GROUP PROJECT

Welcome to the COMP 247 Project - Supervising Learning Project - KSI Collisions Toronto

Relevant Information:

- College: Centennial College
- Program: Software Engineering Technology - Artificial Intelligence
- Term: Summer 2022
- Course: 22M --Supervised Learning (SEC. 001) - COMP247001_2022MW

Group Members

- ., Ripudaman
- Maria, Karan
- Radmy, Mahpara Rafia
- Saenz Huerta, Ronald
- Sidhu, Manipal

COMP 247 Project

Group Project – Developing a predictive machine learning model (classifier) and deploy it as a web API for inference

Dataset

https://data.torontopolice.on.ca/datasets/TorontoPS::ksi/about (https://data.torontopolice.on.ca/datasets/TorontoPS::ksi/about)

Models:

- Logistic Regression
- Random Forest Classifier
- Decision Tree Classifier
- KNeighbors Classifier
- SVC

In [1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import *
from sklearn.metrics import confusion_matrix, accuracy_score, classificati

from sklearn.svm import SVC
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from scipy.stats import randint

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

In [3]:

```
def analyze_data(data):

    print("\n","Data - 20 first rows")
    print(data.head(20))

    print("\n","Data info")
    print(data.info())

    print("\n","Data Shape")
    print(data.shape)

    print("\n","Data - Null Values")
    print(data.isnull().sum())

    print("\n","Data - Null Values String")
    for column in data.columns:
        print("Column:", column, " - Len:", len(data[data[column] == '<Nul

    print("\n","Data - Describe stats")
    stats = data.describe()
    print(data.describe())

    print("\n","Data - Plot histograms")
    hist = data.hist(bins=3, figsize=(9,10))

    print("\n","Data - Plot scatter matrix")
    pd.plotting.scatter_matrix(data, alpha=0.40, figsize=(13,8))
```

In [18]:

```
def analyze_data_unique_values(data, value_counts):
```

```
    print("\n","Data - Unique Values")
    for column in data:
        print("\n", "Column:", column, " - Len:", len(data[column].unique(
        if value_counts:
            print(data[column].value_counts())
```

In [19]:

```
def cleaning_data_initial(data):

    # Replace <null> with nan .
    data = data.replace('<Null>', np.nan)

    # Extract month from date and remove date column
    data['month'] = pd.DatetimeIndex(data['DATE']).month


    #BINARY COLUMNS 0: NULL 1: YES
    binary_columns=['CYCLIST','AUTOMOBILE','MOTORCYCLE','TRUCK','TRSN_CITY_

    for i in binary_columns:
        data[i].replace(np.nan, 0, inplace=True)
        data[i].replace("Yes", 1, inplace=True)
        data[i] = data[i].astype(int)

    # Drop columns that are not required

    # A lot of different values
    drop_columns=['INDEX_','ObjectId','ACCNUM', 'X', 'Y','STREET1', 'STREET
    # Duplicated with HOOD_ID and POLICE_DIVISION
    drop_columns+=['NEIGHBOURHOOD', 'DIVISION']
    # A lot of null values
    drop_columns+=["OFFSET", "PEDTYPE", "PEDACT", "PEDCOND", "CYCLISTYPE",

    drop_columns+=['TIME', 'YEAR', 'DATE', 'WARDNUM', 'INITDIR', 'INVAGE']

    # DROP
    data = data.drop(drop_columns, axis=1)

    # Drop columns with null count greater than 40 % .
    data = data.dropna(axis=1,thresh=(data.shape[0]*0.6))

    # ----------------------------------------------------------------------
    # Drop duplicates values - rows
    data = data.drop_duplicates()
    data.nunique(axis=0)

    return data
```

