Road Traffic Accident Prediction

MGSC-5125-11: Predictive Modelling & Analytics

GROUP H

Submitted to:

Prof. Enayat Rajabi

Submitted by:

Aditya Prashant Gurjar (0274831)

Dinesh Rawat (0270404)

Donna Casa (0272900)

Mohammad Sabir Shah (0269062)

Table of Contents

Table of Figures

Business Understanding

Road traffic accidents pose a significant threat to public safety, causing substantial economic and human losses worldwide. Understanding the factors that contribute to road accidents is crucial for developing effective prevention strategies. This project aims to develop a predictive model using machine learning techniques to identify patterns and relationships in historical road accident data, enabling a more comprehensive understanding of accident-causing factors and the prediction of accident severity. By accurately predicting road accident severity, proactive measures can be implemented to mitigate risks, enhance road safety, and save lives.

Problem Statement:

To determine the severity of road accidents whether it is a slight injury, serious injury, or fatal injury based on multiple factors such as road conditions, age, gender, etc.

The dataset is taken from Kaggle:

https://www.kaggle.com/datasets/saurabhshahane/road-traffic-accidents

The original data was collected from Addis Ababa Sub city police departments for the year 2017 to 2020.

Dataset Description (Metadata):

Rows: 12316, Columns: 32
Categorical variables: 30, Continuous variables: 2

Columns	**Description**
Time	Time of the accident
Day_of_week	Day of the week
Age_band_of_driver	Age band of drivers. 18-30, 31-50 etc
Sex_of_driver	Sex of driver
Educational_level	Educational level. High school, elementary etc
Vehicle_driver_relation	Driver relation to vehicle. Employee, Owner etc.
Driving_experience	Driving experience. 5-10, 2-5 etc.
Type_of_vehicle	Type of vehicle. Automobile, lorry etc.
Owner_of_vehicle	Type of owner
Service_year_of_vehicle	Service year of the vehicle
Defect_of_vehicle	Any defects present in vehicle
Area_accident_occured	Type of area where accident occurred
Lanes_or_Medians	Type of lanes or median
Road_allignment	Alignment of the road
Types_of_Junction	Junction type. Y-shape etc
Road_surface_type	Road surface type. Asphalt, earth etc
Road_surface_conditions	Road surface conditions. Dry, damp etc.
Light_conditions	Light conditions during accident. Daylight, Darkness etc.

Weather_conditions	Weather conditions. Raining, normal etc.
Type_of_collision	Type of collision
Number_of_vehicles_involved	Number of vehicles involved
Number_of_casualties	Number of casualties
Vehicle_movement	Vehicle movement. Going straight, backwards etc
Casualty_class	Class of Casulaty. Driver, pedestrian etc
Sex_of_casualty	Sex of the casualty
Age_band_of_casualty	Age band of casualty
Casualty_severity	Casualty severity
Work_of_casuality	work of casualty
Fitness_of_casuality	average fitness of casualty
Pedestrian_movement	Pedestrain involved if any
Cause_of_accident	Cause of the accident
Accident_severity	Accident severity. Slight injury, serious injury and Fatal

Data Understanding

Libraries Used:

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from sklearn import preprocessing
from sklearn import datasets
from sklearn.feature_selection import chi2
from sklearn.feature_selection import mutual_info_classif
from sklearn.datasets import load_wine
from sklearn.metrics import RocCurveDisplay
from sklearn.feature_selection import SelectKBest
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, f1_score, accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from yellowbrick.classifier import ROCAUC
from yellowbrick.datasets import load_game
warnings.filterwarnings('ignore')
```

Dataset Review:

```python
# reading data from csv and loading it
dataset=pd.read_csv("RTA Dataset.csv")
```

```python
dataset.head()
```

	Time	Day_of_week	Age_band_of_driver	Sex_of_driver	Educational_level	Vehicle_driver_relation	Driving_experience	Type_of_vehicle	Owner_of_vehicle
0	17:02:00	Monday	18-30	Male	Above high school	Employee	1-2yr	Automobile	Owner
1	17:02:00	Monday	31-50	Male	Junior high school	Employee	Above 10yr	Public (> 45 seats)	Owner
2	17:02:00	Monday	18-30	Male	Junior high school	Employee	1-2yr	Lorry (41?100Q)	Owner
3	1:06:00	Sunday	18-30	Male	Junior high school	Employee	5-10yr	Public (> 45 seats)	Governmental
4	1:06:00	Sunday	18-30	Male	Junior high school	Employee	2-5yr	NaN	Owner

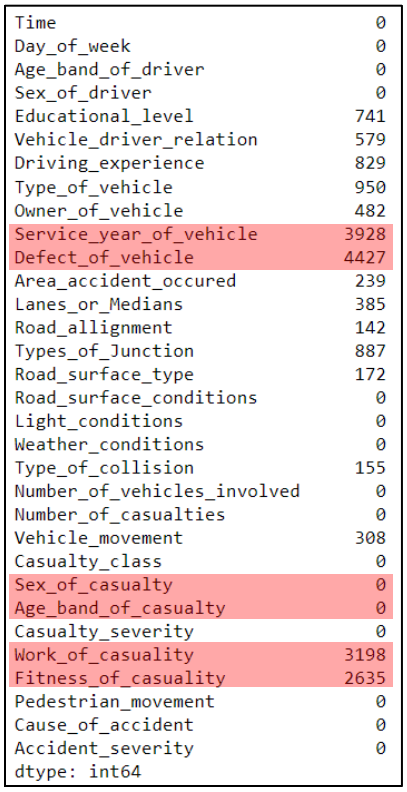
5 rows × 32 columns

We observed that our dataset mostly contains categorical variables and few numerical variables. Also, the dataset needs to be cleaned.

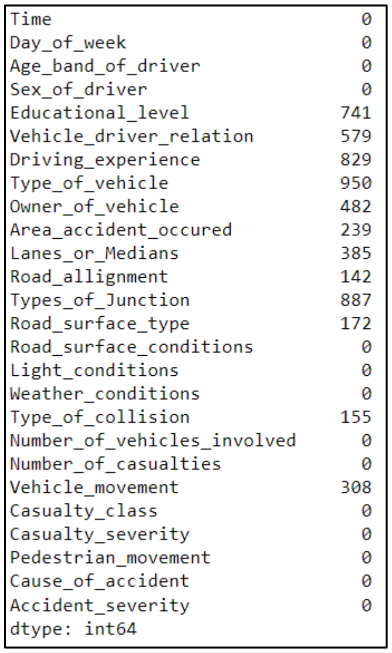
Data Cleaning

Checking For Missing Values:

Checking for columns with high number of missing values and dropping them. Also, dropping columns which does not add value to the problem statement.



```
Time                              0
Day_of_week                       0
Age_band_of_driver                0
Sex_of_driver                     0
Educational_level               741
Vehicle_driver_relation         579
Driving_experience              829
Type_of_vehicle                 950
Owner_of_vehicle                482
Service_year_of_vehicle        3928
Defect_of_vehicle              4427
Area_accident_occured           239
Lanes_or_Medians                385
Road_allignment                 142
Types_of_Junction               887
Road_surface_type               172
Road_surface_conditions           0
Light_conditions                  0
Weather_conditions                0
Type_of_collision               155
Number_of_vehicles_involved       0
Number_of_casualties              0
Vehicle_movement                308
Casualty_class                    0
Sex_of_casualty                   0
Age_band_of_casualty              0
Casualty_severity                 0
Work_of_casuality              3198
Fitness_of_casuality           2635
Pedestrian_movement               0
Cause_of_accident                 0
Accident_severity                 0
dtype: int64
```



```
Time                             0
Day_of_week                      0
Age_band_of_driver               0
Sex_of_driver                    0
Educational_level              741
Vehicle_driver_relation        579
Driving_experience             829
Type_of_vehicle                950
Owner_of_vehicle               482
Area_accident_occured          239
Lanes_or_Medians               385
Road_allignment                142
Types_of_Junction              887
Road_surface_type              172
Road_surface_conditions          0
Light_conditions                 0
Weather_conditions               0
Type_of_collision              155
Number_of_vehicles_involved      0
Number_of_casualties             0
Vehicle_movement               308
Casualty_class                   0
Casualty_severity                0
Pedestrian_movement              0
Cause_of_accident                0
Accident_severity                0
dtype: int64
```

Checking For Null Values:

```
categorical = ['Day_of_week', 'Age_band_of_driver', 'Sex_of_driver', 'Educational_level', 'Vehicle_driver_relation',
               'Driving_experience', 'Type_of_vehicle', 'Owner_of_vehicle', 'Area_accident_occured', 'Lanes_or_Medians',
               'Road_allignment', 'Types_of_Junction', 'Road_surface_type', 'Road_surface_conditions', 'Light_conditions',
               'Weather_conditions', 'Type_of_collision', 'Vehicle_movement', 'Casualty_severity', 'Pedestrian_movement',
               'Cause_of_accident', 'Accident_severity']

print('The categorical variables are',categorical)
```

```
The categorical variables are ['Day_of_week', 'Age_band_of_driver', 'Sex_of_driver', 'Educational_level', 'Vehicle_driver_relat
ion', 'Driving_experience', 'Type_of_vehicle', 'Owner_of_vehicle', 'Area_accident_occured', 'Lanes_or_Medians', 'Road_allignmen
t', 'Types_of_Junction', 'Road_surface_type', 'Road_surface_conditions', 'Light_conditions', 'Weather_conditions', 'Type_of_col
lision', 'Vehicle_movement', 'Casualty_severity', 'Pedestrian_movement', 'Cause_of_accident', 'Accident_severity']
```

```
#for categorical values we can replace the null values with Unknown
for i in categorical:
    dataset[i].fillna('Unknown',inplace=True)
```

```
#now no null values should be in cols
dataset.isna().sum()
```

Output:

```
Time                           0
Day_of_week                    0
Age_band_of_driver             0
Sex_of_driver                  0
Educational_level              0
Vehicle_driver_relation        0
Driving_experience             0
Type_of_vehicle                0
Owner_of_vehicle               0
Area_accident_occured          0
Lanes_or_Medians               0
Road_allignment                0
Types_of_Junction              0
Road_surface_type              0
Road_surface_conditions        0
Light_conditions               0
Weather_conditions             0
Type_of_collision              0
Number_of_vehicles_involved    0
Number_of_casualties           0
Vehicle_movement               0
Casualty_class                 0
Casualty_severity              0
Pedestrian_movement            0
Cause_of_accident              0
Accident_severity              0
dtype: int64
```

We created an array 'categorical' which has all column names with categorical data and replaced the null values with 'Unknown'.

Cleaning Data With Same Meaning:

- From column 'Driving_experience', replaced 'unknown' with 'Unknown'.
- From column 'Type_of_vehicle', replaced all values with 'Public (x seats)' to just 'Public' for simplicity and all values which contained 'Lorry' to just 'Lorry' for simplicity.

Replacing Keyword 'na' To 'Unknown' In The Entire Dataset:

```python
#cleaning keyword na from rows to Unknown
for col in dataset.columns:
    for index, row in dataset.loc[:, [col]].iterrows():
        if(row[col]=='na'):
            dataset.loc[index, col] = 'Unknown'
```

Considering Only Hour Of The Day From Column 'Time':

```
# convert object type column into datetime datatype column
# hh:mm:ss to hh to remove specific time values
dataset['Time'] = pd.to_datetime(dataset['Time'])
```

```
dataframe = dataset.copy()
dataframe['Hour_of_Day'] = dataframe['Time'].dt.hour
dataframe = dataframe.drop('Time', axis=1)
dataframe.head()
```

_involved	Number_of_casualties	Vehicle_movement	Casualty_class	Casualty_severity	Pedestrian_movement	Cause_of_accident	Accident_severity	Hour_of_Day
2	2	Going straight	Unknown	Unknown	Not a Pedestrian	Moving Backward	Slight Injury	17
2	2	Going straight	Unknown	Unknown	Not a Pedestrian	Overtaking	Slight Injury	17
2	2	Going straight	Driver or rider	3	Not a Pedestrian	Changing lane to the left	Serious Injury	17
2	2	Going straight	Pedestrian	3	Not a Pedestrian	Changing lane to the right	Slight Injury	1
2	2	Going straight	Unknown	Unknown	Not a Pedestrian	Overtaking	Slight Injury	1

Section Summary:

The initial dataset comprised 32 columns. Following data cleaning and the removal of columns with missing values, the resulting dataset, named 'dataframe,' now consists of 26 columns. This 'dataframe' will be utilized in the subsequent stage of the project.

Data Exploration

Visualizing Target Variable ‘Accident_severity’:

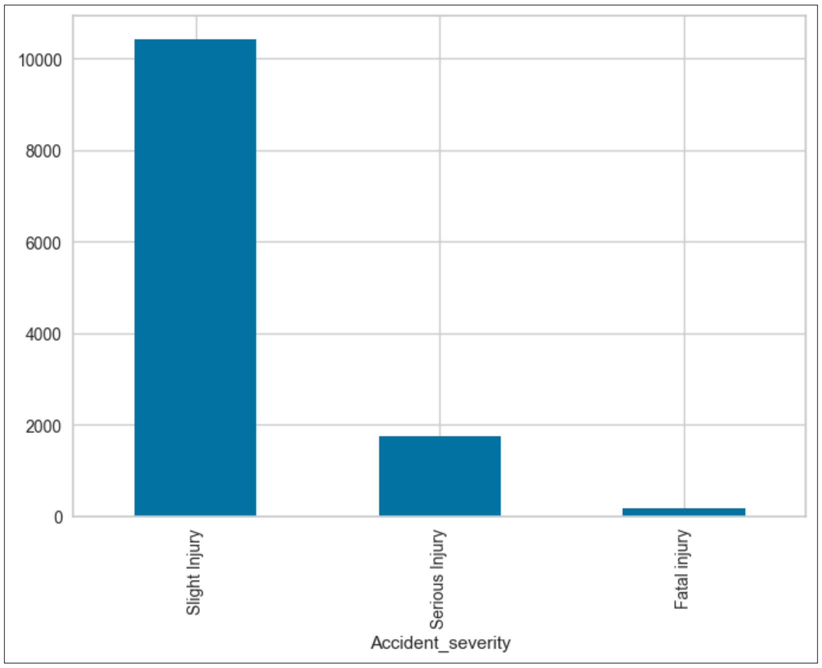


Fig 1: Count of Accident_severity

It is observed from the above bar chart that the dataframe contains more number of 'Slight injury' cases of accident.

Visualizing ‘Education_level’ Of The Drivers:

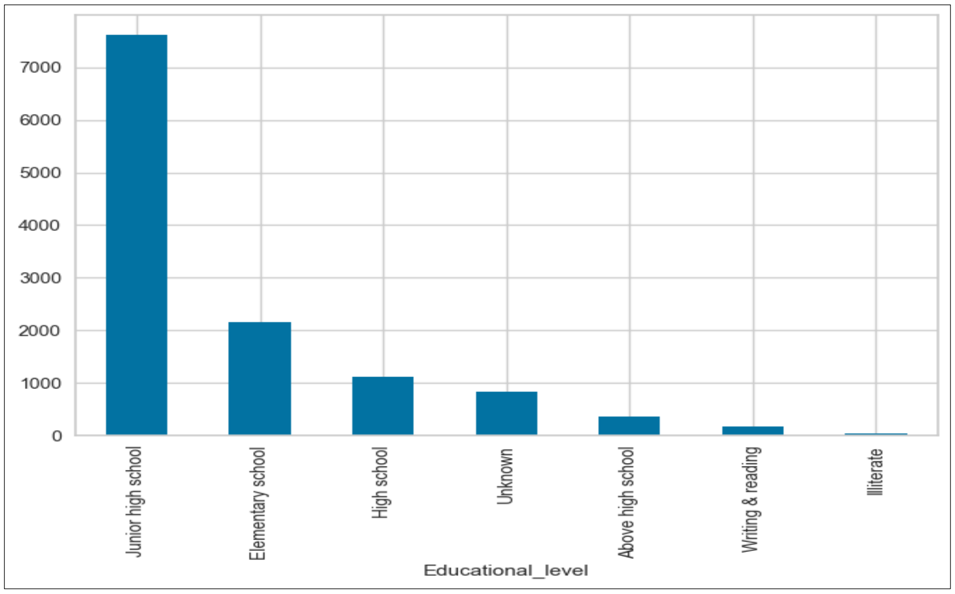


Fig 2: Count of ‘Education_level’ of the drivers

This dataframe contains more number of drivers with education level 'Junior high school'.

Visualizing 'Educational_level' With Target Variable 'Accident_severity’:

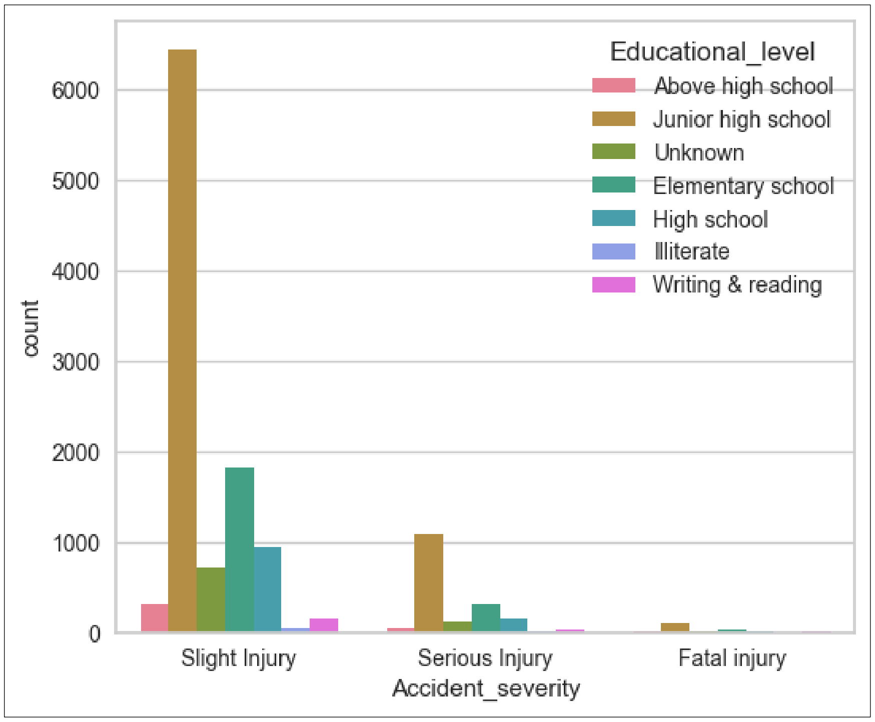


Fig 3: Accident_severity Vs Educational_level

Most drivers in each type of accident severity have education level 'Junior high school'.

Visualizing 'Road_surface_type' With Target Variable 'Accident_severity':

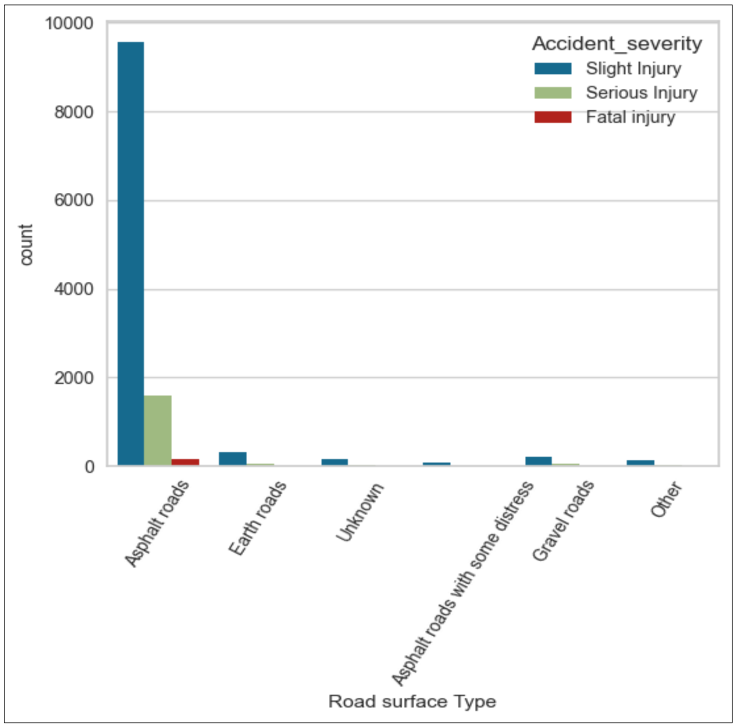


Fig 4: Accident_severity Vs Road_surface_type

Most accidents occur on asphalt roads with slight or serious injury to the driver.

Visualizing 'Road_surface_conditions' With Target Variable 'Accident_severity':

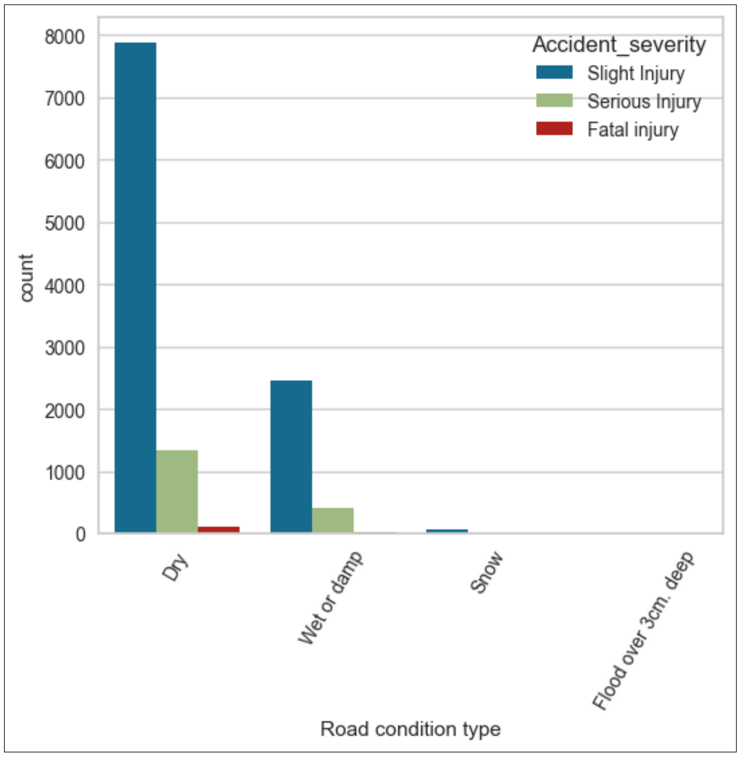


Fig 5: Accident_severity Vs Road_surface_conditions

Most accidents occur on dry and wet/damp road conditions.

Visualizing 'Hour_of_Day' With Target Variable 'Accident_severity':

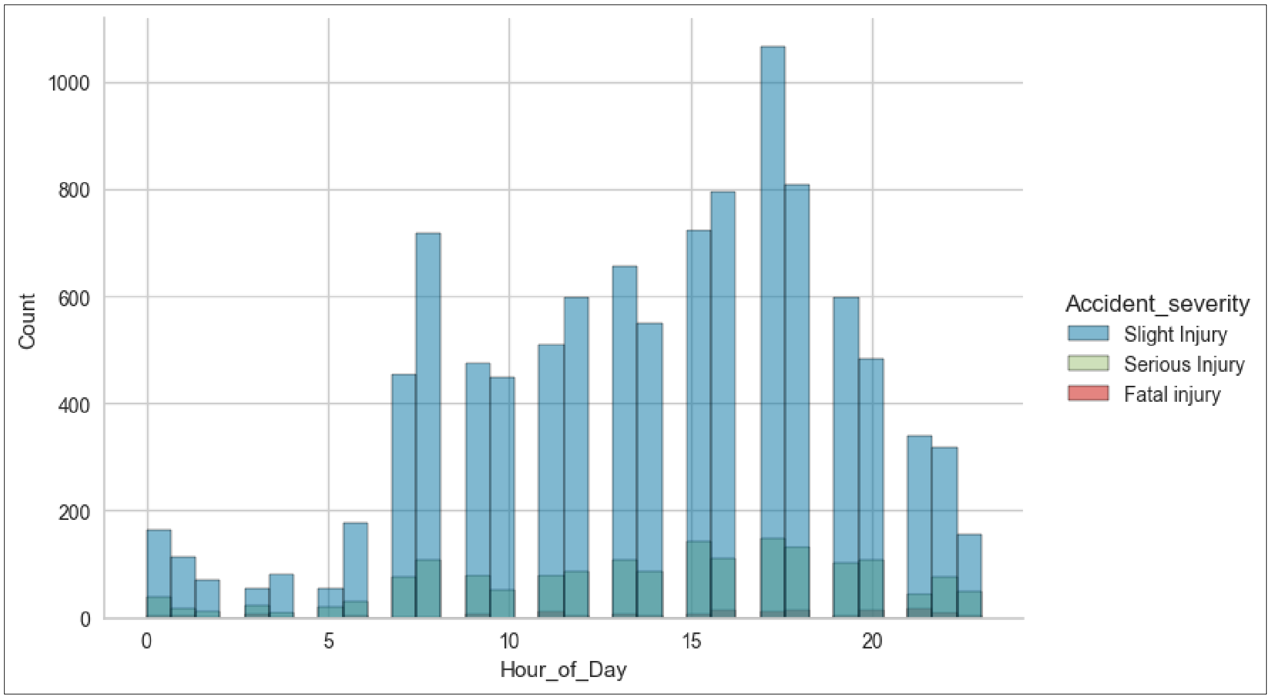
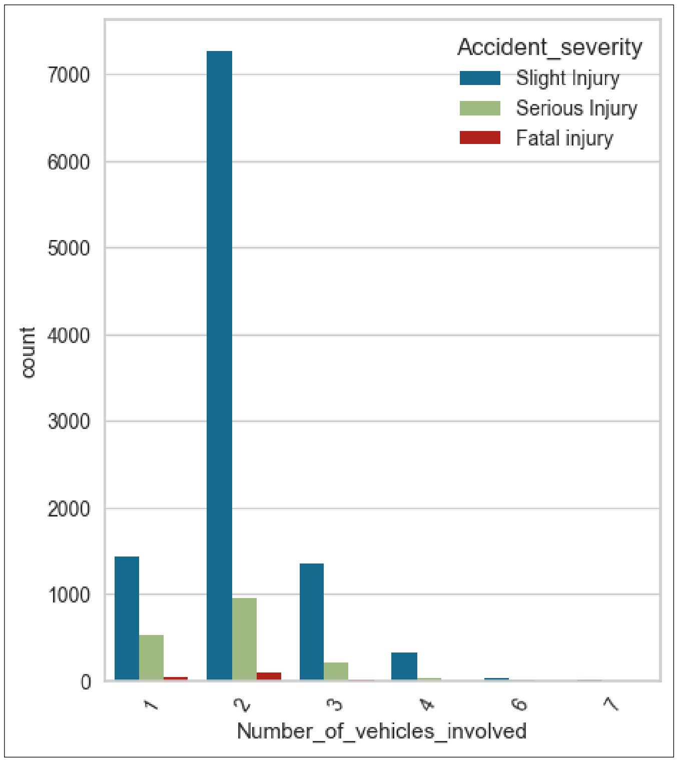


Fig 6: Accident_severity Vs Hour_of_Day

