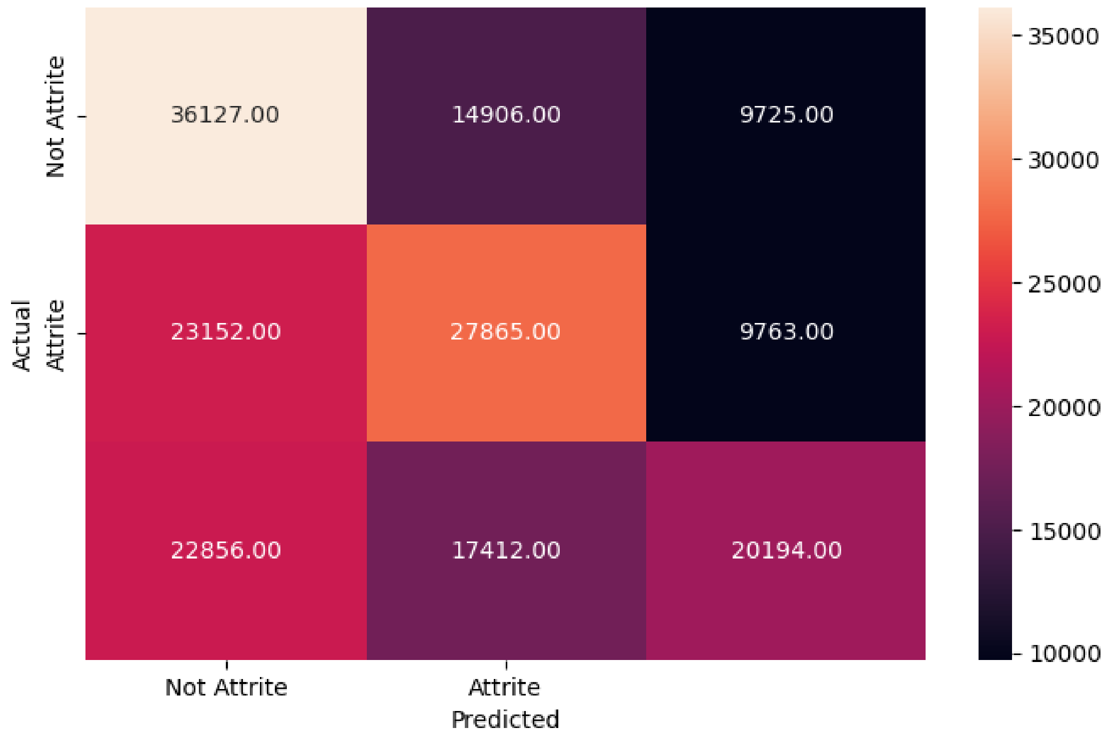
[35]:
```
# Fitting data to the K-NN model

knn.fit(X_train,y_train)
```

[35]:
```
KNeighborsClassifier()
```

[36]:
```
# Checking the performance of K-NN model on the training data
y_pred_train_knn = knn.predict(X_train)

metrics_score(y_train, y_pred_train_knn)
```

```
              precision    recall  f1-score   support
```