In [68]:

```
def cleaning_data_values(data):


    # ----------------------------------------------------------------------
    # Column: ROAD_CLASS
```

```
# Replace the Ramp type with an other existing category
data['ROAD_CLASS'].replace('Expressway Ramp', 'Expressway', inplace=Tr
data['ROAD_CLASS'].replace('Major Arterial Ramp', 'Major Arterial', in

# Replace all null values with Other category
data['ROAD_CLASS'].replace(np.nan, 'Other', inplace=True)


# ----------------------------------------------------------------------
# Column: DISTRICT

# Replace in one category Toronto and East York
data['DISTRICT'].replace('Toronto East York', 'Toronto and East York',

# Replace all null values with Other category
data['DISTRICT'].replace(np.nan, 'Other', inplace=True)


# ----------------------------------------------------------------------
# Column: LOCCOORD

# Replace in existing categories
data['LOCCOORD'].replace('Mid-Block (Abnormal)', 'Mid-Block', inplace=
data['LOCCOORD'].replace('Entrance Ramp Westbound', 'Exit Ramp',inplac
data['LOCCOORD'].replace('Exit Ramp Westbound', 'Exit Ramp',inplace=Tr
data['LOCCOORD'].replace('Exit Ramp Southbound','Exit Ramp',inplace=Tr
data['LOCCOORD'].replace('Park, Private Property, Public Lane', 'Other

# Replace all null values with Other category
data['LOCCOORD'].replace(np.nan, 'Other', inplace=True)


# ----------------------------------------------------------------------
# Column: ACCLOC

# Replace in existing categories
data['ACCLOC'].replace('Intersection Related', 'At Intersection', inpl
data['ACCLOC'].replace('Private Driveway', 'At/Near Private Drive', in

# Replace small values in Other category
data['ACCLOC'].replace('Laneway', 'Other', inplace=True)
data['ACCLOC'].replace('Overpass or Bridge', 'Other', inplace=True)
data['ACCLOC'].replace('Underpass or Tunnel', 'Other', inplace=True)
data['ACCLOC'].replace('Trail', 'Other', inplace=True)

# Replace all null values with Other category
data['ACCLOC'].replace(np.nan, 'Other', inplace=True)


# ----------------------------------------------------------------------
# Column: TRAFFCTL

# Replace small values in Other category
data['TRAFFCTL'].replace('Traffic Controller', 'Other', inplace=True)
data['TRAFFCTL'].replace('Yield Sign', 'Other', inplace=True)
```

```
data['TRAFFCTL'].replace('Streetcar (Stop for)', 'Other', inplace=True
data['TRAFFCTL'].replace('Traffic Gate', 'Other', inplace=True)
data['TRAFFCTL'].replace('School Guard', 'Other', inplace=True)
data['TRAFFCTL'].replace('Police Control', 'Other', inplace=True)

# Replace all null values with Other category
data['TRAFFCTL'].replace(np.nan, 'Other', inplace=True)


# ----------------------------------------------------------------------
# Column: VISIBILITY

# Replace in existing categories
data['VISIBILITY'].replace('Drifting Snow', 'Snow', inplace=True)
data['VISIBILITY'].replace('Freezing Rain', 'Rain',inplace=True)

# Replace small values in Other category
data['VISIBILITY'].replace('Strong wind', 'Other',inplace=True)

# Replace all null values with Other category
data['VISIBILITY'].replace(np.nan, 'Other', inplace=True)


# ----------------------------------------------------------------------
# Column: LIGHT

# Replace in existing categories
data['LIGHT'].replace('Dark, artificial', 'Dark',inplace=True)
data['LIGHT'].replace('Dusk, artificial', 'Dusk',inplace=True)
data['LIGHT'].replace('Daylight, artificial', 'Daylight',inplace=True)
data['LIGHT'].replace('Dawn, artificial', 'Dawn',inplace=True)

# Replace all null values with Other category
data['LIGHT'].replace(np.nan, 'Other', inplace=True)


# ----------------------------------------------------------------------
# Column: RDSFCOND

# Replace in existing categories
data['RDSFCOND'].replace('Loose Snow', 'Snow',inplace=True)
data['RDSFCOND'].replace('Packed Snow', 'Snow',inplace=True)

# Replace small values in Other category
data['RDSFCOND'].replace('Loose Sand or Gravel', 'Other',inplace=True)
data['RDSFCOND'].replace('Spilled liquid', 'Other',inplace=True)

# Replace all null values with Other category
data['RDSFCOND'].replace(np.nan, 'Other', inplace=True)


# ----------------------------------------------------------------------
# Column: ACCLASS

# Replace in existing categories - Property Damage Only is Non-FAtal I
data['ACCLASS'].replace('Property Damage Only', 'Non-Fatal Injury',inp
```

```
# Replace all null values with Non-Fatal Injury category
data['ACCLASS'].replace(np.nan, 'Non-Fatal Injury', inplace=True)

# Replace values with binary classification
data['ACCLASS'].replace('Non-Fatal Injury', 0, inplace=True)
data['ACCLASS'].replace('Fatal', 1, inplace=True)


# ----------------------------------------------------------------------
# Column: IMPACTYPE

# Replace all null values with Non-Fatal Injury category
data['IMPACTYPE'].replace(np.nan, 'Other', inplace=True)


# ----------------------------------------------------------------------
# Column: INVTYPE

# Replace in existing categories
data['INVTYPE'].replace('Witness', 'Pedestrian', inplace=True)
data['INVTYPE'].replace('Pedestrian - Not Hit', 'Pedestrian', inplace=
data['INVTYPE'].replace('Driver - Not Hit', 'Driver', inplace=True)
data['INVTYPE'].replace('Cyclist', 'Cyclist Passenger', inplace=True)
data['INVTYPE'].replace('Cyclist Passenger', 'Cyclist Passenger', inpl
data['INVTYPE'].replace('Motorcycle Driver', 'Motorcycle Passenger', i
data['INVTYPE'].replace('Motorcycle Passenger', 'Motorcycle Passenger'
data['INVTYPE'].replace('Trailer Owner', 'Truck Driver', inplace=True)

# Replace small values in Other category
data['INVTYPE'].replace('Other Property Owner', 'Other',inplace=True)
data['INVTYPE'].replace('Moped Driver', 'Other',inplace=True)
data['INVTYPE'].replace('Wheelchair', 'Other',inplace=True)
data['INVTYPE'].replace('In-Line Skater', 'Other',inplace=True)

# Replace all null values with Other category
data['INVTYPE'].replace(np.nan, 'Other', inplace=True)


# ----------------------------------------------------------------------
# Column: INJURY

# Replace in existing categories
data['INJURY'].replace('Minimal', 'Minor', inplace=True)

# Replace all null values with Non-Fatal Injury category
data['INJURY'].replace(np.nan, 'None', inplace=True)


# ----------------------------------------------------------------------
# Column: VEHTYPE

# Replace all types of truck in only group called 'Truck'
data['VEHTYPE'].replace('Truck - Open', 'Truck', inplace=True)
data['VEHTYPE'].replace('Truck-Tractor', 'Truck', inplace=True)
data['VEHTYPE'].replace('Truck - Closed (Blazer, etc)', 'Truck', inpla
```

```
    data['VEHTYPE'].replace('Truck - Dump', 'Truck', inplace=True)
    data['VEHTYPE'].replace('Truck (other)', 'Truck', inplace=True)
    data['VEHTYPE'].replace('Truck - Tank', 'Truck', inplace=True)
    data['VEHTYPE'].replace('Pick Up Truck', 'Truck', inplace=True)
    data['VEHTYPE'].replace('Tow Truck', 'Truck', inplace=True)
    data['VEHTYPE'].replace('Truck - Car Carrier', 'Truck', inplace=True)

    # Replace all types of 2 wheels in only group called 2 Wheels
    data['VEHTYPE'].replace('Motorcycle', '2 Wheels', inplace=True)
    data['VEHTYPE'].replace('Bicycle', '2 Wheels', inplace=True)
    data['VEHTYPE'].replace('Moped', '2 Wheels', inplace=True)
    data['VEHTYPE'].replace('Off Road - 2 Wheels', '2 Wheels', inplace=Tru

    # Replace all types of automobiles in only group called Automobile
    data['VEHTYPE'].replace('Automobile, Station Wagon', 'Automobile', inp
    data['VEHTYPE'].replace('Taxi', 'Automobile', inplace=True)

    # Replace all types of Emergency Vehicles in only group called Emergen
    data['VEHTYPE'].replace('Police Vehicle', 'Emergency', inplace=True)
    data['VEHTYPE'].replace('Other Emergency Vehicle', 'Emergency', inplac
    data['VEHTYPE'].replace('Fire Vehicle', 'Emergency', inplace=True)

    # Replace all types of Buses in only group called Bus
    data['VEHTYPE'].replace('Municipal Transit Bus (TTC)', 'Bus', inplace=
    data['VEHTYPE'].replace('Street Cars', 'Bus', inplace=True)
    data['VEHTYPE'].replace('Street Car', 'Bus', inplace=True)
    data['VEHTYPE'].replace('Bus (Other) (Go Bus, Gray Coach)', 'Bus', inp
    data['VEHTYPE'].replace('Intercity Bus', 'Bus', inplace=True)
    data['VEHTYPE'].replace('School Bus', 'Bus', inplace=True)

    # Replace all types of Vans in only group called Van
    data['VEHTYPE'].replace('Passenger Van', 'Van', inplace=True)
    data['VEHTYPE'].replace('Delivery Van', 'Van', inplace=True)

    # Replace small values in Other category
    data['VEHTYPE'].replace('Construction Equipment', 'Other',inplace=True

    # Replace all null values with Other category
    data['VEHTYPE'].replace(np.nan, 'Other', inplace=True)


    # ------------------------------------------------------------------
    # Column: POLICE_DIVISION

    #Police Division without 'D' character
    data['POLICE_DIVISION'] = data['POLICE_DIVISION'].str.strip("D")
    data['POLICE_DIVISION'] = data['POLICE_DIVISION'].astype(int)


    # ------------------------------------------------------------------
    # Drop duplicates values - rows
    data = data.drop_duplicates()
    data.nunique(axis=0)
```

Data Preprocessing - Get pipeline transformer, and X, Y train ant test data.

```
    return data
```

In [60]:

```
def get_pipeline_x_y(data=None, test_size=0.20):

    features_columns_categorical = ["ROAD_CLASS", "DISTRICT", "LOCCOORD",
    features_columns_numbers = ['HOUR', 'CYCLIST','AUTOMOBILE','MOTORCYCLE


    #    ('imputer', SimpleImputer(missing_values=np.nan, strategy='constar
    #('imputer', KNNImputer(n_neighbors=2)),


    categorical_pipeline = Pipeline(steps=[
        ('imputer', SimpleImputer(missing_values=np.nan, strategy='constan
        ('encoder', OneHotEncoder(handle_unknown='ignore'))
    ])

    num_pipeline = Pipeline([
        ('imputer', SimpleImputer(missing_values=np.nan, strategy="median"
        ('std_scaler', StandardScaler()),
    ])

    # Full pipeline - Column Transformer
    full_pipeline_transformer = ColumnTransformer([
        ("num", num_pipeline, features_columns_numbers),
        ("cat", categorical_pipeline, features_columns_categorical),
    ])

    X_group = data[features_columns_categorical + features_columns_numbers
    Y_group = data['ACCLASS']


    np.random.seed(2)

    # Divide data in train/test
    X_train, X_test, y_train, y_test = train_test_split(X_group, Y_group,


    return full_pipeline_transformer, X_group, Y_group, X_train, X_test, y
```

