Exploring the Performance of Crime Prediction Using Machine Learning Approaches

Marcus Valerio
VAL25150223

25150223@students.lincoln.ac.uk

School of Computer Science
College of Science
University of Lincoln

Submitted in partial fulfilment of the requirements for the
Degree of BSc(Hons) Computer Science

Supervisor: Dr. Vassilis Cutsuridis

April 2023

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Vassilis Cutsuridis, for his invaluable guidance and support throughout the course of this project. His knowledge has been instrumental in shaping my research and helping me to achieve my goals.

I would also like to extend my thanks to Dr. Mubeen Ghafoor, a lecturer who provided me with valuable insights and feedback on my work. His comments and suggestions have been tremendously helpful in improving the quality of my research.

Finally, I want to express my deepest appreciation to my family for their unwavering support and encouragement. Their patience and understanding have been a constant source of motivation for me, and I am forever grateful.

Abstract

According to the Office for National Statistics (2023), in the year ending September 2022, adults aged 16 years and over experienced 9.1 million offences. These high crime rates can be significant and far-reaching, affecting individuals and society in the short and long term. Machine learning techniques can be implemented to predict crimes that may occur in the future based on information such as the area, date and time. However, it is not clear which approach is best. Therefore, this project aims to evaluate different machine learning approaches to find the best way to predict crime accurately. Four machine learning classifiers will be compared: Naive Bayes, Decision Tree, Naive Bayes and K-Nearest Neighbours. The use of K-Fold Cross Validation and Hyperparameter Tuning will also be measured based on a variety of performance metrics. The findings show that using Hyperparameter Tuning is necessary to yield better results, and the Random Forest classifier with Hyperparameter Tuning yields the best results.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Between 2022 and 2023, the crime rate in the United Kingdom increased by 8% to 75.88 per 1000 people (CrimeRate, 2022). According to the Office for National Statistics (2023), in the year ending September 2022, adults aged 16 years and over experienced 9.1 million offences. These high crime rates can be significant and far-reaching, affecting individuals and society in the short and long term. Victims may experience physical and psychological harm, economic losses, and privacy violations. These effects can also extend to communities, with anti-social behaviour contributing to negative perceptions of safety and the surrounding environment and fear of crime hindering social functioning and community cohesion (Stripe, 2022). Accurate crime prediction can reduce these negative effects, enabling law enforcement agencies to better allocate resources to the places and times most needed. This will improve public safety, reduce crime rates, and result in cost savings as fewer resources will be required to support law enforcement and criminal justice departments.

Predictive policing techniques have been implemented by police departments across the United States, including the Los Angeles Police Department (LAPD), New York Police Department (NYPD), and Chicago Police Department (CPD), to predict areas of gun violence or individuals who may be involved in it. However, these projects have been terminated for various reasons, as analysis has found the models ineffective and inconsistent in selecting and retaining individuals in police databases (Lau, 2020). In contrast, the Manchester Police reduced robberies, burglaries, and thefts by double digits in the first ten weeks of implementing predictive measures (Isafiade, n.d.). Nevertheless, predictive policing methods have raised concerns among human rights groups due to the use of historical data, which could reflect patterns of discrimination and perpetuate them in police practices. "The public is given very little information about how predictive algorithms reach their decisions [...] This lack of transparency and understanding means these programs cannot be properly scrutinised, and life-changing decisions are near-impossible to challenge" (Liberty, n.d.). To address these issues, this project aims to develop a software artefact that trains and tests a range of machine learning (ML) algorithms on publicly available crime data and evaluates their performance using a set of performance metrics. Unlike previous research, this study will utilise Hyperparameter Tuning (HPT) to optimise the ML algorithms and improve their accuracy in predicting crime.

A large amount of data from criminal records is required to predict crime accurately. However, when faced with tens of thousands of records, human reasoning can be overwhelmed, leading to inaccuracies and inefficiencies. Therefore, automating this process with a digital tool that can consistently process this large amount of data and accurately predict crime is necessary. Machine learning, a branch of artificial intelligence and computer science, use data and algorithms to imitate how humans learn from their surroundings.

Machine learning algorithms can quickly and efficiently process large datasets, making it easier to identify patterns and trends (www.ibm.com., n.d.).

Machine learning can be divided into two different learning approaches: supervised learning and unsupervised learning. Supervised learning involves using labelled datasets to train or "supervise" algorithms to predict outputs, while unsupervised learning analyses unlabelled datasets to find patterns within the data. This project will use supervised learning since it is imperative to use previous data, including the crime and its features, to evaluate predictions based on the actual crime. Moreover, since the dataset used for this project uses categorical values, this project becomes a classification problem, not a regression problem. The dataset includes crimes reported in Boston, USA, from 2015 to 2018. The study aims to determine which ML algorithm can most accurately predict which of three crimes based on features given from a set of test data after the model has been trained. As the number of classes being predicted is more than two, the type of classification employed is multiclass. The classification ML algorithms that will be used for this project are Naive Bayes (NB), Decision Tree (DT), Random Forest (RF), and K-Nearest Neighbours (KNN).

In machine learning, model parameters are configuration variables internal to a model whose values are estimated from data, not set manually by the user. Examples include weights in an artificial neural network or the network size. Models with a fixed or variable number of parameters are known as parametric or non-parametric, respectively. In contrast, hyperparameters control the learning process and determine the values of model parameters learned by a machine learning algorithm. While hyperparameters are not part of the resulting model, they significantly influence its performance (Brownlee, 2017). For example, the number of branches in a decision tree is a hyperparameter. Hyperparameter tuning (HPT) is often employed to optimise machine learning models' performance. This involves trying different combinations of hyperparameters to identify the best set for a given problem. Grid search is a commonly used HPT method that divides the hyperparameter domain into a grid and evaluates each combination. A study by Schratz et al. found that machine learning models, except for support vector machines, outperformed parametric models regarding predictive performance (Schrats et al., 2019). This study will implement grid search to optimise hyperparameters for each classification algorithm: Naive Bayes, Decision Tree, Random Forest, and K-Nearest Neighbours. The specific hyperparameters tuned for each algorithm will be determined based on their respective hyperparameter spaces.

1.1 Dissertation Structure

The structure of this dissertation is as follows:

- **Literature Review:** a critical evaluation of relevant academic literature
- **Requirements Analysis:** what the solution must do to produce the desired outcomes
- **Design & Methodology:** outlining how the project will be developed
- **Implementation:** a description of the software's components
- **Results & Discussion:** a presentation of the results and discussion of their success
- **Conclusion:** a summary of the whole project
- **References:** a list of Harvard-style references

- **Appendices:** supplementary content

1.2 Aims & Objectives

1.2.1 Aim

This project aims to develop an application that measures the success of multiple machine learning algorithms in predicting crime.

1.2.2 Objectives

- Extracted a dataset from the public domain and include crime from Boston, USA, from 2015 to 2018
- The data will include the crime type, the district, the year, the month, the day of the week and the hour that the crime was committed and be in a comma-separated values (CSV) format
- Machine learning algorithms will be trained and tested from this data, including Naive Bayes, Decision Tree, Random Forest, and K-Nearest Neighbours
- Metrics will be used to evaluate the performance of the model's predictions, which include accuracy, balanced accuracy, precision, recall, F1-score, ROC AUC score, and confusion matrix
- Implement K-fold cross-validation and hyperparameter tuning into the software artefact to build on the model's performance and show potential improvement

Chapter 2

Literature Review

2.1 Data

The quantity of data required by a machine learning algorithm for crime prediction depends greatly on the complexity of the problem and the number of features. Guerette and Bowers (2009) found that larger datasets made more accurate predictions when predicting future crime hotspots. To support this, research conducted by Gerber (2014) discovered that using more data, which in this case was Twitter messages (tweets), improved the accuracy of their predictive policing model. To counter this point, Mohler et al. (2011) found that smaller datasets can also be effective for crime prediction; a small dataset of crime incidents could accurately predict future crime hotspots. However, the models used in these studies predicted hotspots of crimes using clustering models rather than classification techniques to predict the crimes themselves.

Data cleaning is an important step when dealing with large datasets in machine learning to reduce incorrect entries in the data. A study by Almanie, Mirza and Lor (2015) used multiple stages in their process of data preprocessing in order to help predict criminal hotspots using machine learning models. These included 'data cleaning' to remove missing values, 'data reduction' to decrease the number of attributes and entries for each class, 'data integration', which separated attributes into individual features such as 'Crime_Month' and 'Crime_Day' as opposed to 'Crime_Date', and 'data transformation' to rename spelling errors or similar group classes under different names. This process was necessary to improve the data quality and, in turn, increase the accuracy of the predictions made by the models. Despite employing this crucial step, their results were poor after training and testing the dataset on two ML algorithms, namely the Naive Bayes classifier and Decision Tree classifier, and trying to predict six different types of crime. The Decision Tree classifier only achieved an accuracy of 52.5%, and the Naive Bayes classifier only achieved an accuracy of 42.5%. This study aims to build on the findings of Almanie, Mirza and Lor by investigating the use of more than two machine learning models and hyperparameter tuning to improve the accuracy of crime prediction while using fewer classes.

Research conducted by Ch et al. (2020) aimed to help classify cybercrimes using machine learning techniques to help organisations and governments identify and reduce their impacts on society. Their methodology included using multiple ML algorithms, such as Naive Bayes and Random Forest, classifying three different cybercrimes (identity theft, copyright attack and hacking), using three different train-test split methodologies (50/50, 67/33 and 80/20), and training and testing the models on data containing eight features (incident, offender, victim, harm, year, location, age of the offender and cybercrime). Overall the models used performed well, yielding accuracies of 0.98 for Naive Bayes and 0.80 for Random Forest. These results, predicting only three classes, are higher than the previous results mentioned, which predicted six classes. This suggests that these machine learning algorithms perform better in multiclass classification problems by predicting fewer classes. By testing different train-test split methodologies, it was concluded in the study that the smaller the test set's size, the higher the accuracy was in the model's predictions. This resulted from more data in the training set, which reduced the bias on the predictions made by the model and overfitting, which occurs when the model becomes too complex and starts to memorise the training data instead of learning general patterns (OpenGenus IQ, 2023). However, this study used eight attributes in its data which is more than the previous study by Almanie, Mirza and Lor. Using many attributes can lead to the "curse of dimensionality". This means that most data points are lying far away from each other in high-dimensional space, which leads to sparsity and difficulty in finding meaningful patterns and relationships between features and the target class.

The importance of data preprocessing is further stated and supported by Safat, Asghar and Gillani (2021) in their research, which applied different machine learning models to fit crime data from Chicago and Los Angeles. The paper states, “The accumulated raw data from online repositories usually contains irrelevant information and errors. The overall data will likely have noise, inconsistencies, outliers, bungles, and missing qualities or, more fundamentally, data is inconsistent to start method. Therefore, selecting meaningful data is necessary to eliminate anomalies against outliers, noise, missing values, and other discrepancies. Thus, changing the unfeasible data into possible is manageable to accomplish information handling”. Safat, Asghar and Gillani emphasise the importance of data cleaning and preprocessing techniques to improve data quality and make it suitable for information handling, as well as challenges associated with handling raw data and the importance of effective data cleaning techniques in data preprocessing. This is evident in their results from the study, where every model performed well, with accuracies ranging from 88% to 94%. The findings from Safat, Asghar, and Gillani's study highlight the importance of data cleaning and preprocessing for accurate predictions, directly relevant to this proposed project on predicting crime. This project will use techniques such as feature selection to increase clarity in the data.

