CAR ACCIDENT

Machine Learning

Shilat Shimon and Avigail Yazdi - Haliva

Description of the data 

The dataset provides detailed records of road accidents that occurred during January 2021. It includes information such as the accident date, day of the week, junction control, accident severity, geographical coordinates, lighting and weather conditions, vehicle details, and more.

TARGET 🎯

The data is valuable for analyzing and understanding the factors contributing to road accidents, aiding in the development of strategies for improved road safety.

Loading the data

28s

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

3s

```
import numpy as np
import pandas as pd
import os

filepath = '/content/gdrive/MyDrive/Road Accident Data.csv'
data = pd.read_csv(filepath)
print(data.shape)
data.head()
```

```
(307973, 21)
```

	Accident_Index	Accident Date	Day_of_Week	Junction_Control	Junction_Detail	Accident_Severity	Latitude	Light_Conditions	Local_Authority_(Dist
0	200901BS70001	1/1/2021	Thursday	Give way or uncontrolled	T or staggered junction	Serious	51.512273	Daylight	Kensington and C
1	200901BS70002	1/5/2021	Monday	Give way or uncontrolled	Crossroads	Serious	51.514399	Daylight	Kensington and C

```
#The number of each Accident_Severity- the target
print(data["Accident_Severity"].value_counts())
```

```
Slight     263280
Serious     40740
Fatal        3904
Fetal          49
Name: Accident_Severity, dtype: int64
```

0s

```
data.describe()
```

	Latitude	Longitude	Number_of_Casualties	Number_of_Vehicles	Speed_limit
count	307973.000000	307973.000000	307973.000000	307973.000000	307973.000000
mean	52.487005	-1.368884	1.356882	1.829063	38.866037
std	1.339011	1.356092	0.815857	0.710477	14.032933
min	49.914488	-7.516225	1.000000	1.000000	10.000000
25%	51.485248	-2.247937	1.000000	1.000000	30.000000
50%	52.225943	-1.349258	1.000000	2.000000	30.000000
75%	53.415517	-0.206810	1.000000	2.000000	50.000000
max	60.598055	1.759398	48.000000	32.000000	70.000000

```python
# Number of rows
print("number of rows:" , data.shape[0])

# Column names
print("Column names:" , data.columns.tolist())

# Data types
print("Data types:\n", data.dtypes)
```

```
number of rows: 307973
Column names: ['Accident_Index', 'Accident Date', 'Day_of_Week', 'Junction_Control', 'Junction_Det
Data types:
 Accident_Index                object
Accident Date                  object
Day_of_Week                    object
Junction_Control               object
Junction_Detail                object
Accident_Severity              object
Latitude                      float64
Light_Conditions               object
Local_Authority_(District)     object
Carriageway_Hazards            object
Longitude                     float64
Number_of_Casualties            int64
Number_of_Vehicles              int64
Police_Force                   object
Road_Surface_Conditions        object
Road_Type                      object
Speed_limit                     int64
Time                           object
Urban_or_Rural_Area            object
Weather_Conditions             object
Vehicle_Type                   object
dtype: object
```

Accident_Index: A unique identifier for each accident record.

Accident Date: The date on which the accident occurred (format: DD/MM/YYYY).

Day_of_Week: The day of the week when the accident took place.

Junction_Control: Describes the type of junction control at the accident location (e.g., "Give way or uncontrolled").

Junction_Detail: Provides additional details about the junction where the accident occurred (e.g., "T or staggered junction").

Accident_Severity: Indicates the severity of the accident (e.g., "Serious").

Latitude: The geographic latitude of the accident location.

Light_Conditions: Describes the lighting conditions at the time of the accident (e.g., "Daylight").

Local_Authority_(District): The local authority district where the accident occurred.

Carriageway_Hazards: Describes any hazards present on the carriageway at the time of the accident (e.g., "None").

Longitude: The geographic longitude of the accident location.

Number_of_Casualties: The total number of casualties involved in the accident.

Number_of_Vehicles: The total number of vehicles involved in the accident.

Police_Force: The police force that handled the accident.

Road_Surface_Conditions: Describes the surface conditions of the road at the time of the accident (e.g., "Dry").

Road_Type: Specifies the type of road where the accident occurred (e.g., "One way street").

Speed_limit: The speed limit applicable to the road where the accident occurred.

Time: The time of day when the accident happened (format: HH:MM).

Urban_or_Rural_Area: Indicates whether the accident occurred in an urban or rural area.

Weather_Conditions: Describes the weather conditions at the time of the accident (e.g., "Fine no high winds").

Vehicle_Type: Specifies the type of vehicle involved in the accident (e.g., "Car," "Taxi/Private hire car").

Cleaning the dataset

Remove unnecessary columns - features that are not important for prediction

```
data.drop(['Accident_Index', 'Police_Force', 'Accident Date', 'Local_Authority_(District)', 'Carriageway_Hazards'], axis=1, inplace=True)
print(data.shape)
data.head()
```

```
(307973, 16)
```

Remove rows with any missing values

```
data = data.dropna()
print(data.shape)
data.head()
```

```
(300495, 16)
```

Selecting rows to reduce the database by shuffle in order to maintain the class ratio

```python
import pandas as pd
import pandas as pd
import numpy as np

# Calculate the number of samples to delete from each severity class
# Convert the columns to their original data types
original_dtypes = data.dtypes

class_counts = data['Accident_Severity'].value_counts()
delete_counts = (class_counts * 0.5).astype(int)

# Shuffle the dataset
df_shuffled = data.sample(frac=1, random_state=42)

# Initialize an empty DataFrame to store the balanced dataset
df_balanced = pd.DataFrame(columns=data.columns)

# Iterate through each severity class and delete samples to achieve 50% reduction
for severity, count in delete_counts.items():
    df_cls = df_shuffled[df_shuffled['Accident_Severity'] == severity]
    df_balanced = pd.concat([df_balanced, df_cls.iloc[count:]])

# df_balanced now contains the balanced dataset with 50% of samples deleted while maintaining class rat
data=df_balanced
data = data.astype(original_dtypes)

print(data.shape)
data.head()
```

```
(150248, 16)
```

Before:

```python
[6] #The number of each Accident_Severity- the target
print(data["Accident_Severity"].value_counts())
```

```
Slight     263280
Serious     40740
Fatal        3904
Fetal          49
Name: Accident_Severity, dtype: int64
```

After:

```python
#The number of each Accident_Severity- the target
print(data["Accident_Severity"].value_counts())
```

```
Accident_Severity
Slight     128260
Serious     20042
Fatal        1922
Fetal          24
Name: count, dtype: int64
```

LR- Logistic Regression

```
[ ]  from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
     def measure_error(y_true, y_pred, label):
         return pd.Series({'accuracy':accuracy_score(y_true, y_pred),
                           'precision': precision_score(y_true, y_pred, average='macro'),
                           'recall': recall_score(y_true, y_pred, average='macro'),
                           'f1': f1_score(y_true, y_pred, average='macro')},
                          name=label)
```

```
from sklearn.linear_model import LogisticRegressionCV

# L1 regularized logistic regression
lr_l1 = LogisticRegressionCV(Cs=10, cv=4, penalty='l1', solver='saga').fit(x_train, y_train)
y_test_pred_l1 = lr_l1.predict(x_test)
print(measure_error(y_test, y_test_pred_l1, 'l1'))
```

```
accuracy     0.853644
precision    0.213411
recall       0.250000
f1           0.230261
Name: l1, dtype: float64
```

```
[ ]  # L2 regularized logistic regression
     lr_l2 = LogisticRegressionCV(Cs=10, cv=4, penalty='l2', solver='lbfgs').fit(x_train, y_train)
     y_test_pred_l2 = lr_l2.predict(x_test)
     print(measure_error(y_test, y_test_pred_l2, 'l2'))
```

```
accuracy     0.853644
precision    0.393608
recall       0.250707
f1           0.231789
Name: l2, dtype: float64
```

```
[ ]  from sklearn.linear_model import LogisticRegression

     # Standard logistic regression- none regularization
     lr_no_reg = LogisticRegression(C=100000).fit(x_train, y_train)
     y_test_pred_lr_no_reg = lr_no_reg.predict(x_test)
     print(measure_error(y_test, y_test_pred_lr_no_reg, 'lr_no_reg'))
```

```
accuracy     0.853622
precision    0.316388
recall       0.250239
f1           0.230840
Name: lr_no_reg, dtype: float64
```

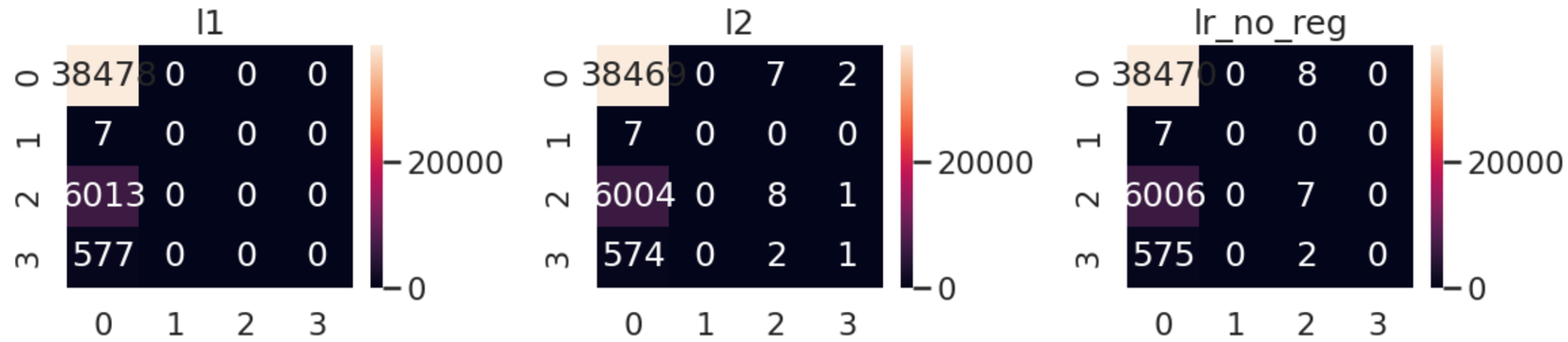
```
coefficients = list()

coeff_labels = ['l1', 'l2','lr_no_reg']
coeff_models = [lr_l1, lr_l2, lr_no_reg]

for lab,mod in zip(coeff_labels, coeff_models):
    coeffs = mod.coef_
    coeff_label = pd.MultiIndex(levels=[[lab], [0,1,2,3]], codes=[[0,0,0,0], [0,1,2,3]])
    coefficients.append(pd.DataFrame(coeffs.T, columns=coeff_label))

coefficients = pd.concat(coefficients, axis=1)

coefficients.sample(10)
```

	l1				l2				lr_no_reg		
	0	1	2	3	0	1	2	3	0	1	2
1	0.0	0.0	0.0	0.0	-0.818379	1.944895	-0.604809	-0.521707	-0.090843	-0.297512	0.14086
41	0.0	0.0	0.0	0.0	-0.054920	-0.068306	0.020803	0.102423	0.183941	-0.271900	0.0722
55	0.0	0.0	0.0	0.0	-0.052812	-0.144606	-0.010377	0.207795	0.173856	-0.040351	-0.03505
23	0.0	0.0	0.0	0.0	0.362906	-0.218598	0.06437				
14	0.0	0.0	0.0	0.0	0.059337	-0.277104	0.15387				
35	0.0	0.0	0.0	0.0	-0.248551	0.902169	-0.30513				
22	0.0	0.0	0.0	0.0	-0.080550	0.003313	-0.00949				
53	0.0	0.0	0.0	0.0	0.281504	-0.108543	0.20056				
5	0.0	0.0	0.0	0.0	0.217807	0.118655	0.13800				
34	0.0	0.0	0.0	0.0	0.278193	-0.528627	0.21150				

High accuracy level -> Low precision, recall and f1 -> Zero coefficients in l1 model -> L1 model always predicts zero.