Get Best Model - Logistic Regression, Decision Tree, Random

In [86]:

```
def get_best_model(data, classifier_model, full_pipeline_transformer, X_tr


    # Initialze the estimators
    estimator_1 = LogisticRegression(random_state=42)
    estimator_2 = DecisionTreeClassifier(random_state=42)
    estimator_3 = RandomForestClassifier(random_state=42)
    estimator_4 = SVC(probability=True, random_state=42)
    estimator_5 = MultinomialNB()
    estimator_6 = KNeighborsClassifier()
```

```
# Initiaze the hyperparameters for each dictionary
if classifier_model == "LogisticRegression":

    param_grid = {
        'classifier__solver': ['lbfgs', 'saga'],
        'classifier__max_iter': [100, 1000],
        'classifier__random_state': [0, 42],
        'classifier__multi_class': ['auto', 'multinomial']
    }

    full_pipeline = Pipeline([
        ('preprocessing', full_pipeline_transformer),
        ('classifier', estimator_1),
    ])


elif classifier_model == "DecisionTreeClassifier":

    #'classifier__min_samples_split': [2, 5, 10, 20],
    #'classifier__min_samples_leaf': [1, 5, 10],
    #'classifier__max_leaf_nodes': [None, 5, 10, 20],

    param_grid = {
        'classifier__criterion': ['gini', 'entropy'],
        'classifier__max_depth': [2 ,5, 10, 25, None],
        'classifier__min_samples_split': [2],
        'classifier__min_samples_leaf': [1],
        'classifier__max_leaf_nodes': [20],
        'classifier__class_weight': [None, {0:1,1:5}, {0:1,1:10}, {0:1
    }

    full_pipeline = Pipeline([
        ('preprocessing', full_pipeline_transformer),
        ('classifier', estimator_2),
    ])

elif classifier_model == "RandomForestClassifier":

    param_grid = {
        'classifier__n_estimators': [10, 50, 100, 250],
        'classifier__max_depth': [5,10,20],
        'classifier__class_weight': [None, {0:1,1:5}, {0:1,1:10}, {0:1
    }

    full_pipeline = Pipeline([
        ('preprocessing', full_pipeline_transformer),
        ('classifier', estimator_3),
    ])

elif classifier_model == "SVC":

    param_grid = {
        'classifier__kernel': ['linear', 'rbf','poly'],
        #'classifier__C': [0.01, 0.1, 1, 10, 100],
        #'classifier__class_weight': [None, {0:1,1:5}, {0:1,1:10}, {0:
        'classifier__gamma': ['auto'],
```

```
        }

        full_pipeline = Pipeline([
            ('preprocessing', full_pipeline_transformer),
            ('classifier', estimator_4),
        ])

    elif classifier_model == "MultinomialNB":

        param_grid = {
            'classifier__alpha': [0, 10, 100]
        }

        full_pipeline = Pipeline([
            ('preprocessing', full_pipeline_transformer),
            ('classifier', estimator_5),
        ])

    elif classifier_model == "KNeighborsClassifier":

        param_grid = {
            'classifier__n_neighbors': [2,5,10,25,50]
        }

        full_pipeline = Pipeline([
            ('preprocessing', full_pipeline_transformer),
            ('classifier', estimator_6),
        ])

    else:
        param_grid = [
            {
                'classifier': [estimator_1],
                'classifier__solver': ['lbfgs', 'saga'],
                'classifier__max_iter': [1000],
                'classifier__random_state': [0, 42],
                'classifier__multi_class': ['auto', 'multinomial']
            },
            {
                'classifier': [estimator_2],
                'classifier__criterion': ['gini', 'entropy'],
                'classifier__max_depth': [2 ,5, 10, 25, None],
                'classifier__min_samples_split': [2],
                'classifier__min_samples_leaf': [1],
                'classifier__max_leaf_nodes': [20],
                'classifier__class_weight': [None, {0:1,1:5}, {0:1,1:10},
            },
            {
                'classifier': [estimator_3],
                'classifier__n_estimators': [10, 50, 100, 250],
                'classifier__max_depth': [5,10,20],
                'classifier__class_weight': [None, {0:1,1:5}, {0:1,1:10},
            },
            {
                'classifier': [estimator_4],
                'classifier__kernel': ['linear', 'rbf','poly'],
```

```
                #'classifier__C': [0.01, 0.1, 1, 10, 100],
                #'classifier__class_weight': [None, {0:1,1:5}, {0:1,1:10},
                'classifier__gamma': ['auto'],
            },
            {
                'classifier': [estimator_5],
                'classifier__alpha': [0, 10, 100]
            },
            {
                'classifier': [estimator_6],
                'classifier__n_neighbors': [2,5,10,25,50]
            }
        ]

        full_pipeline = Pipeline([
            ('preprocessing', full_pipeline_transformer),
            ('classifier', estimator_1),
        ])

    print("************************************************************")
    print("Get Best Estimator/Params of the Model for ", classifier_model)


    gs = GridSearchCV(full_pipeline, param_grid, cv=3, n_jobs=-1, scoring=
    gs.fit(X_train, y_train)


    print("Best Estimator:", gs.best_estimator_)
    print("Best Params:", gs.best_params_)
    print("Best Score:", gs.best_score_)

    # Test data performance
    print("Test Precision:",precision_score(gs.predict(X_test), y_test))
    print("Test Recall:",recall_score(gs.predict(X_test), y_test))
    print("Test ROC AUC Score:",roc_auc_score(gs.predict(X_test), y_test))

    print("Test Accuracy Score = ", accuracy_score(gs.predict(X_test), y_t
    print("Test Confusion Matrix = \n", confusion_matrix(gs.predict(X_test
    print("Test Classification Report = \n", classification_report(gs.pred

    # CONFUSION MATRIX PLOT
    cm = confusion_matrix(gs.predict(X_test), y_test)
    ax = plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells

    # ROC AUC CURVE PLOT
    plot_roc_curve(gs, X_test, y_test)
    plt.show()

    return gs
```

In [ ]:

In [62]:
```
data = pd.read_csv('KSI.csv')
```

Out[62]:

	X	Y	INDEX_	ACCNUM	YEAR	DATE	TIME	HOUR
0	-8.844611e+06	5.412414e+06	3387730	892658	2006	2006/03/11 05:00:00+00	852	8
1	-8.844611e+06	5.412414e+06	3387731	892658	2006	2006/03/11 05:00:00+00	852	8
2	-8.816480e+06	5.434843e+06	3388101	892810	2006	2006/03/11 05:00:00+00	915	9
3	-8.816480e+06	5.434843e+06	3388102	892810	2006	2006/03/11 05:00:00+00	915	9
4	-8.822759e+06	5.424516e+06	3387793	892682	2006	2006/03/12 05:00:00+00	240	2
...	...	...	...	...	...	...	...	...
16855	-8.820837e+06	5.421411e+06	81509784	1636966	2020	2020/08/30 04:00:00+00	1340	13
16856	-8.820068e+06	5.425334e+06	81505452	1650701	2020	2020/09/01 04:00:00+00	1205	12
16857	-8.820068e+06	5.425334e+06	81505453	1650701	2020	2020/09/01 04:00:00+00	1205	12
16858	-8.820068e+06	5.425334e+06	81505454	1650701	2020	2020/09/01 04:00:00+00	1205	12
16859	-8.842562e+06	5.412998e+06	81509748	1650875	2020	2020/09/01 04:00:00+00	1238	12

16860 rows × 57 columns

In [63]:
```
# Analyze data - first step
analyze_data(data)
analyze_data_unique_values(data, False)
```

```
Data - 20 first rows
              X             Y    INDEX    ACCNUM  YEAR
```

Cleaning Data

In [64]:
```
# preprocessing - clean data
```

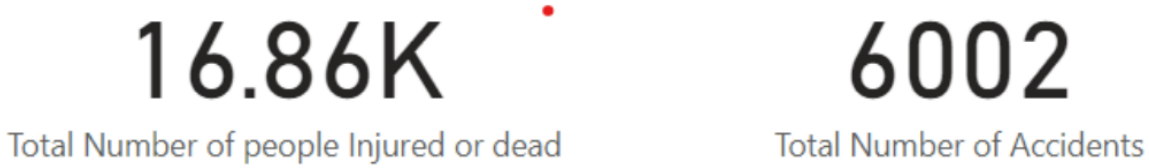
Analyze Data - Data Exploration Stats, Histogram, Graphs

In [65]:
```
# Analyze data - after the first cleaning
analyze_data(data)
analyze_data_unique_values(data, True)
```

```
 Data - 20 first rows
    HOUR      ROAD_CLASS               DISTRICT      LOCCOORD  \
0      8  Major Arterial  Toronto and East York  Intersection
1      8  Major Arterial  Toronto and East York  Intersection
2      9  Major Arterial            Scarborough  Intersection
3      9  Major Arterial            Scarborough  Intersection
4      2  Major Arterial            Scarborough     Mid-Block
5      2  Major Arterial            Scarborough     Mid-Block
6      2  Major Arterial            Scarborough     Mid-Block
7     19  Major Arterial  Toronto and East York  Intersection
8     19  Major Arterial  Toronto and East York  Intersection
9     15  Major Arterial         Etobicoke York  Intersection
10    15  Major Arterial         Etobicoke York  Intersection
11     9  Major Arterial            Scarborough  Intersection
12     9  Major Arterial            Scarborough  Intersection
13     9  Major Arterial            Scarborough  Intersection
14     9  Major Arterial            Scarborough  Intersection
15     9  Major Arterial            Scarborough  Intersection
```