2.2 Machine Learning Algorithms

Iqbal et al. (2013) conducted a study to compare the Naive Bayes and Decision Tree classifiers in crime prediction. The dataset used to train and test the classifiers contains socio-economic data from the 1990 US Census, law enforcement data from the 1990 US LEMAS survey, and crime data from the 1995 FBI UCR. This dataset contains 128 attributes and 1994 instances, although, through feature selection, the number of attributes was reduced to twelve. The study aimed to compare how well the two classifiers could predict a crime category out of ‘Low’, ‘Medium’, and ‘High’. “For Decision Tree, the Accuracy, Precision and Recall are 83.9519%, 83.5% and 84%. On the other hand, Accuracy, Precision and Recall values for Naive Bayes are 70.8124%, 66.4% and 70.8%, respectively”. It can be concluded that the Decision Tree classifier performed better than the Naive Bayes classifier in each performance metric, meaning it is more suitable for crime prediction using multiclass classification. Although, Iqbal et al. used little data for each class for their project. There were only 1994 instances used in the data, which were split into one of the three class types. This means that there were a different number of instances within each class, causing class imbalance. As a result, there may be bias in the model as only the majority class will be predicted. In contrast, the minority classes will be ignored, leading to overfitting and poor generalisation. This can be seen in the confusion matrices produced for each ML model used, as the ‘Low’ class was predicted 1315 times, whereas ‘High’ and ‘Medium’ were only predicted 386 and 293 times, respectively. This project will address these issues by including more data to be used by the ML models and undersampling to reduce the class imbalance. However, to better compare ML algorithms, more should be used and evaluated to see if superior options can yield better results. Iqbal et al. mentioned this point for future study, as they stated they would incorporate more ML algorithms to evaluate and compare their prediction performance. This research project will address this by employing more than two ML algorithms to create a more diverse comparison.

Work conducted by Abba Babakura, Md. Nasir Sulaiman and Yusuf (2014) followed a very similar approach to Iqbal et al. by classifying crime categories as ‘Low’, ‘Medium’ and ‘High’ from data gathered from socio-economic data from the 1990 US Census, law enforcement data from 1990 US LEMAS survey, and the 1995 FBI UCR. However, instead of comparing the Naive Bayes classifier and the Decision Tree classifier, Abba Babakura, Md. Nasir Sulaiman and Yusuf compared the Naive Bayes classifier with the Artificial Neural Network (ANN) classifier. An Artificial Neural Network is loosely inspired by a human brain's structure and function, consisting of many interconnected nodes organised into layers. Each node receives inputs from the nodes in the previous layer, processes that input, and then sends outputs to the nodes in the next layer; This is a forward propagation. Backpropagation involves taking forward propagation's error rate (the difference between the predicted and real value) of forward propagation and feeding this loss backwards through the neural network layers to fine-tune the weights (Al-Masri, 2022). Even though backpropagation is a calculation step used within the ANN algorithm, ANN is often referred to as “Back Propagation” (BP) in their research paper. The Naive Bayes classifier outperformed the Artificial Neural Network classifier when comparing the metrics evaluating the models’ predictions. NB scored an accuracy, precision and recall of 0.90, 0.95 and 0.93, respectively, whereas ANN scored 0.65 for both accuracy and precision and a recall score of 1. Having a recall score of 1 means that ANN perfectly classified each observation of the ‘positive’ class into the correct class. Even though this is the case, the accuracy, which is the proportion of all instances that are correctly classified, regardless of their class, and precision, which measures the proportion of positive identifications that were actually correct, received much lower scores than that of NB. It can be concluded from this work that law-enforcing agencies can take great advantage, using machine learning algorithms like Naive Bayesian to fight crime effectively rather than using ANN. As a result, ANN will not be used to compare ML algorithms in predicting crime. To further state the need for comparisons of more models, Abba Babakura, Md. Nasir Sulaiman and Yusuf planned further to apply other classification algorithms to the crime dataset and evaluate their prediction performance.

Bhanumathi and Greeshma (n.d) used a crime dataset. They predicted the types of crimes in a particular area, which helped speed up the classification of criminal cases and proceed accordingly. This paper uses data from the past 18 years. The ML classifiers used in this comparative study are Decision Tree, Naive Bayes, K-Nearest Neighbours, and Random Forest. Only the accuracy was measured when comparing the models, and in the results, the Decision Tree, Naive Bayes and K-Nearest Neighbours classifiers all predicted crimes with a score of 0.841. Although, the Random Forest Classifier was the only model to score an accuracy of 0.96, implying classes are predicted correctly by the model 96% of the time. One issue with this study is the dataset being used. It is never clearly stated by Bhanumathi and Greeshma where the dataset was taken from, which could pose ethical issues surrounding the exact information included, which may be sensitive, and its use throughout the study. Another issue with this study that needs to be addressed is the lack of variety in performance metrics to gain a deeper insight into how well the algorithms perform. However, it can be said that all four of the algorithms performed well and should be used in further study. As well as this, Using four ML algorithms rather than two is positive. It can pose a diversity of results in future studies, even if it is not evident in Bhanumathi and Greeshma's research.

2.3 Performance Evaluations and Optimisations

In the research conducted by Safat, Asghar and Gillani (2021), which applied different machine learning models to fit crime data from Chicago and Los Angeles, a wide variety of evaluation metrics (or 'parameters', as they are referred to in the study) were used to measure the performance of the models. These included accuracy, precision, recall, and F1-score. Each metric successfully gave evidence of the success of the model's predictions, analysing different aspects using different mathematical algorithms. However, the class balance - the distribution of classes in a classification problem - was never noted. This may have skewed the results in the comparison and caused bias in the model's predictions. In this case, another metric called 'Balanced Accuracy' should have been used as "its use case is when dealing with imbalanced data" (neptune.ai, 2021).

Additional performance metrics are proposed by Tharwat (2021) when introducing detailed overviews of "the classification assessment measures to provide the basics of these measures and to show how it works to serve as a comprehensive source for interested researchers in this field". Such additions are confusion matrices, visual tools that house values used to calculate other metrics such as accuracy, precision, recall and F1-score. As the number of classes being predicted by an ML model increases, so does the column and row size of the confusion matrix, as the predicted classes occupy the columns. In contrast, the actual classes occupy the columns. From this, comparisons can be made about the model's prediction success as the numbers filling the grid represent the number of predictions; thus, more predictions should be made in a diagonal line along the cross-sections where the same predicted and actual classes meet. Another metric that is suggested to be used is the Area Under the ROC Curve (ROC AUC). The Receiver Operating Characteristics (ROC) curve balances the benefits, i.e., true positives, and costs, i.e., false positives. Comparing different classifiers in the ROC curve is difficult because no scalar value represents the expected performance. Therefore, the ROC AUC metric is used to calculate the area under the ROC curve, which tells us the degree or measure of separability or simply how much the model is capable of distinguishing between classes. In order to gain this result in a multiclass classification problem, a ‘One vs Rest’ approach needs to be taken. It compares each class against all the others simultaneously to convert a multiclass classification output into a binary classification output. Grandini, Bagli, and Visani (2020) also suggest using these metrics when evaluating multiclass models. However, they should be averaged using ‘macro’ averaging to give equal weight to each class. While these studies provide an excellent analysis of appropriate metrics for a multiclass classification problem, there is no comparison to suggest some are superior are provide more depth to a model’s performance. Therefore, these metrics will be used, evaluated and discussed in this research project to provide a clear view of the best model for predicting crime.

Cross-validation is a resampling technique that estimates a predictive model’s performance accuracy. In K-Fold cross-validation, once a dataset has been split into training and test sets, the training set is split further into a K number of equally sized groups called folds. The model is trained on K-1 folds and evaluated on the remaining fold for hyperparameter tuning. The importance of this step is shown in studies by Abba Babakura, Md. Nasir Sulaiman and Yusuf (2014) and Iqbal et al. (2013) used ten folds during K-fold cross-validation. While cross-validation helps to determine a more accurate estimate of model performance, using too many folds, such as 10 in the mentioned studies, can increase variance in performance estimates, making them more sensitive to specific subsets of the data rather than overall characteristics. This can result in overfitting, where the model performs well on subsets used in cross-validation but poorly on new data.

Yang and Shami (2020) state in their research exploring the best hyperparameters for data analysts and researchers, “to fit a machine learning model into different problems, its hyperparameters must be tuned. Selecting the best hyper-parameter configuration for machine learning models directly impacts the model’s performance”. Yang and Shami outline a list of hyperparameters that should be used on the ML algorithms in this project. Some suggestions include adjusting the hyperparameter ‘n_neighbor’ for the K-Nearest Neighbours classifier and ‘criterion’, ‘max_depth’, ‘min_samples_split’, ‘min_samples_leaf’, and ‘max_features’ for the Decision Tree classifier. Their paper also outlines the configuration space, which displays a range of options for testing a single hyperparameter. Once testing has been completed among every combination of hyperparameters, the best combination will be used by the model onwards. An example of this configuration space, or search space, is [10,100] for the ‘n_estimators’ hyperparameter in the Random Forest classifier. This information provides clear guidance on approaching hyperparameter tuning when using classification models.

Chapter 3

Requirements Analysis

3.1 Functional Requirements

Before any analysis or datasets are uploaded, the software's necessary libraries, such as taking inputs and outputs and using the necessary classifiers, must be imported. In order for the software artefact to obtain meaningful results, it first needs to read a cleaned dataset containing classes for new data to be classified into (either ‘Drug Violation’, ‘Motor Vehicle Accident Response’ (MVAR), or ‘Towed’), the Boston district where the crime occurred, the year ranging from 2015 to 2018, the month, the day of the week, and finally the hour ranging from 0 to 23. The next feature should be able to encode the categorical (non-numerical) features and classes through label encoding; otherwise, the following processes will not be able to process the data and perform calculations. Data analysis will then need to be performed on the dataset to gain insight into the distribution of features to help make more precise conclusions about how the classifiers performed and where improvements could be made.

The following function should split the dataset into target variables and feature variables. This step is necessary to train the classifier on the feature variables to predict the target variable. A feature to create a test-train split methodology should be implemented that allows a user to choose between a 70/30 split, an 80/20 split, or a 90/10 split.

A function that allows the user to choose between three types of runs should be built. These runs will be training and testing a default model classifier on the data, training and testing a model with k-fold cross-validation, and training and testing a model with hyperparameter tuning and with k-fold cross-validation applied. This ability is vital as it fills the gap in current research surrounding crime prediction using a machine learning approach, as comparisons between the three versions of runs have not been conducted before. The user can also choose within these functions between the classifiers being used. These include Naive Bayes, Decision Tree, Random Forest, and K-Nearest Neighbours.

Finally, to gain insight into the chosen model’s performance, a function should output the results of the metrics used to evaluate the predictions made. The metrics included will be accuracy, balanced accuracy, precision, recall, F1-score, ROC AUC score, and confusion matrix.

3.2 Non-Functional Requirements

For the software artefact to work the first time, the functions mentioned in the Functional Requirements section should be run in order. Otherwise, the environment will not be set up correctly, and the data cannot be processed correctly. The loading time of the dataset should never take longer than a minute but will never be instant, as the dataset includes 34,063 inputs. The steps that require training and testing the models on the data are also not instant. As k-fold cross-validation and hyperparameter tuning are implemented, the processing time increases. Although, the time taken should not exceed 7 minutes. To aid the user's understanding of the processes, comments and descriptions will be present to give detail into how the software's stages operate.

Chapter 4

Design & Methodology

4.1 Project Plan

A breakdown of the project plan can be seen in Figure 1, which represents the project stages in a Gantt chart.