Conclusion:
The database is not balanced -> Most of the samples have a zero classification.

```
[ ] fig, axList = plt.subplots(nrows=1, ncols=3)
    axList = axList.flatten()
    fig.set_size_inches(12, 3)

    #axList[-1].axis('off')

    for ax,lab in zip(axList, coeff_labels):
        sns.heatmap(cm[lab], ax=ax, annot=True, fmt='d');
        ax.set(title=lab);

    plt.tight_layout()
```



Balance The Data

Balance the database and selecting just 'Slight', ‘Fatal’ and 'Serious' classes.

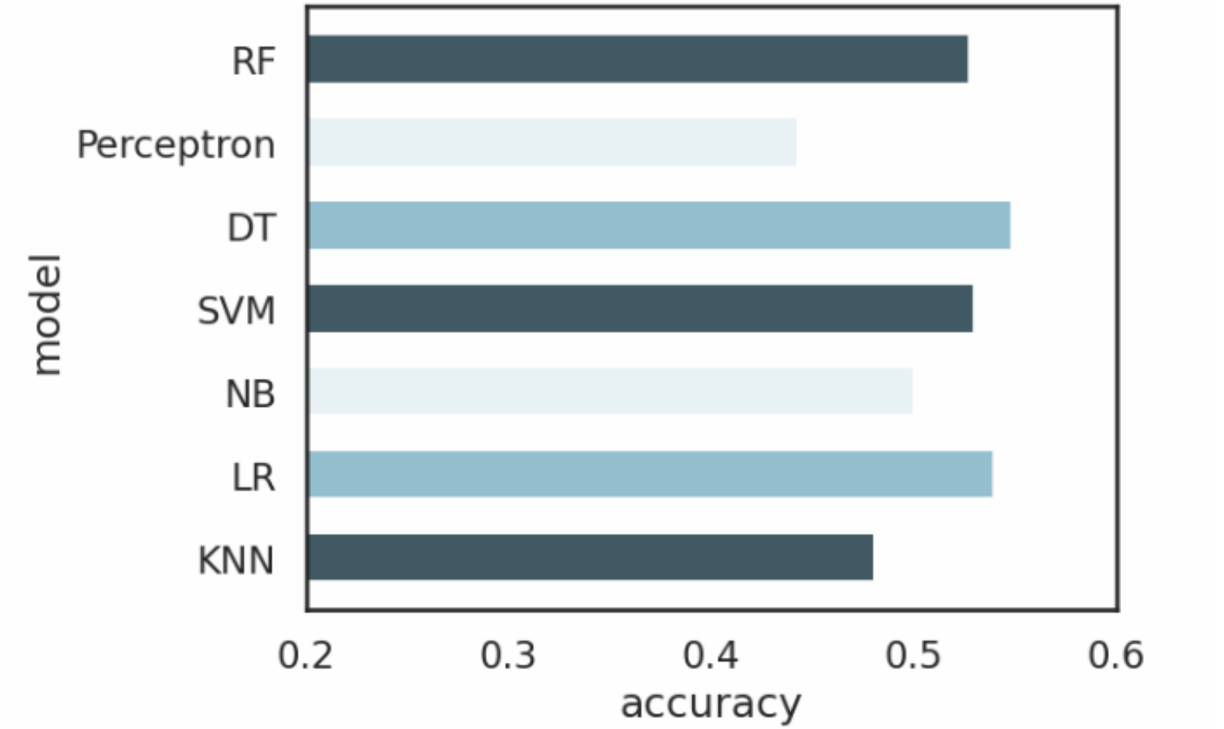
```
[6]  from sklearn.utils import resample

     # Assuming df is your dataframe
     # Remove rows with 'Fetal' class
     df_balanced = data[data['Accident_Severity'] != 'Fetal']

     # Balance the remaining classes
     # Separate majority and minority classes
     majority_class = df_balanced[df_balanced['Accident_Severity'] == 'Slight']
     minority_class = df_balanced[df_balanced['Accident_Severity'].isin(['Serious', 'Fatal'])]

     # Downsample the majority class
     majority_downsampled = resample(majority_class,
                                     replace=False,  # Sample without replacement
                                     n_samples=len(minority_class),  # Match minority class
                                     random_state=42)  # Reproducible results

     # Combine minority class with downsampled majority class
     df_balanced = pd.concat([majority_downsampled, minority_class])

     # Shuffle the data to mix the classes
     df_balanced = df_balanced.sample(frac=1, random_state=42)

     # Check the class distribution
     print(df_balanced['Accident_Severity'].value_counts())
```

```
Accident_Severity
Slight     43928
Serious    40084
Fatal       3844
Name: count, dtype: int64
```

```
[7]  import pandas as pd

     slight_samples = df_balanced[df_balanced['Accident_Severity'] == 'Slight'].sample(n=20000, random_state=1)
     serious_samples = df_balanced[df_balanced['Accident_Severity'] == 'Serious'].sample(n=20000, random_state=1)
     fatal_samples = df_balanced[df_balanced['Accident_Severity'] == 'Fatal']

     df_balanced = pd.concat([slight_samples, serious_samples, fatal_samples], ignore_index=True)

     # Shuffle the dataset
     df_balanced = df_balanced.sample(frac=1, random_state=1).reset_index(drop=True)

     # Check the class distribution
     df_balanced['Accident_Severity'].value_counts()
```

```
Accident_Severity
Serious    20000
Slight     20000
Fatal       3844
Name: count, dtype: int64
```

Models Results



Low success rates-> Trying other combination.

Balance The Data

Balance the database and selecting just 'Slight' and 'Serious' classes.

```
[ ]  from sklearn.utils import resample

     # Assuming df is your dataframe
     # Remove rows with 'Fetal' class
     df_balanced = data[data['Accident_Severity'] != 'Fetal']
     df_balanced = df_balanced[df_balanced['Accident_Severity'] != 'Fatal']

     # Balance the remaining classes
     # Separate majority and minority classes
     majority_class = df_balanced[df_balanced['Accident_Severity'] == 'Slight']
     minority_class = df_balanced[df_balanced['Accident_Severity'] == 'Serious']

     # Downsample the majority class
     majority_downsampled = resample(majority_class,
                                     replace=False,  # Sample without replacement
                                     n_samples=len(minority_class),  # Match minority class
                                     random_state=42)  # Reproducible results

     # Combine minority class with downsampled majority class
     df_balanced = pd.concat([majority_downsampled, minority_class])

     # Shuffle the data to mix the classes
     df_balanced = df_balanced.sample(frac=1, random_state=42)

     # Check the class distribution
     print(df_balanced['Accident_Severity'].value_counts())
```

```
Accident_Severity
Slight     40084
Serious    40084
Name: count, dtype: int64
```

Models Results

KNN

```
k=1, Accuracy=0.5278366803875099
k=2, Accuracy=0.5269635358197164
k=3, Accuracy=0.5343228971768326
k=4, Accuracy=0.5346555236788492
k=5, Accuracy=0.5368175959419567
```

LR

	l1	l2	lr_no_reg
precision	0.589297	0.589296	0.589338
recall	0.589289	0.589289	0.589331
fscore	0.589280	0.589281	0.589323
accuracy	0.589289	0.589289	0.589331
auc	0.589289	0.589289	0.589331

Decision Tree

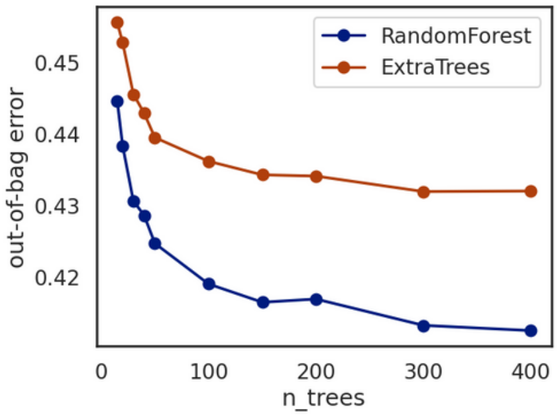
	train	test
accuracy	1.0	0.541266
precision	1.0	0.541402
recall	1.0	0.539376
f1	1.0	0.540387

LR- PCA

	train	test
accuracy	0.530730	0.533949
precision	0.532502	0.535633
recall	0.503617	0.510021
f1	0.517657	0.522513

Naive Base

```
{'gaussian': 0.5439576888534078,
 'bernoulli': 0.5780361241393075,
 'multinomial': 0.5559450154675183}
```



Gathering Conclusions And Rethinking

Trying different combinations of classifications -> About 50% success
(2 classifications) -> Random prediction.

The experiments are attached in Attachment1, Attachment2 and Attachment3 notebooks.

=> Trying with the most successful combination.

