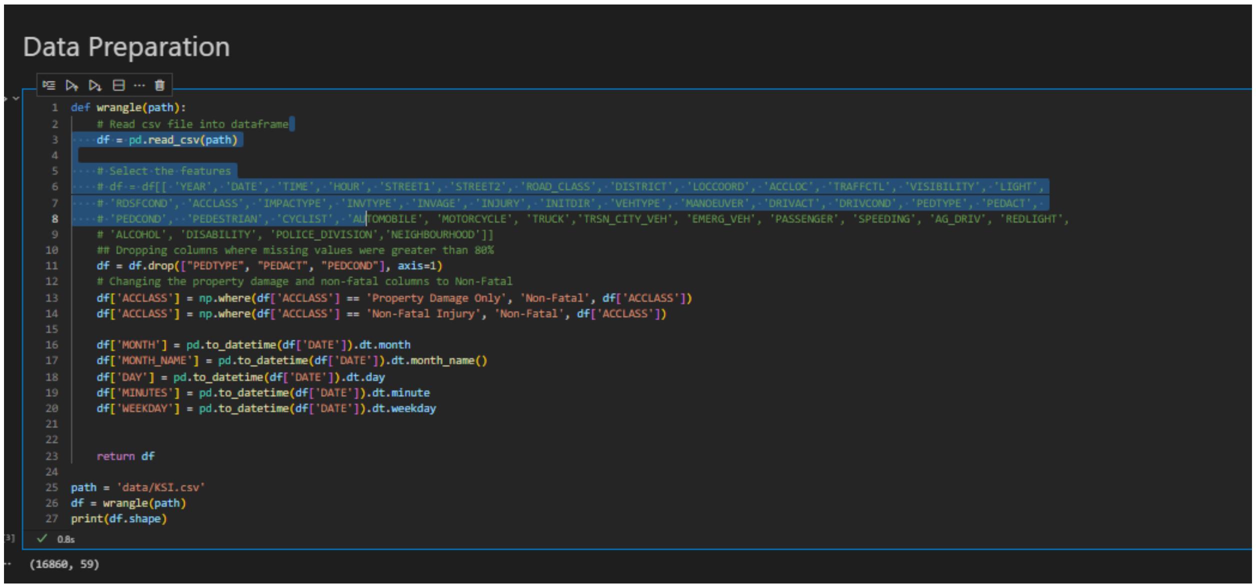
Data Analysis Procedure

The data was first prepared by transforming the target feature **ACCLASS** into two categories - **Non-Fatal** and **Fatal.** The **DATE** feature was also transformed into datetime datatype to get 5 new features namely; **MONTH, MONTH_NAME, MINUTES, DAY,** and **WEEKDAY.**



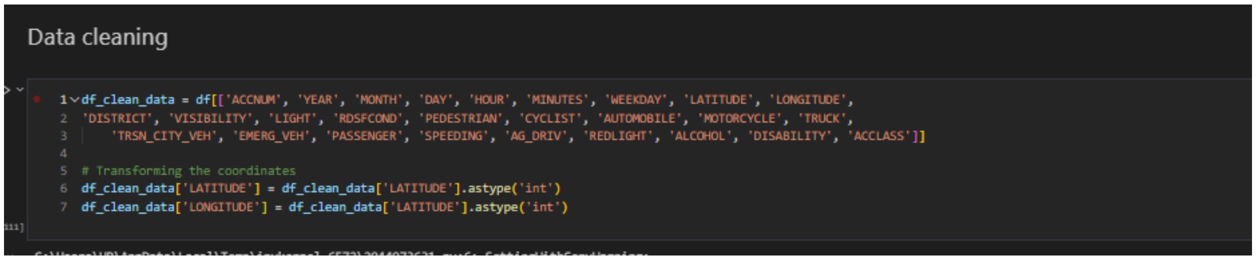
Data Preparation

```python
def wrangle(path):
    # Read csv file into dataframe
    df = pd.read_csv(path)

    # Select the features
    # df = df[[ 'YEAR', 'DATE', 'TIME', 'HOUR', 'STREET1', 'STREET2', 'ROAD_CLASS', 'DISTRICT', 'LOCCOORD', 'ACCLOC', 'TRAFFCTL', 'VISIBILITY', 'LIGHT',
    # 'RDSFCOND', 'ACCLASS', 'IMPACTYPE', 'INVTYPE', 'INVAGE', 'INJURY', 'INITDIR', 'VEHTYPE', 'MANOEUVER', 'DRIVACT', 'DRIVCOND', 'PEDTYPE', 'PEDACT',
    # 'PEDCOND',  'PEDESTRIAN', 'CYCLIST', 'AUTOMOBILE', 'MOTORCYCLE', 'TRUCK','TRSN_CITY_VEH', 'EMERG_VEH', 'PASSENGER', 'SPEEDING', 'AG_DRIV', 'REDLIGHT',
    # 'ALCOHOL', 'DISABILITY', 'POLICE_DIVISION','NEIGHBOURHOOD']]
    ## Dropping columns where missing values were greater than 80%
    df = df.drop(["PEDTYPE", "PEDACT", "PEDCOND"], axis=1)
    # Changing the property damage and non-fatal columns to Non-Fatal
    df['ACCLASS'] = np.where(df['ACCLASS'] == 'Property Damage Only', 'Non-Fatal', df['ACCLASS'])
    df['ACCLASS'] = np.where(df['ACCLASS'] == 'Non-Fatal Injury', 'Non-Fatal', df['ACCLASS'])

    df['MONTH'] = pd.to_datetime(df['DATE']).dt.month
    df['MONTH_NAME'] = pd.to_datetime(df['DATE']).dt.month_name()
    df['DAY'] = pd.to_datetime(df['DATE']).dt.day
    df['MINUTES'] = pd.to_datetime(df['DATE']).dt.minute
    df['WEEKDAY'] = pd.to_datetime(df['DATE']).dt.weekday


    return df

path = 'data/KSI.csv'
df = wrangle(path)
print(df.shape)
```

```
✓ 0.8s
(16860, 59)
```