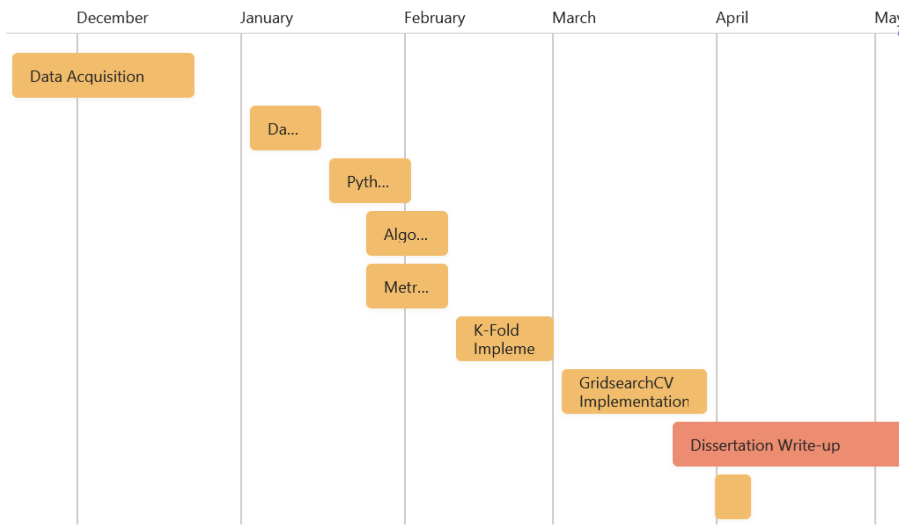


Figure 1: A Gantt chart representing the project plan.

There are nine steps included that span from 18th November 2022 to 11th May 2023. The first step is 'Data Acquisition', planned between 18th November to 23rd December 2022 and requires finding a suitable dataset with the necessary features to predict crime, such as the location, crime, and date from a public source. The second step, 'Dataset Cleaning,' includes removing the null values from the dataset and feature selection to only allow for necessary data to be used by the models. It was planned to run from 2nd January 2023 to 16th January 2023. 'Python Notebook Creation & Organisation' is the third step, running from 17th January 2023 to 2nd February 2023, and requires the setup of the environment to run the artefact and planning of the layout to allow for ease of use and understanding. The next step, planned to be completed from 24th January 2023 to 9th February 2023, is 'Algorithm Selection & Implementation', which meant researching the best machine learning algorithms to be used for the multiclass classification problem from academic sources, such as Bhanumathi and Greeshma (n.d) and Iqbal et al. (2013). 'Metric Selection & Implementation' was planned for the same start and completion dates and required research on the best performance metrics to measure the model's success in predicting crime. 'K-Fold Implementation', which needed k-fold cross-validation to be applied to the software, was planned to run from 10th February 2023 to 1st March 2023. The implementation of hyperparameter tuning through GridserachCV, named 'GridsearchCV Implementation' in the Gantt chart, was planned to run from 2nd March 2023 to 30th March 2023. The 'Testing and Results' stage was planned to run from the 31st of March to the 7th of April 2023, gathering data, including the data distribution and the model's performance. Finally, the 'Dissertation Write-up' was planned to be completed between 23rd March 2023 and 11th May 2023.

Fortunately, these steps were completed within the given timeframe, with very few issues resulting in a holdup. This was due to consistent project engagement and management, with tasks completed weekly, either through research, design or development. Meetings with the project supervisor, Dr. Vassilis Cutsuridis, were fundamental to the project's progress. Held initially on a weekly and then fortnightly basis and lasting between fifteen to twenty minutes, progress was discussed, and guidance was given to steer the research in the correct direction. Even though direct solutions were never given, clues and ideas were to enable gaining knowledge through experience and research. Evidence of project meetings that were held on Microsoft Teams can be seen in Figure 2.

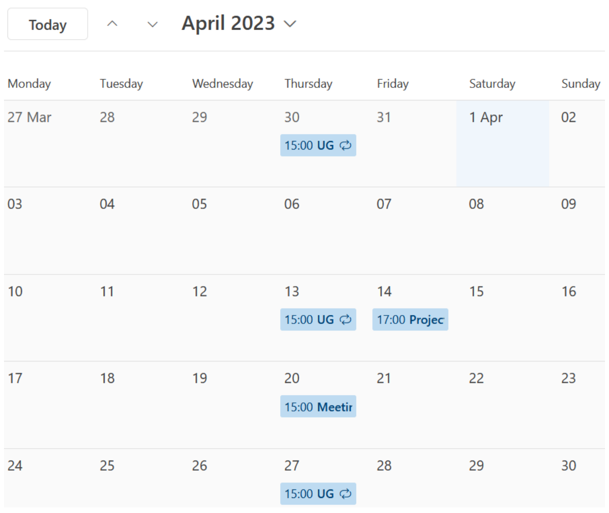


Figure 2: A screenshot of project meetings with the supervisor in April.

4.2 Risk Analysis

Table 1 shows a risk analysis matrix discussing the potential issues in the research project, their likelihoods, the impact they could have, and the mitigation strategy to resolve them.

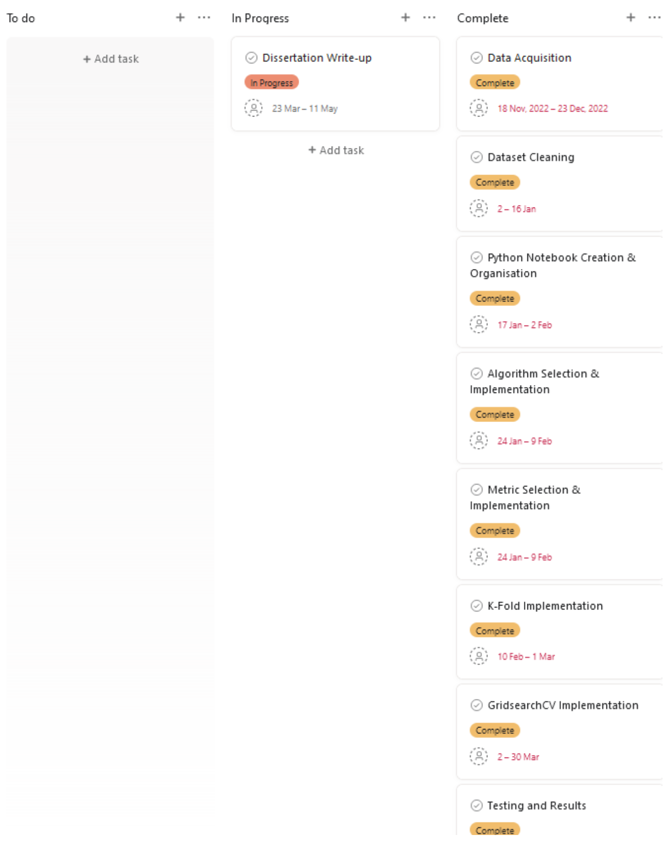
Risk	***Likelihood***	***Impact***	***Mitigation***
Data Quality Issues	High	Inaccurate or incomplete data can significantly affect the performance of the algorithms, leading to incorrect predictions	Clean the dataset through the removal of null values, data reduction, data transformation and feature selection to

			ensure that the data is of sufficient quality
Overfitting	Medium	A common risk in machine learning which occurs when the model fits the noise in the data rather than the underlying patterns	Use K-fold cross-validation by testing the model on different subsets of the data and comparing its performance on each subset
Hyperparameter Tuning Issues	Medium	Finding the optimal values can be a time-consuming and challenging task	Conduct research which lists the best hyperparameters and why they should be used
Security and Privacy Issues	Low	When working with sensitive data like crime data, security and privacy are critical concerns	Do not gather private data, use data from a trusted, public source containing no personal or sensitive information. Ensure the results are not misused by clearly explaining the contents, purpose, and meaning

Table 1: A risk analysis matrix showing issues, likelihoods, impacts and mitigations.

4.3 Software Development Methodology

To manage the project, Kanban was used to visualise the status of the tasks that were required to complete the research project. Kanban is a form of agile development, an iterative software development methodology that puts no pressure on delivering short-term deliverables, such as when using the SCRUM method. Tasks are labelled by status - either ‘To do’, ‘In Progress’ or ‘Complete’ - and move across the board to the next section depending on their progression. The Kanban board used for this project can be seen in Figure 3. The stages used are the same as for the Gantt chart in Figure 1.



To do

+ Add task

In Progress

Dissertation Write-up
In Progress
23 Mar – 11 May

+ Add task

Complete

Data Acquisition
Complete
18 Nov, 2022 – 23 Dec, 2022

Dataset Cleaning
Complete
2 – 16 Jan

Python Notebook Creation & Organisation
Complete
17 Jan – 2 Feb

Algorithm Selection & Implementation
Complete
24 Jan – 9 Feb

Metric Selection & Implementation
Complete
24 Jan – 9 Feb

K-Fold Implementation
Complete
10 Feb – 1 Mar

GridsearchCV Implementation
Complete
2 – 30 Mar

Testing and Results
Complete

Figure 3: Kanban board to track the progress of the project.

Kanban was chosen as the software development methodology because it is not only a tool to help visualise progress, but it helps a user not take on too much work at once and become overwhelmed. Kanban also allows for flexibility and reduces time-constrained pressures that may arise from non-agile development approaches.

4.4 Toolsets & Machine Environments

The software used for this project was Google Colaboratory (Google Colab), an online platform allowing users to write and run code in a Jupyter notebook-style environment. It is hosted on Google Cloud infrastructure, allowing users to execute Python code. Using a Jupyter notebook-style environment allows users to execute code snippets, view the results, and document the analysis process simultaneously. It also allows users to share their entire analysis process in a single document, including code, data, visualisations, and commentary, which makes it easy for others to reproduce work, modify analysis, and build upon findings (Sherrer, 2022).

One advantage of using Google Colab over Jupyter Notebook is its accessibility. As Google Colab is a cloud-based platform, allowing it to be accessed by users as long as they have an internet connection and a web browser; for this project, Mozilla Firefox was used to access the Google Colab environment. There is also a lack of cost with Google Colab over Jupyter Notebook as it is a free survive provided by Google and requires no installation of a local development environment. Therefore, this work can be easily reproduced and developed further by future Computer Scientists or Data Analysts. As well as these points, Google Colab allows free access to powerful hardware such as GPUs, which can significantly speed up the training of machine learning models.

Multiple libraries are used within the Google Colab environment to access the functionality required to use and manipulate the dataset and the chosen machine learning algorithms and metrics. Pandas is a software library used for data manipulation and analysis in Python. It reads the data as a Pandas data frame, allowing for quick analysis and manipulation, essential for splitting data into feature and label sets. In order to access pre-made machine learning algorithms and performance metrics, the scikit-learn (sklearn) library was utilised. This definition of machine learning algorithms allows users to compare their performance and use tools to preprocess data, such as creating train and test sets.