Balance The Data

Balance the database and selecting just 'Slight' and 'Fatal' classes.

```
from sklearn.utils import resample

# Remove rows with 'Fetal' class
df_balanced = data[data['Accident_Severity'] != 'Fetal']
df_balanced = df_balanced[df_balanced['Accident_Severity'] != 'Serious']

# Balance the remaining classes
# Separate majority and minority classes
majority_class = df_balanced[df_balanced['Accident_Severity'] == 'Slight']
minority_class = df_balanced[df_balanced['Accident_Severity'] == 'Fatal']

# Downsample the majority class
majority_downsampled = resample(majority_class,
                                replace=False,  # Sample without replacement
                                n_samples=len(minority_class),  # Match minority class
                                random_state=42)  # Reproducible results

# Combine minority class with downsampled majority class
df_balanced = pd.concat([majority_downsampled, minority_class])

# Shuffle the data to mix the classes
df_balanced = df_balanced.sample(frac=1, random_state=42)

# Check the class distribution
print(df_balanced['Accident_Severity'].value_counts())
```

```
Accident_Severity
Slight    3844
Fatal     3844
Name: count, dtype: int64
```

```
[ ]  data=df_balanced
     print(data.shape)
     data.head()
```

```
(7688, 16)
```

	Day_of_Week	Junction_Control	Junction_Detail	Accident_Severity	Lat:
225222	Saturday	Give way or uncontrolled	T or staggered junction	Slight	52.4
211353	Tuesday	Give way or uncontrolled	T or staggered junction	Fatal	54.4
233646	Tuesday	Give way or uncontrolled	T or staggered junction	Fatal	52.9
303069	Friday	Data missing or out of range	Not at junction or within 20 metres	Fatal	51.9
126383	Friday	Stop sign	Crossroads	Fatal	51.2

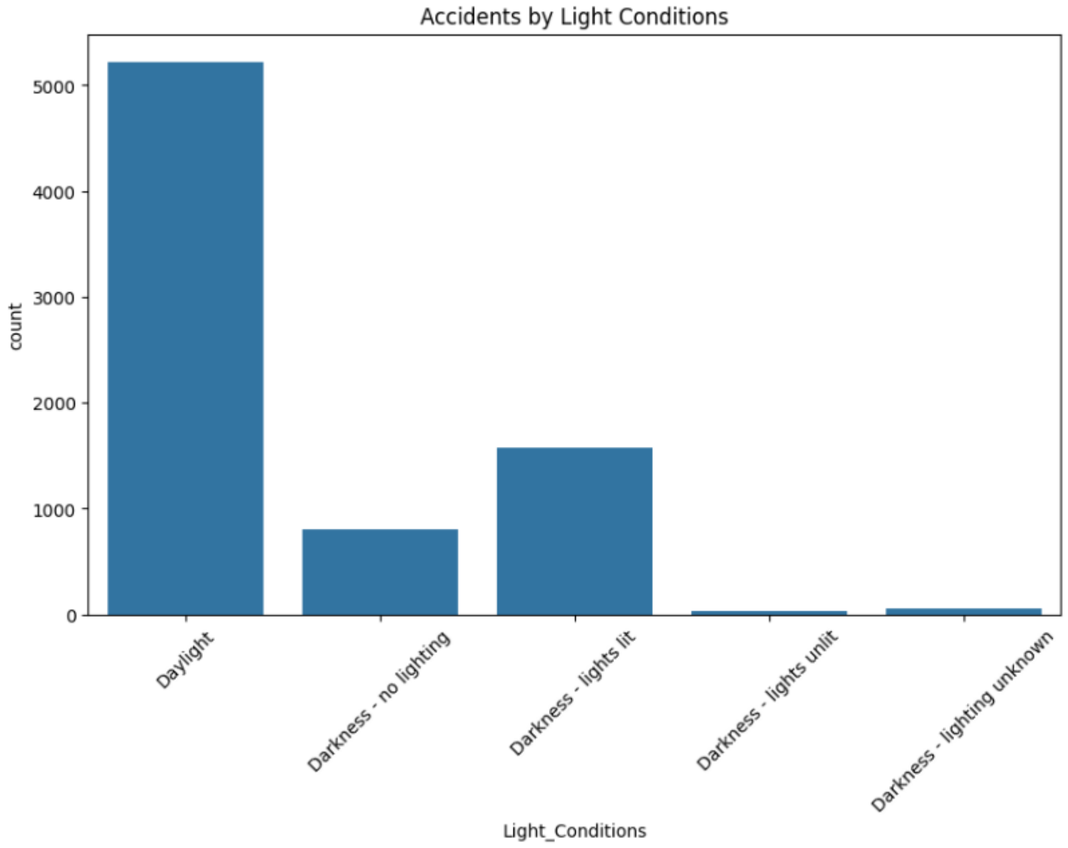
```
[ ]  # Number of rows
     print("number of rows:" , data.shape[0])

     # Column names
     print("Column names:" , data.columns.tolist())

     # Data types
     print("Data types:\n", data.dtypes)
```

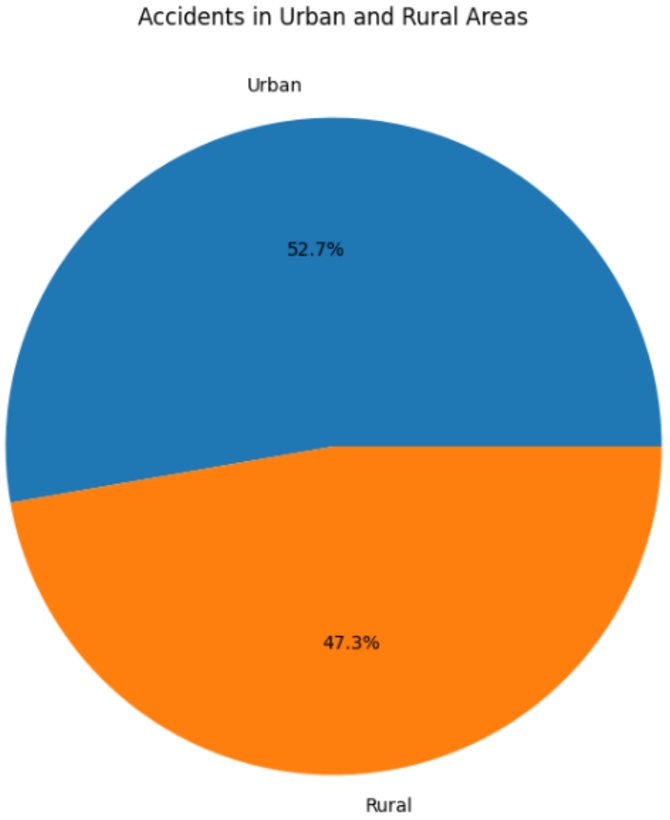
```
number of rows: 7688
Column names: ['Day_of_Week', 'Junction_Control',
Data types:
 Day_of_Week                 object
Junction_Control            object
Junction_Detail             object
Accident_Severity           object
Latitude                   float64
Light_Conditions            object
Longitude                  float64
Number_of_Casualties         int64
Number_of_Vehicles           int64
Road_Surface_Conditions     object
Road_Type                   object
Speed_limit                  int64
Time                        object
Urban_or_Rural_Area         object
Weather_Conditions          object
Vehicle_Type                object
dtype: object
```

```
[ ]  data.describe()
```

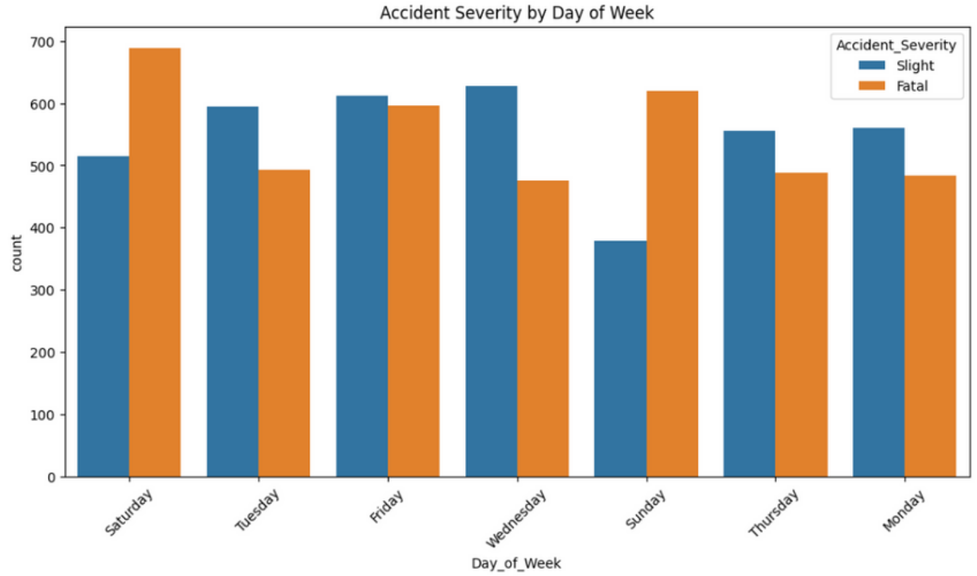
	Latitude	Longitude	Number_of_Casualties	Number_of_Vehicles	Speed_limit
count	7688.000000	7688.000000	7688.000000	7688.000000	7688.000000
mean	52.527337	-1.372048	1.571540	1.810224	42.982570
std	1.369806	1.366614	1.314708	0.819475	15.416298
min	50.099786	-6.432739	1.000000	1.000000	20.000000
25%	51.506540	-2.265708	1.000000	1.000000	30.000000
50%	52.257193	-1.349967	1.000000	2.000000	30.000000
75%	53.424255	-0.195891	2.000000	2.000000	60.000000
max	60.241195	1.751113	48.000000	16.000000	70.000000

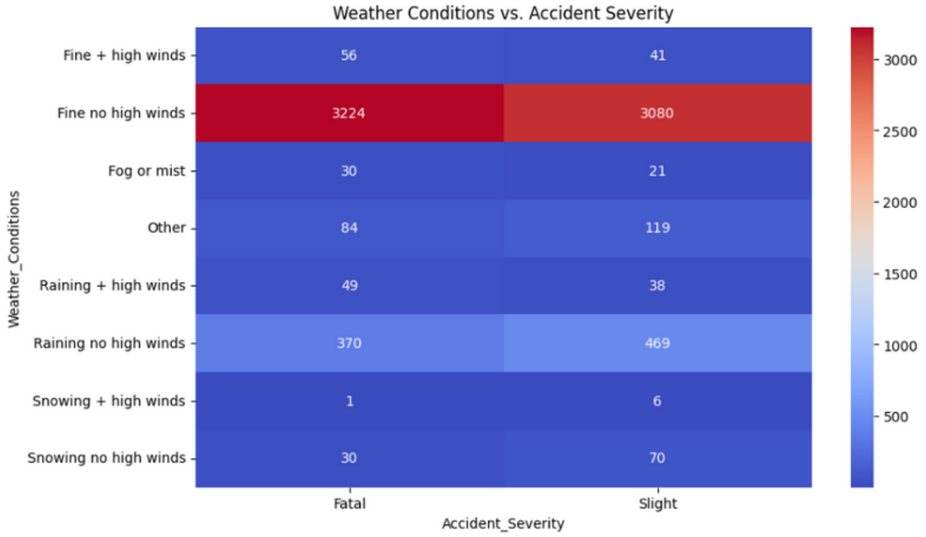
Diagrams


Accidents by Light Conditions
count
5000
4000
3000
2000
1000
0
Daylight
Darkness - no lighting
Darkness - lights lit
Darkness - lights unlit
Darkness - lighting unknown
Light_Conditions