The number of accidents tend to increase during the daytime and start to decrease after evening.

Visualizing Numerical Columns With Target Variable 'Accident_severity':



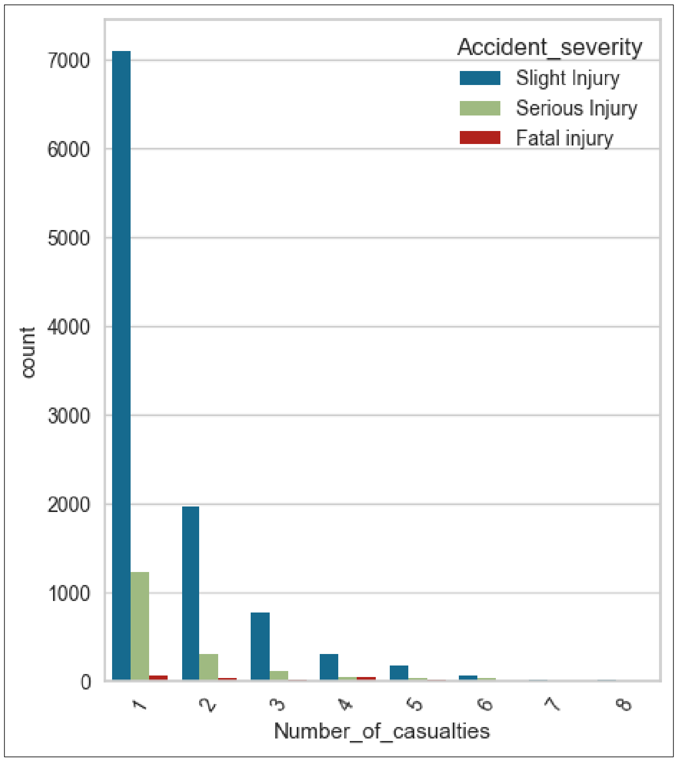


Fig 7: Count of 'Number_of_vehicles_involved' and 'Number_of_casualties'

It is seen that numerical columns do not carry any meaning that would help in solving our problem statement. Hence, we have omitted them in our further steps.

Section Summary:

- Data is highly imbalanced.
- Hour of day seems to be important to predict accident severity.
- Categorical columns need to be encoded.

Feature Selection

Feature selection is a critical step in the process of building machine learning models and conducting data analysis. It involves choosing a subset of relevant and significant features (variables or attributes) from the original set of features in a dataset. The objective is to improve the model's performance, reduce complexity, and enhance interpretability by focusing on the most informative features.

One Hot Encoding And Label Encoding:

We selected all categorical cols as features and ran one hot encoding to convert into dummy variables. We used one hot encoding for our dataset as label encoding may introduce an arbitrary order to categorical values. One-hot encoding avoids this issue by creating binary columns for each category, though it can lead to high-dimensional data.

```
features = ['Day_of_week', 'Age_band_of_driver', 'Sex_of_driver', 'Educational_level', 'Vehicle_driver_relation',
            'Driving_experience', 'Type_of_vehicle', 'Owner_of_vehicle', 'Area_accident_occured', 'Lanes_or_Medians',
            'Road_allignment', 'Types_of_Junction', 'Road_surface_type', 'Road_surface_conditions', 'Light_conditions',
            'Weather_conditions', 'Type_of_collision', 'Vehicle_movement', 'Casualty_severity', 'Pedestrian_movement',
            'Cause_of_accident', 'Hour_of_Day']

feature_dataframe = dataframe[features]
target = dataframe['Accident_severity']
numerical_df = dataset.filter(["Number_of_vehicles_involved", 'Number_of_casualties'], axis=1)
```

```
#encoding into dummy variables using get dummies
x = feature_dataframe[features]
y = target
encoded_df = pd.get_dummies(x, dtype='int', drop_first=True)
encoded_df.shape
```

```
(12316, 150)
```

Target Variable Was Encoded Using Label Encoder:

```
#target encoding using label encoder
label_encoder = preprocessing.LabelEncoder()
label_encoder.fit(y)
print("Encoded labels:",label_encoder.classes_)
target_en= label_encoder.fit_transform(y)
target_en

# 1: serious injury, 2: Slight injury, 0: Fatal Injury
```

```
Encoded labels: ['Fatal injury' 'Serious Injury' 'Slight Injury']

array([2, 2, 1, ..., 1, 2, 2])
```

A final feature data frame was created by merging the continuous features with encoded features.