4.5 Dataset

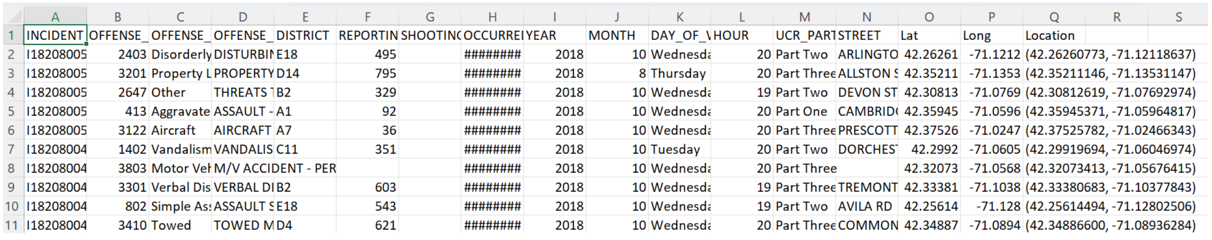
4.5.1 Acquisition

The dataset used for this research project needed to include features suitable for predicting crime, such as multiple crimes, the location within a larger area, the date the crime was recorded and the time. Many studies have used these features, including Almanie, Mirza and Lor (2015) and Abba Babakura, Md. Nasir Sulaiman and Yusuf (2014) classify crimes or crime types based on these attributes. It was necessary to acquire the public domain dataset to reduce the risk of handling personal or sensitive information. The dataset was acquired from Kaggle, an online community that allows users to find datasets they want to use in building AI models, publish datasets, and work with other data scientists and machine learning engineers (www.kaggle.com, n.d.). The specific dataset used for this project includes crimes in Boston, USA, from 2015 to 2018 and "contains records from the new crime incident report system, which includes a reduced set of fields focused on capturing the type of incident as well as when and where it occurred" (Jain, n.d.). This dataset has 2,60,760 rows and 17 columns, which include the following:

- INCIDENT_NUMBER: e.g. I182080058
- OFFENSE_CODE: e.g. 2403
- OFFENSE_CODE_GROUP: e.g. Property Lost
- OFFENSE_DESCRIPTION: e.g. PROPERTY - LOST
- DISTRICT: e.g. D14
- REPORTING_AREA: e.g. 795
- SHOOTING: This field is blank
- OCCURRED_ON_DATE: e.g. 30/08/2018 20:00:00
- YEAR: e.g. 2018
- MONTH: e.g. 8
- DAY_OF_WEEK: e.g. Monday
- HOUR: e.g. 20
- UCR_PART: e.g. Part Three

- STREET: e.g. ALLSTON ST
- LATITUDE: e.g. 42.26260773
- LONGITUDE: e.g. -71.12118637
- LOCATION: e.g. (42.26260773, -71.12118637)

A screenshot of the dataset in Microsoft Excel can be seen in Figure 4.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	INCIDENT	OFFENSE_	OFFENSE_	OFFENSE_	DISTRICT	REPORTIN	SHOOTINC	OCCURREI	YEAR	MONTH	DAY_OF_\	HOUR	UCR_PART	STREET	Lat	Long	Location		
2	I18208005	2403	Disorderly	DISTURBII	E18	495		########	2018	10	Wednesda		20	Part Two	ARLINGTO	42.26261	-71.1212	(42.26260773, -71.12118637)	
3	I18208005	3201	Property L	PROPERTY	D14	795		########	2018	8	Thursday		20	Part Three	ALLSTON S	42.35211	-71.1353	(42.35211146, -71.13531147)	
4	I18208005	2647	Other	THREATS T	B2	329		########	2018	10	Wednesda		19	Part Two	DEVON ST	42.30813	-71.0769	(42.30812619, -71.07692974)	
5	I18208005	413	Aggravate	ASSAULT -	A1	92		########	2018	10	Wednesda		20	Part One	CAMBRID(	42.35945	-71.0596	(42.35945371, -71.05964817)	
6	I18208005	3122	Aircraft	AIRCRAFT	A7	36		########	2018	10	Wednesda		20	Part Three	PRESCOTT	42.37526	-71.0247	(42.37525782, -71.02466343)	
7	I18208004	1402	Vandalism	VANDALIS	C11	351		########	2018	10	Tuesday		20	Part Two	DORCHEST	42.2992	-71.0605	(42.29919694, -71.06046974)	
8	I18208004	3803	Motor Veh	M/V ACCIDENT - PER				########	2018	10	Wednesda		20	Part Three		42.32073	-71.0568	(42.32073413, -71.05676415)	
9	I18208004	3301	Verbal Dis	VERBAL DI	B2	603		########	2018	10	Wednesda		19	Part Three	TREMONT	42.33381	-71.1038	(42.33380683, -71.10377843)	
10	I18208004	802	Simple Ass	ASSAULT S	E18	543		########	2018	10	Wednesda		19	Part Two	AVILA RD	42.25614	-71.128	(42.25614494, -71.12802506)	
11	I18208004	3410	Towed	TOWED M	D4	621		########	2018	10	Wednesda		20	Part Three	COMMON	42.34887	-71.0894	(42.34886600, -71.08936284)	

Figure 4: A screenshot of the dataset acquired from Kaggle, including Boston crime.

Within this dataset, many attributes can be made redundant to reduce the size and complexity, which will help make the classification process more accessible and more accurate. Therefore, many inputs and attributes must be removed to increase the dataset's clarity.

4.5.2 Cleaning

The inputs in the column named 'OFFENSE_CODE_GROUP' will be used as the classes for the machine learning models to predict. However, there are sixty-six different instances within the column. Comparing the results of research conducted by Ch et al. (2020), where only three classes were predicted, and Almanie, Mirza and Lor (2015), where six classes were predicted, it can be concluded that predicting fewer classes in multiclass classification is more optimal and provides better results. As a result, sixty-three classes had to be removed from the dataset manually. The remaining ten classes were chosen by creating a separate Python notebook in Google Colab - called 'DataCleaner.ipynb' - to count the fifteen crimes with the most instances in the dataset. Of these fifteen classes, some had vague names, such as 'Other', or had far fewer instances compared to other classes, which would have led to high levels of class imbalance. The number of classes was dropped to ten.

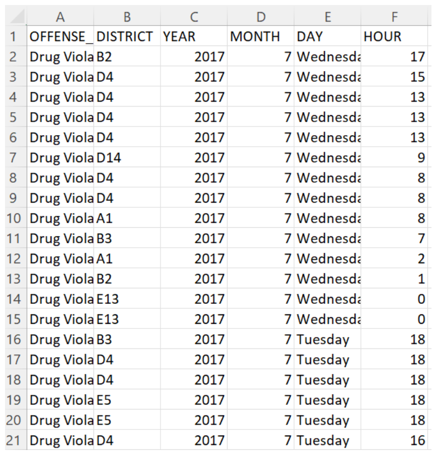
Also, within 'DataCleaner.ipynb', the new dataset with ten classes was analysed to count the number of inputs that contained 'null' or empty values. Removing these erroneous inputs helps to reduce the adverse effect on the performance and accuracy of any machine learning algorithm. After evidence of null values, the rows containing the values were removed or 'dropped' from the Pandas data frame.

After manipulating the data frame to be listed by the crimes, it was clear that some naming was incorrect; instead of 'Investigate Person', four inputs in the data were labelled 'INVESTIGATE PERSON'. Altering this mistake was necessary to maintain validity, so data transformation was undertaken to rename the four outlying instances back to 'Investigate Person'.

Feature selection was the following technique used to improve the data and increase the prediction power of the algorithms by selecting the most critical variables and eliminating the irrelevant ones. Eleven columns were removed, leaving six critical features: OFFENSE_CODE_GROUP, DISTRICT, YEAR, MONTH, DAY_OF_WEEK, and HOUR. The other attributes were not used because they grouped these features, such as OCCURED_ON_DATE grouping YEAR, MONTH, OCCURED_ON_DAY, and HOUR, or they included unimportant information like UCR_PART. The reason the district was chosen over latitude and longitude was that there were fewer but many varieties of districts in the data. In contrast, latitude and longitude represent geographical coordinates with countless variations. This would make it impossible to predict a crime from a number that serves no resemblance with another in a classification problem. This stage was completed manually through the Microsoft Excel interface, as using a visual tool allows for more clarity and certainty compared to Python code. This stage left the dataset with discrete data as it can be easily counted and analysed, as well as necessary for classification rather than regression. The column names were altered too. OFFENSE_CODE_GROUP became 'OFFENSE_GROUP', and DAY_OF_WEEK became 'DAY' to reduce clutter and increase simplicity.

In order to reduce the issue of class imbalance, undersampling was employed to equalise the number of instances in every other class with the class with the least inputs. After counting the occurrences of each class within the data frame, the 'Investigate property' class had the least inputs, with 11354. The differences between the number of inputs in other classes had to be calculated and eliminated to reach 11354. This process of data reduction was also conducted manually through Microsoft Excel.

After testing was conducted further in the project, it was found that the results scored using ten classes in multiclass classification were too low, proving that the classifiers could not successfully predict the correct class. This meant that the confusion matrix produced had to be examined to determine which three classes were the easiest for the classifiers to distinguish by comparing how often they were correctly predicted. The CSV file named 'crimeNoErrorEqual_3.csv' containing the dataset can be seen in Figure 5.



	A	B	C	D	E	F
1	OFFENSE_	DISTRICT	YEAR	MONTH	DAY	HOUR
2	Drug Viola	B2	2017	7	Wednesda	17
3	Drug Viola	D4	2017	7	Wednesda	15
4	Drug Viola	D4	2017	7	Wednesda	13
5	Drug Viola	D4	2017	7	Wednesda	13
6	Drug Viola	D4	2017	7	Wednesda	13
7	Drug Viola	D14	2017	7	Wednesda	9
8	Drug Viola	D4	2017	7	Wednesda	8
9	Drug Viola	D4	2017	7	Wednesda	8
10	Drug Viola	A1	2017	7	Wednesda	8
11	Drug Viola	B3	2017	7	Wednesda	7
12	Drug Viola	A1	2017	7	Wednesda	2
13	Drug Viola	B2	2017	7	Wednesda	1
14	Drug Viola	E13	2017	7	Wednesda	0
15	Drug Viola	E13	2017	7	Wednesda	0
16	Drug Viola	B3	2017	7	Tuesday	18
17	Drug Viola	D4	2017	7	Tuesday	18
18	Drug Viola	D4	2017	7	Tuesday	18
19	Drug Viola	E5	2017	7	Tuesday	18
20	Drug Viola	E5	2017	7	Tuesday	18
21	Drug Viola	D4	2017	7	Tuesday	16

Figure 5: A screenshot of the final, complete dataset called 'crimeNoErrorEqual_3.csv'.

The three classes the chosen machine learning algorithms will predict are ‘Drug Violation’, ‘Motor Vehicle Accident Response’, and ‘Towed’.

4.6 Algorithm Selection & Design

The following machine learning algorithms were selected based on prior research recommendations and findings, including Iqbal et al. (2013), Abba Babakura, Md. Nasir Sulaiman and Yusuf (2014), and Bhanumathi and Greeshma (n.d). They were implemented into the software artefact with the help of the algorithms provided by the sklearn library.

4.6.1 Naive Bayes Classifier

Naive Bayes is a machine learning algorithm based on the Bayes theorem; it calculates the probability of each class given the input features, and the class with the highest probability is the predicted output. The term “Naive” refers to the fact that it assumes all features in the dataset are independent of each other, which is not always the case. The Bayes theorem is expressed below (1):

$$P(A|B) = P(B|A) * P(A) / P(B) \tag{1}$$

When broken down, P(A|B) is the posterior probability of event A given evidence B, P(B|A) is the likelihood of observing evidence B given event A, P(A) is the prior probability of event A and P(B) is the marginal likelihood, often referred to as model evidence.

4.6.2 Decision Tree Classifier

A decision tree is a supervised machine learning algorithm that forms a tree-like structure representing a series of decisions and their possible consequences. It has a hierarchical structure, which consists of a root node (the beginning of a decision tree), branches (subsection of the entire tree), parent nodes (a node which is divided into sub-nodes), child nodes (sub-nodes of parent nodes) and leaf nodes (nodes that do not split). The algorithm recursively splits the data into subsets based on the most significant feature at each tree node. An example can be seen in Figure 6.