Accidents in Urban and Rural Areas
Urban
52.7%
47.3%
Rural

Diagrams


Accident Severity by Day of Week
Accident_Severity
Slight
Fatal
count
0
100
200
300
400
500
600
700
Saturday
Tuesday
Friday
Wednesday
Sunday
Thursday
Monday
Day_of_Week


Weather Conditions vs. Accident Severity
Fine + high winds
Fine no high winds
Fog or mist
Other
Raining + high winds
Raining no high winds
Snowing + high winds
Snowing no high winds
Weather_Conditions
56
41
3224
3080
30
21
84
119
49
38
370
469
1
6
30
70
Fatal
Slight
Accident_Severity
3000
2500
2000
1500
1000
500

Encoding the data

Changing the data type in 'Time' column from continuous to discrete

```
[ ]  for index, row in data.iterrows():
         data.at[index, 'Time'] = int(row['Time'].split(':')[0])
         if data.at[index, 'Time'] == 0:
           data.at[index, 'Time'] = 24
     data['Time'] = data['Time'].astype(int)
     print(data['Time'])
```

```
225222    15
211353    10
233646    16
303069    21
126383     6
          ..
95725      8
107217    19
175764    19
301030     3
273601    18
Name: Time, Length: 7688, dtype: int64
```

change the name of 'Urban_or_Rural_Area' column to 'Is_Urban'

```
0s  data.rename(columns={'Urban_or_Rural_Area': 'Is_Urban'}, inplace=True)
```

Encoding 'Urban_or_Rural_Area' column

```
[ ]  #urban=1, rural=0
     from sklearn.preprocessing import LabelBinarizer

     lb = LabelBinarizer()
     data['Is_Urban'] = lb.fit_transform(data['Is_Urban'])
     print(data['Is_Urban'])
```

```
225222    1
211353    0
233646    1
303069    0
126383    0
         ..
95725     0
107217    0
175764    1
301030    0
273601    0
Name: Is_Urban, Length: 7688, dtype: int64
```

Encoding the target column 'Accident_Severity'

```
from sklearn.preprocessing import OrdinalEncoder

# Sample dataset
category = ['Slight', 'Fatal']

# Initialize OrdinalEncoder
encoder = OrdinalEncoder(categories=[category])

data_reshaped=data['Accident_Severity'].values.reshape(-1,1)

# Fit and transform the data
data['Accident_Severity'] = encoder.fit_transform(data_reshaped)

data['Accident_Severity'] = data['Accident_Severity'].astype(int)
# Display the encoded data
data.head()
```

	Day_of_Week	Junction_Control	Junction_Detail	Accident_Severity	Lat
225222	Saturday	Give way or uncontrolled	T or staggered junction	0	52.4
211353	Tuesday	Give way or uncontrolled	T or staggered junction	1	54.4
233646	Tuesday	Give way or uncontrolled	T or staggered junction	1	52.9
303069	Friday	Data missing or out of range	Not at junction or within 20 metres	1	51.9
126383	Friday	Stop sign	Crossroads	1	51.2

For other object features- OneHotEncoder

```
[ ] from sklearn.preprocessing import OneHotEncoder, LabelEncoder

    # Copy of the data
    data_ohc = data.copy()

    # The encoders
    le = LabelEncoder()
    ohc = OneHotEncoder()

    for col in num_ohc_cols.index:

        # Integer encode the string categories
        dat = le.fit_transform(data_ohc[col]).astype(int)

        # Remove the original column from the dataframe
        data_ohc = data_ohc.drop(col, axis=1)

        # One hot encode the data--this returns a sparse array
        new_dat = ohc.fit_transform(dat.reshape(-1,1))

        # Create unique column names
        n_cols = new_dat.shape[1]
        col_names = ['_'.join([col, str(x)]) for x in range(n_cols)]

        # Create the new dataframe
        new_df = pd.DataFrame(new_dat.toarray(),
                              index=data_ohc.index,
                              columns=col_names)

        # Append the new data to the dataframe
        data_ohc = pd.concat([data_ohc, new_df], axis=1)
```

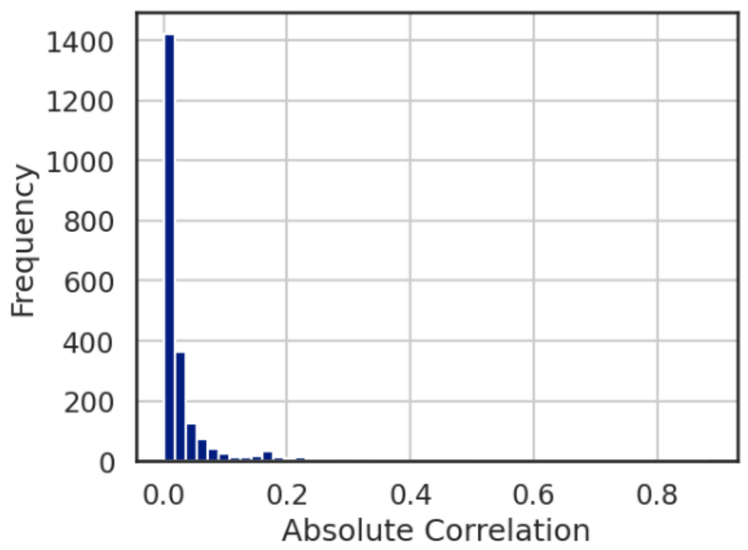
```
[ ] data_ohc
```

	Accident_Severity	Latitude	Longitude	Number_of_Casualties	Number_of_Vehicles	Speed_limit
225222	0	52.449115	-1.821632	1	2	30
211353	1	54.487235	-1.191248	3	2	60
233646	1	52.986152	-2.115708	1	1	30
303069	1	51.919042	-2.912664	1	1	60
126383	1	51.283235	0.065164	2	2	60

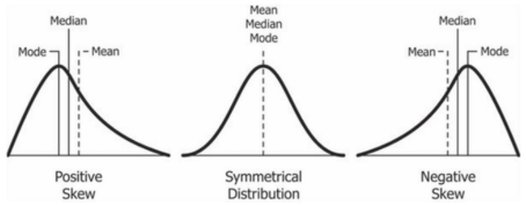
Correlation

```
# Calculate the correlation values
feature_cols = data_ohc.columns[:-1]
corr_values = data_ohc[feature_cols].corr()

# Simplify by emptying all the data below the diagonal
tril_index = np.tril_indices_from(corr_values)

# Make the unused values NaNs
for coord in zip(*tril_index):
    corr_values.iloc[coord[0], coord[1]] = np.NaN

# Stack the data and convert to a data frame
corr_values = (corr_values.stack().to_frame().reset_index()
               .rename(columns={'level_0':'feature1','level_1':'feature2',0:'correlation'}))

# Get the absolute values for sorting
corr_values['abs_correlation'] = corr_values.correlation.abs()
```

```
#A histogram of the absolute value correlations.
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.set_context('talk')
sns.set_style('white')
sns.set_palette('dark')

ax = corr_values.abs_correlation.hist(bins=50)

ax.set(xlabel='Absolute Correlation', ylabel='Frequency');
```



Skewness




```
data= data_ohc
y_col = 'Accident_Severity'
feature_cols = [x for x in data.columns if x != y_col]
x_data = data[feature_cols]
y_data = data[y_col]
```

```
# Create a list of float and int columns (excluding the target) to check for skewing
mask = ((data.dtypes == float) | (data.dtypes == int)) & (data.columns != 'Accident_Severity')
numeric_cols = data.columns[mask]
skew_limit = 0.75
skew_vals = x_data[numeric_cols].skew()

skew_cols = (skew_vals
             .sort_values(ascending=False)
             .to_frame()
             .rename(columns={0:'Skew'})
             .query('abs(Skew) > {0}'.format(skew_limit)))


skew_cols
```

	Skew
Junction_Control_1	50.603026
Vehicle_Type_11	50.603026
Weather_Conditions_6	33.101574
Junction_Control_0	29.181427
Vehicle_Type_0	22.577294
Vehicle_Type_5	21.856086
Road_Surface_Conditions_1	21.856086
Light_Conditions_2	16.482751
Junction_Control_6	15.167853
Junction_Detail_1	12.281099

Reducing skewness by applying transformations that make the distribution more symmetric- log transformation.

```
# Mute the setting wtih a copy warnings
pd.options.mode.chained_assignment = None

for col in skew_cols.index.tolist():
    x_data[col] = np.log1p(x_data[col])
```

Checking that the transformation worked

```
[ ]  # Check the skewness again
     skew_vals_after = x_data[numeric_cols].skew()
     skew_cols_after = (skew_vals_after
                        .sort_values(ascending=False)
                        .to_frame()
                        .rename(columns={0: 'Skew'})
                        .query('abs(Skew) > {0}'.format(skew_limit)))

     skew_cols_after
```

	Skew
Junction_Control_1	50.603026
Vehicle_Type_11	50.603026
Weather_Conditions_6	33.101574
Junction_Control_0	29.181427
Vehicle_Type_0	22.577294
Road_Surface_Conditions_1	21.856086
Vehicle_Type_5	21.856086
Light_Conditions_2	16.482751
Junction_Control_6	15.167853
Junction_Detail_1	12.281099

The skew remains the same...

