CS 5805: Machine Learning 1

Author: Jay Sarode

Instructor: Dr. Reza Jafari

Date: 12/8/2023

Table of Contents

Table of Figures

Table of Tables

Abstract

This project delves into the realm of machine learning (ML) applied to traffic accidents, focusing on identifying relationships between accidents, vehicles involved, and resulting casualties. The primary goal is to harness ML techniques to enhance road safety by predicting and preventing future accidents.

By utilizing diverse data sources, including traffic records, vehicle details, and casualty information, the project begins with the aggregation of a comprehensive dataset. This dataset undergoes meticulous preprocessing to ensure data quality and relevance.

This feature engineering phase involves identifying critical features from the dataset that significantly impact accident outcomes. Techniques such as principal component analysis (PCA) and variance inflation factor analysis are employed to refine the feature set, ensuring a balance between model complexity and predictive power.

Using the engineered features, several classification models are developed to predict accident severity. Models such as Decision Tree, Support Vector Machines (SVM), and Neural Networks are explored and evaluated based on their accuracy, precision, and recall.

Clustering algorithms like K-Means are applied to uncover inherent groupings within the data. This analysis aims to identify patterns in accident causes, vehicle types involved, and casualty characteristics, providing insights into common accident scenarios.

This technique is utilized to discover associations between different variables in the dataset. For example, it might reveal how specific vehicle types, road conditions, and driver behaviors are linked to severe accidents.

The insights gained from classification, clustering, and association rule mining inform various stakeholders, including policymakers, urban planners, and vehicle manufacturers. These insights contribute to developing targeted strategies for accident prevention, such as improving road infrastructure, enacting relevant traffic laws, and enhancing vehicle safety features.

This project emphasizes the potential of machine learning in understanding and mitigating traffic accidents. The combination of classification, clustering, and association rule mining provides a multifaceted view of the factors contributing to road accidents, paving the way for informed and effective interventions to enhance road safety.

Introduction

The Final Term Project (FTP) is an initiative designed to provide hands-on experience in the application of machine learning techniques, with a focus on feature engineering, classification, clustering, and association rule mining. The core objective is to apply the theoretical knowledge gained during the course to a real-world dataset. This project aims to deepen understanding of machine learning methodologies and concepts.

Procedures:

The project follows a structured approach, outlined as follows:

1. **Data Selection and Preprocessing**: The first step involves selecting a relevant dataset that contains comprehensive information on traffic accidents, vehicles involved, and casualties. The preprocessing stage will include data cleaning, normalization, and transformation to ensure the dataset is suitable for machine learning applications.
2. **Feature Engineering**: This critical phase focuses on identifying and creating relevant features from the raw data. Techniques such as feature extraction and selection will be employed to enhance the dataset's predictive power for the subsequent machine learning models.
3. **Application of Machine Learning Algorithms**:
    - **Classification**: Various classification algorithms will be applied to predict specific outcomes, such as the severity of accidents. The performance of different classifiers like Decision Trees, Random Forest, SVM, and Neural Networks will be compared to determine the most effective model.
    - **Clustering**: To discover natural groupings within the data, clustering algorithms like K-Means and other clustering techniques will be used. This analysis aims to identify common patterns and relationships within the dataset.
    - **Association Rule Mining**: This technique will be employed to uncover interesting associations and correlations between different variables in the dataset, providing deeper insights into the factors contributing to traffic accidents.
    - **Regression**: This technique will be used to predict a continuous numerical feature, such as chances of death of a person, if an accident happened, which will be calculated using Ordinary-Least Squares Regression.

4. **Evaluation and Model Selection**: The models will be rigorously evaluated based on various performance metrics such as accuracy, precision, recall, and F1-score. The model demonstrating the highest performance in terms of these metrics will be recommended for classifying the dataset.
5. **Insights and Recommendations**: The project will culminate with a set of insights derived from the analysis and specific recommendations for the selected classifier. These insights will be valuable for stakeholders involved in road safety and traffic management.

Description of the Dataset

The dataset under consideration provides a comprehensive overview of personal injury road accidents in Great Britain (GB) from 2005 to 2014. It is a rich repository of information, encapsulating various dimensions of road safety and accident details. This dataset is structured into four main files: the Accident file, the Vehicle file, the Casualty file, and the Lookup file. Each of these files offers unique insights into different aspects of road accidents.

Accident File

- *Contents*: This primary dataset encompasses extensive details about each accident. It includes data on accident severity, weather conditions, geographic location, date and time, day of the week, and road types.
- *Applications*: This file is crucial for understanding the environmental and temporal factors contributing to road accidents, assisting in identifying high-risk factors and areas.

Vehicle File

- *Contents*: This file delves into specifics about the vehicles involved in the accidents. It covers information on the vehicle type, model, engine size, as well as the age and sex of the driver, and the age of the car.
- *Applications*: Analysis of this data can unveil patterns in vehicle-related factors in accidents, aiding in vehicle safety design and targeted driver safety programs.

Casualty File

1. *Contents*: Focused on individuals affected by the accidents, this file records casualty severity, age, sex, social class, and whether the individual was a pedestrian, driver, or passenger.
2. *Applications*: This file is essential for understanding the human impact of road accidents, informing healthcare and emergency response strategies.

Lookup File

- *Contents*: Serving as a key to interpret the datasets, this file provides textual descriptions of all the variable codes used in the three main files, ensuring clarity and ease of understanding for users.
- *Applications*: It is vital for accurate data interpretation, making the dataset accessible to a wider range of users, including those without a technical background.

The files on accidents, vehicles and casualties are merged together for highly rigorous, combined analysis.

Dependent Variable: The dataset's dependent variable is typically the accident severity, or casualty severity. There might be some features that can be derived from the base features in the dataset.

The dataset on road safety from 2005 to 2014 is of significant value across various industries. Transport authorities can leverage it to refine road safety measures, develop efficient traffic management systems, and design safer roadways. For the automobile industry, data is pivotal in enhancing vehicle safety features and constructing cars that better mitigate injury risks in accidents. Insurance companies benefit from this dataset by gaining insights for more accurate risk assessments, policy pricing, and understanding patterns in accident occurrences. In the healthcare sector, the data aids in formulating improved emergency response strategies and allocating resources effectively in high-risk areas. Lastly, policy makers and researchers can utilize this rich dataset for creating more focused road safety policies and conducting comprehensive safety research, thereby contributing to overall road safety improvements.

Pre-processing of the Dataset

Each of the three files are loaded, i.e., the accidents file, the vehicles file and the casualty file.

Accident File:

```
(1640597, 32)
  Accident_Index  ...  LSOA_of_Accident_Location
0  200501BS00001  ...                 E01002849
1  200501BS00002  ...                 E01002909
2  200501BS00003  ...                 E01002857
3  200501BS00004  ...                 E01002840
4  200501BS00005  ...                 E01002863
```

Vehicle File:

```
(3004425, 22)
  Accident_Index  Vehicle_Reference  ...  Driver_IMD_Decile  Driver_Home_Area_Type
0  200501BS00001                 1  ...                  7                     1
1  200501BS00002                 1  ...                 -1                    -1
2  200501BS00003                 1  ...                  2                     1
3  200501BS00003                 2  ...                  1                     1
4  200501BS00004                 1  ...                  2                     1
```

Casualty File:

```
(3004425, 53)
  Accident_Index  ...  Driver_Home_Area_Type
0  200501BS00001  ...                     1
1  200501BS00002  ...                    -1
2  200501BS00003  ...                     1
3  200501BS00003  ...                     1
4  200501BS00004  ...                     1
```

These files are merged together based on the common ‘Accident_Index’ feature, and the join is done through inner join.

Final Dataset:

```
Final Dataset:
(4287593, 67)
  Accident_Index  Location_Easting_OSGR  ...  Casualty_Type  Casualty_Home_Area_Type
0  200501BS00001               525680.0  ...              0                        1
1  200501BS00002               524170.0  ...             11                        1
2  200501BS00003               524520.0  ...              9                        1
3  200501BS00003               524520.0  ...              9                        1
4  200501BS00004               526900.0  ...              0                        1
```

The final dataset shape is: 4287593 samples and 67 features.

There are some categorical columns, which are already converted to numerical. But for our exploratory data analysis, the features need to be mapped back with a help of a lookup file which has the labels and value pairs of features.

Through the fuzzy matching we map the sheet name with the exact columns name, having the feature similarity score of greater than 80.

A sample can be shown here:

```
                          Actual Column Name  ... Similarity Score
0                                Police_Force  ...               92
1                            Accident_Severity  ...               94
2                                 Day_of_Week  ...               82
3                  Local_Authority_(District)  ...               92
4                   Local_Authority_(Highway)  ...               92
```

The new dataset is saved into the new csv file, which is loaded into the new python file.

In the new python file, the data loaded is loaded into chunks so that the data loads quickly and efficiently.

Here is the gist of the how the data was loaded:

```
Number of chunks read:  (200000, 67)
Number of chunks read:  (200000, 67)
Number of chunks read:  (200000, 67)
Number of chunks read:  (200000, 67)
Number of chunks read:  (200000, 67)
Number of chunks read:  (200000, 67)
```

```
Time taken to read the data:  50.28195023536682
```

	Accident_Index	Location_Easting_OSGR	...	Casualty_Type	Casualty_Home_Area_Type
0	200501BS00001	525680.0	...	0	1
1	200501BS00002	524170.0	...	11	1
2	200501BS00003	524520.0	...	9	1
3	200501BS00003	524520.0	...	9	1
4	200501BS00004	526900.0	...	0	1

The data which has been loaded, was checked for null values. The dataset contained null values, which were dropped.

There are some the columns that have multiple special characters in it. These special characters were replaced by more meaningful characters.

After the data preprocessing, the data is prepared for exploratory data analysis.

Phase 1: Feature Engineering & EDA

Line Plots:

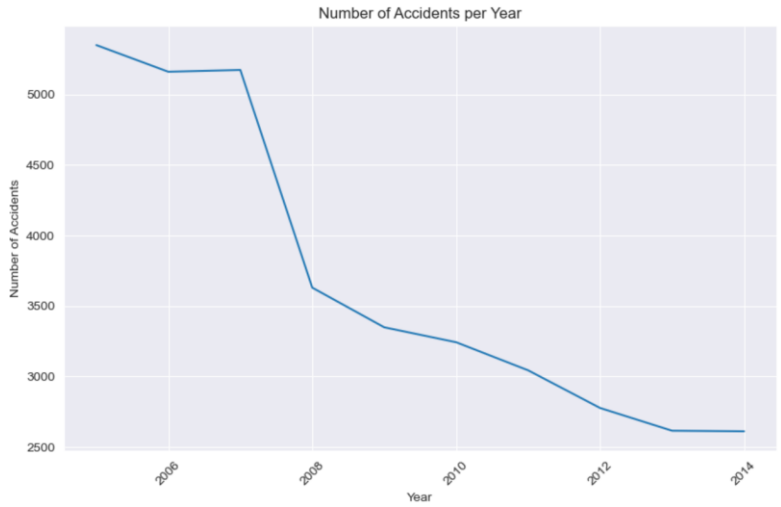

Fig. 1: Line graph of no. of accidents per year

Observations: The graph shows a consistent decline in road accidents from 2005 to 2014, with a steep drop between 2007 and 2008, followed by a more gradual decrease and a potential leveling off in the latter years, suggesting improved road safety or other influencing factors during this period.

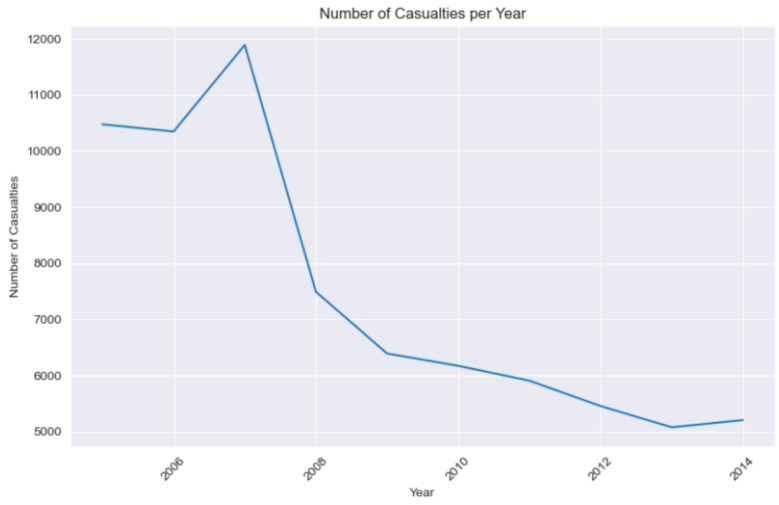

Fig. 2: Line graph of no. of casualties per year

Observations: The graph illustrates a dramatic decline in the number of casualties from road accidents from a peak around 2006 to 2014, with a sharp decrease after 2007 and a general downward trend throughout the period, indicating potential improvements in road safety and emergency response.

Bar Plots:

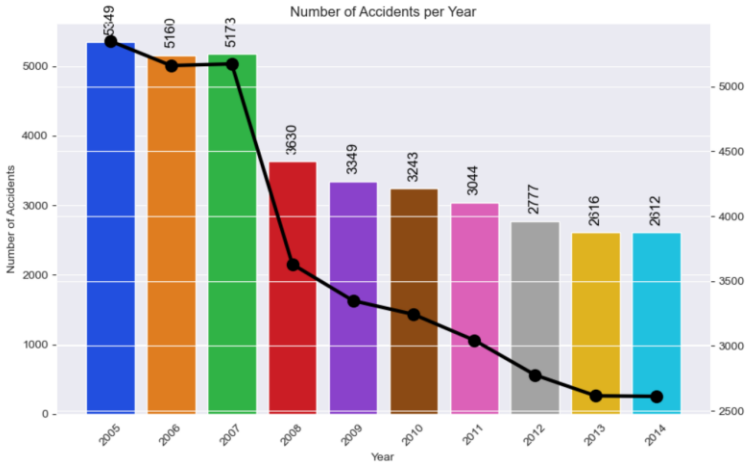

Fig. 3: Bar graph of no. of accidents per year

Observations: The graph depicts a continuous decline in the number of road accidents from 2005 to 2014, with colored bars representing each year and a black line indicating the trend. The sharp decrease between 2007 and 2008 stands out, and the overall trend line shows a steady reduction in accidents over the decade.

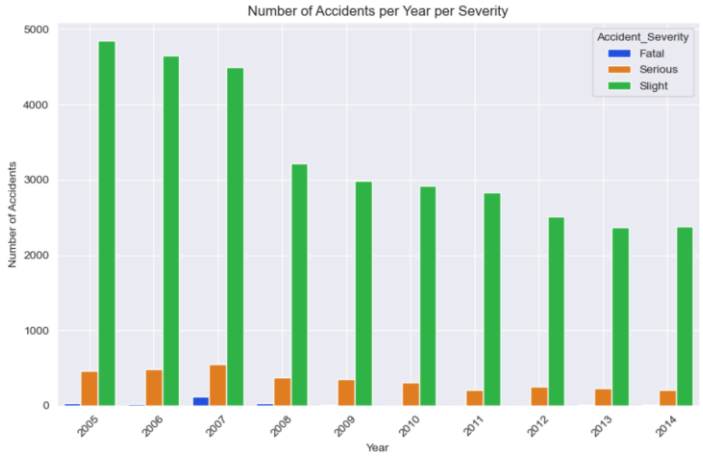

Fig. 4: Bar graph of no. of accidents per year per severity

Observations: The graph presents the number of road accidents from 2005 to 2014, categorized by severity: fatal, serious, and slight. Across the years, there's a noticeable decrease in all types of accidents, with slight accidents being the most common, followed by serious, and then fatal, which are the least frequent. The overall trend shows a significant reduction in the total number of accidents, indicating potential improvements in road safety.

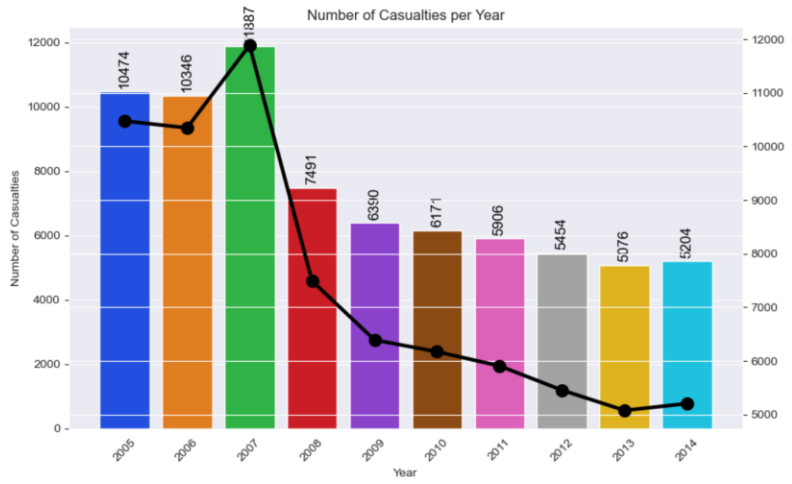

Fig. 5: Bar graph of no. of casualties per year

Observations: The graph indicates a significant downward trend in the number of casualties from road accidents in Great Britain from 2005 to 2014. Notably, there is a sharp decline after 2007. The overall reduction in casualties suggests enhanced road safety and possibly more effective emergency responses during this period.

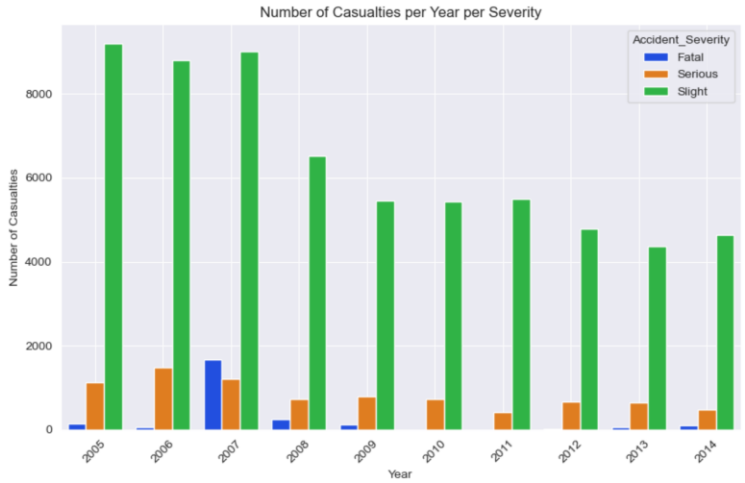

Fig. 6: Bar graph of no. of casualties per year per severity

Observations: The graph shows a consistent decline in road casualties in Great Britain from 2005 to 2014, categorized by severity—fatal, serious, and slight. Slight casualties are the most common, serious casualties less so, and fatal casualties the least, with all categories showing a downward trend over the ten-year period.

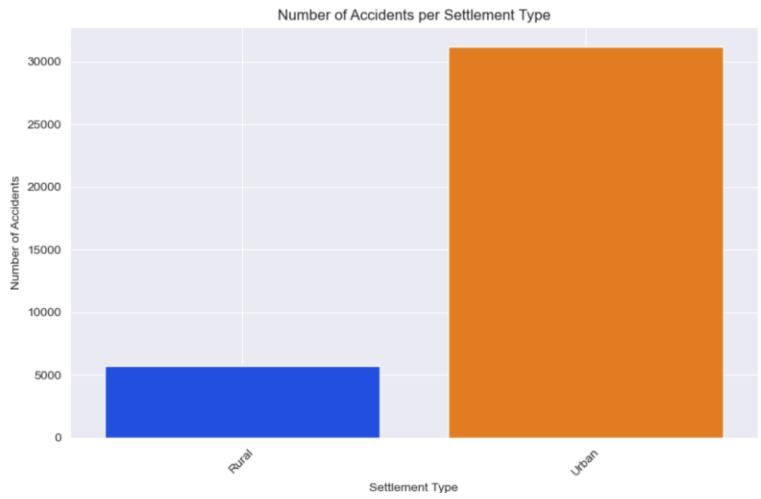

Fig. 7: Bar graph of no. of accidents per year per settlements

Observations: The graph compares the number of road accidents in rural versus urban settings. It shows a significantly higher number of accidents in urban areas compared to rural areas, indicating that urban centers may have more risk factors contributing to road accidents, or simply a higher volume of traffic leading to a greater incidence of accidents.

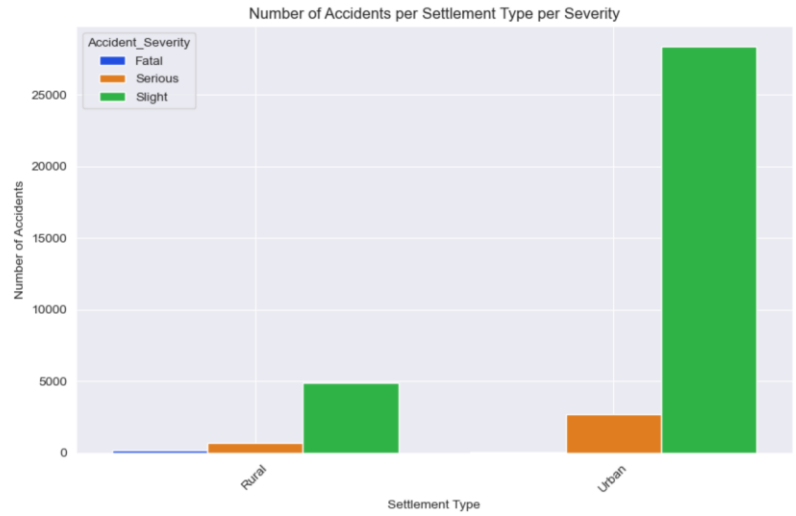

Fig. 8: Bar graph of no. of accidents per year per settlements per severity

Observations: The graph illustrates the distribution of road accidents by severity in rural and urban settings. In both environments, accidents of slight severity are most common, but the total number of accidents is markedly higher in urban areas across all severity levels. Fatal accidents are relatively low in both settings, yet they, along with serious accidents, contribute to the overall higher accident count in urban areas.

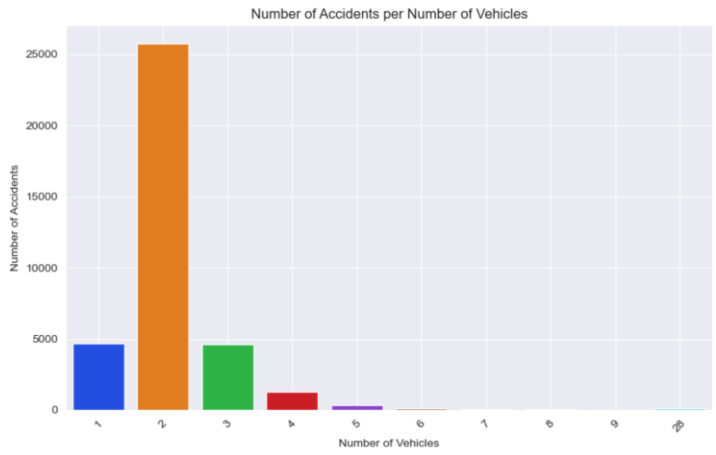

Fig. 9: Bar graph of no. of vehicles affected in accidents.

Observations: The graph shows the number of road accidents correlated to the number of vehicles involved. Most accidents involve two vehicles, followed by accidents with a single vehicle. The frequency of accidents decreases as the number of vehicles involved increases, with incidents involving more than two vehicles becoming progressively rarer. This pattern suggests that the most common road accidents are likely to be between two vehicles or involve a single vehicle.

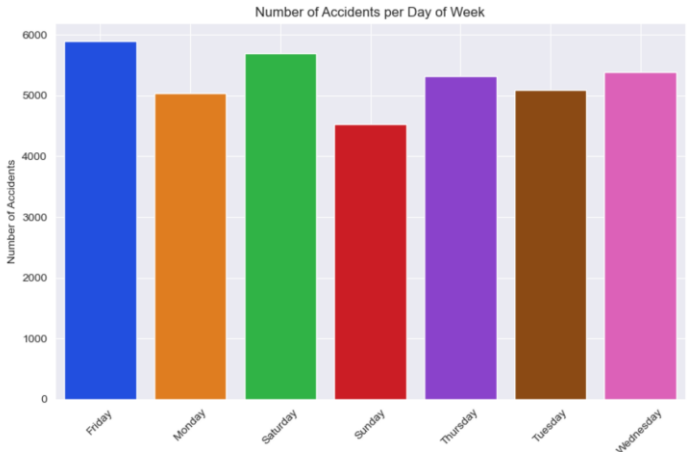

Fig. 10: Bar graph of accidents occurred in weekends/weekdays

Observations: The graph shows the distribution of road accidents across different days of the week. Friday has the highest number of accidents, followed by a somewhat uniform distribution among the other weekdays. Sunday is observed to have the lowest number of accidents. This pattern may indicate varying traffic conditions and behaviors, with Friday potentially having more traffic due to weekend plans and Sunday being quieter on the roads.

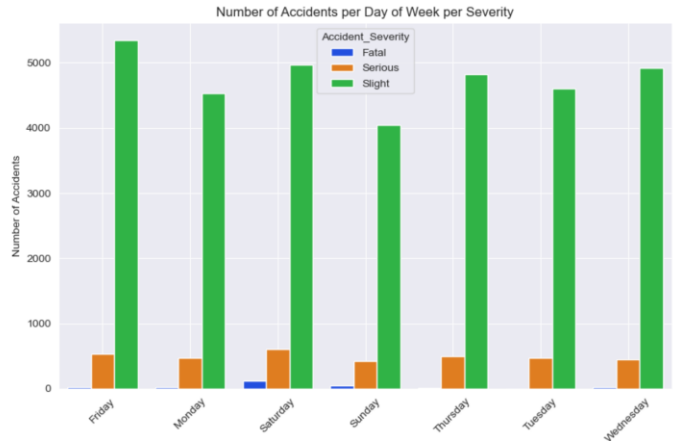

Fig. 11: Bar graph of accidents occurring in weekends/weekdays per severity

Observations: The graph illustrates the number of road accidents by day of the week, categorized by severity (fatal, serious, slight). Slight accidents are the most frequent on all days, with the highest overall number of accidents occurring midweek. Fatal accidents are the least frequent but appear slightly more often during the weekend. This suggests that while slight accidents are common throughout the week, the weekends may pose a higher risk for more severe accidents.

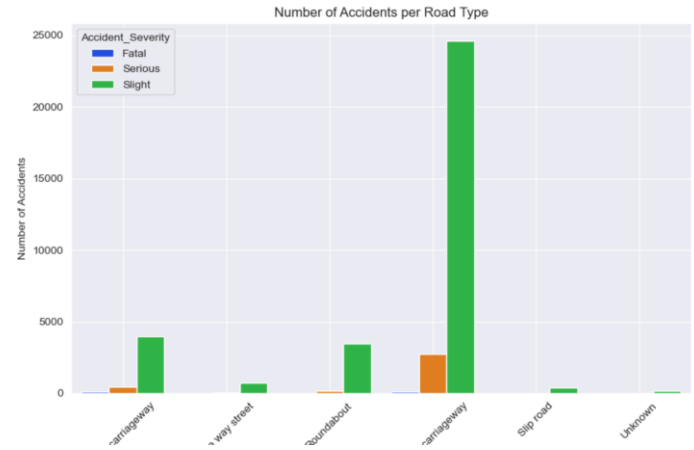

Fig. 12: Bar graph of accidents per road type

Observations: The graph categorizes road accidents by road type and severity. Single carriageways see the highest number of accidents, particularly those classified as slight. Dual carriageways, roundabouts, and slip roads feature fewer accidents, with the least on slip roads. Fatal accidents occur least frequently across all road types. The data indicates that single carriageways might be hotspots for traffic incidents, warranting further safety analysis and interventions.

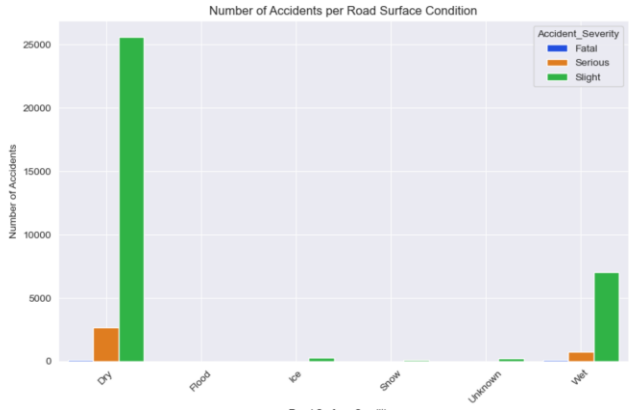

Fig. 13: Bar graph of accidents per road surface conditions

Observations: The graph shows the number of road accidents according to road surface conditions. The majority of accidents occur on dry roads, followed by wet road conditions. Accidents on icy, snowy, or flooded surfaces are comparatively fewer. The 'Unknown' category suggests a non-negligible number of accidents where the road condition wasn't reported or determined. This data may reflect that while adverse weather conditions are risk factors, the sheer volume of traffic on dry roads results in a higher frequency of accidents.

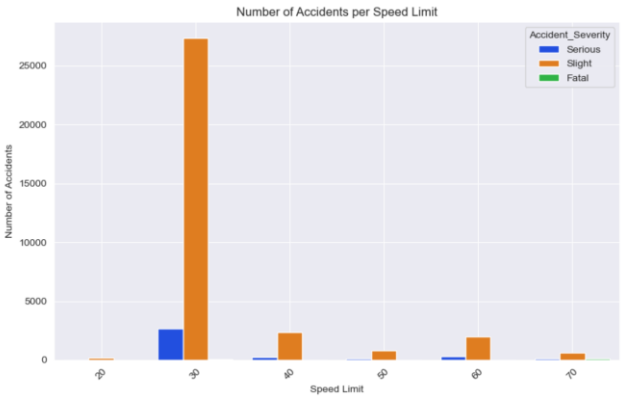

Fig. 14: Bar graph of accidents due to speed limit

Observations: The graph shows the number of road accidents in relation to different speed limits. Most accidents occur in areas with a speed limit of 30 mph, which are typically urban or residential areas. The frequency of accidents decreases significantly as the speed limit increases, with the fewest accidents occurring in the 70 mph speed limit zones. This suggests that lower speed limit areas, despite the reduced speed, may have a higher incidence of accidents, possibly due to increased traffic density or intersections.

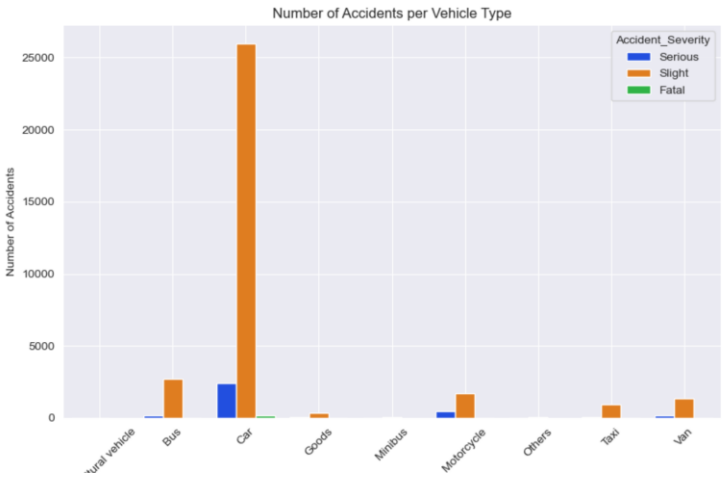

Fig. 15: Bar graph of unique vehicles affected in accidents.

Observations: The graph displays the number of road accidents categorized by vehicle type, along with the severity of the accidents (fatal, serious, slight). Cars are involved in the vast majority of accidents, with a high number of slight severity, followed by serious and fewer fatal accidents. All other vehicle types show significantly fewer accidents by comparison. This data highlights that cars are the most common participants in road accidents, which could be reflective of their prevalence on the roads.

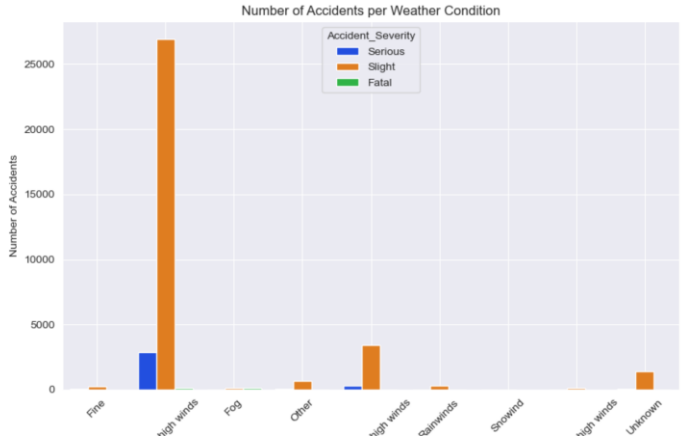

Fig. 16: Bar graph of weather conditions per severity

Observations: The graph depicts road accidents in relation to different weather conditions. The highest number of accidents occur during fine, clear weather, with a significantly smaller number occurring during rain and other weather conditions. Incidents during snow and high winds are even less common. Fatal accidents are relatively low across all weather conditions. This could suggest that although adverse weather conditions are typically considered hazardous, most accidents happen during clear weather conditions.

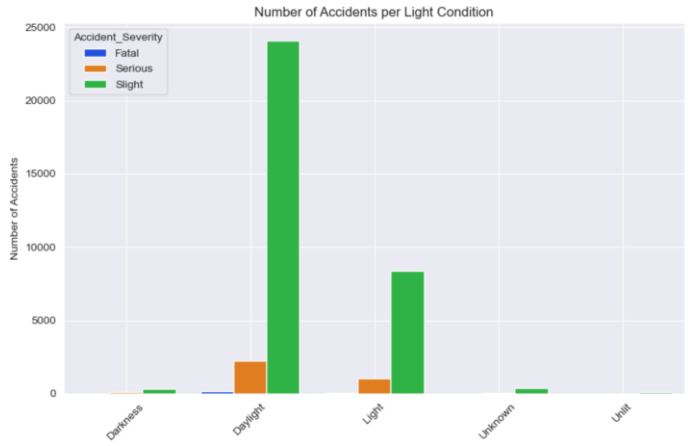

Fig. 17: Bar graph of light conditions per severity

Observations: The chart displays road accidents based on light conditions, showing that the majority occur in daylight, with a significantly smaller number happening in darkness and even fewer when streetlights are present or there is lighting. Accidents under unknown lighting conditions are minimal. The severity of accidents follows a similar pattern across different lighting, with slight accidents being the most common, followed by serious, and then fatal accidents being the least common in all lighting conditions.

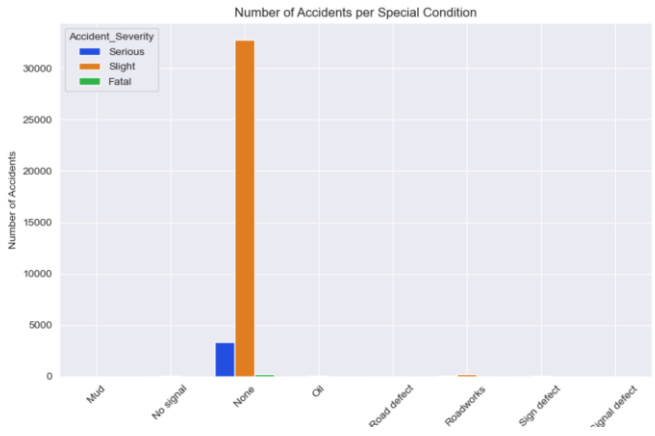

Fig. 18: Bar graph of special conditions per severity

Observations: The chart categorizes road accidents based on special road conditions. The overwhelming majority of accidents occur under no special conditions, indicating that typical driving environments are where most incidents happen. Accidents in conditions like mud, oil, road defects, roadworks, sign defects, and signal defects are significantly less frequent.

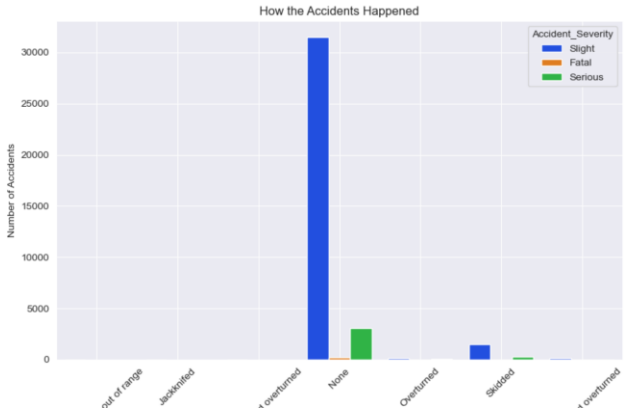

Fig. 19: Bar graph of how accidents happened.

Observations: The chart categorizes road accidents by the manner in which they occurred. A significant majority of the accidents are not specified by any particular manner, indicating a 'None' category. Among the specified categories, 'Overturned' and 'Skidded' represent a small fraction of the accidents. Fatal accidents are the least common in all categories. The dominance of the 'None' category could suggest either a lack of detailed reporting in the data or that most accidents don't involve the vehicle overturning or skidding.

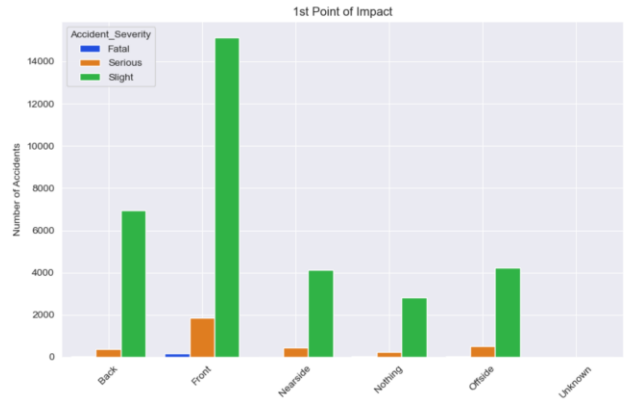

Fig. 20: Bar graph of first point of impact

Observations: The graph categorizes road accidents by the first point of impact on the vehicle. The front of the vehicle is the most common point of impact, with a significant number of both slight and serious accidents, and a smaller number of fatal accidents. The back and offside of the vehicle are the next most common points of impact, but with considerably fewer accidents than the front. Nearside impacts and accidents without a specified impact point (Nothing) are less common, while a very small number of accidents have an unknown point of impact. This suggests that head-on or frontal impacts are the predominant type of accidents.

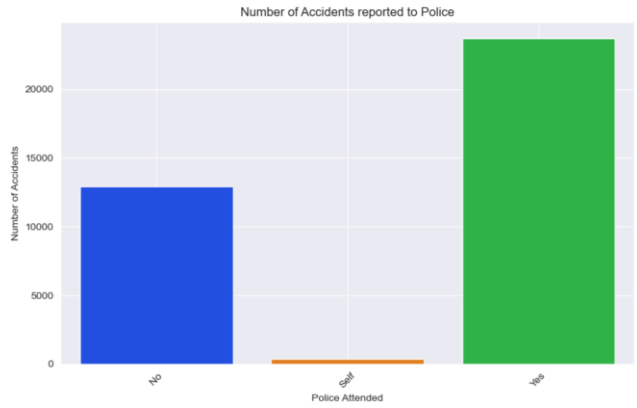

Fig. 21: Bar graph of how accidents happened.

Observations: The graph displays the number of road accidents that were reported to the police. A significant majority of accidents were reported and attended by police, whereas a much smaller number were not reported. There is a negligible category labeled "Self", which could indicate incidents that were self-reported by the involved parties without requiring police at the scene.

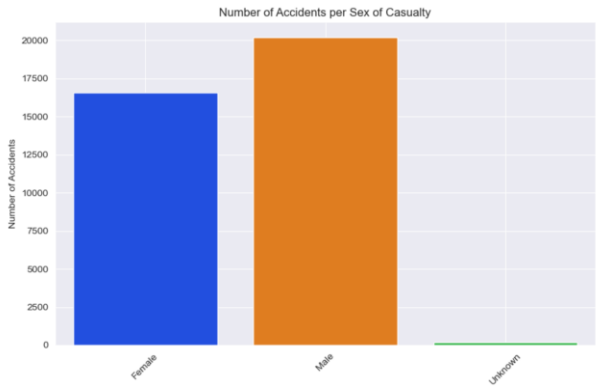

Fig. 22: Bar graph of no. of accidents to sex of casualty

Observations: The chart shows the number of road accidents categorized by the sex of the casualty. There is a relatively even distribution between female and male casualties, with male casualties being slightly higher. A very small number of accidents involve casualties whose sex was unknown.

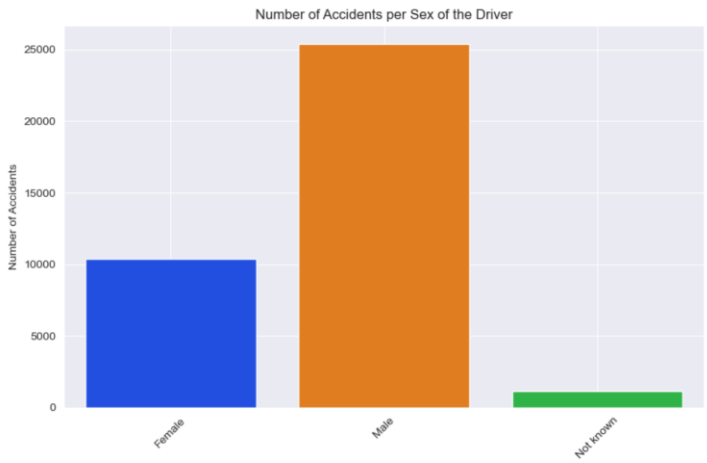

Fig. 23: Bar graph of no. of accidents to sex of the driver

Observations: The graph illustrates the number of road accidents based on the sex of the driver involved. It shows that male drivers are involved in a significantly higher number of accidents than female drivers. There is also a small number of accidents where the sex of the driver is not known.

Pie Charts:

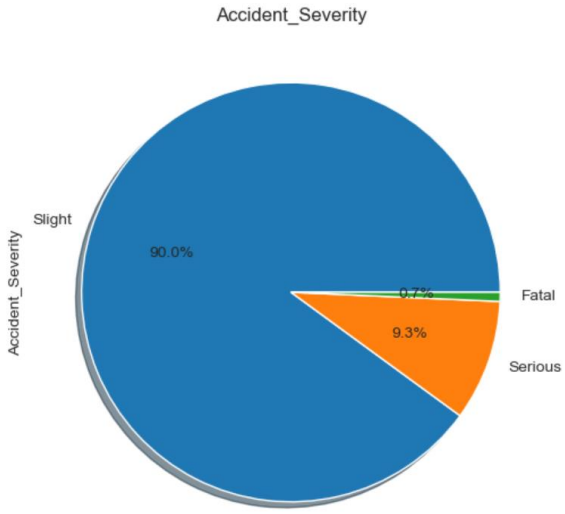

Fig. 24: Pie chart of accident severity

Observations: The pie chart represents the distribution of road accidents by severity. The vast majority, 90%, are classified as slight accidents, indicating minor injuries or damage. Serious accidents account for 9.3% of the total, signifying more significant harm or potential for long-term consequences. Fatal accidents comprise the smallest portion, at 0.7%, indicating a relatively low incidence of accidents resulting in death compared to non-fatal incidents.

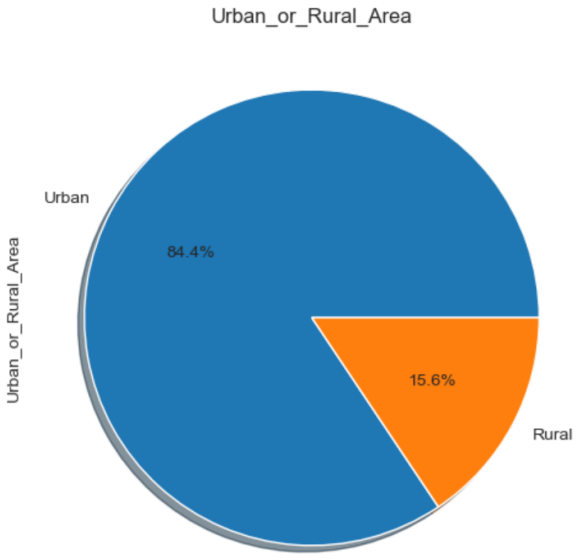

Fig. 25: Pie chart of settlement type

Observations: The pie chart illustrates the distribution of road accidents between urban and rural areas. A significant majority, 84.4%, of accidents occur in urban areas, while the remaining 15.6% take place in rural areas.

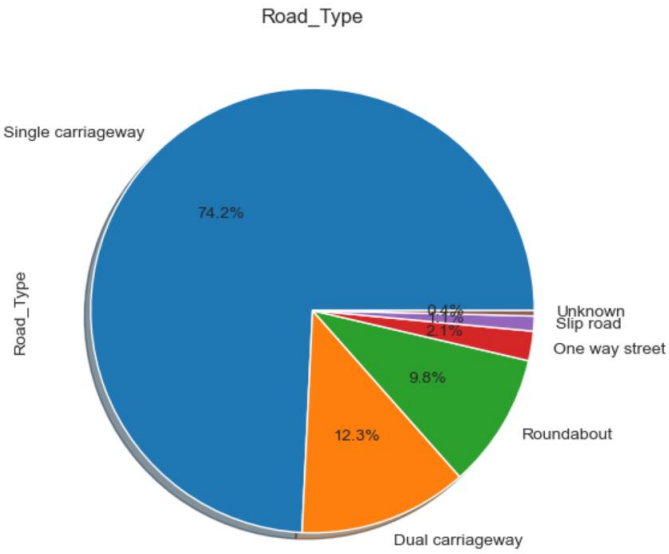

Fig. 26: Pie chart of road type

Observations: The pie chart presents the distribution of road accidents by road type. Most accidents occur on single carriageways, accounting for 74.2% of the total. Dual carriageways are the next most common site for accidents, with 12.3%. Roundabouts see a smaller proportion of accidents, represented by 9.8%. Accidents on one-way streets and slip roads, as well as those on roads of unknown types, make up a very small fraction of the total.

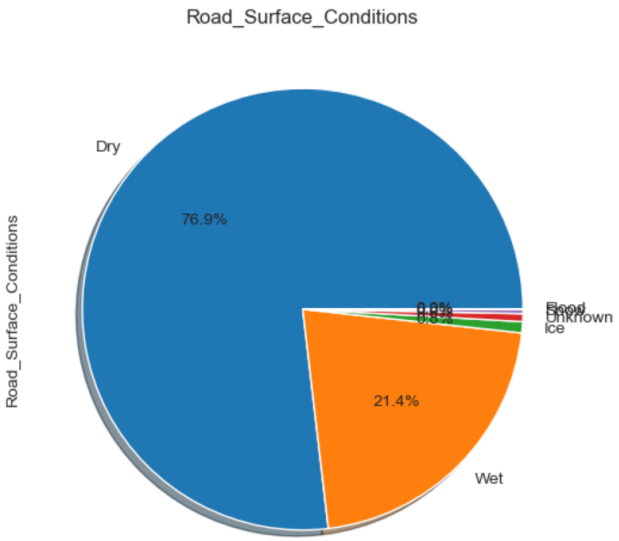

Fig. 27: Pie chart of road surface conditions

Observations: The pie chart indicates the proportion of road accidents occurring under different road surface conditions. A large majority, 76.9%, of accidents happen on dry road surfaces. Wet road conditions account for 21.4% of accidents. A very small percentage occurs in snowy or icy conditions, and an even smaller fraction is categorized under unknown conditions.

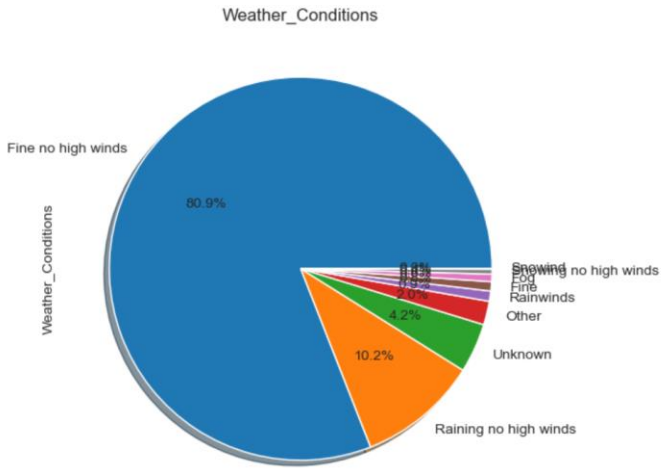

Fig. 28: Pie chart of weather conditions

Observations: The pie chart displays the distribution of road accidents by weather conditions. Most accidents, 80.9%, occur during fine weather with no high winds. Accidents during rainy weather with no high winds account for 10.2%, followed by a small percentage of accidents in other specified weather conditions such as snow and fog. A very small fraction is recorded under unknown weather conditions.

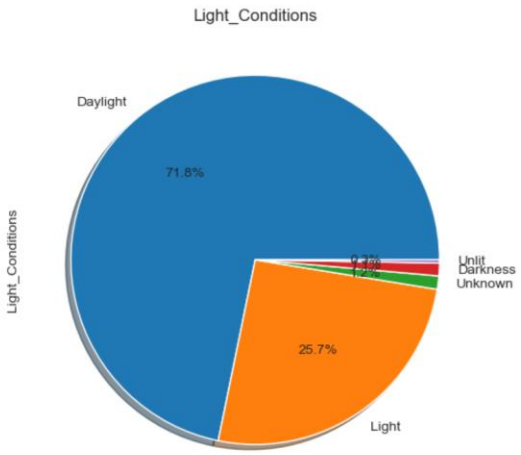

Fig. 29: Pie chart of light conditions

Observations: The pie chart shows the breakdown of road accidents according to the light conditions at the time they occurred. A majority of accidents, 71.8%, happen in daylight. Accidents in light conditions, which may refer to street-lit conditions, account for 25.7%. A very small portion occurs in unlit darkness, and an even smaller fraction is under unknown lighting conditions.

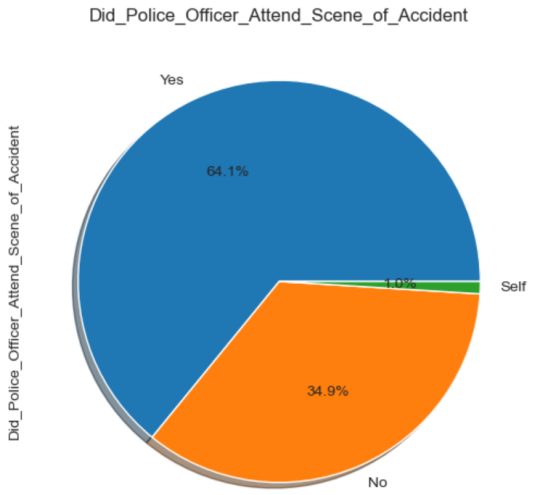

Fig. 30: Pie chart of accidents reported to police.

Observations: The pie chart represents the frequency of police attendance at the scene of road accidents. In 64.1% of the cases, a police officer did attend the scene of the accident. There were 34.9% of accidents where no police officer attended, and a very small

percentage of accidents were reported as 'Self', which may indicate self-reporting to the police without their attendance at the scene.

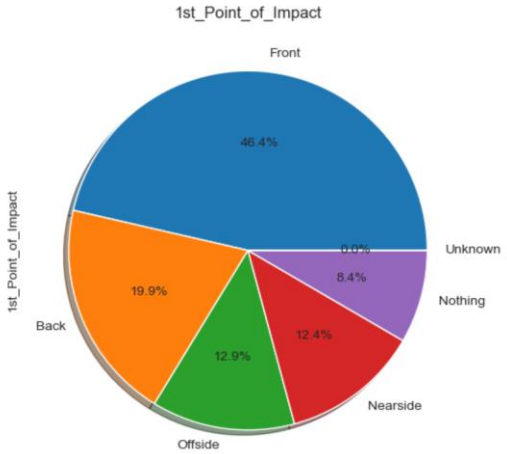

Fig. 31: Pie chart of first point of impact

Observations: The pie chart displays the distribution of the first point of impact in a dataset. The largest segment, colored in blue, represents the 'Front' impact, accounting for 46.4% of the cases. This is followed by 'Back' and 'Offside' impacts, which are 19.9% and 12.9%, respectively. 'Nearside' impacts are at 12.4%, while there are 8.4% of cases with 'Nothing' reported.

Pair Plot:

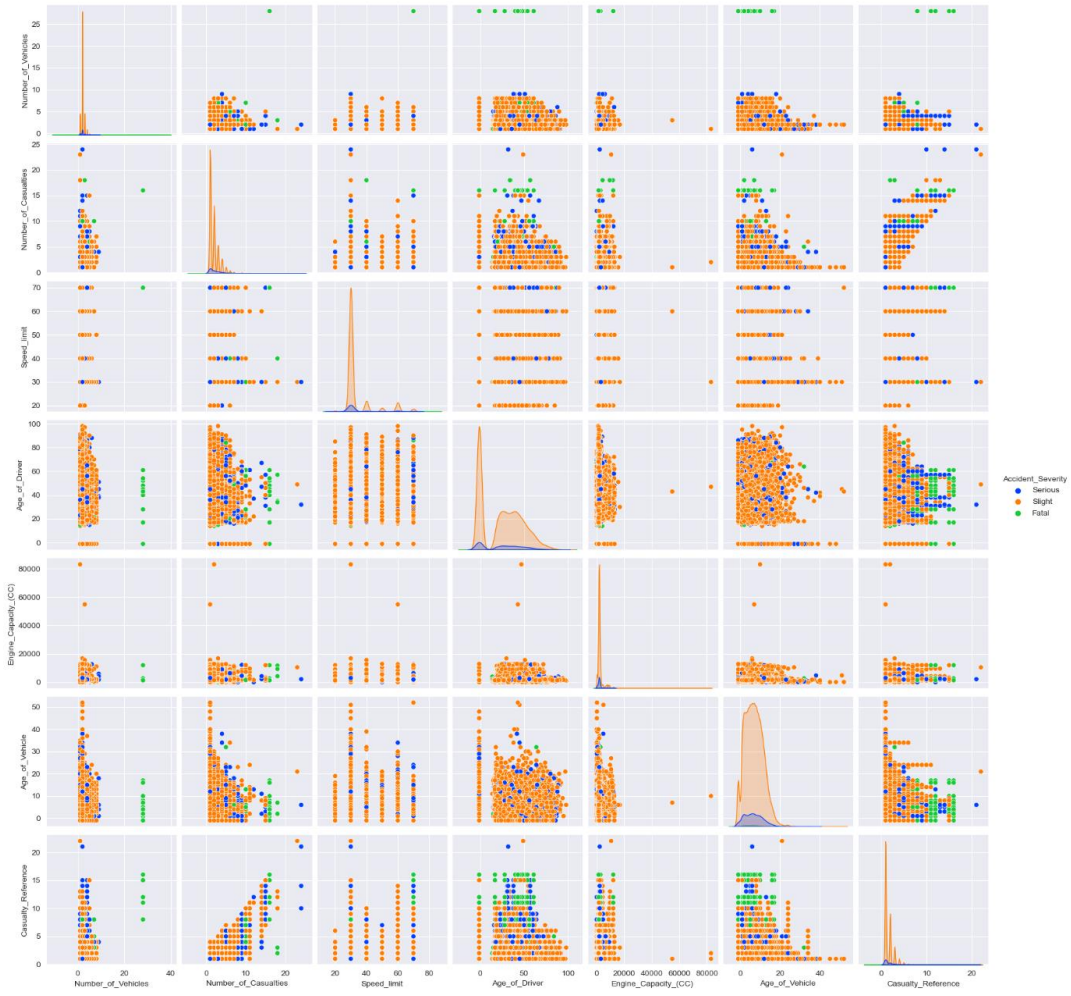

Fig. 32: Pairplot of numerical features

Observations: The pairplot shows the comprehensive overview of how each variable in a dataset relates to the others in the dataset. The diagonal plots, typically histograms, show the distribution of individual variables, which can reveal skewness, peaks, and the range of values. The scatter plots show how variables correlate with each other. Tight clusters may indicate strong correlations, while more dispersed points suggest weaker relationships. Any points that lie far away from the others could be outliers. Here we can see the engine capacity has the outliers.

Heatmap: Covariance Matrix

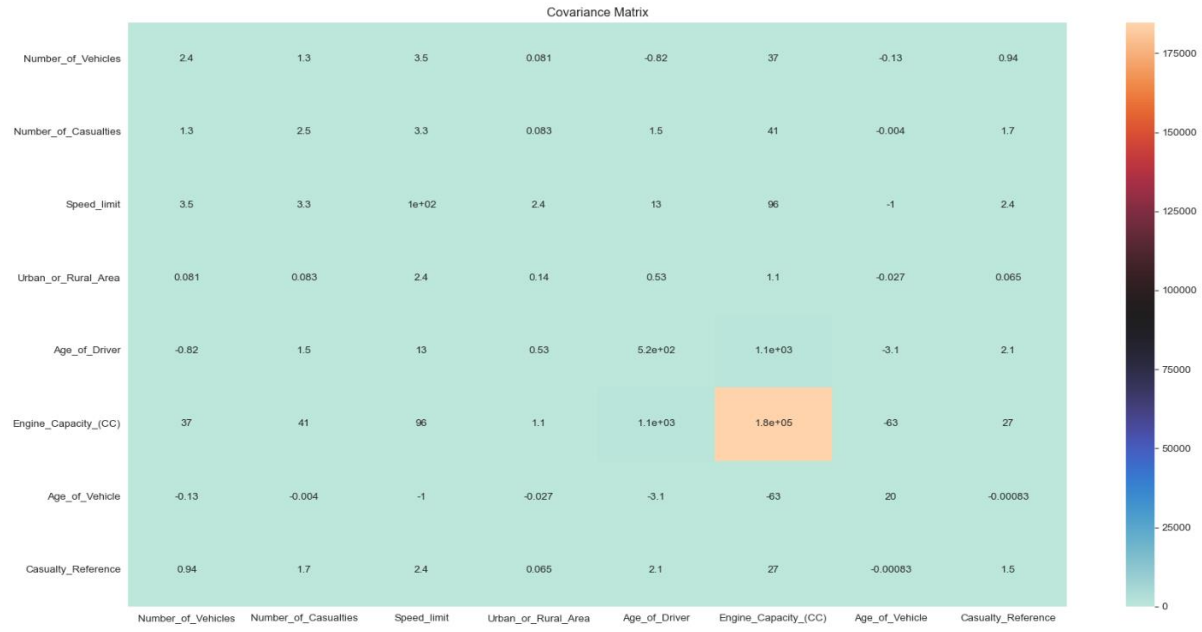

Fig. 33: Covariance Heatmap of numerical features

Observations: The covariance matrix heatmap represents the covariance between different variables related to traffic incidents, such as the number of vehicles and casualties, speed limits, whether the area is urban or rural, the age of the driver and the vehicle, engine capacity, and casualty reference. The most notable figure is the large covariance between engine capacity by itself (1.8e+05). Other values show lower magnitudes, suggesting weaker linear relationships.

Heatmap: Correlation Matrix

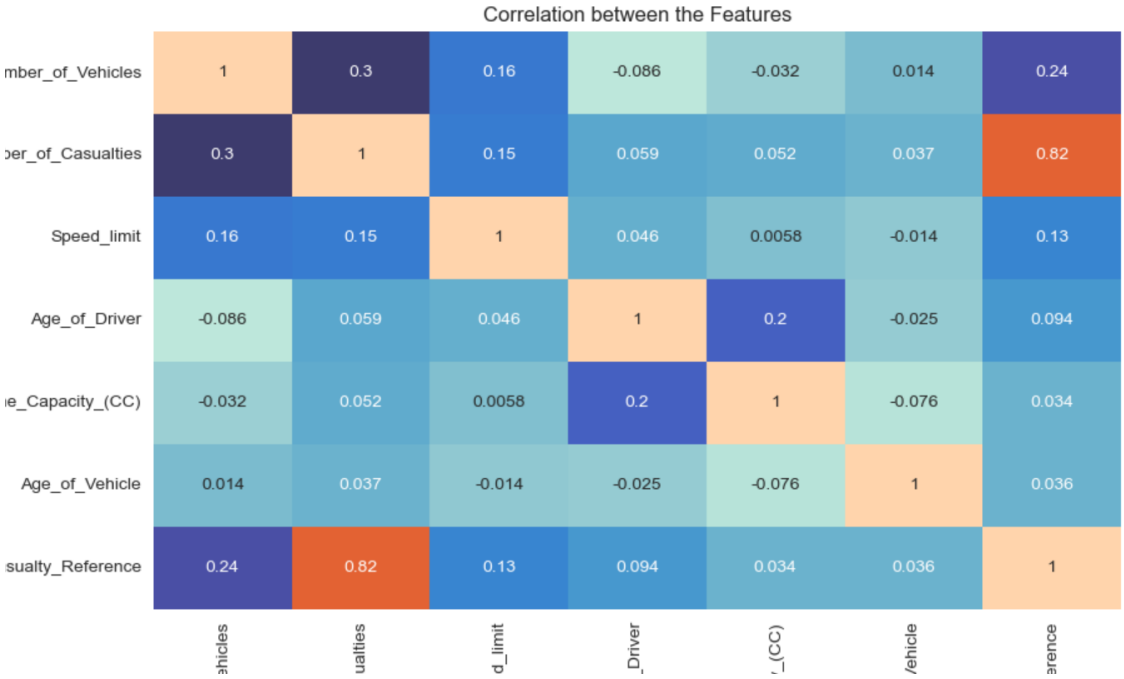

Fig. 34: Heatmap of numerical features

Observations: The heatmap visualizes the correlation coefficients between various variables related to vehicle accidents. The values range from -1 to 1, where 1 indicates a perfect positive correlation, -1 a perfect negative correlation, and 0 no correlation. The strongest positive correlation is between the number of vehicles and casualty reference, while a notable positive correlation is also observed between the number of casualties and casualty reference. Most variables show low to moderate correlation with each other, as indicated by the mix of warm and cool colors.

Outlier Detection:

The data is checked for outlier detection. The outliers are removed from the dataset using the “Interquartile Range” method. The interquartile range (IQR) is a statistical measure used to describe the spread or dispersion of data within a dataset. It is a robust measure of variability that is not influenced by extreme outliers or skewed data distributions. The IQR is defined as the range between the first quartile (Q1) and the third quartile (Q3) of a dataset.

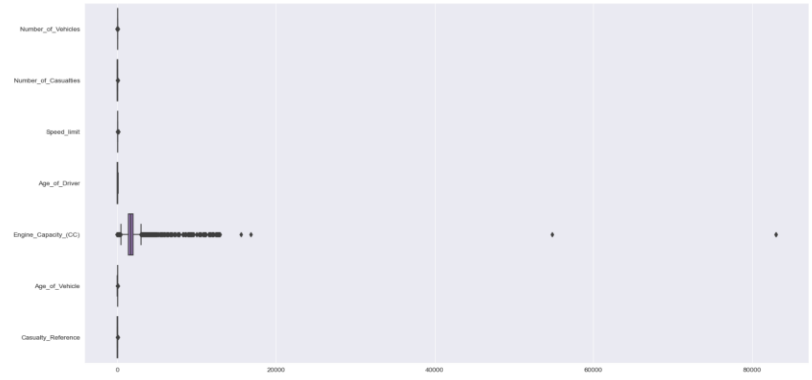

Fig. 35: Box plot of numerical features before Outlier Detection

Observations: The box plot that presents a range of data across various categories, with engine capacity showing the highest number of outliers.

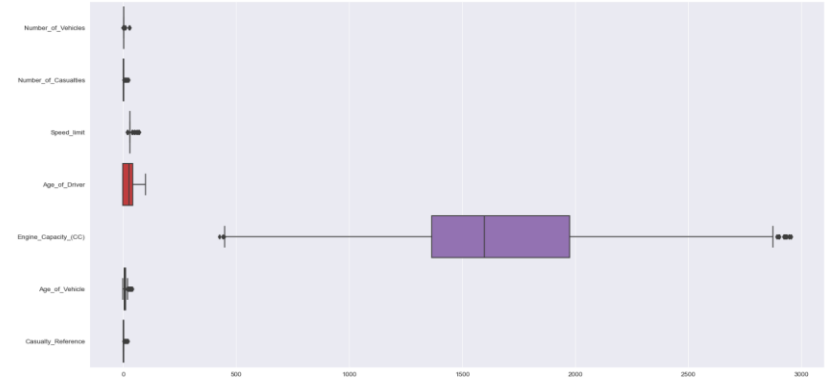

Fig. 36: Box plot of numerical features after Outlier Detection

Observations: The box plot that presents a range of data across various categories. There are some visible outliers in the plot, but these values are not that extreme compared to the former.

Data Balancing:

In many machine learning applications, balancing the target variable is essential because it assures fair and accurate predictions and helps prevent model bias. The model may become biased in favor of the majority class and result in subpar performance on the minority class when the target variable is unbalanced, meaning that one class greatly outnumbers the others. This imbalance may cause the model to mistakenly prioritize accuracy over accurately recognizing uncommon but significant outcomes. Machine learning models are better suited to provide more equitable and dependable predictions across all classes by balancing the target variable, especially in scenarios with imbalanced datasets. This can be done by using appropriate performance metrics like precision-recall instead of accuracy, or by resampling techniques like oversampling the minority class or under sampling the majority class. In this case, the dataset is balanced by using SMOTE, i.e., Synthetic Minority Oversampling Technique, which increases the minority sample in the data, so that the data is balanced.

Here are the results of the data balancing before and after process:

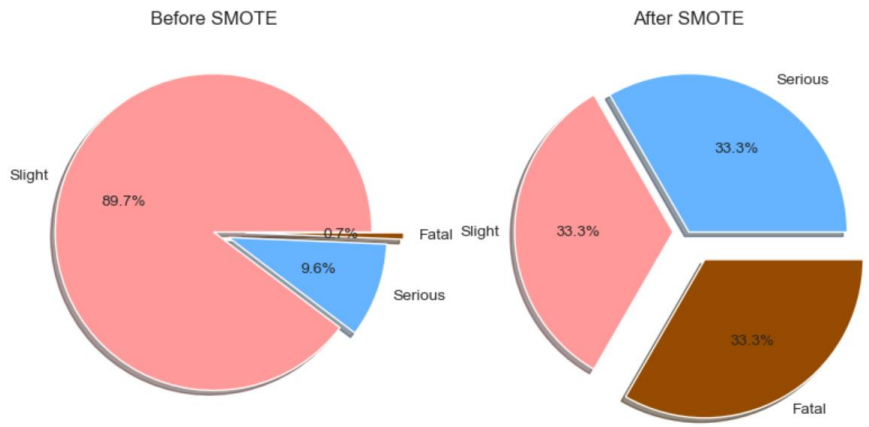

Fig. 37: Data Balancing before and after SMOTE

Observations: The provided pie charts illustrate the distribution of three accident severity categories before and after the application of SMOTE (Synthetic Minority Over-sampling Technique):

- **Before SMOTE**: The chart is overwhelmingly dominated by 'Slight' accidents at 89.7%. 'Serious' accidents make up 9.6%, and 'Fatal' accidents are the least common at only 0.7%.

- **After SMOTE**: The chart shows an equal distribution among the three accident severities, with each category now making up exactly one-third (33.3%) of the data.

Then the data is divided into its respective X and y indices, as well as breaking down X and y into X_train, X_test, y_train, y_test.

Feature Selection:

1. **Principal Component Analysis and Conditional Number**

Principal Component Analysis (PCA) is a dimensionality reduction technique widely used in data analysis and machine learning. Its primary goal is to transform a high-dimensional dataset into a lower-dimensional one while retaining as much of the original data's variability as possible. PCA identifies linear combinations of the original features, known as principal components, which capture the most significant variations in the data. These components are ordered by their importance, with the first component explaining the most variance and subsequent components explaining progressively less.

In this case the dataset is fed to the PCA algorithm, and the features are selected based on higher principal components, the next less component, and so on, for 90% explained cumulative variance.

Here are the results of PCA analysis:

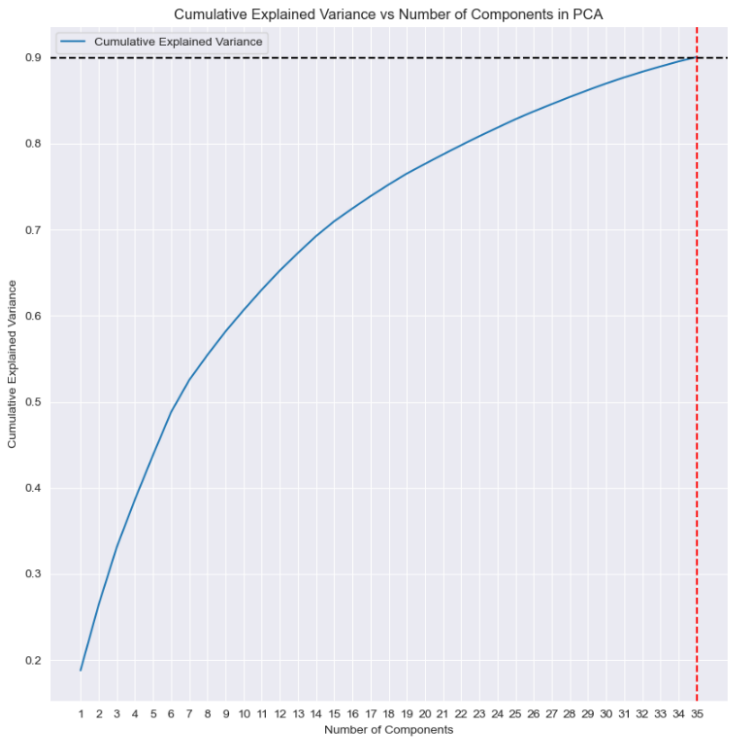

Fig. 38: Principal Component Analysis

Observations: The graph depicts the cumulative explained variance by the number of components, typically used in Principal Component Analysis. The curve increases steeply at the beginning, indicating that the initial components explain a significant amount of

variance within the data. As more components are added, the rate of increase in explained variance slows down, but the cumulative explained variance continues to rise.

The plot has a dashed horizontal line close to the 0.9 mark, which is commonly used as a cutoff for deciding how many components to retain. The vertical dashed line intersects the horizontal line at around 35 components, suggesting that 36 components explain approximately 90% of the variance in the data.

Here is the conditional number of the original data vs the reduced data:

```
Condition Number of Original data: 1.5421158390819538e+18
Condition Number of Reduced data: 6.007484197095177
```

2. Random Forest Analysis: Feature Importance

Feature importance in a Random Forest model is a technique used to identify which features (or variables) in a dataset are most influential in predicting the outcome. In a Random Forest, which is an ensemble of decision trees, each tree is built on a subset of data and features. During this process, the algorithm assesses how much each feature decreases the impurity in a tree; commonly, impurity is measured using criteria like Gini impurity or entropy in classification problems, and variance in regression.

In simple terms, feature importance in a Random Forest can be understood as a way of ranking the features based on how well they contribute to the model's ability to make accurate predictions. Features that lead to the most significant decrease in impurity across all trees in the forest are considered more important. This information can be crucial for understanding the data, improving model performance, and making informed decisions based on the model's outputs.

Here are the results:

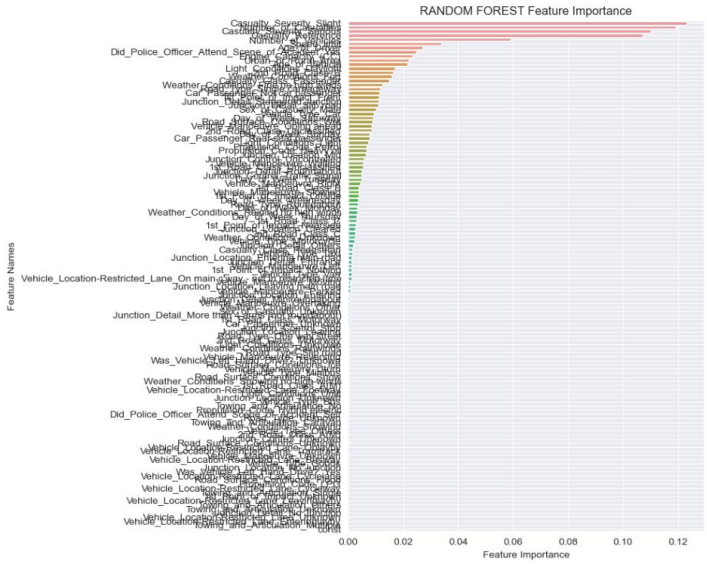

Fig. 39: Feature Importance by Random Forest

Observations: This graph shows that a few features stand out with significantly higher importance scores compared to the rest. The most important feature dominates, with a score just above 0.01, while the next few features have progressively lower scores, indicating a steep drop in importance. Most features have very low importance scores, contributing minimally to the model. The color coding may represent different categories or types of features, with the most important features having distinct colors compared to the rest.

3. Singular Value Decomposition

Singular Value Decomposition (SVD) is a powerful linear algebra technique that can be applied to feature selection, particularly in the context of dimensionality reduction.

SVD decomposes a matrix into three other matrices, capturing the essential elements of the original data in a lower-dimensional space. In feature selection, SVD can be used to transform a high-dimensional dataset into a smaller set of uncorrelated components that still capture most of the variability in the data.

The process begins with the application of SVD to the data matrix to identify the singular values and corresponding singular vectors. These singular values indicate the importance or weight of each component in capturing the data's variance. By examining these values, one can determine the number of components that contribute most significantly to the data structure.

In practice, one might retain components that account for a substantial proportion of the variance, such as 90% or 95%. This cutoff is somewhat arbitrary and may be adjusted depending on the specific needs of the analysis or the acceptable level of information loss.

By reducing the dimensionality of the dataset while preserving its essential characteristics, SVD aids in mitigating issues like overfitting and reduces computational costs. The selected components can be used as new features in a machine learning model, often leading to improved model performance due to the removal of noise and less informative variables.

Here are the results:

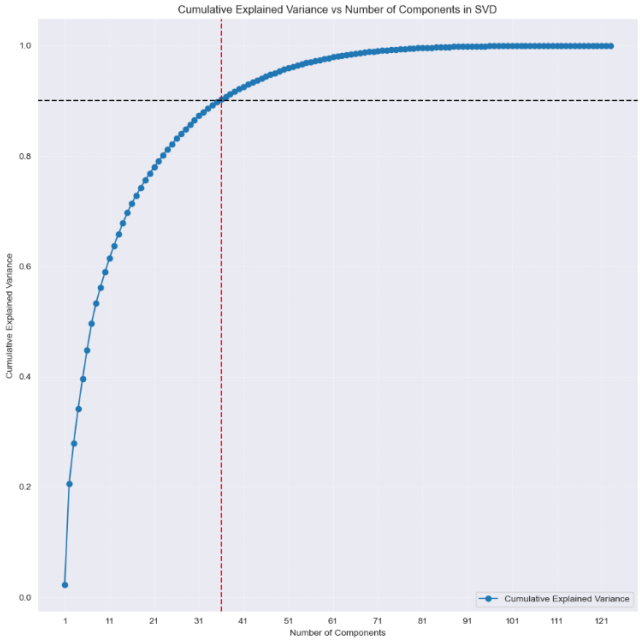

Fig. 40: Singular Value Decomposition

Observations: This graph indicates that as the number of components increases, the cumulative explained variance increases sharply at first, but the rate of increase diminishes around the 20-component mark. Around 90% of the variance is explained by approximately 36 components, as shown by where the curve intersects the dashed line, suggesting that beyond this point, additional components contribute less significantly to explaining the variance in the dataset.

Here is the conditional number of the original data vs the reduced data:

```
Condition Number of Original data: 1.5421158390819538e+18
Condition Number of Reduced data: 16.52127059550843
```

4. Variance Inflation Factor

Feature selection through Variance Inflation Factor (VIF) is a technique used to identify and eliminate multicollinearity within predictor variables in regression models. VIF quantifies how much the variance of an estimated regression coefficient increases if your predictors are correlated. If no factors are correlated, the VIFs will be close to 1. Generally, a VIF above 5-10 indicates a problematic level of collinearity. By calculating VIF for each predictor, one can assess which variables are causing multicollinearity and should be removed or adjusted.

Here are the results of the no. of features removed in each iteration:

Iteration 1:

```
[116 rows x 2 columns]
Number of features removed: 7
```

Iteration 2:

```
[106 rows x 2 columns]
Number of features removed: 10
```

Iteration 3:

```
[106 rows x 2 columns]
Number of features removed: 0
```

Total no. of features removed:

```
Total number of features removed: 17
```

Best Feature Selection Technique:

Feature Selection/Dimensionality Reduction Comparison		
Feature Selection Technique	Number of Features Selected	Number of Features Removed
PCA	35	91
Random Forest	21	103
Singular Value Decomposition Analysis	36	88
Variance Inflation Factor	106	17

Table 1: Feature Selection/ Dimensionality Reduction Comparison

According to me, the Random Forest Feature Importance is the best feature selection technique the Random Forest Feature Importance resulted in fewer features being selected compared to PCA, Singular Value Decomposition, and Variance Inflation Factor with a total of 21 features selected. It can be said that Random Forest was effective at identifying the most relevant features. The ability of Random Forest to handle non-linear relationships and interactions between features, along with its inherent feature selection capabilities, can often make it a superior choice for capturing the most predictive power with the fewest features.

Phase 2: Regression Analysis

Ordinary Least Squares (OLS) regression is a foundational method in statistical modeling used to estimate the relationships between a dependent variable and one or more independent variables. The core principle of OLS regression is to minimize the sum of the squared differences between the observed values and the values predicted by the linear model, known as the residuals. This method of estimation captures the linear association between the independent variables and the dependent variable, providing the line of best fit through the data points that has the least amount of error.

The OLS approach assumes that there is a linear relationship between the variables, the residuals are normally distributed, homoscedastic (i.e., they have constant variance), and independent of each other. The resulting model provides coefficients for each independent variable, which quantify the strength and type of the relationship with the dependent variable. These coefficients are often interpreted as the mean change in the dependent variable for each one-unit change in the independent variable, holding all other variables constant. The significance of these coefficients is typically tested using t-statistics, which inform whether the relationships observed in the sample data are strong enough to be generalized to the population.

Here in this project, the target variable is 'Fuel_Specific_Accident_Impact_Score'.

T-test Analysis:

When an OLS regression is runned, the statsmodels provides a t-test for each regression coefficient (β), which tests the null hypothesis that the coefficient is equal to against the alternative hypothesis that the coefficient is not equal to zero.

The t-statistic is calculated by taking the estimated coefficient and dividing it by its standard error. The resulting t-statistic tells us how many standard deviations the coefficient is away from zero. If the t-statistic is large (which typically means the p-value is small), it suggests that the observed relationship is statistically significant, and the likelihood of the coefficient being zero (no relationship) is small. Conversely, a small t-statistic (with a large p-value) suggests that the evidence is weak against the null hypothesis of no effect.

Here is the sample of t-test in regression with threshold=0.01:

	P-Value
Road_Type_One way street	0.98
Junction_Location_Leaving	0.97
Propulsion_Code_Heavy oil	0.97
Junction_Detail_Entrance	0.97
1st_Point_of_Impact_Unknown	0.97

F-test Analysis:

In regression analysis performed by the statsmodels OLS (Ordinary Least Squares) function, the F-test is a global hypothesis test that evaluates the joint significance of all the regression coefficients. The F-test in OLS regression checks the null hypothesis that all regression coefficients are equal to zero versus the alternative hypothesis that at least one coefficient is different from zero. This test is particularly useful when comparing nested models - models where one is a special case of the other. The F-statistic is calculated as the ratio of the mean squared error of the model with no predictors (the intercept-only model) to the mean squared error of the model with predictors, adjusted for the degrees of freedom. A significant F-test indicates that the explanatory variables, as a set, are related to the response variable, and the model is a better fit than a model without them.

Essentially, the F-test provides insight into the overall validity of the model. A high F-statistic (leading to a low p-value) suggests that the group of variables are collectively significant in explaining the variation in the dependent variable.

Here is the sample of f-test in regression:

	F-Statistic
Road_Type_One way street	98.87
Junction_Location_Leaving	99.72
Propulsion_Code_Heavy oil	100.61
Junction_Detail_Entrance	101.50
1st_Point_of_Impact_Unknown	102.39

Final Regression Model and Prediction of Dependent Variable:

A final regression model represents the culmination of the model-building process where variables deemed statistically significant and theoretically relevant are included to explain the variation in the dependent variable.

Here is the final regression model:

```
                                OLS Regression Results
==============================================================================================
Dep. Variable:     Fuel_Specific_Accident_Impact_Score   R-squared:                       0.479
Model:                                             OLS   Adj. R-squared:                  0.478
Method:                                  Least Squares   F-statistic:                     394.5
Date:                                 Sat, 02 Dec 2023   Prob (F-statistic):               0.00
Time:                                         10:09:48   Log-Likelihood:                 3214.7
No. Observations:                                12464   AIC:                            -6369.
Df Residuals:                                    12434   BIC:                            -6146.
Df Model:                                           29
Covariance Type:                             nonrobust
```

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.5688	0.135	11.592	0.000	1.303	1.834
Number_of_Vehicles	0.8713	0.038	22.783	0.000	0.796	0.946
Number_of_Casualties	0.4946	0.031	15.891	0.000	0.434	0.556
Speed_limit	0.0309	0.009	3.263	0.001	0.012	0.049
Age_of_Driver	0.0513	0.008	6.630	0.000	0.036	0.066
Age_of_Vehicle	0.1236	0.016	7.733	0.000	0.092	0.155
1st_Road_Class_B	-0.0134	0.005	-2.667	0.008	-0.023	-0.004
1st_Road_Class_Motorway	-0.0580	0.020	-2.857	0.004	-0.098	-0.018
Junction_Detail_Slip road	0.0369	0.012	3.173	0.002	0.014	0.060
2nd_Road_Class_B	0.0208	0.007	3.164	0.002	0.008	0.034
Light_Conditions_Daylight	-0.0414	0.010	-4.008	0.000	-0.062	-0.021
Light_Conditions_Light	-0.0346	0.011	-3.235	0.001	-0.056	-0.014
Road_Surface_Conditions_Wet	-0.0113	0.004	-2.730	0.006	-0.019	-0.003
Vehicle_Type_Bus	-0.4838	0.051	-9.452	0.000	-0.584	-0.384
Vehicle_Type_Car	-0.2012	0.024	-8.487	0.000	-0.248	-0.155
Vehicle_Type_Motorcycle	-0.5728	0.026	-22.276	0.000	-0.623	-0.522
Vehicle_Type_Taxi	-0.0775	0.025	-3.080	0.002	-0.127	-0.028
Vehicle_Type_Van	-0.1103	0.025	-4.480	0.000	-0.159	-0.062
Vehicle_Manoeuvre_Going ahead	0.0128	0.004	3.492	0.000	0.006	0.020
Vehicle_Manoeuvre_Reversing	0.0500	0.014	3.628	0.000	0.023	0.077
Junction_Location_Mid	0.0097	0.004	2.686	0.007	0.003	0.017
1st_Point_of_Impact_Front	0.0160	0.004	3.584	0.000	0.007	0.025
1st_Point_of_Impact_Nearside	0.0228	0.006	3.778	0.000	0.011	0.035
1st_Point_of_Impact_Offside	0.0173	0.006	2.974	0.003	0.006	0.029
Propulsion_Code_Hybrid electric	-0.1643	0.027	-6.022	0.000	-0.218	-0.111
Propulsion_Code_Petrol	-0.1652	0.004	-39.437	0.000	-0.173	-0.157
Casualty_Class_Passenger	-0.0186	0.004	-4.676	0.000	-0.026	-0.011
Casualty_Class_Pedestrian	0.0279	0.007	4.112	0.000	0.015	0.041
Casualty_Severity_Serious	-0.3565	0.133	-2.686	0.007	-0.617	-0.096
Casualty_Severity_Slight	-0.8235	0.133	-6.210	0.000	-1.083	-0.564

Omnibus:	5977.781	Durbin-Watson:	2.028
Prob(Omnibus):	0.000	Jarque-Bera (JB):	80423.664
Skew:	1.961	Prob(JB):	0.00
Kurtosis:	14.810	Cond. No.	306.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

The Actual vs Predicted Values Plot:

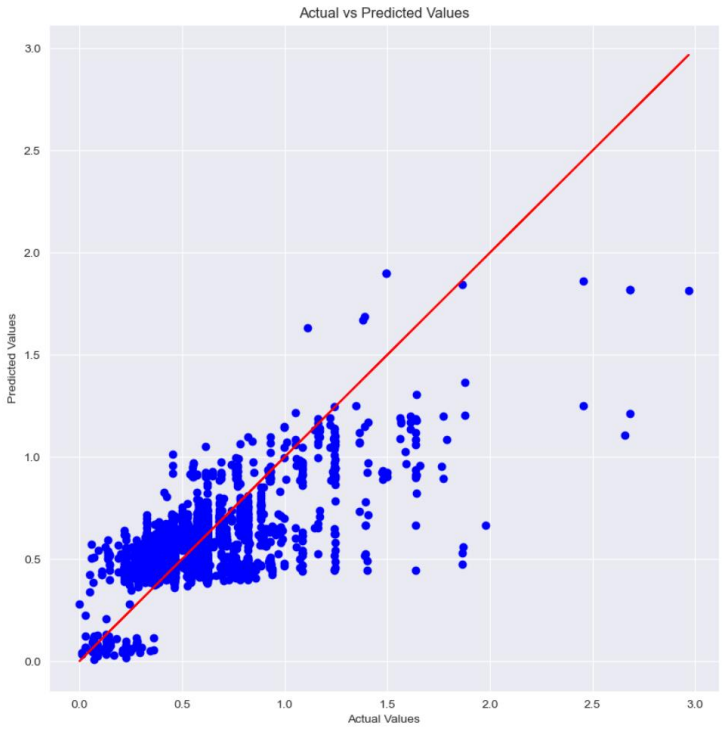

Fig. 41: Actual vs Predicted Values in the final regression model.

Here are the metrics of the final regression model:

```
MSE:0.04
MAE:0.14
RMSE: 0.19
R2:0.48
Adjusted R2:0.48
```

Confidence Interval Analysis:

Confidence interval analysis in regression is a crucial aspect of regression diagnostics, offering a range within which we can be confident that the true population parameter lies. For each coefficient estimated by the regression model, a confidence interval is constructed to provide an estimated range of values with a specified probability. Typically, a 95% confidence interval is used, which implies that if we were to take many random samples from the population and construct intervals in the same way, approximately 95% of these intervals would contain the true value of the parameter.

The width of a confidence interval provides insight into the precision of the estimate: narrower intervals suggest more precise estimates. Confidence intervals that are too wide can indicate a lack of data or high variability in the data. In addition to individual parameter estimates, OLS regression also provides a confidence interval for the prediction, giving a range that likely contains the mean response for given values of the predictors. This aids in understanding the certainty of predictions made by the model. These intervals are constructed based on the standard error of the estimate, reflecting the dispersion of the data points around the regression line.

Here is the sample of the confidence interval (95%) in the model:

```
Confidence Intervals:
                                0     1
const                        1.30  1.83
Number_of_Vehicles           0.80  0.95
Number_of_Casualties         0.43  0.56
Speed_limit                  0.01  0.05
Age_of_Driver                0.04  0.07
Age_of_Vehicle               0.09  0.15
```

Stepwise Regression and Adjusted R-square Analysis:

Backward stepwise regression is a methodical approach for feature selection in which the full model, containing all potential predictors, is considered first. One by one, the least significant predictor is removed from the model, and the model's performance is reassessed. This process is iterated, removing one feature at a time, until the elimination of additional features no longer improves the model performance according to a chosen criterion, like the Akaike Information Criterion (AIC) or the Bayesian Information

Criterion (BIC). This technique simplifies the model by keeping only those features that contribute most significantly to the model's predictive capability, thus addressing potential issues of overfitting and improving model interpretability.

Here are the last samples of removing no. of features using backward stepwise regression:

```
                                OLS Regression Results
==============================================================================================
Dep. Variable:     Fuel_Specific_Accident_Impact_Score   R-squared:                       0.480
Model:                                             OLS   Adj. R-squared:                  0.478
Method:                                  Least Squares   F-statistic:                     369.7
Date:                                 Sat, 02 Dec 2023   Prob (F-statistic):              0.00
Time:                                         10:09:48   Log-Likelihood:                3220.8
No. Observations:                                12464   AIC:                            -6378.
Df Residuals:                                    12432   BIC:                            -6140.
Df Model:                                           31
Covariance Type:                               nonrobust
```

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.5592	0.135	11.523	0.000	1.294	1.824
Number_of_Vehicles	0.8649	0.038	22.571	0.000	0.790	0.940
Number_of_Casualties	0.4891	0.031	15.698	0.000	0.428	0.550
Speed_limit	0.0314	0.009	3.322	0.001	0.013	0.050
Age_of_Driver	0.0505	0.008	6.536	0.000	0.035	0.066
Age_of_Vehicle	0.1229	0.016	7.689	0.000	0.092	0.154
1st_Road_Class_B	-0.0131	0.005	-2.618	0.009	-0.023	-0.003
1st_Road_Class_Motorway	-0.0549	0.020	-2.702	0.007	-0.095	-0.015
Road_Type_Slip road	-0.0402	0.017	-2.422	0.015	-0.073	-0.008
Junction_Detail_Slip road	0.0448	0.012	3.702	0.000	0.021	0.068
2nd_Road_Class_B	0.0210	0.007	3.192	0.001	0.008	0.034
Light_Conditions_Daylight	-0.0408	0.010	-3.948	0.000	-0.061	-0.021
Light_Conditions_Light	-0.0346	0.011	-3.236	0.001	-0.056	-0.014
Road_Surface_Conditions_Wet	-0.0114	0.004	-2.756	0.006	-0.020	-0.003
Vehicle_Type_Bus	-0.4839	0.051	-9.456	0.000	-0.584	-0.384
Vehicle_Type_Car	-0.2001	0.024	-8.440	0.000	-0.247	-0.154
Vehicle_Type_Motorcycle	-0.5741	0.026	-22.328	0.000	-0.624	-0.524
Vehicle_Type_Taxi	-0.0768	0.025	-3.055	0.002	-0.126	-0.028
Vehicle_Type_Van	-0.1100	0.025	-4.467	0.000	-0.158	-0.062
Vehicle_Manoeuvre_Going ahead	0.0128	0.004	3.488	0.000	0.006	0.020
Vehicle_Manoeuvre_Reversing	0.0493	0.014	3.580	0.000	0.022	0.076
Junction_Location_Mid	0.0093	0.004	2.593	0.010	0.002	0.016
1st_Point_of_Impact_Front	0.0156	0.004	3.508	0.000	0.007	0.024
1st_Point_of_Impact_Nearside	0.0218	0.006	3.620	0.000	0.010	0.034
1st_Point_of_Impact_Offside	0.0163	0.006	2.788	0.005	0.005	0.028
Propulsion_Code_Hybrid electric	-0.1657	0.027	-6.073	0.000	-0.219	-0.112
Propulsion_Code_Petrol	-0.1649	0.004	-39.366	0.000	-0.173	-0.157
Casualty_Class_Passenger	-0.0160	0.004	-3.896	0.000	-0.024	-0.008
Casualty_Class_Pedestrian	0.0286	0.007	4.201	0.000	0.015	0.042
Sex_of_Casualty_Male	0.0087	0.004	2.465	0.014	0.002	0.016
Casualty_Severity_Serious	-0.3535	0.133	-2.664	0.008	-0.614	-0.093
Casualty_Severity_Slight	-0.8201	0.133	-6.186	0.000	-1.080	-0.560

Omnibus:	5987.542	Durbin-Watson:	2.029
Prob(Omnibus):	0.000	Jarque-Bera (JB):	80858.022
Skew:	1.964	Prob(JB):	0.00
Kurtosis:	14.843	Cond. No.	314.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
Removed Feature: Road_Type_Slip road
No. of Features Removed: 92

OLS Regression Results

Dep. Variable:	Fuel_Specific_Accident_Impact_Score	R-squared:	0.479
Model:	OLS	Adj. R-squared:	0.478
Method:	Least Squares	F-statistic:	381.7
Date:	Sat, 02 Dec 2023	Prob (F-statistic):	0.00
Time:	10:09:48	Log-Likelihood:	3217.9
No. Observations:	12464	AIC:	-6374.
Df Residuals:	12433	BIC:	-6143.
Df Model:	30		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.5630	0.135	11.550	0.000	1.298	1.828
Number_of_Vehicles	0.8713	0.038	22.786	0.000	0.796	0.946
Number_of_Casualties	0.4906	0.031	15.744	0.000	0.429	0.552
Speed_limit	0.0311	0.009	3.292	0.001	0.013	0.050
Age_of_Driver	0.0508	0.008	6.571	0.000	0.036	0.066
Age_of_Vehicle	0.1228	0.016	7.681	0.000	0.091	0.154
1st_Road_Class_B	-0.0131	0.005	-2.600	0.009	-0.023	-0.003
1st_Road_Class_Motorway	-0.0576	0.020	-2.836	0.005	-0.097	-0.018
Junction_Detail_Slip road	0.0367	0.012	3.157	0.002	0.014	0.060
2nd_Road_Class_B	0.0211	0.007	3.217	0.001	0.008	0.034
Light_Conditions_Daylight	-0.0410	0.010	-3.968	0.000	-0.061	-0.021
Light_Conditions_Light	-0.0350	0.011	-3.266	0.001	-0.056	-0.014
Road_Surface_Conditions_Wet	-0.0111	0.004	-2.692	0.007	-0.019	-0.003
Vehicle_Type_Bus	-0.4838	0.051	-9.452	0.000	-0.584	-0.383
Vehicle_Type_Car	-0.2004	0.024	-8.452	0.000	-0.247	-0.154
Vehicle_Type_Motorcycle	-0.5741	0.026	-22.325	0.000	-0.624	-0.524
Vehicle_Type_Taxi	-0.0772	0.025	-3.070	0.002	-0.127	-0.028
Vehicle_Type_Van	-0.1102	0.025	-4.477	0.000	-0.159	-0.062
Vehicle_Manoeuvre_Going ahead	0.0128	0.004	3.509	0.000	0.006	0.020
Vehicle_Manoeuvre_Reversing	0.0497	0.014	3.611	0.000	0.023	0.077
Junction_Location_Mid	0.0096	0.004	2.660	0.008	0.003	0.017
1st_Point_of_Impact_Front	0.0157	0.004	3.519	0.000	0.007	0.024
1st_Point_of_Impact_Nearside	0.0222	0.006	3.673	0.000	0.010	0.034
1st_Point_of_Impact_Offside	0.0167	0.006	2.856	0.004	0.005	0.028
Propulsion_Code_Hybrid electric	-0.1654	0.027	-6.059	0.000	-0.219	-0.112
Propulsion_Code_Petrol	-0.1648	0.004	-39.324	0.000	-0.173	-0.157
Casualty_Class_Passenger	-0.0161	0.004	-3.911	0.000	-0.024	-0.008
Casualty_Class_Pedestrian	0.0289	0.007	4.248	0.000	0.016	0.042
Sex_of_Casualty_Male	0.0089	0.004	2.515	0.012	0.002	0.016
Casualty_Severity_Serious	-0.3580	0.133	-2.697	0.007	-0.618	-0.098
Casualty_Severity_Slight	-0.8245	0.133	-6.218	0.000	-1.084	-0.565

```
==============================================================================
Omnibus:                     5986.825   Durbin-Watson:                   2.029
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            80838.376
Skew:                           1.964   Prob(JB):                         0.00
Kurtosis:                      14.842   Cond. No.                          314.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
Removed Feature: Sex_of_Casualty_Male
```

Here is the no. of features removed through backward stepwise regression:

```
No. of Features Removed: 93
```

Here is the adjusted r2 after removal of features:

	Adj. R2
Road_Type_One way street	0.48
Junction_Location_Leaving	0.48
Propulsion_Code_Heavy oil	0.48
Junction_Detail_Entrance	0.48
1st_Point_of_Impact_Unknown	0.48

Here is the plot of AIC, BIC, and adj. r2 in 1 plot:

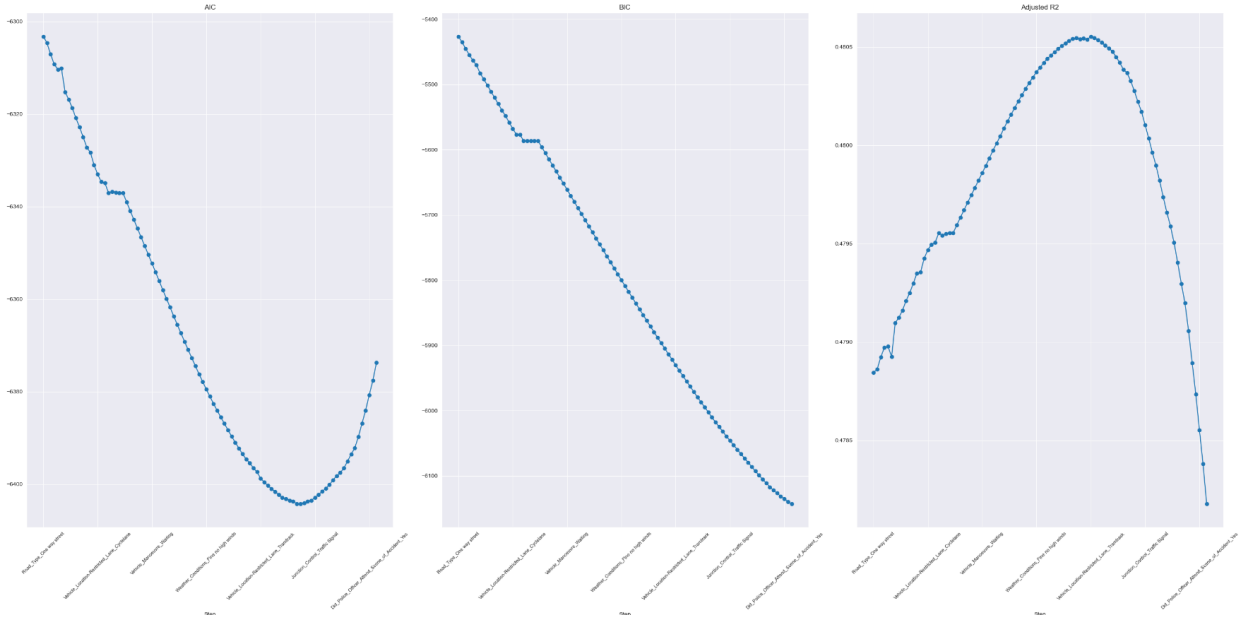

Fig. 42: plot of AIC, BIC, and adj. r2 in 1 plot

Phase 3: Classification Analysis

Classification in machine learning is a type of supervised learning technique where the goal is to predict the categorical class labels of new instances, based on past observations. In this process, the algorithm is trained on a labeled dataset, where each instance belongs to a specific category. This training involves the algorithm learning the association between the input features of the data and their corresponding classes. Common classification algorithms include logistic regression, decision trees, random forests, support vector machines (SVMs), and neural networks, each with unique strengths and suited for different types of data and complexities.

The performance of these algorithms is evaluated using metrics like accuracy, precision, recall, and the F1 score. A critical aspect of classification is ensuring the model's generalizability to new, unseen data, avoiding overfitting to the training data.

Here in this project, we have used different classification algorithms like decision tree, logistic regression, k-neighbors, support vector machines, multilayer perceptron, and much more. Moreover, for the validation of each of the models, following metrics are used:

- Accuracy
- Precision Score
- Recall Score
- Specificity Score
- F1 Score
- AUC Score and ROC Curves
- Stratified K-Fold Cross-Validation Scores

Let’s start by performing the models to predict the target variable, i.e., Accident_Severity, which contains three classes: Slight, Serious, and Fatal. It means it is a multi-class classification problem.

Decision Tree:

The decision tree is a versatile machine learning algorithm which works by creating a tree-like model of decisions, where each node represents a feature in a dataset, and the branches represent decision rules, leading to a prediction at the leaves. This classifier is known for its simplicity, interpretability, and ability to handle both numerical and categorical data effectively.

Here are the results of the baseline Decision Tree:

```
Performance of the baseline decision tree on the test set:
Accuracy of the model: 0.97
Confusion Matrix:
 [[1390    8    0]
 [   6 1357   46]
 [   6   68 1327]]
Precision Score: 0.97
Recall Score: 0.97
Specificity of class 0: 1.00
Specificity of class 1: 0.97
Specificity of class 2: 0.98
F1 Score: 0.97
Fold 1 accuracy: 0.9793250950570342
Fold 2 accuracy: 0.9743346007604563
Fold 3 accuracy: 0.9702946768060836
Fold 4 accuracy: 0.9695817490494296
Fold 5 accuracy: 0.9769431899215593
The mean of AUC for this multi-label classification is 97.61%
```

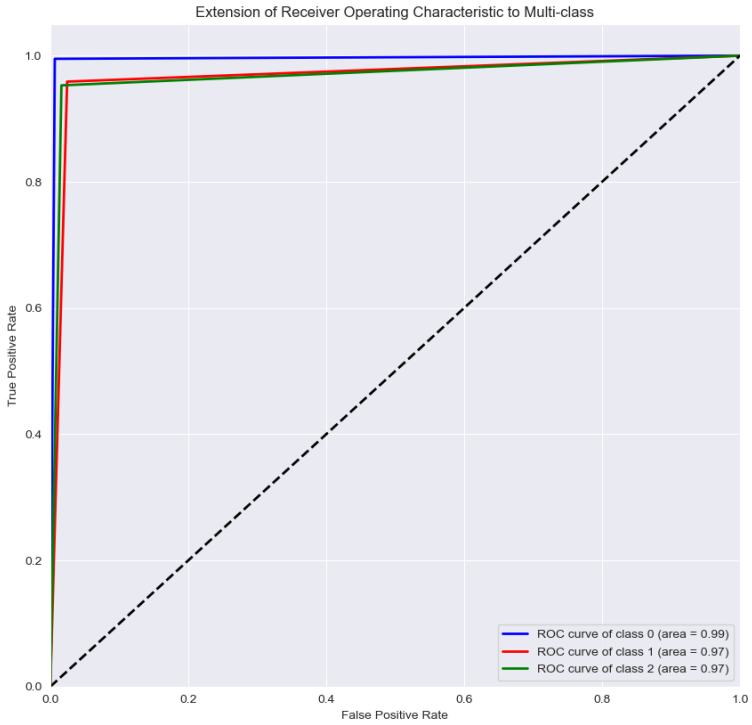

Fig. 43: ROC Curve: Baseline Decision Tree

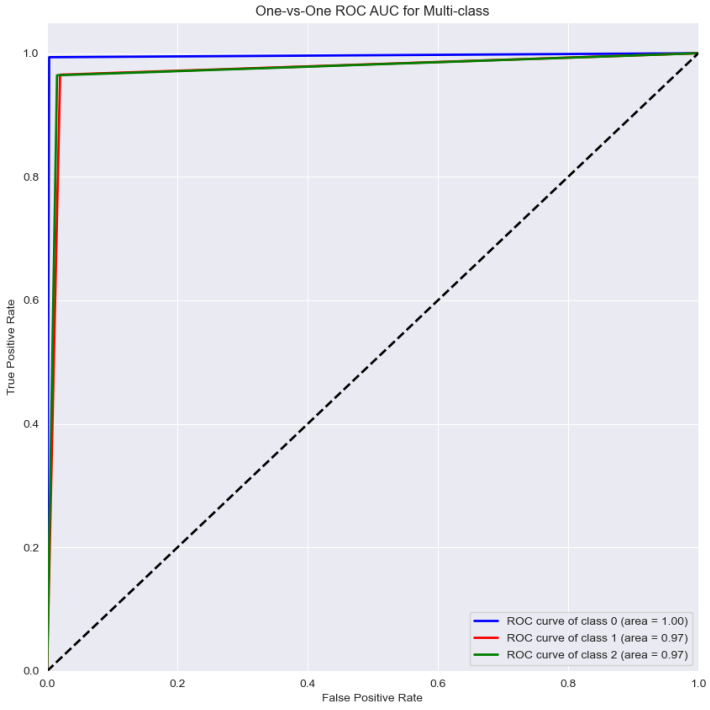

Fig. 44: ROC Curve: One-vs-One- Baseline Decision Tree

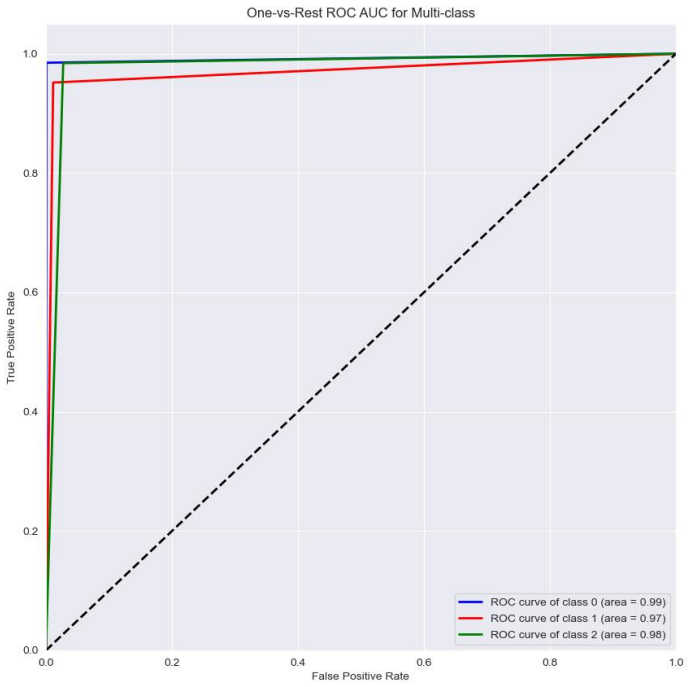

Fig. 45: ROC Curve: One-vs-Rest- Baseline Decision Tree

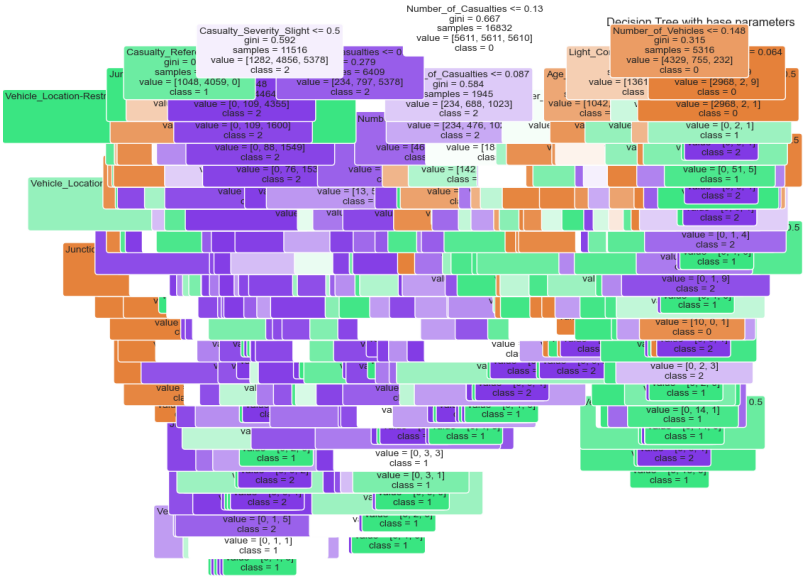

Fig. 46: Baseline Decision Tree Plot

The decision tree is put through a grid search for finding its optimal parameters for improving its performance. Then after finding the best parameters, the decision tree is computed again. Here are the results:

```
 DecisionTreeClassifier(max_depth=9, max_features=4, min_samples_split=5,
                       random_state=5805)
Best Score:
 0.9273950568068215
Best Paramters:
 {'criterion': 'gini', 'max_depth': 9, 'max_features': 4, 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter': 'best'}
Performance of the fine tuned decision tree on the test set:
Accuracy of the model: 0.93
Confusion Matrix:
 [[1323   69    6]
 [  50 1243  116]
 [   7   30 1364]]
Precision Score: 0.93
Recall Score: 0.93
Specificity of class 0: 0.98
Specificity of class 1: 0.96
Specificity of class 2: 0.96
F1 Score: 0.93
Fold 1 accuracy: 0.783032319391635
Fold 2 accuracy: 0.6936787072243346
Fold 3 accuracy: 0.6986692015209125
Fold 4 accuracy: 0.6347433460076045
Fold 5 accuracy: 0.7516044687425719
```

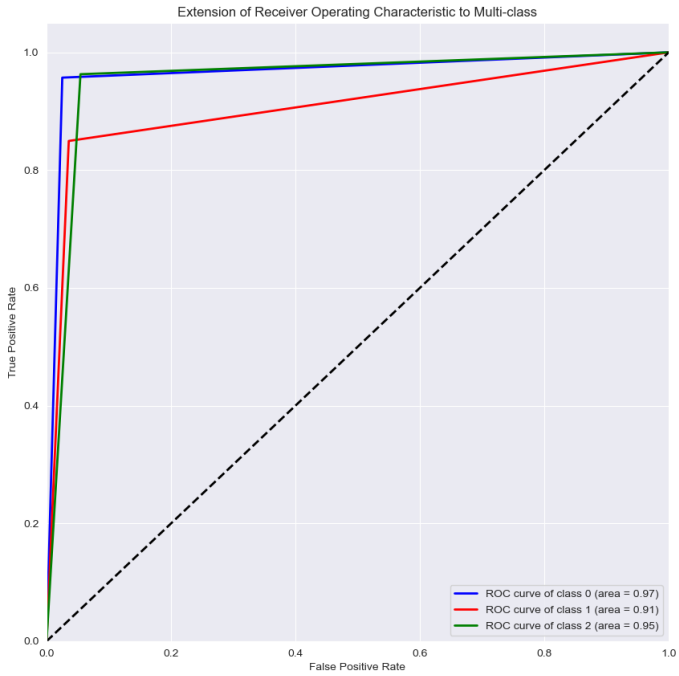

Fig. 47: ROC Curve: Fine Tuned Decision Tree

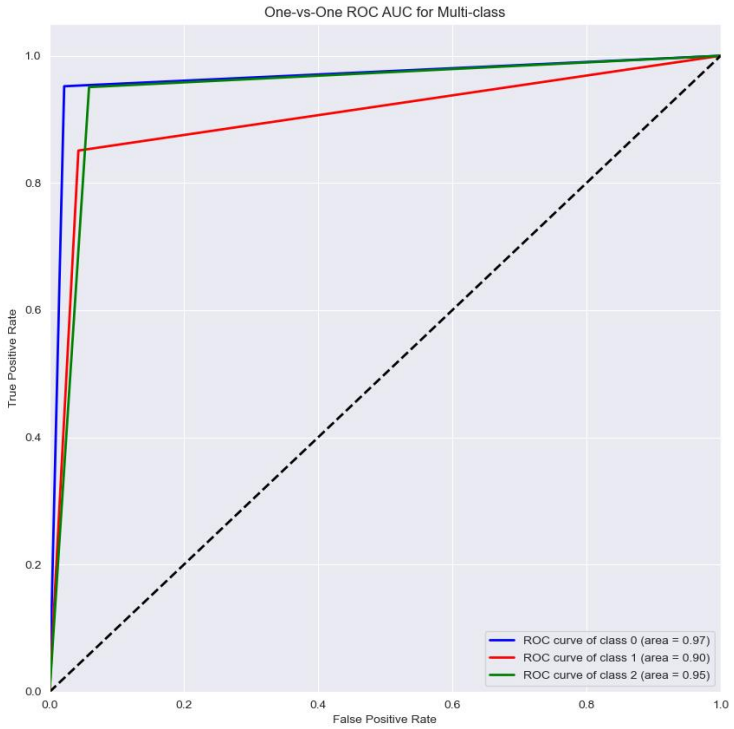

Fig. 48: ROC Curve: One-vs-One- Fine Tuned Decision Tree

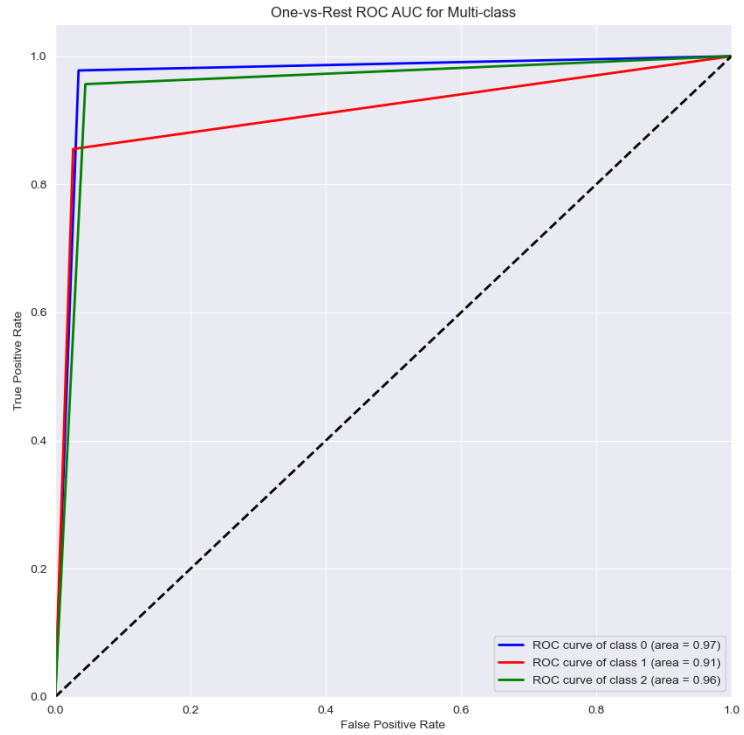

Fig. 49: ROC Curve: One-vs-Rest- Fine Tuned Decision Tree

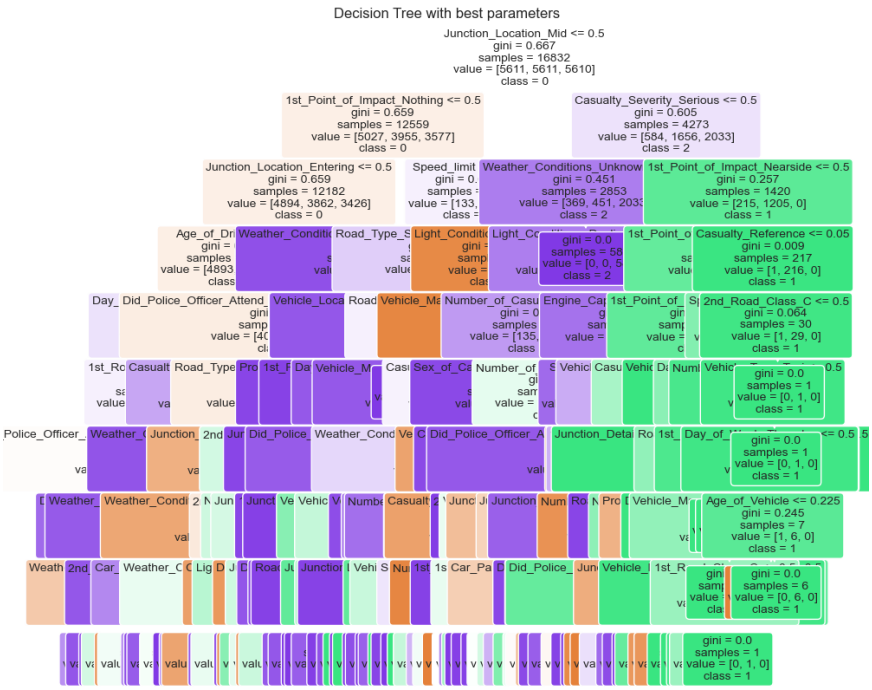

Fig. 50: Fine Tuned Decision Tree plot

The decision tree is put to a grid search of alpha parameter, which is used for cost-complexity pruning, a technique to prevent overfitting. The grid search computes to the optimal alpha, which is put again as a parameter in decision tree.

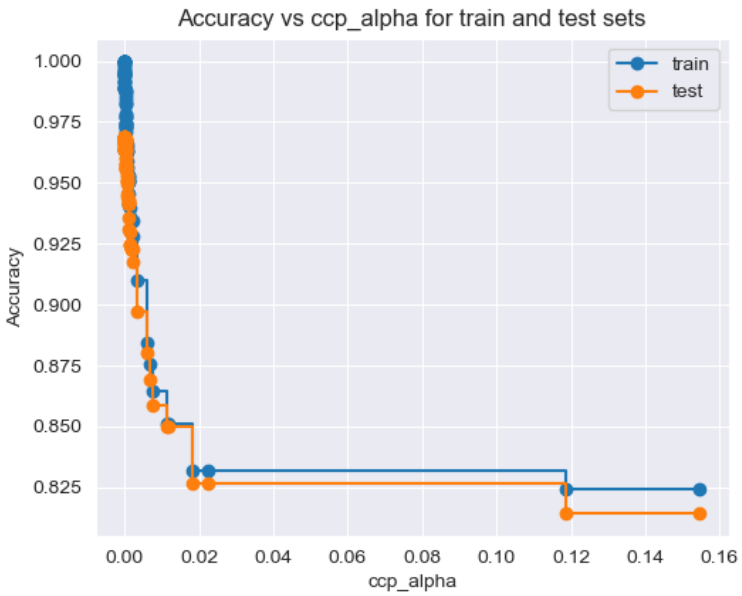

Fig. 51: Finding the optimal alpha.

Here are the results of the optimal cost-complexity parameter, for post-pruning:

```
Optimal ccp_alpha: 7.7930789408303 84e-05
Performance of the optimal ccp alpha decision tree on the test set:
Accuracy of the model: 0.97
Confusion Matrix:
 [[1390    8    0]
 [   8 1349   52]
 [   7   52 1342]]
Precision Score: 0.97
Recall Score: 0.97
Specificity of class 0: 0.99
Specificity of class 1: 0.98
Specificity of class 2: 0.98
F1 Score: 0.97
Fold 1 accuracy: 0.9798003802281369
Fold 2 accuracy: 0.9743346007604563
Fold 3 accuracy: 0.9724334600760456
Fold 4 accuracy: 0.972671102661597
Fold 5 accuracy: 0.9757546945566913
The mean of AUC for this multi-label classification is 97.74%
```

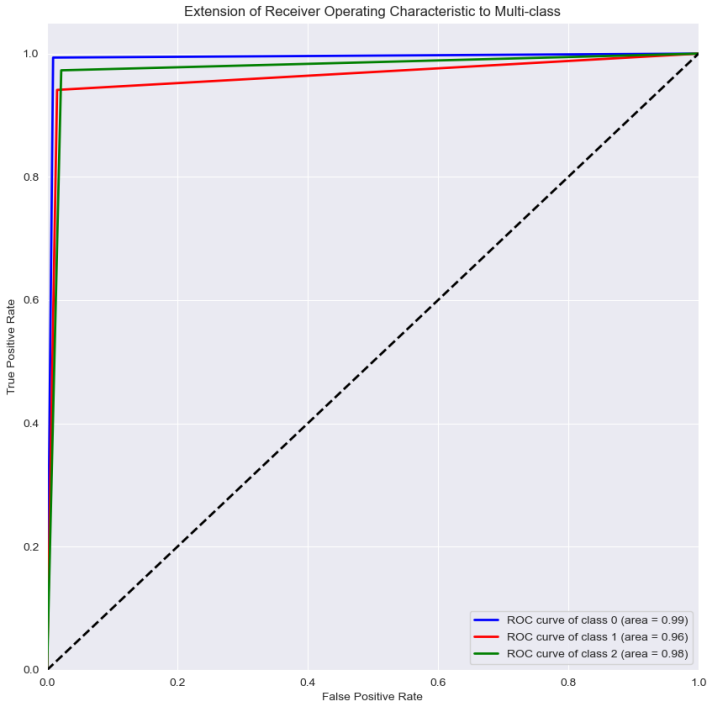

Fig. 52: ROC Curve: Post-Pruned Decision Tree

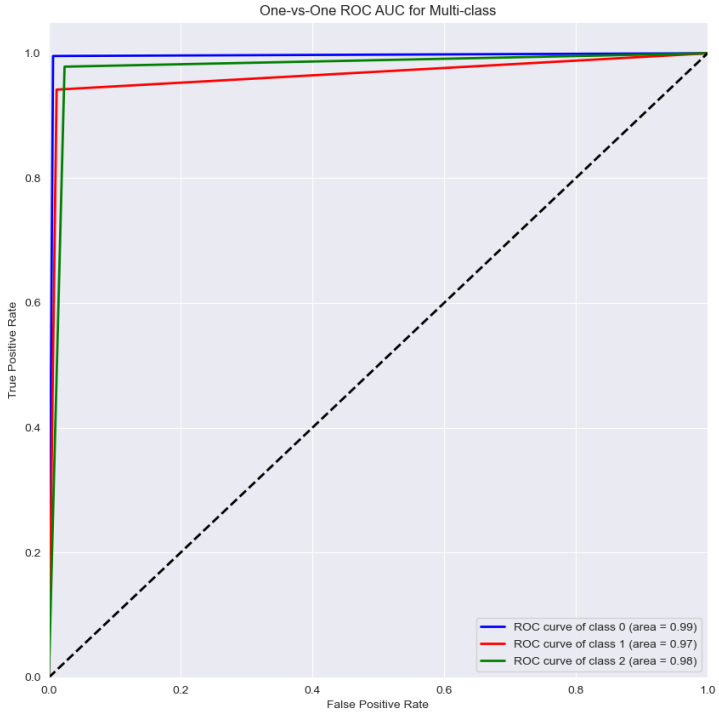

Fig. 53: ROC Curve: One-vs-One- Post-Pruned Decision Tree

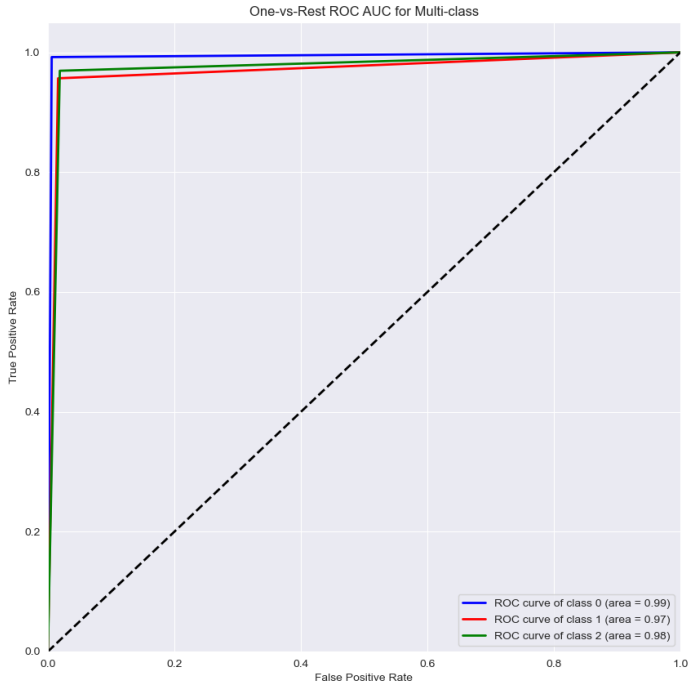

Fig. 54: ROC Curve: One-vs-Rest- Post-Pruned Decision Tree

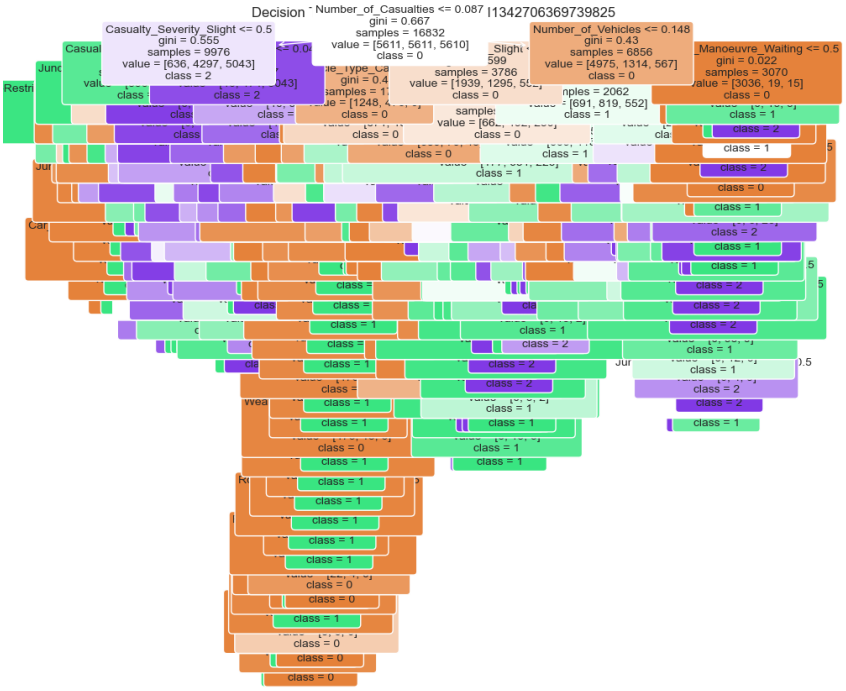

Fig. 55: Post-Pruned Decision Tree plot

Logistic Regression:

Logistic Regression is a statistical method used for binary and multiclass classification, which predicts the probability of a binary outcome by applying a logistic function to a linear combination of predictors. It excels in situations where the dependent variable is categorical, offering a simple yet effective approach for classification problems.

Here are the results of the baseline Logistic Regression:

```
Performance of the baseline logistic regression on the test set:
Accuracy of the model: 0.9
Confusion Matrix:
 [[1313   82    3]
 [ 109 1132  168]
 [  12   49 1340]]
Precision Score: 0.9
Recall Score: 0.9
Specificity of class 0: 0.96
Specificity of class 1: 0.95
Specificity of class 2: 0.94
F1 Score: 0.9
Fold 1 accuracy: 0.9653041825095057
Fold 2 accuracy: 0.9667300380228137
Fold 3 accuracy: 0.9700570342205324
Fold 4 accuracy: 0.964115969581749
Fold 5 accuracy: 0.9664844307107202
The mean of AUC for this multi-label classification is 92.47%
```

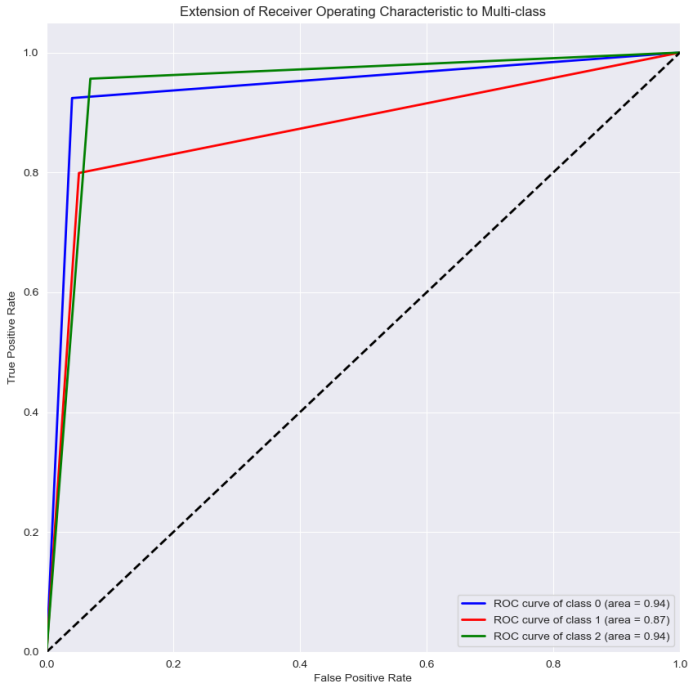

Fig. 56: ROC Curve: Baseline Logistic Regression

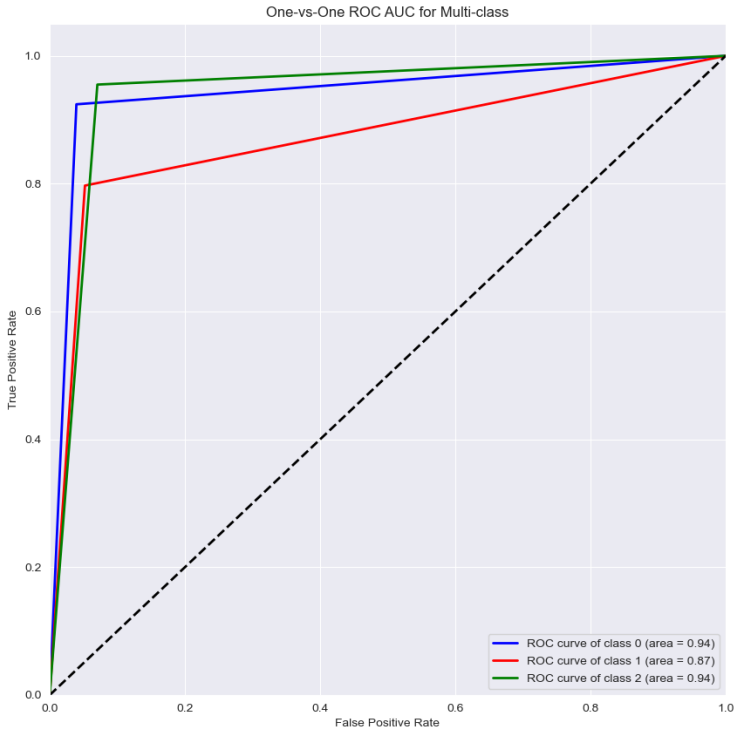

Fig. 57: ROC Curve: One-vs-One- Baseline Logistic Regression

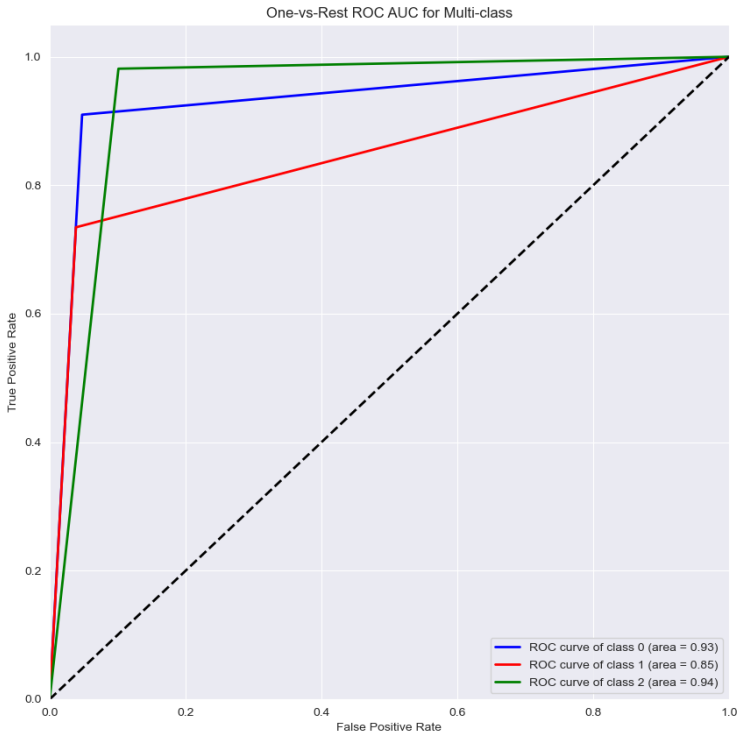

Fig. 58: ROC Curve: One-vs-Rest- Baseline Logistic Regression

The logistic regression is put through a grid search for finding its optimal parameters for improving its performance. Then after finding the best parameters, the logistic regression is computed again. Here are the results:

```
Best Estimator:
 LogisticRegression(C=10, max_iter=2000, random_state=5805, solver='newton-cg')
Best Score:
 0.902501420148479
Best Paramters:
 {'C': 10, 'max_iter': 2000, 'penalty': 'l2', 'solver': 'newton-cg'}
Performance of the fine tuned logistic regression on the test set:
Accuracy of the model: 0.9
Confusion Matrix:
 [[1314   82    2]
 [ 110 1142  157]
 [  10   58 1333]]
Precision Score: 0.9
Recall Score: 0.9
Specificity of class 0: 0.96
Specificity of class 1: 0.95
Specificity of class 2: 0.94
F1 Score: 0.9
Fold 1 accuracy: 0.9681558935361216
Fold 2 accuracy: 0.968393536121673
Fold 3 accuracy: 0.969819391634981
Fold 4 accuracy: 0.9688688212927756
Fold 5 accuracy: 0.9669598288566674
The mean of AUC for this multi-label classification is 92.54%
```

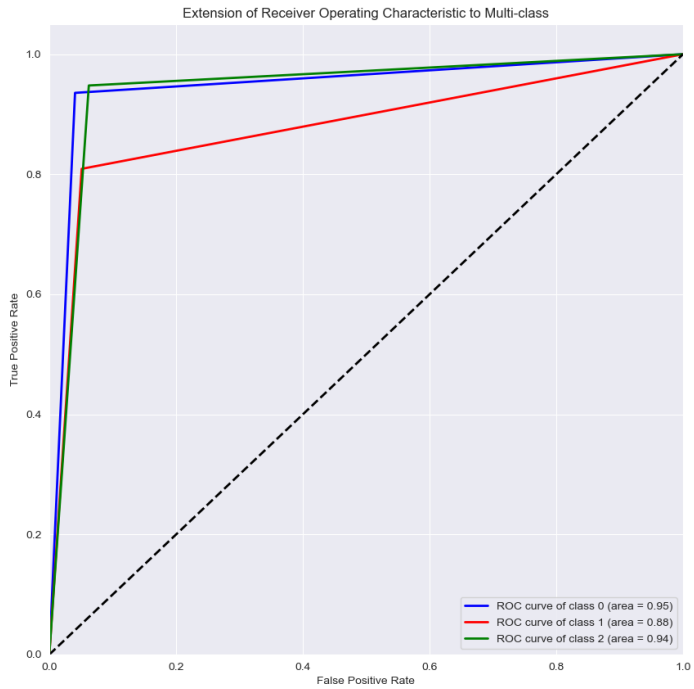

Fig. 59: ROC Curve: Fine Tuned logistic regression

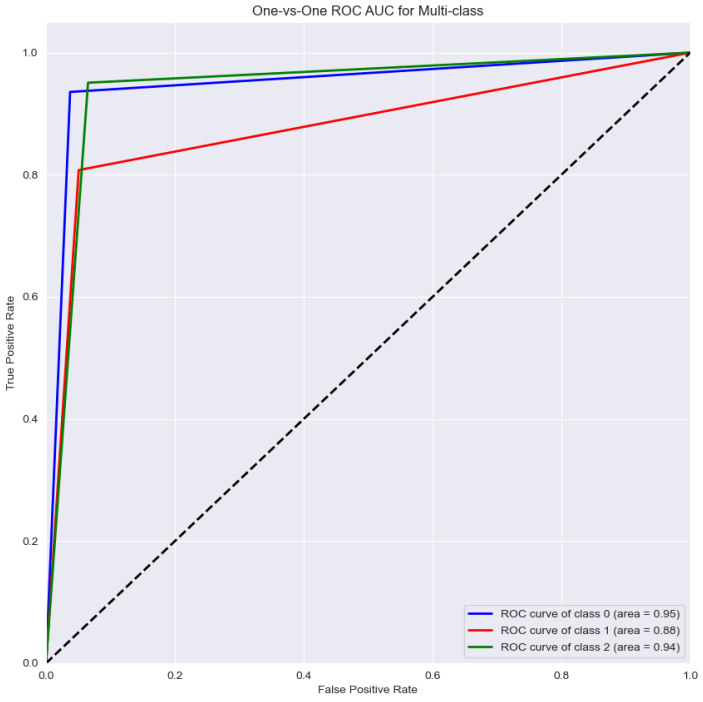

Fig. 60: ROC Curve: One-vs-One- Fine Tuned logistic regression

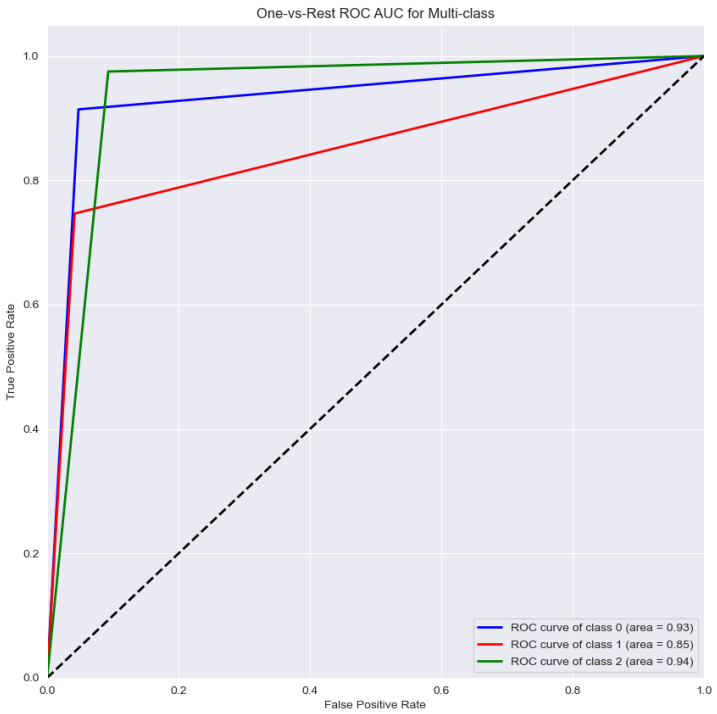

Fig. 61: ROC Curve: One-vs-Rest- Fine Tuned logistic regression

KNN Classifier:

The k-Nearest Neighbors (kNN) classifier is a simple, yet effective machine learning, which operates by identifying the k nearest data points in the feature space to a given query point, and predicts the label based on the majority vote of these neighbors. kNN is non-parametric and instance-based, making it particularly useful in applications where the decision boundary is irregular.

Here are the results of the baseline Logistic Regression:

```
Performance of the baseline knn classifier on the test set:
Accuracy of the model: 0.96
Confusion Matrix:
 [[1396    2    0]
 [  10 1339   60]
 [   7   91 1303]]
Precision Score: 0.96
Recall Score: 0.96
Specificity of class 0: 0.99
Specificity of class 1: 0.97
Specificity of class 2: 0.98
F1 Score: 0.96
Fold 1 accuracy: 0.9405893536121673
Fold 2 accuracy: 0.9505703422053232
Fold 3 accuracy: 0.9486692015209125
Fold 4 accuracy: 0.9453422053231939
Fold 5 accuracy: 0.9517470881863561
The mean of AUC for this multi-label classification is 96.97%
```

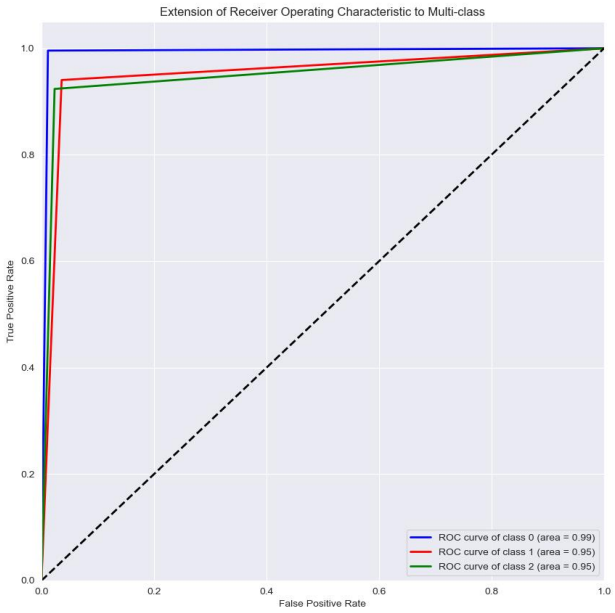

Fig. 62: ROC Curve: Baseline KNN classifier

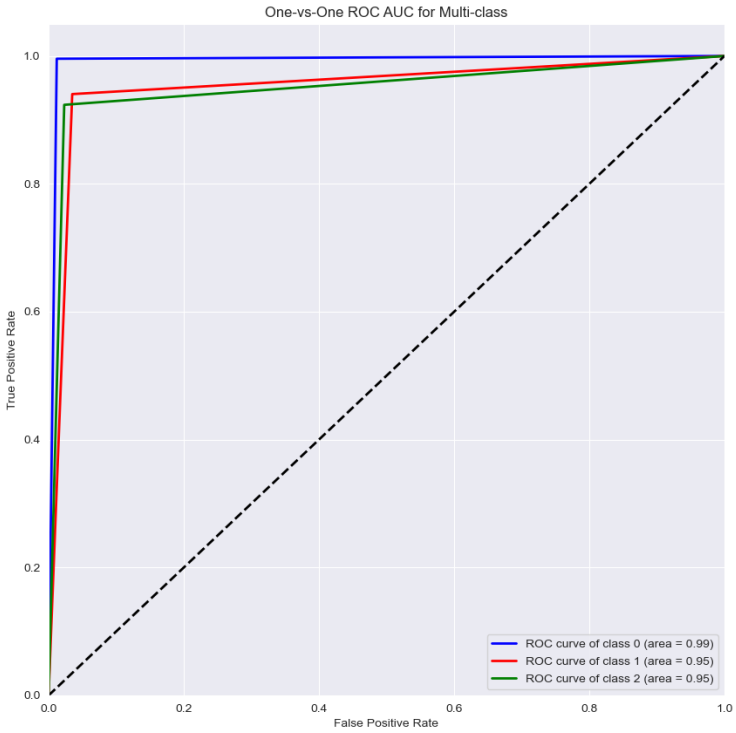

Fig. 63: ROC Curve: One-vs-One- Baseline KNN classifier

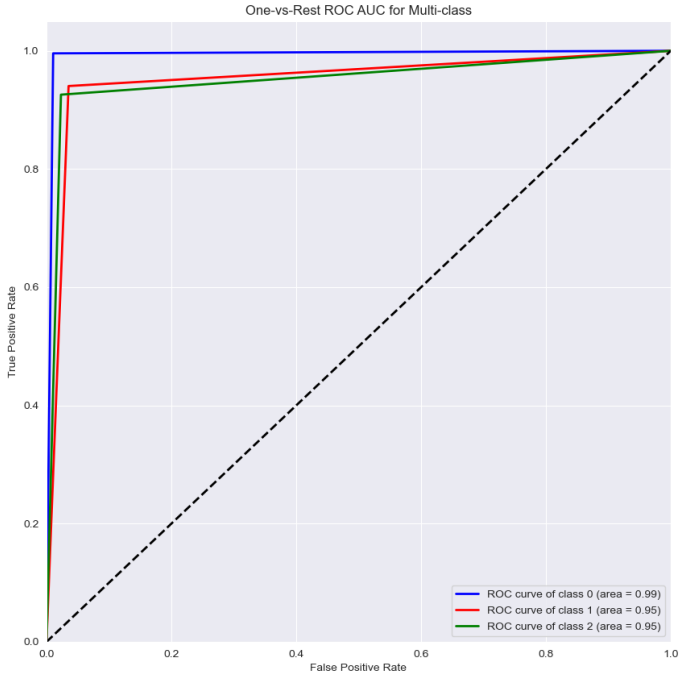

Fig. 64: ROC Curve: One-vs-Rest- Baseline KNN classifier

The KNN classifier is put through a grid search for finding its optimal parameters for improving its performance. Then after finding the best parameters, the KNN classifier is computed again. Here are the results:

```
Best Estimator:
 KNeighborsClassifier(n_neighbors=1)
Best Score:
 0.966430813489637
Best Paramters:
 {'n_neighbors': 1}
Performance of the fine tuned knn classifier on the test set:
Accuracy of the model: 0.97
Confusion Matrix:
 [[1396    2    0]
 [   2 1372   35]
 [   3   72 1326]]
Precision Score: 0.97
Recall Score: 0.97
Specificity of class 0: 1.00
Specificity of class 1: 0.97
Specificity of class 2: 0.99
F1 Score: 0.97
Fold 1 accuracy: 0.9591254752851711
Fold 2 accuracy: 0.9657794676806084
Fold 3 accuracy: 0.9653041825095057
Fold 4 accuracy: 0.9667300380228137
Fold 5 accuracy: 0.967910625148562
The mean of AUC for this multi-label classification is 97.97%
```

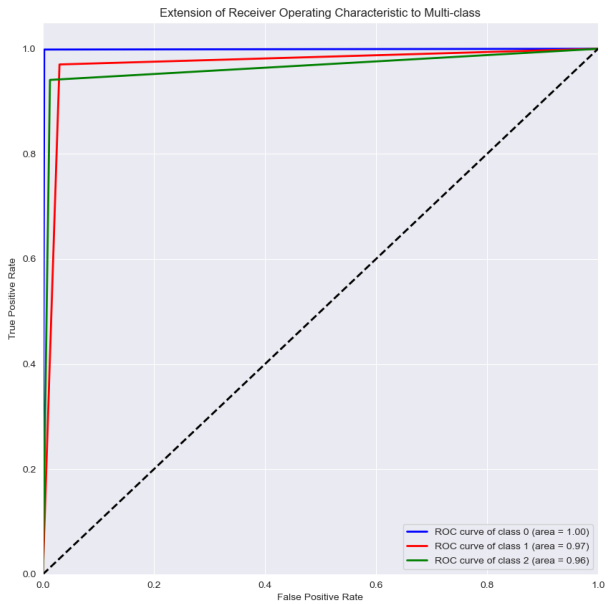

Fig. 65: ROC Curve: Fine Tuned KNN classifier

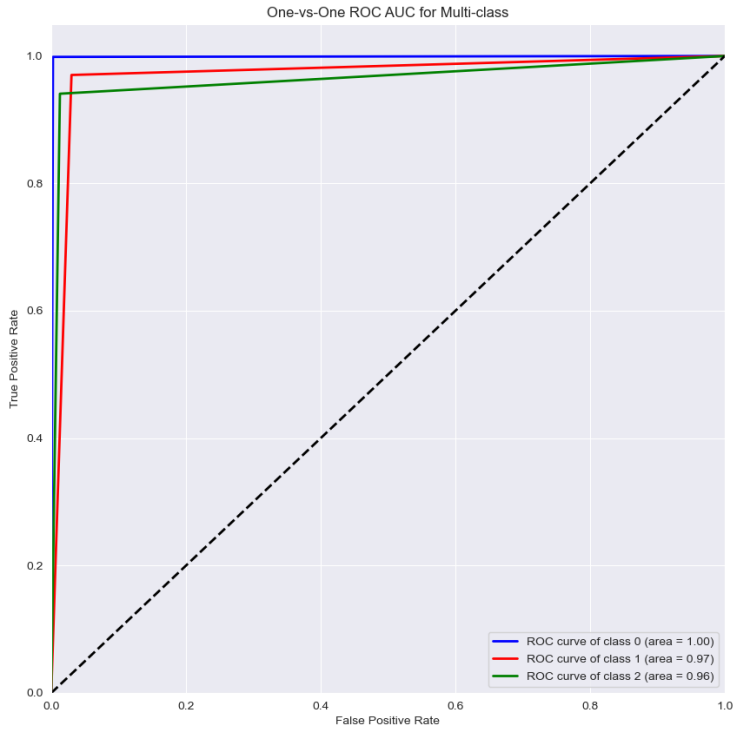

Fig. 66: ROC Curve: One-vs-One- Fine Tuned KNN classifier

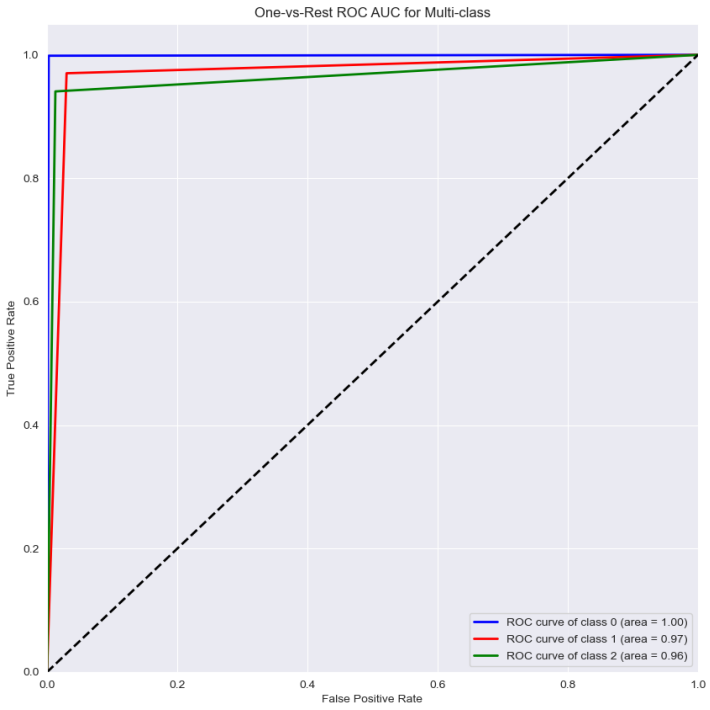

Fig. 67: ROC Curve: One-vs-Rest- Fine Tuned KNN classifier

Fig. 68: ROC Curve: Best k in KNN classifier

Gaussian Naïve Bayes:

Gaussian Naive Bayes is a variant of the Naive Bayes algorithm, particularly suited for continuous data with an assumption that features follow a normal distribution. It operates under the principle of Bayes' theorem with the 'naive' assumption of conditional independence between every pair of features given the class label. This simplicity allows for quick and efficient model training, especially in large datasets.

Here are the results of the baseline Gaussian Naive Bayes:

```
Performance of the baseline gaussian naive bayes on the test set:
Accuracy of the model: 0.79
Confusion Matrix:
 [[ 925  292  181]
 [  92 1016  301]
 [  25    0 1376]]
Precision Score: 0.8
Recall Score: 0.79
Specificity of class 0: 0.96
Specificity of class 1: 0.90
Specificity of class 2: 0.83
F1 Score: 0.78
Fold 1 accuracy: 0.590541825095057
Fold 2 accuracy: 0.5990969581749049
Fold 3 accuracy: 0.5967205323193916
Fold 4 accuracy: 0.6009980988593155
Fold 5 accuracy: 0.6080342286665082
The mean of AUC for this multi-label classification is 84.12%
```

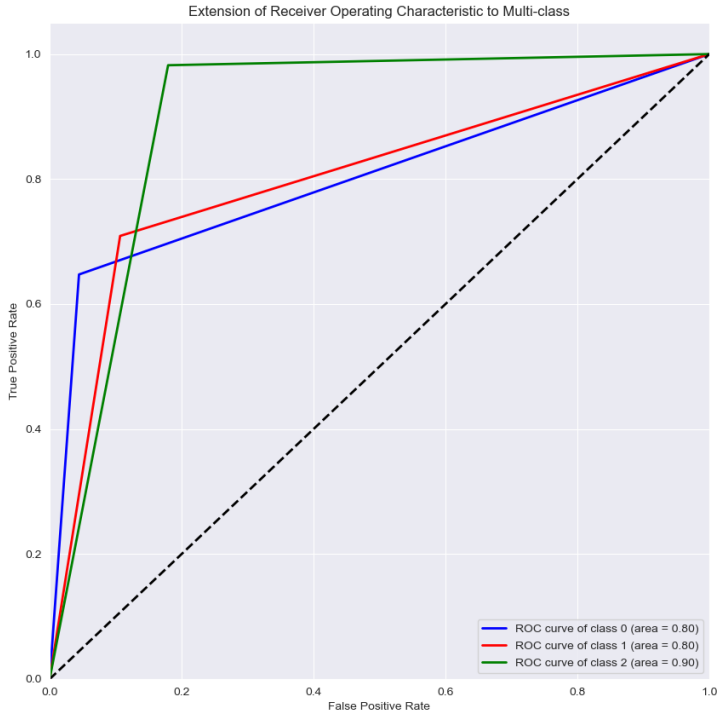

Fig. 69: ROC Curve: Baseline Gaussian Naive Bayes

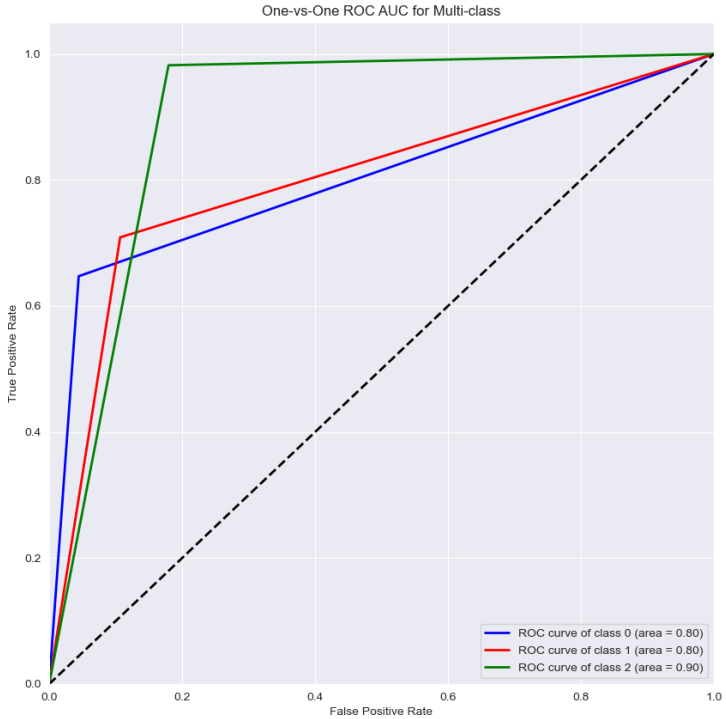

Fig. 70: ROC Curve: One-vs-One- Baseline Gaussian Naive Bayes

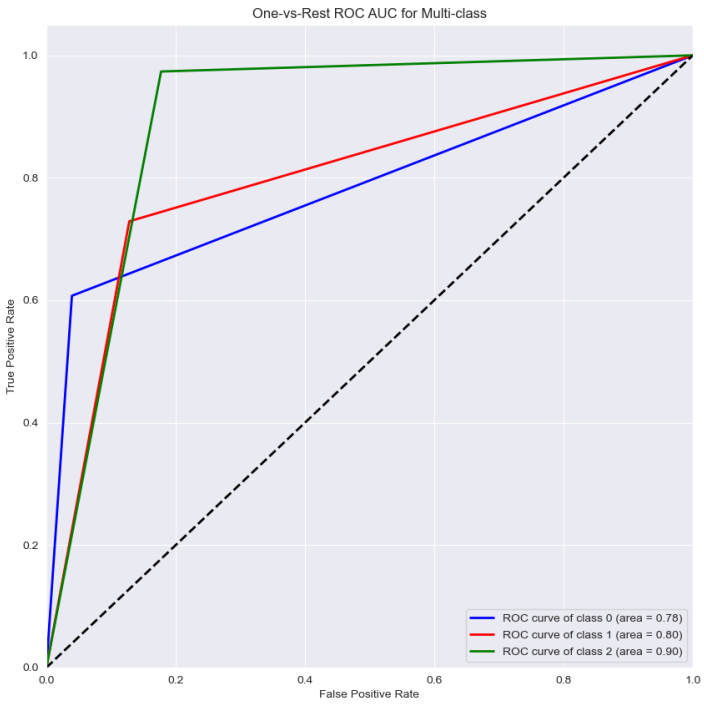

Fig. 71: ROC Curve: One-vs-Rest- Baseline Gaussian Naive Bayes

The Gaussian Naive Bayes is put through a grid search for finding its optimal parameters for improving its performance. Then after finding the best parameters, Gaussian Naive Bayes is computed again. Here are the results:

```
Best Estimator:
 GaussianNB(var_smoothing=0.001)
Best Score:
 0.7682248152836388
Best Paramters:
 {'var_smoothing': 0.001}
Performance of the fine tuned gaussian naive bayes on the test set:
Accuracy of the model: 0.76
Confusion Matrix:
 [[ 802  437  159]
 [  65 1054  290]
 [  38    1 1362]]
Precision Score: 0.78
Recall Score: 0.76
Specificity of class 0: 0.96
Specificity of class 1: 0.84
Specificity of class 2: 0.84
F1 Score: 0.76
Fold 1 accuracy: 0.6634980988593155
Fold 2 accuracy: 0.6758555133079848
Fold 3 accuracy: 0.6720532319391636
Fold 4 accuracy: 0.6708650190114068
Fold 5 accuracy: 0.6838602329450915
The mean of AUC for this multi-label classification is 82.35%
```

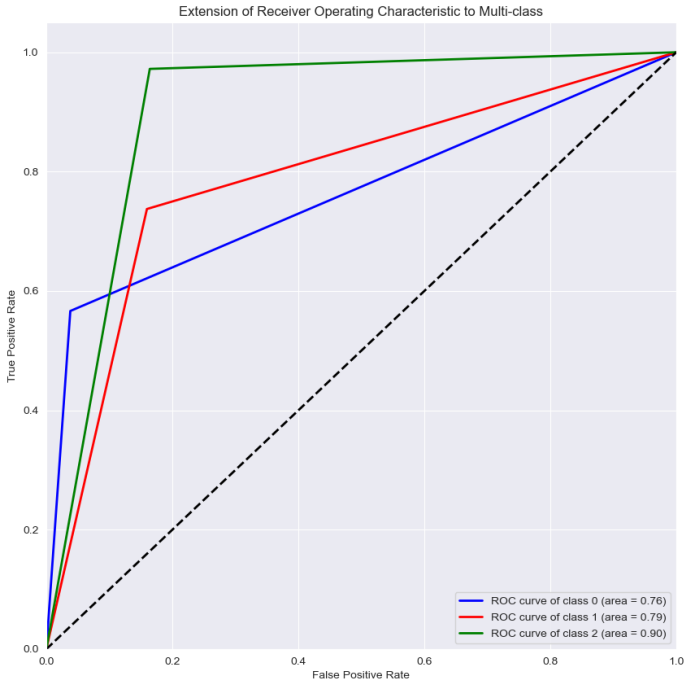

Fig. 72: ROC Curve: Fine Tuned Gaussian Naive Bayes

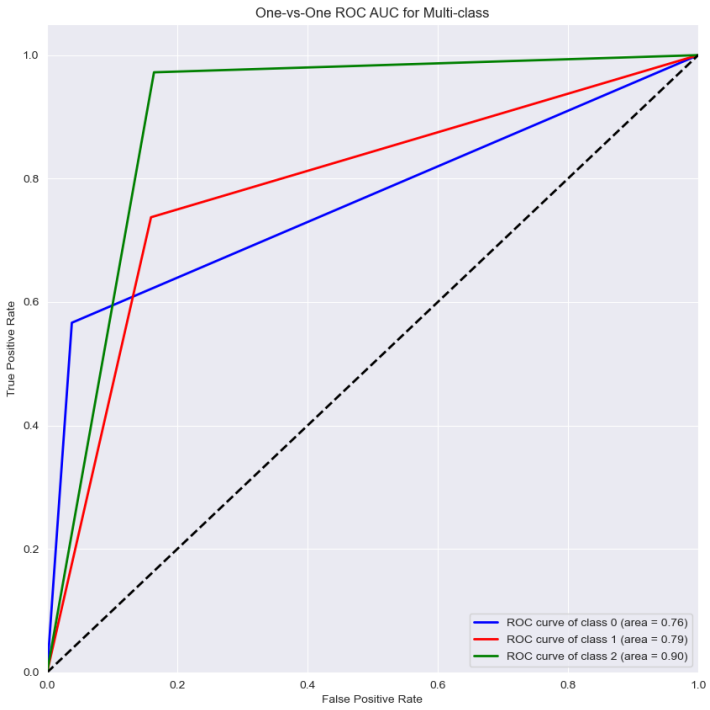

Fig. 73: ROC Curve: One-vs-One- Fine Tuned Gaussian Naive Bayes

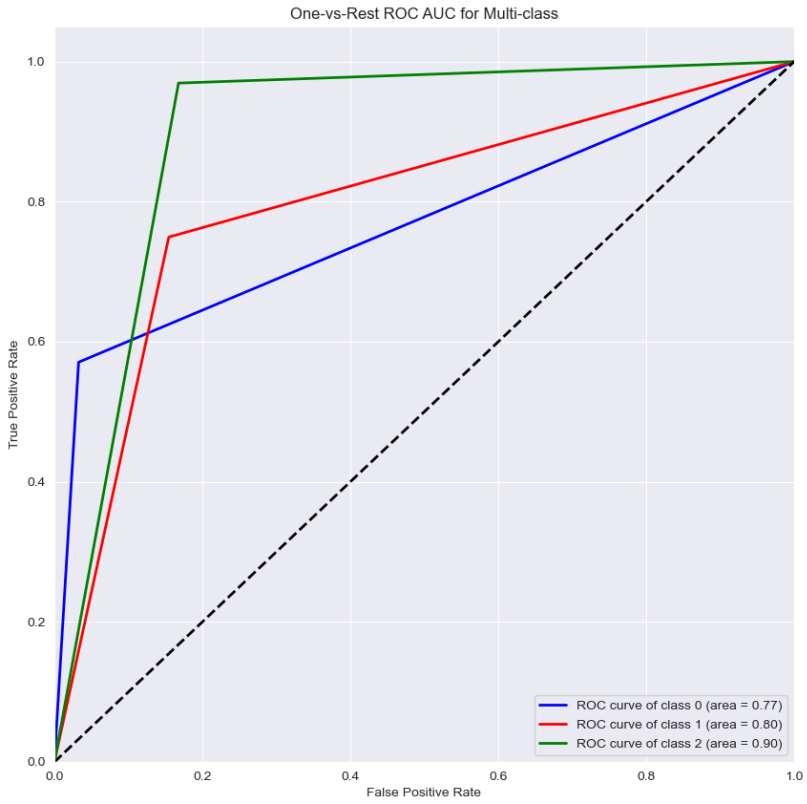

Fig. 74: ROC Curve: One-vs-Rest- Fine Tuned Gaussian Naive Bayes

Support Vector Machine:

Support Vector Machine (SVM) is a powerful and versatile supervised machine learning algorithm, primarily used for classification tasks. It operates by finding the hyperplane that best divides a dataset into classes, with the goal of maximizing the margin between data points of different classes. SVM is particularly effective in high-dimensional spaces and is capable of handling both linear and non-linear classification through the use of kernel functions. Its ability to manage overfitting, even in complex datasets, makes it a popular choice for a variety of applications.

Here are the results of the baseline Support Vector Machine:

```
Performance of the baseline svm on the test set:
Accuracy of the model: 0.95
Confusion Matrix:
 [[1392    6    0]
 [  31 1238  140]
 [   6   33 1362]]
Precision Score: 0.95
Recall Score: 0.95
Specificity of class 0: 0.99
Specificity of class 1: 0.99
Specificity of class 2: 0.95
F1 Score: 0.95
Fold 1 accuracy: 0.9838403041825095
Fold 2 accuracy: 0.9812262357414449
Fold 3 accuracy: 0.9843155893536122
Fold 4 accuracy: 0.9836026615969582
Fold 5 accuracy: 0.9828856667458997
The mean of AUC for this multi-label classification is 96.16%
```

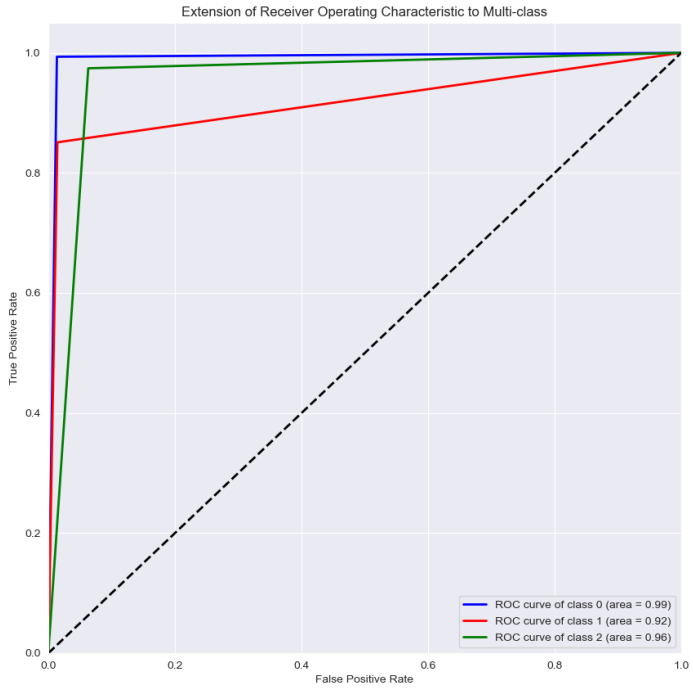

Fig. 75: ROC Curve: Baseline Support Vector Machine

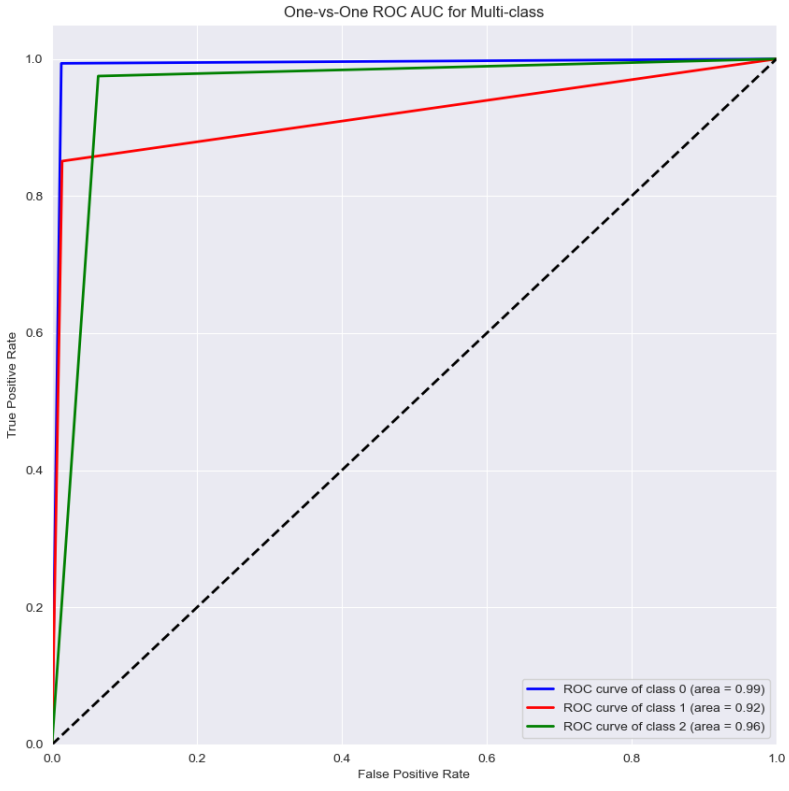

Fig. 76: ROC Curve: One-vs-One- Baseline Support Vector Machine

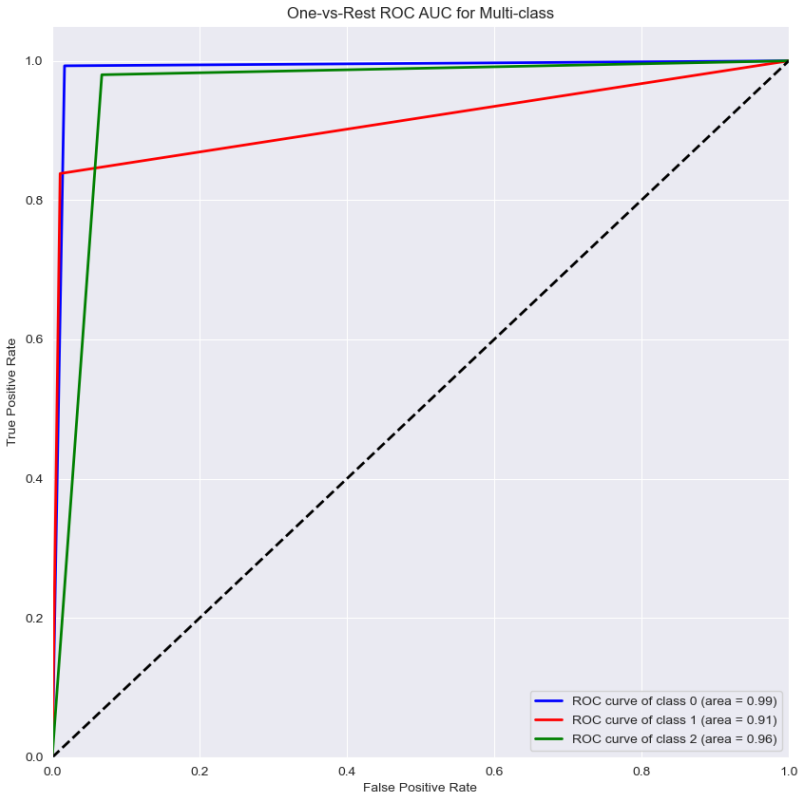

Fig. 77: ROC Curve: One-vs-Rest- Baseline Support Vector Machine

The Support Vector Machine is put through a grid search for finding its optimal parameters for improving its performance. Then after finding the best parameters, the Support Vector Machine is computed again. Here are the results:

```
Best Estimator:
 SVC(C=1, gamma=1, kernel='poly', random_state=5805)
Best Score:
 0.9651238357120709
Best Paramters:
 {'C': 1, 'gamma': 1, 'kernel': 'poly'}
Performance of the fine tuned svm on the test set:
Accuracy of the model: 0.97
Confusion Matrix:
 [[1396    2    0]
 [  10 1352   47]
 [   5   65 1331]]
Precision Score: 0.97
Recall Score: 0.97
Specificity of class 0: 0.99
Specificity of class 1: 0.98
Specificity of class 2: 0.98
F1 Score: 0.97
Fold 1 accuracy: 0.9759980988593155
Fold 2 accuracy: 0.9798003802281369
Fold 3 accuracy: 0.9788498098859315
Fold 4 accuracy: 0.9814638783269962
Fold 5 accuracy: 0.9793201806512954
The mean of AUC for this multi-label classification is 97.70%
```

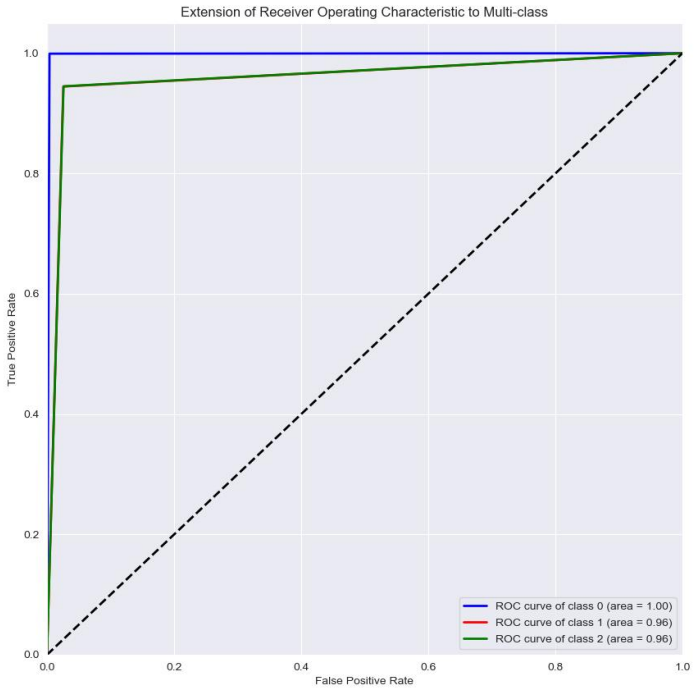

Fig. 78: ROC Curve: Fine Tuned Support Vector Machine

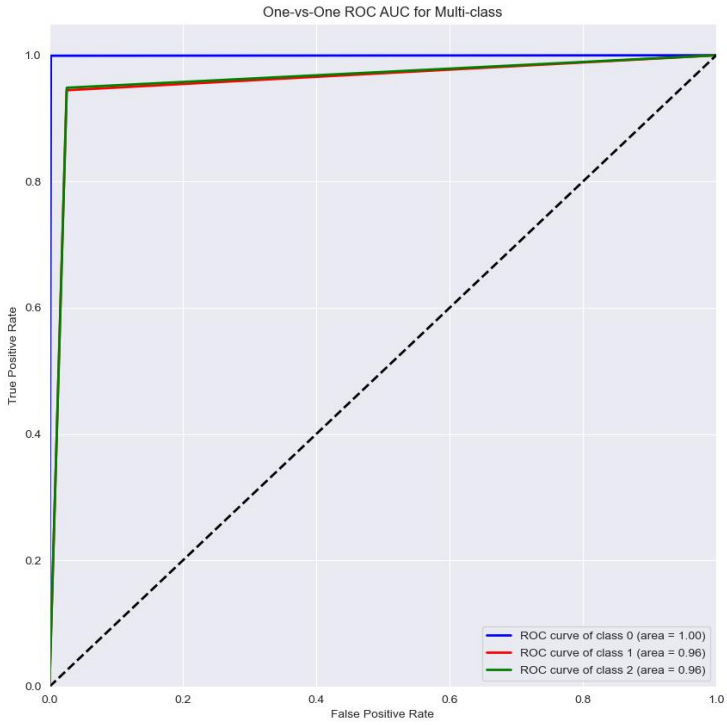

Fig. 79: ROC Curve: One-vs-One- Fine Tuned Support Vector Machine

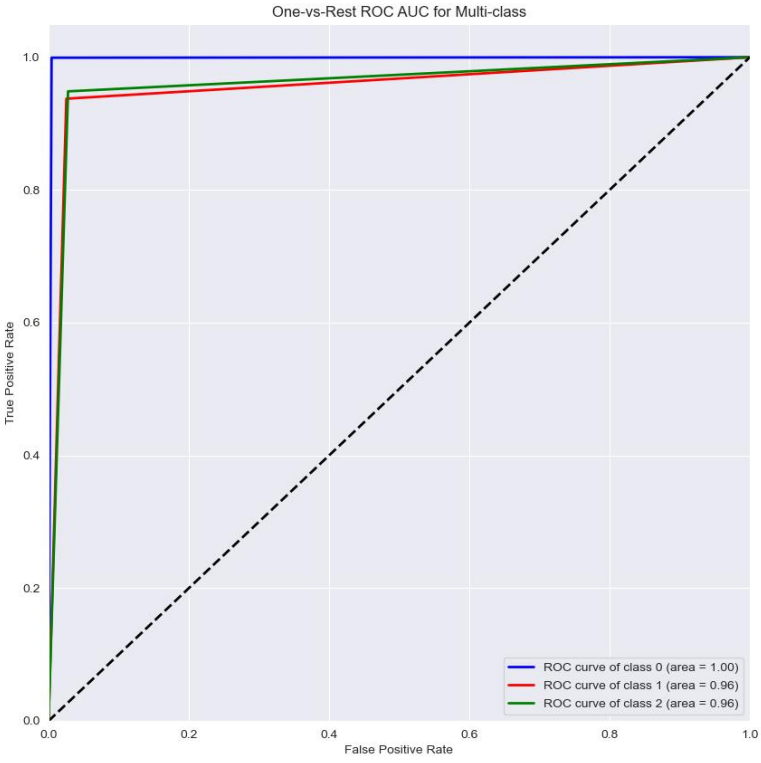

Fig. 80: ROC Curve: One-vs-Rest- Fine Tuned Support Vector Machine

Multilayer Perceptron:

A Multi-Layer Perceptron (MLP) is a type of neural network used for classification tasks, known for its ability to model complex patterns in data. It consists of multiple layers of nodes, including input, hidden, and output layers, with each node typically employing a non-linear activation function. MLPs can capture non-linear relationships in data through their deep architecture and are highly flexible, allowing them to be applied to a wide range of tasks.

Here are the results of the baseline Multi-Layer Perceptron:

```
Performance of the baseline multi layer perceptron on the test set:
Accuracy of the model: 0.96
Confusion Matrix:
 [[1395    3    0]
 [  10 1323   76]
 [   2   57 1342]]
Precision Score: 0.96
Recall Score: 0.96
Specificity of class 0: 1.00
Specificity of class 1: 0.98
Specificity of class 2: 0.97
F1 Score: 0.96
Fold 1 accuracy: 0.9817015209125475
Fold 2 accuracy: 0.9817015209125475
Fold 3 accuracy: 0.9814638783269962
Fold 4 accuracy: 0.9864543726235742
Fold 5 accuracy: 0.9828856667458997
The mean of AUC for this multi-label classification is 97.37%
```

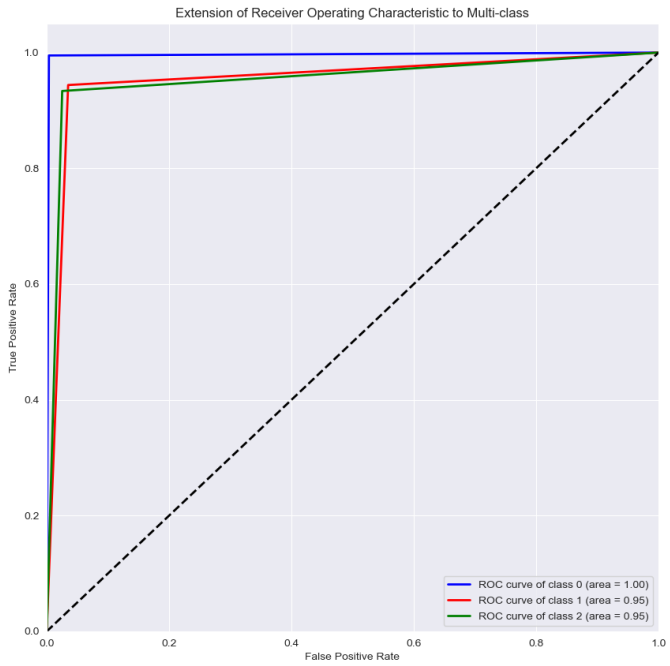

Fig. 81: ROC Curve: Baseline Multi-Layer Perceptron

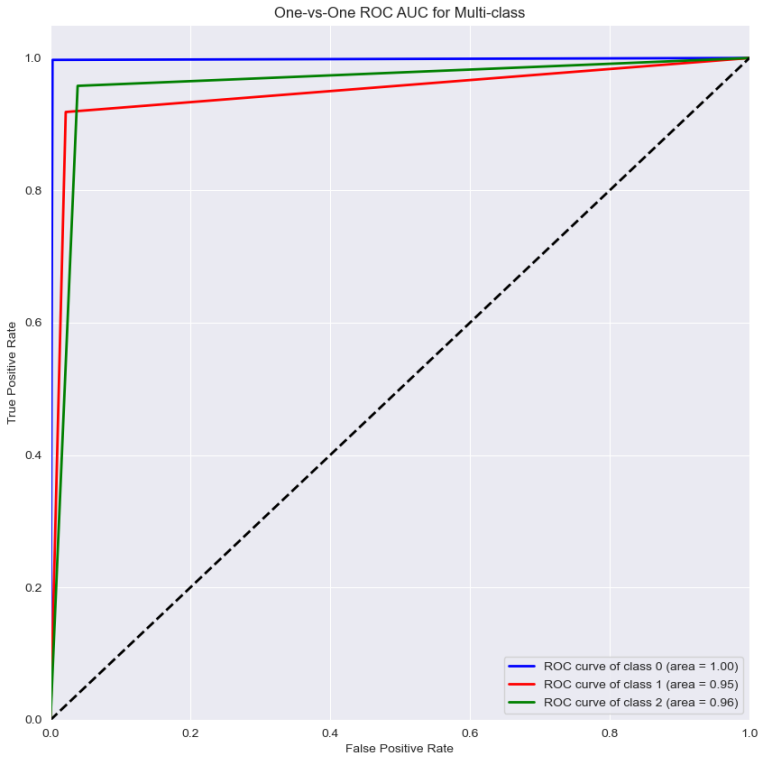

Fig. 82: ROC Curve: One-vs-One- Baseline Multi-Layer Perceptron

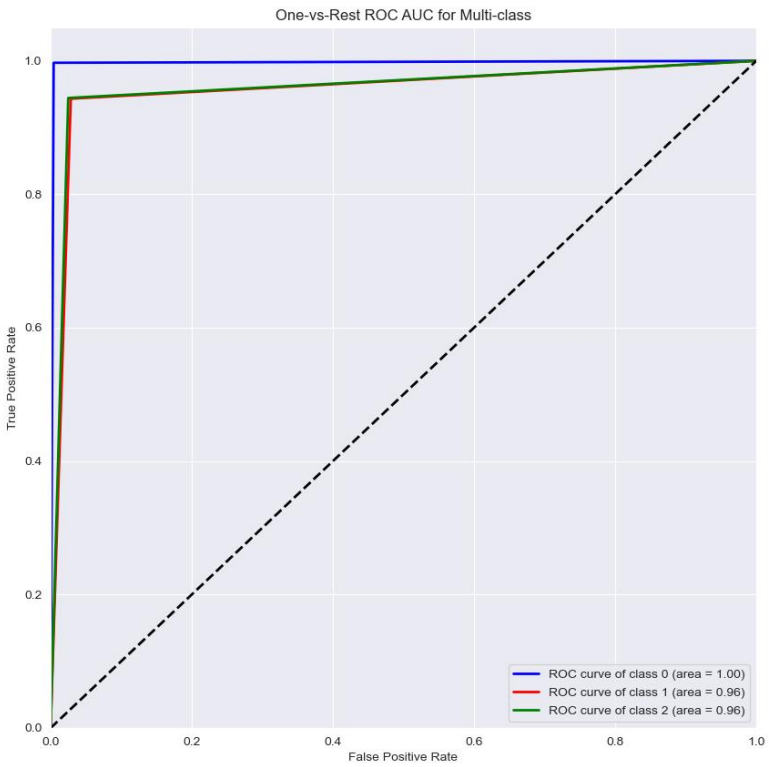

Fig. 83: ROC Curve: One-vs-Rest- Baseline Multi-Layer Perceptron

The Multi-Layer Perceptron is put through a grid search for finding its optimal parameters for improving its performance. Then after finding the best parameters, the Multi-Layer Perceptron is computed again. Here are the results:

```
Best Estimator:
 MLPClassifier(alpha=0.05, hidden_layer_sizes=(50, 100, 50), random_state=5805)
Best Score:
 0.9675599087363793
Best Paramters:
 {'alpha': 0.05, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'constant'}
Performance of the fine tuned multi layer perceptron on the test set:
Accuracy of the model: 0.97
Confusion Matrix:
 [[1395    3    0]
 [   6 1345   58]
 [   3   49 1349]]
Precision Score: 0.97
Recall Score: 0.97
Specificity of class 0: 1.00
Specificity of class 1: 0.98
Specificity of class 2: 0.98
F1 Score: 0.97
Fold 1 accuracy: 0.9786121673003803
Fold 2 accuracy: 0.9809885931558935
Fold 3 accuracy: 0.9788498098859315
Fold 4 accuracy: 0.9838403041825095
Fold 5 accuracy: 0.9812217732350844
The mean of AUC for this multi-label classification is 97.88%
```

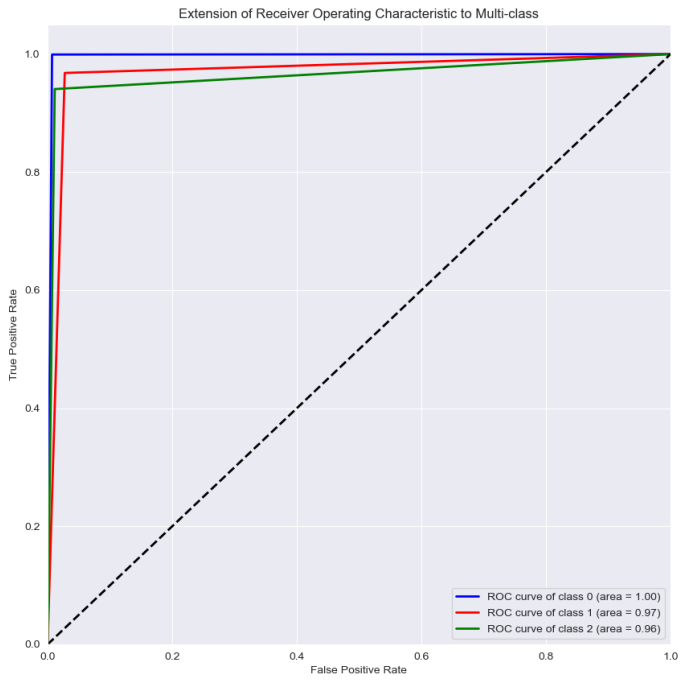

Fig. 84: ROC Curve: Fine Tuned Multi-Layer Perceptron

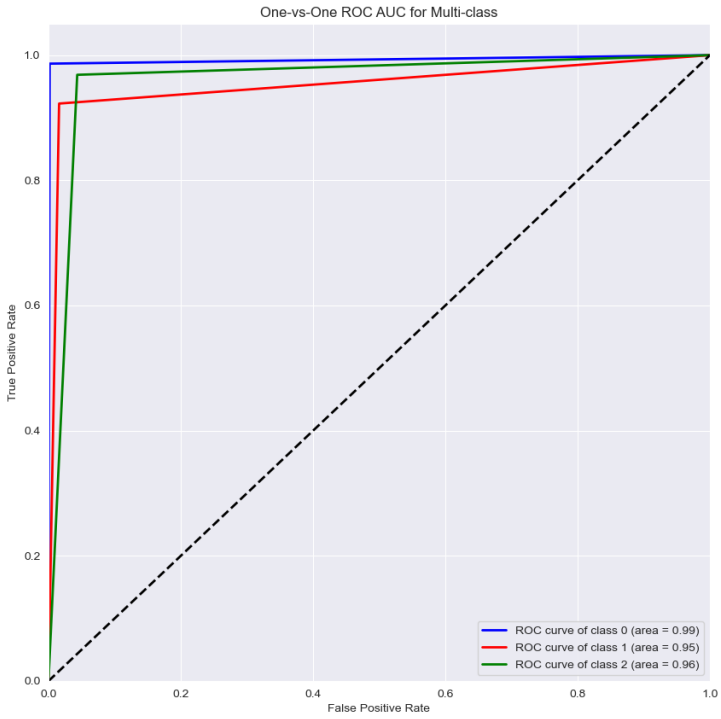

Fig. 85: ROC Curve: One-vs-One- Fine Tuned Multi-Layer Perceptron

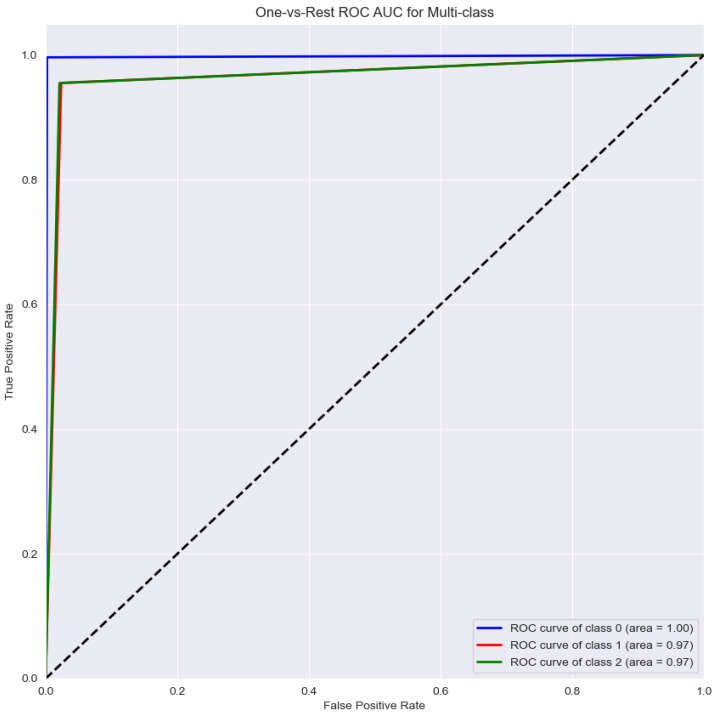

Fig. 86: ROC Curve: One-vs-Rest- Fine Tuned Multi-Layer Perceptron

Random Forest:

Random Forest is an ensemble learning which operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) of the individual trees. This method is known for its high accuracy, robustness, and ease of use, as it handles both categorical and continuous data well. Random Forest reduces overfitting in decision trees by averaging multiple trees, thereby improving predictive performance.

Here are the results of the baseline Random Forest:

```
Performance of the baseline random forest classifier on the test set:
Accuracy of the model: 0.99
Confusion Matrix:
 [[1397    1    0]
 [   0 1377   32]
 [   0   11 1390]]
Precision Score: 0.99
Recall Score: 0.99
Specificity of class 0: 1.00
Specificity of class 1: 1.00
Specificity of class 2: 0.99
F1 Score: 0.99
Fold 1 accuracy: 0.9928707224334601
Fold 2 accuracy: 0.9907319391634981
Fold 3 accuracy: 0.9912072243346007
Fold 4 accuracy: 0.9921577946768061
Fold 5 accuracy: 0.9909674352270026
The mean of AUC for this multi-label classification is 99.22%
```

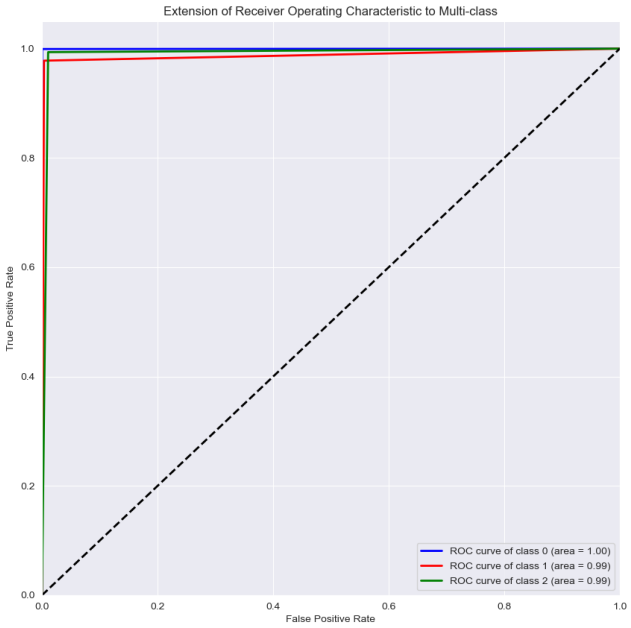

Fig. 87: ROC Curve: Baseline Random Forest

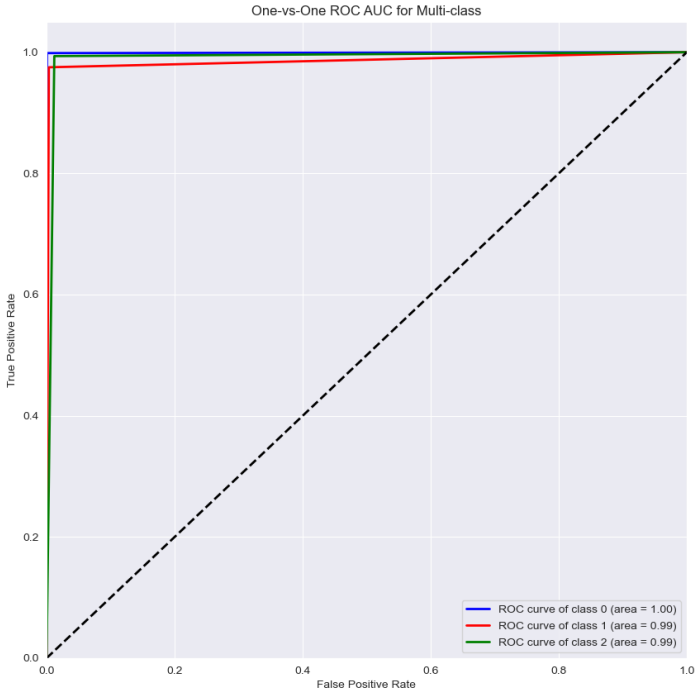

Fig. 88: ROC Curve: One-vs-One- Baseline Random Forest

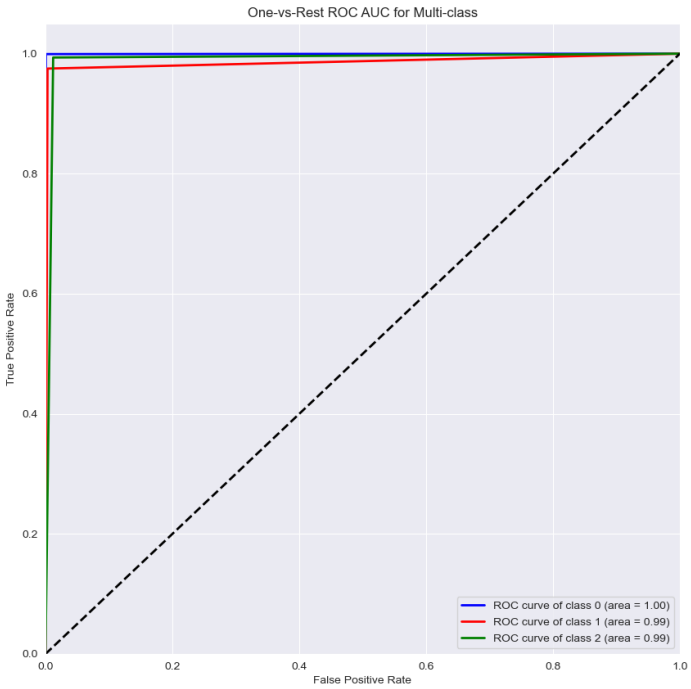

Fig. 89: ROC Curve: One-vs-Rest- Baseline Random Forest

Bagging is a fundamental technique used in Random Forest classifiers to improve model accuracy and robustness. It involves creating multiple subsets of the original dataset through random sampling with replacement, and then training separate decision trees on each subset. The final prediction is typically made by aggregating the predictions of each tree, often through majority voting for classification tasks.

Here are the results of the Bagging:

```
Performance of the random forest classifier with bagging on the test set:
Accuracy of the model: 0.99
Confusion Matrix:
 [[1397    1    0]
 [   0 1371   38]
 [   1   10 1390]]
Precision Score: 0.99
Recall Score: 0.99
Specificity of class 0: 1.00
Specificity of class 1: 1.00
Specificity of class 2: 0.99
F1 Score: 0.99
Fold 1 accuracy: 0.9912072243346007
Fold 2 accuracy: 0.9904942965779467
Fold 3 accuracy: 0.9907319391634981
Fold 4 accuracy: 0.9919201520912547
Fold 5 accuracy: 0.9909674352270026
The mean of AUC for this multi-label classification is 99.11%
```

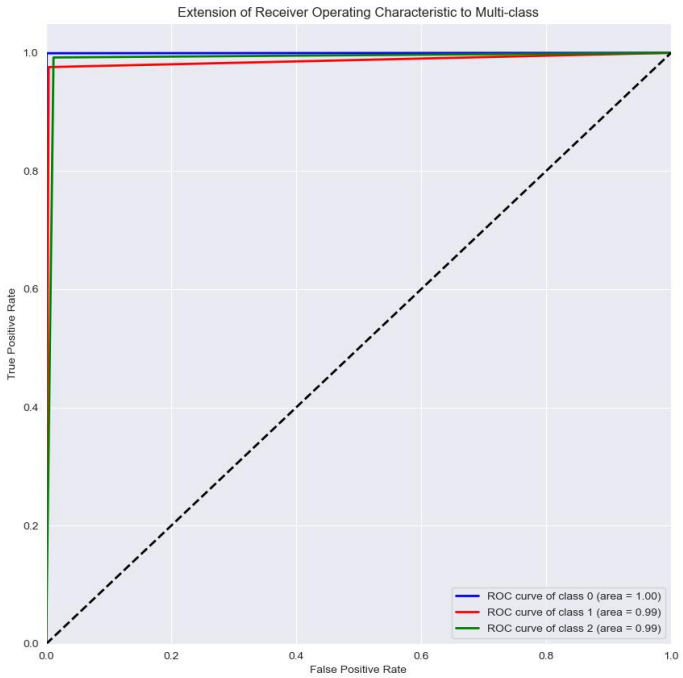

Fig. 90: ROC Curve: Bagging

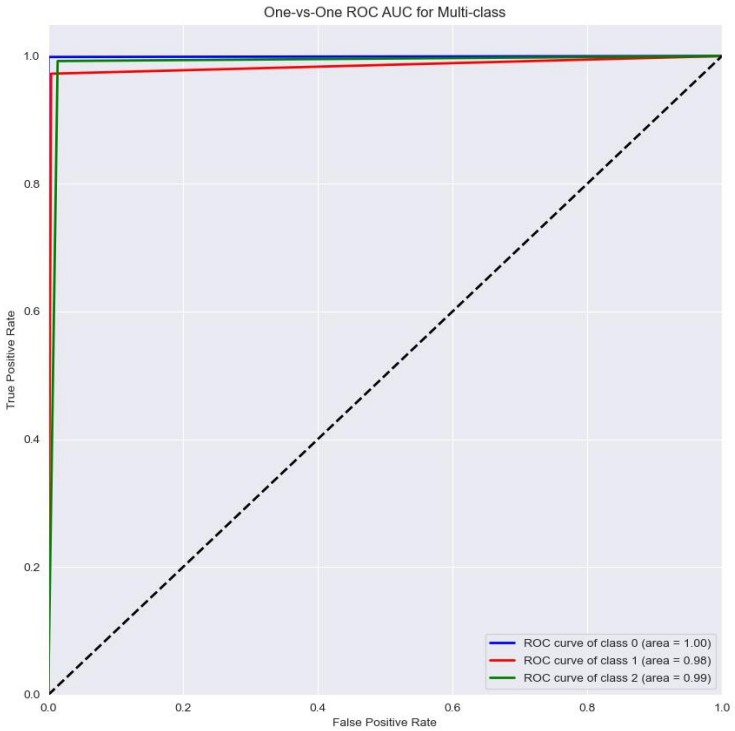

Fig. 91: ROC Curve: One-vs-One- Bagging

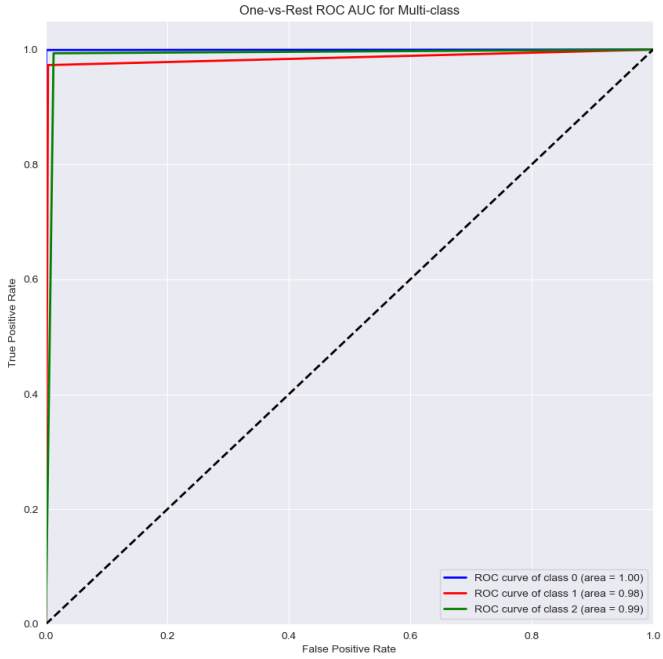

Fig. 92: ROC Curve: One-vs-Rest- Bagging

Stacking is an ensemble learning technique that combines multiple classification models, including Random Forests, to improve predictive performance. In stacking, various base classifiers, like Random Forests, are trained on the same dataset, and their predictions are used as inputs to a final classifier (or meta-classifier) that makes the ultimate prediction. This approach leverages the strengths of individual models, allowing for a more nuanced and accurate classification.

Here are the results of the Stacking:

```
Performance of the random forest classifier with stacking on the test set:
Accuracy of the model: 0.99
Confusion Matrix:
 [[1396    1    1]
 [   0 1375   34]
 [   0   23 1378]]
Precision Score: 0.99
Recall Score: 0.99
Specificity of class 0: 1.00
Specificity of class 1: 0.99
Specificity of class 2: 0.99
F1 Score: 0.99
Fold 1 accuracy: 0.9914448669201521
Fold 2 accuracy: 0.9904942965779467
Fold 3 accuracy: 0.9902566539923955
Fold 4 accuracy: 0.9912072243346007
Fold 5 accuracy: 0.990729736154029
The mean of AUC for this multi-label classification is 98.95%
```

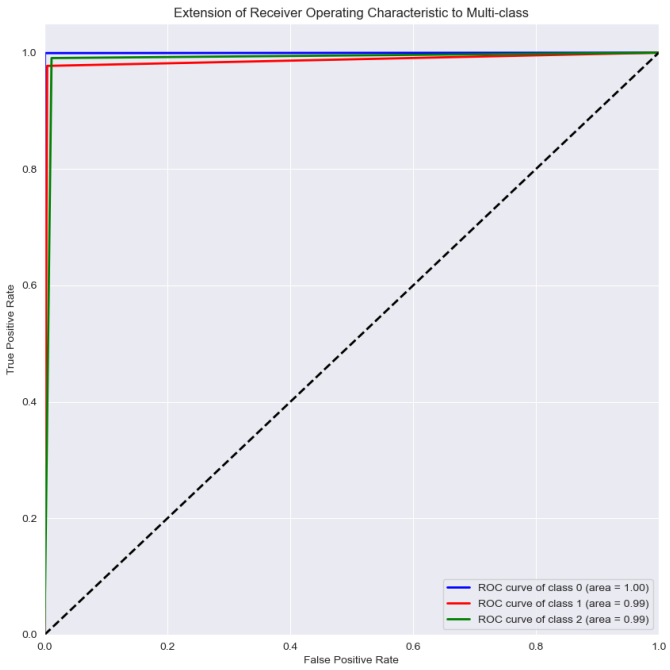

Fig. 93: ROC Curve: Stacking

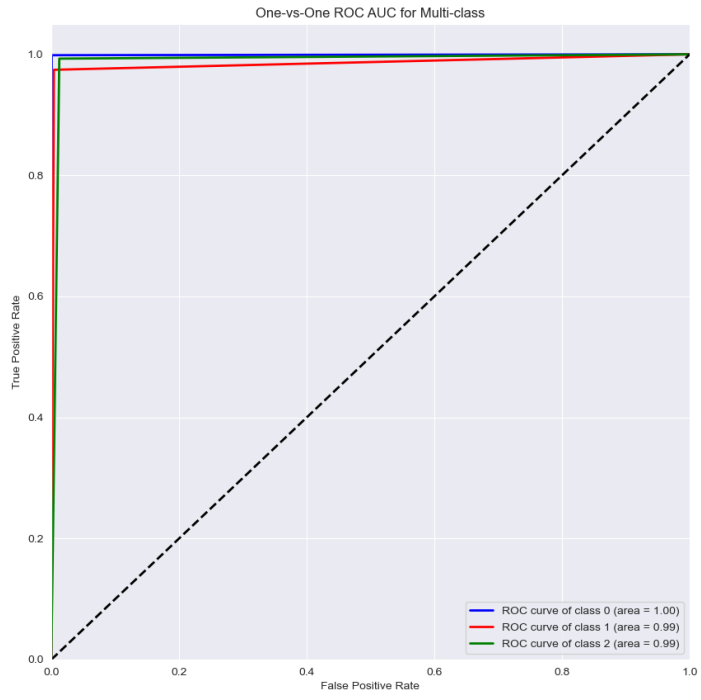

Fig. 94: ROC Curve: One-vs-One- Stacking

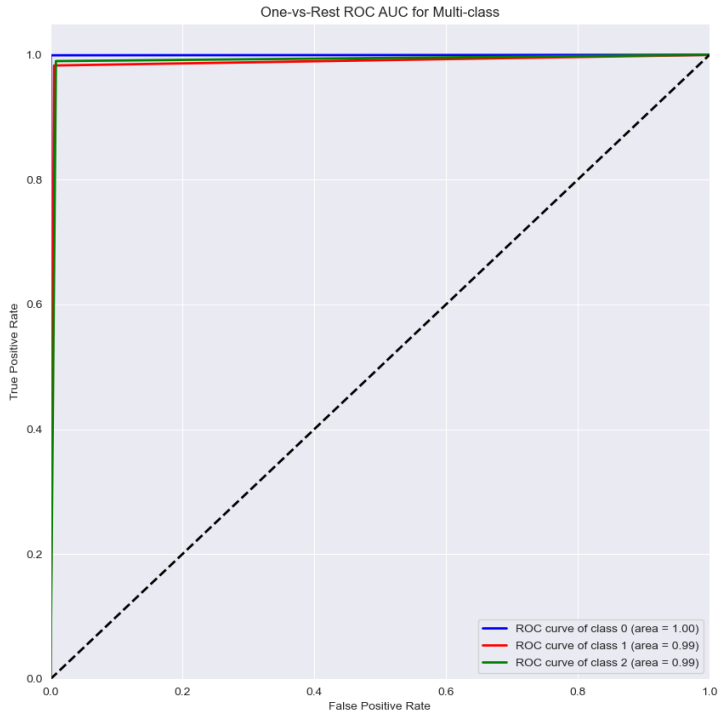

Fig. 95: ROC Curve: One-vs-Rest- Stacking

AdaBoost is a powerful ensemble technique often used in conjunction with Random Forest in classification tasks. This method works by sequentially fitting a series of weak learners, typically decision trees, with each tree focusing on the instances that were misclassified by the previous ones. In a Random Forest context, AdaBoost adjusts the weights of trees, giving more say to those that perform better, thereby creating a strong classifier from a collection of individually weak ones. This approach enhances the overall performance, especially in reducing bias and variance.

Here are the results of the Boosting:

```
Performance of the random forest classifier with boosting on the test set:
Accuracy of the model: 0.99
Confusion Matrix:
 [[1396    2    0]
 [   0 1380   29]
 [   0    8 1393]]
Precision Score: 0.99
Recall Score: 0.99
Specificity of class 0: 1.00
Specificity of class 1: 1.00
Specificity of class 2: 0.99
F1 Score: 0.99
Fold 1 accuracy: 0.9916825095057035
Fold 2 accuracy: 0.9912072243346007
Fold 3 accuracy: 0.9914448669201521
Fold 4 accuracy: 0.9931083650190115
Fold 5 accuracy: 0.9916805324459235
The mean of AUC for this multi-label classification is 99.31%
```

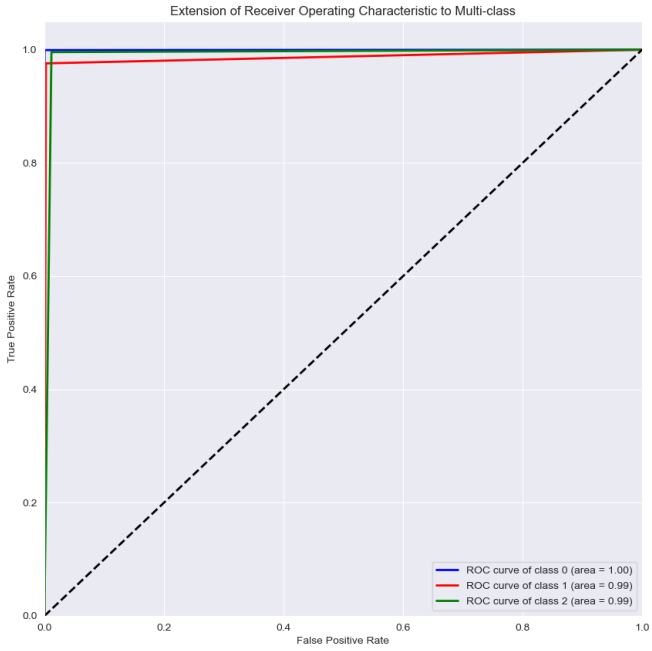

Fig. 96: ROC Curve: Boosting

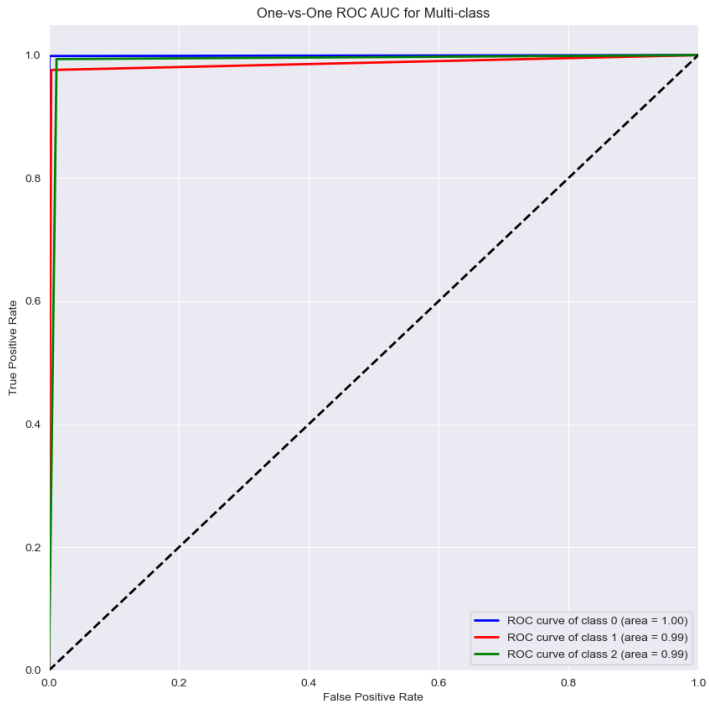

Fig. 97: ROC Curve: One-vs-One- Boosting

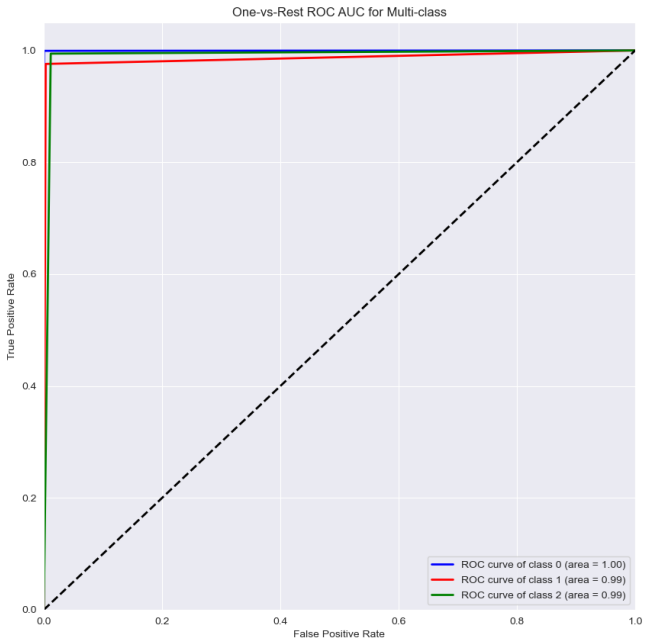

Fig. 98: ROC Curve: One-vs-Rest- Boosting

The Random Forest Classifier is put through a grid search for finding its optimal parameters for improving its performance. Then after finding the best parameters, Random Forest Classifier is computed again. Here are the results:

```
Best Estimator:
 RandomForestClassifier(n_jobs=-1, random_state=5805)
Performance of the random forest classifier with grid search on the test set:
Accuracy of the model: 0.99
Confusion Matrix:
 [[1397    1    0]
 [   0 1377   32]
 [   0   11 1390]]
Precision Score: 0.99
Recall Score: 0.99
Specificity of class 0: 1.00
Specificity of class 1: 1.00
Specificity of class 2: 0.99
F1 Score: 0.99
Fold 1 accuracy: 0.9928707224334601
Fold 2 accuracy: 0.9907319391634981
Fold 3 accuracy: 0.9912072243346007
Fold 4 accuracy: 0.9921577946768061
Fold 5 accuracy: 0.9909674352270026
The mean of AUC for this multi-label classification is 99.22%
```

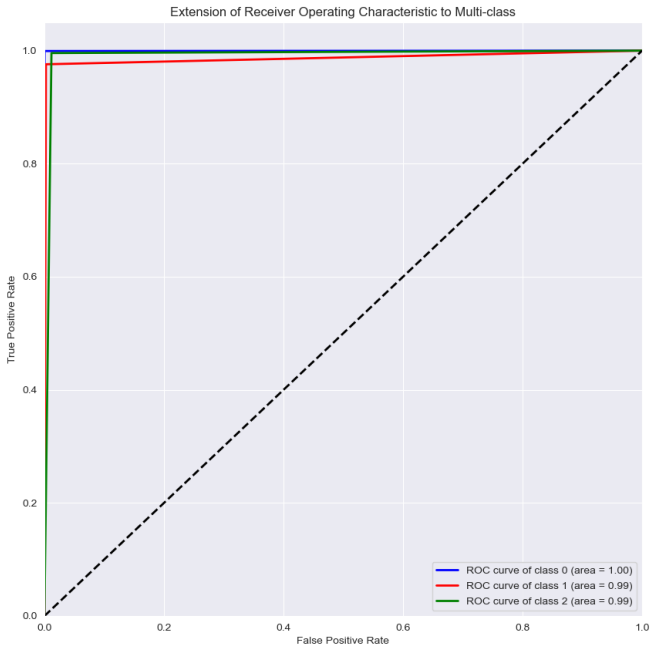

Fig. 99: ROC Curve: Fined-Tuned Random Forest

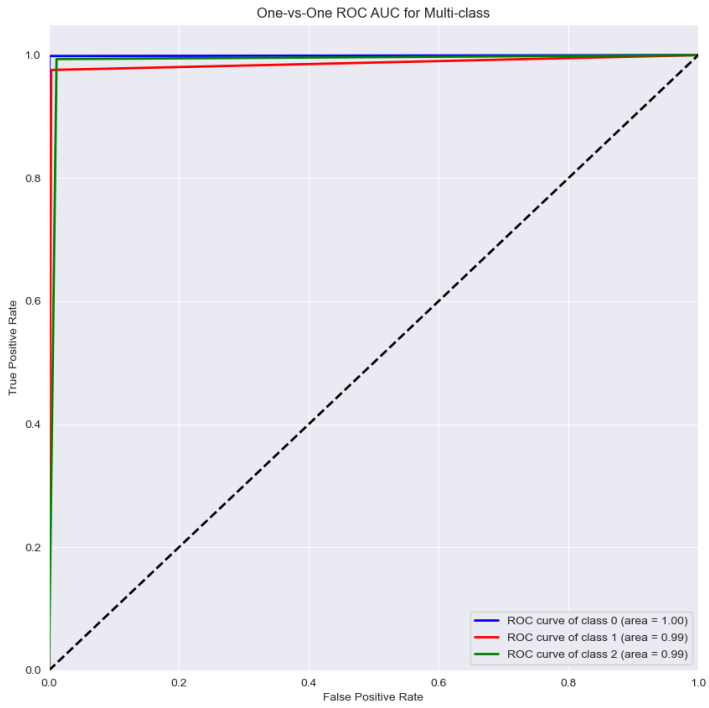

Fig. 100: ROC Curve: One-vs-One- Fine-Tuned Random Forest

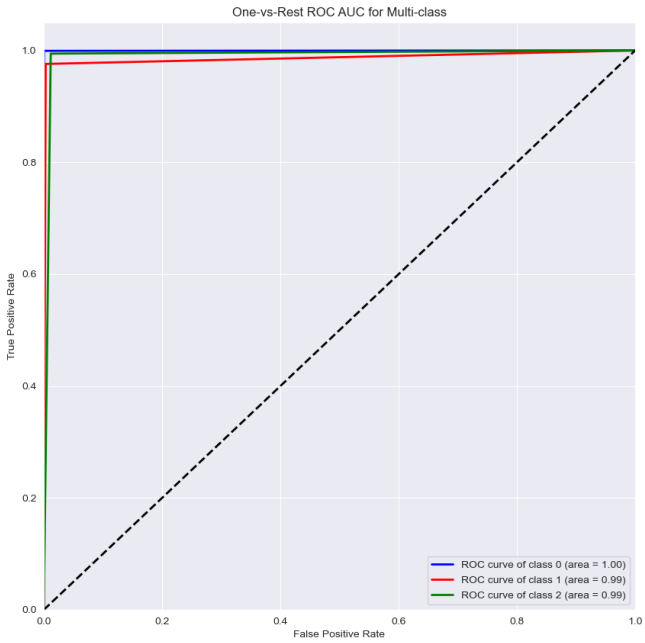

Fig. 101: ROC Curve: One-vs-Rest- Fine-Tuned Random Forest

The respective confusion matrices of the individual models are:

Baseline Decision Tree

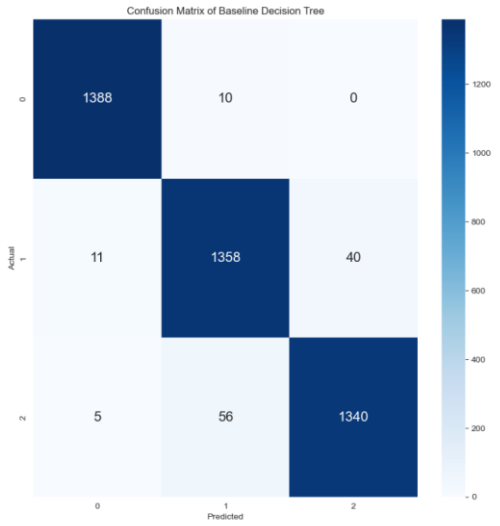

Fig. 102: Graphical Confusion Matrix: Baseline Decision Tree

Fine-Tuned Decision Tree

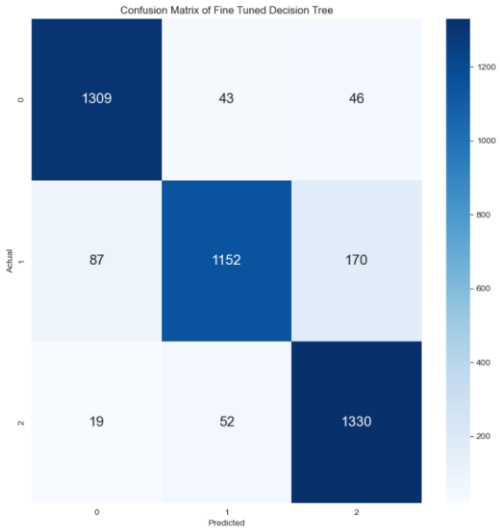

Fig. 103: Graphical Confusion Matrix: Fine-Tuned Decision Tree

Optimal Alpha: Post Pruned Decision Tree

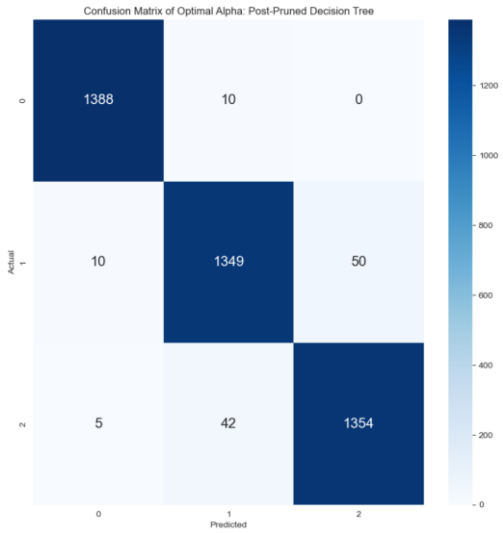

Fig. 104: Graphical Confusion Matrix: Post-Pruned Decision Tree

Baseline Logistic Regression

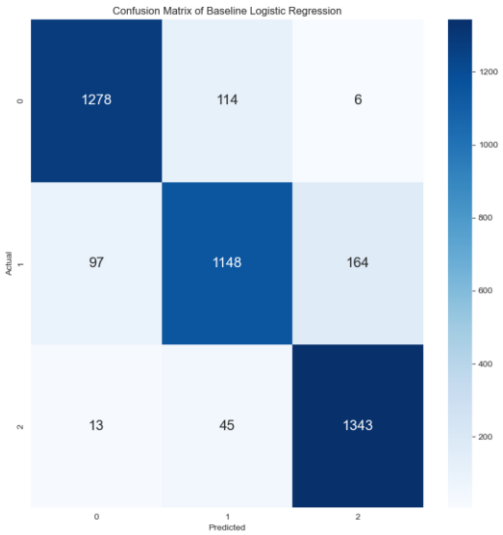

Fig. 105: Graphical Confusion Matrix: Baseline Logistic Regression

Fine-Tuned Logistic Regression

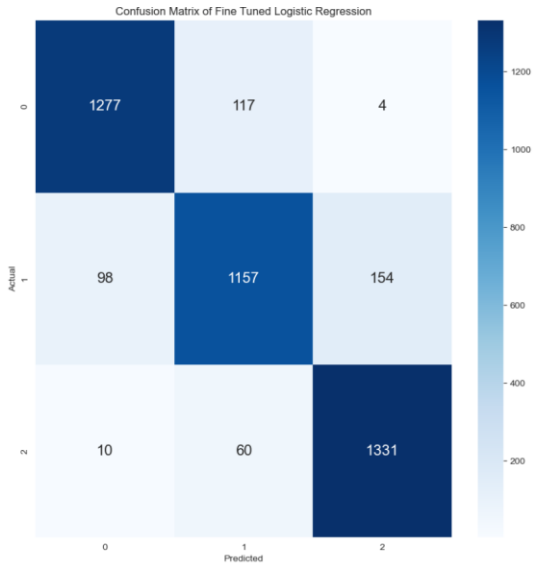

Fig. 106: Graphical Confusion Matrix: Fine-Tuned Logistic Regression

Baseline KNN

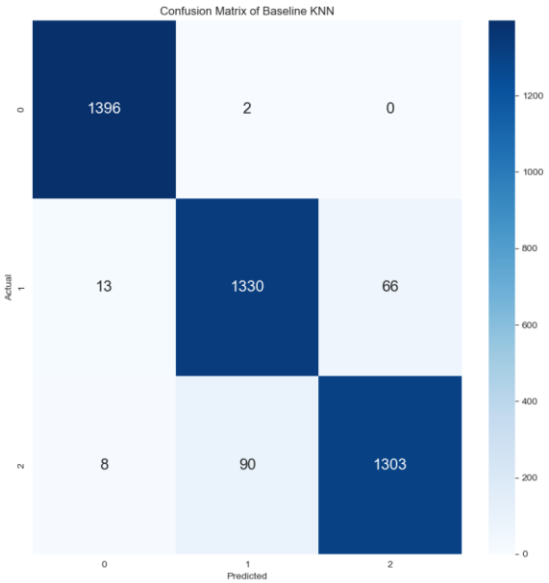

Fig. 107: Graphical Confusion Matrix: Baseline KNN

Fine-Tuned KNN

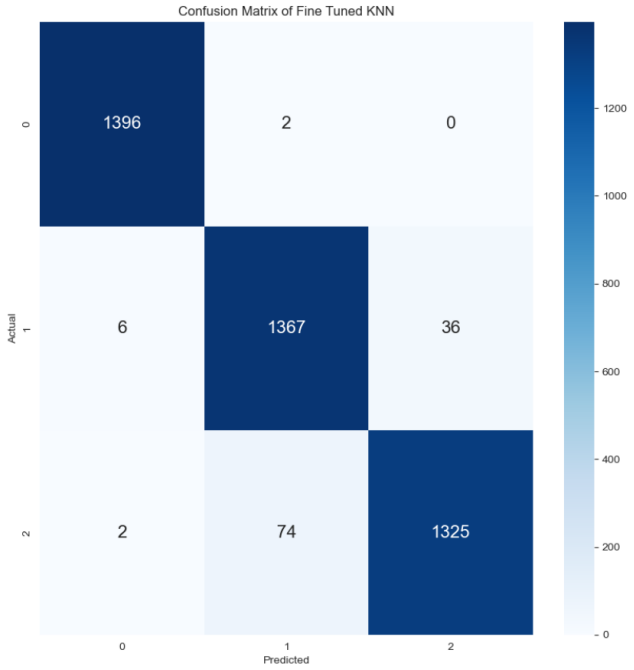

Fig. 108: Graphical Confusion Matrix: Fine-Tuned KNN

Baseline Gaussian Naïve Bayes

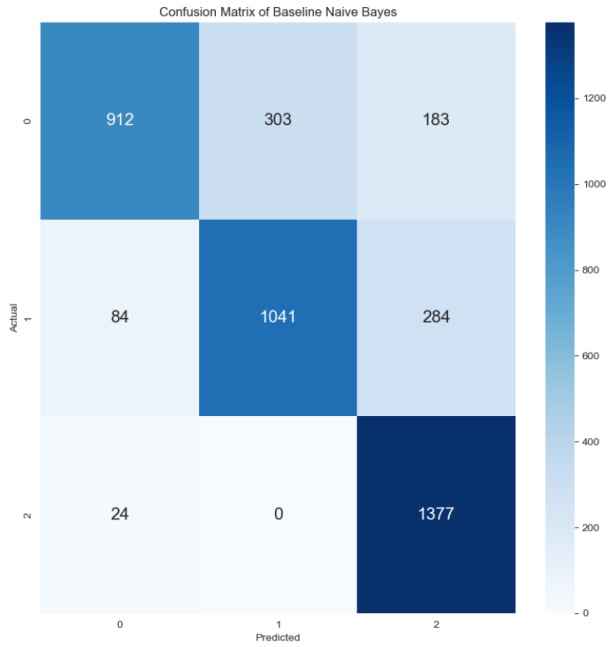

Fig. 109: Graphical Confusion Matrix: Baseline Gaussian Naïve Bayes

Fine-Tuned Gaussian Naïve Bayes

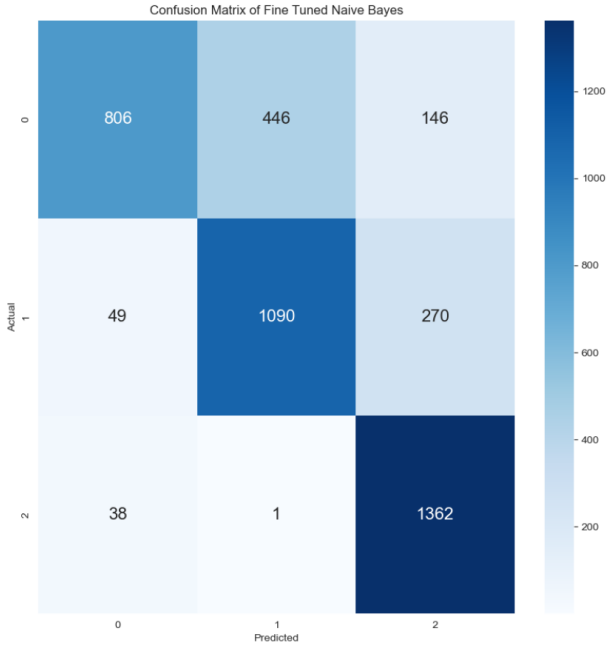

Fig. 110: Graphical Confusion Matrix: Fine-Tuned Gaussian Naïve Bayes

Baseline SVM

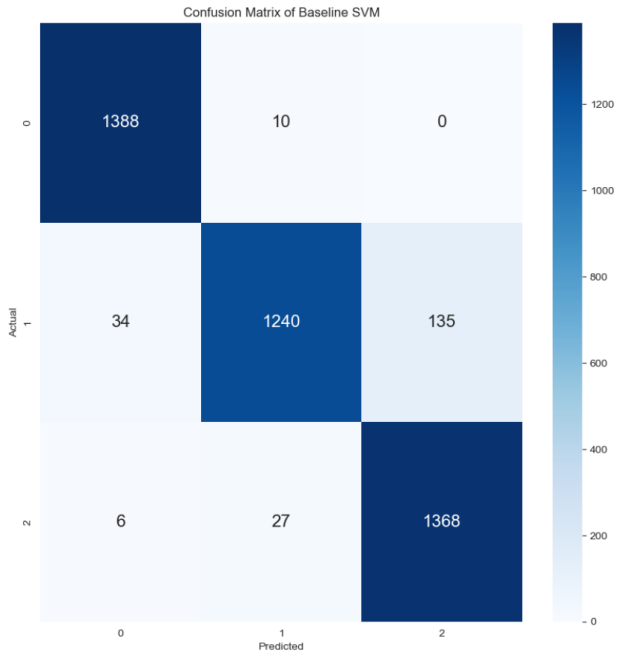

Fig. 111: Graphical Confusion Matrix: Baseline SVM

Fine-Tuned SVM

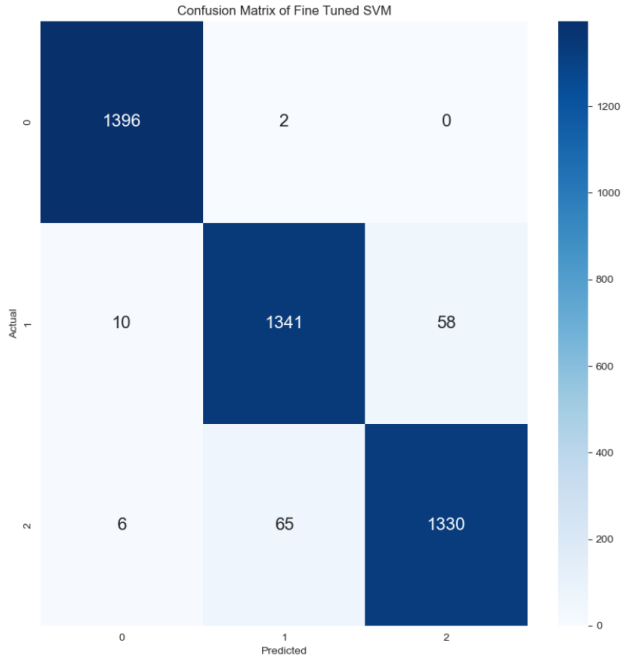

Fig. 112: Graphical Confusion Matrix: Fine-Tuned SVM

Baseline MLP

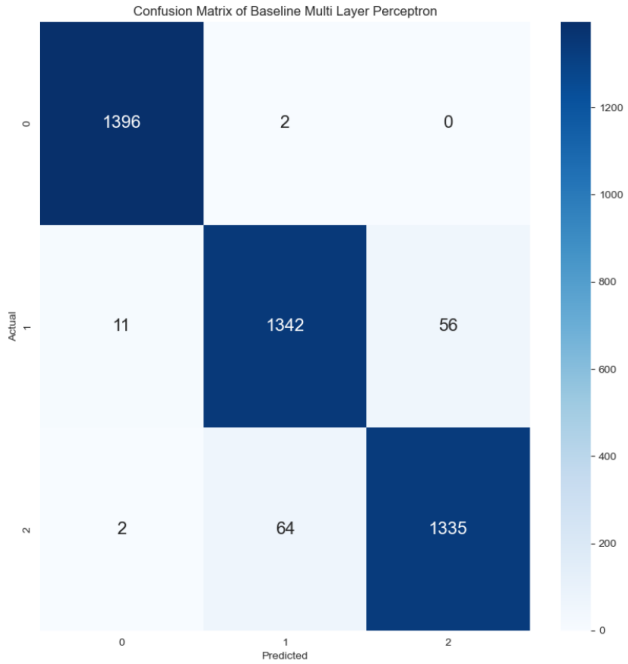

Fig. 113: Graphical Confusion Matrix: Baseline MLP

Fine-Tuned MLP

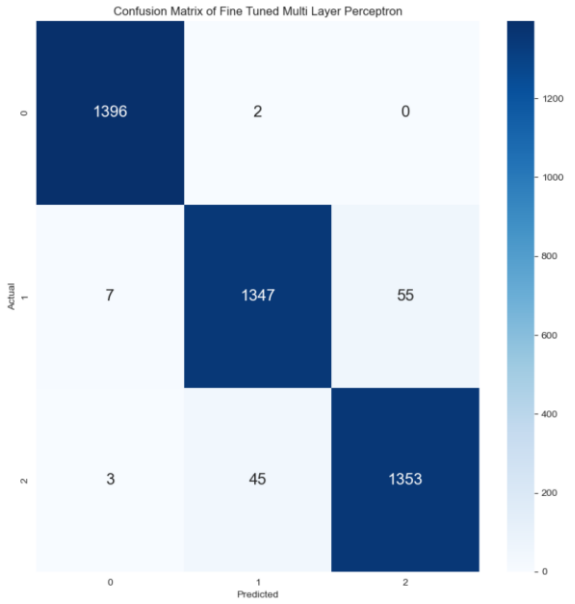

Fig. 114: Graphical Confusion Matrix: Fine-Tuned MLP

Baseline Random Forest

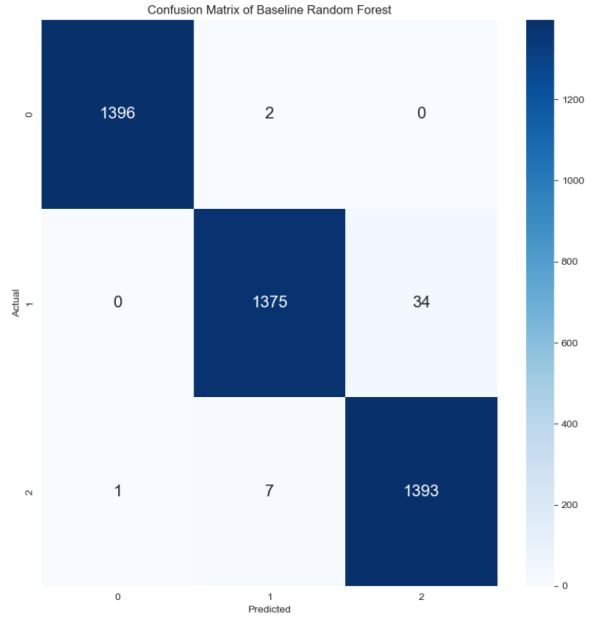

Fig. 115: Graphical Confusion Matrix: Baseline Random Forest

Random Forest with Bagging

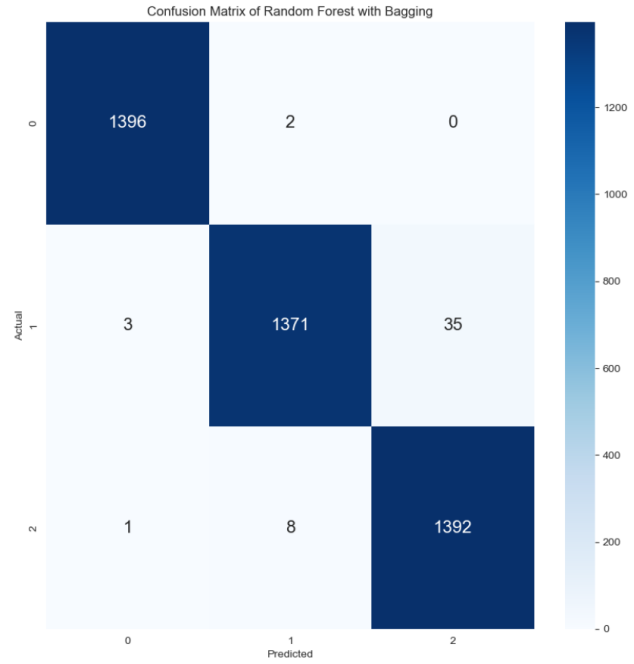

Fig. 116: Graphical Confusion Matrix: Random Forest with Bagging

Random Forest with Stacking

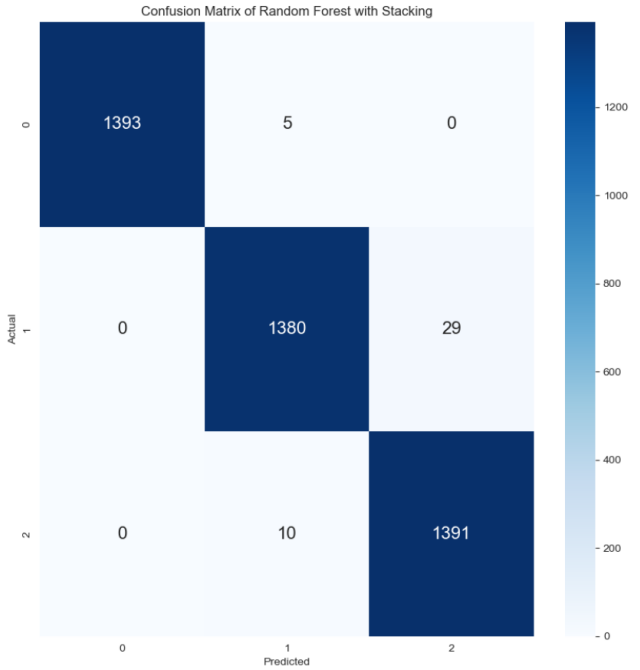

Fig. 117: Graphical Confusion Matrix: Random Forest with Stacking

Random Forest with Boosting

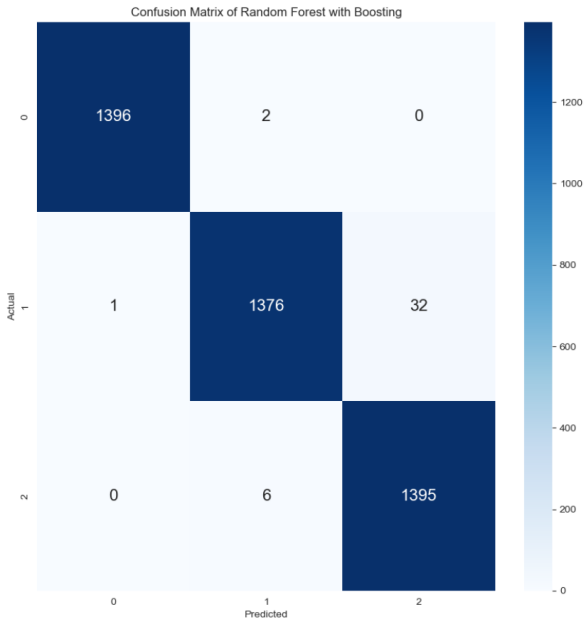

Fig. 118: Graphical Confusion Matrix: Random Forest with Boosting

Fine-Tuned Random Forest

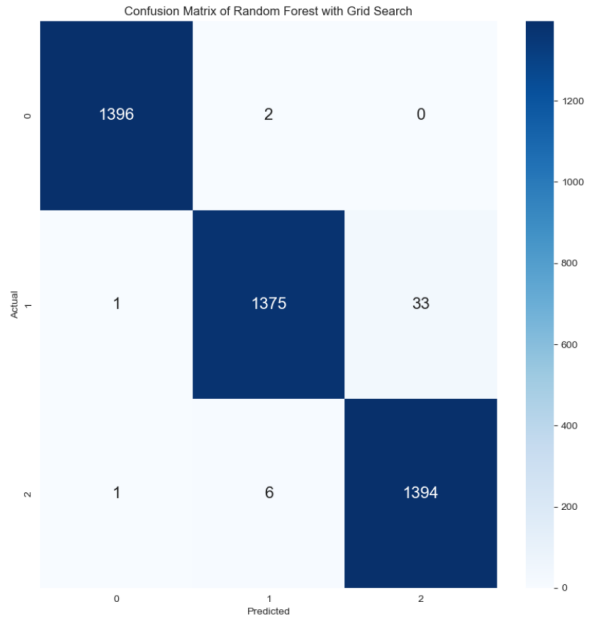

Fig. 119: Graphical Confusion Matrix: Fine-Tuned Random Forest

Best Classification Model:

Model Comparison						
Model	Accuracy	Precision	Recall	F1 Score	Cross Validation Mean Score	AUC Score
Random Forest with Grid Search	0.99	0.99	0.99	0.99	0.99	0.99
Random Forest with Boosting	0.99	0.99	0.99	0.99	0.99	0.99
Random Forest with Stacking	0.99	0.99	0.99	0.99	0.99	0.99
Random Forest with Bagging	0.99	0.99	0.99	0.99	0.99	0.99
Baseline Random Forest	0.99	0.99	0.99	0.99	0.99	0.99
Fine Tuned SVM	0.97	0.97	0.97	0.97	0.98	0.98
Fine Tuned Multi Layer Perceptron	0.97	0.97	0.97	0.97	0.98	0.98
Baseline Decision Tree	0.97	0.97	0.97	0.97	0.97	0.98
Fine Tuned KNN	0.97	0.97	0.97	0.97	0.96	0.98
Optimal Alpha: Post-Pruned Decision Tree	0.97	0.97	0.97	0.97	0.97	0.98
Baseline Multi Layer Perceptron	0.96	0.96	0.96	0.96	0.98	0.97
Baseline KNN	0.96	0.96	0.96	0.96	0.95	0.97
Baseline SVM	0.95	0.95	0.95	0.95	0.98	0.96
Fine Tuned Decision Tree	0.93	0.93	0.93	0.93	0.71	0.95
Fine Tuned Logistic Regression	0.9	0.9	0.9	0.9	0.97	0.93
Baseline Logistic Regression	0.9	0.9	0.9	0.9	0.97	0.92
Baseline Naive Bayes	0.79	0.8	0.79	0.78	0.6	0.84
Fine Tuned Naive Bayes	0.76	0.78	0.76	0.76	0.67	0.82

Table 2: Classification Model Comparison

To determine the "best" model, we typically look for the one with the highest scores across these metrics. It seems that random forest and all of its associated models are overfitting. It is also important to consider the balance between Precision and Recall, and to look at the F1 Score. Cross-validation scores can help assess this, as they indicate how well a model performs across different subsets of the data. According to me the Post-Pruned Decision Tree is the best classifier model, its Cross Validation Mean Score is 0.97, and its AUC Score is 0.98, which suggests that it generalizes well and maintains a high level of predictive performance across different data subsets, as well as the perfect balance between the performance metrics.

Phase 4: Clustering and Association

Clustering:

Clustering is a fundamental technique in unsupervised machine learning where the objective is to segment a dataset into distinct groups such that data points within the same group are more similar to each other than to those in other groups. This similarity is often based on features within the dataset and is measured using a distance metric such as Euclidean, Manhattan, or cosine similarity. Clustering algorithms seek to maximize inter-cluster differences and minimize intra-cluster differences.

There are various algorithms for clustering, each with its own strengths and suited for different kinds of data distributions and applications. K-means clustering is one of the simplest and most popular algorithms, where 'k' refers to the number of clusters to be identified, and the algorithm iteratively assigns data points to the nearest cluster centroid. More advanced techniques like DBSCAN (Density-Based Spatial Clustering of Applications with Noise) can find arbitrarily shaped clusters and can identify outliers as noise. The choice of algorithm and parameters such as the number of clusters often requires domain knowledge and may be guided by iterative experimentation and evaluation measures like silhouette scores.

Here in the project, we have used two kinds of clustering algorithms, mainly K-means clustering and DBSCAN clustering:

K-means Clustering:

K-means clustering is a widely used method of vector quantization that partitions a dataset into K distinct, non-overlapping clusters. It starts with a predetermined number of clusters and randomly assigned centroids, then iteratively assigns each data point to the nearest centroid based on some distance metric, usually Euclidean distance. After all data points are assigned, the positions of the centroids are recalculated as the mean of the points in the cluster, and this process repeats until the centroids no longer move significantly, signifying convergence. This algorithm is favored for its simplicity and efficiency in processing large datasets. However, K-means relies on the user to specify the number of clusters, and it assumes that clusters are spherical and evenly sized, which can be a limitation for complex data structures.

Here are the results of K-means clustering:

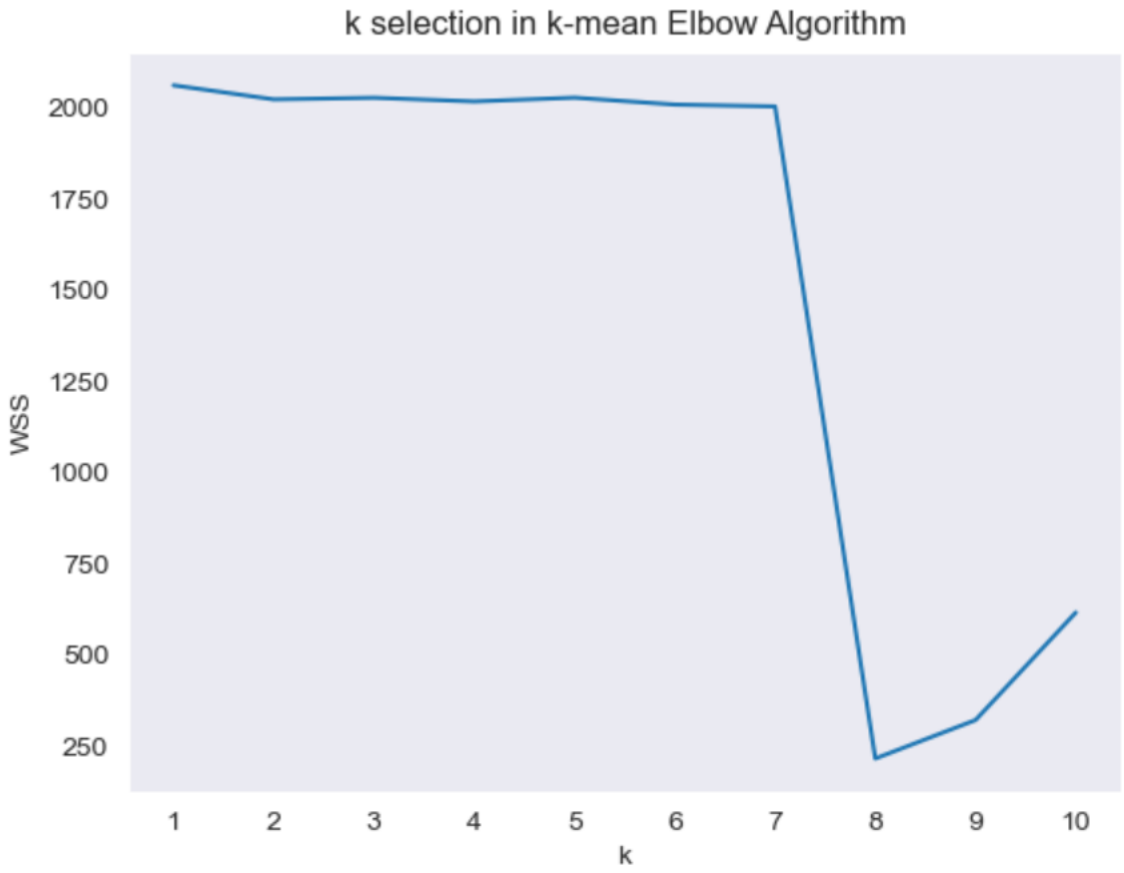

Fig. 120: k selection in K-Means clustering through the Elbow Method

Observations: The graph depicts the Elbow Method being used to select the optimal number of clusters for k-means clustering. A very small elbow appears at k=2, indicating that increasing the number of clusters beyond this point does not result in a substantial decrease in WCSS. This suggests that k=2 might be a good choice for the number of clusters to use in this scenario.

We can also check the optimal no. of clusters through graphing the silhouette scores for each cluster.

Here are the results of graphing for each of the silhouette score for each cluster:

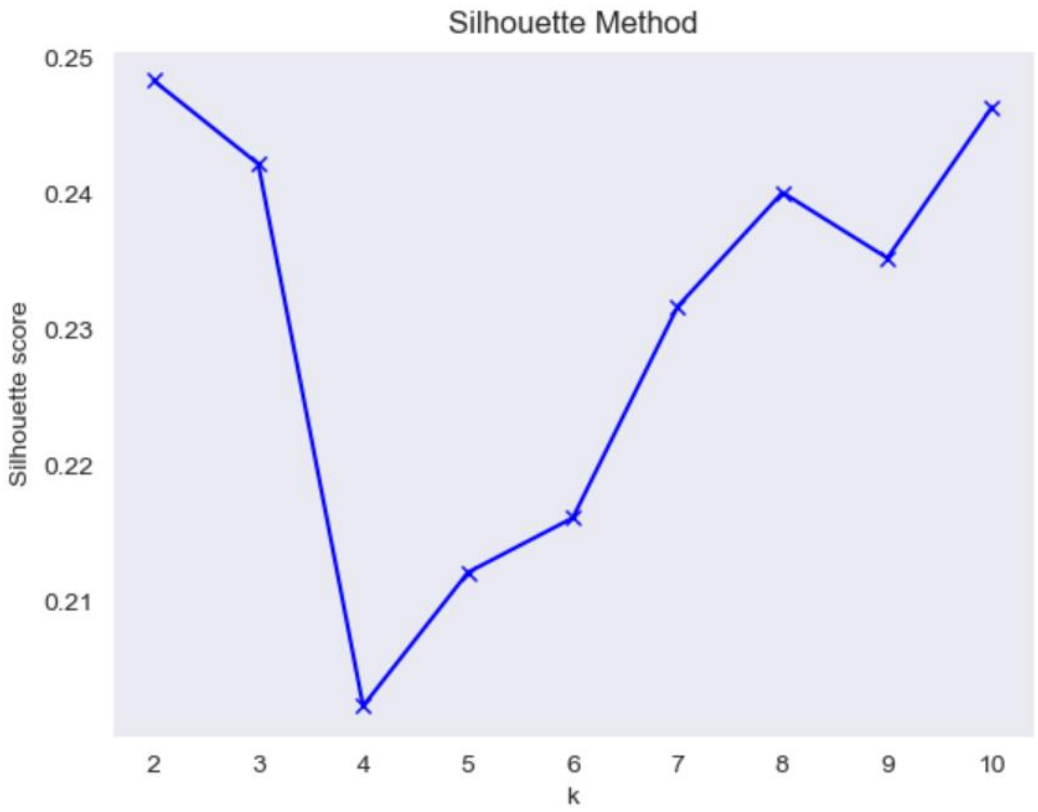

Fig. 121: Silhouette Scores of each cluster

Observations: The graph shows silhouette scores for different numbers of clusters used in k-means clustering, with the silhouette score measuring how similar an object is to its own cluster compared to other clusters, where there can be seen the highest is at k=2, while the lowest is at k=5. This means that the data can be divided into 2 clusters.

DBSCAN Clustering:

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that identifies clusters as high-density areas separated by areas of low density. Distinct from centroid-based algorithms like k-means, DBSCAN does not require a pre-specified number of clusters. It operates on the idea that a cluster forms a region of high density that is surrounded by regions of low density. The core concept in DBSCAN is the classification of points into core points, border points, and noise points. A core point has at least a minimum number of other points within a given radius (epsilon). Border points are reachable from core points but have fewer minimum number of other points within their epsilon neighborhood, and noise points are neither core nor border points.

DBSCAN's ability to find arbitrarily shaped clusters and its robustness to outliers make it suitable for applications with spatial data, such as geographic information systems and recognition of regions of similar density in a dataset. It is particularly effective for datasets where clusters are not spherical and vary in density. The algorithm starts by randomly selecting a point and retrieving all points density-reachable from it. If the point is a core point, a cluster is formed. This process continues until all points are classified into clusters or marked as noise.

Here is the plot of DBSCAN clustering:

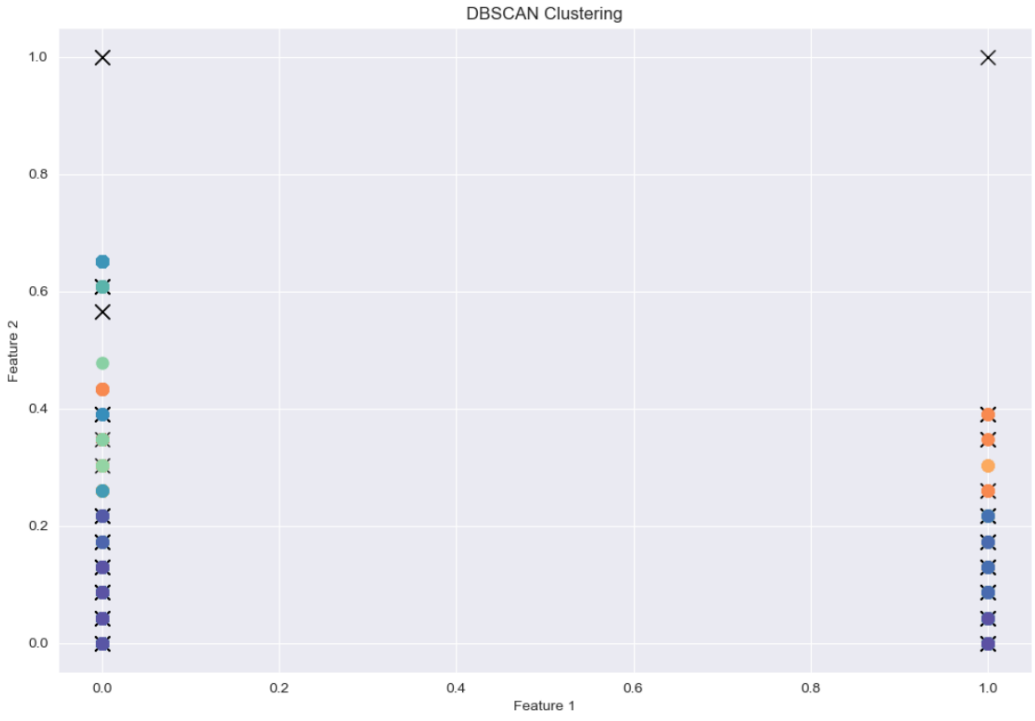

Fig. 122: DBSCAN clustering

Observations: This graph shows different clusters, which also illustrates the algorithm's capability to detect outliers as noise and form clusters based on density. Different colors or symbols might indicate distinct clusters or noise identified by DBSCAN. The distribution suggests that the algorithm has identified several core clusters, with a few points being classified as noise ('X').

Association Rule Mining:

Association rule mining is a data mining process used to discover interesting relations between variables in large databases. It is a rule-based machine learning method for discovering interesting relations between variables in large databases. These rules are derived using measures of interest, such as support, confidence, and lift. Support indicates how frequently the items appear in the dataset, while confidence tells us how often the rule has been found to be true. Lift compares the observed frequency of a rule against the frequency expected if the two sets were independent.

The process of association rule mining consists of two key steps: first, finding all frequent item sets in the dataset that meet the minimum support threshold; second, generating strong association rules from the frequent item sets that meet the minimum confidence threshold. A frequent itemset is one that occurs together more often than a user-specified threshold. From these item sets, association rules are inferred which predict the occurrence of an item based on the occurrences of other items in the transaction.

For the project, Apriori algorithm is used to find the association rules.

Here in this project, the basket is:

Each row represents a level of 'Accident_Severity'. For each level of 'Accident_Severity', it calculates the total 'Number_of_Casualties', the total 'Speed_limit', and the total 'Number_of_Vehicles'.

Here are the sample association rules created by Apriori algorithm in the project:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(Number_of_Casualties)	(Speed_limit)	1.00	1.00	1.00	1.00	1.00	0.00	inf	0.00
1	(Speed_limit)	(Number_of_Casualties)	1.00	1.00	1.00	1.00	1.00	0.00	inf	0.00
2	(Number_of_Casualties)	(Number_of_Vehicles)	1.00	1.00	1.00	1.00	1.00	0.00	inf	0.00
3	(Number_of_Vehicles)	(Number_of_Casualties)	1.00	1.00	1.00	1.00	1.00	0.00	inf	0.00
4	(Number_of_Vehicles)	(Speed_limit)	1.00	1.00	1.00	1.00	1.00	0.00	inf	0.00

Observation: This table shows all the rules have a support and confidence of 1.00, suggesting that the antecedent occurs together with the consequent in all cases within the dataset. The lift is 1.00, and leverage is 0.00 for almost all rules, indicating that the antecedent and consequent are independent of each other. The 'inf' values for conviction suggest that the consequent never occurs without the antecedent.

Recommendations:

From the analysis conducted in this project, a deep understanding of the complexities of the data was achieved. The interplay between different variables and their impact on the severity of accidents was meticulously explored. The practical application of various classification algorithms illuminated the nuances of model performance in real-world data scenarios.

Among the classifiers evaluated, the Post Pruned Decision Tree performed exceptionally well for the selected dataset. It provided a robust model that was less prone to overfitting, thanks to its ensemble approach which combines multiple decision trees. Its inherent feature of evaluating feature importance was instrumental in understanding the underlying patterns within the data.

To further enhance the classification performance, future work could delve into more advanced ensemble methods, such as Gradient Boosting or Extreme Gradient Boosting, and other state-of-the-art algorithms/novel approaches which could refine predictions by correcting the mistakes of prior models. Additionally, hyperparameter tuning like Optuna or Hyperopt and other cross-validation techniques should be rigorously pursued to optimize the models. The incorporation of more data, perhaps from external sources or by featuring engineered variables, could provide additional insights and improve the model's predictive power.

Feature analysis revealed certain variables to be strongly associated with the target variable of accident severity. Notably, the number of vehicles involved, the speed limit, and the number of casualties were significantly correlated. These features could potentially be used to develop a predictive framework that authorities might leverage to allocate resources more efficiently and implement proactive safety measures.

Based on our analysis, we determined that the feature space can be optimally divided into two distinct clusters. This clustering might reflect underlying patterns or groupings in the data that could correlate with the target variable, ‘Accident Severity’.

The findings and methods from this project lay a foundation for a predictive system that could inform policy decisions and lead to the development of targeted strategies for accident prevention and response. The goal for improvement is ongoing, and this project serves as a promising step toward the application of machine learning in enhancing road safety and saving lives.

Appendix:

Phase 1: Feature Engineering and EDA

ML_UK_Accidents_EDA.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
import warnings

pd.set_option('display.float_format', lambda x: '%.2f' % x)
sns.set_style('darkgrid')
warnings.filterwarnings("ignore")

# Read in the data in chunks in for loop and then convert it to a dataframe
and see the time it takes to read the data
start_time = time.time()
accidents_vehicles_casualties = pd.DataFrame()

for chunk in pd.read_csv('UK_Accidents_Merger.csv', chunksize=200000,
low_memory=False):
    print('Number of chunks read: ', chunk.shape)
    accidents_vehicles_casualties = pd.concat([accidents_vehicles_casualties,
chunk])

print("Time taken to read the data: ", time.time() - start_time)

print("Shape of the dataframe: ", accidents_vehicles_casualties.shape)

#---------------------------------------------
# Data Wrangling
#---------------------------------------------

# Check for missing values
print("Missing Values:")
print(accidents_vehicles_casualties.isnull().sum()/accidents_vehicles_casualt
ies.shape[0])

# Drop or impute missing values
accidents_vehicles_casualties.dropna(inplace=True)

# Save the dataframe to a csv file
# accidents_vehicles_casualties.to_csv('UK_Accidents_Merger_Cleaned.csv',
index=False)

# Check for duplicates
print("Duplicates:")
print(accidents_vehicles_casualties.duplicated().sum())
```

```
# Check for data types
print("Data Types:")
print(accidents_vehicles_casualties.dtypes)

# Count number of numeric and non-numeric columns
print("Number of Numeric Columns:")
print(len(accidents_vehicles_casualties.select_dtypes(include=np.number).colu
mns))

print("Number of Non-Numeric Columns:")
print(len(accidents_vehicles_casualties.select_dtypes(exclude=np.number).colu
mns))

#-----------------------------------------
# Viusalizations using Matplotlib, Seaborn
#-----------------------------------------

# Convert date to year
accidents_vehicles_casualties['Date'] =
pd.to_datetime(accidents_vehicles_casualties['Date'])
accidents_vehicles_casualties['Year'] =
accidents_vehicles_casualties['Date'].dt.year

# Manipulate the data
accidents_vehicles_casualties['Junction_Detail'] =
accidents_vehicles_casualties['Junction_Detail'].replace({
    'T or staggered junction': 'Staggered Junction',
    'Private drive or entrance': 'Entrance',
    'Other junction': 'Others',
    'More then 4 arms (not roundabout)': 'Fours',
    'Mini-roundabout': 'Miniroundabout',
    'Not at junction or within 20 metres': 'No Junction',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Junction_Control'] =
accidents_vehicles_casualties['Junction_Control'].replace({
    'Authorised person': 'Authorised',
    'Data missing or out of range': 'Unknown',
    'Give way or uncontrolled': 'Uncontrolled',
    'Stop sign': 'Stop',
    'Auto traffic signal': 'Traffic Signal',
})


accidents_vehicles_casualties['Light_Conditions'] =
accidents_vehicles_casualties['Light_Conditions'].replace({
    'Darkness - no lighting': 'Darkness',
    'Darkness - lighting unknown': 'Unknown',
    'Darkness - lights lit': 'Light',
    'Darkness - lights unlit': 'Unlit'
})

accidents_vehicles_casualties['Weather_Conditions'] =
accidents_vehicles_casualties['Weather_Conditions'].replace({
    'Fine without high winds': 'Fine',
    'Raining without high winds': 'Rain',
```

```
    'Raining + high winds': 'Rainwinds',
    'Fine + high winds': 'Fine',
    'Fog or mist': 'Fog',
    'Snowing without high winds': 'Snow',
    'Snowing + high winds': 'Snowind',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Road_Surface_Conditions'] =
accidents_vehicles_casualties['Road_Surface_Conditions'].replace({
    'Dry': 'Dry',
    'Wet or damp': 'Wet',
    'Frost or ice': 'Ice',
    'Snow': 'Snow',
    'Flood over 3cm. deep': 'Flood',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Did_Police_Officer_Attend_Scene_of_Accident']
=
accidents_vehicles_casualties['Did_Police_Officer_Attend_Scene_of_Accident'].
replace(
    'No - accident was reported using a self completion  form (self rep
only)', 'Self')

accidents_vehicles_casualties['Vehicle_Type'] =
accidents_vehicles_casualties['Vehicle_Type'].replace({
    'Bus or coach (17 or more pass seats)': 'Bus',
    'Van / Goods 3.5 tonnes mgw or under': 'Van',
    'Taxi/Private hire car': 'Taxi',
    'Motorcycle 125cc and under': 'Motorcycle',
    'Motorcycle over 500cc': 'Motorcycle',
    'Goods 7.5 tonnes mgw and over': 'Goods',
    'Motorcycle 50cc and under': 'Motorcycle',
    'Motorcycle over 125cc and up to 500cc': 'Motorcycle',
    'Goods over 3.5t. and under 7.5t': 'Goods',
    'Other vehicle': 'Others',
    'Minibus (8 - 16 passenger seats)': 'Minibus',
    'Agricultural vehicle (includes diggers etc.)': 'Agricultural',
    'Motorcycle - unknown cc': 'Motorcycle'
})

accidents_vehicles_casualties['Towing_and_Articulation'] =
accidents_vehicles_casualties['Towing_and_Articulation'].replace({
    'No tow/articulation': 'No',
    'Articulated vehicle': 'Articulated',
    'Single trailer': 'Single',
    'Other tow': 'Others',
    'Double or multiple trailer': 'Multiple',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Vehicle_Manoeuvre'] =
accidents_vehicles_casualties['Vehicle_Manoeuvre'].replace({
    'Going ahead other': 'Going ahead',
    'Turning right': 'Right',
    'Waiting to go - held up': 'Waiting',
```

```
    'Slowing or stopping': 'Slowing',
    'Turning left': 'Left',
    'Moving off': 'Moving',
    'Waiting to turn right': 'Waiting',
    'Going ahead right-hand bend': 'Going ahead',
    'Going ahead left-hand bend': 'Going ahead',
    'Overtaking moving vehicle - offside': 'Overtaking',
    'Waiting to turn left': 'Waiting',
    'Overtaking static vehicle - offside': 'Overtaking',
    'Changing lane to left': 'Changing lane',
    'Changing lane to right': 'Changing lane',
    'U-turn': 'Uturn',
    'Overtaking - nearside': 'Overtaking',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Vehicle_Location-Restricted_Lane'] =
accidents_vehicles_casualties['Vehicle_Location-Restricted_Lane'].replace({
    'On main c\way - not in restricted lane': 'Mainlane',
    'Bus lane': 'Buslane',
    'Footway (pavement)': 'Footway',
    'Leaving lay-by or hard shoulder': 'Leavinglayby',
    'On lay-by or hard shoulder': 'Onlayby',
    'Busway (including guided busway)': 'Busway',
    'Cycle lane (on main carriageway)': 'Cyclelane',
    'Tram/Light rail track': 'Tramtrack',
    'Entering lay-by or hard shoulder': 'Enteringlayby',
    'Cycleway or shared use footway (not part of  main carriageway)':
'Cycleway',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Junction_Location'] =
accidents_vehicles_casualties['Junction_Location'].replace({
    'Approaching junction or waiting/parked at junction approach':
'Approaching',
    'Mid Junction - on roundabout or on main road': 'Mid',
    'Cleared junction or waiting/parked at junction exit': 'Cleared',
    'Entering from slip road': 'Entering',
    'Leaving main road into minor road': 'Leaving',
    'Entering main road from minor road': 'Entering',
    'Leaving roundabout': 'Leaving',
    'Entering roundabout': 'Entering',
    'Data missing or out of range': 'Unknown',
    'Not at or within 20 metres of junction': 'No Junction',
})

accidents_vehicles_casualties['1st_Point_of_Impact'] =
accidents_vehicles_casualties['1st_Point_of_Impact'].replace({
    'Did not impact': 'Nothing',
    'Data missing or out of range': 'Unknown'
})

accidents_vehicles_casualties['Was_Vehicle_Left_Hand_Drive?'] =
accidents_vehicles_casualties['Was_Vehicle_Left_Hand_Drive?'].replace({
    'Data missing or out of range': 'Unknown'
})
```

```
accidents_vehicles_casualties['Propulsion_Code'] =
accidents_vehicles_casualties['Propulsion_Code'].replace({
    'Gas/Bi-fuel': 'Gas',
    'Petrol/Gas (LPG)': 'LPG',
})

accidents_vehicles_casualties['Casualty_Class'] =
accidents_vehicles_casualties['Casualty_Class'].replace({
    'Driver or rider': 'Driver'
})

accidents_vehicles_casualties['Sex_of_Casualty'] =
accidents_vehicles_casualties['Sex_of_Casualty'].replace({
    'Data missing or out of range': 'Unknown'
})

accidents_vehicles_casualties['Car_Passenger'] =
accidents_vehicles_casualties['Car_Passenger'].replace({
    'Data missing or out of range': 'Unknown'
})

accidents_vehicles_casualties['Urban_or_Rural_Area'] =
accidents_vehicles_casualties['Urban_or_Rural_Area'].replace({
    1: 'Urban',
    2: 'Rural',
    3: 'Unallocated'
})

accidents_vehicles_casualties['Special_Conditions_at_Site'] =
accidents_vehicles_casualties['Special_Conditions_at_Site'].replace({
    'Auto traffic signal - out': 'No signal',
    'Auto signal part defective': 'Signal defect',
    'Oil or diesel': 'Oil',
    'Road sign or marking defective or obscured': 'Sign defect',
    'Road surface defective': 'Road defect'
})


# Histogram plots with KDE and probability plot
cont_columns =
list(accidents_vehicles_casualties.select_dtypes(include=[np.number]).columns
.values)

cont_columns.remove('Location_Easting_OSGR')
cont_columns.remove('Location_Northing_OSGR')
cont_columns.remove('Longitude')
cont_columns.remove('Latitude')
cont_columns.remove('Year')
cont_columns.remove('1st_Road_Number')
cont_columns.remove('2nd_Road_Number')
cont_columns.remove('Pedestrian_Crossing-Human_Control')
cont_columns.remove('Pedestrian_Crossing-Physical_Facilities')
cont_columns.remove('Vehicle_Reference_x')
cont_columns.remove('Vehicle_Reference_y')
cont_columns.remove('Pedestrian_Location')
cont_columns.remove('Pedestrian_Movement')
```

```
cont_columns.remove('Pedestrian_Road_Maintenance_Worker')
cont_columns.remove('Casualty_Home_Area_Type')
cont_columns.remove('Age_Band_of_Driver')
cont_columns.remove('Age_Band_of_Casualty')
cont_columns.remove('Driver_Home_Area_Type')
cont_columns.remove('Bus_or_Coach_Passenger')

print("Numerical Columns: \n", cont_columns)

cat_cols =
list(accidents_vehicles_casualties.select_dtypes(include=['object']).columns)

cat_cols.remove('Accident_Severity')

#----------------------------------------
# Line plots
#----------------------------------------

# Create a line graph of the number of accidents per year
accidents_by_year =
accidents_vehicles_casualties.groupby('Year').count()['Accident_Index'].reset
_index()

plt.figure(figsize=(10, 6))
sns.lineplot(x='Year', y='Accident_Index', data=accidents_by_year,
palette='bright')
plt.xlabel('Year')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Year')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Create a line graph of the number of casualties per year
casualties_by_year =
accidents_vehicles_casualties.groupby('Year').sum()['Number_of_Casualties'].r
eset_index()

plt.figure(figsize=(10, 6))
sns.lineplot(x='Year', y='Number_of_Casualties', data=casualties_by_year,
palette='bright')
plt.xlabel('Year')
plt.ylabel('Number of Casualties')
plt.title('Number of Casualties per Year')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

#----------------------------------------
# Bar graphs
#----------------------------------------

# Create a bar chart of the number of accidents per year with numbers on the
top of plot as well as a line gping thorugh all the plots
plt.figure(figsize=(10, 6))
sns.barplot(x='Year', y='Accident_Index', data=accidents_by_year,
palette='bright')
```

```
plt.xlabel('Year')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Year')
plt.xticks(rotation=45)

for k, v in accidents_by_year['Accident_Index'].items():
    if v > 1000:
        plt.text(k, v + 100, str(v), fontsize=12, color="black", ha="center",
                 rotation=90, va="bottom")

accidents_by_year['Accident_Index'].plot(label='Number of Accidents',
color='black', linewidth=3,
                                         ax=plt.twinx(), marker='o',
markersize=10)
plt.grid(True)
plt.show()

# Create a bar chart of the number of accidents per year per severity
accidents_by_year_severity = accidents_vehicles_casualties.groupby(['Year',
'Accident_Severity']).count()['Accident_Index'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Year', y='Accident_Index', hue='Accident_Severity',
data=accidents_by_year_severity, palette='bright')
plt.xlabel('Year')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Year per Severity')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Create a bar chart of casualties per year
plt.figure(figsize=(10, 6))
sns.barplot(x='Year', y='Number_of_Casualties', data=casualties_by_year,
palette='bright')
plt.xlabel('Year')
plt.ylabel('Number of Casualties')
plt.title('Number of Casualties per Year')
plt.xticks(rotation=45)

for k, v in casualties_by_year['Number_of_Casualties'].items():
    if v > 1000:
        plt.text(k, v + 100, str(v), fontsize=12, color="black", ha="center",
                 rotation=90, va="bottom")

casualties_by_year['Number_of_Casualties'].plot(label='Number of Casualties',
color='black', linewidth=3,
                                                ax=plt.twinx(), marker='o',
markersize=10)
plt.grid(True)
plt.show()

# Create a bar chart of casualties per year per severity
casualties_by_year_severity = accidents_vehicles_casualties.groupby(['Year',
'Accident_Severity']).sum()['Number_of_Casualties'].reset_index()

plt.figure(figsize=(10, 6))
```

```
sns.barplot(x='Year', y='Number_of_Casualties', hue='Accident_Severity',
data=casualties_by_year_severity, palette='bright')
plt.xlabel('Year')
plt.ylabel('Number of Casualties')
plt.title('Number of Casualties per Year per Severity')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Create a bar graph that checks whether majority of the accidents happen
during the urban or rural areas, with line graph of casulties based on the
settlements
accidents_by_settlement =
accidents_vehicles_casualties.groupby('Urban_or_Rural_Area').count()['Acciden
t_Index'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Urban_or_Rural_Area', y='Accident_Index',
data=accidents_by_settlement, palette='bright')
plt.xlabel('Settlement Type')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Settlement Type')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

time.sleep(10)
time.sleep(5)

# Create a graph of urban/rural accidents based on severity
accidents_by_settlement_severity =
accidents_vehicles_casualties.groupby(['Urban_or_Rural_Area',
'Accident_Severity']).count()['Accident_Index'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Urban_or_Rural_Area', y='Accident_Index',
hue='Accident_Severity', data=accidents_by_settlement_severity,
palette='bright')
plt.xlabel('Settlement Type')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Settlement Type per Severity')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Create a graph explaining of many vehicles were involved in accidents based
on the severity, and how many casulaties were involved in the accidents based
on the severity
accidents_by_vehicles =
accidents_vehicles_casualties.groupby('Number_of_Vehicles').count()['Accident
_Index'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Number_of_Vehicles', y='Accident_Index',
data=accidents_by_vehicles, palette='bright')
plt.xlabel('Number of Vehicles')
plt.ylabel('Number of Accidents')
```

```
plt.title('Number of Accidents per Number of Vehicles')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Create a graph to check whether many accidents happened in the weekdays or
the weekends
accidents_by_day =
accidents_vehicles_casualties.groupby('Day_of_Week').count()['Accident_Index'
].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Day_of_Week', y='Accident_Index', data=accidents_by_day,
palette='bright')
plt.xlabel('Day of Week')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Day of Week')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Create a graph to check whether many accidents happened in the weekdays or
the weekends based on severity
accidents_by_day_severity =
accidents_vehicles_casualties.groupby(['Day_of_Week',
'Accident_Severity']).count()['Accident_Index'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Day_of_Week', y='Accident_Index', hue='Accident_Severity',
data=accidents_by_day_severity, palette='bright')
plt.xlabel('Day of Week')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Day of Week per Severity')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Create a bar graph to check whether accident severity is based on the type
of road
accidents_by_road = accidents_vehicles_casualties.groupby(['Road_Type',
'Accident_Severity']).count()['Accident_Index'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Road_Type', y='Accident_Index', data=accidents_by_road,
palette='bright', hue='Accident_Severity')
plt.xlabel('Road Type')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Road Type')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

#Create a groupby function depicting the road surface conditions during the
accidents
accidents_by_surface =
accidents_vehicles_casualties.groupby('Road_Surface_Conditions').count()['Acc
ident_Index'].reset_index()
```

```
# Create a bar graph to check whether accident severity is based on the
surface condition of the road
accidents_by_surface_severity =
accidents_vehicles_casualties.groupby(['Road_Surface_Conditions',
'Accident_Severity']).count()['Accident_Index'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Road_Surface_Conditions', y='Accident_Index',
data=accidents_by_surface_severity, palette='bright',
hue='Accident_Severity')
plt.xlabel('Road Surface Condition')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Road Surface Condition')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Create a graph depicting severity if accidents based on the speed limits
accidents_by_speed = accidents_vehicles_casualties.groupby(['Speed_limit',
'Accident_Severity']).count()['Accident_Index'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Speed_limit', y='Accident_Index', data=accidents_by_speed,
palette='bright', hue='Accident_Severity')
plt.xlabel('Speed Limit')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Speed Limit')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Create a graph depicting which kind of vehicles were involved in accidents
accidents_by_vehicle = accidents_vehicles_casualties.groupby(['Vehicle_Type',
'Accident_Severity']).count()['Accident_Index'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Vehicle_Type', y='Accident_Index', data=accidents_by_vehicle,
palette='bright', hue='Accident_Severity')
plt.xlabel('Vehicle Type')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Vehicle Type')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

#Create a groupby function depicting the weather conditions during the
accidents
accidents_by_weather =
accidents_vehicles_casualties.groupby('Weather_Conditions').count()['Accident
_Index'].reset_index()

# Create a graph depicting the weather conditions during the accidents based
on severity
accidents_by_weather_severity =
accidents_vehicles_casualties.groupby(['Weather_Conditions',
'Accident_Severity']).count()['Accident_Index'].reset_index()
```

```
plt.figure(figsize=(10, 6))
sns.barplot(x='Weather_Conditions', y='Accident_Index',
data=accidents_by_weather_severity, palette='bright',
hue='Accident_Severity')
plt.xlabel('Weather Condition')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Weather Condition')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

#Create a groupby function depicting the light conditions during the
accidents
accidents_by_light =
accidents_vehicles_casualties.groupby('Light_Conditions').count()['Accident_I
ndex'].reset_index()

# Create a graph depicting the light conditions during the accidents based on
severity
accidents_by_light_severity =
accidents_vehicles_casualties.groupby(['Light_Conditions',
'Accident_Severity']).count()['Accident_Index'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Light_Conditions', y='Accident_Index',
data=accidents_by_light_severity, palette='bright', hue='Accident_Severity')
plt.xlabel('Light Condition')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Light Condition')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

#Create a groupby function depicting the special conditions during the
accidents
accidents_by_special =
accidents_vehicles_casualties.groupby('Special_Conditions_at_Site').count()['
Accident_Index'].reset_index()

# Create a graph depicting the special conditions during the accidents based
on severity
accidents_by_special_severity =
accidents_vehicles_casualties.groupby(['Special_Conditions_at_Site',
'Accident_Severity']).count()['Accident_Index'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Special_Conditions_at_Site', y='Accident_Index',
data=accidents_by_special_severity, palette='bright',
hue='Accident_Severity')
plt.xlabel('Special Condition')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Special Condition')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

```
# Create a graph of how the accidents happened
accidents_happen_how =
accidents_vehicles_casualties.groupby(['Skidding_and_Overturning',
'Accident_Severity']).count()['Accident_Index'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Skidding_and_Overturning', y='Accident_Index',
data=accidents_happen_how, palette='bright', hue='Accident_Severity')
plt.xlabel('Accident Incidents')
plt.ylabel('Number of Accidents')
plt.title('How the Accidents Happened')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Create a graph of what is the 1st point of impact
accidents_by_impact =
accidents_vehicles_casualties.groupby(['1st_Point_of_Impact',
'Accident_Severity']).count()['Accident_Index'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='1st_Point_of_Impact', y='Accident_Index',
data=accidents_by_impact, palette='bright', hue='Accident_Severity')
plt.xlabel('1st Point of Impact')
plt.ylabel('Number of Accidents')
plt.title('1st Point of Impact')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Create a graph based on the how many accidents were reported to the police
accidents_by_police =
accidents_vehicles_casualties.groupby('Did_Police_Officer_Attend_Scene_of_Acc
ident').count()['Accident_Index'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Did_Police_Officer_Attend_Scene_of_Accident',
y='Accident_Index', data=accidents_by_police, palette='bright')
plt.xlabel('Police Attended')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents reported to Police')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

time.sleep(10)
time.sleep(10)
time.sleep(10)

# Create a graph depicting no of accidents to the sex of the casualty
accidents_by_sex =
accidents_vehicles_casualties.groupby('Sex_of_Casualty').count()['Accident_In
dex'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Sex_of_Casualty', y='Accident_Index', data=accidents_by_sex,
```

```
palette='bright')
plt.xlabel('Sex of Casualty')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Sex of Casualty')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Create a graph depicting no of accidents to the sex of the driver
accidents_by_sex_driver =
accidents_vehicles_casualties.groupby('Sex_of_Driver').count()['Accident_Inde
x'].reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Sex_of_Driver', y='Accident_Index',
data=accidents_by_sex_driver, palette='bright')
plt.xlabel('Sex of the Driver')
plt.ylabel('Number of Accidents')
plt.title('Number of Accidents per Sex of the Driver')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

#----------------------------------------
# Pie charts
#----------------------------------------

def pie_plts(df, column):
    plt.figure(figsize=(10, 6))
    df[column].value_counts().plot(kind='pie', autopct='%1.1f%%',
shadow=True)
    plt.title(column)
    time.sleep(5)
    plt.show()

# Create a pie chart of accident severity
pie_plts(accidents_vehicles_casualties, 'Accident_Severity')

# Create a pie chart of settlement type
pie_plts(accidents_vehicles_casualties, 'Urban_or_Rural_Area')

# Create a pie chart of road type accidents
pie_plts(accidents_vehicles_casualties, 'Road_Type')

# Create a pie chart of road surface conditions
pie_plts(accidents_vehicles_casualties, 'Road_Surface_Conditions')

# Create a pie chart of weather conditions
pie_plts(accidents_vehicles_casualties, 'Weather_Conditions')

# Create a pie chart of light conditions
pie_plts(accidents_vehicles_casualties, 'Light_Conditions')

# Create a pie chart of whether police attended the scene of accident
pie_plts(accidents_vehicles_casualties,
'Did_Police_Officer_Attend_Scene_of_Accident')
```

```
time.sleep(10)

# Create a pie chart of first point of impact
pie_plts(accidents_vehicles_casualties, '1st_Point_of_Impact')

time.sleep(10)

#----------------------------------------
# Count plots
#----------------------------------------

def count_plts(df, column):
    plt.figure(figsize=(10, 6))
    sns.countplot(x=column, data=df)
    plt.xlabel(column)
    plt.ylabel('Number of Accidents')
    plt.title('Number of Accidents per ' + column)
    plt.xticks(rotation=45)
    plt.grid(True)
    time.sleep(10)
    plt.show()

# Count plot of number of vechicles affected
count_plts(accidents_vehicles_casualties, 'Number_of_Vehicles')

# Count plot of number of casualties affected
count_plts(accidents_vehicles_casualties, 'Number_of_Casualties')


# Count plot of age of the vehicles that were involved in accidents
count_plts(accidents_vehicles_casualties, 'Age_of_Vehicle')

time.sleep(10)
time.sleep(10)

#----------------------------------------
# Pairplot
#----------------------------------------

# Create a pairplot of selected features
pairplot_vars = pd.DataFrame(accidents_vehicles_casualties, columns=
cont_columns)
pairplot_vars['Accident_Severity'] =
accidents_vehicles_casualties['Accident_Severity']

sns.pairplot(data=pairplot_vars, hue="Accident_Severity" , palette='bright')
plt.title('Pairplot of Selected Features')
plt.show()
```

ML_UK_Accidents_Feature_Engg.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split as tts
import statsmodels.api as sm
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import TruncatedSVD
from statsmodels.stats.outliers_influence import variance_inflation_factor
from prettytable import PrettyTable
import warnings

pd.set_option('display.float_format', lambda x: '%.2f' % x)
sns.set_style('darkgrid')
warnings.filterwarnings("ignore")

# Read in the data in chunks in for loop and then convert it to a dataframe
and see the time it takes to read the data
start_time = time.time()
accidents_vehicles_casualties = pd.DataFrame()

for chunk in pd.read_csv('UK_Accidents_Merger.csv', chunksize=200000,
low_memory=False):
    print('Number of chunks read: ', chunk.shape)
    accidents_vehicles_casualties = pd.concat([accidents_vehicles_casualties,
chunk])

print("Time taken to read the data: ", time.time() - start_time)

print("Shape of the dataframe: ", accidents_vehicles_casualties.shape)

# Feature Engineering
# Check for missing values
print("Missing Values:")
print(accidents_vehicles_casualties.isnull().sum()/accidents_vehicles_casualt
ies.shape[0])

# Drop or impute missing values
accidents_vehicles_casualties.dropna(inplace=True)

# Check for duplicates
print("Duplicates:")
print(accidents_vehicles_casualties.duplicated().sum())

accidents_vehicles_casualties['Date'] =
pd.to_datetime(accidents_vehicles_casualties['Date'])
accidents_vehicles_casualties['Year'] =
```

```
accidents_vehicles_casualties['Date'].dt.year

# Drop unnecessary columns
accidents_vehicles_casualties.drop(['Location_Easting_OSGR',
'Location_Northing_OSGR', 'Longitude', 'Latitude',
                                    'Police_Force',
'Local_Authority_(District)', 'Local_Authority_(Highway)',
                                    '1st_Road_Number', '2nd_Road_Number',
'Accident_Index', 'Date', 'Time',
                                    'LSOA_of_Accident_Location', 'Year',
'Journey_Purpose_of_Driver', 'Sex_of_Driver',
                                    'Skidding_and_Overturning',
'Special_Conditions_at_Site', 'Carriageway_Hazards',
                                    'Hit_Object_in_Carriageway',
'Hit_Object_off_Carriageway', 'Vehicle_Leaving_Carriageway',
                                    'Hit_Object_off_Carriageway',
'Driver_IMD_Decile', 'Age_of_Casualty', 'Casualty_Type',
                                    'Vehicle_Reference_x',
'Vehicle_Reference_y', 'Pedestrian_Crossing-Human_Control',
                                    'Pedestrian_Crossing-
Physical_Facilities', 'Pedestrian_Road_Maintenance_Worker',
'Casualty_Home_Area_Type',
                                    'Age_Band_of_Driver',
'Age_Band_of_Casualty', 'Pedestrian_Location', 'Pedestrian_Movement',
'Bus_or_Coach_Passenger',
                                    'Driver_Home_Area_Type'], axis=1,
inplace=True)

# New Shape of the dataframe
print("New Shape of the dataframe: ", accidents_vehicles_casualties.shape)

accidents_vehicles_casualties['Junction_Detail'] =
accidents_vehicles_casualties['Junction_Detail'].replace({
    'T or staggered junction': 'Staggered Junction',
    'Private drive or entrance': 'Entrance',
    'Other junction': 'Others',
    'More then 4 arms (not roundabout)': 'Fours',
    'Mini-roundabout': 'Miniroundabout',
    'Not at junction or within 20 metres': 'No Junction',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Junction_Control'] =
accidents_vehicles_casualties['Junction_Control'].replace({
    'Authorised person': 'Authorised',
    'Data missing or out of range': 'Unknown',
    'Give way or uncontrolled': 'Uncontrolled',
    'Stop sign': 'Stop',
    'Auto traffic signal': 'Traffic Signal',
})


accidents_vehicles_casualties['Light_Conditions'] =
accidents_vehicles_casualties['Light_Conditions'].replace({
    'Darkness - no lighting': 'Darkness',
    'Darkness - lighting unknown': 'Unknown',
    'Darkness - lights lit': 'Light',
```

```
    'Darkness - lights unlit': 'Unlit'
})

accidents_vehicles_casualties['Weather_Conditions'] =
accidents_vehicles_casualties['Weather_Conditions'].replace({
    'Fine without high winds': 'Fine',
    'Raining without high winds': 'Rain',
    'Raining + high winds': 'Rainwinds',
    'Fine + high winds': 'Fine',
    'Fog or mist': 'Fog',
    'Snowing without high winds': 'Snow',
    'Snowing + high winds': 'Snowind',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Road_Surface_Conditions'] =
accidents_vehicles_casualties['Road_Surface_Conditions'].replace({
    'Dry': 'Dry',
    'Wet or damp': 'Wet',
    'Frost or ice': 'Ice',
    'Snow': 'Snow',
    'Flood over 3cm. deep': 'Flood',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Did_Police_Officer_Attend_Scene_of_Accident']
=
accidents_vehicles_casualties['Did_Police_Officer_Attend_Scene_of_Accident'].
replace(
    'No - accident was reported using a self completion  form (self rep
only)', 'Self')

accidents_vehicles_casualties['Vehicle_Type'] =
accidents_vehicles_casualties['Vehicle_Type'].replace({
    'Bus or coach (17 or more pass seats)': 'Bus',
    'Van / Goods 3.5 tonnes mgw or under': 'Van',
    'Taxi/Private hire car': 'Taxi',
    'Motorcycle 125cc and under': 'Motorcycle',
    'Motorcycle over 500cc': 'Motorcycle',
    'Goods 7.5 tonnes mgw and over': 'Goods',
    'Motorcycle 50cc and under': 'Motorcycle',
    'Motorcycle over 125cc and up to 500cc': 'Motorcycle',
    'Goods over 3.5t. and under 7.5t': 'Goods',
    'Other vehicle': 'Others',
    'Minibus (8 - 16 passenger seats)': 'Minibus',
    'Agricultural vehicle (includes diggers etc.)': 'Agricultural',
    'Motorcycle - unknown cc': 'Motorcycle'
})

accidents_vehicles_casualties['Towing_and_Articulation'] =
accidents_vehicles_casualties['Towing_and_Articulation'].replace({
    'No tow/articulation': 'No',
    'Articulated vehicle': 'Articulated',
    'Single trailer': 'Single',
    'Other tow': 'Others',
    'Double or multiple trailer': 'Multiple',
    'Data missing or out of range': 'Unknown',
```

```
})

accidents_vehicles_casualties['Vehicle_Manoeuvre'] =
accidents_vehicles_casualties['Vehicle_Manoeuvre'].replace({
    'Going ahead other': 'Going ahead',
    'Turning right': 'Right',
    'Waiting to go - held up': 'Waiting',
    'Slowing or stopping': 'Slowing',
    'Turning left': 'Left',
    'Moving off': 'Moving',
    'Waiting to turn right': 'Waiting',
    'Going ahead right-hand bend': 'Going ahead',
    'Going ahead left-hand bend': 'Going ahead',
    'Overtaking moving vehicle - offside': 'Overtaking',
    'Waiting to turn left': 'Waiting',
    'Overtaking static vehicle - offside': 'Overtaking',
    'Changing lane to left': 'Changing lane',
    'Changing lane to right': 'Changing lane',
    'U-turn': 'Uturn',
    'Overtaking - nearside': 'Overtaking',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Vehicle_Location-Restricted_Lane'] =
accidents_vehicles_casualties['Vehicle_Location-Restricted_Lane'].replace({
    'On main c\way - not in restricted lane': 'Mainlane',
    'Bus lane': 'Buslane',
    'Footway (pavement)': 'Footway',
    'Leaving lay-by or hard shoulder': 'Leavinglayby',
    'On lay-by or hard shoulder': 'Onlayby',
    'Busway (including guided busway)': 'Busway',
    'Cycle lane (on main carriageway)': 'Cyclelane',
    'Tram/Light rail track': 'Tramtrack',
    'Entering lay-by or hard shoulder': 'Enteringlayby',
    'Cycleway or shared use footway (not part of  main carriageway)':
'Cycleway',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Junction_Location'] =
accidents_vehicles_casualties['Junction_Location'].replace({
    'Approaching junction or waiting/parked at junction approach':
'Approaching',
    'Mid Junction - on roundabout or on main road': 'Mid',
    'Cleared junction or waiting/parked at junction exit': 'Cleared',
    'Entering from slip road': 'Entering',
    'Leaving main road into minor road': 'Leaving',
    'Entering main road from minor road': 'Entering',
    'Leaving roundabout': 'Leaving',
    'Entering roundabout': 'Entering',
    'Data missing or out of range': 'Unknown',
    'Not at or within 20 metres of junction': 'No Junction',
})

accidents_vehicles_casualties['1st_Point_of_Impact'] =
accidents_vehicles_casualties['1st_Point_of_Impact'].replace({
    'Did not impact': 'Nothing',
```

```
    'Data missing or out of range': 'Unknown'
})

accidents_vehicles_casualties['Was_Vehicle_Left_Hand_Drive?'] =
accidents_vehicles_casualties['Was_Vehicle_Left_Hand_Drive?'].replace({
    'Data missing or out of range': 'Unknown'
})

accidents_vehicles_casualties['Propulsion_Code'] =
accidents_vehicles_casualties['Propulsion_Code'].replace({
    'Gas/Bi-fuel': 'Gas',
    'Petrol/Gas (LPG)': 'LPG',
})

accidents_vehicles_casualties['Casualty_Class'] =
accidents_vehicles_casualties['Casualty_Class'].replace({
    'Driver or rider': 'Driver'
})

accidents_vehicles_casualties['Sex_of_Casualty'] =
accidents_vehicles_casualties['Sex_of_Casualty'].replace({
    'Data missing or out of range': 'Unknown'
})

accidents_vehicles_casualties['Car_Passenger'] =
accidents_vehicles_casualties['Car_Passenger'].replace({
    'Data missing or out of range': 'Unknown'
})

# Define categorical and continuous columns
cat_cols =
list(accidents_vehicles_casualties.select_dtypes(include=['object']).columns)
cat_cols.remove('Accident_Severity')

cont_cols =
list(accidents_vehicles_casualties.select_dtypes(include=[np.number]).columns
)

# Anomaly Detection
# Check for outliers
plt.figure(figsize=(20, 10))
sns.boxplot(data=accidents_vehicles_casualties, orient='h')
plt.show()

# Remove outliers
def iqr(df):
    q1 = df.quantile(0.25)
    q3 = df.quantile(0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr
    return lower, upper

lower, upper = iqr(accidents_vehicles_casualties['Engine_Capacity_(CC)'])
accidents_vehicles_casualties =
accidents_vehicles_casualties[(accidents_vehicles_casualties['Engine_Capacity
_(CC)'] > lower) & (accidents_vehicles_casualties['Engine_Capacity_(CC)'] <
```

```
upper)]

# After removing outliers
plt.figure(figsize=(20, 10))
sns.boxplot(data=accidents_vehicles_casualties, orient='h')
plt.show()

# Heatmap of covariance matrix
plt.figure(figsize=(20, 10))
sns.heatmap(accidents_vehicles_casualties[cont_cols].cov(), annot=True,
cmap='icefire')
plt.title('Covariance Matrix')
plt.show()

# Heatmap of correlation matrix
plt.figure(figsize=(20, 10))
sns.heatmap(accidents_vehicles_casualties[cont_cols].corr(), annot=True,
cmap='icefire')
plt.title('Correlation Matrix')
plt.show()

# One-hot the categorical columns
accidents_vehicles_casualties = pd.get_dummies(accidents_vehicles_casualties,
columns=cat_cols, drop_first=True)

# Scale the continuous columns
def scale(x):
    return (x - x.min()) / (x.max() - x.min())

accidents_vehicles_casualties[cont_cols] =
accidents_vehicles_casualties[cont_cols].apply(scale, axis=0)


accidents_vehicles_casualties = accidents_vehicles_casualties.iloc[: : 2]

# Divide the data into features and target
X, y = accidents_vehicles_casualties.drop('Accident_Severity', axis=1),
accidents_vehicles_casualties['Accident_Severity']

#Oversample the minority class
oversample = SMOTE()
X_smote, y_smote = oversample.fit_resample(X, y)

# Check the balance of the classes
print('Processisng SMOTE...')
fig, ax = plt.subplots(1, 2, figsize = (10, 5))
y.value_counts().plot.pie(explode = [0, 0.1, 0.2], autopct = "%1.1f%%", ax =
ax[0], colors = ['#ff9999', '#66b3ff', '#964B00'],
                          shadow = True)
ax[0].set_title("Genres Before SMOTE")
ax[0].set_ylabel('')
y_smote.value_counts().plot.pie(explode = [0, 0.1, 0.2], autopct = "%1.1f%%",
ax = ax[1],  colors = ['#66b3ff', '#ff9999', '#964B00'],
                               shadow = True)
ax[1].set_title("Genres After SMOTE")
ax[1].set_ylabel('')
```

```
print("Overall dataset values have been increased.")
print("Original size:\n", y.value_counts())
print("New size:\n", y_smote.value_counts())
plt.show()

X_smote = sm.add_constant(X_smote)

print('New size of dataset:', X_smote.shape, y_smote.shape)

# Split the data into train and test
X_train, X_test, y_train, y_test = tts(X_smote, y_smote, test_size=0.2,
random_state=5805, shuffle=True)

print('Train size:', X_train.shape, y_train.shape)
print('Test size:', X_test.shape, y_test.shape)

#Feature Selection

# Create a table that mentions the feature selection technique and the number
of features selected and removed
feature_selection = pd.DataFrame(columns=['Feature Selection Technique',
'Number of Features Selected', 'Number of Features Removed'])

# 1. Perform PCA Analysis and Conditional Number
pca = PCA(svd_solver="full", n_components=0.9, random_state=5805)
X_pca = pca.fit_transform(X_smote)
print("Original Shape: ", X_smote.shape)
print("Reduced Shape: ", X_pca.shape)

print("Number of features needed to explain more than 90% of the dependent
variance:",
    np.where(np.cumsum(pca.explained_variance_ratio_) > 0.9)[0][0] + 1, )

plt.figure(figsize=(10, 10))
plt.plot(
    np.arange(1, len(np.cumsum(pca.explained_variance_ratio_)) + 1, 1),
    np.cumsum(pca.explained_variance_ratio_), label="Cumulative Explained
Variance",)
plt.xticks(np.arange(1, len(np.cumsum(pca.explained_variance_ratio_)) + 1,
1))
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("Cumulative Explained Variance vs Number of Components in PCA")
plt.grid(True)
plt.axvline(x=(np.where(np.cumsum(pca.explained_variance_ratio_) > 0.9)[0][0]
+ 1, ),
            color="red", linestyle="--")
plt.axhline(y=0.9, color="black", linestyle="--")
plt.legend()
plt.show()

# Conditional Number of Original and Reduced data
print('Condition Number of Original data:', np.linalg.cond(X_smote))
print('Condition Number of Reduced data:', np.linalg.cond(X_pca))

feature_selection = feature_selection.append({'Feature Selection Technique':
'PCA',
```

```
                                                'Number of Features
Selected': np.where(np.cumsum(pca.explained_variance_ratio_) > 0.9)[0][0] +
1,
                                                'Number of Features Removed':
X_smote.shape[1] - np.where(np.cumsum(pca.explained_variance_ratio_) >
0.9)[0][0] + 1,},
                                                 ignore_index=True)

# 2. Perform Random Forest Analysis
rfc = RandomForestClassifier(n_estimators=100, random_state=5805, n_jobs=-1)
rfc.fit(X_train, y_train)

def plot_feature_importance_with_elimination(importance, names, model_type,
threshold=0.01):
    feature_importance = np.array(importance)
    feature_names = np.array(names)
    data = {"feature_names": feature_names, "feature_importance":
feature_importance}
    fi_df = pd.DataFrame(data)

    fi_df.sort_values(by=["feature_importance"], ascending=False,
inplace=True)

    plt.figure(figsize=(10, 8))
    sns.barplot(x=fi_df["feature_importance"], y=fi_df["feature_names"])
    plt.title(f"{model_type} Feature Importance")
    plt.xlabel("Feature Importance")
    plt.ylabel("Feature Names")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    removed_features = fi_df[fi_df["feature_importance"] <
threshold]["feature_names"]
    print("Removed Features:", removed_features.tolist())

    selected_features = fi_df[fi_df["feature_importance"] >=
threshold]["feature_names"]
    print("Selected Features:", selected_features.tolist())

    return removed_features, selected_features


removed, selected = plot_feature_importance_with_elimination(
    rfc.feature_importances_, X_train.columns, "RANDOM FOREST")

feature_selection = feature_selection.append({'Feature Selection Technique':
'Random Forest',
                                                'Number of Features
Selected': len(selected),
                                                'Number of Features Removed':
len(removed)},
                                                 ignore_index=True)

# 3. Singular Value Decomposition Analysis
# First SVD with full components
svd_full = TruncatedSVD(n_components=X_smote.shape[1] - 1, random_state=5805,
```

```
algorithm='arpack')
X_svd_full = svd_full.fit_transform(X_smote)

cumulative_variance_ratio = np.cumsum(svd_full.explained_variance_ratio_)

n_components_90 = np.argmax(cumulative_variance_ratio >= 0.9) + 1

# Now SVD with 90% variance explained
svd_90 = TruncatedSVD(n_components=n_components_90)
X_reduced_90 = svd_90.fit_transform(X_smote)

print("Original Shape: ", X_smote.shape)
print("Reduced Shape: ", X_reduced_90.shape)

plt.figure(figsize=(10, 10))
plt.plot(np.arange(1, len(cumulative_variance_ratio) + 1, 1),
cumulative_variance_ratio,
         label="Cumulative Explained Variance", marker='o')
plt.xticks(np.arange(1, len(cumulative_variance_ratio) + 1, 1))
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.xticks(np.arange(1, len(cumulative_variance_ratio) + 1, 10))
plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)
plt.title("Cumulative Explained Variance vs Number of Components in SVD")
plt.axvline(x=n_components_90, color="red", linestyle="--")
plt.axhline(y=0.9, color='black', linestyle="--")
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()


# Conditional Number of Original and Reduced data
print('Condition Number of Original data:', np.linalg.cond(X_smote))
print('Condition Number of Reduced data:', np.linalg.cond(X_reduced_90))

feature_selection = feature_selection.append({'Feature Selection Technique':
'Singular Value Decomposition Analysis',
                                              'Number of Features
Selected': n_components_90,
                                              'Number of Features Removed':
X_smote.shape[1] - n_components_90},
                                               ignore_index=True)

# 4. Variance Inflation Factor Analysis
# Calculate VIF and count the number of features removed
vif = pd.DataFrame()
vif["vif"] = [variance_inflation_factor(X_smote, i) for i in
range(X_smote.shape[1])]
vif["Features"] = X_smote.columns
print(vif)

# Drop the columns with highest VIF > 10
vif_filtered = vif[vif['vif']<10]
print(vif_filtered)

vif_iter1 = X.shape[1] - vif_filtered.shape[0]
print("Number of features removed:", vif_iter1)
```

```
# Calculate VIF again and count the number of features removed
vif_second = pd.DataFrame()
vif_second["vif"] =
[variance_inflation_factor(X_smote[vif_filtered['Features']], i) for i in
range(X_smote[vif_filtered['Features']].shape[1])]
vif_second["Features"] = X_smote[vif_filtered['Features']].columns
print(vif_second)

# Drop the columns with highest VIF > 10
vif_filtered_second = vif_second[vif_second['vif']<10]
print(vif_filtered_second)

vif_iter2 = vif_filtered.shape[0] - vif_filtered_second.shape[0]
print("Number of features removed:", vif_iter2)

# Calculate VIF again and count the number of features removed
vif_third = pd.DataFrame()
vif_third["vif"] =
[variance_inflation_factor(X_smote[vif_filtered_second['Features']], i) for i
in range(X_smote[vif_filtered_second['Features']].shape[1])]
vif_third["Features"] = X_smote[vif_filtered_second['Features']].columns
print(vif_third)

# Drop the columns with highest VIF > 10
vif_filtered_third = vif_third[vif_third['vif']<10]
print(vif_filtered_third)

vif_iter3 = vif_filtered_second.shape[0] - vif_filtered_third.shape[0]
print("Number of features removed:", vif_iter3)

# Calculate total number of features removed
vif_total = vif_iter1 + vif_iter2 + vif_iter3
print("Total number of features removed:", vif_total)

feature_selection = feature_selection.append({'Feature Selection Technique':
'Variance Inflation Factor',
                                              'Number of Features
Selected': vif_filtered_third.shape[0],
                                              'Number of Features Removed':
vif_total},
                                               ignore_index=True)

# Put all the feature selection techniques in a table
def create_feature_selecction_table(df, name):
    x = PrettyTable()
    x.title = f"{name} Comparison"
    x.field_names = df.columns

    for index, row in df.iterrows():
        x.add_row(row)

    print(x)

create_feature_selecction_table(feature_selection, 'Feature
Selection/Dimensionality Reduction')
```

```
# Best Feature Selection Technique is Random Forest. So, we will use the
features selected by Random Forest.
```

Phase 2: Regression Analysis

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
import statsmodels.api as sm
from sklearn.model_selection import train_test_split as tts
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error as mae
import warnings

pd.set_option('display.float_format', lambda x: '%.2f' % x)
sns.set_style('darkgrid')
warnings.filterwarnings("ignore")

# Read in the data in chunks in for loop and then convert it to a dataframe
and see the time it takes to read the data
start_time = time.time()
accidents_vehicles_casualties = pd.DataFrame()

for chunk in pd.read_csv('UK_Accidents_Merger.csv', chunksize=200000,
low_memory=False):
    print('Number of chunks read: ', chunk.shape)
    accidents_vehicles_casualties = pd.concat([accidents_vehicles_casualties,
chunk])

print("Time taken to read the data: ", time.time() - start_time)

print("Shape of the dataframe: ", accidents_vehicles_casualties.shape)

# Feature Engineering
# Check for missing values
print("Missing Values:")
print(accidents_vehicles_casualties.isnull().sum()/accidents_vehicles_casualt
ies.shape[0])

# Drop or impute missing values
accidents_vehicles_casualties.dropna(inplace=True)

# Check for duplicates
print("Duplicates:")
print(accidents_vehicles_casualties.duplicated().sum())

accidents_vehicles_casualties['Date'] =
pd.to_datetime(accidents_vehicles_casualties['Date'])
accidents_vehicles_casualties['Year'] =
accidents_vehicles_casualties['Date'].dt.year

# Drop unnecessary columns
```

```
accidents_vehicles_casualties.drop(['Location_Easting_OSGR',
'Location_Northing_OSGR', 'Longitude', 'Latitude',
                                    'Police_Force',
'Local_Authority_(District)', 'Local_Authority_(Highway)',
                                    '1st_Road_Number', '2nd_Road_Number',
'Accident_Index', 'Date', 'Time',
                                    'LSOA_of_Accident_Location', 'Year',
'Journey_Purpose_of_Driver', 'Sex_of_Driver',
                                    'Skidding_and_Overturning',
'Special_Conditions_at_Site', 'Carriageway_Hazards',
                                    'Hit_Object_in_Carriageway',
'Hit_Object_off_Carriageway', 'Vehicle_Leaving_Carriageway',
                                    'Hit_Object_off_Carriageway',
'Driver_IMD_Decile', 'Age_of_Casualty', 'Casualty_Type',
                                    'Vehicle_Reference_x',
'Vehicle_Reference_y', 'Pedestrian_Crossing-Human_Control',
                                    'Pedestrian_Crossing-
Physical_Facilities', 'Pedestrian_Road_Maintenance_Worker',
'Casualty_Home_Area_Type',
                                    'Age_Band_of_Driver',
'Age_Band_of_Casualty', 'Pedestrian_Location', 'Pedestrian_Movement',
'Bus_or_Coach_Passenger',
                                    'Driver_Home_Area_Type'], axis=1,
inplace=True)

# New Shape of the dataframe
print("New Shape of the dataframe: ", accidents_vehicles_casualties.shape)

accidents_vehicles_casualties['Junction_Detail'] =
accidents_vehicles_casualties['Junction_Detail'].replace({
    'T or staggered junction': 'Staggered Junction',
    'Private drive or entrance': 'Entrance',
    'Other junction': 'Others',
    'More then 4 arms (not roundabout)': 'Fours',
    'Mini-roundabout': 'Miniroundabout',
    'Not at junction or within 20 metres': 'No Junction',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Junction_Control'] =
accidents_vehicles_casualties['Junction_Control'].replace({
    'Authorised person': 'Authorised',
    'Data missing or out of range': 'Unknown',
    'Give way or uncontrolled': 'Uncontrolled',
    'Stop sign': 'Stop',
    'Auto traffic signal': 'Traffic Signal',
})


accidents_vehicles_casualties['Light_Conditions'] =
accidents_vehicles_casualties['Light_Conditions'].replace({
    'Darkness - no lighting': 'Darkness',
    'Darkness - lighting unknown': 'Unknown',
    'Darkness - lights lit': 'Light',
    'Darkness - lights unlit': 'Unlit'
})
```

```
accidents_vehicles_casualties['Weather_Conditions'] =
accidents_vehicles_casualties['Weather_Conditions'].replace({
    'Fine without high winds': 'Fine',
    'Raining without high winds': 'Rain',
    'Raining + high winds': 'Rainwinds',
    'Fine + high winds': 'Fine',
    'Fog or mist': 'Fog',
    'Snowing without high winds': 'Snow',
    'Snowing + high winds': 'Snowind',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Road_Surface_Conditions'] =
accidents_vehicles_casualties['Road_Surface_Conditions'].replace({
    'Dry': 'Dry',
    'Wet or damp': 'Wet',
    'Frost or ice': 'Ice',
    'Snow': 'Snow',
    'Flood over 3cm. deep': 'Flood',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Did_Police_Officer_Attend_Scene_of_Accident']
=
accidents_vehicles_casualties['Did_Police_Officer_Attend_Scene_of_Accident'].
replace(
    'No - accident was reported using a self completion  form (self rep
only)', 'Self')

accidents_vehicles_casualties['Vehicle_Type'] =
accidents_vehicles_casualties['Vehicle_Type'].replace({
    'Bus or coach (17 or more pass seats)': 'Bus',
    'Van / Goods 3.5 tonnes mgw or under': 'Van',
    'Taxi/Private hire car': 'Taxi',
    'Motorcycle 125cc and under': 'Motorcycle',
    'Motorcycle over 500cc': 'Motorcycle',
    'Goods 7.5 tonnes mgw and over': 'Goods',
    'Motorcycle 50cc and under': 'Motorcycle',
    'Motorcycle over 125cc and up to 500cc': 'Motorcycle',
    'Goods over 3.5t. and under 7.5t': 'Goods',
    'Other vehicle': 'Others',
    'Minibus (8 - 16 passenger seats)': 'Minibus',
    'Agricultural vehicle (includes diggers etc.)': 'Agricultural',
    'Motorcycle - unknown cc': 'Motorcycle'
})

accidents_vehicles_casualties['Towing_and_Articulation'] =
accidents_vehicles_casualties['Towing_and_Articulation'].replace({
    'No tow/articulation': 'No',
    'Articulated vehicle': 'Articulated',
    'Single trailer': 'Single',
    'Other tow': 'Others',
    'Double or multiple trailer': 'Multiple',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Vehicle_Manoeuvre'] =
```

```
accidents_vehicles_casualties['Vehicle_Manoeuvre'].replace({
    'Going ahead other': 'Going ahead',
    'Turning right': 'Right',
    'Waiting to go - held up': 'Waiting',
    'Slowing or stopping': 'Slowing',
    'Turning left': 'Left',
    'Moving off': 'Moving',
    'Waiting to turn right': 'Waiting',
    'Going ahead right-hand bend': 'Going ahead',
    'Going ahead left-hand bend': 'Going ahead',
    'Overtaking moving vehicle - offside': 'Overtaking',
    'Waiting to turn left': 'Waiting',
    'Overtaking static vehicle - offside': 'Overtaking',
    'Changing lane to left': 'Changing lane',
    'Changing lane to right': 'Changing lane',
    'U-turn': 'Uturn',
    'Overtaking - nearside': 'Overtaking',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Vehicle_Location-Restricted_Lane'] =
accidents_vehicles_casualties['Vehicle_Location-Restricted_Lane'].replace({
    'On main c\way - not in restricted lane': 'Mainlane',
    'Bus lane': 'Buslane',
    'Footway (pavement)': 'Footway',
    'Leaving lay-by or hard shoulder': 'Leavinglayby',
    'On lay-by or hard shoulder': 'Onlayby',
    'Busway (including guided busway)': 'Busway',
    'Cycle lane (on main carriageway)': 'Cyclelane',
    'Tram/Light rail track': 'Tramtrack',
    'Entering lay-by or hard shoulder': 'Enteringlayby',
    'Cycleway or shared use footway (not part of  main carriageway)':
'Cycleway',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Junction_Location'] =
accidents_vehicles_casualties['Junction_Location'].replace({
    'Approaching junction or waiting/parked at junction approach':
'Approaching',
    'Mid Junction - on roundabout or on main road': 'Mid',
    'Cleared junction or waiting/parked at junction exit': 'Cleared',
    'Entering from slip road': 'Entering',
    'Leaving main road into minor road': 'Leaving',
    'Entering main road from minor road': 'Entering',
    'Leaving roundabout': 'Leaving',
    'Entering roundabout': 'Entering',
    'Data missing or out of range': 'Unknown',
    'Not at or within 20 metres of junction': 'No Junction',
})

accidents_vehicles_casualties['1st_Point_of_Impact'] =
accidents_vehicles_casualties['1st_Point_of_Impact'].replace({
    'Did not impact': 'Nothing',
    'Data missing or out of range': 'Unknown'
})
```

```
accidents_vehicles_casualties['Was_Vehicle_Left_Hand_Drive?'] =
accidents_vehicles_casualties['Was_Vehicle_Left_Hand_Drive?'].replace({
    'Data missing or out of range': 'Unknown'
})

accidents_vehicles_casualties['Propulsion_Code'] =
accidents_vehicles_casualties['Propulsion_Code'].replace({
    'Gas/Bi-fuel': 'Gas',
    'Petrol/Gas (LPG)': 'LPG',
})

accidents_vehicles_casualties['Casualty_Class'] =
accidents_vehicles_casualties['Casualty_Class'].replace({
    'Driver or rider': 'Driver'
})

accidents_vehicles_casualties['Sex_of_Casualty'] =
accidents_vehicles_casualties['Sex_of_Casualty'].replace({
    'Data missing or out of range': 'Unknown'
})

accidents_vehicles_casualties['Car_Passenger'] =
accidents_vehicles_casualties['Car_Passenger'].replace({
    'Data missing or out of range': 'Unknown'
})


# Define categorical and continuous columns
cat_cols =
list(accidents_vehicles_casualties.select_dtypes(include=['object']).columns)
cat_cols.remove('Accident_Severity')

cont_cols =
list(accidents_vehicles_casualties.select_dtypes(include=[np.number]).columns
)

# Anomaly Detection
# Check for outliers
plt.figure(figsize=(20, 10))
sns.boxplot(data=accidents_vehicles_casualties, orient='h')
plt.show()

# Remove outliers
def iqr(df):
    q1 = df.quantile(0.25)
    q3 = df.quantile(0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr
    return lower, upper

lower, upper = iqr(accidents_vehicles_casualties['Engine_Capacity_(CC)'])
accidents_vehicles_casualties =
accidents_vehicles_casualties[(accidents_vehicles_casualties['Engine_Capacity
_(CC)'] > lower) & (accidents_vehicles_casualties['Engine_Capacity_(CC)'] <
upper)]
```

```
# One-hot the categorical columns
accidents_vehicles_casualties = pd.get_dummies(accidents_vehicles_casualties,
columns=cat_cols, drop_first=True)

# Scale the continuous columns
def scale(x):
    return (x - x.min()) / (x.max() - x.min())

accidents_vehicles_casualties[cont_cols] =
accidents_vehicles_casualties[cont_cols].apply(scale, axis=0)


accidents_vehicles_casualties = accidents_vehicles_casualties.iloc[: : 2]

accidents_vehicles_casualties['Accident_Severity'] =
accidents_vehicles_casualties['Accident_Severity'].replace({
    'Fatal': 3,
    'Serious': 2,
    'Slight': 1
})

#Create a new column for the for the target varaible: fuel specific accident
impact score
accidents_vehicles_casualties['Fuel_Specific_Accident_Impact_Score'] =
accidents_vehicles_casualties['Accident_Severity'] *
accidents_vehicles_casualties['Engine_Capacity_(CC)']
accidents_vehicles_casualties['Fuel_Specific_Accident_Impact_Score'] =
accidents_vehicles_casualties['Fuel_Specific_Accident_Impact_Score'].astype(f
loat)

accidents_vehicles_casualties.drop(['Accident_Severity',
'Engine_Capacity_(CC)'], axis=1, inplace=True)

print("Shape of the dataframe after feature engineering: ",
accidents_vehicles_casualties.shape)

# Split the data into train and test
X =
accidents_vehicles_casualties.drop(['Fuel_Specific_Accident_Impact_Score'],
axis=1)
y = accidents_vehicles_casualties['Fuel_Specific_Accident_Impact_Score']

X = sm.add_constant(X)

X_train, X_test, y_train, y_test = tts(X, y, test_size=0.2,
random_state=5805)


#Display the AIC, BIC and Adjusted R2 as a predictive accuracy for each
elimination inside the table
def backward_stepwise_with_summary_pval(X_train, y_train):
    table = pd.DataFrame(columns=["AIC", "BIC", "Adj. R2", "P-Value"])
    p_val_tab = pd.DataFrame(columns=["P-Value"])
    f_stat_tab = pd.DataFrame(columns=["F-Statistic"])
    adj_r2_tab = pd.DataFrame(columns=["Adj. R2"])
    removed_features = []
    counter = 0
```

```
    model = sm.OLS(y_train, X_train).fit()
    print(model.summary())

    while model.pvalues.max() >= 0.01:
        table.loc[model.pvalues.idxmax(), "AIC"] = model.aic
        table.loc[model.pvalues.idxmax(), "BIC"] = model.bic
        table.loc[model.pvalues.idxmax(), "Adj. R2"] = model.rsquared_adj
        table.loc[model.pvalues.idxmax(), "P-Value"] = model.pvalues.max()
        p_val_tab.loc[model.pvalues.idxmax(), "P-Value"] =
model.pvalues.max()
        f_stat_tab.loc[model.pvalues.idxmax(), "F-Statistic"] = model.fvalue
        adj_r2_tab.loc[model.pvalues.idxmax(), "Adj. R2"] =
model.rsquared_adj
        X_train.drop(model.pvalues.idxmax(), axis=1, inplace=True)
        X_test.drop(model.pvalues.idxmax(), axis=1, inplace=True)
        print("Removed Feature:", model.pvalues.idxmax())
        removed_features.append(model.pvalues.idxmax())
        counter += 1
        print("No. of Features Removed:", counter)
        model = sm.OLS(y_train, X_train).fit()
        print(model.summary())

    print("Removed Features:", removed_features)
    return table, p_val_tab, f_stat_tab, adj_r2_tab


table, t_stat, f_stat, adj_r2  = backward_stepwise_with_summary_pval(X_train,
y_train)
print(table)
print(t_stat)
print(f_stat)
print(adj_r2)

# Fit the model with the selected features
lin_reg_final = sm.OLS(y_train, X_train).fit()
print(lin_reg_final.summary())

confidence_intervals = lin_reg_final.conf_int()
print("Confidence Intervals:\n", confidence_intervals)


def plot_backward_stepwise_summary(table):
    fig, ax = plt.subplots(1, 3, figsize=(30, 15))

    # Define the tick locations
    tick_locations = range(0, len(table.index), 15)

    # Plot AIC
    ax[0].plot(table["AIC"], marker="o")
    ax[0].set_title("AIC")
    ax[0].set_xlabel("Step")
    ax[0].set_xticks(tick_locations)
    ax[0].set_xticklabels([table.index[i] for i in tick_locations],
rotation=45)

    # Plot BIC
    ax[1].plot(table["BIC"], marker="o")
```

```
    ax[1].set_title("BIC")
    ax[1].set_xlabel("Step")
    ax[1].set_xticks(tick_locations)
    ax[1].set_xticklabels([table.index[i] for i in tick_locations],
rotation=45)

    # Plot Adjusted R2
    ax[2].plot(table["Adj. R2"], marker="o")
    ax[2].set_title("Adjusted R2")
    ax[2].set_xlabel("Step")
    ax[2].set_xticks(tick_locations)
    ax[2].set_xticklabels([table.index[i] for i in tick_locations],
rotation=45)

    for axis in ax:
        axis.tick_params(axis='x', which='major', labelsize=8)
        axis.grid(True)

    plt.tight_layout()
    plt.show()


plot_backward_stepwise_summary(table)

# Prediciton of dependent variable
y_pred = lin_reg_final.predict(X_test)

# Plot the predicted values against the actual values (diffent colors) with
line of best fit
plt.figure(figsize=(10, 10))
plt.scatter(y_test, y_pred, color='blue')
plt.plot(y_test, y_test, color='red')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.show()

# Do predictions on the test set and evaluate the model
print(f"MSE:{mse(y_test, y_pred):.2f}")
print(f"MAE:{mae(y_test, y_pred):.2f}")
print(f"RMSE:{np.sqrt(mse(y_test, y_pred)): .2f}")
print(f"R2:{lin_reg_final.rsquared:.2f}")
print(f"Adjusted R2:{lin_reg_final.rsquared_adj:.2f}")
```

Phase 3: Classification Analysis

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
from itertools import cycle
from imblearn.over_sampling import SMOTE
import statsmodels.api as sm
from sklearn.model_selection import train_test_split as tts
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import label_binarize
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier,
AdaBoostClassifier, StackingClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.multiclass import OneVsOneClassifier, OneVsRestClassifier
from sklearn.metrics import confusion_matrix, roc_auc_score, accuracy_score,
recall_score, roc_curve, precision_score, f1_score, auc
from sklearn.model_selection import StratifiedKFold
from scipy import mean
from prettytable import PrettyTable
import warnings
from sklearn.exceptions import ConvergenceWarning

pd.set_option('display.float_format', lambda x: '%.2f' % x)
sns.set_style('darkgrid')
warnings.filterwarnings("ignore")
warnings.simplefilter('ignore', category=FutureWarning)
warnings.simplefilter("ignore", category=ConvergenceWarning)

# Read in the data in chunks in for loop and then convert it to a dataframe
and see the time it takes to read the data
start_time = time.time()
accidents_vehicles_casualties = pd.DataFrame()

for chunk in pd.read_csv('UK_Accidents_Merger.csv', chunksize=200000,
low_memory=False):
    print('Number of chunks read: ', chunk.shape)
    accidents_vehicles_casualties = pd.concat([accidents_vehicles_casualties,
chunk])

print("Time taken to read the data: ", time.time() - start_time)
```

```
print("Shape of the dataframe: ", accidents_vehicles_casualties.shape)

# Feature Engineering
# Check for missing values
print("Missing Values:")
print(accidents_vehicles_casualties.isnull().sum()/accidents_vehicles_casualt
ies.shape[0])

# Drop or impute missing values
accidents_vehicles_casualties.dropna(inplace=True)

# Check for duplicates
print("Duplicates:")
print(accidents_vehicles_casualties.duplicated().sum())

accidents_vehicles_casualties['Date'] =
pd.to_datetime(accidents_vehicles_casualties['Date'])
accidents_vehicles_casualties['Year'] =
accidents_vehicles_casualties['Date'].dt.year

# Drop unnecessary columns
accidents_vehicles_casualties.drop(['Location_Easting_OSGR',
'Location_Northing_OSGR', 'Longitude', 'Latitude',
                                    'Police_Force',
'Local_Authority_(District)', 'Local_Authority_(Highway)',
                                    '1st_Road_Number', '2nd_Road_Number',
'Accident_Index', 'Date', 'Time',
                                    'LSOA_of_Accident_Location', 'Year',
'Journey_Purpose_of_Driver', 'Sex_of_Driver',
                                    'Skidding_and_Overturning',
'Special_Conditions_at_Site', 'Carriageway_Hazards',
                                    'Hit_Object_in_Carriageway',
'Hit_Object_off_Carriageway', 'Vehicle_Leaving_Carriageway',
                                    'Hit_Object_off_Carriageway',
'Driver_IMD_Decile', 'Age_of_Casualty', 'Casualty_Type',
                                    'Vehicle_Reference_x',
'Vehicle_Reference_y', 'Pedestrian_Crossing-Human_Control',
                                    'Pedestrian_Crossing-
Physical_Facilities', 'Pedestrian_Road_Maintenance_Worker',
'Casualty_Home_Area_Type',
                                    'Age_Band_of_Driver',
'Age_Band_of_Casualty', 'Pedestrian_Location', 'Pedestrian_Movement',
'Bus_or_Coach_Passenger',
                                    'Driver_Home_Area_Type'], axis=1,
inplace=True)

# New Shape of the dataframe
print("New Shape of the dataframe: ", accidents_vehicles_casualties.shape)

accidents_vehicles_casualties['Junction_Detail'] =
accidents_vehicles_casualties['Junction_Detail'].replace({
    'T or staggered junction': 'Staggered Junction',
    'Private drive or entrance': 'Entrance',
    'Other junction': 'Others',
    'More then 4 arms (not roundabout)': 'Fours',
    'Mini-roundabout': 'Miniroundabout',
    'Not at junction or within 20 metres': 'No Junction',
```

```
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Junction_Control'] =
accidents_vehicles_casualties['Junction_Control'].replace({
    'Authorised person': 'Authorised',
    'Data missing or out of range': 'Unknown',
    'Give way or uncontrolled': 'Uncontrolled',
    'Stop sign': 'Stop',
    'Auto traffic signal': 'Traffic Signal',
})


accidents_vehicles_casualties['Light_Conditions'] =
accidents_vehicles_casualties['Light_Conditions'].replace({
    'Darkness - no lighting': 'Darkness',
    'Darkness - lighting unknown': 'Unknown',
    'Darkness - lights lit': 'Light',
    'Darkness - lights unlit': 'Unlit'
})

accidents_vehicles_casualties['Weather_Conditions'] =
accidents_vehicles_casualties['Weather_Conditions'].replace({
    'Fine without high winds': 'Fine',
    'Raining without high winds': 'Rain',
    'Raining + high winds': 'Rainwinds',
    'Fine + high winds': 'Fine',
    'Fog or mist': 'Fog',
    'Snowing without high winds': 'Snow',
    'Snowing + high winds': 'Snowind',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Road_Surface_Conditions'] =
accidents_vehicles_casualties['Road_Surface_Conditions'].replace({
    'Dry': 'Dry',
    'Wet or damp': 'Wet',
    'Frost or ice': 'Ice',
    'Snow': 'Snow',
    'Flood over 3cm. deep': 'Flood',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Did_Police_Officer_Attend_Scene_of_Accident']
=
accidents_vehicles_casualties['Did_Police_Officer_Attend_Scene_of_Accident'].
replace(
    'No - accident was reported using a self completion  form (self rep
only)', 'Self')

accidents_vehicles_casualties['Vehicle_Type'] =
accidents_vehicles_casualties['Vehicle_Type'].replace({
    'Bus or coach (17 or more pass seats)': 'Bus',
    'Van / Goods 3.5 tonnes mgw or under': 'Van',
    'Taxi/Private hire car': 'Taxi',
    'Motorcycle 125cc and under': 'Motorcycle',
    'Motorcycle over 500cc': 'Motorcycle',
```

```
    'Goods 7.5 tonnes mgw and over': 'Goods',
    'Motorcycle 50cc and under': 'Motorcycle',
    'Motorcycle over 125cc and up to 500cc': 'Motorcycle',
    'Goods over 3.5t. and under 7.5t': 'Goods',
    'Other vehicle': 'Others',
    'Minibus (8 - 16 passenger seats)': 'Minibus',
    'Agricultural vehicle (includes diggers etc.)': 'Agricultural',
    'Motorcycle - unknown cc': 'Motorcycle'
})

accidents_vehicles_casualties['Towing_and_Articulation'] =
accidents_vehicles_casualties['Towing_and_Articulation'].replace({
    'No tow/articulation': 'No',
    'Articulated vehicle': 'Articulated',
    'Single trailer': 'Single',
    'Other tow': 'Others',
    'Double or multiple trailer': 'Multiple',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Vehicle_Manoeuvre'] =
accidents_vehicles_casualties['Vehicle_Manoeuvre'].replace({
    'Going ahead other': 'Going ahead',
    'Turning right': 'Right',
    'Waiting to go - held up': 'Waiting',
    'Slowing or stopping': 'Slowing',
    'Turning left': 'Left',
    'Moving off': 'Moving',
    'Waiting to turn right': 'Waiting',
    'Going ahead right-hand bend': 'Going ahead',
    'Going ahead left-hand bend': 'Going ahead',
    'Overtaking moving vehicle - offside': 'Overtaking',
    'Waiting to turn left': 'Waiting',
    'Overtaking static vehicle - offside': 'Overtaking',
    'Changing lane to left': 'Changing lane',
    'Changing lane to right': 'Changing lane',
    'U-turn': 'Uturn',
    'Overtaking - nearside': 'Overtaking',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Vehicle_Location-Restricted_Lane'] =
accidents_vehicles_casualties['Vehicle_Location-Restricted_Lane'].replace({
    'On main c\way - not in restricted lane': 'Mainlane',
    'Bus lane': 'Buslane',
    'Footway (pavement)': 'Footway',
    'Leaving lay-by or hard shoulder': 'Leavinglayby',
    'On lay-by or hard shoulder': 'Onlayby',
    'Busway (including guided busway)': 'Busway',
    'Cycle lane (on main carriageway)': 'Cyclelane',
    'Tram/Light rail track': 'Tramtrack',
    'Entering lay-by or hard shoulder': 'Enteringlayby',
    'Cycleway or shared use footway (not part of  main carriageway)':
'Cycleway',
    'Data missing or out of range': 'Unknown',
})
```

```
accidents_vehicles_casualties['Junction_Location'] =
accidents_vehicles_casualties['Junction_Location'].replace({
    'Approaching junction or waiting/parked at junction approach':
'Approaching',
    'Mid Junction - on roundabout or on main road': 'Mid',
    'Cleared junction or waiting/parked at junction exit': 'Cleared',
    'Entering from slip road': 'Entering',
    'Leaving main road into minor road': 'Leaving',
    'Entering main road from minor road': 'Entering',
    'Leaving roundabout': 'Leaving',
    'Entering roundabout': 'Entering',
    'Data missing or out of range': 'Unknown',
    'Not at or within 20 metres of junction': 'No Junction',
})

accidents_vehicles_casualties['1st_Point_of_Impact'] =
accidents_vehicles_casualties['1st_Point_of_Impact'].replace({
    'Did not impact': 'Nothing',
    'Data missing or out of range': 'Unknown'
})

accidents_vehicles_casualties['Was_Vehicle_Left_Hand_Drive?'] =
accidents_vehicles_casualties['Was_Vehicle_Left_Hand_Drive?'].replace({
    'Data missing or out of range': 'Unknown'
})

accidents_vehicles_casualties['Propulsion_Code'] =
accidents_vehicles_casualties['Propulsion_Code'].replace({
    'Gas/Bi-fuel': 'Gas',
    'Petrol/Gas (LPG)': 'LPG',
})

accidents_vehicles_casualties['Casualty_Class'] =
accidents_vehicles_casualties['Casualty_Class'].replace({
    'Driver or rider': 'Driver'
})

accidents_vehicles_casualties['Sex_of_Casualty'] =
accidents_vehicles_casualties['Sex_of_Casualty'].replace({
    'Data missing or out of range': 'Unknown'
})

accidents_vehicles_casualties['Car_Passenger'] =
accidents_vehicles_casualties['Car_Passenger'].replace({
    'Data missing or out of range': 'Unknown'
})

# Define categorical and continuous columns
cat_cols =
list(accidents_vehicles_casualties.select_dtypes(include=['object']).columns)
cat_cols.remove('Accident_Severity')

cont_cols =
list(accidents_vehicles_casualties.select_dtypes(include=[np.number]).columns
)

# Anomaly Detection
```

```
# Check for outliers
plt.figure(figsize=(20, 10))
sns.boxplot(data=accidents_vehicles_casualties, orient='h')
plt.show()

# Remove outliers
def iqr(df):
    q1 = df.quantile(0.25)
    q3 = df.quantile(0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr
    return lower, upper

lower, upper = iqr(accidents_vehicles_casualties['Engine_Capacity_(CC)'])
accidents_vehicles_casualties =
accidents_vehicles_casualties[(accidents_vehicles_casualties['Engine_Capacity
_(CC)'] > lower) & (accidents_vehicles_casualties['Engine_Capacity_(CC)'] <
upper)]

# One-hot the categorical columns
accidents_vehicles_casualties = pd.get_dummies(accidents_vehicles_casualties,
columns=cat_cols, drop_first=True)

# Scale the continuous columns
def scale(x):
    return (x - x.min()) / (x.max() - x.min())

accidents_vehicles_casualties[cont_cols] =
accidents_vehicles_casualties[cont_cols].apply(scale, axis=0)


accidents_vehicles_casualties = accidents_vehicles_casualties.iloc[: : 4]

# Divide the data into features and target
X, y = accidents_vehicles_casualties.drop('Accident_Severity', axis=1),
accidents_vehicles_casualties['Accident_Severity']

#Oversample the minority class
oversample = SMOTE()
X_smote, y_smote = oversample.fit_resample(X, y)

# Check the balance of the classes
print('Processisng SMOTE...')
fig, ax = plt.subplots(1, 2, figsize = (10, 5))
y.value_counts().plot.pie(explode = [0, 0.1, 0.2], autopct = "%1.1f%%", ax =
ax[0], colors = ['#ff9999', '#66b3ff', '#964B00'],
                          shadow = True)
ax[0].set_title("Genres Before SMOTE")
ax[0].set_ylabel('')
y_smote.value_counts().plot.pie(explode = [0, 0.1, 0.2], autopct = "%1.1f%%",
ax = ax[1],  colors = ['#66b3ff', '#ff9999', '#964B00'],
                                shadow = True)
ax[1].set_title("Genres After SMOTE")
ax[1].set_ylabel('')

print("Overall dataset values have been increased.")
```

```
print("Original size:\n", y.value_counts())
print("New size:\n", y_smote.value_counts())
plt.show()

X_smote = sm.add_constant(X_smote)

print('New size of dataset:', X_smote.shape, y_smote.shape)

# Split the data into train and test
X_train, X_test, y_train, y_test = tts(X_smote, y_smote, test_size=0.2,
random_state=5805)

print('Train size:', X_train.shape, y_train.shape)
print('Test size:', X_test.shape, y_test.shape)

# Encode target variable
le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)

# Feature Selection
rfc = RandomForestClassifier(n_estimators=100, random_state=5805, n_jobs=-1)
rfc.fit(X_train, y_train)

def plot_feature_importance_with_elimination(importance, names, model_type,
threshold=0.01):
    feature_importance = np.array(importance)
    feature_names = np.array(names)
    data = {"feature_names": feature_names, "feature_importance":
feature_importance}
    fi_df = pd.DataFrame(data)

    fi_df.sort_values(by=["feature_importance"], ascending=False,
inplace=True)

    plt.figure(figsize=(10, 8))
    sns.barplot(x=fi_df["feature_importance"], y=fi_df["feature_names"])
    plt.title(f"{model_type} Feature Importance")
    plt.xlabel("Feature Importance")
    plt.ylabel("Feature Names")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    removed_features = fi_df[fi_df["feature_importance"] <
threshold]["feature_names"]
    print("Removed Features:", removed_features.tolist())

    selected_features = fi_df[fi_df["feature_importance"] >=
threshold]["feature_names"]
    print("Selected Features:", selected_features.tolist())

    return removed_features, selected_features


removed, selected = plot_feature_importance_with_elimination(
    rfc.feature_importances_, X_train.columns, "RANDOM FOREST")
```

```
X_train, X_test = X_train[selected], X_test[selected]

X_train_knn = np.ascontiguousarray(X_train)
X_test_knn = np.ascontiguousarray(X_test)

print('Train size:', X_train.shape, y_train.shape)
print('Test size:', X_test.shape, y_test.shape)

# Model Building
def roc_auc_score_multiclass(actual_class, pred_class, average="macro"):
    unique_class = set(actual_class)
    roc_auc_dict = {}
    for per_class in unique_class:
        other_class = [x for x in unique_class if x != per_class]

        new_actual_class = [0 if x in other_class else 1 for x in
actual_class]
        new_pred_class = [0 if x in other_class else 1 for x in pred_class]

        roc_auc = roc_auc_score(new_actual_class, new_pred_class,
average=average)
        roc_auc_dict[per_class] = roc_auc

    return roc_auc_dict
def plot_multiclass_roc(y_test, y_pred, classes):
    y_test_binarized = label_binarize(y_test, classes=classes)
    y_pred_binarized = label_binarize(y_pred, classes=classes)

    n_classes = len(classes)
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i],
y_pred_binarized[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure(figsize=(10, 10))

    colors = cycle(['blue', 'red', 'green', 'orange', 'purple', 'cyan',
'magenta'])
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=2,
                 label='ROC curve of class {0} (area =
{1:0.2f})'.format(classes[i], roc_auc[i]))

    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Extension of Receiver Operating Characteristic to Multi-
class')
    plt.legend(loc="lower right")
    plt.show()
```

```
def specificity_calc(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    specificity_scores = {}

    for class_index in range(len(cm)):
        tn = np.sum(cm) - np.sum(cm[class_index, :]) - np.sum(cm[:,
class_index]) + cm[class_index, class_index]
        fp = np.sum(cm[:, class_index]) - cm[class_index, class_index]

        specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
        specificity_scores[class_index] = specificity

    for class_index in specificity_scores:
        print(f'Specificity of class {class_index}:
{specificity_scores[class_index]:.2f}')

# Create a function tahat will create an ovo model and plot multiclass roc
auc curve
def train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test,
model):
    ovo = OneVsOneClassifier(model)
    ovo.fit(X_train, y_train)
    y_pred= ovo.predict(X_test)

    classes = list(set(y_train))
    y_test_binarized = label_binarize(y_test, classes=classes)
    y_pred_binarized = label_binarize(y_pred, classes=classes)

    n_classes = len(classes)
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i],
y_pred_binarized[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Plotting
    plt.figure(figsize=(10, 10))
    colors = cycle(['blue', 'red', 'green', 'orange', 'purple', 'cyan',
'magenta'])
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=2,
                 label='ROC curve of class {0} (area =
{1:0.2f})'.format(classes[i], roc_auc[i]))

    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('One-vs-One ROC AUC for Multi-class')
    plt.legend(loc="lower right")
    plt.show()
```

```
# Create a function tahat will create an ovr model and plot multiclass roc
auc curve
def train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test,
model):
    ovr = OneVsRestClassifier(model)
    ovr.fit(X_train, y_train)

    y_pred = ovr.predict(X_test)

    classes = list(set(y_train))
    y_test_binarized = label_binarize(y_test, classes=classes)
    y_pred_binarized = label_binarize(y_pred, classes=classes)

    # Compute ROC curve and ROC area for each class
    n_classes = len(classes)
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i],
y_pred_binarized[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Plotting
    plt.figure(figsize=(10, 10))
    colors = cycle(['blue', 'red', 'green', 'orange', 'purple', 'cyan',
'magenta'])
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=2,
                 label='ROC curve of class {0} (area =
{1:0.2f})'.format(classes[i], roc_auc[i]))

    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('One-vs-Rest ROC AUC for Multi-class')
    plt.legend(loc="lower right")
    plt.show()

def train_plot_multiclass_knn_roc_auc_ovo(X_train, X_test, y_train, y_test,
model):
    X_train = np.ascontiguousarray(X_train)
    X_test = np.ascontiguousarray(X_test)

    ovo = OneVsOneClassifier(model)
    ovo.fit(X_train, y_train)

    y_pred = ovo.predict(X_test)

    classes = list(set(y_train))
    y_test_binarized = label_binarize(y_test, classes=classes)
    y_pred_binarized = label_binarize(y_pred, classes=classes)

    # Compute ROC curve and ROC area for each class
```

```
    n_classes = len(classes)
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i],
y_pred_binarized[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Plotting
    plt.figure(figsize=(10, 10))
    colors = cycle(['blue', 'red', 'green', 'orange', 'purple', 'cyan',
'magenta'])
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=2,
                 label='ROC curve of class {0} (area =
{1:0.2f})'.format(classes[i], roc_auc[i]))

    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('One-vs-One ROC AUC for Multi-class')
    plt.legend(loc="lower right")
    plt.show()

def train_plot_multiclass_knn_roc_auc_ovr(X_train, X_test, y_train, y_test,
model):
    X_train = np.ascontiguousarray(X_train)
    X_test = np.ascontiguousarray(X_test)

    ovr = OneVsRestClassifier(model)
    ovr.fit(X_train, y_train)

    y_pred = ovr.predict(X_test)

    classes = list(set(y_train))
    y_test_binarized = label_binarize(y_test, classes=classes)
    y_pred_binarized = label_binarize(y_pred, classes=classes)

    n_classes = len(classes)
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i],
y_pred_binarized[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure(figsize=(10, 10))
    colors = cycle(['blue', 'red', 'green', 'orange', 'purple', 'cyan',
'magenta'])
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=2,
```

```
                label='ROC curve of class {0} (area =
{1:0.2f})'.format(classes[i], roc_auc[i]))

    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('One-vs-Rest ROC AUC for Multi-class')
    plt.legend(loc="lower right")
    plt.show()

def kfold_cross_validation(model, X, y, k=5):
    kf = StratifiedKFold(n_splits=k, shuffle=True, random_state=5805)

    scores = []
    fold = 0
    for train_index, test_index in kf.split(X, y):
        fold += 1
        X_train_kf, X_test_kf = X.iloc[train_index], X.iloc[test_index]
        y_train_kf, y_test_kf = y.iloc[train_index], y.iloc[test_index]

        model.fit(X_train_kf, y_train_kf)
        y_pred_kf = model.predict(X_test_kf)

        accuracy_kf = accuracy_score(y_test_kf, y_pred_kf)
        scores.append(accuracy_kf)
        print(f'Fold {fold} accuracy: {accuracy_kf}')

    avg_scores = mean(scores)

    return avg_scores

def kfold_cross_validation_knn(model, X, y, k=5):
    kf = StratifiedKFold(n_splits=k, shuffle=True, random_state=5805)

    scores = []
    fold = 0
    for train_index, test_index in kf.split(X, y):
        fold += 1
        X_train_kf, X_test_kf = X.iloc[train_index], X.iloc[test_index]
        y_train_kf, y_test_kf = y.iloc[train_index], y.iloc[test_index]

        X_train_kf = np.ascontiguousarray(X_train_kf)
        X_test_kf = np.ascontiguousarray(X_test_kf)

        model.fit(X_train_kf, y_train_kf)
        y_pred_kf = model.predict(X_test_kf)

        accuracy_kf = accuracy_score(y_test_kf, y_pred_kf)
        scores.append(accuracy_kf)
        print(f'Fold {fold} accuracy: {accuracy_kf}')

    avg_scores = mean(scores)

    return avg_scores
```

```
def create_confusion_matrix_heatmap(y_test, y_pred, model_name):
    cm = confusion_matrix(y_test, y_pred)

    plt.figure(figsize=(10, 10))
    sns.heatmap(cm, cmap='Blues', annot=True, fmt='d', annot_kws={'size':
16})
    plt.title(f'Confusion Matrix of {model_name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

def find_best_k(X_train, y_train, X_test, y_test, k_range):
    error_rates = []
    accuracies = []

    for k in range(1, k_range + 1):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        y_pred = knn.predict(X_test)
        error = 1 - accuracy_score(y_test, y_pred)
        accuracy = accuracy_score(y_test, y_pred)

        error_rates.append(error)
        accuracies.append(accuracy)

    # Plotting
    plt.figure(figsize=(10, 6))
    plt.plot(range(1, k_range + 1), error_rates, color='blue',
linestyle='dashed', marker='o',
             markerfacecolor='red', markersize=10)
    plt.title('Error Rate vs. K Value')
    plt.xlabel('K')
    plt.ylabel('Error Rate')
    plt.show()

    best_k = accuracies.index(max(accuracies)) + 1
    return best_k, error_rates, accuracies


model_table = pd.DataFrame(columns=['Model', 'Accuracy', 'Precision',
'Recall', 'F1 Score', 'Cross Validation Mean Score', 'AUC Score'])

# 1a. Baseline Decision Tree
dtc = DecisionTreeClassifier(random_state=5805)
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)

y_pred_proba = dtc.predict_proba(X_test)

print('Performance of the baseline decision tree on the test set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Precision Score:', precision_score(y_test, y_pred,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred,
average='weighted').round(2))
```

```
specificity_calc(y_test, y_pred)


print('F1 Score:', f1_score(y_test, y_pred, average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy
scores = kfold_cross_validation(dtc, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred, average='macro')
micro_precision = precision_score(y_test, y_pred, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred, average='macro')
micro_recall = recall_score(y_test, y_pred, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_pred, average='macro')
micro_f1 = f1_score(y_test, y_pred, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred)

plot_multiclass_roc(y_test, y_pred, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, dtc)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, dtc)

create_confusion_matrix_heatmap(y_test, y_pred, 'Baseline Decision Tree')


model_table = model_table.append({'Model': 'Baseline Decision Tree',
                                  'Accuracy': accuracy_score(y_test,
y_pred).round(2),
                                  'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                  'Cross Validation Mean Score':
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)


# Tree Visualization
plt.figure(figsize=(10, 10))
plot_tree(dtc, filled=True, fontsize=10, rounded=True,
feature_names=list(X_smote.columns), class_names=['0','1', '2'])
```

```
plt.title('Decision Tree with base parameters')
plt.show()

# 1b. Fine Tuned Decision Tree
tuned_parameters = [{'max_depth': range(1, 10),
                     'min_samples_split': range(2, 10),
                     'min_samples_leaf':range(1, 10),
                     'max_features':range(1, 5),
                     'splitter':['best','random'],
                     'criterion':['gini','entropy', 'log_loss']}]

model_dtc = DecisionTreeClassifier(random_state=5805)

clf = GridSearchCV(model_dtc, tuned_parameters, cv=5, scoring='accuracy',
n_jobs=-1, verbose=False)
clf.fit(X_train, y_train)

print("Best Estimator: \n", clf.best_estimator_)
print("Best Score: \n", clf.best_score_)
print("Best Paramters: \n", clf.best_params_)

best_dtc = clf.best_estimator_
best_dtc.fit(X_train, y_train)

y_pred_grid = best_dtc.predict(X_test)

print('Performance of the fine tuned decision tree on the test set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred_grid).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred_grid))
print('Precision Score:', precision_score(y_test, y_pred_grid,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred_grid,
average='weighted').round(2))

specificity_calc(y_test, y_pred_grid)

print('F1 Score:', f1_score(y_test, y_pred_grid,
average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation(best_dtc, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred_grid, average='macro')
micro_precision = precision_score(y_test, y_pred_grid, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred_grid, average='macro')
micro_recall = recall_score(y_test, y_pred_grid, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_pred_grid, average='macro')
micro_f1 = f1_score(y_test, y_pred_grid, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred_grid)
```

```
plot_multiclass_roc(y_test, y_pred_grid, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, best_dtc)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, best_dtc)

create_confusion_matrix_heatmap(y_test, y_pred_grid, 'Fine Tuned Decision
Tree')

model_table = model_table.append({'Model': 'Fine Tuned Decision Tree',
                                  'Accuracy': accuracy_score(y_test,
y_pred_grid).round(2),
                                  'Precision': precision_score(y_test,
y_pred_grid, average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred_grid,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred_grid,
average='weighted').round(2),
                                  'Cross Validation Mean Score':
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)

plt.figure(figsize=(10, 10))
plot_tree(best_dtc, filled=True, fontsize=10, rounded=True,
feature_names=list(X_smote.columns), class_names=['0','1', '2'])
plt.title('Decision Tree with best parameters')
plt.show()

# 1c. Optimal Alpha and Pruned Decision Tree
path = best_dtc.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
print(ccp_alphas)
print(impurities)

accuracy_train, accuracy_test = [], []

for i in ccp_alphas:
    model = DecisionTreeClassifier(random_state=5805, ccp_alpha=i)
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
    accuracy_train.append(accuracy_score(y_train, y_train_pred))
    accuracy_test.append(accuracy_score(y_test, y_test_pred))

plt.figure(figsize=(5, 5))
plt.plot(ccp_alphas, accuracy_train, marker='o', label='train',
drawstyle='steps-post')
plt.plot(ccp_alphas, accuracy_test, marker='o', label='test',
drawstyle='steps-post')
plt.xlabel('ccp_alpha')
```

```
plt.ylabel('Accuracy')
plt.title('Accuracy vs ccp_alpha for train and test sets')
plt.legend()
plt.grid(True)
plt.show()

print('Optimal ccp_alpha:',
ccp_alphas[accuracy_test.index(max(accuracy_test))])

model = DecisionTreeClassifier(random_state=5805,
ccp_alpha=ccp_alphas[accuracy_test.index(max(accuracy_test))])
model.fit(X_train, y_train)

y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

print('Performance of the optimal ccp alpha decision tree on the test set:')
print('Accuracy of the model:', accuracy_score(y_test, y_test_pred).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_test_pred))
print('Precision Score:', precision_score(y_test, y_test_pred,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_test_pred,
average='weighted').round(2))

specificity_calc(y_test, y_test_pred)

print('F1 Score:', f1_score(y_test, y_test_pred,
average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation(model, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_test_pred, average='macro')
micro_precision = precision_score(y_test, y_test_pred, average='micro')

# Recall
macro_recall = recall_score(y_test, y_test_pred, average='macro')
micro_recall = recall_score(y_test, y_test_pred, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_test_pred, average='macro')
micro_f1 = f1_score(y_test, y_test_pred, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_test_pred)

plot_multiclass_roc(y_test, y_test_pred, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, model)

# One-vs-Rest
```

```
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, model)

create_confusion_matrix_heatmap(y_test, y_test_pred, 'Optimal Alpha: Post-
Pruned Decision Tree')

model_table = model_table.append({'Model': 'Optimal Alpha: Post-Pruned
Decision Tree',
                                  'Accuracy': accuracy_score(y_test,
y_test_pred).round(2),
                                  'Precision': precision_score(y_test,
y_test_pred, average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_test_pred,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_test_pred,
average='weighted').round(2),
                                  'Cross Validation Mean Score':
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)

plt.figure(figsize=(10, 10))
plot_tree(model, filled=True, fontsize=10, rounded=True,
feature_names=list(X_smote.columns), class_names=['0','1', '2'])
plt.title(f'Decision Tree with ccp_alpha =
{ccp_alphas[accuracy_test.index(max(accuracy_test))]}')
plt.show()


# 2a. Baseline Logistic Regression
log_reg = LogisticRegression(random_state=5805)
log_reg.fit(X_train, y_train)

y_pred = log_reg.predict(X_test)

print('Performance of the baseline logistic regression on the test set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Precision Score:', precision_score(y_test, y_pred,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred,
average='weighted').round(2))

specificity_calc(y_test, y_pred)

print('F1 Score:', f1_score(y_test, y_pred, average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation(log_reg, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred, average='macro')
micro_precision = precision_score(y_test, y_pred, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred, average='macro')
micro_recall = recall_score(y_test, y_pred, average='micro')
```

```
# F1 Score
macro_f1 = f1_score(y_test, y_pred, average='macro')
micro_f1 = f1_score(y_test, y_pred, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred)

plot_multiclass_roc(y_test, y_pred, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, log_reg)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, log_reg)

create_confusion_matrix_heatmap(y_test, y_pred, 'Baseline Logistic
Regression')

model_table = model_table.append({'Model': 'Baseline Logistic Regression',
                                  'Accuracy': accuracy_score(y_test,
y_pred).round(2),
                                  'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                  'Cross Validation Mean Score':
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)

# 2b. Fine Tuned Logistic Regression
tuned_parameters = [{'penalty': ['l1', 'l2', 'None'],
                     'C': [0.1, 1, 10],
                     'solver': ['newton-cg', 'lbfgs'],
                     'max_iter': [2000, 3000]}]

model_log_reg = LogisticRegression(random_state=5805)

clf = GridSearchCV(model_log_reg, tuned_parameters, cv=5, scoring='accuracy',
n_jobs=-1, verbose=False)
clf.fit(X_train, y_train)

print("Best Estimator: \n", clf.best_estimator_)
print("Best Score: \n", clf.best_score_)
print("Best Paramters: \n", clf.best_params_)

best_log_reg = clf.best_estimator_
best_log_reg.fit(X_train, y_train)

y_pred_grid = best_log_reg.predict(X_test)
```

```
print('Performance of the fine tuned logistic regression on the test set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred_grid).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred_grid))
print('Precision Score:', precision_score(y_test, y_pred_grid,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred_grid,
average='weighted').round(2))

specificity_calc(y_test, y_pred_grid)

print('F1 Score:', f1_score(y_test, y_pred_grid,
average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation(best_log_reg, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred_grid, average='macro')
micro_precision = precision_score(y_test, y_pred_grid, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred_grid, average='macro')
micro_recall = recall_score(y_test, y_pred_grid, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_pred_grid, average='macro')
micro_f1 = f1_score(y_test, y_pred_grid, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred_grid)

plot_multiclass_roc(y_test, y_pred_grid, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test,
best_log_reg)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test,
best_log_reg)

create_confusion_matrix_heatmap(y_test, y_pred_grid, 'Fine Tuned Logistic
Regression')

model_table = model_table.append({'Model': 'Fine Tuned Logistic Regression',
                                  'Accuracy': accuracy_score(y_test,
y_pred_grid).round(2),
                                  'Precision': precision_score(y_test,
y_pred_grid, average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred_grid,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred_grid,
average='weighted').round(2),
```

```
                                    'Cross Validation Mean Score':
scores.round(2),
                                      'AUC Score': auc_mean.round(2)},
ignore_index=True)


# 3a. Baseline KNN
knn = KNeighborsClassifier()
knn.fit(X_train_knn, y_train)

y_pred = knn.predict(X_test_knn)

print('Performance of the baseline knn classifier on the test set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Precision Score:', precision_score(y_test, y_pred,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred,
average='weighted').round(2))

specificity_calc(y_test, y_pred)

print('F1 Score:', f1_score(y_test, y_pred, average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation_knn(knn, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred, average='macro')
micro_precision = precision_score(y_test, y_pred, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred, average='macro')
micro_recall = recall_score(y_test, y_pred, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_pred, average='macro')
micro_f1 = f1_score(y_test, y_pred, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred)

plot_multiclass_roc(y_test, y_pred, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_knn_roc_auc_ovo(X_train, X_test, y_train, y_test, knn)

# One-vs-Rest
train_plot_multiclass_knn_roc_auc_ovr(X_train, X_test, y_train, y_test, knn)

create_confusion_matrix_heatmap(y_test, y_pred, 'Baseline KNN')

model_table = model_table.append({'Model': 'Baseline KNN',
```

```
                                    'Accuracy': accuracy_score(y_test,
y_pred).round(2),
                                    'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                    'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                    'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                    'Cross Validation Mean Score':
scores.round(2),
                                      'AUC Score': auc_mean.round(2)},
ignore_index=True)

# 3b. Fine Tuned KNN
knn = KNeighborsClassifier()
k_range = list(range(1, 31))
param_grid = dict(n_neighbors=k_range)

clf = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy',
return_train_score=True, verbose=1, n_jobs=-1)
clf.fit(X_train_knn, y_train)

print("Best Estimator: \n", clf.best_estimator_)
print("Best Score: \n", clf.best_score_)
print("Best Paramters: \n", clf.best_params_)

best_knn = clf.best_estimator_
best_knn.fit(X_train_knn, y_train)

y_pred_grid = best_knn.predict(X_test_knn)

print('Performance of the fine tuned knn classifier on the test set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred_grid).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred_grid))
print('Precision Score:', precision_score(y_test, y_pred_grid,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred_grid,
average='weighted').round(2))

specificity_calc(y_test, y_pred_grid)

print('F1 Score:', f1_score(y_test, y_pred_grid,
average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation_knn(best_knn, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred_grid, average='macro')
micro_precision = precision_score(y_test, y_pred_grid, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred_grid, average='macro')
micro_recall = recall_score(y_test, y_pred_grid, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_pred_grid, average='macro')
```

```
micro_f1 = f1_score(y_test, y_pred_grid, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred_grid)

plot_multiclass_roc(y_test, y_pred_grid, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_knn_roc_auc_ovo(X_train, X_test, y_train, y_test,
best_knn)

# One-vs-Rest
train_plot_multiclass_knn_roc_auc_ovr(X_train, X_test, y_train, y_test,
best_knn)

create_confusion_matrix_heatmap(y_test, y_pred_grid, 'Fine Tuned KNN')

# Selection of best k
best_k, error_rates, accuracies = find_best_k(X_train_knn, y_train,
X_test_knn, y_test, 30)
print('Best k:', best_k)

model_table = model_table.append({'Model': 'Fine Tuned KNN',
                                  'Accuracy': accuracy_score(y_test,
y_pred_grid).round(2),
                                  'Precision': precision_score(y_test,
y_pred_grid, average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred_grid,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred_grid,
average='weighted').round(2),
                                  'Cross Validation Mean Score':
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)


# 4a. Naive Bayes
gnb = GaussianNB()
gnb.fit(X_train, y_train)

y_pred = gnb.predict(X_test)

print('Performance of the baseline gaussian naive bayes on the test set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Precision Score:', precision_score(y_test, y_pred,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred,
average='weighted').round(2))

specificity_calc(y_test, y_pred)
```

```
print('F1 Score:', f1_score(y_test, y_pred, average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation(gnb, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred, average='macro')
micro_precision = precision_score(y_test, y_pred, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred, average='macro')
micro_recall = recall_score(y_test, y_pred, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_pred, average='macro')
micro_f1 = f1_score(y_test, y_pred, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred)

plot_multiclass_roc(y_test, y_pred, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, gnb)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, gnb)

create_confusion_matrix_heatmap(y_test, y_pred, 'Baseline Naive Bayes')


model_table = model_table.append({'Model': 'Baseline Naive Bayes',
                                  'Accuracy': accuracy_score(y_test,
y_pred).round(2),
                                  'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                  'Cross Validation Mean Score':
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)

# 4b. Fine Tuned Naive Bayes
gnb = GaussianNB()

tuned_parameters = [{'var_smoothing': [1e-2, 1e-3]}]

clf = GridSearchCV(gnb, tuned_parameters, cv=5, scoring='accuracy',
return_train_score=True, verbose=1, n_jobs=-1)
clf.fit(X_train, y_train)
```

```
print("Best Estimator: \n", clf.best_estimator_)
print("Best Score: \n", clf.best_score_)
print("Best Paramters: \n", clf.best_params_)

best_gnb = clf.best_estimator_
best_gnb.fit(X_train, y_train)

y_pred_grid = best_gnb.predict(X_test)

print('Performance of the fine tuned gaussian naive bayes on the test set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred_grid).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred_grid))
print('Precision Score:', precision_score(y_test, y_pred_grid,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred_grid,
average='weighted').round(2))

specificity_calc(y_test, y_pred_grid)

print('F1 Score:', f1_score(y_test, y_pred_grid,
average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation(best_gnb, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred_grid, average='macro')
micro_precision = precision_score(y_test, y_pred_grid, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred_grid, average='macro')
micro_recall = recall_score(y_test, y_pred_grid, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_pred_grid, average='macro')
micro_f1 = f1_score(y_test, y_pred_grid, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred_grid)

plot_multiclass_roc(y_test, y_pred_grid, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, best_gnb)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, best_gnb)

create_confusion_matrix_heatmap(y_test, y_pred_grid, 'Fine Tuned Naive
Bayes')

model_table = model_table.append({'Model': 'Fine Tuned Naive Bayes',
```

```
                                    'Accuracy': accuracy_score(y_test,
y_pred_grid).round(2),
                                    'Precision': precision_score(y_test,
y_pred_grid, average='weighted').round(2),
                                    'Recall': recall_score(y_test, y_pred_grid,
average='weighted').round(2),
                                    'F1 Score': f1_score(y_test, y_pred_grid,
average='weighted').round(2),
                                    'Cross Validation Mean Score':
scores.round(2),
                                      'AUC Score': auc_mean.round(2)},
ignore_index=True)


# 5a. Baseline SVM
svc = SVC(random_state=5805)
svc.fit(X_train, y_train)

y_pred = svc.predict(X_test)

print('Performance of the baseline svm on the test set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Precision Score:', precision_score(y_test, y_pred,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred,
average='weighted').round(2))

specificity_calc(y_test, y_pred)

print('F1 Score:', f1_score(y_test, y_pred, average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation(svc, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred, average='macro')
micro_precision = precision_score(y_test, y_pred, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred, average='macro')
micro_recall = recall_score(y_test, y_pred, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_pred, average='macro')
micro_f1 = f1_score(y_test, y_pred, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred)

plot_multiclass_roc(y_test, y_pred, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
```

```
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, svc)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, svc)

create_confusion_matrix_heatmap(y_test, y_pred, 'Baseline SVM')

model_table = model_table.append({'Model': 'Baseline SVM',
                                  'Accuracy': accuracy_score(y_test,
y_pred).round(2),
                                  'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                  'Cross Validation Mean Score':
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)

# 5b. Fine Tuned SVM
svc = SVC(random_state=5805)

tuned_parameters = [{'kernel': ['rbf', 'poly'],
                     'gamma': [0.1, 1],
                     'C': [0.1, 1]}]

clf = GridSearchCV(svc, tuned_parameters, cv=5, scoring='accuracy', n_jobs=-
1, verbose=False)
clf.fit(X_train, y_train)

print("Best Estimator: \n", clf.best_estimator_)
print("Best Score: \n", clf.best_score_)
print("Best Paramters: \n", clf.best_params_)

best_svc = clf.best_estimator_
best_svc.fit(X_train, y_train)

y_pred_grid = best_svc.predict(X_test)

print('Performance of the fine tuned svm on the test set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred_grid).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred_grid))
print('Precision Score:', precision_score(y_test, y_pred_grid,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred_grid,
average='weighted').round(2))

specificity_calc(y_test, y_pred_grid)

print('F1 Score:', f1_score(y_test, y_pred_grid,
average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation(best_svc, X_smote, y_smote)
```

```
# Precision
macro_precision = precision_score(y_test, y_pred_grid, average='macro')
micro_precision = precision_score(y_test, y_pred_grid, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred_grid, average='macro')
micro_recall = recall_score(y_test, y_pred_grid, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_pred_grid, average='macro')
micro_f1 = f1_score(y_test, y_pred_grid, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred_grid)

plot_multiclass_roc(y_test, y_pred_grid, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, best_svc)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, best_svc)

create_confusion_matrix_heatmap(y_test, y_pred_grid, 'Fine Tuned SVM')

model_table = model_table.append({'Model': 'Fine Tuned SVM',
                                  'Accuracy': accuracy_score(y_test,
y_pred_grid).round(2),
                                  'Precision': precision_score(y_test,
y_pred_grid, average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred_grid,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred_grid,
average='weighted').round(2),
                                  'Cross Validation Mean Score':
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)


# 6a. Multi Layer Perceptron
mlp = MLPClassifier(random_state=5805)
mlp.fit(X_train, y_train)

y_pred = mlp.predict(X_test)

print('Performance of the baseline multi layer perceptron on the test set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Precision Score:', precision_score(y_test, y_pred,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred,
```

```
average='weighted').round(2))

specificity_calc(y_test, y_pred)

print('F1 Score:', f1_score(y_test, y_pred, average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation(mlp, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred, average='macro')
micro_precision = precision_score(y_test, y_pred, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred, average='macro')
micro_recall = recall_score(y_test, y_pred, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_pred, average='macro')
micro_f1 = f1_score(y_test, y_pred, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred)

plot_multiclass_roc(y_test, y_pred, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, mlp)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, mlp)

create_confusion_matrix_heatmap(y_test, y_pred, 'Baseline Multi Layer
Perceptron')


model_table = model_table.append({'Model': 'Baseline Multi Layer Perceptron',
                                  'Accuracy': accuracy_score(y_test,
y_pred).round(2),
                                  'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                  'Cross Validation Mean Score':
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)


# 6b. Fine Tuned Multi Layer Perceptron
mlp = MLPClassifier(random_state=5805)
```

```
tuned_parameters = [{'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50),
(100,)],
                     'alpha': [0.0001, 0.05],
                     'learning_rate': ['constant','adaptive']}]

clf = GridSearchCV(mlp, tuned_parameters, cv=5, scoring='accuracy', n_jobs=-
1, verbose=False)
clf.fit(X_train, y_train)

print("Best Estimator: \n", clf.best_estimator_)
print("Best Score: \n", clf.best_score_)
print("Best Paramters: \n", clf.best_params_)

best_mlp = clf.best_estimator_
best_mlp.fit(X_train, y_train)

y_pred_grid = best_mlp.predict(X_test)

print('Performance of the fine tuned multi layer perceptron on the test
set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred_grid).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred_grid))
print('Precision Score:', precision_score(y_test, y_pred_grid,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred_grid,
average='weighted').round(2))

specificity_calc(y_test, y_pred_grid)

print('F1 Score:', f1_score(y_test, y_pred_grid,
average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation(best_mlp, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred_grid, average='macro')
micro_precision = precision_score(y_test, y_pred_grid, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred_grid, average='macro')
micro_recall = recall_score(y_test, y_pred_grid, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_pred_grid, average='macro')
micro_f1 = f1_score(y_test, y_pred_grid, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred_grid)

plot_multiclass_roc(y_test, y_pred_grid, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')
```

```
# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, best_mlp)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, best_mlp)

create_confusion_matrix_heatmap(y_test, y_pred_grid, 'Fine Tuned Multi Layer
Perceptron')


model_table = model_table.append({'Model': 'Fine Tuned Multi Layer
Perceptron',
                                  'Accuracy': accuracy_score(y_test,
y_pred_grid).round(2),
                                  'Precision': precision_score(y_test,
y_pred_grid, average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred_grid,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred_grid,
average='weighted').round(2),
                                  'Cross Validation Mean Score':
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)


# 7a. Baseline Random Forest
rfc = RandomForestClassifier(random_state=5805)
rfc.fit(X_train, y_train)

y_pred = rfc.predict(X_test)

print('Performance of the baseline random forest classifier on the test
set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Precision Score:', precision_score(y_test, y_pred,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred,
average='weighted').round(2))

specificity_calc(y_test, y_pred)

print('F1 Score:', f1_score(y_test, y_pred, average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation(rfc, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred, average='macro')
micro_precision = precision_score(y_test, y_pred, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred, average='macro')
micro_recall = recall_score(y_test, y_pred, average='micro')

# F1 Score
```

```
macro_f1 = f1_score(y_test, y_pred, average='macro')
micro_f1 = f1_score(y_test, y_pred, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred)

plot_multiclass_roc(y_test, y_pred, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, rfc)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, rfc)

create_confusion_matrix_heatmap(y_test, y_pred, 'Baseline Random Forest')


model_table = model_table.append({'Model': 'Baseline Random Forest',
                                  'Accuracy': accuracy_score(y_test,
y_pred).round(2),
                                  'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                  'Cross Validation Mean Score':
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)

# 7b. Random Forest with Bagging
rfc = RandomForestClassifier(random_state=5805, n_jobs=-1)

bag = BaggingClassifier(rfc, n_estimators=100, random_state=5805, n_jobs=-1)
bag.fit(X_train, y_train)

y_pred = bag.predict(X_test)

print('Performance of the random forest classifier with bagging on the test
set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Precision Score:', precision_score(y_test, y_pred,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred,
average='weighted').round(2))

specificity_calc(y_test, y_pred)

print('F1 Score:', f1_score(y_test, y_pred, average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
```

```
scores = kfold_cross_validation(bag, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred, average='macro')
micro_precision = precision_score(y_test, y_pred, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred, average='macro')
micro_recall = recall_score(y_test, y_pred, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_pred, average='macro')
micro_f1 = f1_score(y_test, y_pred, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred)

plot_multiclass_roc(y_test, y_pred, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, bag)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, bag)

create_confusion_matrix_heatmap(y_test, y_pred, 'Random Forest with Bagging')

model_table = model_table.append({'Model': 'Random Forest with Bagging',
                                  'Accuracy': accuracy_score(y_test,
y_pred).round(2),
                                  'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                  'Cross Validation Mean Score':
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)


# 7c. Random Forest with Stacking
rfc = RandomForestClassifier(random_state=5805, n_jobs=-1)

final_estimator = RandomForestClassifier(random_state=5805, n_jobs=-1)

stack = StackingClassifier(estimators=[('rfc', rfc)],
final_estimator=final_estimator, cv=5)
stack.fit(X_train, y_train)

y_pred = stack.predict(X_test)
```

```
print('Performance of the random forest classifier with stacking on the test
set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Precision Score:', precision_score(y_test, y_pred,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred,
average='weighted').round(2))

specificity_calc(y_test, y_pred)

print('F1 Score:', f1_score(y_test, y_pred, average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation(stack, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred, average='macro')
micro_precision = precision_score(y_test, y_pred, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred, average='macro')
micro_recall = recall_score(y_test, y_pred, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_pred, average='macro')
micro_f1 = f1_score(y_test, y_pred, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred)

plot_multiclass_roc(y_test, y_pred, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, stack)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, stack)

create_confusion_matrix_heatmap(y_test, y_pred, 'Random Forest with
Stacking')


model_table = model_table.append({'Model': 'Random Forest with Stacking',
                                  'Accuracy': accuracy_score(y_test,
y_pred).round(2),
                                  'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                  'Cross Validation Mean Score':
```

```
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)

# 7d. Random Forest with Boosting
rfc = RandomForestClassifier(random_state=5805, n_jobs=-1)

boost = AdaBoostClassifier(rfc, n_estimators=100, random_state=5805)
boost.fit(X_train, y_train)

y_pred = boost.predict(X_test)

print('Performance of the random forest classifier with boosting on the test
set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Precision Score:', precision_score(y_test, y_pred,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred,
average='weighted').round(2))

specificity_calc(y_test, y_pred)

print('F1 Score:', f1_score(y_test, y_pred, average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation(boost, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred, average='macro')
micro_precision = precision_score(y_test, y_pred, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred, average='macro')
micro_recall = recall_score(y_test, y_pred, average='micro')

# F1 Score
macro_f1 = f1_score(y_test, y_pred, average='macro')
micro_f1 = f1_score(y_test, y_pred, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred)

plot_multiclass_roc(y_test, y_pred, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, boost)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, boost)

create_confusion_matrix_heatmap(y_test, y_pred, 'Random Forest with
Boosting')
```

```
model_table = model_table.append({'Model': 'Random Forest with Boosting',
                                  'Accuracy': accuracy_score(y_test,
y_pred).round(2),
                                  'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                  'Cross Validation Mean Score':
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)

# 7e. Random Forest Grid Search
rfc = RandomForestClassifier(random_state=5805, n_jobs=-1)

tuned_parameters = [{'n_estimators': [100, 200],
                     'min_samples_split': [2, 5],
                     'min_samples_leaf': [1, 2]}]

clf = GridSearchCV(rfc, tuned_parameters, cv=5, scoring='accuracy', n_jobs=-
1, verbose=False)
clf.fit(X_train, y_train)

print("Best Estimator: \n", clf.best_estimator_)

best_rfc = clf.best_estimator_

y_pred = best_rfc.predict(X_test)

print('Performance of the random forest classifier with grid search on the
test set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Precision Score:', precision_score(y_test, y_pred,
average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred,
average='weighted').round(2))

specificity_calc(y_test, y_pred)

print('F1 Score:', f1_score(y_test, y_pred, average='weighted').round(2))

# Apply k-Fold Cross Validation based on accuracy and find the average mean
scores = kfold_cross_validation(best_rfc, X_smote, y_smote)

# Precision
macro_precision = precision_score(y_test, y_pred, average='macro')
micro_precision = precision_score(y_test, y_pred, average='micro')

# Recall
macro_recall = recall_score(y_test, y_pred, average='macro')
micro_recall = recall_score(y_test, y_pred, average='micro')
```

```
# F1 Score
macro_f1 = f1_score(y_test, y_pred, average='macro')
micro_f1 = f1_score(y_test, y_pred, average='micro')

# ROC-AUC
roc_auc_dict = roc_auc_score_multiclass(y_test, y_pred)

plot_multiclass_roc(y_test, y_pred, np.unique(y_test))

auc_mean = mean(list(roc_auc_dict.values()))
print(f'The mean of AUC for this multi-label classification is '
      f'{100*mean(list(roc_auc_dict.values())):.2f}%')

# One-vs-One
train_plot_multiclass_roc_auc_ovo(X_train, X_test, y_train, y_test, best_rfc)

# One-vs-Rest
train_plot_multiclass_roc_auc_ovr(X_train, X_test, y_train, y_test, best_rfc)

create_confusion_matrix_heatmap(y_test, y_pred, 'Random Forest with Grid
Search')

model_table = model_table.append({'Model': 'Random Forest with Grid Search',
                                  'Accuracy': accuracy_score(y_test,
y_pred).round(2),
                                  'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                  'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                  'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                   'Cross Validation Mean Score':
scores.round(2),
                                    'AUC Score': auc_mean.round(2)},
ignore_index=True)


def plot_model_comparison_table(table, name):
    table = table.sort_values(by='Accuracy', ascending=False)
    table = table.round(2)
    table = table.reset_index(drop=True)

    x = PrettyTable()
    x.title = f"{name} Comparison"
    x.field_names = table.columns

    for index, row in table.iterrows():
        x.add_row(row)

    print(x)

plot_model_comparison_table(model_table, 'Model')
```

Phase 4: Clustering & Association

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
from imblearn.over_sampling import SMOTE
import statsmodels.api as sm
from sklearn.model_selection import train_test_split as tts
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import silhouette_score
from mlxtend.frequent_patterns import apriori, association_rules
import warnings


pd.set_option('display.float_format', lambda x: '%.2f' % x)
sns.set_style('darkgrid')
warnings.filterwarnings("ignore")

# Read in the data in chunks in for loop and then convert it to a dataframe
and see the time it takes to read the data
start_time = time.time()
accidents_vehicles_casualties = pd.DataFrame()

for chunk in pd.read_csv('UK_Accidents_Merger.csv', chunksize=200000,
low_memory=False):
    print('Number of chunks read: ', chunk.shape)
    accidents_vehicles_casualties = pd.concat([accidents_vehicles_casualties,
chunk])

print("Time taken to read the data: ", time.time() - start_time)

print("Shape of the dataframe: ", accidents_vehicles_casualties.shape)

# Feature Engineering
# Check for missing values
print("Missing Values:")
print(accidents_vehicles_casualties.isnull().sum()/accidents_vehicles_casualt
ies.shape[0])

# Drop or impute missing values
accidents_vehicles_casualties.dropna(inplace=True)

# Check for duplicates
print("Duplicates:")
print(accidents_vehicles_casualties.duplicated().sum())

accidents_vehicles_casualties['Date'] =
pd.to_datetime(accidents_vehicles_casualties['Date'])
accidents_vehicles_casualties['Year'] =
accidents_vehicles_casualties['Date'].dt.year
```

```
# Drop unnecessary columns
accidents_vehicles_casualties.drop(['Location_Easting_OSGR',
'Location_Northing_OSGR', 'Longitude', 'Latitude',
                                    'Police_Force',
'Local_Authority_(District)', 'Local_Authority_(Highway)',
                                    '1st_Road_Number', '2nd_Road_Number',
'Accident_Index', 'Date', 'Time',
                                    'LSOA_of_Accident_Location', 'Year',
'Journey_Purpose_of_Driver', 'Sex_of_Driver',
                                    'Skidding_and_Overturning',
'Special_Conditions_at_Site', 'Carriageway_Hazards',
                                    'Hit_Object_in_Carriageway',
'Hit_Object_off_Carriageway', 'Vehicle_Leaving_Carriageway',
                                    'Hit_Object_off_Carriageway',
'Driver_IMD_Decile', 'Age_of_Casualty', 'Casualty_Type',
                                    'Vehicle_Reference_x',
'Vehicle_Reference_y', 'Pedestrian_Crossing-Human_Control',
                                    'Pedestrian_Crossing-
Physical_Facilities', 'Pedestrian_Road_Maintenance_Worker',
'Casualty_Home_Area_Type',
                                    'Age_Band_of_Driver',
'Age_Band_of_Casualty', 'Pedestrian_Location', 'Pedestrian_Movement',
'Bus_or_Coach_Passenger',
                                    'Driver_Home_Area_Type'], axis=1,
inplace=True)

# New Shape of the dataframe
print("New Shape of the dataframe: ", accidents_vehicles_casualties.shape)

accidents_vehicles_casualties['Junction_Detail'] =
accidents_vehicles_casualties['Junction_Detail'].replace({
    'T or staggered junction': 'Staggered Junction',
    'Private drive or entrance': 'Entrance',
    'Other junction': 'Others',
    'More then 4 arms (not roundabout)': 'Fours',
    'Mini-roundabout': 'Miniroundabout',
    'Not at junction or within 20 metres': 'No Junction',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Junction_Control'] =
accidents_vehicles_casualties['Junction_Control'].replace({
    'Authorised person': 'Authorised',
    'Data missing or out of range': 'Unknown',
    'Give way or uncontrolled': 'Uncontrolled',
    'Stop sign': 'Stop',
    'Auto traffic signal': 'Traffic Signal',
})


accidents_vehicles_casualties['Light_Conditions'] =
accidents_vehicles_casualties['Light_Conditions'].replace({
    'Darkness - no lighting': 'Darkness',
    'Darkness - lighting unknown': 'Unknown',
    'Darkness - lights lit': 'Light',
    'Darkness - lights unlit': 'Unlit'
```

```
})

accidents_vehicles_casualties['Weather_Conditions'] =
accidents_vehicles_casualties['Weather_Conditions'].replace({
    'Fine without high winds': 'Fine',
    'Raining without high winds': 'Rain',
    'Raining + high winds': 'Rainwinds',
    'Fine + high winds': 'Fine',
    'Fog or mist': 'Fog',
    'Snowing without high winds': 'Snow',
    'Snowing + high winds': 'Snowind',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Road_Surface_Conditions'] =
accidents_vehicles_casualties['Road_Surface_Conditions'].replace({
    'Dry': 'Dry',
    'Wet or damp': 'Wet',
    'Frost or ice': 'Ice',
    'Snow': 'Snow',
    'Flood over 3cm. deep': 'Flood',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Did_Police_Officer_Attend_Scene_of_Accident']
=
accidents_vehicles_casualties['Did_Police_Officer_Attend_Scene_of_Accident'].
replace(
    'No - accident was reported using a self completion  form (self rep
only)', 'Self')

accidents_vehicles_casualties['Vehicle_Type'] =
accidents_vehicles_casualties['Vehicle_Type'].replace({
    'Bus or coach (17 or more pass seats)': 'Bus',
    'Van / Goods 3.5 tonnes mgw or under': 'Van',
    'Taxi/Private hire car': 'Taxi',
    'Motorcycle 125cc and under': 'Motorcycle',
    'Motorcycle over 500cc': 'Motorcycle',
    'Goods 7.5 tonnes mgw and over': 'Goods',
    'Motorcycle 50cc and under': 'Motorcycle',
    'Motorcycle over 125cc and up to 500cc': 'Motorcycle',
    'Goods over 3.5t. and under 7.5t': 'Goods',
    'Other vehicle': 'Others',
    'Minibus (8 - 16 passenger seats)': 'Minibus',
    'Agricultural vehicle (includes diggers etc.)': 'Agricultural',
    'Motorcycle - unknown cc': 'Motorcycle'
})

accidents_vehicles_casualties['Towing_and_Articulation'] =
accidents_vehicles_casualties['Towing_and_Articulation'].replace({
    'No tow/articulation': 'No',
    'Articulated vehicle': 'Articulated',
    'Single trailer': 'Single',
    'Other tow': 'Others',
    'Double or multiple trailer': 'Multiple',
    'Data missing or out of range': 'Unknown',
})
```

```
accidents_vehicles_casualties['Vehicle_Manoeuvre'] =
accidents_vehicles_casualties['Vehicle_Manoeuvre'].replace({
    'Going ahead other': 'Going ahead',
    'Turning right': 'Right',
    'Waiting to go - held up': 'Waiting',
    'Slowing or stopping': 'Slowing',
    'Turning left': 'Left',
    'Moving off': 'Moving',
    'Waiting to turn right': 'Waiting',
    'Going ahead right-hand bend': 'Going ahead',
    'Going ahead left-hand bend': 'Going ahead',
    'Overtaking moving vehicle - offside': 'Overtaking',
    'Waiting to turn left': 'Waiting',
    'Overtaking static vehicle - offside': 'Overtaking',
    'Changing lane to left': 'Changing lane',
    'Changing lane to right': 'Changing lane',
    'U-turn': 'Uturn',
    'Overtaking - nearside': 'Overtaking',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Vehicle_Location-Restricted_Lane'] =
accidents_vehicles_casualties['Vehicle_Location-Restricted_Lane'].replace({
    'On main c\way - not in restricted lane': 'Mainlane',
    'Bus lane': 'Buslane',
    'Footway (pavement)': 'Footway',
    'Leaving lay-by or hard shoulder': 'Leavinglayby',
    'On lay-by or hard shoulder': 'Onlayby',
    'Busway (including guided busway)': 'Busway',
    'Cycle lane (on main carriageway)': 'Cyclelane',
    'Tram/Light rail track': 'Tramtrack',
    'Entering lay-by or hard shoulder': 'Enteringlayby',
    'Cycleway or shared use footway (not part of  main carriageway)':
'Cycleway',
    'Data missing or out of range': 'Unknown',
})

accidents_vehicles_casualties['Junction_Location'] =
accidents_vehicles_casualties['Junction_Location'].replace({
    'Approaching junction or waiting/parked at junction approach':
'Approaching',
    'Mid Junction - on roundabout or on main road': 'Mid',
    'Cleared junction or waiting/parked at junction exit': 'Cleared',
    'Entering from slip road': 'Entering',
    'Leaving main road into minor road': 'Leaving',
    'Entering main road from minor road': 'Entering',
    'Leaving roundabout': 'Leaving',
    'Entering roundabout': 'Entering',
    'Data missing or out of range': 'Unknown',
    'Not at or within 20 metres of junction': 'No Junction',
})

accidents_vehicles_casualties['1st_Point_of_Impact'] =
accidents_vehicles_casualties['1st_Point_of_Impact'].replace({
    'Did not impact': 'Nothing',
    'Data missing or out of range': 'Unknown'
```

```
})

accidents_vehicles_casualties['Was_Vehicle_Left_Hand_Drive?'] =
accidents_vehicles_casualties['Was_Vehicle_Left_Hand_Drive?'].replace({
    'Data missing or out of range': 'Unknown'
})

accidents_vehicles_casualties['Propulsion_Code'] =
accidents_vehicles_casualties['Propulsion_Code'].replace({
    'Gas/Bi-fuel': 'Gas',
    'Petrol/Gas (LPG)': 'LPG',
})

accidents_vehicles_casualties['Casualty_Class'] =
accidents_vehicles_casualties['Casualty_Class'].replace({
    'Driver or rider': 'Driver'
})

accidents_vehicles_casualties['Sex_of_Casualty'] =
accidents_vehicles_casualties['Sex_of_Casualty'].replace({
    'Data missing or out of range': 'Unknown'
})

accidents_vehicles_casualties['Car_Passenger'] =
accidents_vehicles_casualties['Car_Passenger'].replace({
    'Data missing or out of range': 'Unknown'
})

# Define categorical and continuous columns
cat_cols =
list(accidents_vehicles_casualties.select_dtypes(include=['object']).columns)
cat_cols.remove('Accident_Severity')

cont_cols =
list(accidents_vehicles_casualties.select_dtypes(include=[np.number]).columns
)

# Anomaly Detection
# Check for outliers
plt.figure(figsize=(20, 10))
sns.boxplot(data=accidents_vehicles_casualties, orient='h')
plt.show()

# Remove outliers
def iqr(df):
    q1 = df.quantile(0.25)
    q3 = df.quantile(0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr
    return lower, upper

lower, upper = iqr(accidents_vehicles_casualties['Engine_Capacity_(CC)'])
accidents_vehicles_casualties =
accidents_vehicles_casualties[(accidents_vehicles_casualties['Engine_Capacity
_(CC)'] > lower) & (accidents_vehicles_casualties['Engine_Capacity_(CC)'] <
upper)]
```

```
# One-hot the categorical columns
accidents_vehicles_casualties = pd.get_dummies(accidents_vehicles_casualties,
columns=cat_cols, drop_first=True)

# Scale the continuous columns
def scale(x):
    return (x - x.min()) / (x.max() - x.min())

accidents_vehicles_casualties[cont_cols] =
accidents_vehicles_casualties[cont_cols].apply(scale, axis=0)

# Divide the data into features and target
X, y = accidents_vehicles_casualties.drop('Accident_Severity', axis=1),
accidents_vehicles_casualties['Accident_Severity']

#Oversample the minority class
oversample = SMOTE()
X_smote, y_smote = oversample.fit_resample(X, y)

# Check the balance of the classes
print('Processisng SMOTE...')
fig, ax = plt.subplots(1, 2, figsize = (10, 5))
y.value_counts().plot.pie(explode = [0, 0.1, 0.2], autopct = "%1.1f%%", ax =
ax[0], colors = ['#ff9999', '#66b3ff', '#964B00'],
                          shadow = True)
ax[0].set_title("Genres Before SMOTE")
ax[0].set_ylabel('')
y_smote.value_counts().plot.pie(explode = [0, 0.1, 0.2], autopct = "%1.1f%%",
ax = ax[1],  colors = ['#66b3ff', '#ff9999', '#964B00'],
                                shadow = True)
ax[1].set_title("Genres After SMOTE")
ax[1].set_ylabel('')

print("Overall dataset values have been increased.")
print("Original size:\n", y.value_counts())
print("New size:\n", y_smote.value_counts())
plt.show()

X_smote = sm.add_constant(X_smote)

print('New size of dataset:', X_smote.shape, y_smote.shape)

# Split the data into train and test
X_train, X_test, y_train, y_test = tts(X_smote, y_smote, test_size=0.2,
random_state=5805)

print('Train size:', X_train.shape, y_train.shape)
print('Test size:', X_test.shape, y_test.shape)

le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)

# Feature Selection
rfc = RandomForestClassifier(n_estimators=100, random_state=5805, n_jobs=-1)
rfc.fit(X_train, y_train)
```

```
def plot_feature_importance_with_elimination(importance, names, model_type,
threshold=0.01):
    feature_importance = np.array(importance)
    feature_names = np.array(names)
    data = {"feature_names": feature_names, "feature_importance":
feature_importance}
    fi_df = pd.DataFrame(data)

    fi_df.sort_values(by=["feature_importance"], ascending=False,
inplace=True)

    plt.figure(figsize=(10, 8))
    sns.barplot(x=fi_df["feature_importance"], y=fi_df["feature_names"])
    plt.title(f"{model_type} Feature Importance")
    plt.xlabel("Feature Importance")
    plt.ylabel("Feature Names")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    removed_features = fi_df[fi_df["feature_importance"] <
threshold]["feature_names"]
    print("Removed Features:", removed_features.tolist())

    selected_features = fi_df[fi_df["feature_importance"] >=
threshold]["feature_names"]
    print("Selected Features:", selected_features.tolist())

    return removed_features, selected_features


removed, selected = plot_feature_importance_with_elimination(
    rfc.feature_importances_, X_train.columns, "RANDOM FOREST")

X_train, X_test = X_train[selected], X_test[selected]

X = X[selected]

# Clustering

# K-Means Clustering
def calculate_WSS(points, kmax):
    sse = []
    for k in range(1, kmax + 1):
        kmeans = KMeans(n_clusters=k, random_state=5805).fit(points)
        centroids = kmeans.cluster_centers_
        pred_clusters = kmeans.predict(points)
        curr_sse = 0

        for i in range(len(points)):
            curr_center = centroids[pred_clusters[i]]
            curr_sse += (points[i, 0] - curr_center[0]) ** 2 + (points[i, 1]
- curr_center[1]) ** 2

        sse.append(curr_sse)
```

```
    return sse


points_array = X.to_numpy()

k = 10
sse = calculate_WSS(points_array, k)

# Plotting code remains the same
plt.figure()
plt.plot(np.arange(1, k+1, 1), sse)
plt.xticks(np.arange(1, k+1, 1))
plt.grid()
plt.xlabel('k')
plt.ylabel('WSS')
plt.title('k selection in k-mean Elbow Algorithm')
plt.show()


# Silhouette Score
sil = []
kmax = 10

for k in range(2, kmax+1):
    kmeans = KMeans(n_clusters=k, random_state=5805
).fit(points_array)
    labels = kmeans.labels_
    sil.append(silhouette_score(points_array, labels, metric='euclidean'))
plt.figure()
plt.plot(np.arange(2, k + 1, 1), sil, 'bx-')
plt.xticks(np.arange(2,k+1,1))
plt.grid()
plt.xlabel('k')
plt.ylabel('Silhouette score')
plt.title('Silhouette Method')
plt.show()

# DBSCAN Clustering
dbscan = DBSCAN(eps=0.4, min_samples=5)
dbscan.fit(points_array)

dbscan_y = dbscan.labels_

unique_labels = np.unique(dbscan_y)

plt.figure(figsize=(12, 8))

for label in unique_labels:
    class_member_mask = (dbscan_y == label)
    if label == -1:

        plt.plot(points_array[class_member_mask, 0],
points_array[class_member_mask, 1], 'x', color='black',
                 label='Noise', markersize=10)
    else:

        color = plt.cm.Spectral(float(label) / len(unique_labels))
```

```
        plt.plot(points_array[class_member_mask, 0],
points_array[class_member_mask, 1], 'o', color=color,
                 label=f'Cluster {label}', markersize=8)

plt.title('DBSCAN Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# Association Rules
# Create a basket such that that the basket should groupby on accident
severity agg on number of casualties, speed limit and number of vehicles
basket =
accidents_vehicles_casualties.groupby('Accident_Severity').agg({'Number_of_Ca
sualties': 'sum', 'Speed_limit': 'sum', 'Number_of_Vehicles':
'sum'}).reset_index()
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

numeric_columns = ['Number_of_Casualties', 'Speed_limit',
'Number_of_Vehicles']
basket_sets = basket[numeric_columns].applymap(encode_units)

basket_sets['Accident_Severity'] = basket['Accident_Severity']
basket_sets.set_index('Accident_Severity', inplace=True)

# Apply the Apriori algorithm
freq_items = apriori(basket_sets, min_support=0.07, use_colnames=True)
rules = association_rules(freq_items, metric="lift", min_threshold=1)
print(rules.head(5).to_string())
```

References:

[1] https://www.kaggle.com/datasets/benoit72/uk-accidents-10-years-history-with-many-variables

[2] https://matplotlib.org/stable/plot_types/index

[3] https://seaborn.pydata.org/api.html

[4] https://scikit-learn.org/stable/supervised_learning.html

[5] https://www.statsmodels.org/stable/regression.html

[6]https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/#association_rules-association-rules-generation-from-frequent-itemsets

[7]https://www.statsmodels.org/stable/generated/statsmodels.stats.outliers_influence.variance_inflation_factor.html