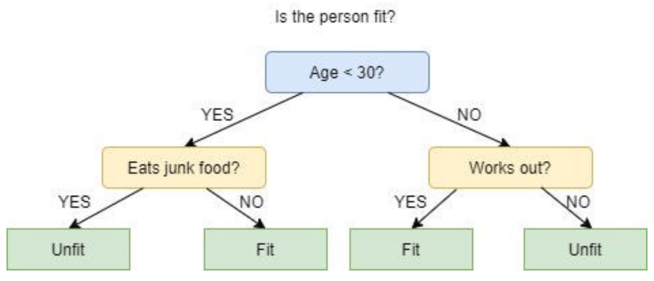


Figure 6: A decision tree example taken from Sakkaf (2020).

A decision tree is a flow chart to help visualise a decision-making process and map out different courses of action and their potential categorical outputs (classes).

4.6.3 Random Forest Classifier

A random forest classifier is a machine learning algorithm that trains several decision tree classifiers on various subsets of the dataset and employs averaging to improve predictive accuracy while mitigating the risk of overfitting. While a decision tree combines a series of decisions, a random forest combines multiple decision trees. As a result, the random forest approach may require more time due to its complexity, but it can result in more precise predictions. An example can be seen in Figure 7.

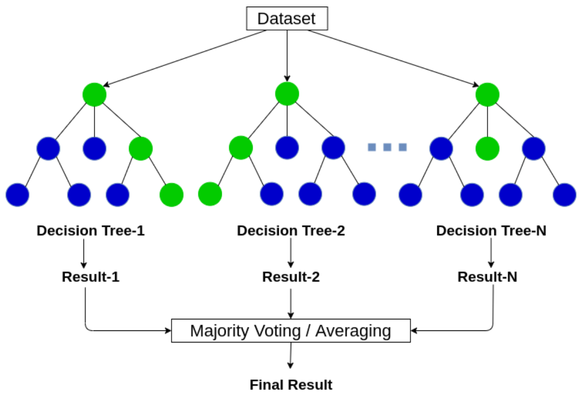


Figure 7: An example of a decision tree from Analytics Vidhya (2021).

4.6.4 K-Nearest Neighbours Classifier

In the K-nearest neighbours (KNN) algorithm, classifications are made by considering the proximity of an individual data point to others in the dataset, assuming that similar points tend to be located near each other. The concept of similarity, also known as distance, proximity, or closeness, is central to KNN, which calculates the distance between points on a graph. While there are various methods for computing distance, the Euclidean distance, measured as the straight-line distance between two points, is commonly used and is the default in the sklearn implementation of KNN when the power parameter P is set to 2. The 'K' in K-nearest neighbours represents the number of samples closest to the data point, calculated by the Euclidian distance. The majority class represented in this sample is the class given to the original data point. An example of the K-nearest neighbours classifier can be seen in Figure 8.

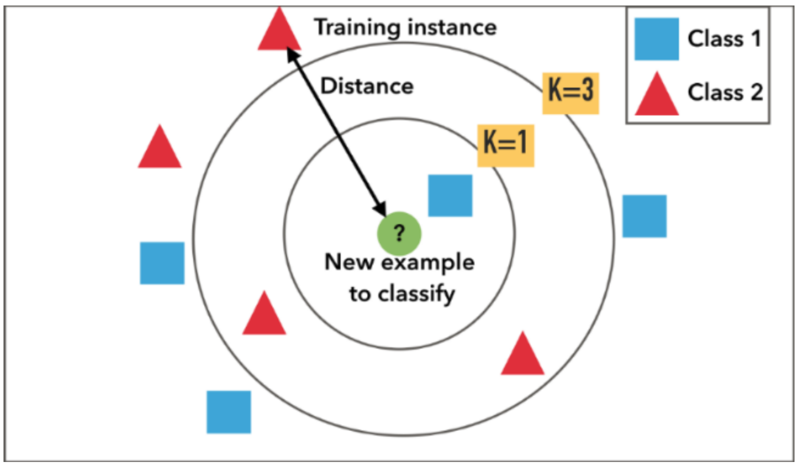


Figure 8: An example of the KNN classifier taken from Maheshwari (2019).

The Euclidian distance can be calculated by using the following formula (2):

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (2)$$

In the formula, (x_1, y_1) are the coordinates of the first point, (x_2, y_2) are the coordinates of the second point, and d is the distance between (x_1, y_1) and (x_2, y_2). The function provided by sklearn - 'KNeighborsClassifier' - uses K=5 as default.

4.7 Train-Test Split Methodology

In a classification problem, the train-test split methodology is utilised to assess the performance of a machine learning algorithm. This involves dividing a dataset into two subsets: training and test sets. The training set is used to train or fit the model, while the test set assesses the quality of the machine learning model's fit. To achieve this, the feature set, which includes the district, year, month, day, and hour, is fed into the model, and the predicted class is compared to the actual class from the label set. The primary objective is determining the model's effectiveness in predicting new data.

This is how the model is expected to be used in practice. It is trained on available data with known inputs and outputs using supervised learning and then used to predict new examples where the expected output or target values are unknown.

In this research, a train-test split methodology of 80/20 will be used. This means 80% of the dataset will be used as the training set, and the remaining 20% will be used as the test set. The decision to use a smaller test set was based on findings from Ch et al.'s (2020) research, which revealed a positive correlation between the test set's size and the accuracy of the model's predictions. The study showed that having more data in the training set reduced bias and overfitting, which can happen when a model becomes too complex and memorises the training data instead of learning general patterns. Joseph (2022) also suggests using a smaller test set, namely the 80/20 split methodology, “A commonly used ratio is 80:20 [...]. Other ratios, such as 70:30, 60:40, and even 50:50, are also used in practice. There is no clear guidance on the best or optimal ratio for a given dataset. The 80:20 split draws its justification from the well-known Pareto principle, but that is again just a thumb rule used by practitioners”. Sklearn provides a function called ‘train_test_split’, which splits arrays or matrices into random and test subsets.

4.8 K-Fold Cross-Validation

Cross-validation is a popular methodology employed in machine learning to evaluate the performance of models when data is limited. Its primary objective is to predict how well the model will perform when predicting new, unseen data. The process involves dividing the data into k groups or 'folds,' Each fold is used to assess the model trained on the remaining k-1 folds. This process is repeated k times, with each fold being tested once. By averaging the results of these iterations, a more reliable estimate of the model's performance can be obtained, reducing overfitting compared to a simple train-test split. This technique is widely used and referred to as k-fold cross-validation, providing a more objective and realistic assessment of the model's ability (pramod, 2023).

The k-fold cross-validation procedure follows a specific sequence of steps. First, the training set is randomly shuffled and divided into k groups. For each group, it is used as the validation set, while the remaining groups are used as the training set. Then, a model is trained on the training set and evaluated on the validation set, retaining the evaluation score while discarding the model. This process is repeated for every fold in the dataset. Finally, the model's performance is summarised using the evaluation scores, which helps to identify the optimal parameters for the model to be tested on by the test set. A visual representation of this process can be seen in Figure 8. Using this technique, it is possible to obtain a more reliable estimate of the model's performance while reducing the risk of overfitting. This is why k-fold cross-validation is a widely-used methodology.

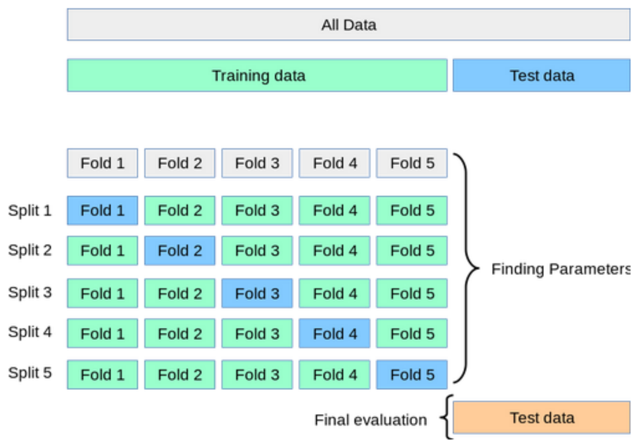


Figure 8: The k-fold cross-validation procedure visualised, taken from SciKit-Learn (2009).

After examining the results and discussions from studies conducted by Abba Babakura, Md. Nasir Sulaiman and Yusuf (2014) and Iqbal et al. (2013), the value of K chosen for this research project is five. Although cross-validation helps obtain a more accurate estimate of model performance, using too many folds, such as 10 in the mentioned studies, can increase variance in performance estimates, causing them to be more sensitive to specific subsets of data rather than overall characteristics. This can lead to overfitting, where the model performs well on the subsets used in cross-validation but poorly on new data. From now, k-fold cross-validation will be referred to as either ‘5-fold cross-validation’ or ‘5-fold’.

4.9 Hyperparameter Tuning

Hyperparameters control the learning process and determine the values of model parameters (internal variables of a model) learned by a machine learning algorithm. While hyperparameters are not part of the resulting model, they significantly influence its performance (Brownlee, 2017). Hyperparameter tuning tests different hyperparameter configurations when training a model to give optimised values for hyperparameters, maximising a model's predictive accuracy (Google Cloud, n.d.). Sklearn's 'GridSearchCV' function can be employed to implement grid search, which identifies the optimal hyperparameter values from a predefined set of hyperparameters in a grid. By iterating through the predefined hyperparameters and fitting a model on the training set, this function enables the selection of the best hyperparameters from the list (Mujtaba, 2020). Using GridSearchCV gives the ability to use 5-fold cross-validation. Using 5-fold cross-validation and grid search makes it possible to have more meaningful results compared to an ordinary train-test split with minimal tuning (Beheshti, 2022).

The hyperparameters to be tuned for the Random Forest and K-Nearsest Neighbours classifiers were taken from research by Yang and Shami (2020). These are ‘n_neighbor’ for the K-Nearest Neighbours classifier and ‘criterion’, ‘max_depth’, ‘min_samples_split’, ‘min_samples_leaf’, and ‘max_features’ for the Random Forest classifier. The Decision Tree classifier uses the same hyperparameters to tune as Random Forest, apart from ‘n_estimators’, as this hyperparameter determines the number of trees to use for Random Forest. For the Naive Bayes classifier, ‘var_smoothing’ will be tuned, which was suggested by coderzcolumn.com (n.d.). This value accepts a float specifying a portion of the most significant variance of all the features added to variance for smoothing.

4.10 Performance Metrics

The metrics used in this research project were a result of conclusions made from previous studies conducted by Safat, Asghar and Gillani (2021) and Tharwat (2021). These metrics are accuracy, balanced accuracy, precision, recall, F1-score, ROC AUC score and confusion matrix. These metrics are provided by sklearn and help to portray a model's prediction performance. A metric score of 0.7 (70%) is considered 'good' or 'acceptable' for a model when considering accuracy, balanced accuracy, precision, recall and F1-score. A model with a ROC AUC score of 0.5 (50%) is no better than a model that performs random guessing (Google Developers, n.d.). Therefore, 0.7 to 0.8 is considered acceptable. Any scores above 0.8 are considered excellent, but a model which receives a score of 0.9 or above is considered outstanding. The metrics precision, recall, and F1-score should be averaged using 'macro' averaging to give equal weight to each class in a multiclass classification setting.