Figure 1.1: data preparation function.

Next, the dataset was cleaned because it contained lots of low and high cardinality features.



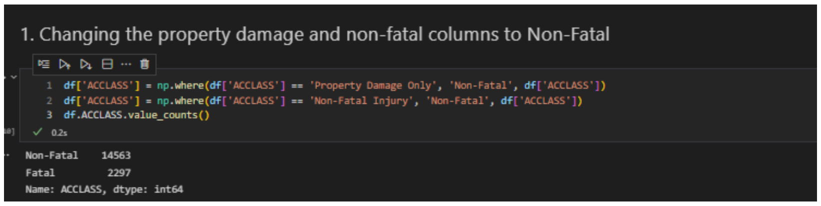
Data cleaning

```python
df_clean_data = df[['ACCNUM', 'YEAR', 'MONTH', 'DAY', 'HOUR', 'MINUTES', 'WEEKDAY', 'LATITUDE', 'LONGITUDE',
'DISTRICT', 'VISIBILITY', 'LIGHT', 'RDSFCOND', 'PEDESTRIAN', 'CYCLIST', 'AUTOMOBILE', 'MOTORCYCLE', 'TRUCK',
    'TRSN_CITY_VEH', 'EMERG_VEH', 'PASSENGER', 'SPEEDING', 'AG_DRIV', 'REDLIGHT', 'ALCOHOL', 'DISABILITY', 'ACCLASS']]

# Transforming the coordinates
df_clean_data['LATITUDE'] = df_clean_data['LATITUDE'].astype('int')
df_clean_data['LONGITUDE'] = df_clean_data['LATITUDE'].astype('int')
```

Figure 1.2: data preparation function.

Data analysis of this data was driven using the following questions:

- What story does the data tell regarding the types of collisions?
- What are the accident numbers against years and month?
- What is the fatality heatmap of those that were Fatally and Non-Fatally Injured?
- What is the rate of Fatality over years?
- What areas did accidents occur the most?
- Where are the top 10 Neighborhoods with the most collisions?
- What is the road collision distribution?

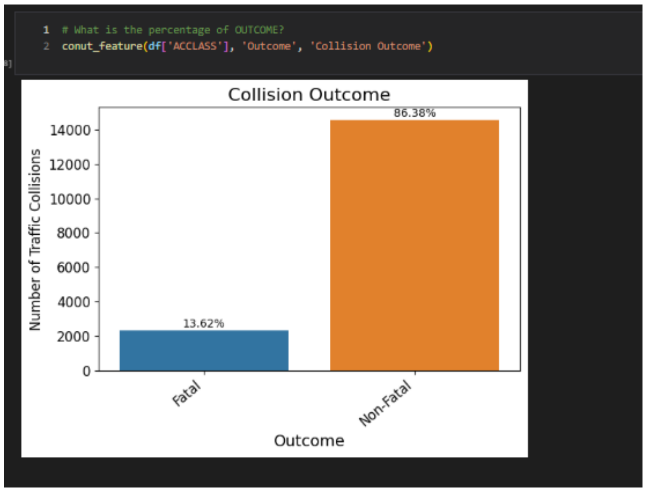


1. Changing the property damage and non-fatal columns to Non-Fatal

```
df['ACCLASS'] = np.where(df['ACCLASS'] == 'Property Damage Only', 'Non-Fatal', df['ACCLASS'])
df['ACCLASS'] = np.where(df['ACCLASS'] == 'Non-Fatal Injury', 'Non-Fatal', df['ACCLASS'])
df.ACCLASS.value_counts()
```