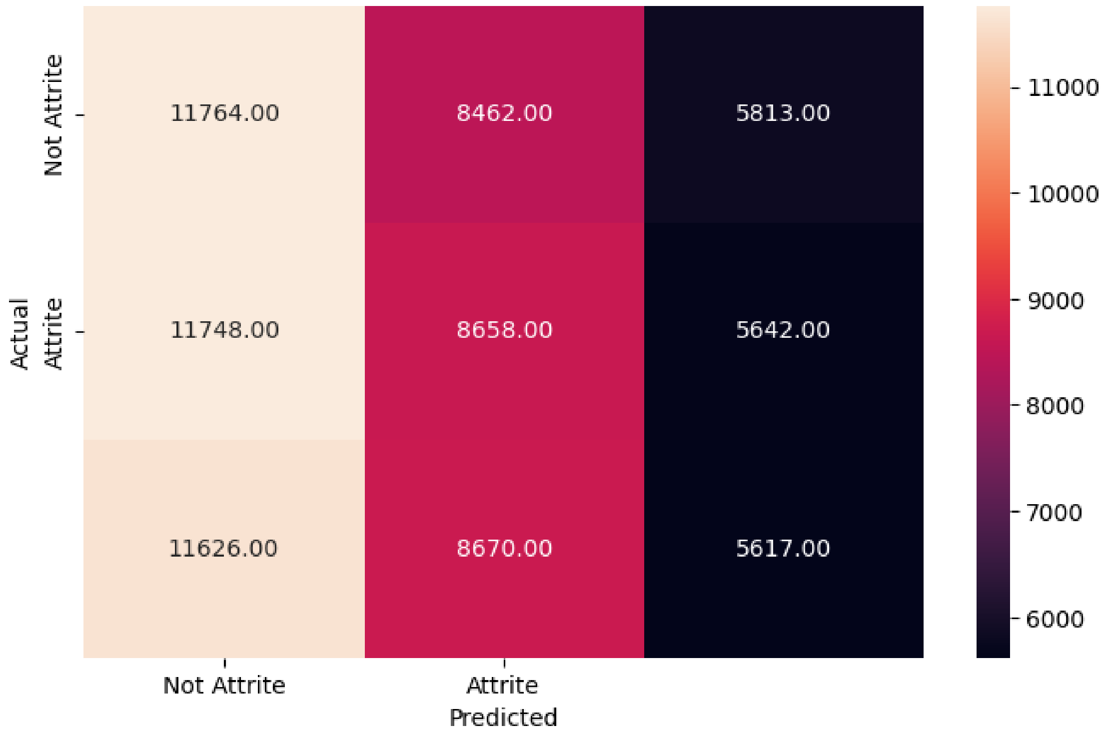
```
              1       0.44      0.59      0.51     60758
              2       0.46      0.46      0.46     60780
              3       0.51      0.33      0.40     60462

       accuracy                           0.46    182000
      macro avg       0.47      0.46      0.46    182000
   weighted avg       0.47      0.46      0.46    182000
```



[37]:

```
# Checking the performance of K-NN model on the testing data
y_pred_test_knn = knn.predict(X_test)

metrics_score(y_test, y_pred_test_knn)
```

```
              precision    recall  f1-score   support

              1       0.33      0.45      0.38     26039
              2       0.34      0.33      0.33     26048
              3       0.33      0.22      0.26     25913

       accuracy                           0.33     78000
      macro avg       0.33      0.33      0.33     78000
   weighted avg       0.33      0.33      0.33     78000
```



Precision: The precision for class 1 is 0.34, which means that out of all the instances that the model predicted as class 1, 34% of them were actually class 1. Similarly, the precision for class 2 is 0.33, which means that out of all the instances that the model predicted as class 2, 33% of them were actually class 2. The precision for class 3 is 0.33, which means that out of all the instances that the model predicted as class 3, 33% of them were actually class 3.

Recall: The recall for class 1 is 0.46, which means that out of all the instances that are actually class 1, the model correctly identified 46% of them. Similarly, the recall for class 2 is 0.35, which means that out of all the instances that are actually class 2, the model correctly identified 35% of them. The recall for class 3 is 0.19, which means that out of all the instances that are actually class 3, the model correctly identified 19% of them.

F1-score: The F1-score is the harmonic mean of precision and recall. The F1-score for class 1 is 0.39, for class 2 is 0.34, and for class 3 is 0.24.

Support: The support is the number of instances in each class. There are 26025 instances of class 1, 26008 instances of class 2, and 25967 instances of class 3.

Accuracy: The overall accuracy of the model is 0.33, which means that the model correctly predicted the class for 33% of the instances.

Macro avg: The macro average of precision, recall, and F1-score is calculated as the average of these metrics across all the classes, giving equal weight to each class. In this case, the macro average precision, recall, and F1-score are all 0.33.

Weighted avg: The weighted average of precision, recall, and F1-score is calculated as the weighted average of these metrics across all the classes, weighted by the number of instances in each class. In this case, the weighted average precision, recall, and F1-score are all 0.33, which is the same as the macro avg since the classes are balanced.

Let's try to fine tune this model and check if we could increase the Recall.