Applying Other Transformations

```
[ ]  # Winsorization (replace values above 95th percentile with 95th percentile value)
     for col in skew_cols.index.tolist():
         percentile_95 = np.percentile(x_data[col], 95)
         x_data[col] = np.where(x_data[col] > percentile_95, percentile_95, x_data[col])
```

```
[ ]  from sklearn.preprocessing import RobustScaler

     # Create RobustScaler object
     scaler = RobustScaler()

     # Apply RobustScaler to the skewed column
     x_data[skew_cols.index.tolist()] = scaler.fit_transform(x_data[skew_cols.index.tolist()])
```

The skew improved

	Skew
Junction_Detail_6	3.860414
Junction_Detail_0	2.945544
Junction_Control_2	2.865649
Junction_Control_5	2.584877
Light_Conditions_3	2.578084
Weather_Conditions_5	2.507637
Day_of_Week_3	2.203288
Day_of_Week_1	2.128326
Day_of_Week_4	2.128326
Day_of_Week_5	2.057347
Day_of_Week_6	2.034502

Scaling ⚖️

Using 'MinMaxScaler' when you need to bound the data to a specific range, especially if your data does not follow a normal distribution or if you are using algorithms that require the data to be on a similar scale, such as KNN.

```
[ ]  # Mute the sklearn warning
     import warnings
     warnings.filterwarnings('ignore', module='sklearn')

     from sklearn.preprocessing import MinMaxScaler

     msc = MinMaxScaler()
     x_data = pd.DataFrame(msc.fit_transform(x_data),  # this is an np.array, not a dataframe.
                           columns=x_data.columns)
     x_data
```

	Latitude	Longitude	Number_of_Casualties	Number_of_Vehicles	Speed_limit	Time	Is_Urban	Vehi
0	0.487209	0.563440	0.000000	0.584963	0.2	0.608696	1.0	
1	0.892835	0.640467	0.756471	0.584963	0.8	0.391304	0.0	
2	0.595572	0.527506	0.000000	0.000000	0.2	0.652174	1.0	
3	0.379179	0.430124	0.000000	0.000000	0.8	0.869565	0.0	
4	0.248163	0.793991	0.442507	0.584963	0.8	0.217391	0.0	

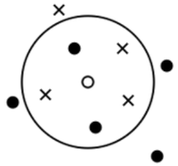
Train And Test

```
[ ]  from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test = train_test_split(x_data, y_data,stratify=y_data, test_size=0.3)
```

```
[ ]  x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

```
((5381, 67), (5381,), (2307, 67), (2307,))
```

KNN



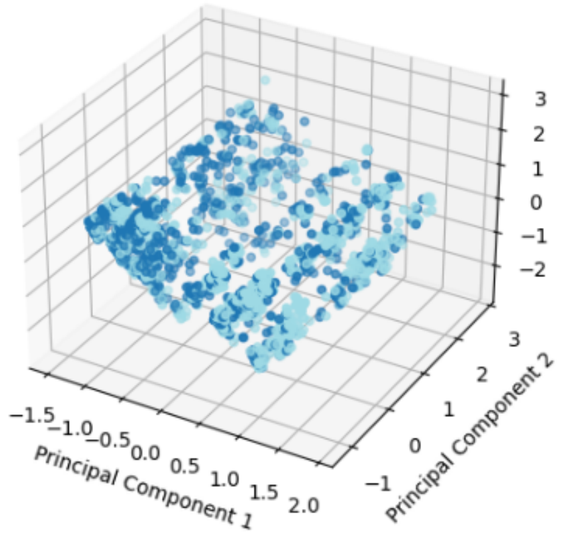
```
[ ]  # Function to calculate the % of values that were correctly predicted

     def accuracy(real, predict):
         return sum(real == predict) / float(real.shape[0])
```

```
[ ]  from sklearn.neighbors import KNeighborsClassifier

     results_dict={}
     for k in range(1,6):
         knn = KNeighborsClassifier(n_neighbors=k)

         knn.fit(x_train, y_train)

         y_pred = knn.predict(x_test)

         # Calculate accuracy
         acc = accuracy(y_test, y_pred)

         # Store results in the dictionary
         results_dict[k] = acc

     for k, result in results_dict.items():
         print(f'k={k}, Accuracy={result}')
```

```
k=1, Accuracy=0.6176853055916776
k=2, Accuracy=0.599479843953186
k=3, Accuracy=0.6246207195491981
k=4, Accuracy=0.6272214997832684
k=5, Accuracy=0.6480277416558301
```

PCA

```
[ ]  from sklearn import decomposition
     import matplotlib.pyplot as plt

     pca = decomposition.PCA(n_components=3, whiten=True)
     pca.fit(x_train)
     test_2d = pca.fit_transform(x_test)
```



	train	test
accuracy	0.615499	0.614651
precision	0.611171	0.609053
recall	0.634572	0.641248
f1	0.622652	0.624736

LR- Logistic Regression

'saga': This is an extension of the SAG solver that also supports L1 regularization. It is recommended for large datasets and problems with a large number of features. It can handle both L1- lasso and L2- ridge penalties and is suitable for multi-class classification.

```
from sklearn.linear_model import LogisticRegressionCV

# L1 regularized logistic regression
lr_l1 = LogisticRegressionCV(Cs=10, cv=4, penalty='l1', solver='saga').fit(x_train, y_train)
y_test_pred_l1 = lr_l1.predict(x_test)
print("l1:", accuracy(y_test, y_test_pred_l1))
```

```
l1: 0.6913740788903338
```

```
[ ]  print(lr_l1.Cs_)
     best_C = lr_l1.C_[0]
     print("Best C value:", best_C)
```

```
[1.00000000e-04 7.74263683e-04 5.99484250e-03 4.64158883e-02
 3.59381366e-01 2.78255940e+00 2.15443469e+01 1.66810054e+02
 1.29154967e+03 1.00000000e+04]
Best C value: 166.81005372000558
```

```
[ ]  # L2 regularized logistic regression
     lr_l2 = LogisticRegressionCV(Cs=10, cv=4, penalty='l2', solver='lbfgs').fit(x_train, y_train)
     y_test_pred_l2 = lr_l2.predict(x_test)
     print("l2:", accuracy(y_test, y_test_pred_l2))
```

```
l2: 0.6905071521456437
```

```
[ ]  print(lr_l2.Cs_)
     best_C = lr_l2.C_[0]
     print("Best C value:", best_C)
```

```
[1.00000000e-04 7.74263683e-04 5.99484250e-03 4.64158883e-02
 3.59381366e-01 2.78255940e+00 2.15443469e+01 1.66810054e+02
 1.29154967e+03 1.00000000e+04]
Best C value: 2.782559402207126
```

```
[ ]  # Standard logistic regression- none regularization
     lr_no_reg = LogisticRegression(C=100000).fit(x_train, y_train)
     y_test_pred_lr_no_reg = lr_no_reg.predict(x_test)
     print("lr_no_reg:", accuracy(y_test, y_test_pred_lr_no_reg))
```

```
lr_no_reg: 0.6909406155179887
```

Coefficients

```
[ ]  coeff_labels = ['l1', 'l2','lr_no_reg']
     coeff_models = [lr_l1, lr_l2, lr_no_reg]
     for lab,mod in zip(coeff_labels, coeff_models):
       print(lab, ":", np.max(np.abs(mod.coef_)))
```

```
l1 : 1.6653924157245013
l2 : 1.4202423418727952
lr_no_reg : 1.657904781505105
```

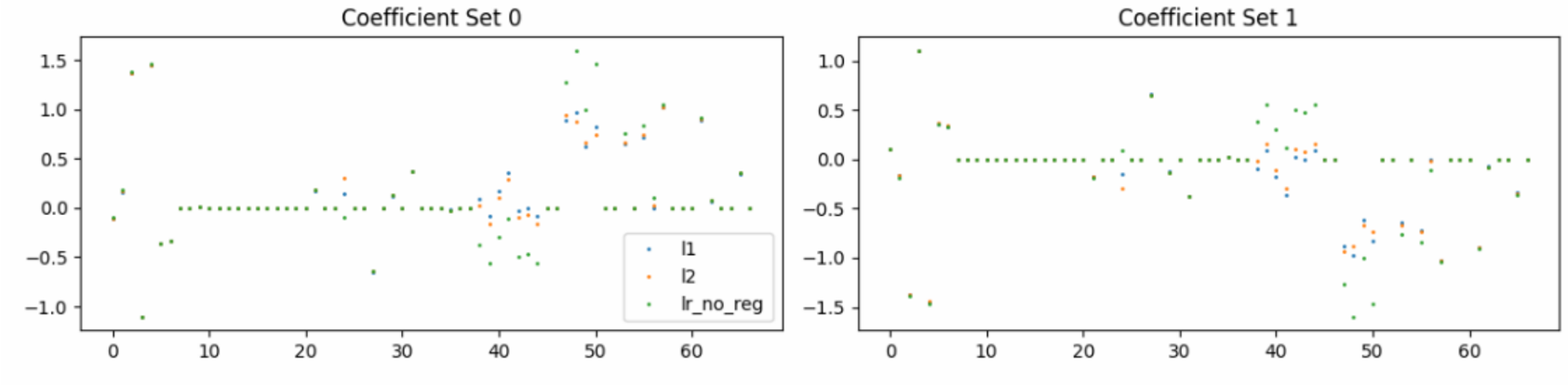
Sum of the coefficients

```
[ ]  coeff_labels = ['l1', 'l2','lr_no_reg']
     coeff_models = [lr_l1, lr_l2, lr_no_reg]
     for lab,mod in zip(coeff_labels, coeff_models):
       print(lab, ":", np.sum(np.abs(mod.coef_)))
```

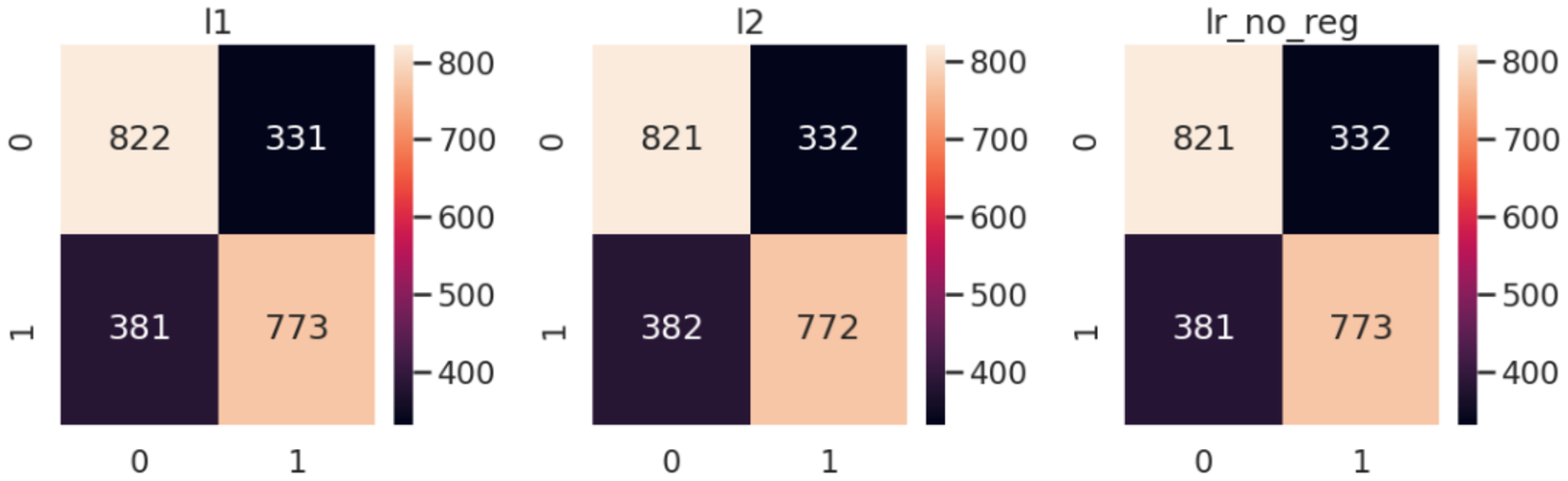
```
l1 : 16.312400922044908
l2 : 14.016863198257104
lr_no_reg : 17.602255287962016
```