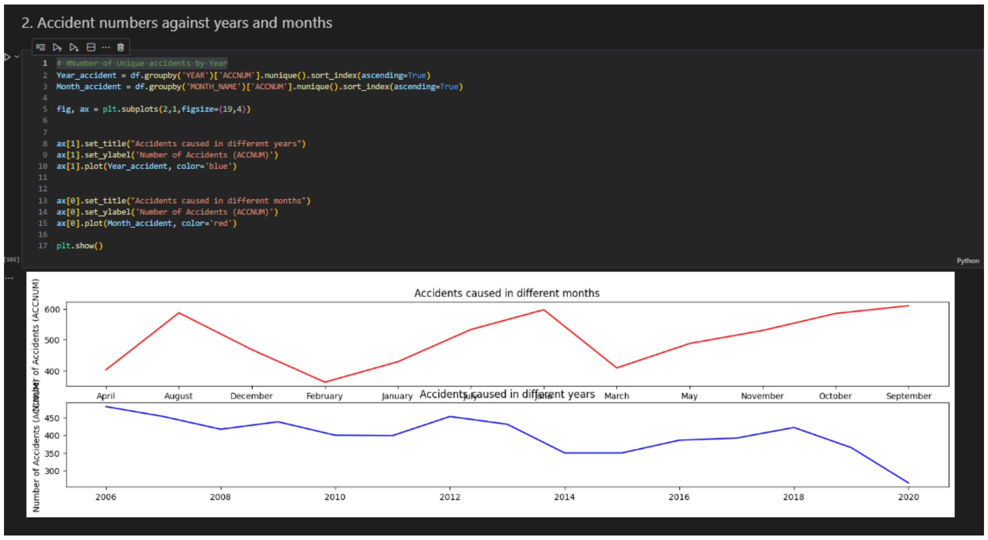
```
Non-Fatal    14563
Fatal         2297
Name: ACCLASS, dtype: int64
```

Figure 2.1: What story does the data tell regarding the types of collisions?



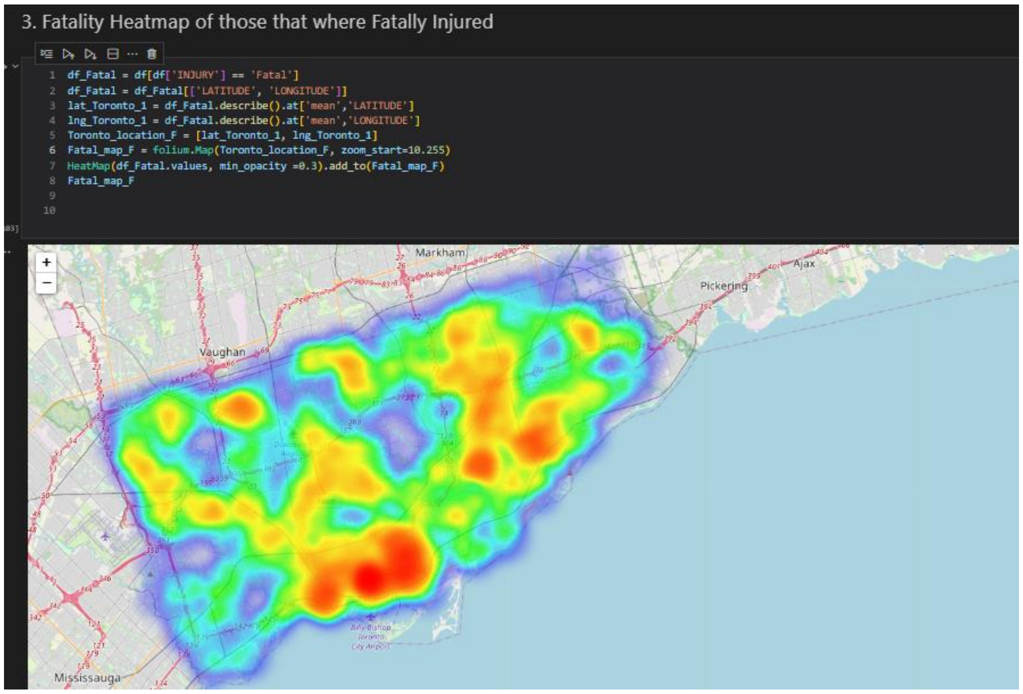
```
# What is the percentage of OUTCOME?
conut_feature(df['ACCLASS'], 'Outcome', 'Collision Outcome')
```

Figure 2.2: What story does the data tell regarding the types of collisions?