4.10.1 Confusion Matrix

A confusion matrix presents a summary of predictions in matrix format, indicating the quantity of correct and incorrect predictions for each class. It assists in identifying the classes that the model is confusing with other classes, allowing for a better understanding of its performance (Sciencedirect.com, 2019). In the matrix, the predicted classes occupy the columns, and the rows represent the actual classes. The individual spaces in the matrix represent four values in a binary classification confusion matrix. These are true positive (TP), false positive (FP), true negative (TN), and false negative (FN). True positive refers to the number of correctly predicted classes, while false positive refers to the number of incorrectly predicted classes. True negative represents the number of correctly predicted no-event classes, while false negative represents the number of incorrectly predicted no-event classes. An example can be seen in Figure 9.

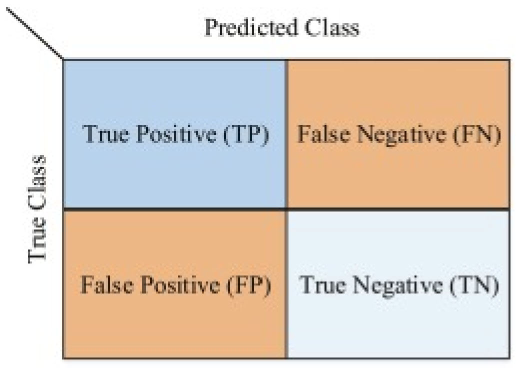


Figure 9: An example confusion matrix taken from Sciencedirect.com (2019).

In multiclass confusion matrices, the matrix is not 2x2 but corresponds to the number of classes in the dataset. If a model is successful, more predictions should fall in a diagonal line along the cross-sections where the same predicted and actual classes meet.

4.10.2 Accuracy

Accuracy is the fraction of correctly classified instances from all predictions made by the classifier. The Accuracy formula considers the total number of True Positive and True

Negative values in the numerator and the sum of all the entries in the confusion matrix in the denominator (Grandini, Bagli and Visani, 2020) (3).

$$Accuracy = \frac{TP + TN}{TP+TN+FP+FN} \tag{3}$$

4.10.3 Precision & Macro Precision

Precision measures the proportion of correct positive predictions made by the model out of all positive predictions. It is calculated by dividing the number of true positive predictions by the total number of positive predictions, and it quantifies the accuracy of the model's positive predictions (neptune.ai, 2021) (4).

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Macro\ Precision = \frac{Precision_1 + Precision_2 \ldots + Precision_n}{n} \tag{5}$$

The macro precision is calculated by adding the precision values for each class and dividing it by the number of classes. For this project, the value n will be three (5).

4.10.4 Recall & Macro Recall

Recall is the fraction of true positives among all the correct events (Pahwa, 2017). It is calculated by dividing the true positives by the true positives and false negatives (6). The macro recall is calculated the same way as macro precision but with the recall values substituted for the precision values (7).

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

$$Macro\ Recall = \frac{Recall_1 + Recall_2 \ldots + Recall_n}{n} \tag{7}$$

4.10.5 F1-Score & Macro F1-Score

F1-score is the harmonic mean of precision and recall and is often used when the distribution of classes is uneven (neptune.ai, 2021). The formula for the F1-score is below (8), with the formula for the macro F1-score beneath (9).

$$F1\ score\ = 2 \times (\frac{Precision \times Recall}{Precision + Recall}) \tag{8}$$

$$Macro\ F1\ score\ = \frac{F1\ score_1 + F1\ score_2 \ldots + F1\ score_n}{n} \tag{9}$$

4.10.6 Balanced Accuracy

Balanced accuracy deals with imbalanced datasets and defines the mean of sensitivity and specificity (Scikit-learn.org, 2010) (10). Sensitivity is the True Positive Rate (TPR) or recall. Specificity (the True Negative Rate (TNR)) calculates the ratio of accurately identified negatives to the total number of negative predictions the model could make (11).

$$Balanced\ Accuracy\ = \frac{Sensitivity + Specificity}{2} \tag{10}$$

$$Specificity\ = \frac{TN}{TN + FP} \tag{11}$$

4.10.7 ROC AUC Score

The ROC AUC score is a helpful tool for evaluating the trade-off between true positive rates and false positive rates ((FPR) the total number of false positives divided by the sum of false positives and true negatives (12)) of a predictive model. It can be viewed using the ROC curve, which shows how well the model can distinguish between the two classes. The score is obtained by measuring the area beneath the curve. An example ROC curve can be seen in Figure 10. A 'One vs Rest' approach is required to use the ROC AUC score in a multiclass classification setting, which involves comparing each class against all the others simultaneously to transform a multiclass output into a binary output.

$$False\ Positive\ Rate\ = \frac{FP}{TN + FP} \tag{12}$$

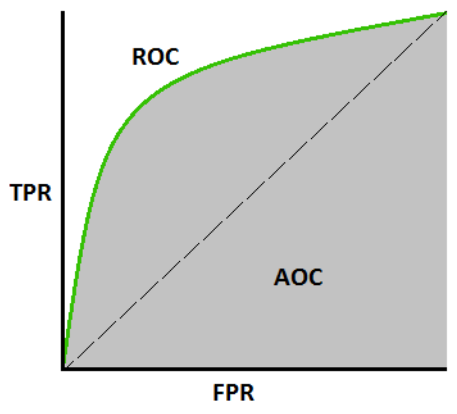


Figure 10: An example ROC Curve taken from Narkhede (2018)

4.11 Design

The design of the software artefact follows an ordered structure of code blocks that should be processed in order for the setup and classifiers to work. Each step is in a dropdown section with code blocks or other dropdown sections with Python code. There are five critical steps in the software, which are listed as follows:

- Import necessary libraries
- Load the dataset, calculate the distribution within the data and encode the categorical values
- Set the label and feature sets from the data frame
- Create a train-test split
- Choose a model within the three types of runs

In the first section, all the required libraries need to be imported to the Google Colab notebook, such as 'pandas', the machine learning classifiers from sklearn and 'seaborn' which provides a colour-coded matrix for the confusion matrix. In the second section, the dataset must be uploaded from the computer's files and stored as a Pandas data frame. Information about the data distribution can be calculated and displayed to the user to give insight into the occurrence of each value across the data frame. As well as this, the categorical features - the crime, district and day - need to be encoded into numerical values to be understood and processed by sklearn's functions. This is accomplished by ordering each instance per feature and converting it into a number, starting from zero. As a result, the classes, for instance, are converted from 'Drug violation', 'Motor Vehicle Accident Response' and 'Towed' to '0', '1' and '2', respectively. The third section then splits the data frame into a label set containing the classes and a feature set containing the features. Notably, the order of these values corresponds to each other to have the features and classes match.

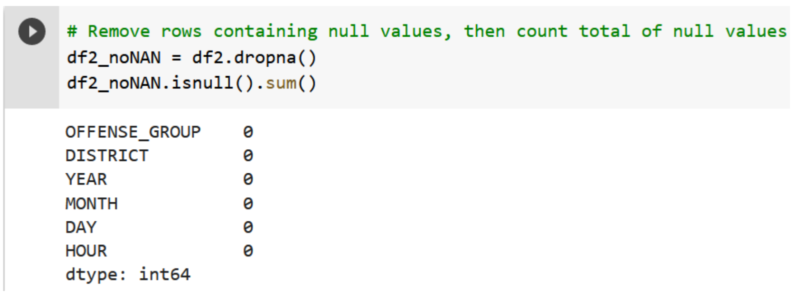
The penultimate step allows users to choose between a 70/30 train-test split, an 80/20 train-test split, or a 90/10 train-test split. However, for this research project, only the 80/20 split will be used to make predictions and gather results and conclusions. The final step involves three more dropdown blocks. Each block contains a different version of how the classifiers will be processed. The first version, or ‘run’, will be called the ‘Standard run’. This is a simple process where each of the four ML classifiers is fitted to the training data and tested on the test data. The second version is the ‘5-Fold Cross-Validation run’, which implements 5-fold into a process similar to the standard run. The final version is named ‘Hyperparameter Tuning & 5-Fold Cross-Validation run’(HPT run), which implements HPT with 5-fold cross-validation for the desired ML model; the best hyperparameter pairings are also printed to the user. A ‘Model performance’ code block is present for each run and outputs the scores for each performance metric. From this, a discussion can be made about the best model and run method.

Chapter 5

Implementation

5.1 DataCleaner.ipynb

The Google Colab notebook ‘DataCleaner.ipynb’ provides code that counts the instances of each class used in the data cleaning and reduction stages. Once this information has helped reduce the number of crimes in the dataset, the new dataset is read into the notebook and converted to a Pandas data frame. There are 1105 instances of null values in the district column. The rows containing the null values were removed through the following code snippet in Figure 11.

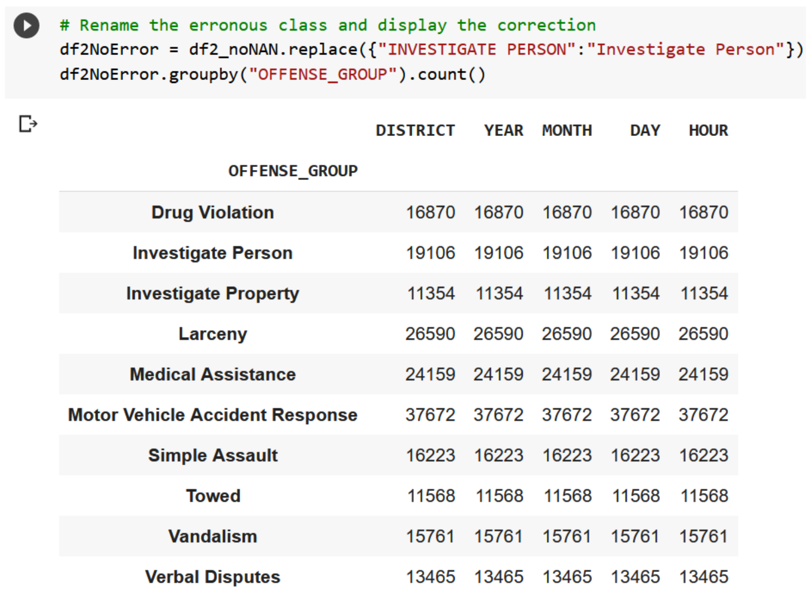


```
# Remove rows containing null values, then count total of null values
df2_noNAN = df2.dropna()
df2_noNAN.isnull().sum()
```

```
OFFENSE_GROUP    0
DISTRICT         0
YEAR             0
MONTH            0
DAY              0
HOUR             0
dtype: int64
```

Figure 11: Screenshot showing removing null values in the data frame.

The output displays the number of null values in each column, which is now none. By grouping and displaying the data frame by the ‘OFFENSE_GROUP’ column (crimes), it can be seen that there are four instances of ‘Investigate Person’ being named ‘INVESTIGATE PERSON’. Figure 12 shows how this issue was solved by replacing the text value and displaying the correct list of crimes in the fixed data frame.