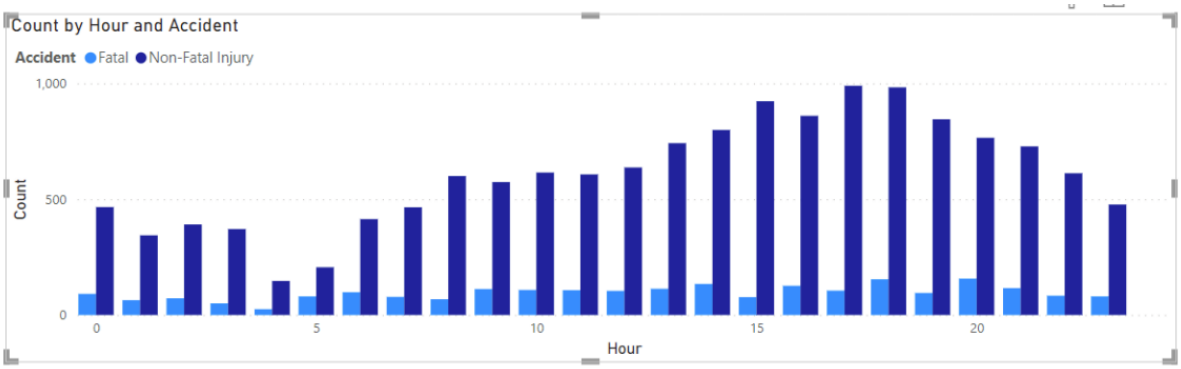
Analyze Data with Power BI

Count of accidents



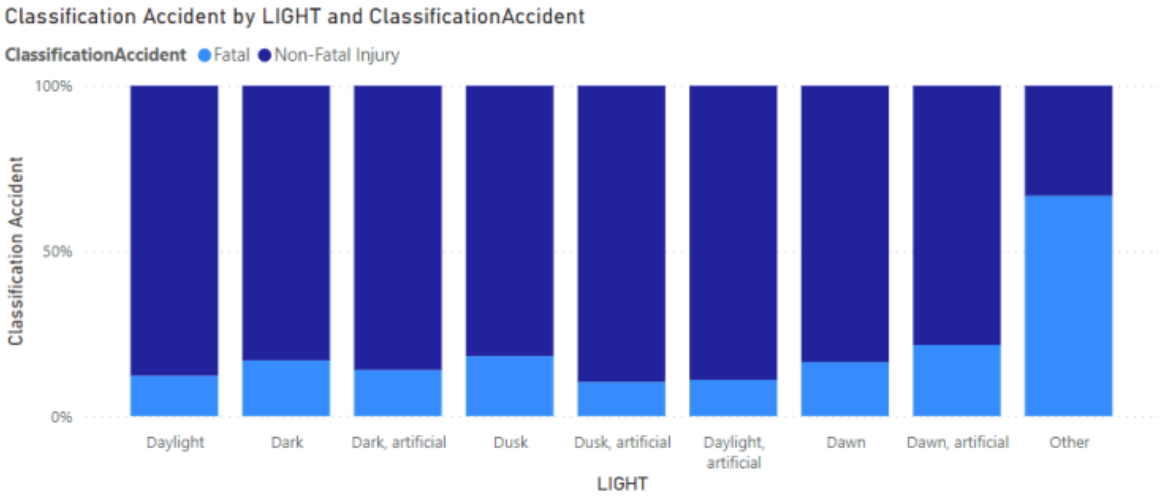
Relation between accidents and time

We can see that the total number of accident are high in evening hours where as no such conclusion can be derived for fatal accident

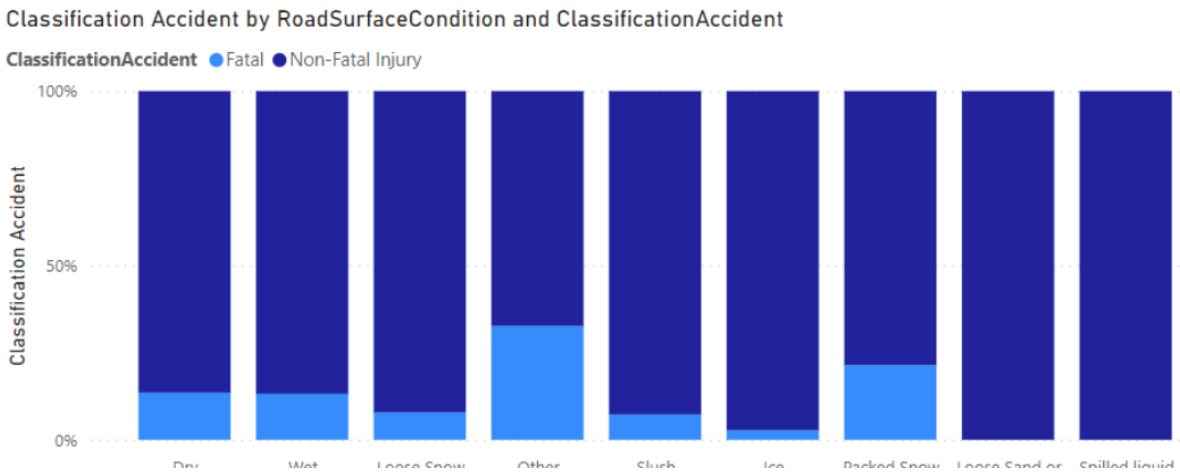


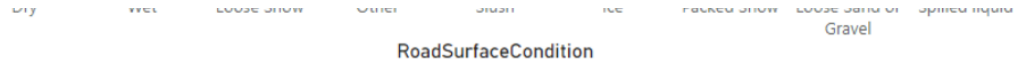
Percentage of accident by Light

We can see that the total number of accident are high in evening hours where as no such conclusion can be derived for fatal accident



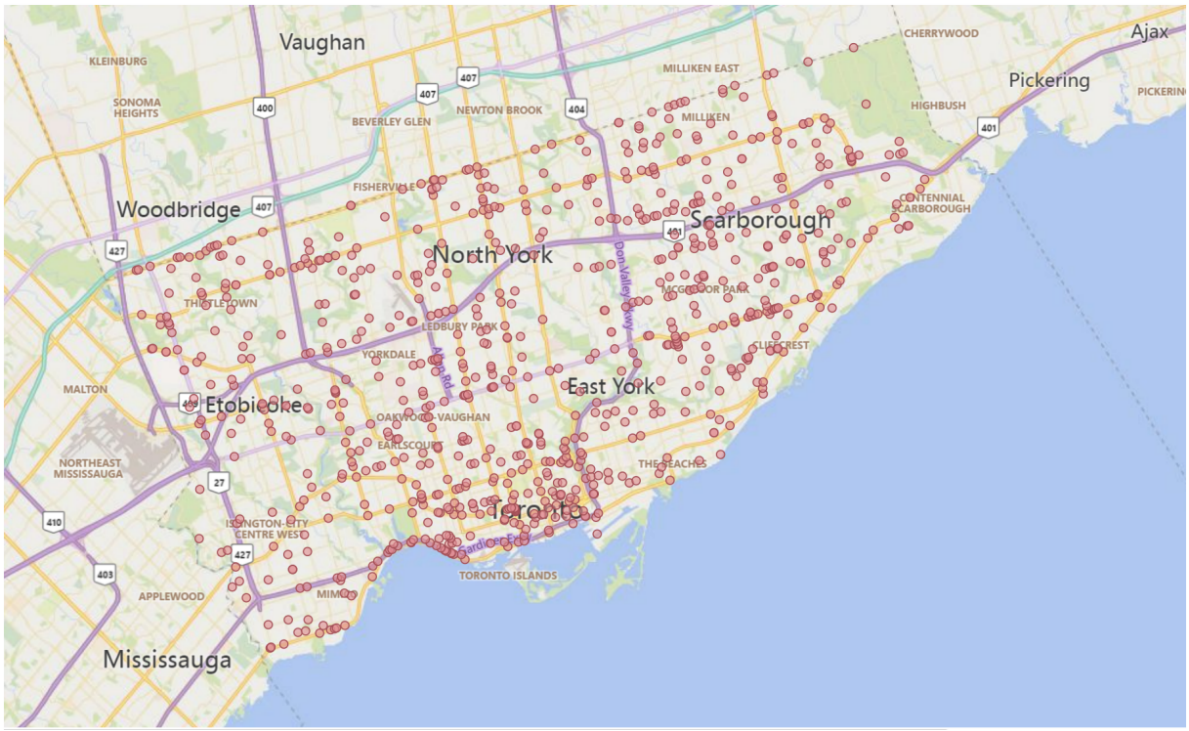
Percentage of accident by Road Condition