2. Accident numbers against years and months

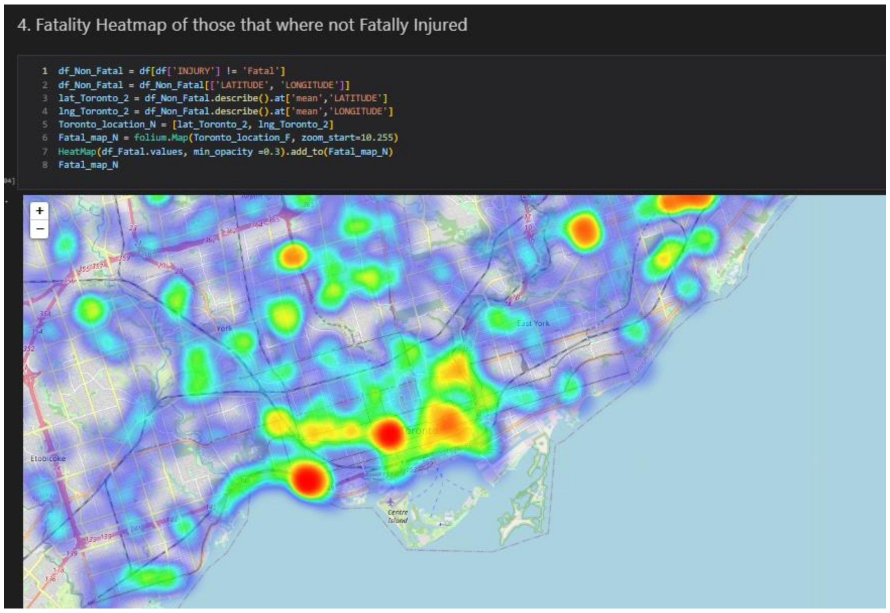
```
# #Number of Unique accidents by Year
Year_accident = df.groupby('YEAR')['ACCNUM'].nunique().sort_index(ascending=True)
Month_accident = df.groupby('MONTH_NAME')['ACCNUM'].nunique().sort_index(ascending=True)

fig, ax = plt.subplots(2,1,figsize=(19,4))


ax[1].set_title("Accidents caused in different years")
ax[1].set_ylabel('Number of Accidents (ACCNUM)')
ax[1].plot(Year_accident, color='blue')


ax[0].set_title("Accidents caused in different months")
ax[0].set_ylabel('Number of Accidents (ACCNUM)')
ax[0].plot(Month_accident, color='red')

plt.show()
```

Figure 3: Accident numbers against years and month


3. Fatality Heatmap of those that where Fatally Injured

```
df_Fatal = df[df['INJURY'] == 'Fatal']
df_Fatal = df_Fatal[['LATITUDE', 'LONGITUDE']]
lat_Toronto_1 = df_Fatal.describe().at['mean','LATITUDE']
lng_Toronto_1 = df_Fatal.describe().at['mean','LONGITUDE']
Toronto_location_F = [lat_Toronto_1, lng_Toronto_1]
Fatal_map_F = folium.Map(Toronto_location_F, zoom_start=10.255)
HeatMap(df_Fatal.values, min_opacity =0.3).add_to(Fatal_map_F)
Fatal_map_F
```

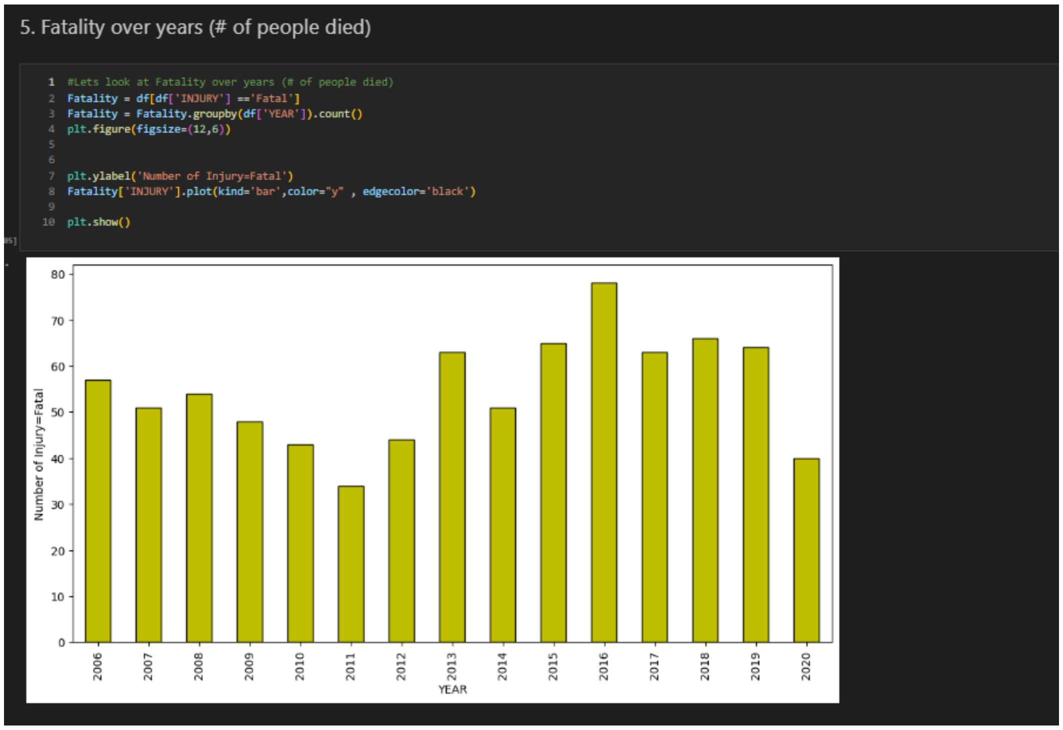
Figure 4: Fatality Heatmap of those that were Fatally Injured