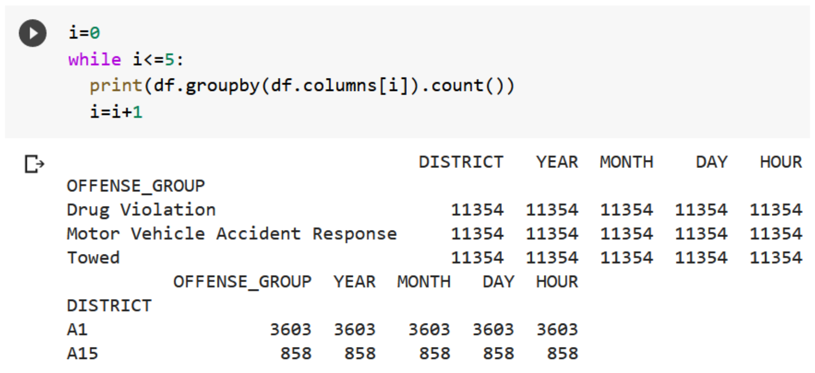
```
# Rename the erronous class and display the correction
df2NoError = df2_noNAN.replace({"INVESTIGATE PERSON":"Investigate Person"})
df2NoError.groupby("OFFENSE_GROUP").count()
```

OFFENSE_GROUP	DISTRICT	YEAR	MONTH	DAY	HOUR
Drug Violation	16870	16870	16870	16870	16870
Investigate Person	19106	19106	19106	19106	19106
Investigate Property	11354	11354	11354	11354	11354
Larceny	26590	26590	26590	26590	26590
Medical Assistance	24159	24159	24159	24159	24159
Motor Vehicle Accident Response	37672	37672	37672	37672	37672
Simple Assault	16223	16223	16223	16223	16223
Towed	11568	11568	11568	11568	11568
Vandalism	15761	15761	15761	15761	15761
Verbal Disputes	13465	13465	13465	13465	13465

Figure 12: A screenshot showing the replacement of the incorrect crime name.

5.2 DataAnalysis.ipynb

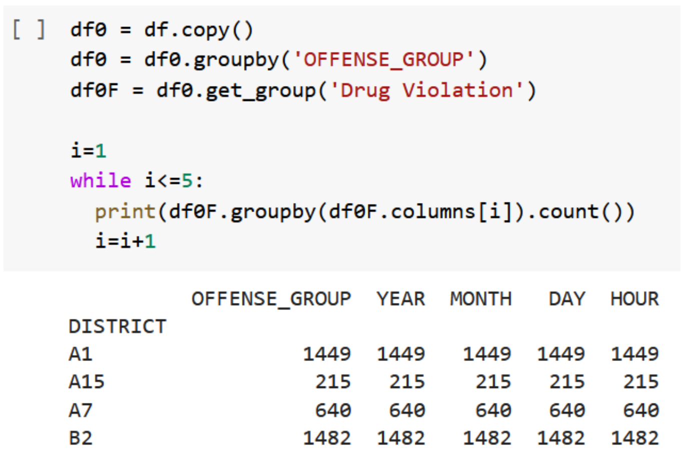
The second Google Colab notebook, ‘DataAnalysis.ipynb’, is the primary software artefact of this research project. Once the required dataset has been loaded into the notebook and read into a Pandas data frame, The distributions among the data can be calculated. Figure 13 shows a code snippet of how the whole data frame’s distribution was calculated, whereas Figure 14 shows how it is calculated among each class, for example, ‘Drug Violation’.



```
i=0
while i<=5:
  print(df.groupby(df.columns[i]).count())
  i=i+1
```

```
                                 DISTRICT   YEAR  MONTH    DAY   HOUR
OFFENSE_GROUP
Drug Violation                      11354  11354  11354  11354  11354
Motor Vehicle Accident Response     11354  11354  11354  11354  11354
Towed                               11354  11354  11354  11354  11354
              OFFENSE_GROUP  YEAR  MONTH   DAY  HOUR
DISTRICT
A1                     3603  3603   3603  3603  3603
A15                     858   858    858   858   858
```

Figure 13: A code snippet showing the occurrence of each instance in each feature for the data frame.

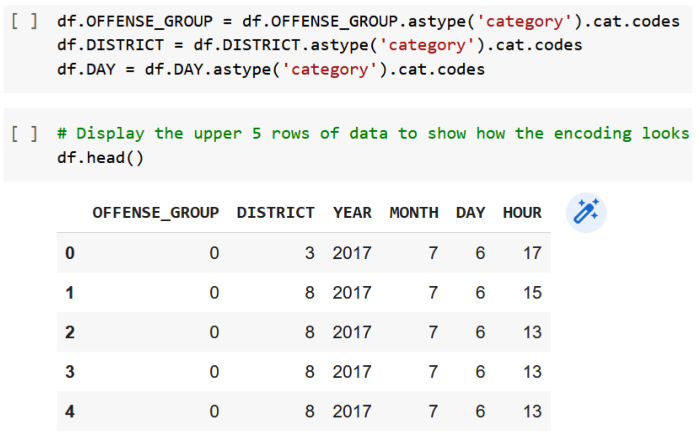


```
[ ]  df0 = df.copy()
     df0 = df0.groupby('OFFENSE_GROUP')
     df0F = df0.get_group('Drug Violation')

     i=1
     while i<=5:
       print(df0F.groupby(df0F.columns[i]).count())
       i=i+1
```

```
               OFFENSE_GROUP  YEAR  MONTH   DAY  HOUR
DISTRICT
A1                      1449  1449   1449  1449  1449
A15                      215   215    215   215   215
A7                       640   640    640   640   640
B2                      1482  1482   1482  1482  1482
```

Figure 14: A code snippet showing the occurrence of each instance in each feature for the 'Drug Violation' class.

The crime, district and day columns needed to be converted to numerical values to be processed by the sklearn libraries. This process can be seen in Figure 15.



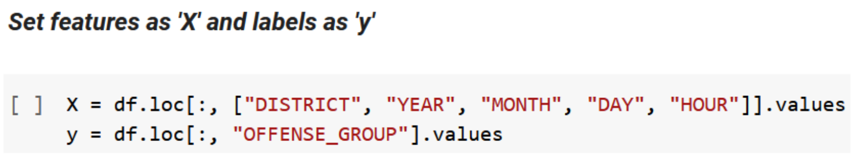
```
[ ]  df.OFFENSE_GROUP = df.OFFENSE_GROUP.astype('category').cat.codes
     df.DISTRICT = df.DISTRICT.astype('category').cat.codes
     df.DAY = df.DAY.astype('category').cat.codes
```

```
[ ]  # Display the upper 5 rows of data to show how the encoding looks
     df.head()
```

	OFFENSE_GROUP	DISTRICT	YEAR	MONTH	DAY	HOUR
0	0	3	2017	7	6	17
1	0	8	2017	7	6	15
2	0	8	2017	7	6	13
3	0	8	2017	7	6	13
4	0	8	2017	7	6	13

Figure 15: The encoding of the categorical valued columns.

In order to use the sklearn libraries to split the data and use the ML classifiers, the data frame would have to be split into a label and feature set. The code to complete this task is displayed in Figure 16.

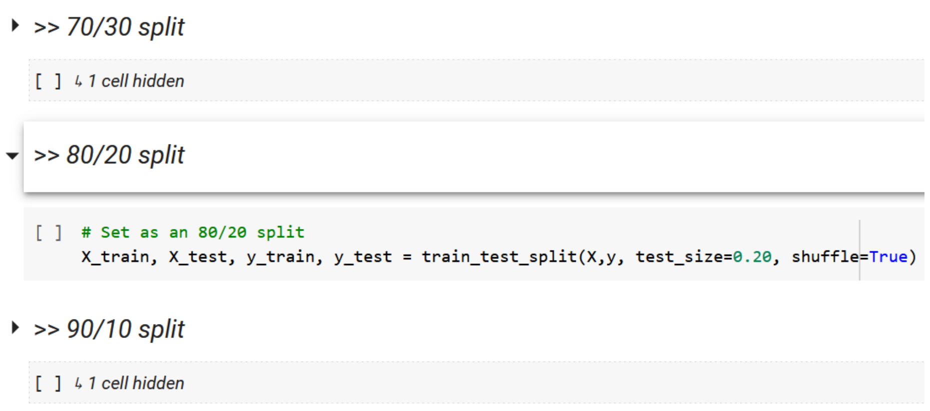


Set features as 'X' and labels as 'y'

```
[ ]  X = df.loc[:, ["DISTRICT", "YEAR", "MONTH", "DAY", "HOUR"]].values
     y = df.loc[:, "OFFENSE_GROUP"].values
```

Figure 16: A screenshot showing the creation of the label and feature sets.

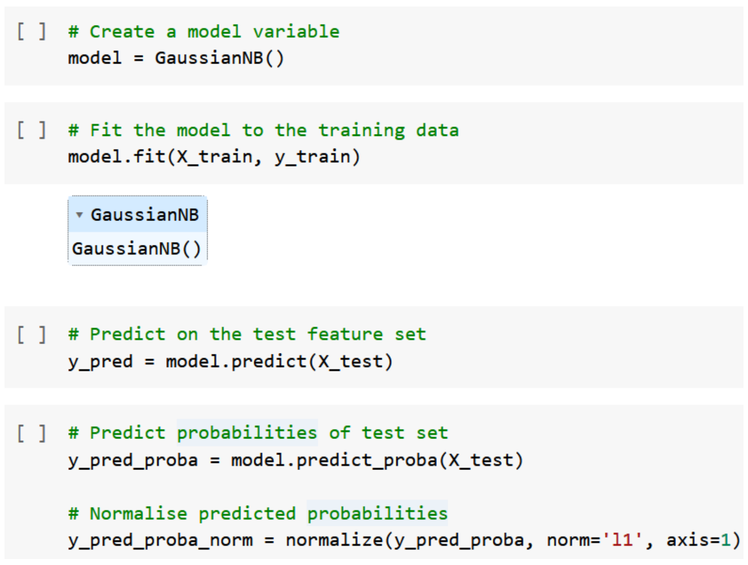
If this software were to be used by a different researcher, it would be possible for them to choose the train-test split methodology they required through the code in Figure 17. Although, for this research project, only the 80/20 split will be used.



```
>> 70/30 split
[ ] ↳ 1 cell hidden

>> 80/20 split
[ ] # Set as an 80/20 split
    X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20, shuffle=True)

>> 90/10 split
[ ] ↳ 1 cell hidden
```

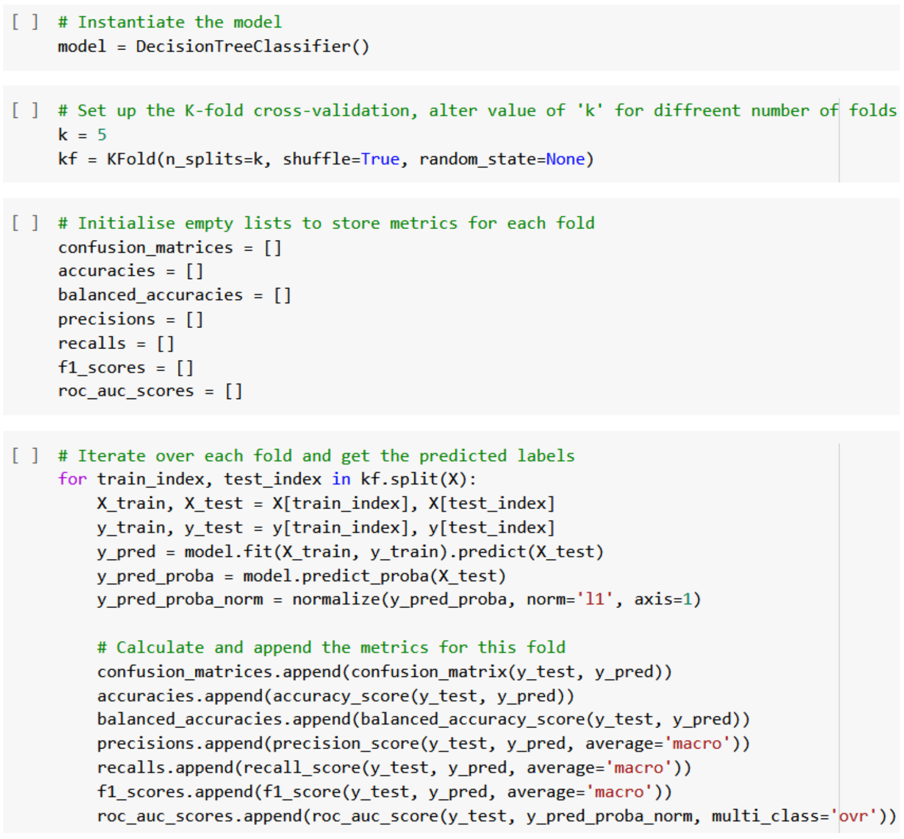
Figure 17: A code snippet showing how the data frame is split randomly into the training and test sets.