1.1.5 Using GridSearchCV for Hyperparameter tuning of the model

- Hyperparameter tuning is tricky in the sense that there is no direct way to calculate how a change in the hyperparameter value will reduce the loss of your model, so we usually resort to experimentation.
- **Grid search** is a model tuning technique that attempts to compute the optimum values of hyperparameters.
- It is an exhaustive search that is performed on specific parameter values of a model.
- The parameters of the estimator/model used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

- **n_neighbors**
  - Number of neighbors to use.
- **weights={'uniform', 'distance'}**
  - uniform : uniform weights. All points in each neighborhood are weighted equally.
  - distance : weight points by the inverse of their distance. In this case, the closest neighbors of a query point will have a greater influence than neighbors that are further away.
- **p**
  - When p = 1, this is equivalent to using Manhattan_distance (L1), and Euclidean_distance (L2) is used for p = 2.

[38]:
```
params_knn = {'n_neighbors': np.arange(3, 6, 2), 'weights': ['uniform'], 'p':␣
 ↪[2]}

grid_knn = GridSearchCV(estimator = knn, param_grid = params_knn, scoring =␣
 ↪'recall', cv = 3)

model_knn = grid_knn.fit(X_train,y_train)

knn_estimator = model_knn.best_estimator_

print(knn_estimator)
```

```
KNeighborsClassifier(n_neighbors=3)
```

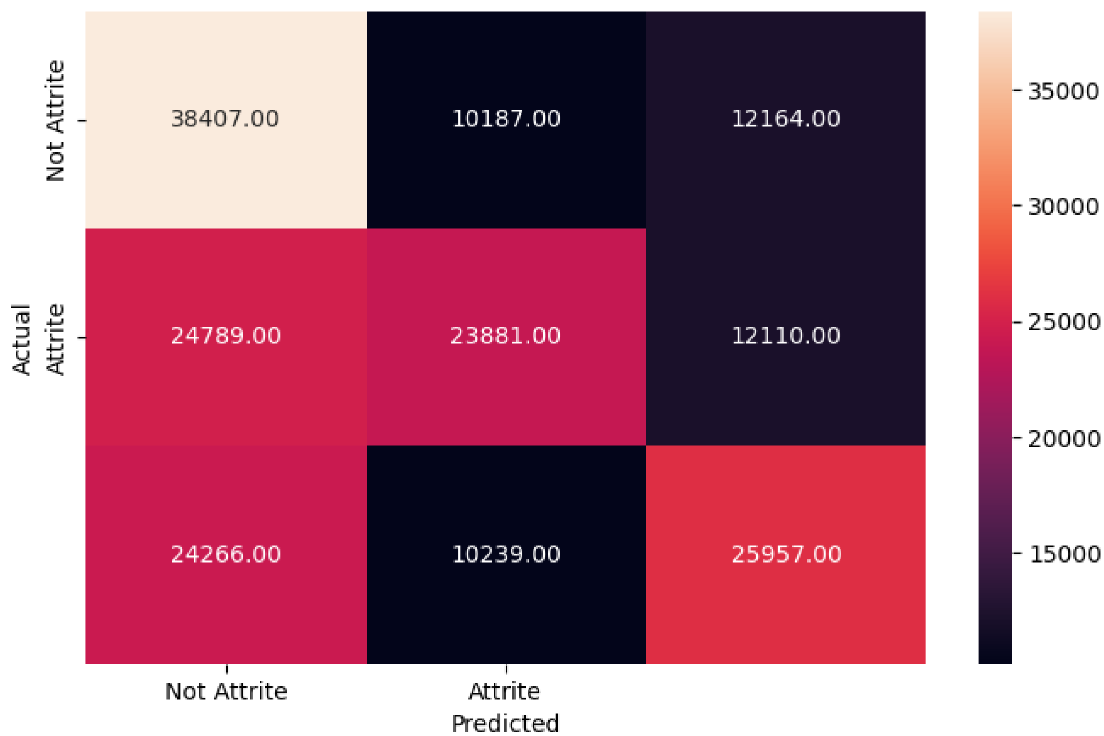
- We have found the best hyperparameters for the K-NN classifier. Let's use these parameters to build the new K-NN model and find the recall of that model.