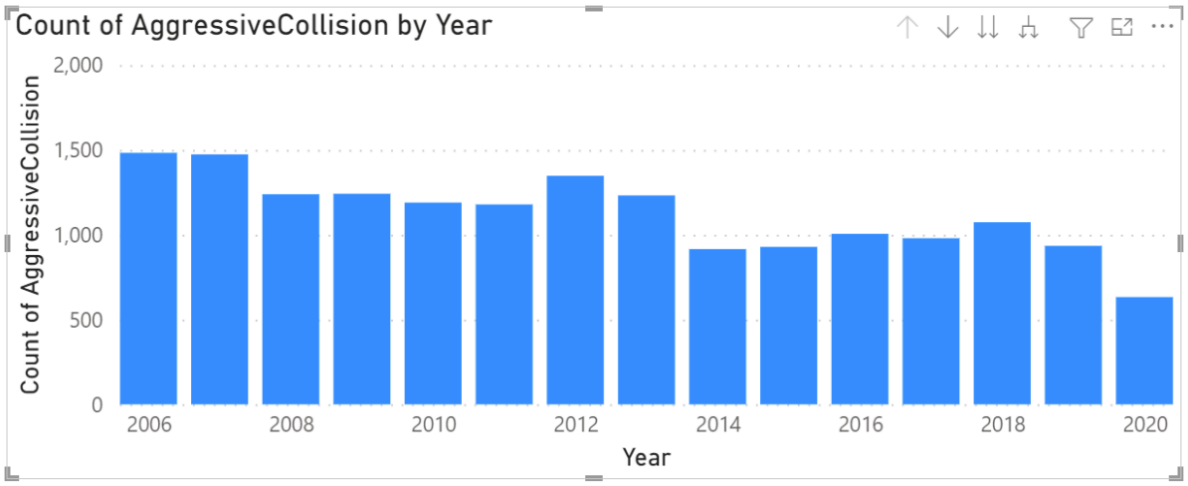


The Spots of all the fatal accidents

We can clearly see the hot spots



Counts of accident are falling

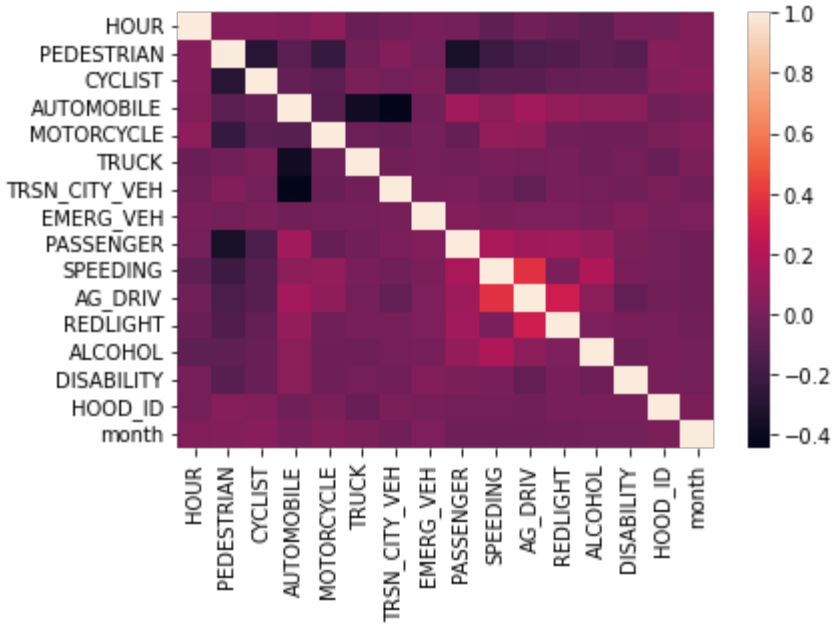


Correlation in the dataset

```
Trying to find corelation in the dataset
```

In [66]:
```
corr_matrix = data.corr()
sns.heatmap(corr_matrix)
```

Out[66]: <AxesSubplot:>



Cleaning Data - Replace Values

In [69]:
```
# preprocessing - clean data
```

In [ ]:

Analyze Data - Data Exploration Stats, Histogram, Graphs

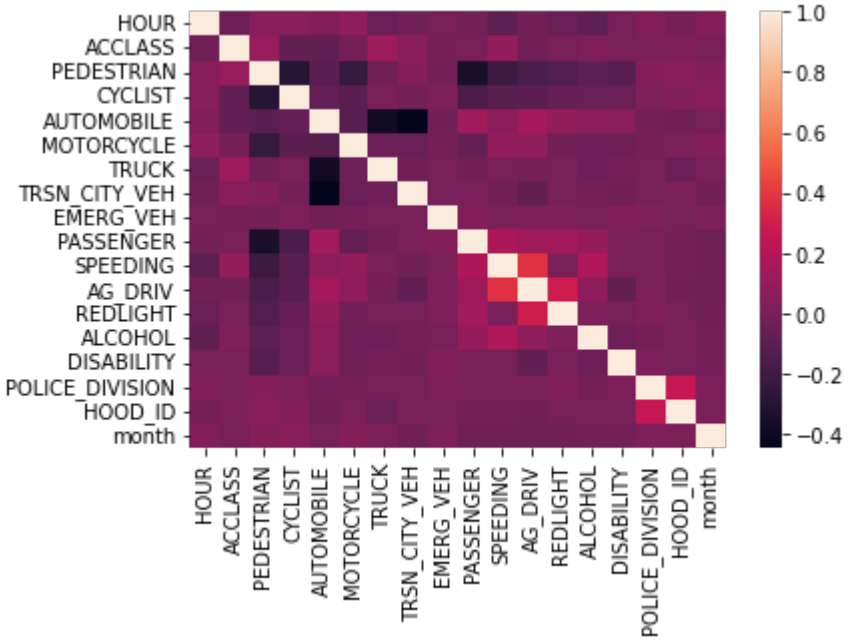
In [70]:
```
# Analyze data - after the second cleaning
analyze_data(data)
```

Data - 20 first rows

Correlation of the variables

In [73]:
```
corr_matrix = data.corr()
sns.heatmap(corr_matrix)
```

Out[73]: <AxesSubplot:>



In [79]:
```
corr_matrix = data.corr()
corr_matrix["ACCLASS"].sort_values(ascending=False)
```

Out[79]:
```
ACCLASS             1.000000
TRUCK               0.114711
PEDESTRIAN          0.100861
SPEEDING            0.089580
TRSN_CITY_VEH       0.048213
ALCOHOL             0.021518
HOOD_ID             0.015462
POLICE_DIVISION     0.007411
REDLIGHT           -0.000108
month              -0.001364
PASSENGER          -0.003197
DISABILITY         -0.004044
MOTORCYCLE         -0.012923
EMERG_VEH          -0.015988
AG_DRIV            -0.029194
HOUR               -0.037810
CYCLIST            -0.078454
AUTOMOBILE         -0.084198
Name: ACCLASS, dtype: float64
```

Build Classification Models

In [80]:
```
full_pipeline_transformer, X_group, Y_group, X_train, X_test, y_train, y_t

print(full_pipeline_transformer, X_group, Y_group, X_train, X_test, y_trai
```