4. Fatality Heatmap of those that where not Fatally Injured

```
df_Non_Fatal = df[df['INJURY'] != 'Fatal']
df_Non_Fatal = df_Non_Fatal[['LATITUDE', 'LONGITUDE']]
lat_Toronto_2 = df_Non_Fatal.describe().at['mean','LATITUDE']
lng_Toronto_2 = df_Non_Fatal.describe().at['mean','LONGITUDE']
Toronto_location_N = [lat_Toronto_2, lng_Toronto_2]
Fatal_map_N = folium.Map(Toronto_location_F, zoom_start=10.255)
HeatMap(df_Fatal.values, min_opacity =0.3).add_to(Fatal_map_N)
Fatal_map_N
```

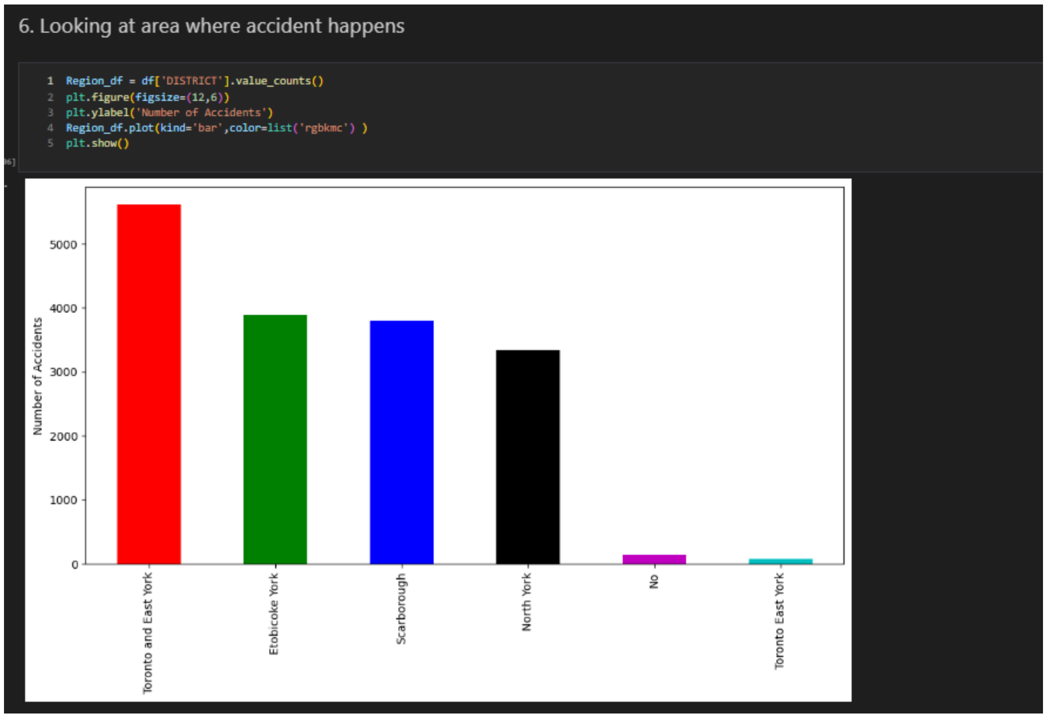
Figure 5: Fatality Heatmap of those that were Non-Fatally Injured.