```
[ ]  coefficients = []

     coeff_labels = ['l1', 'l2', 'lr_no_reg']
     coeff_models = [lr_l1, lr_l2, lr_no_reg]

     for label, model in zip(coeff_labels, coeff_models):
         coeffs = model.coef_
         if len(coeffs.shape) == 1:
             # Reshape 1D array to 2D array with one row
             coeffs = np.array([coeffs, -coeffs])
         else:
             # For multi-class classification, negate coefficients of one class
             coeffs = np.vstack((coeffs, -coeffs))
         for i in range(coeffs.shape[0]):
             coeff_label = pd.MultiIndex.from_tuples([(label, f'class_{i}')])
             coefficients.append(pd.DataFrame(coeffs[i], index=x_train.columns, columns=coeff_label))

     coefficients_df = pd.concat(coefficients, axis=1)
     coefficients_df.sample(10)
```

	l1		l2		lr_no_reg	
	class_0	class_1	class_0	class_1	class_0	class_1
Day_of_Week_4	-0.023735	0.023735	-0.096465	0.096465	-0.498671	0.498671
Junction_Control_1	0.000000	-0.000000	0.000000	-0.000000	0.000000	-0.000000
Junction_Control_2	0.886338	-0.886338	0.935522	-0.935522	1.267495	-1.267495
Road_Type_1	0.000000	-0.000000	0.000000	-0.000000	0.000000	-0.000000
Latitude	-0.105740	0.105740	-0.108477	0.108477	-0.097871	0.097871
Road_Type_4	0.000000	-0.000000	0.000000	-0.000000	0.000000	-0.000000
Junction_Detail_3	0.150560	-0.150560	0.298457	-0.298457	-0.090555	0.090555
Junction_Control_4	0.617634	-0.617634	0.669229	-0.669229	0.999025	-0.999025
Junction_Detail_7	0.000000	-0.000000	0.000000	-0.000000	0.000000	-0.000000
Day_of_Week_1	-0.084301	0.084301	-0.156606	0.156606	-0.557509	0.557509



Results



	l1	l2	lr_no_reg
precision	0.691740	0.690872	0.691291
recall	0.691374	0.690507	0.690941
fscore	0.691232	0.690365	0.690804
accuracy	0.691374	0.690507	0.690941
auc	0.691383	0.690516	0.690950

Naive bayes

```
print(data.dtypes)
```

```
Accident_Severity          int64
Latitude                 float64
Longitude                float64
Number_of_Casualties       int64
Number_of_Vehicles         int64
                           ...
Road_Type_0              float64
Road_Type_1              float64
Road_Type_2              float64
Road_Type_3              float64
Road_Type_4              float64
Length: 69, dtype: object
```

MultinomialNB model knows to convert continuous data into discrete data. BernoulliNB model doesn't do it in a good way.

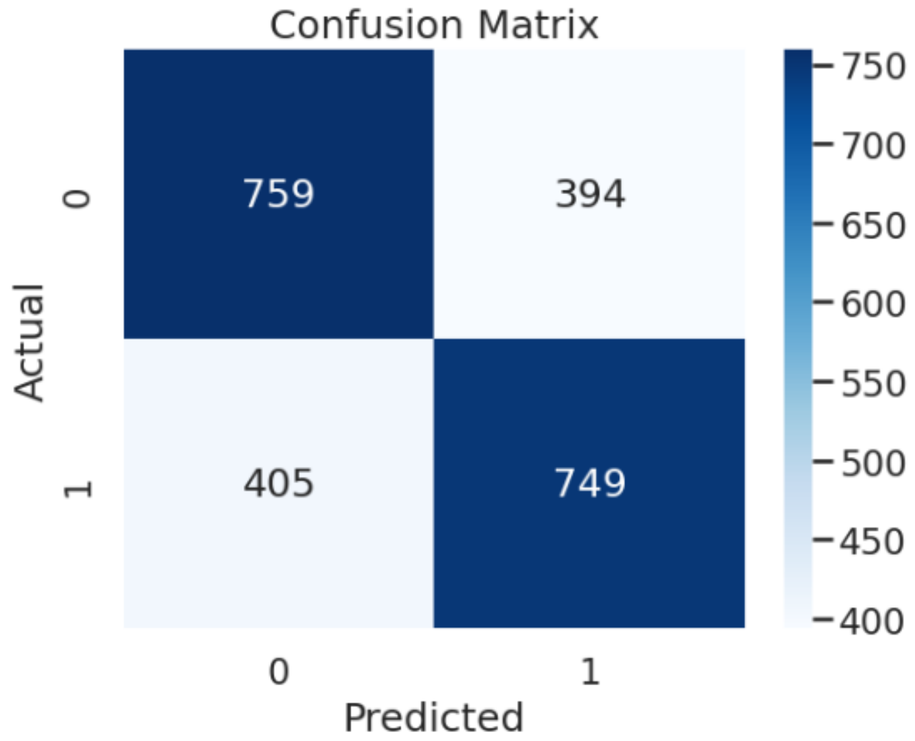
Results

```
[ ]  from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
     from sklearn.model_selection import cross_val_score

     nb = {'gaussian': GaussianNB(),
           'bernoulli': BernoulliNB(),
           'multinomial': MultinomialNB()}
     scores = {}
     cv_N = 4
     for key, model in nb.items():
         s = cross_val_score(model, x_data, y_data, cv=cv_N, n_jobs=cv_N, scoring='accuracy')
         scores[key] = np.mean(s)
     scores
```

```
{'gaussian': 0.6545265348595213,
 'bernoulli': 0.6389177939646202,
 'multinomial': 0.6352757544224765}
```

GaussianNB Results

```
[102] #Fit a GaussianNB to the training split.
      model = GaussianNB()
      model.fit(x_train, y_train)

      #Get predictions on the test set.
      predictions = model.predict(x_test)
```



Discretize The Data

```
[ ] # Create X_discrete from X using .rank(pct=True)
    X_discrete = x_data.rank(pct=True)
    X_discrete
```

	Latitude	Longitude	Number_of_Casualties	Number_of_Vehicles	Speed_limit	Time	Is_Urban	Vehicle_Type_0	Vehicle_Type_1	Vehicle_Type_2
0	0.536941	0.366415	0.339815	0.619147	0.265609	0.541103	0.736342	0.500065	0.500065	0.603993
1	0.908169	0.537721	0.906998	0.619147	0.777640	0.270421	0.236342	0.500065	0.500065	0.603993
2	0.662201	0.295005	0.339815	0.181907	0.265609	0.611993	0.736342	0.500065	0.500065	0.603993
3	0.447190	0.140609	0.339815	0.181907	0.777640	0.895096	0.236342	0.500065	0.500065	0.103993
4	0.129943	0.885276	0.774909	0.619147	0.777640	0.090010	0.236342	0.500065	0.500065	0.603993
...	...	...	...	...	...	...	...	...	...	...
7683	0.587019	0.983091	0.339815	0.181907	0.948946	0.174493	0.236342	0.500065	0.500065	0.603993
7684	0.266519	0.656868	0.906998	0.937305	0.570695	0.814581	0.236342	0.500065	0.500065	0.603993
7685	0.164412	0.893470	0.339815	0.619147	0.265609	0.814581	0.736342	0.500065	0.500065	0.603993
7686	0.696410	0.130983	0.339815	0.181907	0.777640	0.044875	0.236342	0.500065	0.500065	0.603993
7687	0.072581	0.542144	0.339815	0.937305	0.777640	0.760731	0.236342	0.500065	0.500065	0.603993

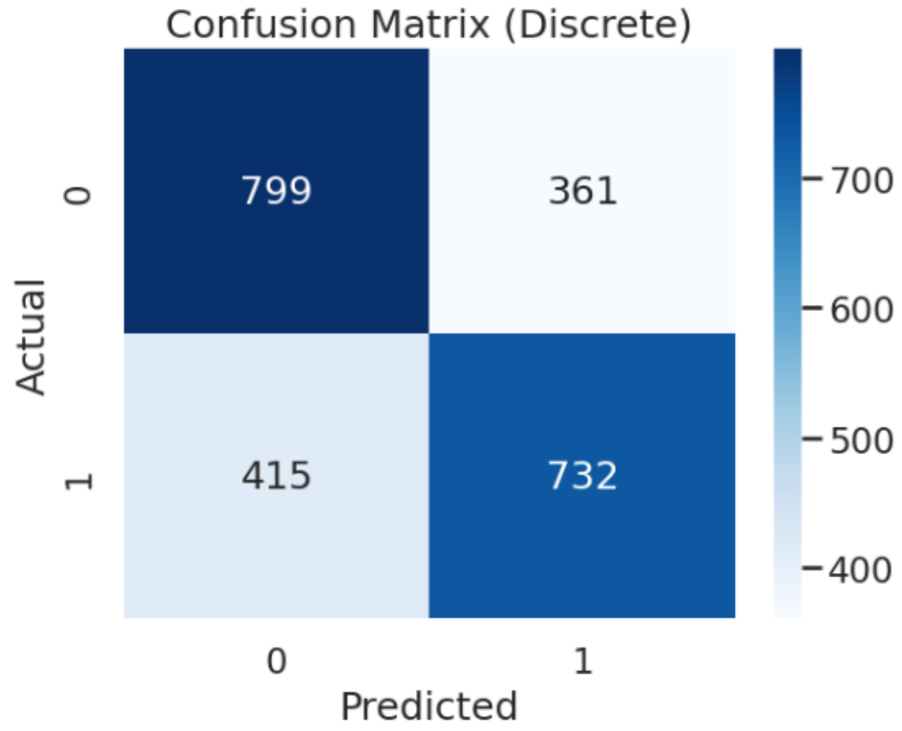
7688 rows × 67 columns

```
[ ] # Modify X_discrete so that it is indeed discrete
    X_discrete = X_discrete.applymap(lambda x: int(round(x * 100)))  # Convert percentile ranks to integers (rounded) and extract first 2 digits
    X_discrete
```

	Latitude	Longitude	Number_of_Casualties	Number_of_Vehicles	Speed_limit	Time	Is_Urban	Vehicle_Type_0	Vehicle_Type_1	Vehicle_Type_2
0	54	37	34	62	27	54	74	50	50	60
1	91	54	91	62	78	27	24	50	50	60
2	66	30	34	18	27	61	74	50	50	60
3	45	14	34	18	78	90	24	50	50	10
4	13	89	77	62	78	9	24	50	50	60

MultinomialNB Results

```
# Split X_discrete and y into training and test datasets
X_train_discrete, X_test_discrete, y_train_discrete, y_test_discrete =
train_test_split(X_discrete, y_data, test_size=0.3, random_state=42)
# Fit a MultinomialNB to the training split
model_discrete = MultinomialNB()
model_discrete.fit(X_train_discrete, y_train_discrete)

# Get predictions on the test set
predictions_discrete = model_discrete.predict(X_test_discrete)
```




```
[ ]  from sklearn.metrics import accuracy_score

     # Calculate the accuracy of the model
     accuracy = accuracy_score(y_test_discrete, predictions_discrete)
     print(f"Accuracy: {accuracy:.2f}")

     Accuracy: 0.66
```

SVM-Support Vector Machines

```
[ ] from sklearn.svm import SVC
    from sklearn.model_selection import GridSearchCV
    from sklearn.metrics import classification_report

    param_grid = {
        'C': [0.1, 1, 100],
        'degree': [3, 5],
        'kernel': ['poly']
    }
    # Create the grid search object
    grid_search = GridSearchCV(SVC(), param_grid, cv=4, scoring='accuracy', verbose=3, n_jobs=-1)

    # Fit the grid search to the data
    grid_search.fit(x_train, y_train)

    # Print the best parameters
    print("Best parameters:", grid_search.best_params_)

    # Predict using the best model
    y_pred = grid_search.predict(x_test)

    # Print the classification report
    print(classification_report(y_test, y_pred))
```

```
Fitting 4 folds for each of 6 candidates, totalling 24 fits
Best parameters: {'C': 1, 'degree': 3, 'kernel': 'poly'}
              precision    recall  f1-score   support

           0       0.69      0.73      0.71      1154
           1       0.71      0.67      0.69      1153

    accuracy                           0.70      2307
   macro avg       0.70      0.70      0.70      2307
weighted avg       0.70      0.70      0.70      2307
```

The model has achieved an accuracy of approximately 70%.