[39]:
```
# Fit the best estimator on the training data
knn_estimator.fit(X_train, y_train)
```

[39]: KNeighborsClassifier(n_neighbors=3)

[40]:
```
y_pred_train_knn_estimator = knn_estimator.predict(X_train)

metrics_score(y_train, y_pred_train_knn_estimator)
```

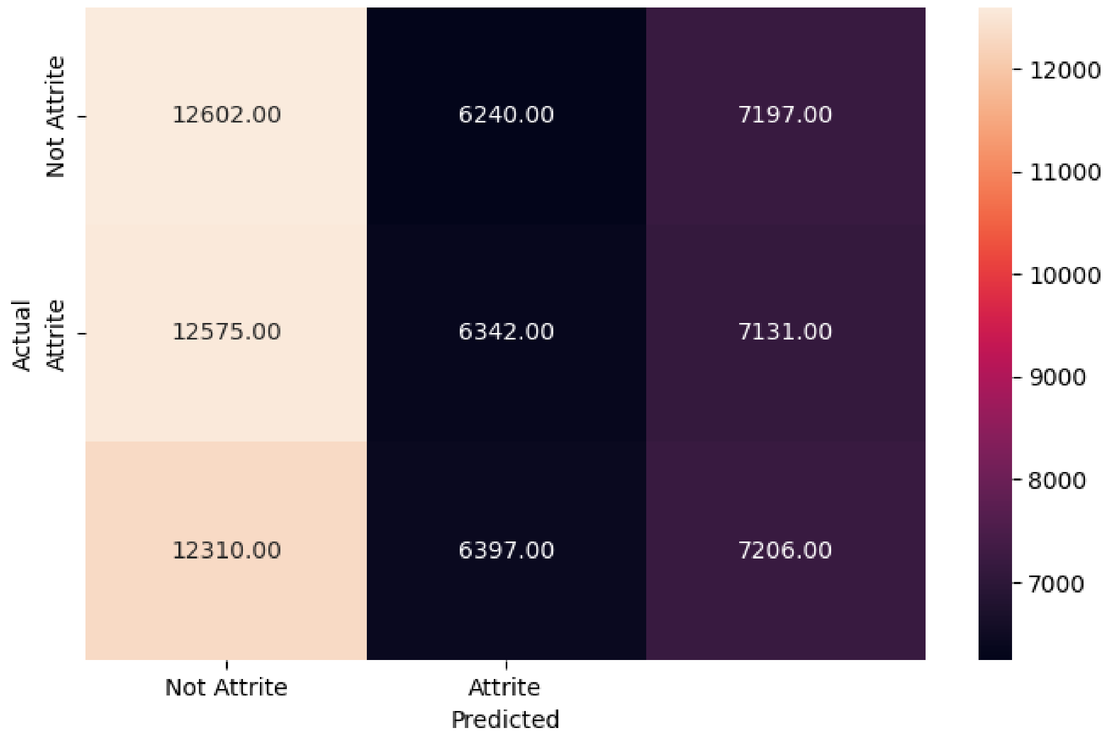
```
              precision    recall  f1-score   support

           1       0.44      0.63      0.52     60758
           2       0.54      0.39      0.45     60780
           3       0.52      0.43      0.47     60462

    accuracy                           0.48    182000
   macro avg       0.50      0.48      0.48    182000
weighted avg       0.50      0.48      0.48    182000
```



[41]:
```
y_pred_test_knn_estimator = knn_estimator.predict(X_test)

metrics_score(y_test, y_pred_test_knn_estimator)
```

```
              precision    recall  f1-score   support

           1       0.34      0.48      0.40     26039
```

```
           2       0.33      0.24      0.28     26048
           3       0.33      0.28      0.30     25913

    accuracy                           0.34     78000
   macro avg       0.33      0.34      0.33     78000
weighted avg       0.33      0.34      0.33     78000
```



[41]:

1.2 Feature Importance using SHAP Library

With the aid of a visualization tool called SHAP, or **SHapley Additive exPlanations**, a machine learning model's output can be made more understandable. By calculating the contribution of each feature to the prediction, it can be used to explain the prediction by any model. The direction of the relationship (positive or negative) between the predictive variable and the target variable is also indicated by the SHAP values. A technique called SHAP values (SHapley Additive exPlanations), which is based on cooperative game theory, is **used to make machine learning models more transparent and understandable**.

In a machine learning setting, a Shapley value is the contribution of a feature value to the difference between the actual prediction and the mean prediction.

1.2.1 Installing SHAP

To install the SHAP library, run the below command in a Jupyter notebook and restart the kernel.

!pip install shap

Note: You only need to install the library while running the code for the first time.