```
feature_df = pd.concat([encoded_df, numerical_df], axis="columns")
feature_list = list(feature_df.columns)
feature_df
```

Imbalance Data Treatment Using SMOTE:

SMOTE stands for Synthetic Minority Over-sampling Technique which is a method commonly used to address imbalanced datasets in machine learning. In our case “serious injury” and “slight injury” are underrepresented in dataset. To remove this imbalance, we ran SMOTE on our feature set and target variables.

```
#synthetic minority oversampling technique
oversample = SMOTE()
X_smoted, y_smoted = oversample.fit_resample(feature_df, target_en)
y_smoted = pd.Series(y_smoted)
X_smoted.shape, y_smoted.shape
```

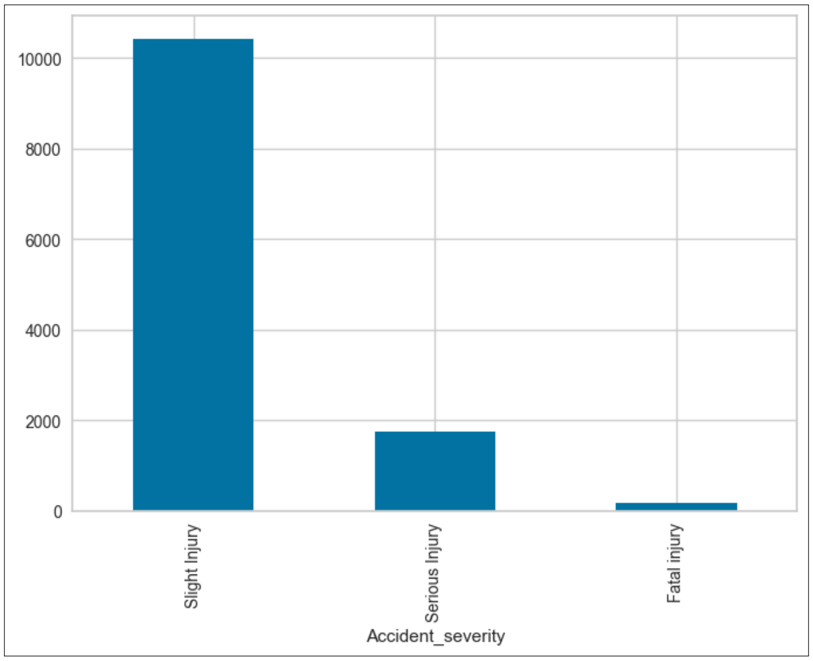


Fig 8: Target variable before SMOTE

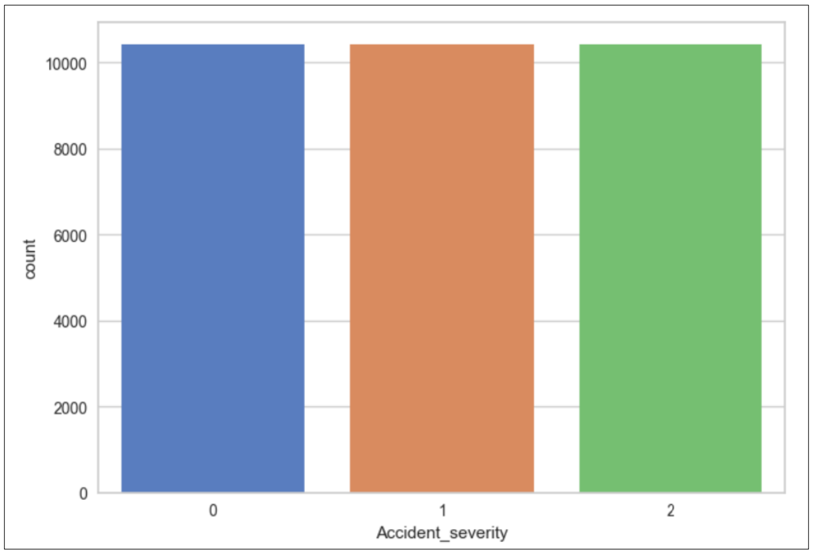
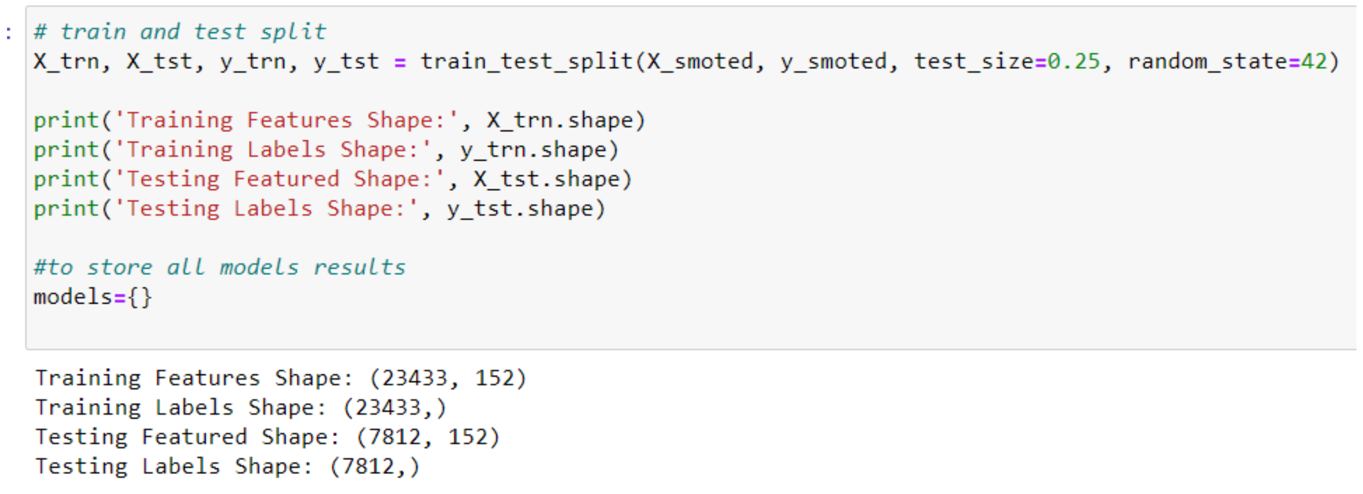


Fig 9: Target variable after SMOTE

Splitting Data Into Training And Test Data:

Dataset was split into training and testing using “train_test_split” from “sklearn.model_selection”.

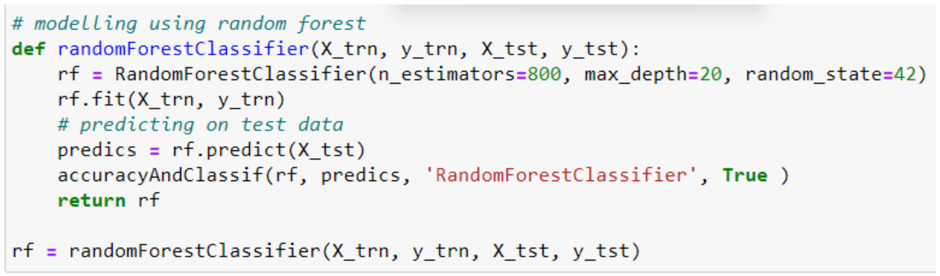


```
# train and test split
X_trn, X_tst, y_trn, y_tst = train_test_split(X_smoted, y_smoted, test_size=0.25, random_state=42)

print('Training Features Shape:', X_trn.shape)
print('Training Labels Shape:', y_trn.shape)
print('Testing Featured Shape:', X_tst.shape)
print('Testing Labels Shape:', y_tst.shape)

#to store all models results
models={}
```

```
Training Features Shape: (23433, 152)
Training Labels Shape: (23433,)
Testing Featured Shape: (7812, 152)
Testing Labels Shape: (7812,)
```

Random Forest Classifier For Feature Selection:

Random Forest Classifier model was run on training set and using its feature importance functionality feature list with their importance score was extracted.



```
# modelling using random forest
def randomForestClassifier(X_trn, y_trn, X_tst, y_tst):
    rf = RandomForestClassifier(n_estimators=800, max_depth=20, random_state=42)
    rf.fit(X_trn, y_trn)
    # predicting on test data
    predics = rf.predict(X_tst)
    accuracyAndClassif(rf, predics, 'RandomForestClassifier', True )
    return rf