5. Fatality over years (# of people died)

```
#Lets look at Fatality over years (# of people died)
Fatality = df[df['INJURY'] =='Fatal']
Fatality = Fatality.groupby(df['YEAR']).count()
plt.figure(figsize=(12,6))


plt.ylabel('Number of Injury=Fatal')
Fatality['INJURY'].plot(kind='bar',color="y" , edgecolor='black')

plt.show()
```

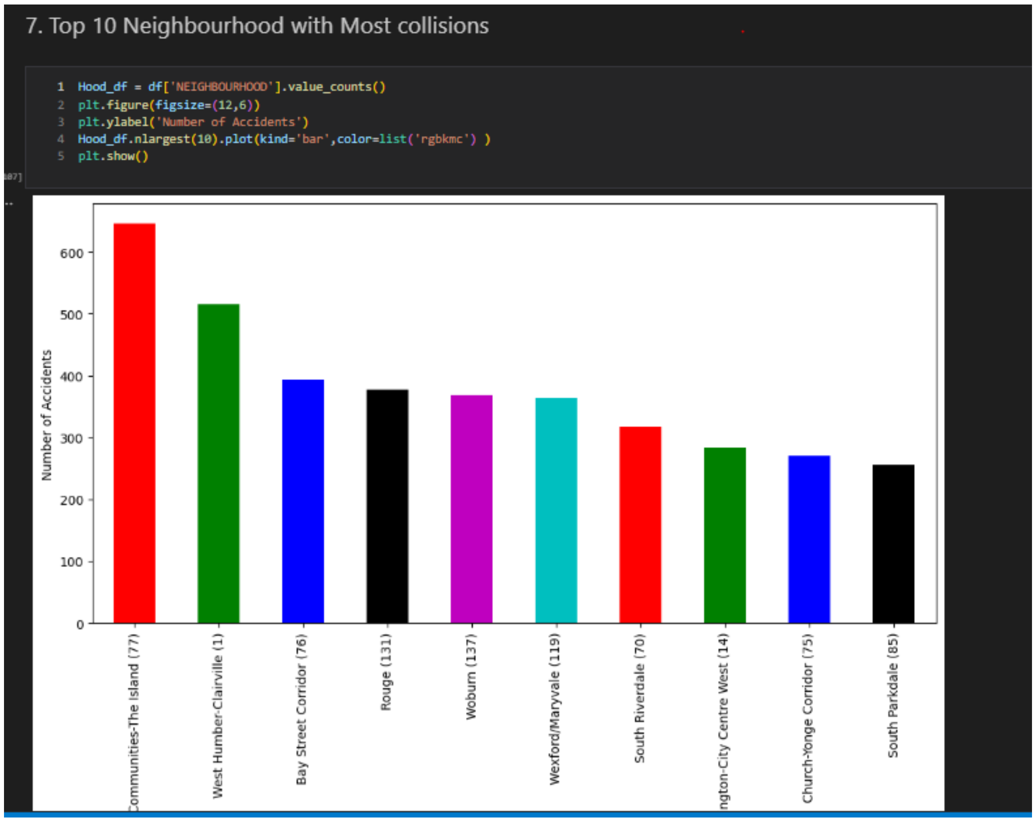
Figure 6: The rate of Fatality over years (# of people died)



6. Looking at area where accident happens

```python
Region_df = df['DISTRICT'].value_counts()
plt.figure(figsize=(12,6))
plt.ylabel('Number of Accidents')
Region_df.plot(kind='bar',color=list('rgbkmc') )
plt.show()
```

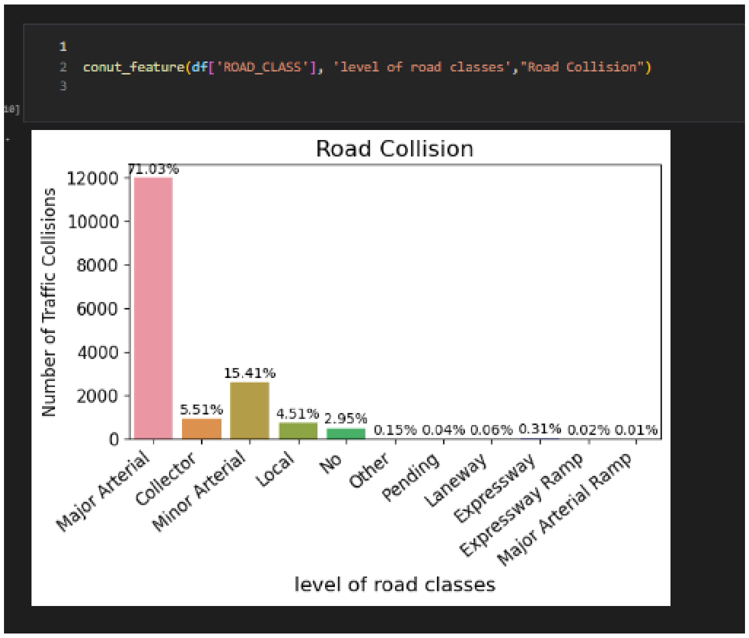
Figure 7: Areas where accidents occur the most



7. Top 10 Neighbourhood with Most collisions

```python
Hood_df = df['NEIGHBOURHOOD'].value_counts()
plt.figure(figsize=(12,6))
plt.ylabel('Number of Accidents')
Hood_df.nlargest(10).plot(kind='bar',color=list('rgbkmc') )
plt.show()
```

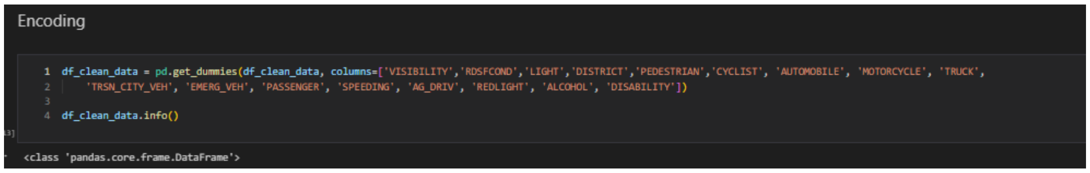
Figure 8: Areas where accidents occur the most



```
conut_feature(df['ROAD_CLASS'], 'level of road classes',"Road Collision")
```

Figure 9: Road collision distribution

The data was encoded by getting the dummies of all categorical features in the clean data, to build a scalable machine learning model. As seen below.



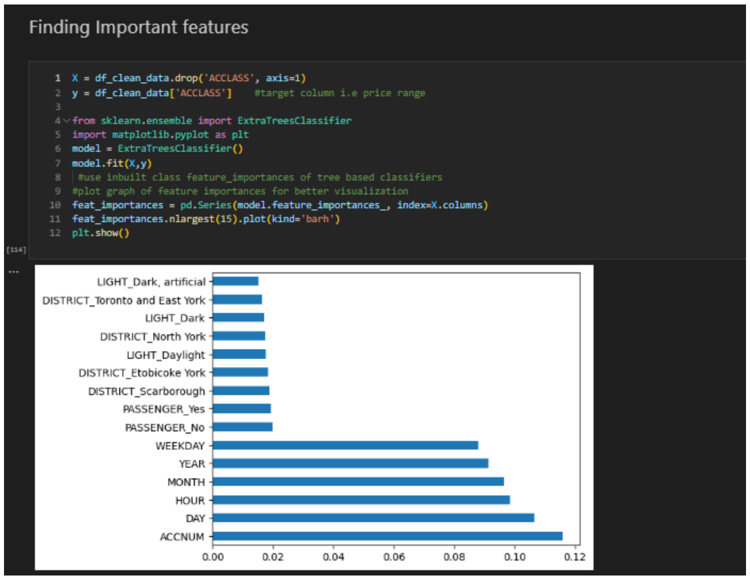
Encoding

```
df_clean_data = pd.get_dummies(df_clean_data, columns=['VISIBILITY','RDSFCOND','LIGHT','DISTRICT','PEDESTRIAN','CYCLIST', 'AUTOMOBILE', 'MOTORCYCLE', 'TRUCK',
    'TRSN_CITY_VEH', 'EMERG_VEH', 'PASSENGER', 'SPEEDING', 'AG_DRIV', 'REDLIGHT', 'ALCOHOL', 'DISABILITY'])

df_clean_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

Figure 10: Data Encoding

The most important feature was gotten by applying the *ExtraTreeClassifier* class from the *sklearn* module.

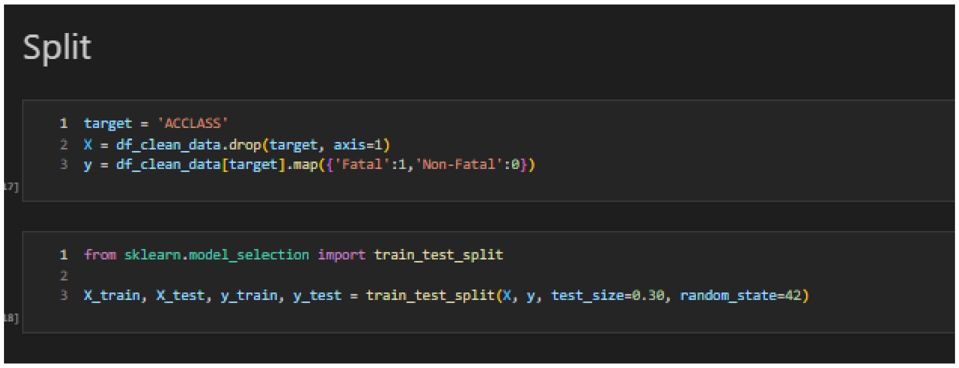


Finding Important features