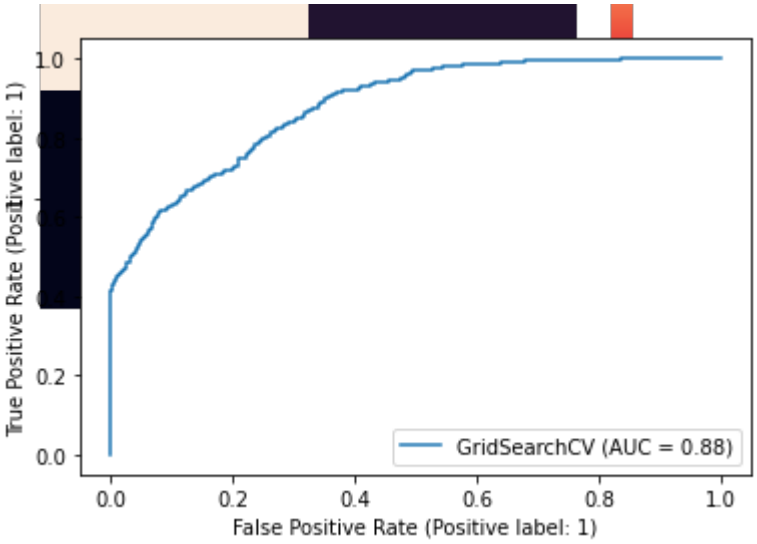
```
ColumnTransformer(transformers=[('num',
                                 Pipeline(steps=[('imputer',
                                                  SimpleImputer(strategy=
'median')),
                                                 ('std_scaler',
                                                  StandardScaler())]),
                                 ['HOUR', 'CYCLIST', 'AUTOMOBILE', 'MOTOR
CYCLE',
                                  'TRUCK', 'TRSN_CITY_VEH', 'EMERG_VEH',
                                  'SPEEDING', 'AG_DRIV', 'REDLIGHT', 'ALC
OHOL',
                                  'DISABILITY', 'PEDESTRIAN', 'PASSENGER
',
                                  'POLICE_DIVISION', 'HOOD_ID', 'month
']),
                                ('cat',
                                 Pipeline(steps=[('imputer',
                                                  SimpleImputer(fill_valu
e='missing',
```

Testing models

Logistic Regression

In [81]:
```
gs_logistic = get_best_model(data, 'LogisticRegression', full_pipeline_tra
```

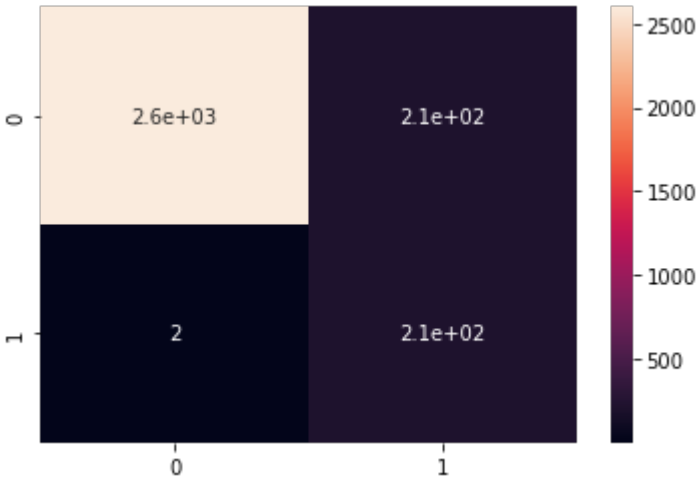
```
**********************************************************
Get Best Estimator/Params of the Model for  LogisticRegression
Best Estimator: Pipeline(steps=[('preprocessing',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imput
er',
                                                                   Simple
Imputer(strategy='median')),
                                                                  ('std_s
caler',
                                                                   Standa
rdScaler())]),
                                                  ['HOUR', 'CYCLIST',
                                                   'AUTOMOBILE', 'MOTORCY
CLE',
                                                   'TRUCK', 'TRSN_CITY_VE
H',
                                                   'EMERG_VEH', 'SPEEDING
',
                                                   'AG_DRIV', 'REDLIGHT',
                                                   'ALCOHOL', 'DISABILITY
',
                                                   'PEDESTRIAN', 'PASSENG
ER',
                                                   'POLICE_DIVISION', 'HO
OD...
                                                  Pipeline(steps=[('imput
er',
                                                                   Simple
Imputer(fill_value='missing',

strategy='constant')),
                                                                  ('encod
er',
                                                                   OneHot
Encoder(handle_unknown='ignore'))]),
                                                  ['ROAD_CLASS', 'DISTRIC
T',
                                                   'LOCCOORD', 'ACCLOC',
                                                   'TRAFFCTL', 'VISIBILIT
Y',
                                                   'LIGHT', 'RDSFCOND',
                                                   'IMPACTYPE', 'INVTYPE
',
                                                   'INJURY', 'VEHTYPE
'])])),
                ('classifier',
                 LogisticRegression(max_iter=1000, multi_class='multinomi
al',
                                    random_state=0))])
Best Params: {'classifier__max_iter': 1000, 'classifier__multi_class': 'm
ultinomial', 'classifier__random_state': 0, 'classifier__solver': 'lbfgs
```

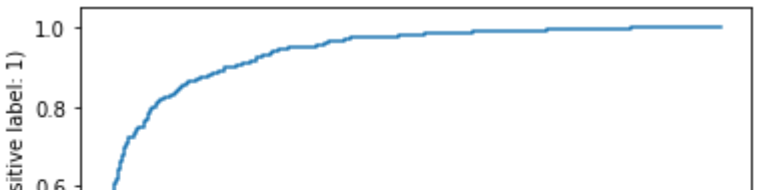


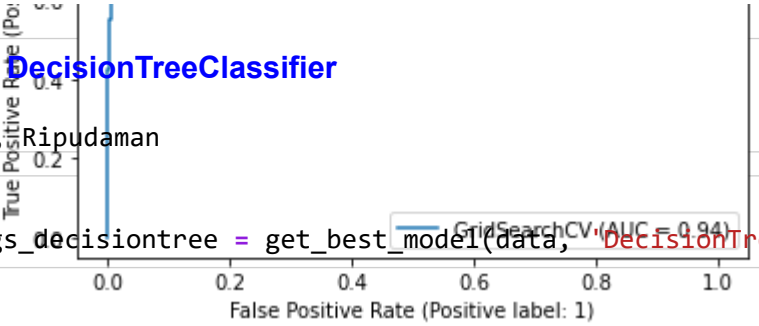
RandomForestClassifier

In [82]:

```
**********************************************************
Get Best Estimator/Params of the Model for  RandomForestClassifier
Best Estimator: Pipeline(steps=[('preprocessing',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imput
er',
                                                                   Simple
Imputer(strategy='median')),
                                                                  ('std_s
caler',
                                                                   Standa
rdScaler())]),
                                                  ['HOUR', 'CYCLIST',
                                                   'AUTOMOBILE', 'MOTORCY
CLE',
                                                   'TRUCK', 'TRSN_CITY_VE
H',
                                                   'EMERG_VEH', 'SPEEDING
',
                                                   'AG_DRIV', 'REDLIGHT',
                                                   'ALCOHOL', 'DISABILITY
',
                                                   'PEDESTRIAN', 'PASSENG
ER',
                                                   'POLICE_DIVISION', 'HO
OD...
                                                  Pipeline(steps=[('imput
er',
                                                                   Simple
Imputer(fill_value='missing',
strategy='constant')),
                                                                  ('encod
er',
                                                                   OneHot
```