rf = randomForestClassifier(X_trn, y_trn, X_tst, y_tst)
```

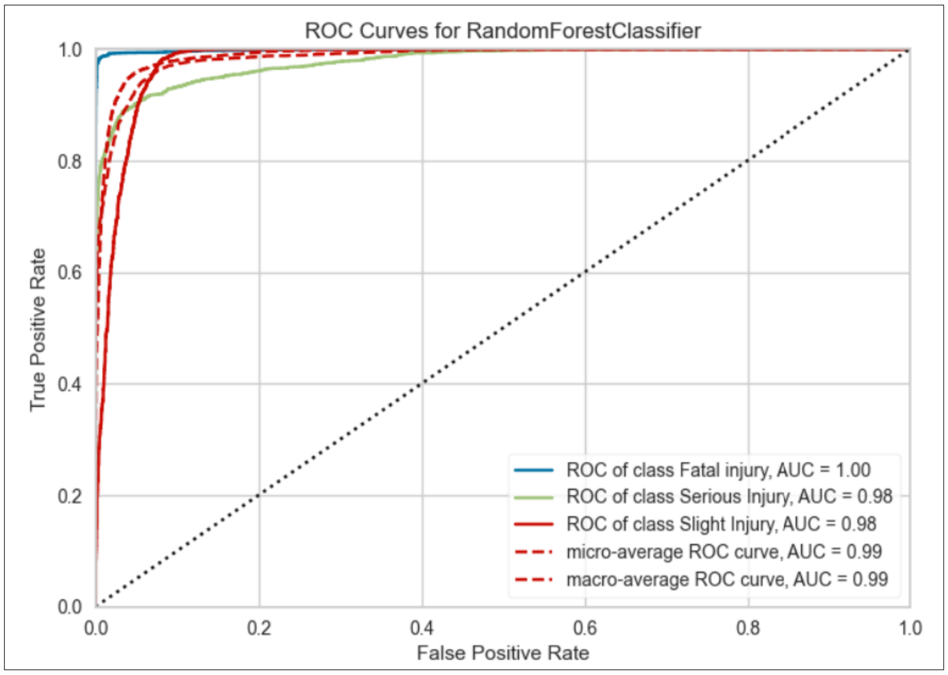


Fig 10: ROC curves for Random Forest Classifier Feature Selection

Ordering Feature Importance:

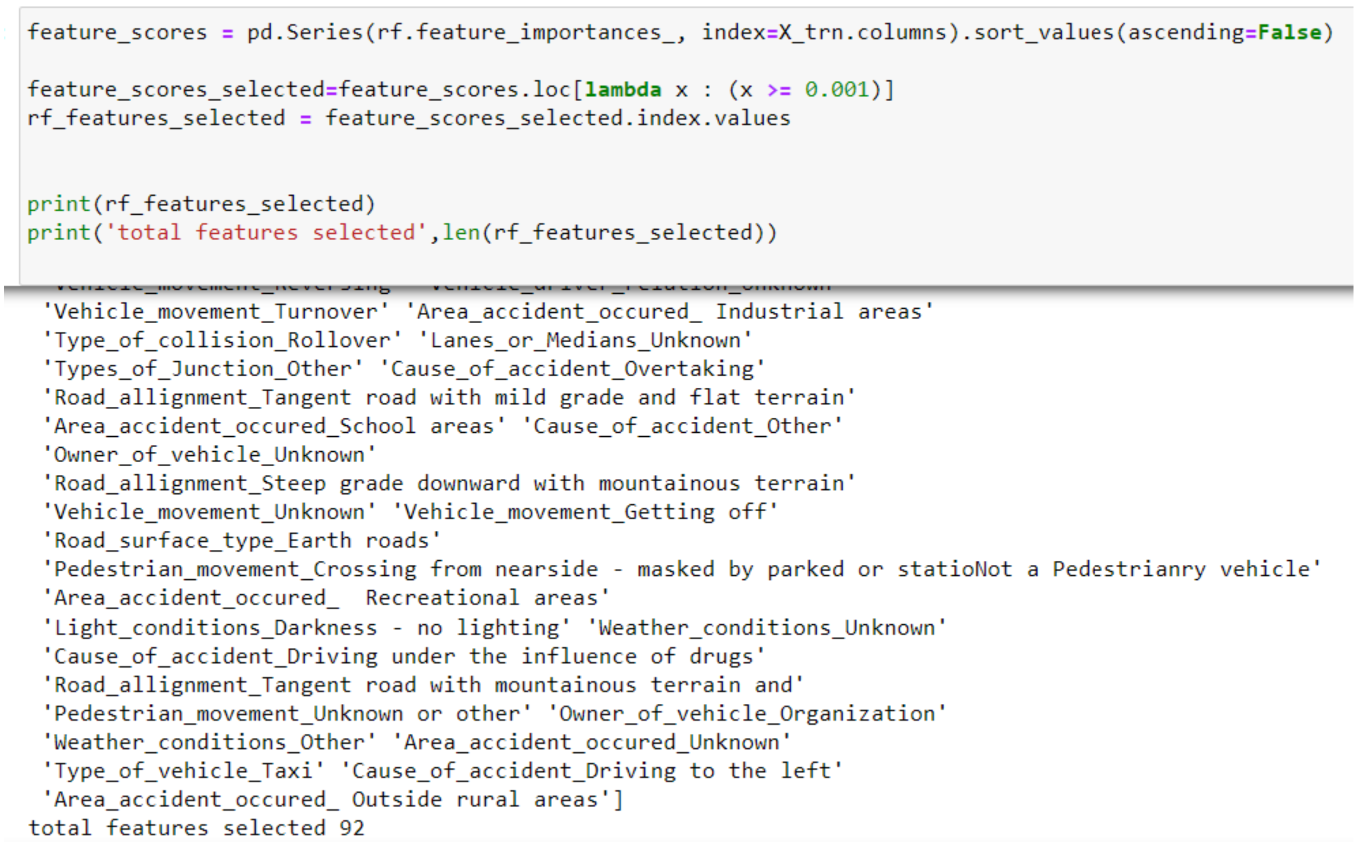
```python
importances = list(rf.feature_importances_)

# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 3)) for feature, importance in zip(feature_list, importances)]

# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
# Print out the feature and importances
[print('Feature: {:20} Importance: {}'.format(*pair)) for pair in feature_importances]
```

```
Feature: Hour_of_Day          Importance: 0.073
Feature: Number_of_casualties Importance: 0.07
Feature: Number_of_vehicles_involved Importance: 0.053
Feature: Lanes_or_Medians_Two-way (divided with broken lines road marking) Importance: 0.023
Feature: Light_conditions_Daylight Importance: 0.023
Feature: Driving_experience_5-10yr Importance: 0.021
Feature: Vehicle_movement_Going straight Importance: 0.021
Feature: Casualty_severity_Unknown Importance: 0.02
Feature: Educational_level_Junior high school Importance: 0.019
Feature: Type_of_vehicle_Lorry Importance: 0.019
Feature: Area_accident_occured_Office areas Importance: 0.019
Feature: Area_accident_occured_Other Importance: 0.019
Feature: Type_of_collision_Vehicle with vehicle collision Importance: 0.019
Feature: Age_band_of_driver_Unknown Importance: 0.018
Feature: Casualty_severity_3  Importance: 0.018
Feature: Cause_of_accident_No distancing Importance: 0.018
Feature: Age_band_of_driver_31-50 Importance: 0.017
Feature: Types_of_Junction_Y Shape Importance: 0.016
Feature: Driving_experience_Above 10yr Importance: 0.015
Feature: Lanes_or_Medians_Undivided Two way Importance: 0.015
Feature: Types_of_Junction_No junction Importance: 0.015
Feature: Road_surface_conditions_Wet or damp Importance: 0.015
Feature: Driving_experience_Below 1yr Importance: 0.014
```

```
Feature: Weather_conditions_Raining and Windy Importance: 0.0
Feature: Weather_conditions_Snow Importance: 0.0
Feature: Weather_conditions_Windy Importance: 0.0
Feature: Type_of_collision_Collision with roadside-parked vehicles Importance: 0.0
Feature: Type_of_collision_Fall from vehicles Importance: 0.0
Feature: Type_of_collision_Other Importance: 0.0
Feature: Type_of_collision_With Train Importance: 0.0
Feature: Vehicle_movement_Overtaking Importance: 0.0
Feature: Vehicle_movement_Parked Importance: 0.0
Feature: Vehicle_movement_Stopping Importance: 0.0
Feature: Vehicle_movement_U-Turn Importance: 0.0
Feature: Vehicle_movement_Waiting to go Importance: 0.0
```

Based on the feature scores we could see that many features had a score of 0.0. All the features having a score of 0.0 were eliminated and feature set was updated with only features having feature score greater than 0.

Selecting Features Based On Random Forest Classifier Importance:



```
feature_scores = pd.Series(rf.feature_importances_, index=X_trn.columns).sort_values(ascending=False)

feature_scores_selected=feature_scores.loc[lambda x : (x >= 0.001)]
rf_features_selected = feature_scores_selected.index.values


print(rf_features_selected)
print('total features selected',len(rf_features_selected))
```

```
 'Vehicle_movement_Turnover' 'Area_accident_occured_ Industrial areas'
 'Type_of_collision_Rollover' 'Lanes_or_Medians_Unknown'
 'Types_of_Junction_Other' 'Cause_of_accident_Overtaking'
 'Road_allignment_Tangent road with mild grade and flat terrain'
 'Area_accident_occured_School areas' 'Cause_of_accident_Other'
 'Owner_of_vehicle_Unknown'
 'Road_allignment_Steep grade downward with mountainous terrain'
 'Vehicle_movement_Unknown' 'Vehicle_movement_Getting off'
 'Road_surface_type_Earth roads'
 'Pedestrian_movement_Crossing from nearside - masked by parked or statioNot a Pedestrianry vehicle'
 'Area_accident_occured_  Recreational areas'
 'Light_conditions_Darkness - no lighting' 'Weather_conditions_Unknown'
 'Cause_of_accident_Driving under the influence of drugs'
 'Road_allignment_Tangent road with mountainous terrain and'
 'Pedestrian_movement_Unknown or other' 'Owner_of_vehicle_Organization'
 'Weather_conditions_Other' 'Area_accident_occured_Unknown'
 'Type_of_vehicle_Taxi' 'Cause_of_accident_Driving to the left'
 'Area_accident_occured_ Outside rural areas']
total features selected 92
```

After eliminating features with score 0.0, we selected 92 features for further process.

Section Summary:

This completed our feature selection process and finally we selected 92 features from a total of 152 features.

Model Development and Evaluation

1. Random Forest Classifier:

The Random Forest Classifier is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputs the class that is the mode of the classes (classification) of the individual trees.

We ran random forest classifier again on our updated feature dataset with predictions and overall classification scores.