```
[ ]  from sklearn.svm import SVC
     from sklearn.model_selection import GridSearchCV

     # Define parameters for cross-validation
     param_grid = {'C': [0.1, 1, 10, 100],
                   'gamma': [0.01, 0.1, 10]}

     # Create SVM model
     svm = SVC(kernel='rbf')

     # Perform grid search with cross-validation
     grid_search = GridSearchCV(svm, param_grid, cv=4, scoring='accuracy')
     grid_search.fit(x_train, y_train)

     # Get the best parameters
     best_params = grid_search.best_params_

     print("Best Parameters:", best_params)

     # Predict using the best model
     y_pred = grid_search.predict(x_test)

     # Print the classification report
     print(classification_report(y_test, y_pred))
```

```
Best Parameters: {'C': 1, 'gamma': 0.1}
              precision    recall  f1-score   support

           0       0.67      0.70      0.68      1153
           1       0.69      0.66      0.67      1154

    accuracy                           0.68      2307
   macro avg       0.68      0.68      0.68      2307
weighted avg       0.68      0.68      0.68      2307
```

Based on these results, the SVM with RBF kernel performs slightly worse than the SVM with polynomial (poly) kernel in terms of F1-score and accuracy.

Decision Tree

```
[ ]  from sklearn.tree import DecisionTreeClassifier

     dt = DecisionTreeClassifier(random_state=42)
     dt = dt.fit(x_train, y_train)
     print("node count: " ,dt.tree_.node_count,"max depth: ", dt.tree_.max_depth)
```

```
node count:  2667 max depth:  28
```

```
[ ]  # The error on the training and test data sets
     y_train_pred = dt.predict(x_train)
     y_test_pred = dt.predict(x_test)

     train_test_full_error = pd.concat([measure_error(y_train, y_train_pred, 'train'),
                                        measure_error(y_test, y_test_pred, 'test')],
                                        axis=1)

     train_test_full_error
```

	train	test
accuracy	1.0	0.622887
precision	1.0	0.627240
recall	1.0	0.606586
f1	1.0	0.616740

```
[ ] from io import StringIO
    from IPython.display import Image, display

    from sklearn.tree import export_graphviz

    try:
        import pydotplus
        pydotplus_installed = True

    except:
        print('PyDotPlus must be installed to execute the remainder of the cells associated with this question.')
        print('Please see the instructions for this question for details.')
        pydotplus_installed = False
    if pydotplus_installed:

        # Create an output destination for the file
        dot_data = StringIO()

        export_graphviz(dt, out_file=dot_data, filled=True)
        graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
        print(graph)
        # View the tree image
        filename = 'Car_Accidents_tree.png'
        graph.write_png(filename)
        img = Image(filename=filename)
        display(img)

    else:
        print('This cell not executed because PyDotPlus could not be loaded.')
```

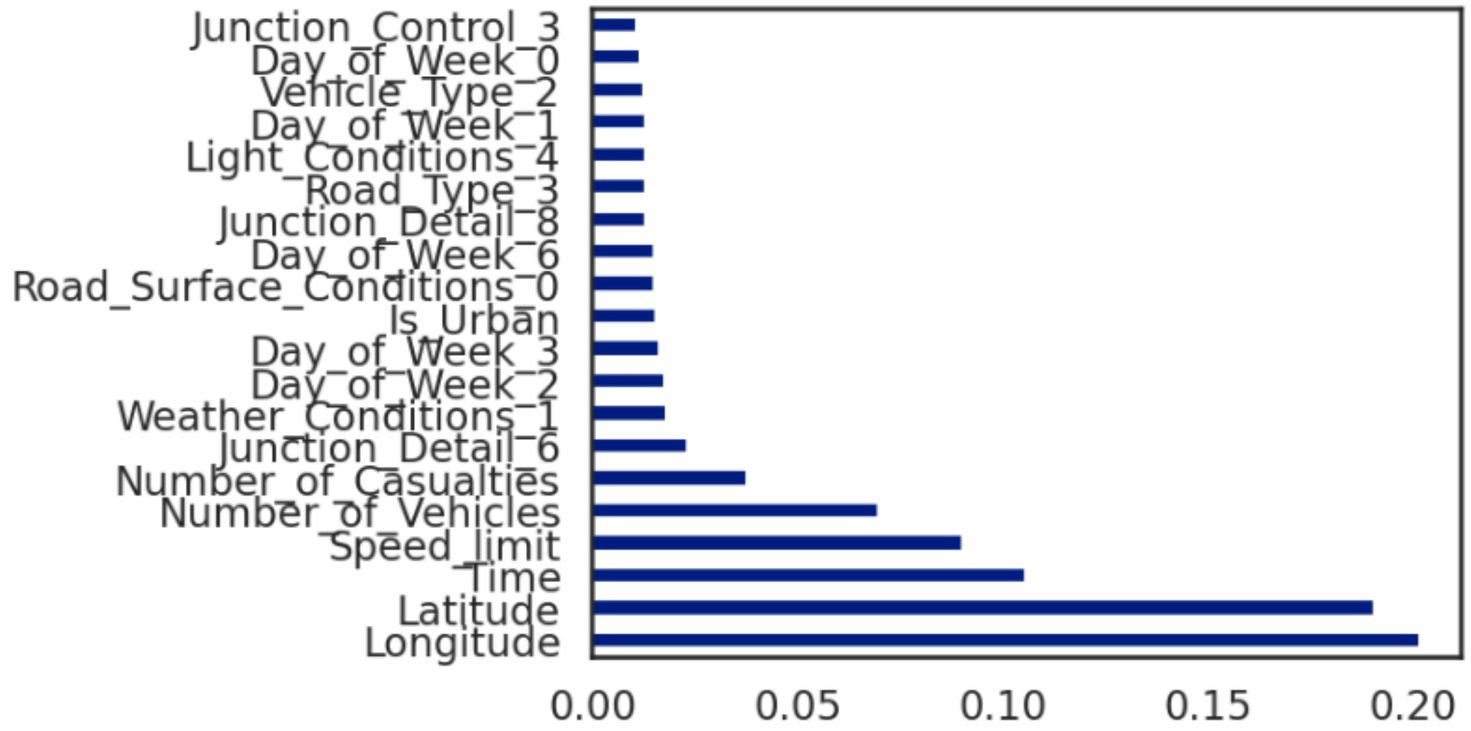
```
<pydotplus.graphviz.Dot object at 0x7b5e5cf5b340>
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.552293 to fit
```

The tree is very large and can be seen in the notebook

Decision Trees- Importance Of Features

```
[ ]  feature_imp = pd.Series(dt.feature_importances_, index=x_train.columns).sort_values(ascending=False)
     feature_imp.nlargest(20).plot(kind='barh')
```

```
<Axes: >
```



Pruning ✂

```
[ ]  # Prune the decision tree using cost complexity pruning
     path = dt.cost_complexity_pruning_path(x_train, y_train)
     ccp_alphas, impurities = path.ccp_alphas, path.impurities

     # Create a list of decision trees for each alpha
     clfs = []
     for ccp_alpha in ccp_alphas:
         clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
         clf.fit(x_train, y_train)
         clfs.append(clf)

     # Find the best alpha
     test_scores = [clf.score(x_test, y_test) for clf in clfs]
     best_alpha = ccp_alphas[np.argmax(test_scores)]

     # Refit the tree with the best alpha
     pruned_tree = DecisionTreeClassifier(random_state=42, ccp_alpha=best_alpha)
     pruned_tree.fit(x_train, y_train)