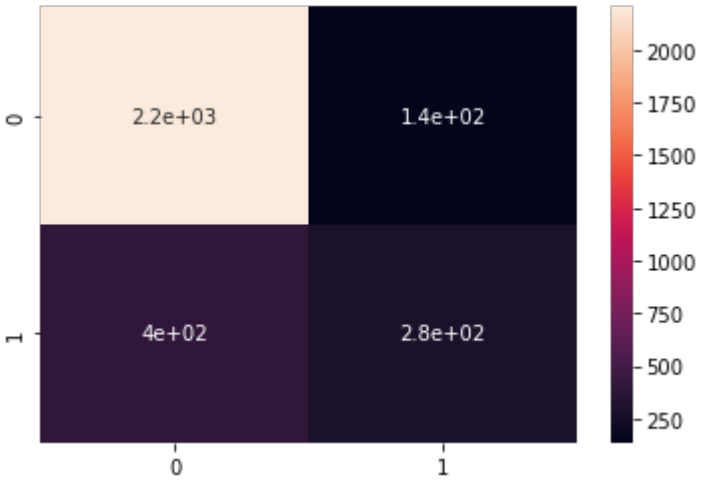


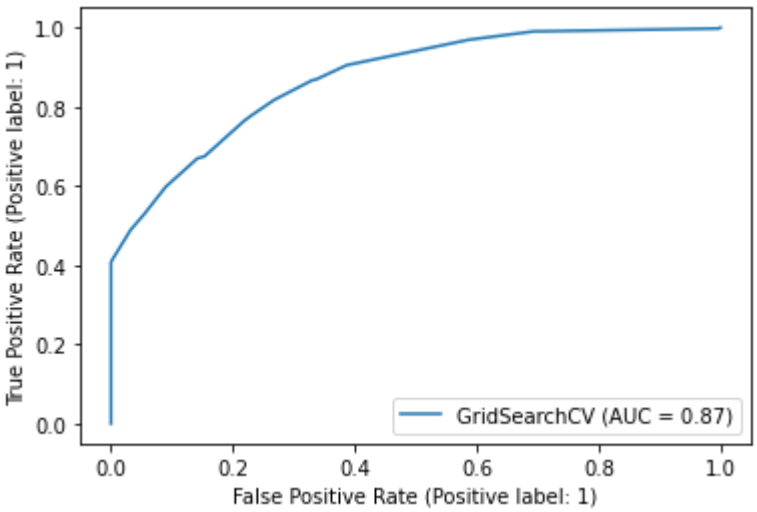
DecisionTreeClassifier

By .,Ripudaman

In [83]:
```
gs_decisiontree = get_best_model(data, 'DecisionTreeClassifier', full_pipe
```

```
************************************************************
Get Best Estimator/Params of the Model for  DecisionTreeClassifier
Best Estimator: Pipeline(steps=[('preprocessing',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imput
er',
                                                                   Simple
Imputer(strategy='median')),
                                                                  ('std_s
caler',
                                                                   Standa
rdScaler())]),
                                                  ['HOUR', 'CYCLIST',
                                                   'AUTOMOBILE', 'MOTORCY
CLE',
                                                   'TRUCK', 'TRSN_CITY_VE
H',
                                                   'EMERG_VEH', 'SPEEDING
',
                                                   'AG_DRIV', 'REDLIGHT',
                                                   'ALCOHOL', 'DISABILITY
',
                                                   'PEDESTRIAN', 'PASSENG
```



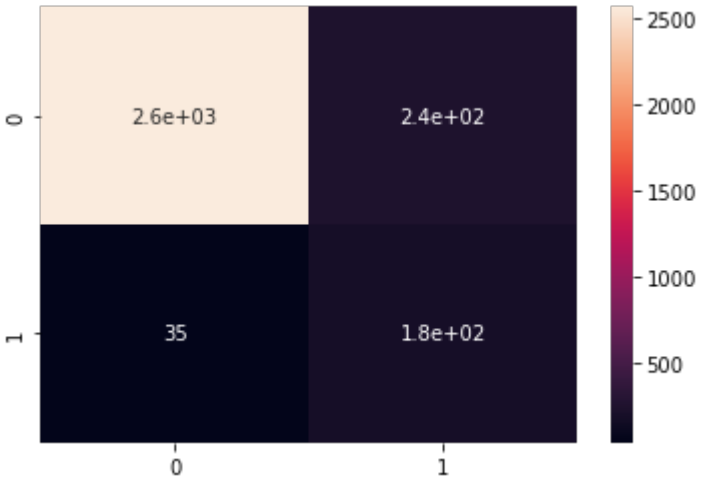


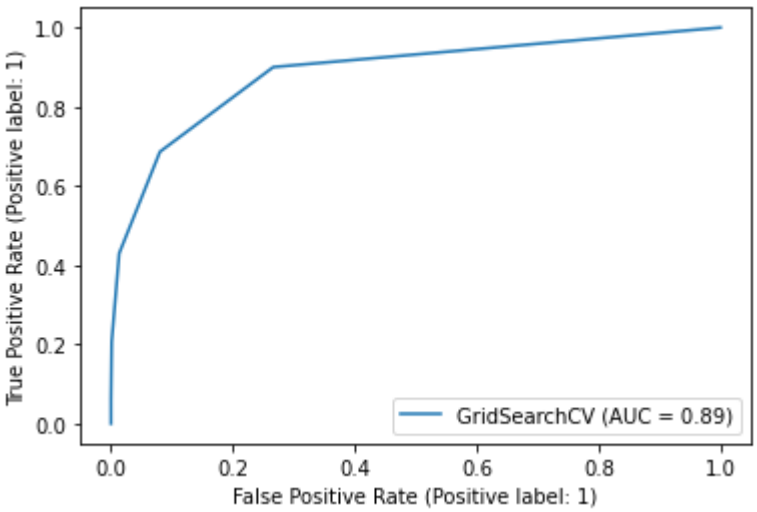
KNeighborsClassifier

In [84]:

```
gs_kneighbors = get_best_model(data, 'KNeighborsClassifier', full_pipeline
```

```
**********************************************************
Get Best Estimator/Params of the Model for  KNeighborsClassifier
Best Estimator: Pipeline(steps=[('preprocessing',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imput
er',
                                                                  Simple
Imputer(strategy='median')),
                                                                  ('std_s
caler',
                                                                  Standa
rdScaler())]),
                                                  ['HOUR', 'CYCLIST',
                                                   'AUTOMOBILE', 'MOTORCY
CLE',
                                                   'TRUCK', 'TRSN_CITY_VE
H'
```





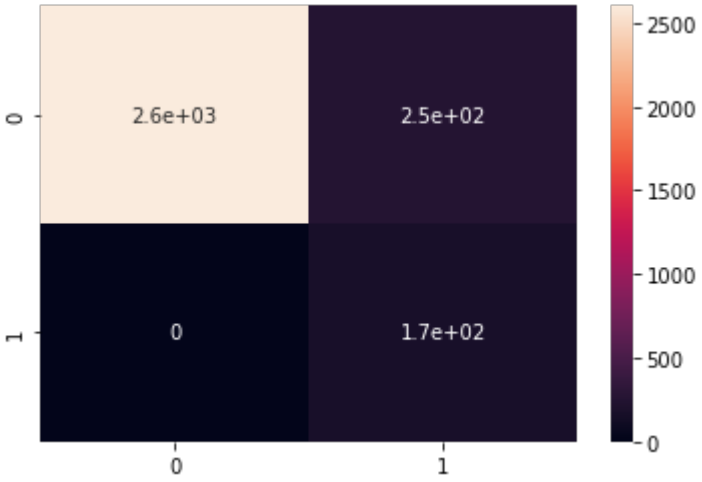
SVC

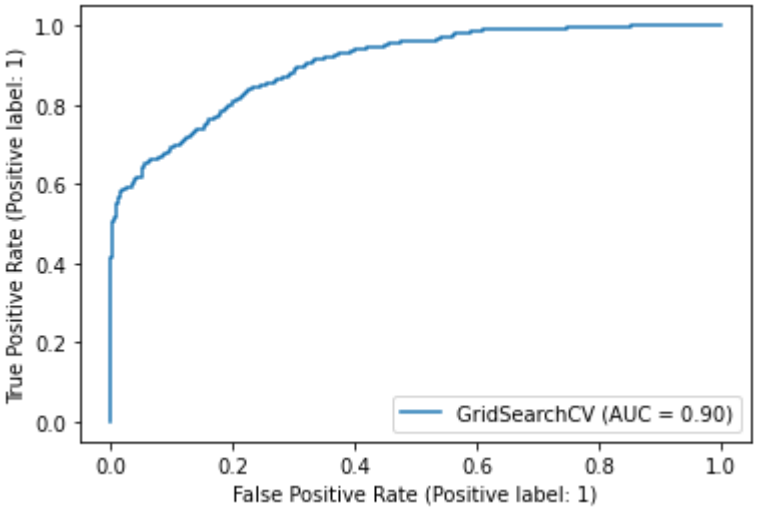
In [85]:

```
gs_svc = get_best_model(data, 'SVC', full_pipeline_transformer, X_train, X
```

```
**********************************************************
Get Best Estimator/Params of the Model for  SVC
Best Estimator: Pipeline(steps=[('preprocessing',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imput
er',
                                                                   Simple
Imputer(strategy='median')),
                                                                  ('std_s
caler',
                                                                   Standa
rdScaler())]),
                                                  ['HOUR', 'CYCLIST',
                                                   'AUTOMOBILE', 'MOTORCY
CLE',
                                                   'TRUCK', 'TRSN_CITY_VE
H',
                                                   'EMERG_VEH', 'SPEEDING
',
                                                   'AG_DRIV', 'REDLIGHT',
                                                   'ALCOHOL', 'DISABILITY
',
                                                   'PEDESTRIAN', 'PASSENG
ER',
                                                   'POLICE_DIVISION', 'HO
OD...
                                                   'month']),
                                                 ('cat',
                                                  Pipeline(steps=[('imput
er',
                                                                   Simple
Imputer(fill_value='missing',

strategy='constant')),
                                                                  ('encod
er',
                                                                   OneHot
Encoder(handle_unknown='ignore'))]),
                                                  ['ROAD_CLASS', 'DISTRIC
T',
                                                   'LOCCOORD', 'ACCLOC',
                                                   'TRAFFCTL', 'VISIBILIT
Y',
                                                   'LIGHT', 'RDSFCOND',
                                                   'IMPACTYPE', 'INVTYPE
',
                                                   'INJURY', 'VEHTYPE
'])])),
                ('classifier',
                 SVC(gamma='auto', probability=True, random_state=42))])
Best Params: {'classifier__gamma': 'auto', 'classifier__kernel': 'rbf'}
Best Score: 0.8857828761006091
Test Precision: 0.40617577197149646
Test Recall: 1.0
Test ROC AUC Score: 0.95617110799439
Test Accuracy Score =  0.9173006946741648
```

```
Test Confusion Matrix =
 [[2602  250]
 [   0  171]]
Test Classification Report =
              precision    recall  f1-score   support

           0       1.00      0.91      0.95      2852
           1       0.41      1.00      0.58       171

    accuracy                           0.92      3023
   macro avg       0.70      0.96      0.77      3023
```