```
#updating training and test data with only new features
X_trn = X_trn[rf_features_selected]
X_tst = X_tst[rf_features_selected]
randomForestClassifier(X_trn, y_trn, X_tst, y_tst)
```

```
train score :  0.9958178636964964
f1 score for RandomForestClassifier:  0.9357757329748744
classification for RandomForestClassifier
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2599
           1       0.94      0.87      0.91      2639
           2       0.88      0.95      0.91      2574

    accuracy                           0.94      7812
   macro avg       0.94      0.94      0.94      7812
weighted avg       0.94      0.94      0.94      7812
```

We see that in the second run also we were able to achieve accuracy of 94% even after reducing features from 152 to 92.

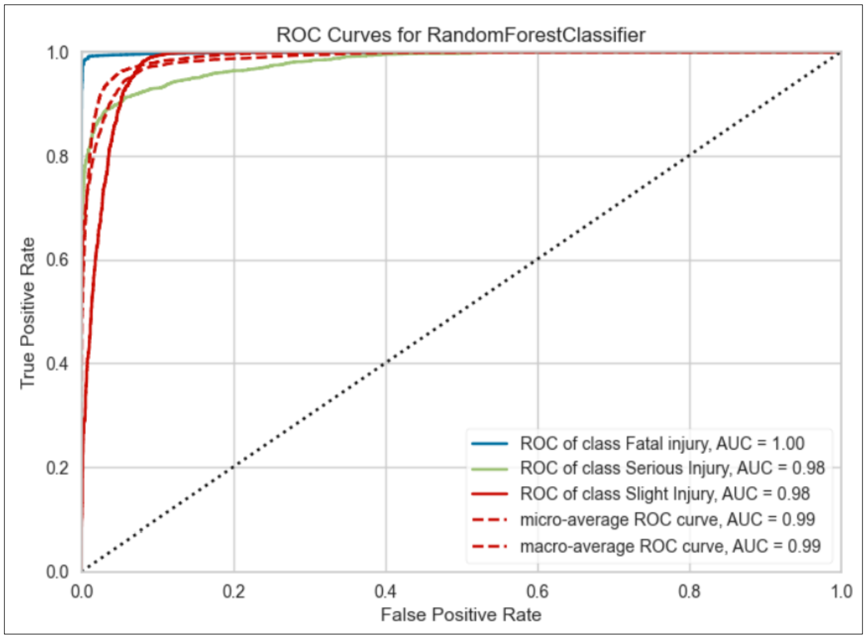


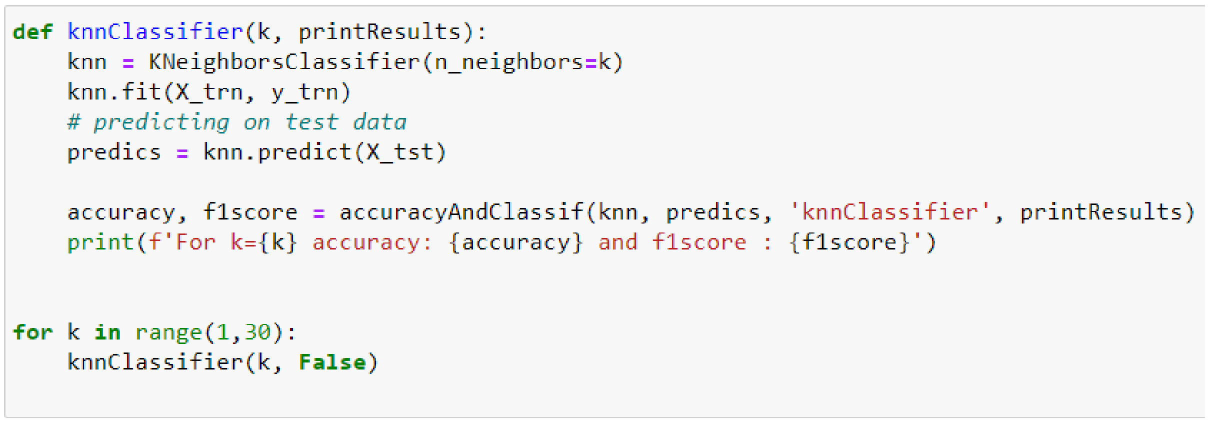
Fig 11: ROC curves for Random Forest Classifier Model

The ROC curves for all target classes in random forest are closer to the top left indicating good performance with good AUC scores.

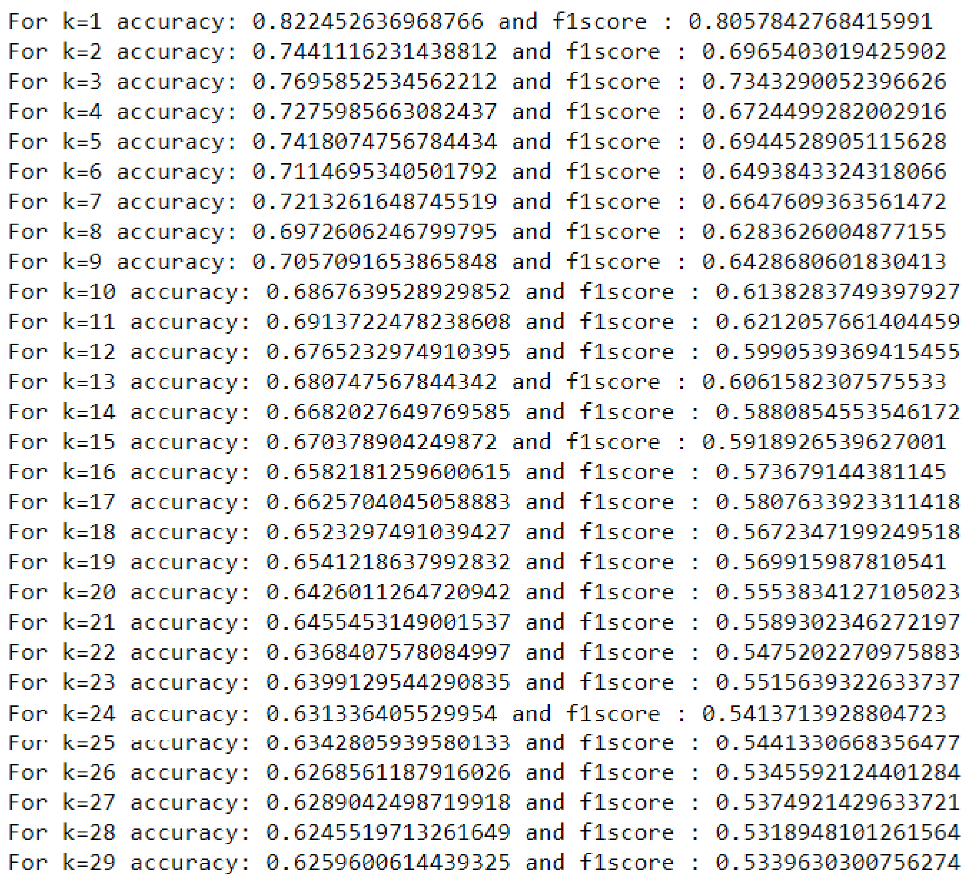
2. KNN Classifier:

The k-Nearest Neighbours (KNN) algorithm is a simple classification or regression algorithm. For classification given a data point, the algorithm looks at the k-nearest neighbours in the feature space. The class most common among these neighbours is assigned to the data point. The parameter k represents the number of neighbours to consider.

We used “KNeighborsClassifier” from “sklearn.neighbors” and ran it initially for values of k starting from 1 to 29.



```python
def knnClassifier(k, printResults):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_trn, y_trn)
    # predicting on test data
    predics = knn.predict(X_tst)

    accuracy, f1score = accuracyAndClassif(knn, predics, 'knnClassifier', printResults)
    print(f'For k={k} accuracy: {accuracy} and f1score : {f1score}')


for k in range(1,30):
    knnClassifier(k, False)
```



```
For k=1 accuracy: 0.822452636968766 and f1score : 0.8057842768415991
For k=2 accuracy: 0.7441116231438812 and f1score : 0.6965403019425902
For k=3 accuracy: 0.7695852534562212 and f1score : 0.7343290052396626
For k=4 accuracy: 0.7275985663082437 and f1score : 0.6724499282002916
For k=5 accuracy: 0.7418074756784434 and f1score : 0.6944528905115628
For k=6 accuracy: 0.7114695340501792 and f1score : 0.6493843324318066
For k=7 accuracy: 0.7213261648745519 and f1score : 0.6647609363561472
For k=8 accuracy: 0.6972606246799795 and f1score : 0.6283626004877155
For k=9 accuracy: 0.7057091653865848 and f1score : 0.6428680601830413
For k=10 accuracy: 0.6867639528929852 and f1score : 0.6138283749397927
For k=11 accuracy: 0.6913722478238608 and f1score : 0.6212057661404459
For k=12 accuracy: 0.6765232974910395 and f1score : 0.5990539369415455
For k=13 accuracy: 0.680747567844342 and f1score : 0.6061582307575533
For k=14 accuracy: 0.6682027649769585 and f1score : 0.5880854553546172
For k=15 accuracy: 0.670378904249872 and f1score : 0.5918926539627001
For k=16 accuracy: 0.6582181259600615 and f1score : 0.573679144381145
For k=17 accuracy: 0.6625704045058883 and f1score : 0.5807633923311418
For k=18 accuracy: 0.6523297491039427 and f1score : 0.5672347199249518
For k=19 accuracy: 0.6541218637992832 and f1score : 0.569915987810541
For k=20 accuracy: 0.6426011264720942 and f1score : 0.5553834127105023
For k=21 accuracy: 0.6455453149001537 and f1score : 0.5589302346272197
For k=22 accuracy: 0.6368407578084997 and f1score : 0.5475202270975883
For k=23 accuracy: 0.6399129544290835 and f1score : 0.5515639322633737
For k=24 accuracy: 0.631336405529954 and f1score : 0.5413713928804723
For k=25 accuracy: 0.6342805939580133 and f1score : 0.5441330668356477
For k=26 accuracy: 0.6268561187916026 and f1score : 0.5345592124401284
For k=27 accuracy: 0.6289042498719918 and f1score : 0.5374921429633721
For k=28 accuracy: 0.6245519713261649 and f1score : 0.5318948101261564
For k=29 accuracy: 0.6259600614439325 and f1score : 0.5339630300756274
```

After evaluating the values we see that knn gives the best accuracy for k=1, so we ran it again for k=1 and extract its classification information.