```
X = df_clean_data.drop('ACCLASS', axis=1)
y = df_clean_data['ACCLASS']    #target column i.e price range

from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model = ExtraTreesClassifier()
model.fit(X,y)
 #use inbuilt class feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(15).plot(kind='barh')
plt.show()
```

Figure 11: Feature Extraction

The dataset was then split into a training and a test set using a 70:30 ratio.



Split

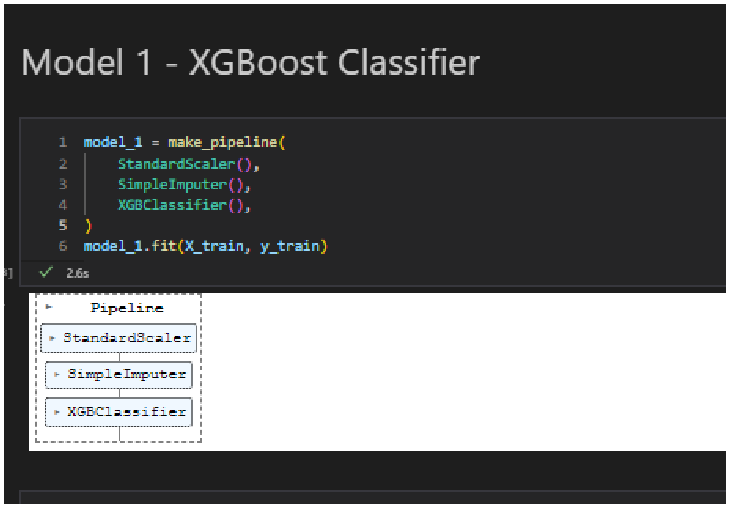
```
target = 'ACCLASS'
X = df_clean_data.drop(target, axis=1)
y = df_clean_data[target].map({'Fatal':1,'Non-Fatal':0})
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

Figure 12: Road collision distribution

Afterwards, the training set was used to train each model and the test set was used to evaluate the performance of each model. The training and test data were used on two different machine learning model which are the 1. Xtreeme boosting classifier and the Support Vector Machine Classifier.

1. Xtreeme boosting classifier



Model 1 - XGBoost Classifier

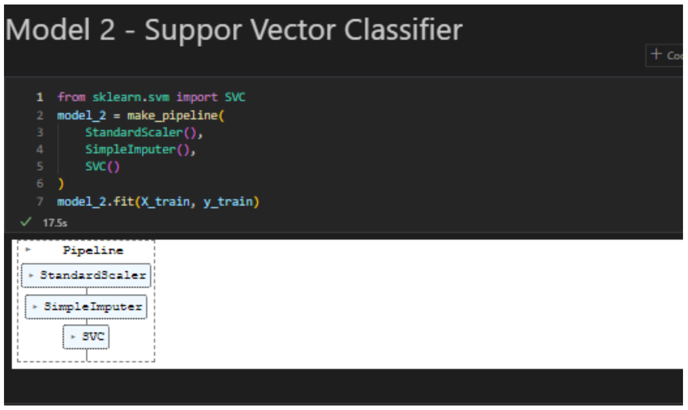
```
model_1 = make_pipeline(
    StandardScaler(),
    SimpleImputer(),
    XGBClassifier(),
)
model_1.fit(X_train, y_train)
```

✓ 2.6s

```
Pipeline
StandardScaler
SimpleImputer
XGBClassifier
```

Figure 13: XGBoost Classifier

2. Support Vector Machine Classifier



Model 2 - Suppor Vector Classifier

```
from sklearn.svm import SVC
model_2 = make_pipeline(
    StandardScaler(),
    SimpleImputer(),
    SVC()
)
model_2.fit(X_train, y_train)
```

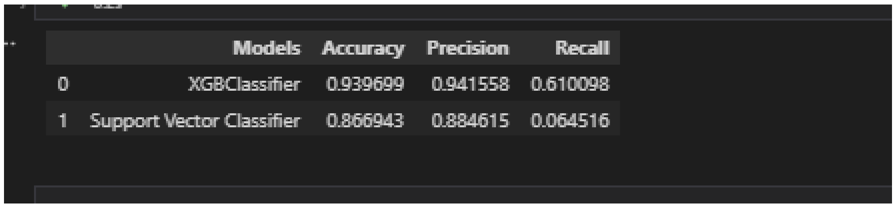
✓ 17.5s

```
Pipeline
StandardScaler
SimpleImputer
SVC
```

Figure 14: SVM Classifier

Both models were evaluated based on the following metrics.

- Accuracy
- Precision
- Recall



	Models	Accuracy	Precision	Recall
0	XGBClassifier	0.939699	0.941558	0.610098
1	Support Vector Classifier	0.866943	0.884615	0.064516

Figure 15: Model Evaluations

Conclusion

The test performance for the XGBoost Classifier model was better than the SVM Classifier model with a difference of **0.057144.** The **XGboost** took less time to run compared to the **SVM** Classifier. The Xgboost model generated robust and more significant results than the SVM Classifier as shown in ***figure 15,*** that's why the XGBoost model is the best to use for this case study.