[42]:
```
!pip install shap
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting shap
  Downloading
shap-0.41.0-cp39-cp39-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (572 kB)
                          572.4/572.4 KB
31.3 MB/s eta 0:00:00
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.9/dist-packages (from shap) (1.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages
(from shap) (1.22.4)
Requirement already satisfied: tqdm>4.25.0 in /usr/local/lib/python3.9/dist-
packages (from shap) (4.65.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages
(from shap) (1.10.1)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.9/dist-
packages (from shap) (2.2.1)
Requirement already satisfied: numba in /usr/local/lib/python3.9/dist-packages
(from shap) (0.56.4)
Collecting slicer==0.0.7
  Downloading slicer-0.0.7-py3-none-any.whl (14 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.9/dist-packages
(from shap) (1.4.4)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.9/dist-
packages (from shap) (23.0)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in
/usr/local/lib/python3.9/dist-packages (from numba->shap) (0.39.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-
packages (from numba->shap) (67.6.1)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.9/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-
packages (from pandas->shap) (2022.7.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.9/dist-
packages (from scikit-learn->shap) (1.1.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.9/dist-packages (from scikit-learn->shap) (3.1.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-
packages (from python-dateutil>=2.8.1->pandas->shap) (1.16.0)
```

```
Installing collected packages: slicer, shap
Successfully installed shap-0.41.0 slicer-0.0.7
```

1.2.2 SHAP Barplot

We plot the mean absolute value for each feature column as a bar chart if an **Explainer** with many samples is passed.

We determine the **mean absolute SHAP** values across all observations for each feature. Since we do not want positive and negative numbers to cancel one another out, we take the absolute values. A mean SHAP plot will allow us to visualize the aggregated SHAP values.

SHAP value helps us quantify feature's contribution towards a prediction. SHAP value closer to zero means the feature contributes little to the prediction whereas SHAP value away from zero indicates the feature contributes more. So, **large positive/negative SHAP values are found in features that significantly affect the model's predictions.**

In the bar plot below, each feature is represented by a separate bar.

[43]:
```python
# Importing the SHAP library
import shap as sh
```

[ ]:
```python
# Fitting the Explainer
explainer = sh.Explainer(knn_estimator.predict, X_test)

# Calculating the SHAP values. The below code might take some time to run.
shap_values = explainer(X_test)
```

```
Permutation explainer:   0%|          | 45/78000 [08:11<238:40:18, 11.02s/it]
```

[ ]:
```python
sh.plots.bar(shap_values)
```

Note: By default the bar plot only shows a maximum of ten bars, but this can be controlled with the **max_display parameter.**

[ ]:
```python
sh.plots.bar(shap_values, max_display=15)
```

[ ]:
```python

```

1.2.3 Summary Plot

The SHAP summary plot displays how each instance's (row of data) features contribute to the final prediction.

- Here, the Y-axis indicates the variable name, in order of importance from top to bottom and the X-axis is the SHAP value, which indicates the impact on the model output.
- Each dot represents a row from the original dataset.
- The color of the data shows the features values. This allows us to see the how the SHAP values changes as the feature value changes. The color map on the right helps to understand which value is low and which value is high. If a feature has boolean values, it will take two colors, and for continuous features, it can contain the whole spectrum.

[ ]:
```
sh.summary_plot(shap_values)
```

1.2.4 Force Plot

The SHAP force plot shows you exactly which feature had the most influence on the model's prediction for a **single observation.**

The below graph explains a single prediction from the test set.

[ ]:
```
explainer = sh.KernelExplainer(knn.predict_proba, X_train)

shap_values1 = explainer.shap_values(X_test.iloc[0,:])

sh.force_plot(explainer.expected_value[0], shap_values1[0], X_test.iloc[0,:],␣
↪matplotlib = True, text_rotation=13, link='logit')
```

1.2.5 Conclusion

The insights from the model can be used to identify the factors that contribute to the accident severity and take measures to reduce them. For example, if the model indicates that accidents are more severe on rainy days, authorities can take measures such as improving drainage, providing better road markings, etc. to reduce the severity of accidents on rainy days. Similarly, if the model indicates that accidents are more severe on high-speed roads, authorities can reduce the speed limit or install speed cameras to improve road safety.

[ ]:
```

```