     print("After pruning - Depth:", pruned_tree.get_depth())
```

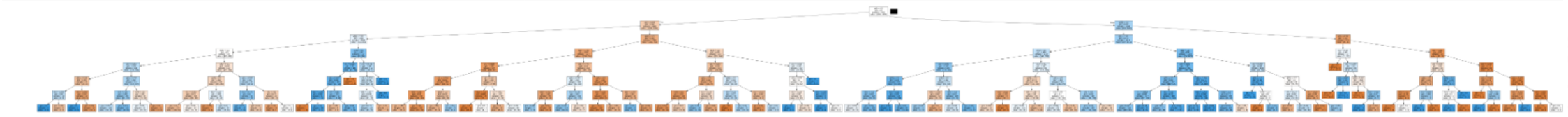
```
After pruning - Depth: 11
```

	train	test
accuracy	1.0	0.623754
precision	1.0	0.627803
recall	1.0	0.607112
f1	1.0	0.617284

Decision trees

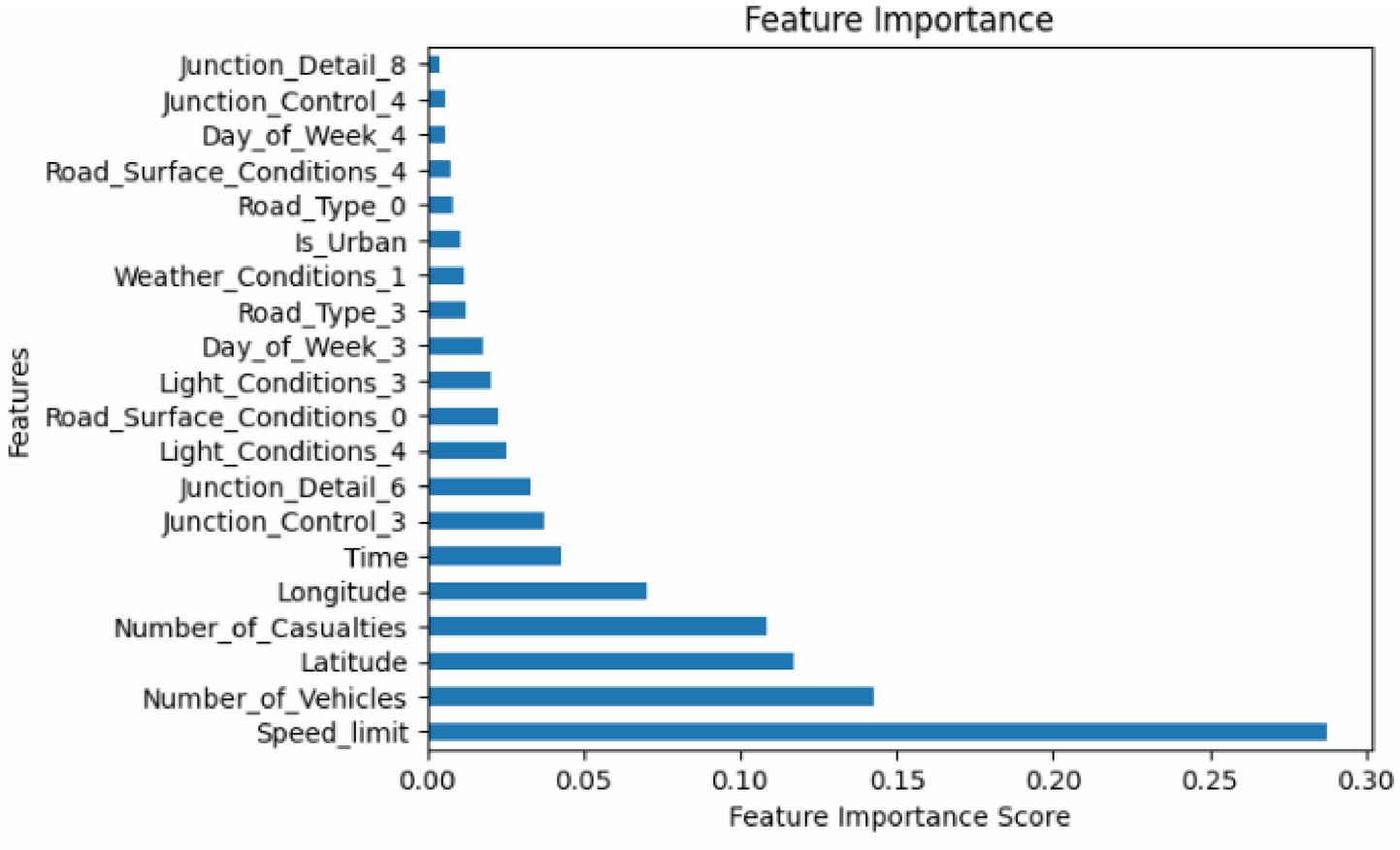
```
[ ]  from sklearn.model_selection import GridSearchCV
     param_grid = {'max_depth':range(1, dt.tree_.max_depth+1, 2),
                   'max_features': range(1, len(dt.feature_importances_)+1)}

     GR = GridSearchCV(DecisionTreeClassifier(random_state=42),
                       param_grid=param_grid,
                       scoring='accuracy',
                       n_jobs=-1)

     GR = GR.fit(x_train, y_train)
     print("node count: " ,GR.best_estimator_.tree_.node_count,"max depth: ",  GR.best_estimator_.tree_.max_depth)
```

```
node count:  209 max depth:  7
```



	train	test
accuracy	0.738524	0.683572
precision	0.716066	0.661094
recall	0.790335	0.753899
f1	0.751369	0.704453

Decision Trees- Importance Of Features



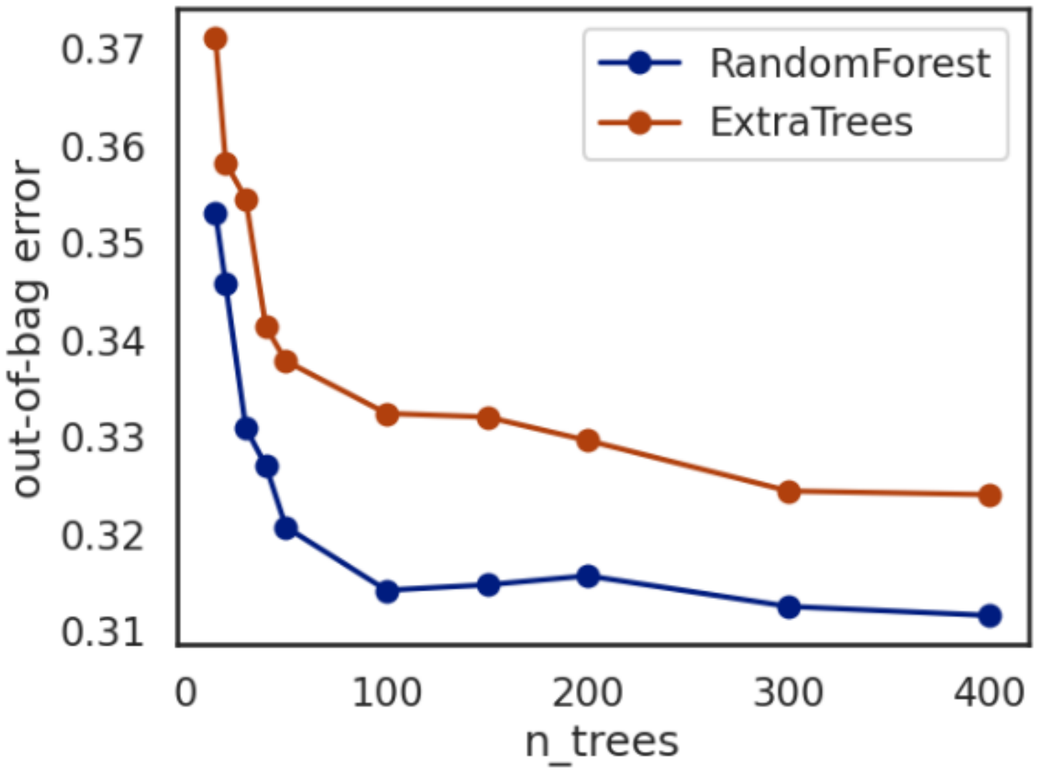
Perceptron

```
[ ]  from sklearn.linear_model import Perceptron
     from sklearn.metrics import cohen_kappa_score

     clf = Perceptron(max_iter=100)
     clf.fit(x_train, y_train)

     accuracy = clf.score(x_test , y_test)
     print( 'Accuracy is ', accuracy)
     y_pred = clf.predict(x_test)
     kappa = cohen_kappa_score(y_pred, y_test)
     print('Choen_kappa_score', kappa)
```

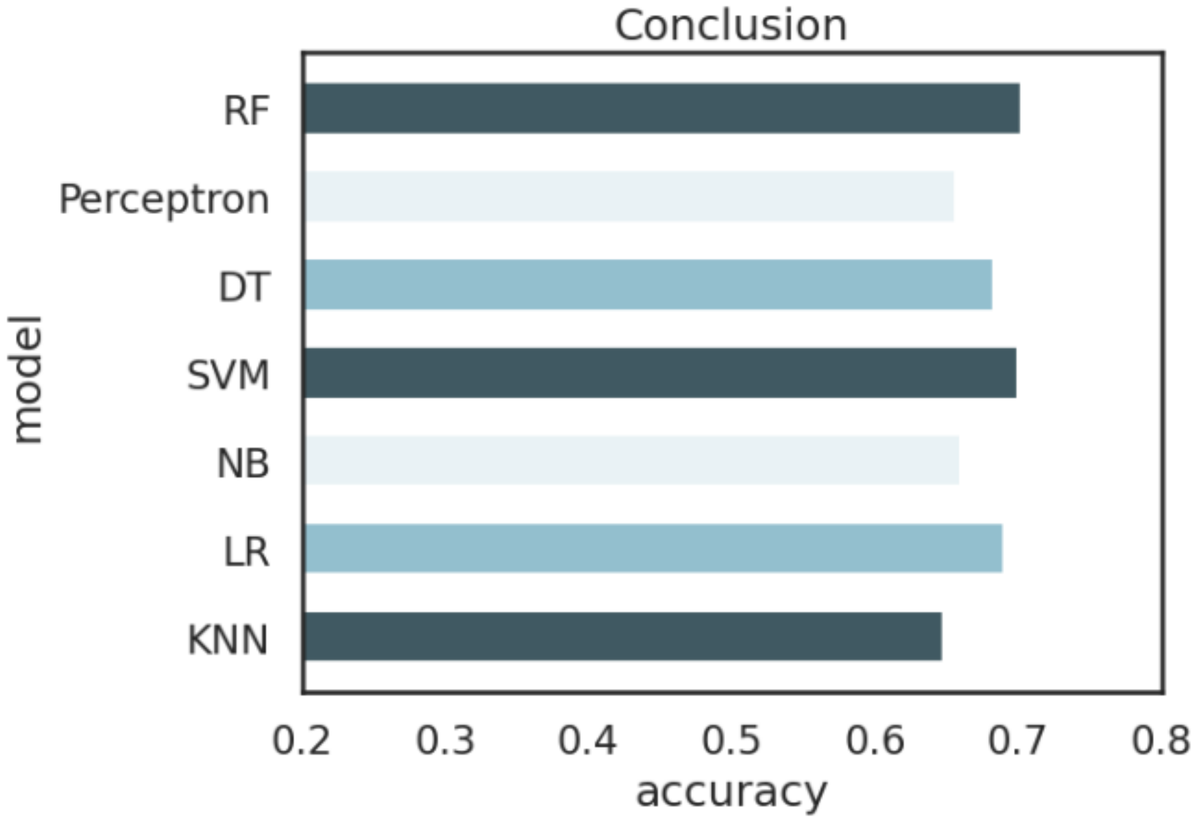
```
Accuracy is  0.6571304724750758
Choen_kappa_score 0.3142693196981613
```

Random Forest And Extra Trees

n_trees	RandomForest	ExtraTrees
15.0	0.352908	0.370935
20.0	0.345661	0.358112
30.0	0.330794	0.354395
40.0	0.326891	0.341201
50.0	0.320572	0.337670
100.0	0.314068	0.332280
150.0	0.314626	0.331909
200.0	0.315555	0.329493
300.0	0.312395	0.324289
400.0	0.311466	0.323917



Summary


Conclusion
RF
Perceptron
DT
SVM
NB
LR
KNN
model
0.2
0.3
0.4
0.5
0.6
0.7
0.8
accuracy

Bibliography

https://www.kaggle.com/datasets/nextmillionaire/car-accident-dataset