```
#from above we run for k=1 and see detailed results
knnClassifier(1, True)
```

```
train score :  1.0
f1 score for knnClassifier:  0.8057842768415991
classification for knnClassifier
              precision    recall  f1-score   support

           0       0.92      1.00      0.96      2599
           1       0.69      0.98      0.81      2639
           2       0.99      0.48      0.64      2574

    accuracy                           0.82      7812
   macro avg       0.87      0.82      0.81      7812
weighted avg       0.87      0.82      0.81      7812

For k=1 accuracy: 0.822452636968766 and f1score : 0.8057842768415991
```

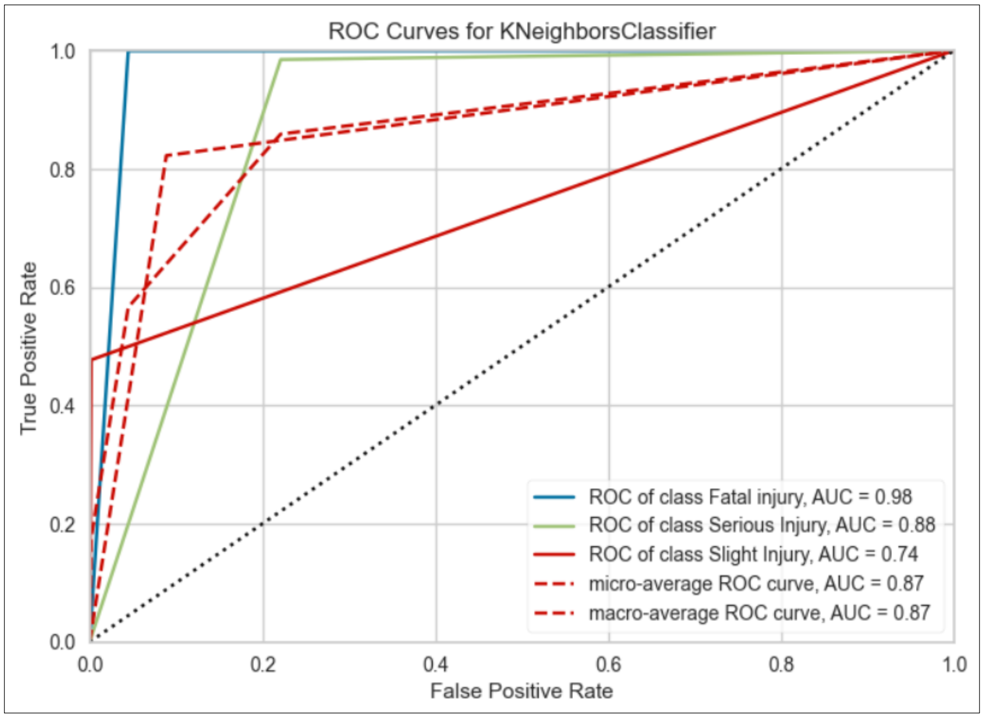


Fig 12: ROC curves for KNN Classifier Model

ROC curves for K neighbours classifier is good for fatal injury and serious injury class but does not show good performance for Slight injury class prediction.

3. ADA Boost Classifier:

AdaBoost, short for Adaptive Boosting, is an ensemble learning method that is used for both classification and regression tasks. AdaBoost focuses on improving the accuracy of weak learners (classifiers or regressors) by combining them into a strong learner.

We used “AdaBoostClassifier” from “sklearn.ensemble” package and ran it on training sets with predictions.

```python
def adaBoostClassifier():
    ada = AdaBoostClassifier(n_estimators=800, learning_rate=1, random_state=100)
    ada.fit(X_trn, y_trn)

    # predicting on test data
    predics = ada.predict(X_tst)

    accuracyAndClassif(ada, predics, 'AdaBoostClassifier', True)

adaBoostClassifier()
```

```
train score :  0.7683181837579482
f1 score for AdaBoostClassifier:  0.762212020518482
classification for AdaBoostClassifier
              precision    recall  f1-score   support

           0       0.80      0.75      0.77      2599
           1       0.66      0.67      0.66      2639
           2       0.83      0.87      0.85      2574

    accuracy                           0.76      7812
   macro avg       0.76      0.76      0.76      7812
weighted avg       0.76      0.76      0.76      7812
```

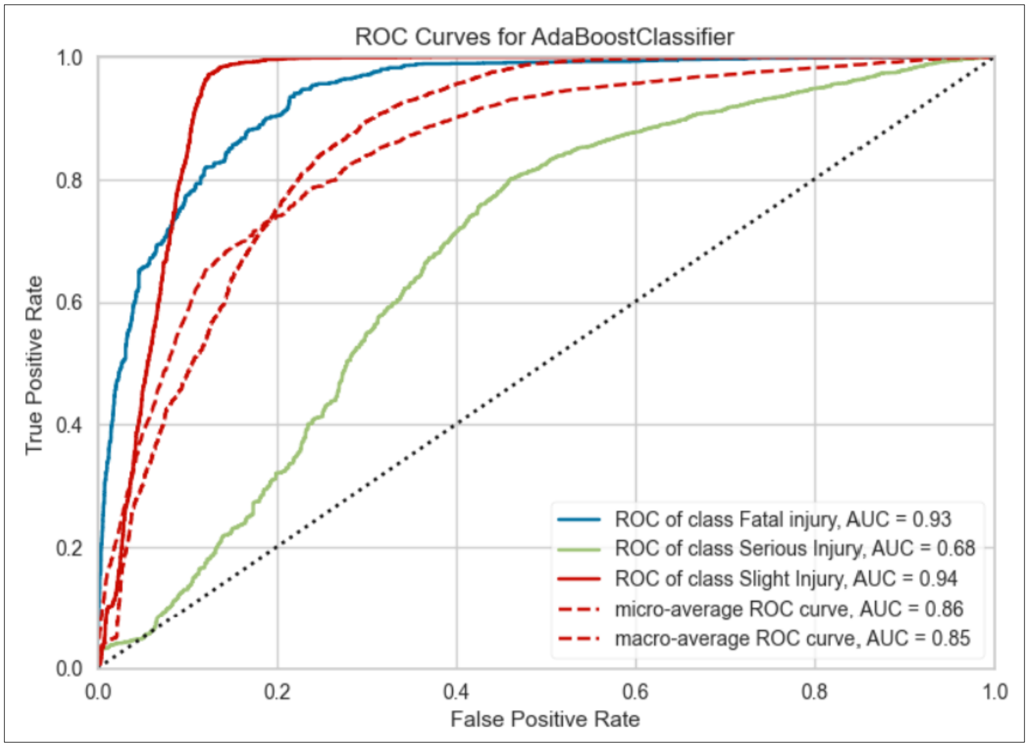


Fig 13: ROC curves for ADA Boost Classifier Model

ROC curves for ADA boost classifier show overall less deviance to the left showing overall less performance with exception for slight injury predictions.

4. Gradient Boosting Classifier:

Gradient Boosting is another popular ensemble learning method, like AdaBoost, used for both classification and regression tasks. It builds a series of weak learners (typically decision trees) sequentially, with each one trying to correct the errors of the previous one.

We used “AdaBoostClassifier” from “sklearn.ensemble” package and ran it on training sets with predictions.

```python
def gradientBoostingClassifier():
    gbc = GradientBoostingClassifier(n_estimators=1000, learning_rate=1, random_state=100)
    gbc.fit(X_trn, y_trn)
    # predicting on test data
    predics = gbc.predict(X_tst)

    accuracyAndClassif(gbc, predics, 'GradientBoostingClassifier', True)

gradientBoostingClassifier()
```

```
train score :  0.9617206503648701
f1 score for GradientBoostingClassifier:  0.902386431194146
classification for GradientBoostingClassifier
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      2599
           1       0.88      0.84      0.86      2639
           2       0.85      0.88      0.86      2574

    accuracy                           0.90      7812
   macro avg       0.90      0.90      0.90      7812
weighted avg       0.90      0.90      0.90      7812
```

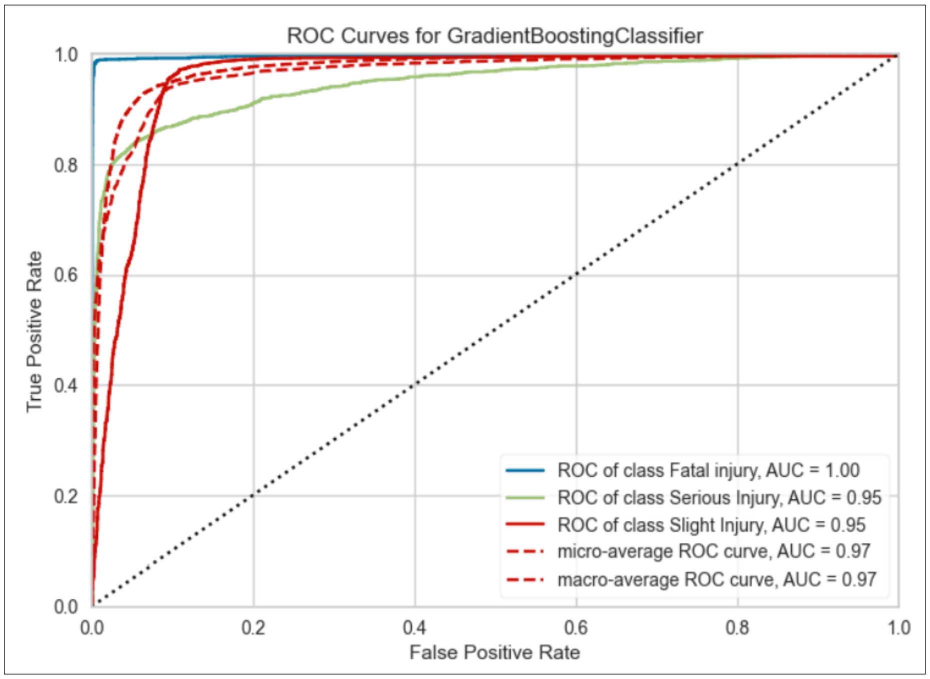


Fig 14: ROC curves for Gradient Boosting Classifier Model

The ROC curves for all target classes in gradient boosting are closer to the top left indicating good performance with good AUC scores.

5. Support Vector Machines:

Support Vector Machines (SVM) is a powerful supervised machine learning algorithm used for classification and regression tasks. SVM aims to find the hyperplane that best separates different classes in the feature space.

We used “SVC” with default parameters from “sklearn. svm” package and ran it on training sets with predictions.

```
def SVM():
    svc=SVC()
    svc.fit(X_trn, y_trn)
    # predicting on test data
    predics = svc.predict(X_tst)

    accuracyAndClassif(svc, predics, 'SupportVectorMachine', True)

SVM()
```

```
train score :  0.829343233900909
f1 score for SupportVectorMachine:  0.8196935725734678
classification for SupportVectorMachine
              precision    recall  f1-score   support

           0       0.82      0.88      0.85      2599
           1       0.83      0.63      0.72      2639
           2       0.83      0.97      0.89      2574

    accuracy                           0.83      7812
   macro avg       0.83      0.83      0.82      7812
weighted avg       0.83      0.83      0.82      7812
```

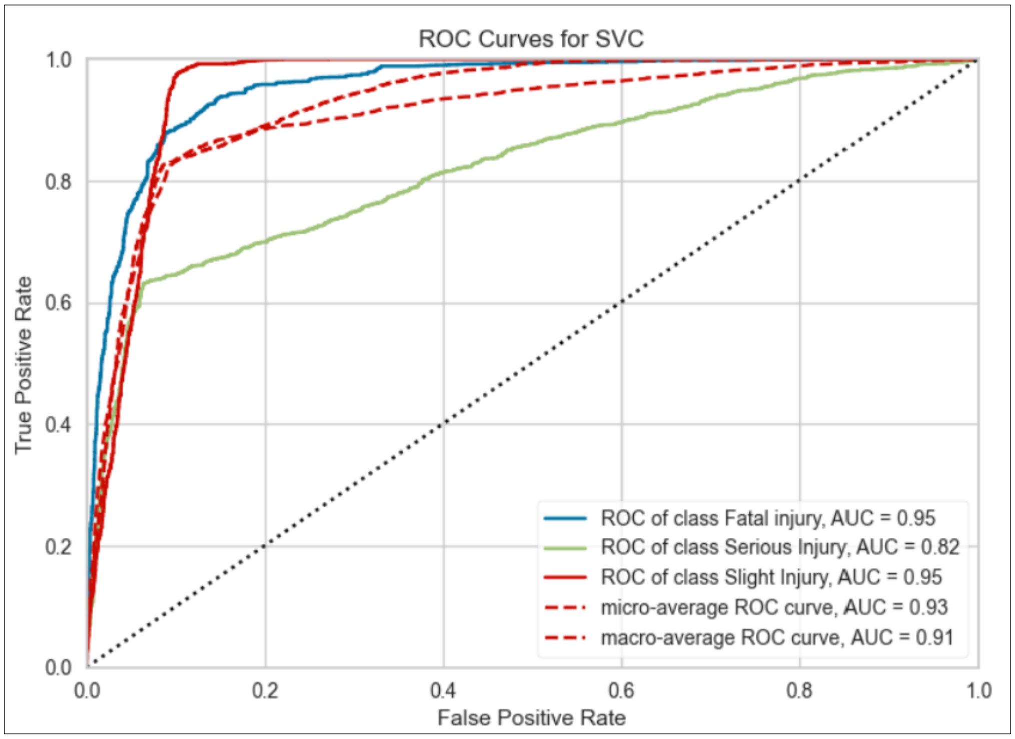


Fig 15: ROC curves for Support Vector Machines Model

The ROC curves for all target classes in SVC are closer to the top left indicating good performance with good AUC scores but less than gradient boosting and random forest classifier.

6. Decision Tree Classifier:

Decision Tree is a Supervised Machine Learning Algorithm that uses a set of rules to make decisions. It creates the classification model by building a decision tree. Each node in the tree specifies a test on an attribute, each branch descending from that node corresponds to one of the possible values for that attribute. Gini impurity and entropy are two commonly used measures to evaluate the quality of a split at each node.

Gini impurity measures the probability of incorrectly classifying a randomly chosen element in the dataset.

Entropy is a measure of disorder or uncertainty in a set of data points.

We used “DecisionTreeClassifier” with default parameters from “sklearn.tree” package and ran it on training sets with predictions with both gini and entropy measures.

```python
def DecisionTreeClassifierModel(X_tst):
    dtc_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100, max_depth=20, min_samples_leaf=5)
    dtc_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100, max_depth = 20, min_samples_leaf = 5)

    dtc_gini.fit(X_trn, y_trn)
    dtc_entropy.fit(X_trn, y_trn)
    # predicting on test data
    predics_gini = dtc_gini.predict(X_tst)
    predics_entropy = dtc_entropy.predict(X_tst)
    accuracyAndClassif(dtc_gini, predics_gini, 'decision-tree-gini', True)
    accuracyAndClassif(dtc_entropy, predics_entropy, 'decision-tree-entropy', True)

DecisionTreeClassifierModel(X_tst)
```

A. Gini:

```
train score :  0.9092305722698758
f1 score for decision-tree-gini:  0.834416857086885
classification for decision-tree-gini
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      2599
           1       0.77      0.78      0.78      2639
           2       0.81      0.74      0.77      2574

    accuracy                           0.84      7812
   macro avg       0.83      0.84      0.83      7812
weighted avg       0.83      0.84      0.83      7812
```

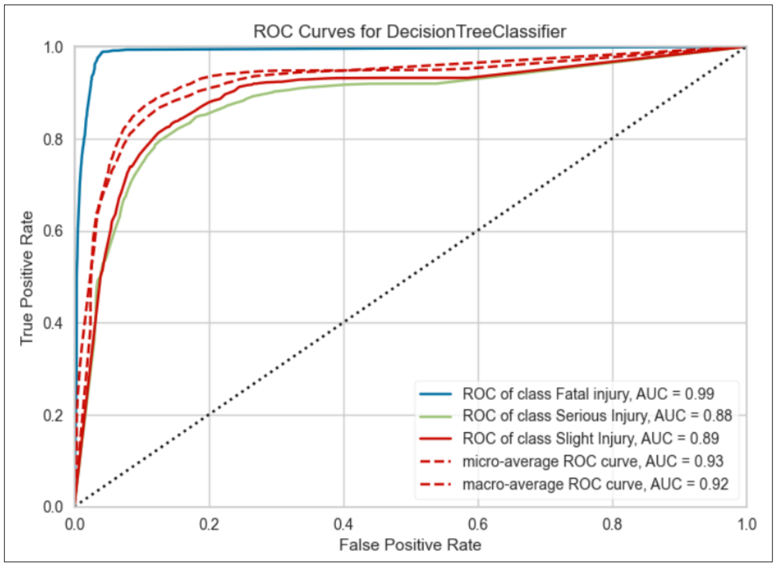


Fig 16: ROC curves for Decision Tree Classifier (Gini) Model

B. Entropy:

```
train score :  0.9065420560747663
f1 score for decision-tree-entropy:  0.8337333246507095
classification for decision-tree-entropy
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      2599
           1       0.77      0.77      0.77      2639
           2       0.81      0.75      0.78      2574

    accuracy                           0.84      7812
   macro avg       0.83      0.84      0.83      7812
weighted avg       0.83      0.84      0.83      7812
```

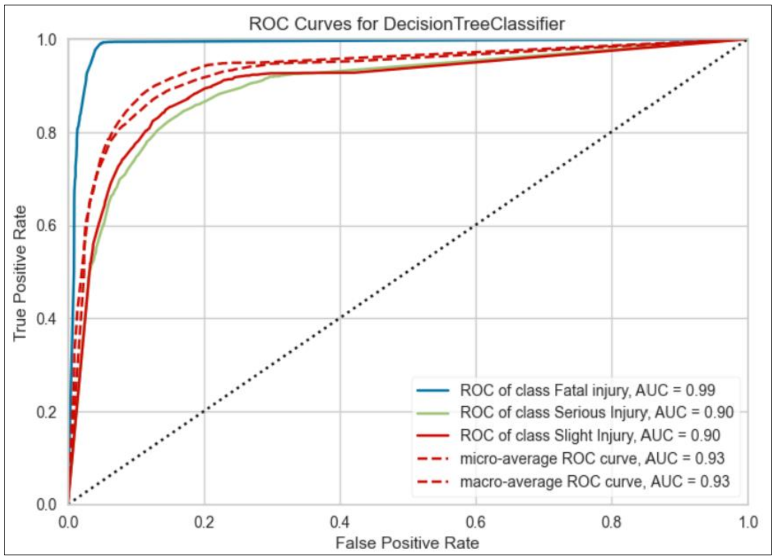


Fig 17: ROC curves for Decision Tree Classifier (Entropy) Model

The ROC curves in decision trees is closest to the left for Fatal injury than others. Overall it shows good performance for all classes with good AUC scores.

Comparing All Models:

After running all the models and we compared all with each other with final accuracy and f1 scores.

```
alg=[]
acc=[]
f1scores=[]
for key in models:
    alg.append(key)
    acc.append(models[key]['accuracy'])
    f1scores.append(models[key]['f1score'])

Accuracy_Scores=pd.DataFrame({'Algorithms':alg, 'Accuracy': acc, 'f1scores': f1scores})
Accuracy_Scores.sort_values(by='Accuracy',ascending=False)
```

	Algorithms	Accuracy	f1scores
0	RandomForestClassifier	0.935868	0.935776
3	GradientBoostingClassifier	0.902586	0.902386
5	decision-tree-gini	0.836150	0.834417
6	decision-tree-entropy	0.836150	0.833733
4	SupportVectorMachine	0.826293	0.819694
1	knnClassifier	0.822453	0.805784
2	AdaBoostClassifier	0.762673	0.762212

From the results we see that random forest classifier and gradient boosting classifier performed well than others with scores more than 90% and random forest classifier being the best model with accuracy and f1 score of approx. 94%(93.5%).

Cross Validation:

Cross-validation is a resampling technique used in machine learning to assess the performance of a predictive model. It helps to provide a more robust estimate of a model's performance by partitioning the dataset into multiple subsets and evaluating the model multiple times.

As a last step we ran cross validation on random forest to avoid any case of overfitting and verify the results.

```
#cross validation for randomforestclassifier
# cv indicates folds in startifiedKfolds technique
## checking for overfiting
score = cross_val_score(RandomForestClassifier(n_estimators=800, max_depth=20, random_state=42),
                        X_smoted, y_smoted,scoring='accuracy', cv=10)
print(f'score are {score}')
print(f'score mean: {score.mean()}')
```

```
score are [0.70944    0.83296    0.97056    0.976      0.98272    0.98207426
 0.97823303 0.97023047 0.9737516  0.97151088]
score mean: 0.9347480256081946
```

Conclusion

We saw that for most folds, the accuracy is above 90% with mean accuracy equal to 93.4% which is very similar to 93.5% we got above. Thus, we conclude that **random forest classifier is the best model to predict accident severity for the given dataset.**

References

- GeeksforGeeks. (2023, November 15). Machine Learning Algorithms. Geeks for Geeks. https://www.geeksforgeeks.org/machine-learning-algorithms/

- Huong Ngo. (2022, July 30). How to Clean Your Data in Python. Towards Data Science. https://towardsdatascience.com/how-to-clean-your-data-in-python-8f178638b98d

- Shahane, S. (2022, January 26). Road Traffic Accidents. Kaggle. https://www.kaggle.com/datasets/saurabhshahane/road-traffic-accidents