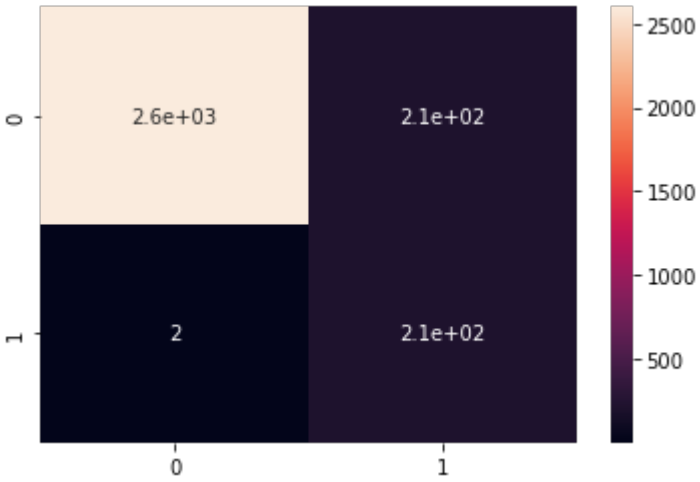
In [ ]:
```
#### ALL MODELS - BEST MODEL
```

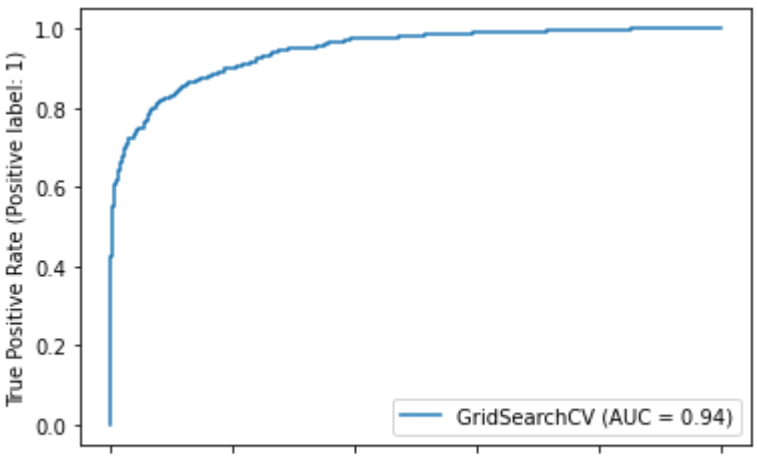
In [87]:
```
gs_all = get_best_model(data, 'ALL', full_pipeline_transformer, X_train, X
```

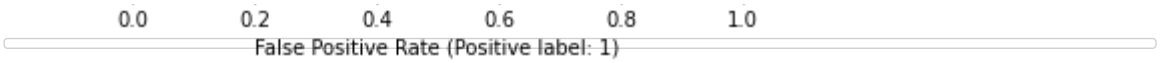
```
**********************************************************
Get Best Estimator/Params of the Model for  ALL
```

```
C:\Users\gmi_r\anaconda3\lib\site-packages\sklearn\model_selection\_searc
h.py:922: UserWarning: One or more of the test scores are non-finite: [0.
88172326 0.88165179 0.88172326 0.88165162 0.88194745 0.88194144
 0.88194745 0.88194007 0.79644394 0.84888876 0.84879471 0.84879471
 0.84879471 0.79644394 0.8449205  0.85440757 0.85440757 0.85440757
 0.79644394 0.8447095  0.85913766 0.859801   0.859801   0.79644394
 0.84009016 0.85377873 0.85301047 0.85301047 0.80119419 0.84375561
 0.85595615 0.85374176 0.85374176 0.79644394 0.84795683 0.85234213
 0.85200216 0.85200216 0.80119419 0.83470881 0.84416226 0.84416226
 0.84416226 0.79831062 0.84756242 0.85261274 0.85261274 0.85261274
 0.8480353  0.87165063 0.87542915 0.87691953 0.88888327 0.90657255
 0.90687546 0.9081309  0.90387658 0.92685205 0.92944079 0.93145717
 0.85699574 0.87532093 0.87828968 0.87811059 0.88657488 0.89624005
 0.89906619 0.90185237 0.89855492 0.92170611 0.92691185 0.93008869
 0.85770906 0.8734701  0.87738306 0.8774629  0.88443776 0.89277381
Best Estimator: Pipeline(steps=[('preprocessing',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imput
er',
                                                                   Simple
Imputer(strategy='median')),
                                                                  ('std_s
caler',
                                                                   Standa
rdScaler())]),
                                                  ['HOUR', 'CYCLIST',
                                                   'AUTOMOBILE', 'MOTORCY
CLE',
                                                   'TRUCK', 'TRSN_CITY_VE
H',
                                                   'EMERG_VEH', 'SPEEDING
',
                                                   'AG_DRIV', 'REDLIGHT',
                                                   'ALCOHOL', 'DISABILITY
',
                                                   'PEDESTRIAN', 'PASSENG
ER',
                                                   'POLICE_DIVISION', 'HO
OD...
                                                  Pipeline(steps=[('imput
er',
                                                                   Simple
Imputer(fill_value='missing',

strategy='constant')),
                                                                  ('encod
er',
                                                                   OneHot
Encoder(handle_unknown='ignore'))]),
                                                  ['ROAD_CLASS', 'DISTRIC
T',
                                                   'LOCCOORD', 'ACCLOC',
                                                   'TRAFFCTL', 'VISIBILIT
Y',
                                                   'LIGHT', 'RDSFCOND',
```

```
                                                    'IMPACTYPE', 'INVTYPE
',
                                                    'INJURY', 'VEHTYPE
'])])),
                ('classifier',
                 RandomForestClassifier(max_depth=20, n_estimators=250,
                                        random_state=42))])
Best Params: {'classifier': RandomForestClassifier(max_depth=20, n_estima
tors=250, random_state=42), 'classifier__class_weight': None, 'classifier
__max_depth': 20, 'classifier__n_estimators': 250}
Best Score: 0.9314571740975975
Test Precision: 0.498812351543943
Test Recall: 0.9905660377358491
Test ROC AUC Score: 0.9577518911553666
Test Accuracy Score =  0.9295401918623883
Test Confusion Matrix =
 [[2600  211]
 [   2  210]]
Test Classification Report =
              precision    recall  f1-score   support

           0       1.00      0.92      0.96      2811
           1       0.50      0.99      0.66       212

    accuracy                           0.93      3023
   macro avg       0.75      0.96      0.81      3023
weighted avg       0.96      0.93      0.94      3023
```







In [ ]: