DEATH IN AMERICA

Erim Hayretci
Vasco de Noronha
Deniz Can Dinkci
Andrew Ize-Iyamu

Investigating Death in America

Erim Hayretci, Vasco de Noronha, Deniz Can Dinkci and Andrew Ize-Iyamu

Imperial College London, England

This report was compiled on June 24, 2024

Abstract

This project aims to investigate mortality causes in the United States through a diverse set of machine learning models applied to a data frame that includes 2.8 million recorded deaths from 2015.
Our team members analysed four main causes of death: alcohol and drug overdose, homicide, suicide, and motor accidents. For alcohol and drug overdose, a Support Vector Machine (SVM) model with an RBF kernel achieved a test accuracy of 77.8%, precision of 73.1%, and recall of 81.0%. Homicide predictions used a Logistic Regression model, yielding a validation accuracy of 82.8%, precision of 83.5%, and recall of 81.6%. Suicide risk was assessed using a Decision Tree model, resulting in a test accuracy of 75.6%, precision of 75.7%, and recall of 75.2%. Motor accidents were predicted using a Random Forest model with an accuracy of 78.4%, precision of 75.8%, and recall of 83.4%
Considering the independence and randomness of the death types examined in this study, these models prove that with correct data-driven approaches, the majority of fatalities may be precluded by implementing correct strategies for the targeted risk groups.

Keywords: *Alcohol, Drug Overdose, Homicide, Suicide, Motor Accidents*

Contents

1. Introduction

Death in the USA is a significant and complex issue, with various causes contributing to the overall mortality rate. In 2020, there were approximately **3.4 million deaths** in the USA, with heart disease, cancer, and COVID-19 being the leading causes [1]. The preventability of many of these deaths highlights the importance of early detection, effective treatment, and lifestyle modifications. However, the accessibility and cost of healthcare services remain substantial barriers. In 2019, Americans spent an average of **$11,072 per person** on healthcare, making it the most expensive healthcare system in the world [2].

This paper aims to use different models and techniques to prevent death in America via successful diagnoses and preventative measures. By leveraging data analysis and machine learning, we can identify high-risk individuals and provide timely interventions, potentially saving countless lives and reducing healthcare costs.

2. Methodology

The dataset for this analysis was obtained from the Centres for Disease Control and Prevention (CDC), which annually releases comprehensive data detailing every death in the USA. The dataset includes 2.8 million samples and 77 attributes, providing an extensive resource for analysing mortality in America. It is important to note that this dataset is specific to the United States and cannot be generalised to other countries. The primary objective of the models developed in this report was to identify the significant variables associated with each cause of death, such as age and gender. Each of the four research questions addressed different causes of death, employing distinct modelling techniques. Consequently, much of the data filtering was conducted individually, tailored to each model's requirements.

However, certain columns were deemed irrelevant to all our research questions as a group and were consequently removed. These columns included Hispanic recode, race bridge, injury at work, autopsy, burial, and activity. This collaborative filtering ensured that our models focused on the most pertinent variables, enhancing the accuracy and relevance of our findings.

3. Alcohol and Drug Overdose - Erim Hayretci

3.1. Aim

Aim of this study is to predict the likelihood of male death caused by overdose on alcohol and psychoactive drugs, based on our dataset covering 2.8 million recorded deaths in the United States for year 2015.

3.2. Introduction

Global trends indicate a concerning rise in alcohol and drug overconsumption across various demographics. According to NIH, combined deaths of overdose on alcohol and drugs have escalated from 62.30 to 97.72 per 100,000 from 2000 to 2009 [3]. The gender-specific effects are also notable. The WHO estimates that of the 67,000 deaths from alcohol poisoning worldwide, men outnumbered women by a ratio of 4 to 1.[4] Regarding drug-related cases, data suggests that in 2015, men died of overdose at twice the rate of women, which increased to 3.2 by 2021[5]. These trends that are influenced by sociocultural, behavioural, and biological factors that differentiate how genders interact with substances, indicate that focusing a machine learning algorithm on particularly male individuals may help uncover subtle patterns and risk factors, ensuring the identification of at-risk individuals and providing targeted precautions and support.

3.3. Methodology

3.3.1. Filtering, Balancing & Splitting Data

The dataset was initially filtered, revealing an all-male subset that contained 1,308,641 data points. Subsequently the set was then split into two groups based on whether individuals died from substance overconsumption, or other reasons. As expected, the dataset was extremely imbalanced which was confirmed via a confusion matrix, revealing 7,868 positive cases against 1,300,773 negative cases. To balance the data for training our algorithm, undersampling has been performed using the stratified sampling method. The major dataset (negative class) was divided into 18 age-differentiated strata. Random sampling was then conducted within each stratum, proportional to their representation in the minority group. This approach ensures the exclusion of underage individuals and prevents age-based sampling bias, leading to a more representative dataset.

Given the abundance of data points for training the model effectively, the final data is split into 60% for training and 40% for evaluation. While 9,441 data points are used to train the model by fitting its parameters, the remaining is divided into equal parts for validation and testing. The validation set is used to fine-tune the model by predicting responses for its observations, while the test set provides an unbiased evaluation of the model's final performance based its optimised parameters.

3.3.2. SVM Algorithm

To achieve our aim, a Support Vector Machine (SVM) algorithm is preferred, that uses our final data to find a hyperplane in an N-dimensional space that classifies tested data points. A kernelised model is employed to handle non-linear relationships within the data, enabling more precise differentiation between unbiased positive and negative cases, which, in our scenario, represent deaths caused by overdoses and other reasons.

3.3.3. Defining Hyperparameters

Essential hyperparameters for the kernelised SVM model include:

- **Kernel**: Transforms the input data into a higher-dimensional space with a set of reduction rules to make it easier to find a hyperplane that bounds the data points effectively. Kernel settings are expressed as mathematical function types.
- **Gamma** γ: Determines how far the influence of a single data point reaches. It affects the shape of the decision boundary where a high gamma value means that each point has a narrow area of influence, making the boundary tightly surround the points which can cause overfitting. Usually set between 1/n to 6/n where n represents the number of input fields.
- **C (Control)**: Controls the trade-off between achieving a low error on the training data and controlling regularization to avoid overfitting. Covers all positive values starting with 0.

3.3.4. Optimisation

Kernel Selection

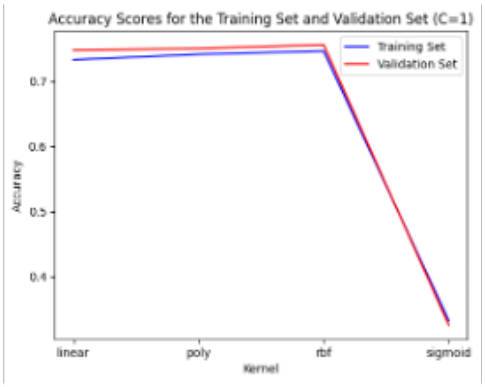

Figure 1. Accuracy of the Training and Validation Sets

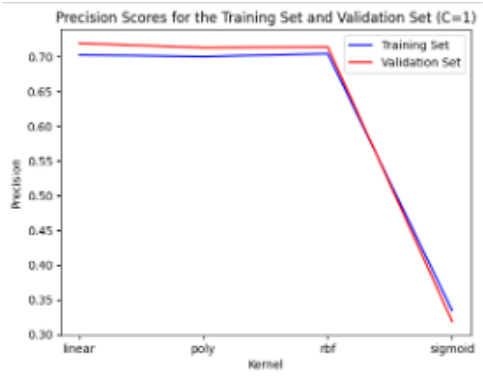

Figure 2. Precision of the Training and Validation Sets

Figure 3. Recall of the Training and Validation Sets

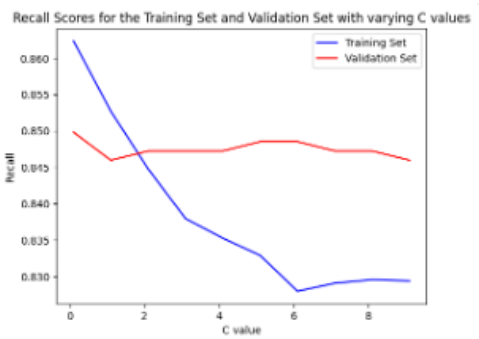

Figure 6. Recall of the Training and Validation Sets with varying C values

Apart from the sigmoid function, all three kernel types provide similar decision boundaries regarding the accuracy, precision, and recall scores of the model. However, through further investigation, RBF (Radial Basis Function) is found to offer the best performance with 75.6% accuracy and 71.3% precision in detecting correct individuals dying from substance overdose with a recall score of 85.3%. This is noted to be the default setting of the SVM algorithm.

C Value Selection

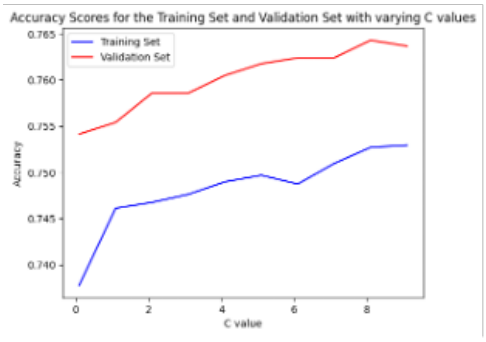

Figure 4. Accuracy of the Training and Validation Sets with varying C values

Figure 5. Precision of the Training and Validation Sets with varying C values

Due to the computational constraints of the device used for modelling, the continuous control variable, C, is tested at 10 evenly spaced points between 0 and 10. It is generally observed that increasing values of C lead to lower error margins. However, as large values of control can cause overfitting by reducing regularization, a C value of 3.0 is selected for the testing model.

Gamma Value Selection

Figure 7. Accuracy of the Training and Validation Sets with varying Gamma values

Figure 8. Precision of the Training and Validation Sets with varying Gamma values

Figure 9. Recall of the Training and Validation Sets with varying Gamma values

The default gamma value for the RBF kernel function is set to 'scaled,' which is 0.1 in our case. Testing gamma values between 1/n and 6/n showed a positive correlation with performance metrics. However, despite the positive trend, the high number of data points justifies using a simpler model with a value of 0.1 to avoid overfitting, where each point would excessively influence the model.

3.4. Results

Table 1. SVM algorithm results with selected hyperparameters

	Training Set	Validation Set	Testing Set
Accuracy	0.746	0.756	0.778
Precision	0.704	0.713	0.731
Recall	0.853	0.847	0.810

3.5. Discussion

3.5.1. Validation

The optimisation of the hyperparameters in our SVM model is observed to significantly improve its performance. By tuning the C and gamma variables and selecting the RBF kernel structure, we achieved a testing accuracy of 77.8% and precision of 73.1% which are both 2-3% higher than the scores of training and validation sets. This indicates that the model has effectively learned the underlying patterns in the data, despite the inherent randomness of predicting deaths due to overdosing.

Precision stands out as the lowest score amongst other, which refers to the risk of approaching males that do not carry the likelihood of overdosing, as at-risk individuals. The high recall score (81%), on the other hand, ensures that majority of risky cases could be accurately detected. In our scenario this trade-off is favourable, since offering a more widespread set of precautions is better than overlooking certain parts of the population which is likely to have fatal results.

3.5.2. Suggested Improvements

Future work on this model must implement a more extensive parameter optimisation process with higher computational power, testing a wider range of C and gamma values to further optimize the model's performance. Additionally, the population sample's coverage must be extended, including cases from other countries, for a more comprehensive dataset. This expansion would account for different policies and cultural attitudes towards alcohol and drug consumption, potentially improving model's generalizability across different demographics.

4. Homicide - Vasco de Noronha

4.1. Methodology

In 2015, **18,882 people** in America died from homicide-related causes. To address this, a Logistic Regression Model was developed, aiming to find associations between independent variables and the binary dependent variable of homicide-related death. This model's goal is to help mitigate future homicides in the USA by identifying the most significant predictors and informing laws and policies. Forward selection was employed to create a model with high sensitivity, identifying the features that most strongly correlate with homicide-related deaths.

4.1.1. Filtering the Data

America has one of the **highest homicide rates among affluent nations**, with approximately 5 homicides per 100,000 people, a rate stable since 2000, according to a paper from the NCJ [6]. To analyse the factors affecting this, we processed the data by sorting the manner of death column, coding homicides as 1 and other deaths as 0. We further filtered the data by removing unnecessary columns, such as autopsy status and burial information. Education index values from pre- and post-1983 were combined, and rows with any missing values or unknown variables were removed. This resulted in a dataset of 2,147,246 data points for the forward selection algorithm.

4.1.2. Balancing the Data

Due to the computational power of my device, we decided to randomly sample 20,000 deaths from the 2.15 million in our filtered dataset. We performed a confusion matrix on the dataset to test for imbalance. When examining the manner of death, specifically homicides, the dataset was highly imbalanced, with significantly more non-homicide cases (19,828) than homicide cases (172). This imbalance is likely due to the high proportion of other related deaths compared to homicides. As a result, the dataset would almost always predict a non-homicide death. To better understand the variables that influence homicide-related deaths, we needed to balance the data. Therefore, a new dataset was created, containing 10,000 homicide deaths and 10,000 non-homicide-related deaths. Despite the large size of the dataset, the effect of overfitting is likely to be minimal even though we are under-sampling.

4.1.3. Splitting the Data

The filtered and balanced data was randomly split into 50% for training, 25% for validation, and 25% for testing. Additionally, the data was split in 80/10/10, 70/15/15, and 60/20/20 proportions, as the split ratio affects the model's final accuracy. These various splits have been further analysed in the validation section 4.2.2.

4.2. Results

4.2.1. Feature Selection

A forward selection algorithm was implemented to identify the features with the highest predictive power. Initially, a model with no features was created. The feature with the largest correlation to homicide-related deaths was then added to the model. This process was repeated until no more variables remained or until the next variable changed the model's validation accuracy by less than 0.005. The final features were then saved and used to calculate their accuracy on the test set to prevent overfitting the model.

4.2.2. Validation

Many different data splits were used to find the most appropriate one. Figures 10–13 illustrate the relationship between average model accuracy and model complexity (number of variables). A common trend across all figures is that the validation data has a slightly higher mean accuracy than the training data. While this could be due to random variations, it is unlikely given the significant amount of data points, which results in a very accurate model. Figure 11, representing the 60/20/20 split, fits the model best up to a model complexity of 5, where the validation and training data accuracies begin to diverge slightly. Therefore, the 60/20/20 split was used during the logistic regression forward selection to select key features and create a model.

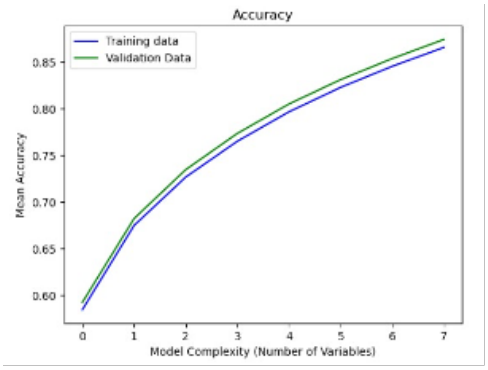

Figure 10. 50/25/25 split Model Complexity vs Model Accuracy

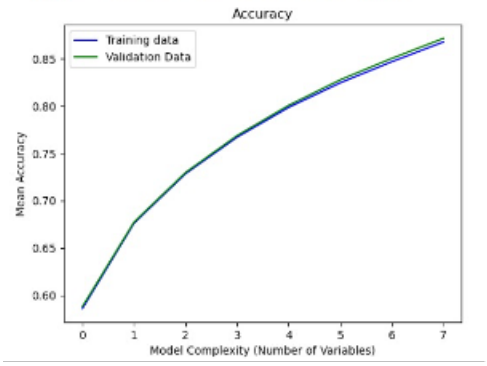

Figure 11. 60/20/20 split Model Complexity vs Model Accuracy

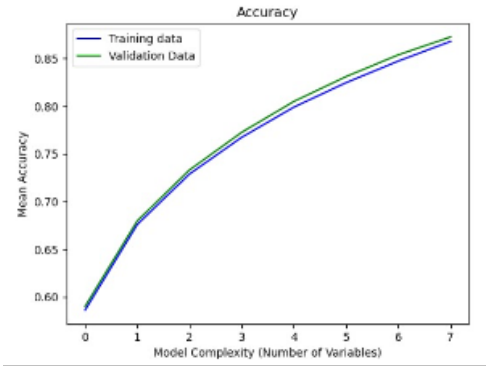

Figure 12. 70/15/15 split Model Complexity vs Model Accuracy

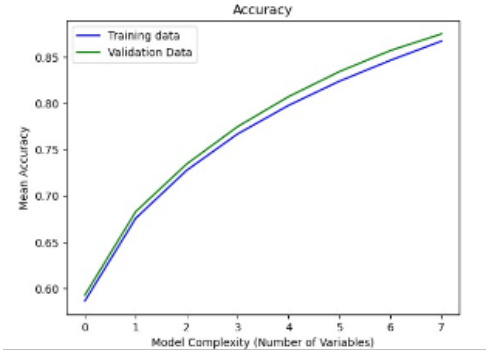

Figure 13. 80/10/10 split Model Complexity vs Model Accuracy

4.2.3. Final Results

Table 2. Results of Logistic Regression Forward Selection Model

	Accuracy	Precision	Recall
4 Training	0.798	0.807	0.788
4 Validation	0.801	0.812	0.785
5 Train	0.824	0.829	0.817
5 Validation	0.828	0.835	0.816

4.3. Discussion

The model identifies the features that most affected homicide-related deaths in the USA in 2015. These features, in order, are: Age, Race, Resident Status, Education, Marital Status, Day of the Week of Death, and Month of Death. Interestingly, the final model included all these features except the month of death, which was surprising, as it was assumed that the season would significantly affect homicide rates.

As the selected features were added, the validation accuracy increased from 0.869 to 0.878. The model's accuracy peaked at a model complexity of 4 with an accuracy of 0.867. The accuracies don't diverge greatly at complexities greater than 4, suggesting minimal overfitting in this model. This aligns with the graph in Figure 11.

The model should have high sensitivity, as it is better to overpredict the likelihood of homicide-related deaths. In the USA, a homicide has an estimated cost of $**9 million** per case [7], making it more cost-effective to overpredict and target individuals at risk of homicide rather than underpredict and miss opportunities for prevention. As mentioned, the model strongly fits the validation data up to a complexity of 4 or 5, so the results for accuracy, precision, and recall were plotted in Table 2.

For both the validation and training sets, the model values for accuracy, precision, and recall are high. However, at a complexity of 5, the model's accuracy starts to diverge slightly from the validation dataset. Therefore, a model complexity of 4 was selected to minimize error. The model achieved a relatively high recall of 0.788, resulting in high sensitivity.

To conclude, the model produced is satisfactory. However, logistic regression can still suffer from issues such as linearity and other limitations. Therefore, it is suggested to use a different model for further research to better understand and mitigate homicides in the USA.

5. Suicide - Deniz Can Dinkci

5.1. Aim

The goal of this study is to predict the probability of an individual committing suicide by investigated their background and demographics, based on an inclusive data having all deaths in the USA in 2015.

5.2. Methodology

Decision trees are non-parametric, tree-like flowchart models thar are utilized for classification and regression. Each internal node represents a feature or attribute, each branch denotes a decision rule related on that attribute, and each leaf node indicates an outcome or prediction. In the case of predicting suicides, it will be used as a classification. They structurization of the nodes is done to maximize information gain. Furthermore, as the tree splits down further, the Gini impurity decreases which ensures that the classification become progressively more reliable.

5.2.1. Data Filtering and Pre-Processing

To use a decision tree, all individual parameters are first binarized. The "manner of death" column is also binarized, assigning a value of 1 for suicides and 0 for other types of deaths. Next, unnecessary columns, such as the "method of disposition," are removed from the

dataset. Due to the significant imbalance in the data, with only 43,029 suicides compared to over 2.7 million other deaths, the dataset is undersampled to create a more balanced dataset. The dataset comprises exactly 20000 records, which 10000 of people who committed suicide and 10000 people who died from other reasons is selected to prevent overfitting.

5.2.2. Splitting the Data

The filtered data was first divided into a training set and a secondary set, with 70% allocated to the training set and 30% to the secondary set. The secondary set was then split evenly, creating a test set and a validation set, each containing half of the data from the secondary set.

This 70/30 split was chosen over the 80/20 split because of the considerably large dataset, allowing a larger portion to be allocated to validation and testing. A larger validation and testing set reduces overfitting and results in more reliable metrics.

5.2.3. Hyperparameter Optimisation

To ensure the model did not overfit, graphs displaying accuracy, precision, and recall were plotted against maximum depth and minimum impurity decrease.

Maximum Depth

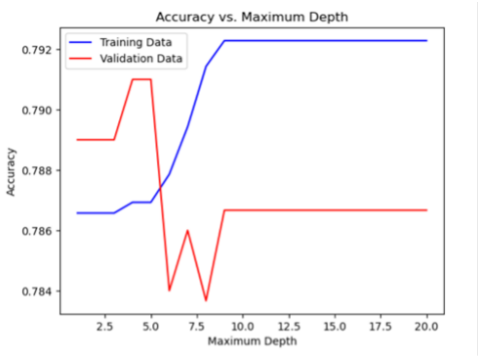

Figure 14

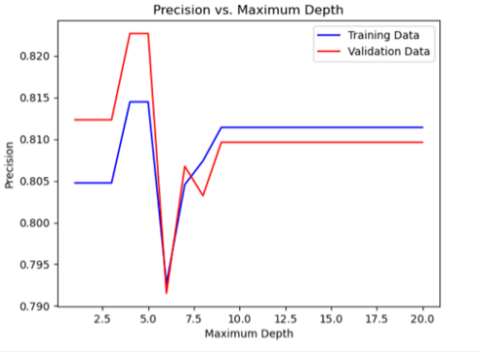

Figure 15

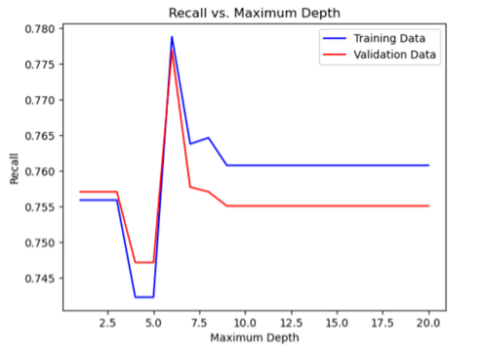

Figure 16

Figure 14 shows that the validation data peaks between maximum depths of 3 and 4, with a decline afterwards and overfitting increases after this point. Figure 15 indicates a spike in precision around the same depth values, making a depth of 4 a logical choice to pick.

Minimum Impurity Decrease

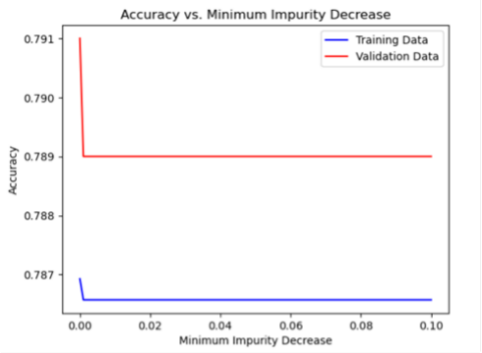

Figure 17

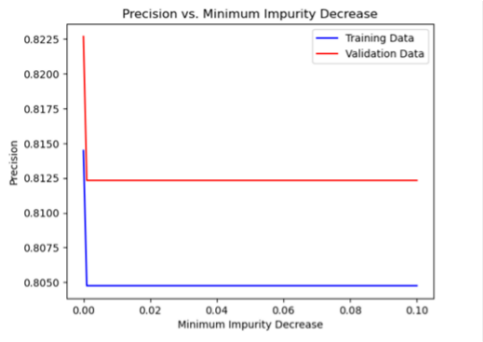

Figure 18

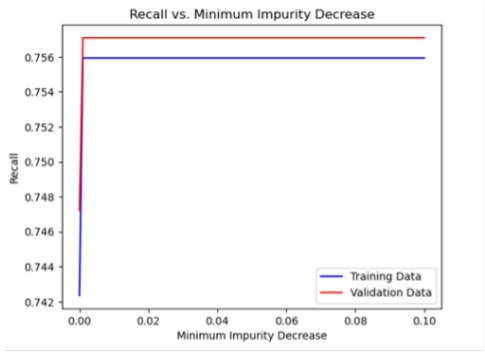

Figure 19

Minimum impurity decrease regulates the creation of new leaf nodes in a decision tree. The accuracy is constant across different values, peaking between 0 and 0.001. Although precision is lowest from 0.001 to 0.1, this comes at the at a high recall value. So, choosing the value as 0.001 is logical.

5.3. Results

5.3.1. The Decision Tree

The decision tree was constructed using the chosen hyperparameters: a minimum impurity decrease of 0.001 and a maximum depth of 4. The Gini coefficient on the leaf nodes ranged from 0 to 0.5, but none of the end leaves had a value of 0, indicating that there is a chance of misclassification for new data at every branch.

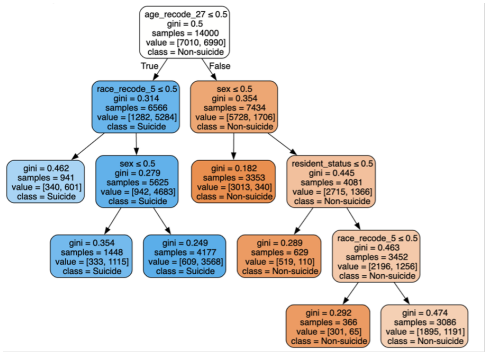

Figure 20. Decision Tree

Each of the condition statements in the nodes of the decision tree is based on whether a condition is less than or equal to 0.5. This is because the data has been binarized. If the data point is less than 0.5, the person does not exhibit that characteristic. If it is greater, they do.

5.3.2. Final Results

Table 3. Metrics of the Performance

	Training Set	Validation Set	Testing Set
Accuracy	0.787	0.805	0.756
Precision	0.789	0.812	0.757
Recall	0.789	0.812	0.752

Table 3 summarizes the performance metrics since all values in the table exceeds 70%, it indicates a great and realistic performance. The small differences between the testing dataset and validation and training datasets suggest that the model is representative of multiple datasets. Although the testing dataset has slightly lower scores, it indicates that the dataset does not overfit.

5.4. Discussion

The decision tree analysis reveals several factors that significantly influence the likelihood of suicide. Key factors identified include race recode, sex, and resident status. Each of these variables has been binarized, with the decision points in the tree indicating whether a condition is less than or equal to 0.5.

The model's decision process starts with the race recode variable. If the race recode is less than or equal to 0.5, the model proceeds to evaluate sex. If sex is less than or equal to 0.5, it further evaluates based on resident status. This hierarchical decision-making indicates the relative importance of these factors in predicting suicide.

The Gini impurity values give important insights into the model's predictions capability. The lower Gini values are higher purity and less chance of misclassification at that node occurs. For instance, a Gini impurity of 0.249 for sex ≤ 0.5 and samples $= 4177$ shows a relatively high certainty in predicting suicide for this subgroup. Meaning sex is the leading candidate of suicide and males as can be seen from the decision tree are suiciding more relative to women. Conversely, higher Gini values, such as 0.474 for race recode ≤ 0.5 and samples $= 3068$, indicate a higher chance of misclassification, suggesting less reliability in these predictions.

Though the model performed well, there are limitations to using a decision tree. Firstly, there is no guarantee that the sequence of locally optimal decisions will lead to a global optimum. Secondly, decision trees are unstable; small changes in the data split can significantly impact the resulting model. This instability can be mitigated by using the random forest method. While random forests have a higher computational cost, this would not be an issue for this relatively small dataset.

To improve the dataset, a bigger data set can be used, even though we had the possibility to us 80000 persons, we decided to go with 20000 because of the computational power of the computer we use. Secondly, this data set is taken from 2015 in USA meaning it might not be correct to implement the model to other countries with different demographics.

6. Motor Accidents - Andrew Ize-Iyamu

6.1. Aim

The model aims to determine the likelihood of motor accidents being the cause of death for individuals in the US. With crash deaths rising by 20% from 2011 to 2021 and costing the US $340 billion in 2022 [8], this research identifies key factors influencing motor accidents. By informing government bodies like the IIHS with data-driven evidence, we aim to enhance road safety regulations, reduce annual fatalities, and alleviate the socioeconomic burden of road-related accidents.

6.2. Methodology

6.2.1. Model Selection

Given the non-linear nature of predicting motor accident fatalities, a classification ML algorithm seemed the most appropriate. The available options included Decision Tree, Random Forests, and SVM. The Random Forests method was selected due to its ensemble approach, which reduces overfitting — a common issue with single decision trees. Despite SVMs' sensitivity to errors and outliers, Random Forests offer easier hyperparameter tuning and computational efficiency, making it a more cost-effective and reliable method for government bodies to generate predictions.

6.2.2. Feature Engineering

The "manner of death" column was filtered out from the updated Kaggle dataset due to its generality, while the "cause of death" column provided more detailed information. Two recoded education columns were combined to address NaN values, as Random Forests cannot handle them. Despite Random Forests' ability to manage discrete variables, remaining features were binarized to simplify decision

boundaries, enhance accuracy, and reduce overfitting and computational costs. Post-binarization, there were 36,233 motor accident fatalities and 2,481,907 other fatalities, indicating a significant class imbalance. To counteract this, we under-sampled the majority class, thus 36,233 samples were randomly selected from the majority. Then the processed dataset was standardized then split into training (80%), validation (10%), and test sets (10%) to avoid overfitting and ensure optimal model performance.

6.3. Hyperparameter Tuning

6.3.1. Prioritising Key Performance Metric

Keeping track of the ML model's performance is important, as we need to ensure that the model can provide stable and reliable predictions on whether the cause of death is related to a motor accident. Our performance metrics include:

- *Accuracy*: measures the overall correctness of the model.
- *Precision*: evaluates the correctness of positive predictions.
- *Recall*: measures the model's ability to identify all relevant instances.

In our context, the model must highlight key features impacting the likelihood of death via motoring incidents. Minimizing false negatives is crucial, as failing to identify high-risk individuals may lead to inadequate resource allocation by the US government. Consequently, recall is our primary performance metric for benchmarking and comparing the model during hyperparameter tuning. While false positives are less critical because they may still enhance safety, precision remains important. Excessive false positives could strain the government's budget by spreading resources too thin, thus reducing policy effectiveness. Accuracy is unsuitable for this classification model as it doesn't differentiate between type I and type II errors, failing to account for the skewed costs heavily favouring type I errors.

6.3.2. Hyperparameters

For the Random Forest ML method, the following parameters were available to modify:

- *N Estimators*: Number of decision tress in the random forest.
- *Maximum Depth*:Maximum depth of each decision tree.
- *Minimum Samples Leaf*: Minimum number of samples required to be a leaf node
- *Minimum Samples Split*: Minimum number of samples required to split an internal node.
- *Maximum Leaf Nodes*: Maximum number of leaf nodes in each tree.

6.3.3. Model 1: Benchmarking

Table 4. Model 1 Performance Metrics against the Training and Validation Set

Performance Metric (%)	Training Set	Validation Set
Accuracy	78.5	77.5
Precision	77.4	75.9
Recall	81.0	79.6

- *Max Depth*: Number of decision tress in the random forest.
- *Minimum Samples Split*: 0.1 (i.e. 10% of the samples).

In the table above, we can see that for the validation set, the model's recall (79.6%) is relatively higher than the model's precision (75.9%), and only marginally higher than the model's mean accuracy (77.5%). The purpose of the first model is to act as a benchmark for any future iterations of the ensemble ML method, so that we can clearly see any improvements in the model's performance post-revision of hyperparameters.

6.3.4. Model 2: Recall against Maximum Depth

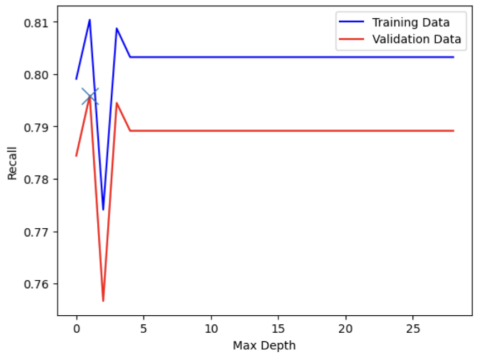

Figure 21. Recall of Model 2 against Maximum Depth

As seen in the figure above, the maximum recall for the model occurred when the maximum depth was 1. However, this would lead to underfitting within our dataset, because only variable is used by the model to make a prediction on whether the cause of death of an individual a result of a motor accident is. Thus, in exchange for marginally lower recall of 75.5% (as seen in Table 5), our model's maximum depth was increased to 4, which borders the point at which the data's recall remains unchanged at maximum depths greater than 4. However, the model's precision for the validation set has marginally improved by 1.8%. This will ensure that the model avoids underfitting the test data.

Table 5. Model 2 Performance Metrics against the Training and Validation set

Performance Metric (%)	Training Set	Validation Set
Accuracy	77.4	75.5
Precision	79.0	77.7
Recall	78.4	77.3

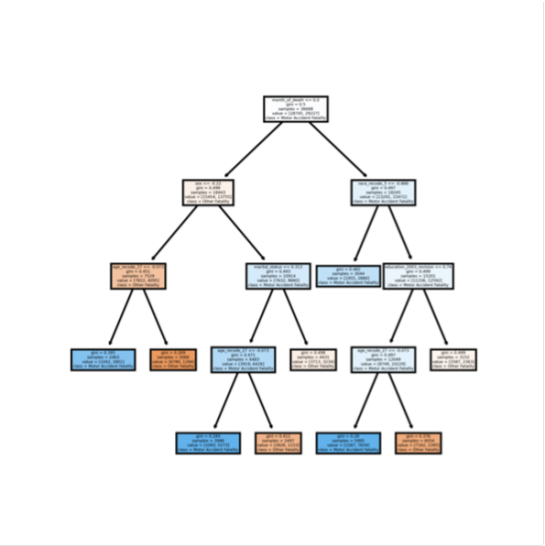

Figure 22. Estimator 4 Decision tree for the Random Forest in Model 2

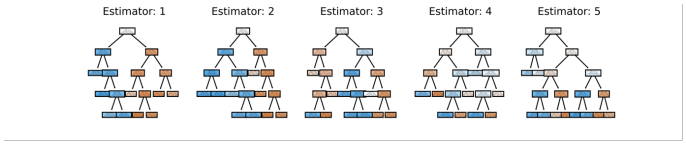

Figure 23. Random Forest in Model 2

6.3.5. Model 3: Optimising the Hyperparameters

To ensure the final model generated has the highest recall possible, the use of a very powerful computational function in the sklearn library called the *RandomizedSearchCV* allows the computer to run multiple random forests simultaneously, testing various combinations of the remaining hyperparameters until the most optimal is attained. So, after running 1000 different fits for the model with randomly assigned hyperparameters, the following optimal hyperparameters were selected:

- *N Estimators*: 39
- *Maximum Depth*: 4
- *Minimum Samples Leaf*: 33
- *Minimum Samples Split*: 0.2
- *Maximum Leaf Nodes*: 20

Some of the hyperparameters seemed nonsensical due to the randomised nature of the method to select the most optimal outcome, such as the maximum leaf nodes variable (820). The variable was reduced to a more appropriate value, since the maximum leaf nodes possible for a max depth of 4 is $2^4 = 16$. Thus, maximum number of leaf nodes was changed to 20.

6.4. Results

6.4.1. Final Model

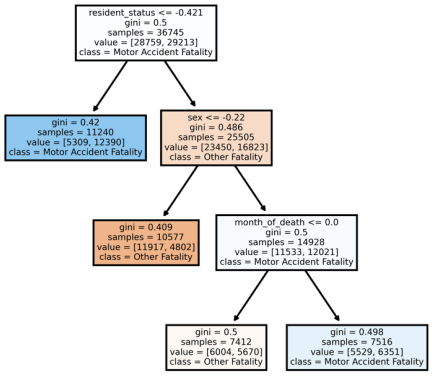

Figure 24. Estimator 2 Decision tree for the Random Forest in Model 3

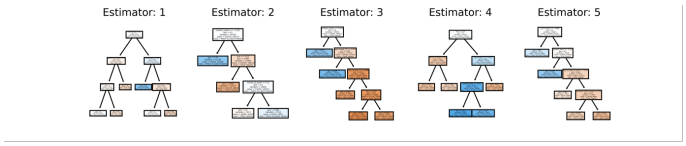

Figure 25. Random Forest in Model 3

6.4.2. Analysis

Compared to the previous models, there's been a considerable improvement in the recall for the validation set by 6.5%. However, the precision had fallen marginally by 2.7%, while the accuracy remained unchanged. This suggested that the model could perform well against unseen data, as seen with how the high recall and accuracy of the model on the test set is close to the performance on the training set.

Table 6. Model 3 Performance Metrics against the Training, Validation and Test set

Performance Metric (%)	**Training**	**Validation**	**Testing**
Accuracy	78.5	77.3	78.4
Precision	76.5	75.0	75.8
Recall	83.4	82	83.4

6.5. Discussion

6.5.1. Feature Importance

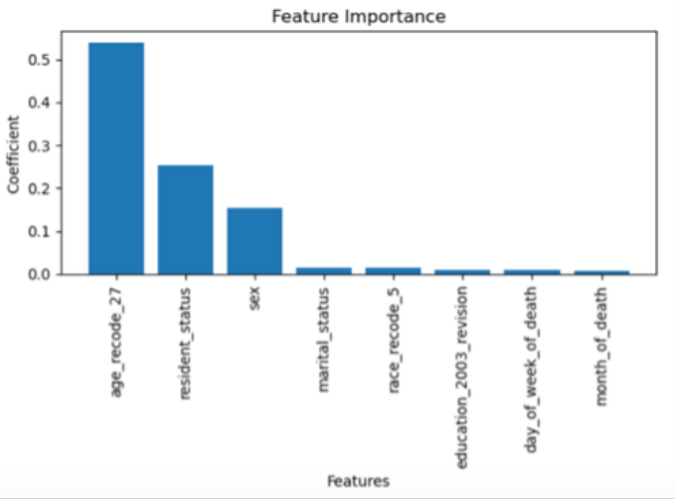

Figure 26. Feature Importance Chart for Final Model

As seen in Figure 26, the 3 most significant features for our Random Forest classifier are, in order of most significant:

1. Age
2. Resident Status
3. Sex

According to Statista [9], the largest group of licensed drivers in the US in 2021 was aged 30-34, underscoring the importance of age in our model. Younger individuals are more likely to die in motor accidents, suggesting higher recklessness among this demographic. The US government should raise road safety awareness among the younger generation to deter reckless driving and encourage adherence to safety rules. Non-residents in the US are less likely to die in motor accidents, likely due to high barriers for obtaining insurance and licenses. These barriers may result in a higher proportion of safer drivers among non-residents. Implementing similar barriers for US residents could reduce motor accident fatalities. Men are more likely than women to die in motor accidents, as they drive more miles annually and engage in riskier behaviours, such as not wearing seat belts or ignoring traffic signals. Consequently, men face a higher likelihood of encountering accidents. Road safety campaigns specifically targeting young men could help minimize and deter unlawful behaviours on roads and highways, further reducing fatalities. By addressing these factors, the US government can make informed decisions to improve road safety and reduce the socioeconomic burden of motor accidents.

6.5.2. Limitations of the Predictive Model

Our data and predictive model have limitations. The dataset lacks variables like weather, state, city, and years as a licensed driver, which

could provide more context and help US road safety bodies target critical regions in the country. Additionally, our Random Forest model could benefit from more rigorous hyperparameter tuning. Increasing iterations from 1,000 to 1,000,000 and narrowing the range of hyperparameter values could improve optimisation and ensure more sensible values for the optimised model.

7. Conclusion

The team created a series of models to predict deaths caused by suicide, homicide, motor accidents, and alcohol and drug abuse. Since each model addresses different questions, it's not feasible to determine which model is most relevant to this dataset. All models could benefit from larger datasets. The models' specificity to subsets of the data and the fact that they were trained on data from a specific country and demographic group means they may not be as relevant to the wider population.

The highest accuracy was achieved with the logistic regression forward selection model, which had a test accuracy of 0.788 and it follows by the random forest method with an test accuracy of 0.784. Through all methods, age and sex was the most significant factors in predicting these types of deaths.

This paper aims to use different models and techniques to prevent death by applying data analysis and machine learning techniques. Implementing those techniques can make us quickly identify high-risk individuals and provide timely interventions, potentially saving countless lives and reducing healthcare costs. While the current models show promise, further improvements with larger and more diverse datasets will improve the methods in addressing the complex issue of mortality in the USA.

References

[1] J. Xu, S. L. Murphy, K. D. Kochanek, and E. Arias, "Deaths: Final Data for 2019," *National Vital Statistics Reports*, vol. 70, no. 8, Jul. 2021. [Online]. Available: https://www.cdc.gov/nchs/data/nvsr/nvsr70/nvsr70-08-508.pdf.

[2] R. Tikkanen and M. K. Abrams, "U.S. health care from a global perspective, 2019: Higher spending, worse outcomes?," Jan. 2020. [Online]. Available: https://www.commonwealthfund.org/publications/issue-briefs/2020/jan/us-health-care-global-perspective-2019.

[3] A. M. White, R. W. Hingson, I.-j. Pan, and H. yi, "Hospitalizations for Alcohol and Drug Overdoses in Young Adults Ages 18–24 in the United States, 1999–2008: Results From the Nationwide Inpatient Sample," *Journal of Studies on Alcohol and Drugs*, vol. 72, no. 5, pp. 774–786, 2011. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3357438/#:~:text=Rates%20of%20drug%20overdoses%20were, (Accessed: Jun. 19, 2024).

[4] W. H. Organization, "Global Status Report on Alcohol and Health 2018," vol. 70, no. 8, Jul. 2018. [Online]. Available: https://books.google.co.uk/books?hl=en&lr=&id=qnOyDwAAQBAJ&oi=fnd&pg=PR7&ots=a2luNGwiho&sig=_JI5uwqwc75IyRgbVXnWyBPG0C8&redir_esc=y#v=onepage&q&f=false, (Accessed: Jun. 19, 2024).

[5] N. I. of Health (NIH), "Men died of overdose at 2-3 times greater a rate than women in the U.S. in 2020-2021," Jun. 2023. [Online]. Available: https://www.nih.gov/news-events/news-releases/men-died-overdose-2-3-times-greater-rate-women-us-2020-2021.

[6] R. Roth, *Why Is the United States the Most Homicidal Nation in the Affluent World? | Office of Justice Programs*, Dec. 2013. [Online]. Available: https://www.ojp.gov/ncjrs/virtual-library/abstracts/why-united-states-most-homicidal-nation-affluent-world.

[7] K. E. McCollister, M. T. French, and H. Fang, "The cost of crime to society: New crime-specific estimates for policy and program evaluation," *Drug and Alcohol Dependence*, vol. 108, no. 1–2, pp. 98–109, Apr. 2010. DOI: https://doi.org/10.1016/j.drugalcdep.2009.12.002.

[8] I.-H. crash testing and highway safety, "Fatality facts 2022: Yearly snapshot," Jun. 2022. [Online]. Available: https://www.iihs.org/topics/fatality-statistics/detail/yearly-snapshot#:~:text=A%20total%20of%2042%2C514%20people, (Accessed: Jun. 19, 2024).

[9] Statista, *U.S.: licensed drivers by age group*. [Online]. Available: https://www.statista.com/statistics/206311/total-number-of-us-licensed-drivers-in-2010-by-age/#:~:text=In%202021%2C%20about%2017.44%20million, (accessed: Jun.22.2024).

8. Appendix

8.1. Preprocessing Dataset

8.1.1. Importing PANDAS library and filtering CSV file

```
import pandas as pd  # Import the pandas library for data manipulation

# Read the entire CSV file into a DataFrame
data = pd.read_csv('2015_data.csv')

# Print the entire DataFrame to the console
print(data)

# Read the CSV file again, but this time only load specific columns into a new DataFrame
filtered_dataset = pd.read_csv('2015_data.csv', usecols=[
    'resident_status', 'education_1989_revision', 'education_2003_revision',
    'education_reporting_flag', 'month_of_death', 'sex', 'age_recode_27',
    'marital_status', 'day_of_week_of_death', 'manner_of_death', '358_cause_recode',
    'race_recode_5'
])

# Create a new DataFrame from the filtered dataset
df27 = pd.DataFrame(filtered_dataset)

# Save the new DataFrame to a new CSV file
df27.to_csv('new_data27.csv')
```

8.1.2. Converting data labels from strings to integers

```
# Create a copy of the df27 DataFrame
df27_1 = df27.copy()

# Replace 'M' with 1 and 'F' with 0 in the 'sex' column
df27_1['sex'].replace('M', 1, inplace=True)
df27_1['sex'].replace('F', 0, inplace=True)

# Replace marital status codes with numeric values
# 'W' (Widowed) -> 0
df27_1['marital_status'].replace('W', 0, inplace=True)
# 'S' (Single) -> 1
df27_1['marital_status'].replace('S', 1, inplace=True)
# 'M' (Married) -> 2
df27_1['marital_status'].replace('M', 2, inplace=True)
# 'D' (Divorced) -> 3
df27_1['marital_status'].replace('D', 3, inplace=True)
```

```
# 'U' (Unknown) -> 4
df27_1['marital_status'].replace('U', 4, inplace=True)

# Save the modified DataFrame to a new CSV file
df27_1.to_csv('new_data27_1.csv', index=False)

# Create another copy of the modified DataFrame for further processing
df27_2 = df27_1.copy()
```

8.1.3. Removing all rows that contain NaN values in the dataset

```
# Drop the 'education_reporting_flag' column from df27_2
df27_2.drop('education_reporting_flag', axis=1, inplace=True)

# Filter out rows where 'education_1989_revision' equals 99
df27_3 = df27_2[df27_2.education_1989_revision != 99]

# Print rows where 'education_1989_revision' still equals 99 (should be empty)
print(df27_3.loc[df27_3['education_1989_revision'] == 99])

# Filter out rows where 'education_2003_revision' equals 9
df27_4 = df27_3[df27_3.education_2003_revision != 9]

# Print rows where 'education_2003_revision' still equals 9 (should be empty)
print(df27_4.loc[df27_4['education_2003_revision'] == 9])

# (This line is redundant and can be removed since df27_4 is already created above)
# Filter out rows where 'education_2003_revision' equals 9
df27_4 = df27_3[df27_3.education_2003_revision != 9]

# Print rows where 'education_2003_revision' still equals 9 (should be empty)
print(df27_4.loc[df27_4['education_2003_revision'] == 9])

# Filter out rows where 'age_recode_27' equals 27
df27_5 = df27_4[df27_4.age_recode_27 != 27]

# Print rows where 'age_recode_27' still equals 27 (should be empty)
print(df27_5.loc[df27_4['age_recode_27'] == 27])

# Filter out rows where 'marital_status' equals 4
df27_6 = df27_5[df27_5.marital_status != 4]

# Print rows where 'marital_status' still equals 4 (should be empty)
print(df27_6.loc[df27_6['marital_status'] == 4])

# Filter out rows where 'day_of_week_of_death' equals 9
df27_7 = df27_6[df27_6.day_of_week_of_death != 9]

# Print rows where 'day_of_week_of_death' still equals 9 (should be empty)
print(df27_7.loc[df27_7['day_of_week_of_death'] == 9])

# Filter out rows where 'manner_of_death' equals 'Blank'
df27_8 = df27_7[df27_7.manner_of_death != 'Blank']

# Print rows where 'manner_of_death' still equals 'Blank' (should be empty)
print(df27_8.loc[df27_8['manner_of_death'] == 'Blank'])

# Save the final DataFrame to a new CSV file
df27_8.to_csv('new_data27_2.csv', index=False)
```

8.1.4. Binarization of dataset

```
# Create a copy of df27_8
df27_9 = df27_8.copy()

# Replace 'resident_status' values 2, 3, and 4 with 0
df27_9['resident_status'].replace([2,3,4], 0, inplace=True)

# Replace 'education_1989_revision' values 0-12 with 0 and values 13-17 with 1
df27_9['education_1989_revision'].replace([0,1,2,3,4,5,6,7,8,9,10,11,12], 0, inplace=True)
df27_9['education_1989_revision'].replace([13,14,15,16,17], 1, inplace=True)

# Replace 'education_2003_revision' values 0-4 with 0 and values 5-8 with 1
df27_9['education_2003_revision'].replace([0,1,2,3,4], 0, inplace=True)
df27_9['education_2003_revision'].replace([5,6,7,8], 1, inplace=True)

# Replace 'month_of_death' values 10-3 with 0 and values 4-9 with 1
df27_9['month_of_death'].replace([10,11,12,0,1,2,3], 0, inplace=True)
df27_9['month_of_death'].replace([4,5,6,7,8,9], 1, inplace=True)

# Replace 'age_recode_27' values 1-17 with 0 and values 18-26 with 1
df27_9['age_recode_27'].replace([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17], 0, inplace=True)
df27_9['age_recode_27'].replace([18,19,20,21,22,23,24,25,26], 1, inplace=True)

# Replace 'marital_status' values 0, 1, and 3 with 0, and value 2 with 1
df27_9['marital_status'].replace([0,1,3], 0, inplace=True)
df27_9['marital_status'].replace(2, 1, inplace=True)

# Replace 'day_of_week_of_death' values 1 and 7 with 0, and values 2-6 with 1
df27_9['day_of_week_of_death'].replace([1,7], 0, inplace=True)
df27_9['day_of_week_of_death'].replace([2,3,4,5,6], 1, inplace=True)

# Replace 'race_recode_5' values 0, 2, 3, and 4 with 0
df27_9['race_recode_5'].replace([0,2,3,4], 0, inplace=True)

# Print the final DataFrame to the console
print(df27_9)

# Save the final binarised DataFrame to a new CSV file
df27_9.to_csv('binarised_data.csv', index=False)
```

8.1.5. Reimporting the binarized dataset

```
import pandas as pd  # Import the pandas library for data manipulation
import numpy as np  # Import the numpy library for numerical operations

# Read the CSV file into a DataFrame
data = pd.read_csv('binarised_data.csv')

# Create a new DataFrame from the loaded data
df1 = pd.DataFrame(data)

# Drop the 'Unnamed: 0' column from the DataFrame
df1.drop('Unnamed: 0', axis=1, inplace=True)

# Print the DataFrame to the console
print(df1)
```

8.1.6. Binarizing the cause of death column for motor accidents

```
# Define a function to replace specific values
def replace_values(x):
    # If x equals 384, 389, 390, or 391, return 0
    if x == 384 or x == 389 or x == 390 or x == 391:
        return 0
    # If x is less than 384 or greater than 399, return 0
    elif x < 384 or x > 399:
        return 0
    # Otherwise, return 1
    else:
        return 1

# Apply the replace_values function to the '358_cause_recode' column
df1['358_cause_recode'] = df1['358_cause_recode'].apply(replace_values)

# Print the DataFrame after replacing values
print("After Replacing:\n", df1)
```

8.1.7. Combining the education columns of the data

```
# Set reference and target column names
reference_column = 'education_1989_revision'
target_column = 'education_2003_revision'

# Define a function to apply the mapping
def apply_mapping(row):
    # Check if the target column value is NaN
    if pd.isna(row[target_column]):
        # Convert reference column value to string
        left_value = str(row[reference_column])
        # If reference column value is "0" or "1", return it
        if left_value == "0" or left_value == "1":
            return left_value
    # Otherwise, return the original target column value
    return row[target_column]

# Apply the apply_mapping function to the target column
df1[target_column] = df1.apply(apply_mapping, axis=1)

# Drop the 'manner_of_death' column from the DataFrame
df1.drop('manner_of_death', axis=1, inplace=True)

# Drop the 'education_1989_revision' column from the DataFrame
df1.drop('education_1989_revision', axis=1, inplace=True)

# Drop rows with any NaN values
df2 = df1.dropna()
```

8.2. Alcohol and Drug Overdose - Erim Hayretci

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import numpy as np
from sklearn import svm
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.linear_model import LogisticRegression

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

file_path = '/content/drive/MyDrive/new.csv'

df = pd.read_csv(file_path)

#Filtering the dataset to take all male individuals

del df['Unnamed: 0']
del df['education_1989_revision']
filt1 = df['sex'] == 1
all_male = df[filt1]

print(all_male)


#Filtering out deaths from alcohol or substance overdose
filt2 = all_male['358_cause_recode'] == 178
filt3 = all_male['358_cause_recode'] == 179

positive = pd.concat([all_male[filt2], all_male[filt3]])

#Undersampling male dataframe

age_proportions = positive['age_recode_27'].value_counts(normalize=True)

total = []

#Stratified sampling based on age proportions of overdose deaths
for i in age_proportions.keys():
  filt5 = all_male['age_recode_27'] == i
  age_group = all_male[filt5]
  specific_age = age_group.groupby('age_recode_27', group_keys=False).apply(lambda x, frac=age_proportions[i]: x.sample(frac=frac))
  total.append(specific_age)

final = pd.concat(total)

print(final)

negative = final.groupby('age_recode_27', group_keys=False).apply(lambda x: x.sample(frac=0.09640))

print(negative)


dataset = pd.concat([positive, negative])

#Binarising cause of death
```

```
for index, row in dataset.iterrows():
  if row['358_cause_recode'] == 178:
    dataset.loc[index, '358_cause_recode'] = 1
  elif row['358_cause_recode'] == 179:
    dataset.loc[index, '358_cause_recode'] = 1
  else:
    dataset.loc[index, '358_cause_recode'] = 0

print(dataset)


from sklearn.model_selection import
    train_test_split

x_pre = dataset[['resident_status','
    education_2003_revision',
                 'month_of_death', '
    age_recode_27', 'marital_status',
                 'day_of_week_of_death', '
    manner_of_death', 'race_recode_5']]
x = x_pre.fillna(0)
y = dataset['358_cause_recode']

#Splitting a training set
x_train, x_test_val, y_train, y_test_val =
train_test_split(x, y, test_size=0.6, shuffle =
    True, random_state = 0)

#Splitting validation and testing sets
x_val, x_test, y_val, y_test =
train_test_split(x_test_val, y_test_val,
    test_size=0.5,
                 shuffle = True, random_state =
    0)

print(x_train.shape)
print(x_val.shape)
print(x_test.shape)

print(y_train.shape)
print(y_val.shape)
print(y_test.shape)

from sklearn.metrics import accuracy_score,
    precision_score, recall_score


#Accuracy, preicision and recall scores with
    different kernel types
kernel_list = ['linear', 'poly', 'rbf', 'sigmoid
    ']

accuracy_training = []
accuracy_validation =[]
precision_training = []
precision_validation =[]
recall_training = []
recall_validation =[]

for i in kernel_list:
  model = svm.SVC(kernel=i)
  model.fit(x_train, y_train)
  y_pred = model.predict(x_train)
  accuracy_training.append(accuracy_score(
    y_train, y_pred))
  precision_training.append(precision_score(
    y_train, y_pred))
  recall_training.append(recall_score(y_train,
    y_pred))

for i in kernel_list:
  model = svm.SVC(kernel=i)
  model.fit(x_val, y_val)
  y_pred = model.predict(x_val)
  accuracy_validation.append(accuracy_score(
    y_val, y_pred))
  precision_validation.append(precision_score(
    y_val, y_pred))
  recall_validation.append(recall_score(y_val,
    y_pred))

print(accuracy_training)
print(accuracy_validation)
print(precision_training)
print(precision_validation)
print(recall_training)
print(recall_validation)

#Kernel - Accuracy
plt.plot(kernel_list, accuracy_training, 'b',
    label = 'Training Set')
plt.plot(kernel_list, accuracy_validation, 'r',
    label = 'Validation Set')
plt.ylabel('Accuracy')
plt.xlabel('Kernel')
plt.title('Accuracy Scores for the Training Set
    and Validation Set (C=1)')
plt.legend()

#Kernel - Precision
plt.plot(kernel_list, precision_training, 'b',
    label = 'Training Set')
plt.plot(kernel_list, precision_validation, 'r',
    label = 'Validation Set')
plt.ylabel('Precision')
plt.xlabel('Kernel')
plt.title('Precision Scores for the Training Set
    and Validation Set (C=1)')
plt.legend()

#Kernel - Recall
plt.plot(kernel_list, recall_training, 'b',label
    = 'Training Set')
plt.plot(kernel_list, recall_validation, 'r',
    label = 'Validation Set')
plt.ylabel('Recall')
plt.xlabel('Kernel')
plt.title('Recall Scores for the Training Set
    and Validation Set (C=1)')
plt.legend()


#Accuracy, precision and recall scores with
    different C values
accuracy_training_c =[]
accuracy_validation_c =[]
precision_training_c =[]
precision_validation_c =[]
recall_training_c =[]
recall_validation_c =[]
y_data_c = []

for i in np.arange(0.1,10,1):
  y_data_c.append(i)

for i in y_data_c:
  model = svm.SVC(C=i)
  model.fit(x_train, y_train)
  y_pred = model.predict(x_train)
  accuracy_training_c.append(accuracy_score(
    y_train, y_pred))
  precision_training_c.append(precision_score(
    y_train, y_pred))
  recall_training_c.append(recall_score(y_train,
    y_pred))

for i in y_data_c:
  model = svm.SVC(C=i)
  model.fit(x_val, y_val)
  y_pred = model.predict(x_val)
  accuracy_validation_c.append(accuracy_score(
    y_val, y_pred))
  precision_validation_c.append(precision_score(
    y_val, y_pred))
  recall_validation_c.append(recall_score(y_val,
    y_pred))

print(accuracy_training_c)
print(accuracy_validation_c)
print(precision_training_c)
print(precision_validation_c)
print(recall_training_c)
```

```
print(recall_validation_c)

#C - Accuracy
plt.plot(y_data_c, accuracy_training_c, 'b', label = 'Training Set')
plt.plot(y_data_c, accuracy_validation_c, 'r', label = 'Validation Set')
plt.ylabel('Accuracy')
plt.xlabel('C value')
plt.title('Accuracy Scores for the Training Set and Validation Set
'with varying C values')
plt.legend()

#C - Precision
plt.plot(y_data_c, precision_training_c, 'b', label = 'Training Set')
plt.plot(y_data_c, precision_validation_c, 'r', label = 'Validation Set')
plt.ylabel('Precision')
plt.xlabel('C value')
plt.title('Precision Scores for the Training Set and Validation Set
'with varying C values')
plt.legend()

#C - Recall
plt.plot(y_data_c, recall_training_c, 'b',label = 'Training Set')
plt.plot(y_data_c, recall_validation_c, 'r', label = 'Validation Set')
plt.ylabel('Recall')
plt.xlabel('C value')
plt.title('Recall Scores for the Training Set and Validation Set
'with varying C values')
plt.legend()


#Accuracy, precision and recall scores with different gamma values
accuracy_training_g =[]
accuracy_validation_g =[]
precision_training_g =[]
precision_validation_g =[]
recall_training_g =[]
recall_validation_g =[]
y_data_g = []

for i in np.arange(0.1,0.6,0.05):
  y_data_g.append(i)

for i in y_data_g:
  model = svm.SVC(gamma=i)
  model.fit(x_train, y_train)
  y_pred = model.predict(x_train)
  accuracy_training_g.append(accuracy_score(y_train, y_pred))
  precision_training_g.append(precision_score(y_train, y_pred))
  recall_training_g.append(recall_score(y_train, y_pred))

for i in y_data_g:
  model = svm.SVC(gamma=i)
  model.fit(x_val, y_val)
  y_pred = model.predict(x_val)
  accuracy_validation_g.append(accuracy_score(y_val, y_pred))
  precision_validation_g.append(precision_score(y_val, y_pred))
  recall_validation_g.append(recall_score(y_val, y_pred))

print(accuracy_training_g)
print(accuracy_validation_g)
print(precision_training_g)
print(precision_validation_g)
print(recall_training_g)
print(recall_validation_g)

#Gamma - Accuracy
plt.plot(y_data_g, accuracy_training_g, 'b', label = 'Training Set')
plt.plot(y_data_g, accuracy_validation_g, 'r', label = 'Validation Set')
plt.ylabel('Accuracy')
plt.xlabel('Gamma value')
plt.title('Accuracy Scores for the Training Set and Validation Set
'with varying Gamma values')
plt.legend()

#Gamma - Precision
plt.plot(y_data_g, precision_training_g, 'b', label = 'Training Set')
plt.plot(y_data_g, precision_validation_g, 'r', label = 'Validation Set')
plt.ylabel('Precision')
plt.xlabel('Gamma')
plt.title('Precision Scores for the Training Set and Validation Set
'with varying Gamma values')
plt.legend()

#Gamma - Recall
plt.plot(y_data_g, recall_training_g, 'b',label = 'Training Set')
plt.plot(y_data_g, recall_validation_g, 'r', label = 'Validation Set')
plt.ylabel('Recall')
plt.xlabel('Gamma')
plt.title('Recall Scores for the Training Set and Validation Set
'with varying Gamma values')
plt.legend()


#Testing Set
model = svm.SVC(kernel='rbf',C=3,gamma=0.1)
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
print('test')
print(accuracy_score(y_test, y_pred))
print(precision_score(y_test, y_pred))
print(recall_score(y_test, y_pred))
```

8.3. Homicide - Vasco de Noronha

8.3.1. Filtering the dataset

```
# Combining to make 1 education column

import pandas as pd

# json code from data set was used to create this map and to make education one column which would have worked better for my model, it does this by finding nan values in the education column that we kept and looking at the older education index to fill in the gaps and preserve as many data points as possible.

mapping = {
    '10': '3',
    '11': '3',
    '12': '3',
    '13': '4',
    '14': '5',
    '15': '6',
    '16': '7',
    '17': '8',
    '99': '9',
    '00': '9',
    '01': '1', '02': '1', '03': '1', '04': '1',
    '05': '1', '06': '1', '07': '1', '08': '1',
    '09': '3'
}

```

```

df = pd.read_csv('2015_filtered.csv', low_memory
    =False)

reference_column = 'education_1989_revision'
target_column = 'education_2003_revision'

def apply_mapping(row):
    if pd.isna(row[target_column]):
        left_value = str(row[reference_column])
        if left_value in mapping:
            return mapping[left_value]
    return row[target_column]

# Apply the function to the target column
df[target_column] = df.apply(apply_mapping, axis
    =1)

df = df.drop('education_1989_revision', axis=1)

df.to_csv('2015_filtered_2.csv', index=False)

print(df.tail())

# Checking for Nan Values

import pandas as pd

df = pd.read_csv('2015_filtered_2.csv',
    low_memory=False)

has_nan = df.isnull().values.any()
nan_counts = df.isnull().sum()
rows_with_nan = df[df.isnull().any(axis=1)]

print(f"DataFrame contains NaN values: {has_nan}
    ")
print("Count of NaN values in each column:")
print(nan_counts)
print("Rows with NaN values:")
print(rows_with_nan)

if has_nan:
    print("The dataset has NaN values.")
else:
    print("The dataset has zero NaN values.")

# Removing all columns with Nan Values

import pandas as pd

df = pd.read_csv('2015_filtered_2.csv',
    low_memory=False)

# Define the columns to check for NaN values
columns_to_check = ['education_2003_revision', '
    manner_of_death']  # replace with your
    actual column names

# Remove rows with NaN values in the specified
    columns
df_cleaned = df.dropna(subset=columns_to_check)

df_cleaned.to_csv('2015_filtered_3.csv', index=
    False)

df = pd.read_csv('2015_filtered_3.csv',
    low_memory=False)

has_nan = df.isnull().values.any()
nan_counts = df.isnull().sum()
rows_with_nan = df[df.isnull().any(axis=1)]

print(f"DataFrame contains NaN values: {has_nan}
    ")
print("Count of NaN values in each column:")
print(nan_counts)
print("Rows with NaN values:")
print(rows_with_nan)

if has_nan:
    print("The dataset has NaN values.")
else:
    print("The dataset has zero NaN values.")
```

8.3.2. Balancing the dataset

```
import pandas as pd

input_file_path = '2015_filtered_3.csv'
df = pd.read_csv(input_file_path, low_memory=
    False)
# low memory set to false due to large dataset

# Separate the data into two groups based on
    manner of death
homicides = df[df['manner_of_death'] == 3]
non_homicides = df[df['manner_of_death'] != 3]

homicides_count = homicides.shape[0]
non_homicides_count = non_homicides.shape[0]


homicides_sample = homicides.sample(n=10000,
    random_state=42)
non_homicides_sample = non_homicides.sample(n
    =10000, random_state=42)

combined_sample = pd.concat([homicides_sample,
    non_homicides_sample])

output_file_path = '2015_20000_filtered_even10.
    csv'

combined_sample.to_csv(output_file_path, index=
    False)
```

8.3.3. Splitting the dataset

```
import pandas as pd

input_file_path = '2015_20000_filtered_even10.
    csv'
df = pd.read_csv(input_file_path, low_memory=
    False)

# Separate the data into two groups based on
    manner of death
# 3 was a homicide
homicides = df[df['manner_of_death'] == 3]
non_homicides = df[df['manner_of_death'] != 3]

# Determine the sample sizes
total_samples = 20000
train_size = int(total_samples * 0.6)
val_test_size = (total_samples - train_size) //
    2

#splitting into different dfs
homicides_train = homicides.sample(n=train_size,
     random_state=42)
non_homicides_train = non_homicides.sample(n=
    train_size, random_state=42)
train_sample = pd.concat([homicides_train,
    non_homicides_train])

remaining_homicides = homicides.drop(
    homicides_train.index)
remaining_non_homicides = non_homicides.drop(
    non_homicides_train.index)

homicides_val = remaining_homicides.sample(n=
    val_test_size, random_state=42)
non_homicides_val = remaining_non_homicides.
    sample(n=val_test_size, random_state=42)
val_sample = pd.concat([homicides_val,
    non_homicides_val])
homicides_test = remaining_homic
```

```
# Save all dfs to different files
train_sample.to_csv('60.20.20_train', index=False)
homicides_test.to_csv('60.20.20_test', index=False)
val_sample.to_csv('60.20.20_val', index=False)
```

8.3.4. Forward Selection algorithm

```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
import warnings

warnings.filterwarnings('ignore')

def select_column_to_add(X_train, y_train, X_val, y_val, columns_in_model, columns_to_test):
    acc_best = 0
    column_best = None

    if len(columns_in_model) == 0:
        acc_best = 0
    elif len(columns_in_model) == 1:
        mod = LogisticRegression(C=1e9).fit(X_train[columns_in_model].values.reshape(-1, 1), y_train)
        acc_best = accuracy_score(y_val, mod.predict(X_val[columns_in_model].values.reshape(-1, 1)))
    else:
        mod = LogisticRegression(C=1e9).fit(X_train[columns_in_model], y_train)
        acc_best = accuracy_score(y_val, mod.predict(X_val[columns_in_model]))

    for column in columns_to_test:
        mod = LogisticRegression(C=1e9).fit(X_train[columns_in_model + [column]], y_train)
        y_pred = mod.predict(X_val[columns_in_model + [column]])
        acc = accuracy_score(y_val, y_pred)

        if acc > acc_best:
            acc_best = acc
            column_best = column

    if column_best is not None:
        print('Adding {} to the model'.format(column_best))
        print('The new best validation accuracy is {}'.format(acc_best))
        columns_in_model_updated = columns_in_model + [column_best]
    else:
        print('Did not add anything to the model')
        columns_in_model_updated = columns_in_model

    return columns_in_model_updated, acc_best

def auto_forward_selection(X_train, y_train, X_val, y_val, max_num_of_features):
    columns_to_test = list(X_train.columns)
    columns_in_model = []
    current_acc = 0

    for i in range(0, max_num_of_features):
        columns_in_model, acc_best = select_column_to_add(X_train, y_train, X_val, y_val, columns_in_model, columns_to_test)

        if acc_best == current_acc:
            break
        else:
            current_acc = acc_best
            for feature in columns_in_model:
                if feature in columns_to_test:
                    columns_to_test.remove(feature)

    return columns_in_model, acc_best

df_train = pd.read_csv('60.20.20_train.csv', low_memory=False)
df_val = pd.read_csv('60.20.20_val.csv', low_memory=False)

df_train['manner_of_death'] = df_train['manner_of_death'].apply(lambda x: 1 if x == 3 else 0)
df_val['manner_of_death'] = df_val['manner_of_death'].apply(lambda x: 1 if x == 3 else 0)

X_train = df_train.drop(columns=['manner_of_death', 'Unnamed: 0', '358_cause_recode'])
y_train = df_train['manner_of_death']
X_val = df_val.drop(columns=['manner_of_death', 'Unnamed: 0', '358_cause_recode'])
y_val = df_val['manner_of_death']

max_num_of_features = 9

columns_selected, best_accuracy = auto_forward_selection(X_train, y_train, X_val, y_val, max_num_of_features)

print("Final selected columns:", columns_selected)
print("Best validation accuracy:", best_accuracy)
```

8.3.5. Validation

```
import itertools
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression as logreg
from sklearn.metrics import accuracy_score, precision_score, recall_score

df_train = pd.read_csv('50.25.25_train.csv', low_memory=False)
df_val = pd.read_csv('50.25.25_val.csv', low_memory=False)
df_test = pd.read_csv('50.25.25_test.csv', low_memory=False)

# binarizing manner of death, homicide =1, non-homicide death = 0
df_train['manner_of_death'] = df_train['manner_of_death'].apply(lambda x: 1 if x == 3 else 0)
df_val['manner_of_death'] = df_val['manner_of_death'].apply(lambda x: 1 if x == 3 else 0)
df_test['manner_of_death'] = df_test['manner_of_death'].apply(lambda x: 1 if x == 3 else 0)

# Split features and target variable
# removed 358_cause_recode due to the cause being dependent on manner of death
X_train = df_train.drop(columns=['manner_of_death', 'Unnamed: 0', '358_cause_recode'])
y_train = df_train['manner_of_death']
X_val = df_val.drop(columns=['manner_of_death', 'Unnamed: 0', '358_cause_recode'])
```

```
y_val = df_val['manner_of_death']
X_test = df_test.drop(columns=['manner_of_death', 'Unnamed: 0', '358_cause_recode'])
y_test = df_test['manner_of_death']

def combinations(attributes, group_size):
    return list(itertools.combinations(attributes, group_size))

# Initialize plot data dictionary
plot_data = {
    'Acc_train': [], 'Prec_train': [], 'Rec_train': [],
    'Acc_val': [], 'Prec_val': [], 'Rec_val': []
}

# List of columns to test
# All columns left after filtering
columns_to_test = [
    'age_recode_27', 'race_recode_5', 'resident_status', 'education_2003_revision', 'month_of_death', 'sex',
    'marital_status', 'day_of_week_of_death' ]

# Function gets plot data
def get_plot_data(column_to_test):
    for i in range(len(column_to_test)):
        trial_predictors = combinations(column_to_test, i + 1)
        acc_list = []
        prec_list = []
        rec_list = []
        acc_val_list = []
        prec_val_list = []
        rec_val_list = []

        for col_comb in trial_predictors:
            X = X_train[list(col_comb)]
            X_Test = X_test[list(col_comb)]
            y = y_train
            mylr = logreg(C=1e9)
            mylr.fit(X, y)

            predicted_y = mylr.predict(X)
            acc = accuracy_score(y, predicted_y)
            prec = precision_score(y, predicted_y)
            rec = recall_score(y, predicted_y)

            predicted_y_test = mylr.predict(X_Test)
            acc_val = accuracy_score(y_test, predicted_y_test)
            prec_val = precision_score(y_test, predicted_y_test)
            rec_val = recall_score(y_test, predicted_y_test)

            acc_list.append(acc)
            prec_list.append(prec)
            rec_list.append(rec)
            acc_val_list.append(acc_val)
            prec_val_list.append(prec_val)
            rec_val_list.append(rec_val)

        if acc_list and acc_val_list:
            plot_data['Acc_train'].append(np.mean(acc_list))
            plot_data['Prec_train'].append(np.mean(prec_list))
            plot_data['Rec_train'].append(np.mean(rec_list))
            plot_data['Acc_val'].append(np.mean(acc_val_list))
            plot_data['Prec_val'].append(np.mean(prec_val_list))
            plot_data['Rec_val'].append(np.mean(rec_val_list))

get_plot_data(columns_to_test)

df = pd.DataFrame(plot_data)
print(df)

# Plot the results on a graph
plt.figure()
plt.plot(df['Acc_train'], 'b')
plt.plot(df['Acc_val'], 'g')
plt.title('Accuracy')
plt.xlabel('Model Complexity (Number of Variables)')
plt.ylabel('Mean Accuracy')
plt.legend(['Training data', 'Validation Data'])
plt.show()

# Second model

import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.linear_model import LogisticRegression
import warnings
import itertools
import matplotlib.pyplot as plt

warnings.filterwarnings('ignore')

def select_column_to_add(X_train, y_train, X_val, y_val, columns_in_model, columns_to_test):
    acc_best = 0
    column_best = None

    if len(columns_in_model) == 0:
        acc_best = 0
    elif len(columns_in_model) == 1:
        mod = LogisticRegression(C=1e9).fit(X_train[columns_in_model].values.reshape(-1, 1), y_train)
        acc_best = accuracy_score(y_val, mod.predict(X_val[columns_in_model].values.reshape(-1, 1)))
    else:
        mod = LogisticRegression(C=1e9).fit(X_train[columns_in_model], y_train)
        acc_best = accuracy_score(y_val, mod.predict(X_val[columns_in_model]))

    for column in columns_to_test:
        mod = LogisticRegression(C=1e9).fit(X_train[columns_in_model + [column]], y_train)
        y_pred = mod.predict(X_val[columns_in_model + [column]])
        acc = accuracy_score(y_val, y_pred)

        if acc > acc_best:
            acc_best = acc
            column_best = column

    if column_best is not None:
        print('Adding {} to the model. Validation accuracy: {}'.format(column_best, acc_best))
        columns_in_model_updated = columns_in_model + [column_best]
    else:
        print('Did not add anything to the model')
        columns_in_model_updated = columns_in_model

    return columns_in_model_updated, acc_best

def auto_forward_selection(X_train, y_train, X_val, y_val, max_num_of_features):
    columns_to_test = list(X_train.columns)
    columns_in_model = []
    current_acc = 0
    plot_data = {
```

```
        "Acc_train": [], "Prec_train": [], "Rec_train": [],
        "Acc_val": [], "Prec_val": [], "Rec_val": []
    }

    for i in range(0, max_num_of_features):
        columns_in_model, acc_best = select_column_to_add(X_train, y_train, X_val, y_val, columns_in_model, columns_to_test)

        if acc_best == current_acc:
            break
        else:
            current_acc = acc_best
            for feature in columns_in_model:
                if feature in columns_to_test:
                    columns_to_test.remove(feature)

        # Train the model with the selected features and record metrics
        mod = LogisticRegression(C=1e9).fit(X_train[columns_in_model], y_train)
        y_train_pred = mod.predict(X_train[columns_in_model])
        y_val_pred = mod.predict(X_val[columns_in_model])

        plot_data["Acc_train"].append(accuracy_score(y_train, y_train_pred))
        plot_data["Prec_train"].append(precision_score(y_train, y_train_pred))
        plot_data["Rec_train"].append(recall_score(y_train, y_train_pred))
        plot_data["Acc_val"].append(accuracy_score(y_val, y_val_pred))
        plot_data["Prec_val"].append(precision_score(y_val, y_val_pred))
        plot_data["Rec_val"].append(recall_score(y_val, y_val_pred))

    return columns_in_model, plot_data

df_train = pd.read_csv('60.20.20_train.csv', low_memory=False)
df_val = pd.read_csv('60.20.20_val.csv', low_memory=False)
df_test = pd.read_csv('60.20.20_test.csv', low_memory=False)

df_train['manner_of_death'] = df_train['manner_of_death'].apply(lambda x: 1 if x == 3 else 0)
df_val['manner_of_death'] = df_val['manner_of_death'].apply(lambda x: 1 if x == 3 else 0)
df_test['manner_of_death'] = df_test['manner_of_death'].apply(lambda x: 1 if x == 3 else 0)

X_train = df_train.drop(columns=['manner_of_death', 'Unnamed: 0', '358_cause_recode'])
y_train = df_train['manner_of_death']
X_val = df_val.drop(columns=['manner_of_death', 'Unnamed: 0', '358_cause_recode'])
y_val = df_val['manner_of_death']
X_test = df_test.drop(columns=['manner_of_death', 'Unnamed: 0', '358_cause_recode'])
y_test = df_test['manner_of_death']

# number of columns left essentially
max_num_of_features = 8


columns_selected, plot_data = auto_forward_selection(X_train, y_train, X_val, y_val, max_num_of_features)

print("Final selected columns:", columns_selected)
print("Validation accuracies:", plot_data["Acc_val"])

# Create DataFrame from plot data
df_plot = pd.DataFrame(plot_data)
print(df_plot)

# Plot the results
plt.figure()
plt.plot(df_plot["Acc_train"], "b")
plt.plot(df_plot["Acc_val"], "g")
plt.title("Accuracy")
plt.xlabel("Model Complexity (Number of Variables)")
plt.ylabel("Mean Accuracy")
plt.legend(["Training data", "Validation Data"])
plt.show()

# Test the final model on the test set
means = X_train.mean(axis=0)
stds = X_train.std(axis=0)
X_train_standardised = (X_train - means) / stds
X_test_standardised = (X_test - means) / stds

X_final_train = X_train_standardised[columns_selected]
X_final_test = X_test_standardised[columns_selected]

final_model = LogisticRegression(C=1e9)
final_model.fit(X_final_train, y_train)
predicted_y_test = final_model.predict(X_final_test)

print("Test set accuracy:", accuracy_score(y_test, predicted_y_test))
print("Test set precision:", precision_score(y_test, predicted_y_test))
print("Test set recall:", recall_score(y_test, predicted_y_test))
```

8.4. Suicide - Deniz Can Dinkci

8.4.1. Creating the CSV as required

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the original dataset
df = pd.read_csv('binarised_data.csv')


df = df.drop(columns=['Unnamed: 0', '358_cause_recode'])


mapping = {
    '10': '3', '11': '3', '12': '3', '13': '4', '14': '5', '15': '6', '16': '7', '17': '8', '99': '9', '00': '9', '01': '1', '02': '1', '03': '1', '04': '1', '05': '1', '06': '1', '07': '1', '08': '1', '09': '3'
}

# merging two education clomuns
reference_column = 'education_1989_revision'
target_column = 'education_2003_revision'

```

```
def apply_mapping(row):
    if pd.isna(row[target_column]):
        left_value = str(row[reference_column])
        if left_value in mapping:
            return mapping[left_value]
    return row[target_column]


df['education'] = df.apply(apply_mapping, axis=1)


df = df.drop(columns=[reference_column, target_column])

# Binarize the 'manner_of_death' for my case
df['manner_of_death'] = df['manner_of_death'].apply(lambda x: 1 if x == 2.0 else 0)


non_manner_2_df = df[df['manner_of_death'] == 0]
manner_2_df = df[df['manner_of_death'] == 1]

# Randomly select 10,000 examples from each both suicide and other deaths
random_non_manner_2_sample = non_manner_2_df.sample(n=10000, random_state=42)
random_manner_2_sample = manner_2_df.sample(n=10000, random_state=42)


final_dataset = pd.concat([random_non_manner_2_sample, random_manner_2_sample])

# Saving the final csv
final_dataset.to_csv('final_dataset_20000.csv', index=False)


print(final_dataset.head())
```

8.4.2. Splitting the variables

```
df = pd.read_csv('final_dataset_20000.csv')


X = df.drop(columns=['manner_of_death'])
y = df['manner_of_death']


imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

# Split the data into training (70%) and temporary (30%) sets
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, shuffle=True, random_state=0)

# Further split the temporary set into validation 15% and 15%
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=0)


print(f'Training set: {X_train.shape}, {y_train.shape}')
print(f'Validation set: {X_val.shape}, {y_val.shape}')
print(f'Test set: {X_test.shape}, {y_test.shape}')
```

8.4.3. Accuracy vs. Maximum Depth

```
max_depths = range(1, 21)  # From 1 to 20
for max_depth in max_depths:
    dt = DecisionTreeClassifier(max_depth=max_depth, random_state=0)
    dt.fit(X_train, y_train)
    y_pred_train = dt.predict(X_train)
    y_pred_val = dt.predict(X_val)

    acc_train.append(accuracy_score(y_train, y_pred_train))
    acc_val.append(accuracy_score(y_val, y_pred_val))

# Plot the results
plt.figure()
plt.plot(max_depths, acc_train, 'b', label='Training Data')
plt.plot(max_depths, acc_val, 'r', label='Validation Data')
plt.xlabel('Maximum Depth')
plt.ylabel('Accuracy')
plt.legend(['Training Data', 'Validation Data'])
plt.title('Accuracy vs. Maximum Depth')
plt.show()
```

8.4.4. Precision vs. Maximum Depth

```
max_depths = range(1, 21)  # From 1 to 20
for max_depth in max_depths:
    dt = DecisionTreeClassifier(max_depth=max_depth, random_state=0)
    dt.fit(X_train, y_train)
    y_pred_train = dt.predict(X_train)
    y_pred_val = dt.predict(X_val)

    prec_train.append(precision_score(y_train, y_pred_train))
    prec_val.append(precision_score(y_val, y_pred_val))

# Plot the results
plt.figure()
plt.plot(max_depths, prec_train, 'b', label='Training Data')
plt.plot(max_depths, prec_val, 'r', label='Validation Data')
plt.xlabel('Maximum Depth')
plt.ylabel('Precision')
plt.legend(['Training Data', 'Validation Data'])
plt.title('Precision vs. Maximum Depth')
plt.show()
```

8.4.5. Recall vs. Maximum Depth

```
max_depths = range(1, 21)  # From 1 to 20
for max_depth in max_depths:
    dt = DecisionTreeClassifier(max_depth=max_depth, random_state=0)
    dt.fit(X_train, y_train)
    y_pred_train = dt.predict(X_train)
    y_pred_val = dt.predict(X_val)

    rec_train.append(recall_score(y_train, y_pred_train))
    rec_val.append(recall_score(y_val, y_pred_val))

# Plot the results
plt.figure()
plt.plot(max_depths, rec_train, 'b', label='Training Data')
plt.plot(max_depths, rec_val, 'r', label='Validation Data')
plt.xlabel('Maximum Depth')
```

```
plt.ylabel('Recall')
plt.legend(['Training Data', 'Validation Data'])
plt.title('Recall vs. Maximum Depth')
plt.show()
```

8.4.6. Accuracy vs. Minimum Impurity Decrease

```
min_impurity_decreases = [i / 1000 for i in range(0, 101)]  # From 0 to 0.1 in increments of 0.001
for min_impurity_decrease in min_impurity_decreases:
    dt = DecisionTreeClassifier(max_depth=5, min_impurity_decrease=min_impurity_decrease, random_state=0)
    dt.fit(X_train, y_train)
    y_pred_train = dt.predict(X_train)
    y_pred_val = dt.predict(X_val)

    acc_train.append(accuracy_score(y_train, y_pred_train))
    acc_val.append(accuracy_score(y_val, y_pred_val))

# Plot the results
plt.figure()
plt.plot(min_impurity_decreases, acc_train, 'b', label='Training Data')
plt.plot(min_impurity_decreases, acc_val, 'r', label='Validation Data')
plt.xlabel('Minimum Impurity Decrease')
plt.ylabel('Accuracy')
plt.legend(['Training Data', 'Validation Data'])
plt.title('Accuracy vs. Minimum Impurity Decrease')
plt.show()
```

8.4.7. Precision vs. Minimum Impurity Decrease

```
min_impurity_decreases = [i / 1000 for i in range(0, 101)]  # From 0 to 0.1 in increments of 0.001
for min_impurity_decrease in min_impurity_decreases:
    dt = DecisionTreeClassifier(max_depth=5, min_impurity_decrease=min_impurity_decrease, random_state=0)
    dt.fit(X_train, y_train)
    y_pred_train = dt.predict(X_train)
    y_pred_val = dt.predict(X_val)

    prec_train.append(precision_score(y_train, y_pred_train))
    prec_val.append(precision_score(y_val, y_pred_val))

# Plot the results
plt.figure()
plt.plot(min_impurity_decreases, prec_train, 'b', label='Training Data')
plt.plot(min_impurity_decreases, prec_val, 'r', label='Validation Data')
plt.xlabel('Minimum Impurity Decrease')
plt.ylabel('Precision')
plt.legend(['Training Data', 'Validation Data'])
plt.title('Precision vs. Minimum Impurity Decrease')
plt.show()
```

8.4.8. Recall vs. Minimum Impurity Decrease

```
min_impurity_decreases = [i / 1000 for i in range(0, 101)]  # From 0 to 0.1 in increments of 0.001
for min_impurity_decrease in min_impurity_decreases:
    dt = DecisionTreeClassifier(max_depth=5, min_impurity_decrease=min_impurity_decrease, random_state=0)
    dt.fit(X_train, y_train)
    y_pred_train = dt.predict(X_train)
    y_pred_val = dt.predict(X_val)

    rec_train.append(recall_score(y_train, y_pred_train))
    rec_val.append(recall_score(y_val, y_pred_val))

# Plot the results
plt.figure()
plt.plot(min_impurity_decreases, rec_train, 'b', label='Training Data')
plt.plot(min_impurity_decreases, rec_val, 'r', label='Validation Data')
plt.xlabel('Minimum Impurity Decrease')
plt.ylabel('Recall')
plt.legend(['Training Data', 'Validation Data'])
plt.title('Recall vs. Minimum Impurity Decrease')
plt.show()
```

8.5. Motor Accidents - Andrew Ize-Iyamu

8.5.1. Balancing and exporting the dataset

```
# Filter df2 to create df1_1 containing rows where '358_cause_recode' equals 1
df1_1 = df2.loc[df2['358_cause_recode'] == 1]

# Filter df2 to create df1_0 containing rows where '358_cause_recode' equals 0
df1_0 = df2.loc[df2['358_cause_recode'] == 0]

# Randomly sample 36233 rows from df1_0 to create reduced_df1_0
reduced_df1_0 = df1_0.sample(n=36233)

# Concatenate df1_1 and reduced_df1_0 to create the final DataFrame
final_df = pd.concat([df1_1, reduced_df1_0])

# Print the final DataFrame to the console
print(final_df)

# Save the final DataFrame to a CSV file
final_df.to_csv('true_final_data.csv', index=False)
```

8.5.2. Importing binarised data

```
import pandas as pd  # Import the pandas library for data manipulation
import numpy as np  # Import the numpy library for numerical operations
import matplotlib.pyplot as plt  # Import the matplotlib library for plotting

# Read the CSV file into a DataFrame
data = pd.read_csv('true_final_data.csv')

# Create a new DataFrame from the loaded data
df = pd.DataFrame(data)

# Drop the 'Unnamed: 0' column from the DataFrame
df.drop('Unnamed: 0', axis=1, inplace=True)

# Print the DataFrame to the console
print(df)
```

8.5.3. Standardising and splitting the dataset

```
from sklearn.model_selection import train_test_split
import pandas as pd

# Assuming df is already defined and contains your data

# Define features (X) and target (y)
X = df[['resident_status', 'education_2003_revision', 'month_of_death', 'sex', 'age_recode_27', 'marital_status', 'day_of_week_of_death', 'race_recode_5']]
y = df['358_cause_recode']

# Split the data into train, validation, and test sets
# First split: train and combined validation/test (X_test_val, y_test_val)
X_train, X_test_val, y_train, y_test_val = train_test_split(X, y, test_size=0.2, shuffle=True, random_state=0)

# Second split: split X_test_val and y_test_val into validation and test sets
X_val, X_test, y_val, y_test = train_test_split(X_test_val, y_test_val, test_size=0.5, random_state=0)

# Calculate means and standard deviations of features from training set
X_means = X_train.mean(axis=0)
X_stds = X_train.std(axis=0)

# Standardize the data using the mean and standard deviation from the training set
X_train = (X_train - X_means) / X_stds
X_val = (X_val - X_means) / X_stds
X_test = (X_test - X_means) / X_stds

# Print the shapes of the datasets to verify the splits and standardization
print('X shape: {}'.format(X.shape))
print('X_train shape: {}'.format(X_train.shape))
print('X_test shape: {}'.format(X_test.shape))
print('X_val shape: {}'.format(X_val.shape))
```

8.5.4. Model 1: Benchmarking

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Define the Random Forest classifier with specified parameters
model = RandomForestClassifier(max_depth=2, min_samples_split=0.1, random_state=0)

# Train the model on the training data
model = model.fit(X_train, y_train)

# Predictions on the training, validation, and test sets
ypred_train = model.predict(X_train)
ypred_val = model.predict(X_val)
ypred_test = model.predict(X_test)

# Calculate accuracy, precision, and recall scores for the training set
acc_train = accuracy_score(y_train, ypred_train)
prec_train = precision_score(y_train, ypred_train)
rec_train = recall_score(y_train, ypred_train)

# Calculate accuracy, precision, and recall scores for the validation set
acc_val = accuracy_score(y_val, ypred_val)
prec_val = precision_score(y_val, ypred_val)
rec_val = recall_score(y_val, ypred_val)

# Calculate accuracy, precision, and recall scores for the test set
acc_test = accuracy_score(y_test, ypred_test)
prec_test = precision_score(y_test, ypred_test)
rec_test = recall_score(y_test, ypred_test)

# Print the results for the training set
print('Performance Metrics for Model 1 against Training set:')
print('Accuracy Training: {}'.format(acc_train))
print('Precision Training: {}'.format(prec_train))
print('Recall Training: {}'.format(rec_train))
print('')

# Print the results for the validation set
print('Performance Metrics for Model 1 against Validation set:')
print('Accuracy Validation: {}'.format(acc_val))
print('Precision Validation: {}'.format(prec_val))
print('Recall Validation: {}'.format(rec_val))
print('')
```

8.5.5. Model 2: Plotting Recall of Model 2 against Maximum Depth

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Lists to store recall scores for training and validation sets
rec_train = []
rec_val = []

# Loop through different max_depth values from 1 to 14
for i in range(1, 15):
    # Initialize RandomForestClassifier with current max_depth and other parameters
    dt = RandomForestClassifier(max_depth=i, min_samples_split=0.1, random_state=0)

    # Train the model on the training data
    dt.fit(X_train, y_train)

    # Predictions on training and validation sets
    ypred_train = dt.predict(X_train)
    ypred_val = dt.predict(X_val)

    # Calculate recall scores and append to rec_train and rec_val lists
    rec_train.append(recall_score(y_train, ypred_train))
    rec_val.append(recall_score(y_val, ypred_val))

# Plot recall scores for training and validation sets across different max_depth values
plt.figure()
plt.plot(rec_train, 'b', label='Training Data')
plt.plot(rec_val, 'r', label='Validation Data')
plt.xlabel('Max Depth')
plt.ylabel('Recall')
plt.legend()
plt.xticks(np.arange(0, 14, step=1))
plt.grid(True)
max_recall = np.max(rec_val)
```

```
plt.plot(np.argmax(rec_val), max_recall, marker='x', markersize=15)  # Highlight maximum recall point
plt.title('Recall vs Max Depth')
plt.show()

# Choose the best max_depth based on validation performance
best_max_depth = np.argmax(rec_val) + 1  # +1 because max_depth starts from 1

# Train the final model using the best max_depth on the entire training set
model = RandomForestClassifier(max_depth=best_max_depth, min_samples_split=0.1, random_state=0)
model.fit(X_train, y_train)

# Predictions on training, validation, and test sets using the final model
ypred_train = model.predict(X_train)
ypred_val = model.predict(X_val)

# Calculate accuracy, precision, and recall scores for the training set
acc_train = accuracy_score(y_train, ypred_train)
prec_train = precision_score(y_train, ypred_train)
rec_train = recall_score(y_train, ypred_train)

# Calculate accuracy, precision, and recall scores for the validation set
acc_val = accuracy_score(y_val, ypred_val)
prec_val = precision_score(y_val, ypred_val)
rec_val = recall_score(y_val, ypred_val)

# Print the results for the training set
print('Performance Metrics for Model 2 against Training set:')
print('Accuracy Training: {}'.format(acc_train))
print('Precision Training: {}'.format(prec_train))
print('Recall Training: {}'.format(rec_train))
print('')

# Print the results for the validation set
print('Performance Metrics for Model 2 against Validation set:')
print('Accuracy Validation: {}'.format(acc_val))
print('Precision Validation: {}'.format(prec_val))
print('Recall Validation: {}'.format(rec_val))
print('')
```

8.5.6. Plotting Estimator 4 Decision Tree for Model 2

```
from sklearn import tree
import matplotlib.pyplot as plt

# Assuming model is your trained RandomForestClassifier
# and you want to plot the 4th estimator (index 3) as an example
# Change the index (3) to plot a different tree from the Random Forest

plt.figure(figsize=(4, 4), dpi=800)  # Set the figure size and DPI

# Plot the individual tree
tree.plot_tree(model.estimators_[3],
               feature_names=X_train.columns,
               class_names=['Other Fatality', 'Motor Accident Fatality'],
               filled=True);

plt.savefig('rf_individualtree1.png')  # Save the figure
plt.show()  # Display the plot
```

8.5.7. Plotting Random Forest for Model 2

```
import matplotlib.pyplot as plt
from sklearn import tree

# Assuming model is your trained RandomForestClassifier
# and you want to plot the first 5 estimators
# Adjust the range (0, 5) to plot different sets of trees

fig, axes = plt.subplots(nrows=1, ncols=5, figsize=(10, 2), dpi=900)  # Adjust figsize and dpi as needed

for index in range(0, 5):
    tree.plot_tree(model.estimators_[index],
                   feature_names=X_train.columns,
                   class_names=['Other Fatality', 'Motor Accident Fatality'],
                   filled=True,
                   ax=axes[index])

    axes[index].set_title('Estimator: ' + str(index + 1), fontsize=11)

fig.savefig('rf_5trees1.png')  # Save the figure with all trees
plt.show()  # Display the plot
```

8.5.8. Using Random Search Cross Validation to find optimal combination of hyperparameters

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform
from sklearn.metrics import recall_score

# Initialize the Random Forest model
rf = RandomForestClassifier(random_state=42)

# Define the parameter distribution for Randomized Search
param_dist = {
    'n_estimators': randint(1, 500),           # Number of trees in the forest
    'max_depth': randint(1, 10),               # Maximum depth of the trees
    'min_samples_leaf': randint(1, 100),       # Minimum number of samples required to be at a leaf node
    'min_samples_split': uniform(0.01, 0.49), # Minimum number of samples required to split an internal node
    'max_leaf_nodes': randint(1, 300)          # Maximum number of leaf nodes
}

# Perform Randomized Search with Cross-Validation
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist,
    n_iter=200,   # Number of parameter settings that are sampled
    scoring='recall',  # Use recall as the scoring metric
    cv=5,         # Number of cross-validation folds
    verbose=2,
    random_state=42,
    n_jobs=-1     # Use all available cores
)

```

```
# Fit the Randomized Search to find the best parameters
random_search.fit(X_train, y_train)

# Get the best parameters and best score
best_params = random_search.best_params_
best_recall = random_search.best_score_

print(f"Best Parameters: {best_params}")
print(f"Best Recall: {best_recall}")
```

8.5.9. Final Model

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Define and initialize the RandomForestClassifier with specified hyperparameters
model = RandomForestClassifier(max_depth=4,
                               max_leaf_nodes=36,
                               min_samples_leaf=33,
                               min_samples_split=0.2,
                               n_estimators=39,
                               random_state=42)
 # Set random_state for reproducibility

# Fit the model on the training data
model.fit(X_train, y_train)

# Predictions on training set
ypred_train = model.predict(X_train)

# Predictions on validation set
ypred_val = model.predict(X_val)

# Predictions on test set
ypred_test = model.predict(X_test)

# Calculate evaluation metrics for training set
acc_train = accuracy_score(y_train, ypred_train)
prec_train = precision_score(y_train, ypred_train)
rec_train = recall_score(y_train, ypred_train)

# Calculate evaluation metrics for validation set
acc_val = accuracy_score(y_val, ypred_val)
prec_val = precision_score(y_val, ypred_val)
rec_val = recall_score(y_val, ypred_val)

# Calculate evaluation metrics for test set
acc_test = accuracy_score(y_test, ypred_test)
prec_test = precision_score(y_test, ypred_test)
rec_test = recall_score(y_test, ypred_test)

# Print the results for the training set
print('Performance Metrics for Model 3 against Training set:')
print('Accuracy Training: {}'.format(acc_train))
print('Precision Training: {}'.format(prec_train))
print('Recall Training: {}'.format(rec_train))
print('')

# Print the results for the validation set
print('Performance Metrics for Model 3 against Validation set:')
print('Accuracy Validation: {}'.format(acc_val))
print('Precision Validation: {}'.format(prec_val))
print('Recall Validation: {}'.format(rec_val))
print('')

# Print the results for the test set
print('Performance Metrics for Model 3 against Test set:')
print('Accuracy Test: {}'.format(acc_test))
print('Precision Test: {}'.format(prec_test))
print('Recall Test: {}'.format(rec_test))
print('')
```

8.5.10. Plotting Estimator 2 Decision Tree for Final Model

```
from sklearn import tree
import matplotlib.pyplot as plt

# Assuming 'model' is your trained RandomForestClassifier and you want to plot the second estimator (index 1)
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(4, 4), dpi=800)

# Plot the tree
tree.plot_tree(model.estimators_[1],  # Change index to plot different trees
               feature_names=X_train.columns,
               class_names=('Other Fatality', 'Motor Accident Fatality'),
               filled=True,
               ax=axes)

axes.set_title('Estimator 2', fontsize=11)  # Set title for the plot

# Save the figure
fig.savefig('rf_individualtree2.png')

# Display the plot (optional)
plt.show()
```

8.5.11. Plotting Random Forest for Final Model

```
import matplotlib.pyplot as plt
from sklearn import tree

# Assuming 'model' is your trained RandomForestClassifier
fig, axes = plt.subplots(nrows=1, ncols=5, figsize=(20, 4), dpi=900)  # Adjust figsize as needed

# Plot each tree in a separate subplot
for index in range(5):  # Plotting 5 trees as per your code
    tree.plot_tree(model.estimators_[index],
                   feature_names=X_train.columns,
                   class_names=('Other Fatality', 'Motor Accident Fatality'),
                   filled=True,
                   ax=axes[index])

    axes[index].set_title('Estimator: ' + str(index + 1), fontsize=11)

# Adjust layout to prevent overlapping
plt.tight_layout()

# Save the figure
fig.savefig('rf_5trees2.png')

# Display the plot (optional)
plt.show()
```

8.5.12. Feature Importance Plot

```
import numpy as np
import matplotlib.pyplot as plt
```

```
3
4  # Assuming 'model' is your trained
       RandomForestClassifier
5  importances = model.feature_importances_
6
7  # Sort the feature importance indices in
       descending order
8  sorted_indices = np.argsort(importances)[::-1]
9
10 # Plotting the feature importances
11 plt.figure(figsize=(10, 6))  # Adjust figsize as
       needed
12 plt.bar(range(X_train.shape[1]), importances[
       sorted_indices], align='center', alpha=0.8)
13 plt.xticks(range(X_train.shape[1]), X_train.
       columns[sorted_indices], rotation=90)
14 plt.xlabel('Features')
15 plt.ylabel('Importance Score')
16 plt.title('Feature Importances')
17
18 plt.tight_layout()
19 plt.show()
```