Figure 18 represents the implementation of the Naive Bayes classifier into the notebook in the standard run. Figure 19 shows how the Decision Tree classifier was implemented with a 5-fold cross-validation algorithm in the 5-fold cross-validation run.



```
[ ] # Create a model variable
    model = GaussianNB()

[ ] # Fit the model to the training data
    model.fit(X_train, y_train)

    ▾ GaussianNB
    GaussianNB()

[ ] # Predict on the test feature set
    y_pred = model.predict(X_test)

[ ] # Predict probabilities of test set
    y_pred_proba = model.predict_proba(X_test)

    # Normalise predicted probabilities
    y_pred_proba_norm = normalize(y_pred_proba, norm='l1', axis=1)
```

Figure 18: The implementation of the Naive Bayes classifier.



```
[ ] # Instantiate the model
    model = DecisionTreeClassifier()
```

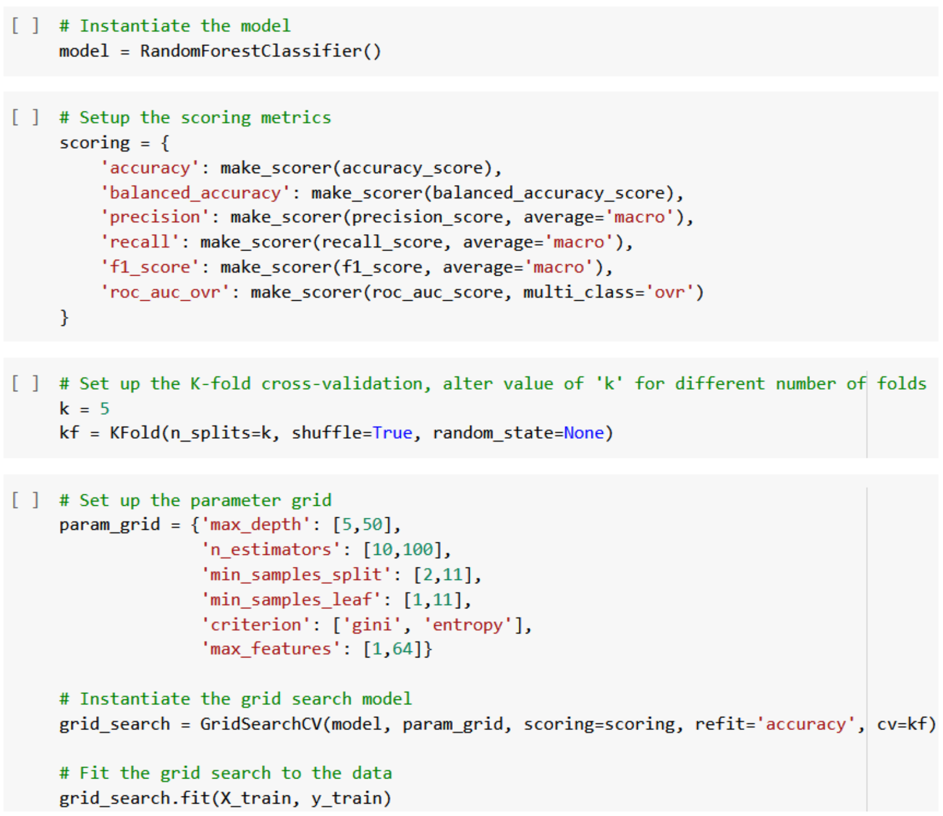
```
[ ] # Set up the K-fold cross-validation, alter value of 'k' for diffreent number of folds
    k = 5
    kf = KFold(n_splits=k, shuffle=True, random_state=None)
```

```
[ ] # Initialise empty lists to store metrics for each fold
    confusion_matrices = []
    accuracies = []
    balanced_accuracies = []
    precisions = []
    recalls = []
    f1_scores = []
    roc_auc_scores = []
```

```
[ ] # Iterate over each fold and get the predicted labels
    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        y_pred = model.fit(X_train, y_train).predict(X_test)
        y_pred_proba = model.predict_proba(X_test)
        y_pred_proba_norm = normalize(y_pred_proba, norm='l1', axis=1)

        # Calculate and append the metrics for this fold
        confusion_matrices.append(confusion_matrix(y_test, y_pred))
        accuracies.append(accuracy_score(y_test, y_pred))
        balanced_accuracies.append(balanced_accuracy_score(y_test, y_pred))
        precisions.append(precision_score(y_test, y_pred, average='macro'))
        recalls.append(recall_score(y_test, y_pred, average='macro'))
        f1_scores.append(f1_score(y_test, y_pred, average='macro'))
        roc_auc_scores.append(roc_auc_score(y_test, y_pred_proba_norm, multi_class='ovr'))
```

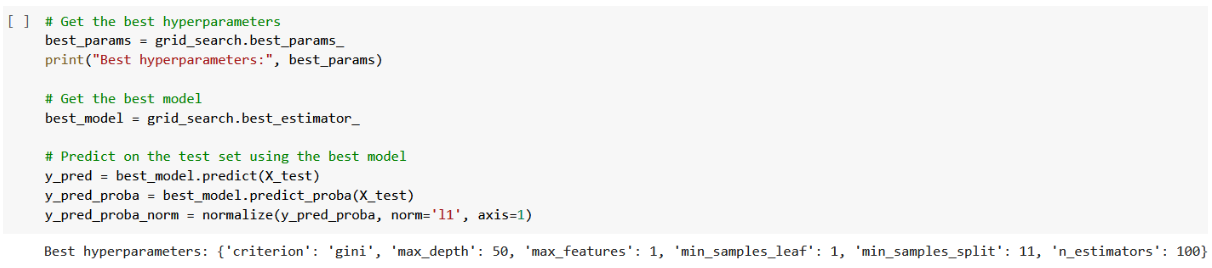
Figure 19: A screenshot showing the Decision Tree classifier with 5-fold cross-validation.

Figures 20 and 21 demonstrate how the Random Forest classifier was implemented with GridSeachCV to incorporate hyperparameter tuning with 5-fold. Figure 20 shows the setup and how the training data was fitted to the model. Figure 21 shows which hyperparameters were the best choice and how predictions were made on the test set.



```
[ ]  # Instantiate the model
     model = RandomForestClassifier()

[ ]  # Setup the scoring metrics
     scoring = {
         'accuracy': make_scorer(accuracy_score),
         'balanced_accuracy': make_scorer(balanced_accuracy_score),
         'precision': make_scorer(precision_score, average='macro'),
         'recall': make_scorer(recall_score, average='macro'),
         'f1_score': make_scorer(f1_score, average='macro'),
         'roc_auc_ovr': make_scorer(roc_auc_score, multi_class='ovr')
     }

[ ]  # Set up the K-fold cross-validation, alter value of 'k' for different number of folds
     k = 5
     kf = KFold(n_splits=k, shuffle=True, random_state=None)

[ ]  # Set up the parameter grid
     param_grid = {'max_depth': [5,50],
                   'n_estimators': [10,100],
                   'min_samples_split': [2,11],
                   'min_samples_leaf': [1,11],
                   'criterion': ['gini', 'entropy'],
                   'max_features': [1,64]}

     # Instantiate the grid search model
     grid_search = GridSearchCV(model, param_grid, scoring=scoring, refit='accuracy', cv=kf)

     # Fit the grid search to the data
     grid_search.fit(X_train, y_train)
```

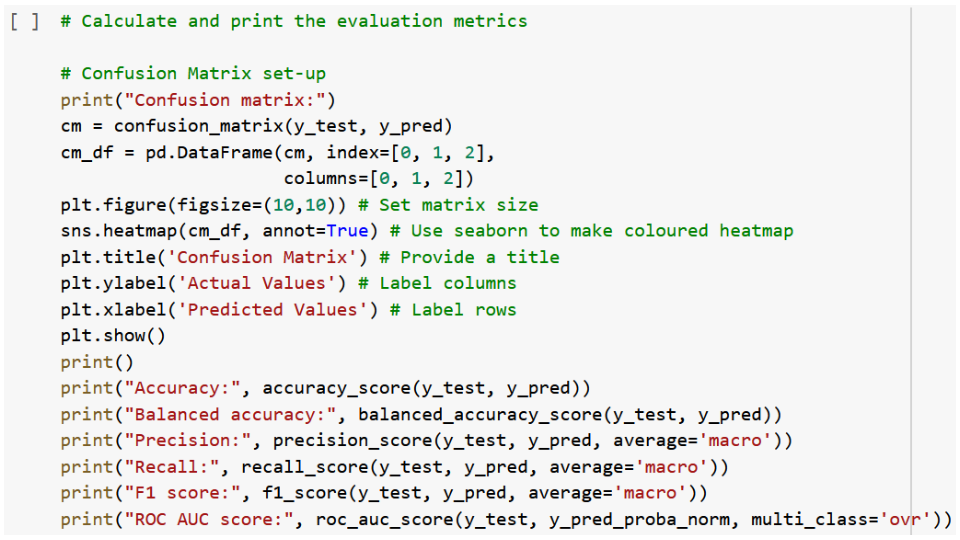
Figure 20: The code used to set up variables for GridSearchCV and fit the Random Forest Classifier.



```
[ ]  # Get the best hyperparameters
     best_params = grid_search.best_params_
     print("Best hyperparameters:", best_params)

     # Get the best model
     best_model = grid_search.best_estimator_

     # Predict on the test set using the best model
     y_pred = best_model.predict(X_test)
     y_pred_proba = best_model.predict_proba(X_test)
     y_pred_proba_norm = normalize(y_pred_proba, norm='l1', axis=1)

     Best hyperparameters: {'criterion': 'gini', 'max_depth': 50, 'max_features': 1, 'min_samples_leaf': 1, 'min_samples_split': 11, 'n_estimators': 100}
```

Figure 21: A screenshot of the best hyperparameters chosen and how the model predicts the test set.

In order to evaluate the model's performance, the metrics must be displayed to the user. This process can be seen in the code from Figure 22.



```
[ ]  # Calculate and print the evaluation metrics

     # Confusion Matrix set-up
     print("Confusion matrix:")
     cm = confusion_matrix(y_test, y_pred)
     cm_df = pd.DataFrame(cm, index=[0, 1, 2],
                          columns=[0, 1, 2])
     plt.figure(figsize=(10,10)) # Set matrix size
     sns.heatmap(cm_df, annot=True) # Use seaborn to make coloured heatmap
     plt.title('Confusion Matrix') # Provide a title
     plt.ylabel('Actual Values') # Label columns
     plt.xlabel('Predicted Values') # Label rows
     plt.show()
     print()
     print("Accuracy:", accuracy_score(y_test, y_pred))
     print("Balanced accuracy:", balanced_accuracy_score(y_test, y_pred))
     print("Precision:", precision_score(y_test, y_pred, average='macro'))
     print("Recall:", recall_score(y_test, y_pred, average='macro'))
     print("F1 score:", f1_score(y_test, y_pred, average='macro'))
     print("ROC AUC score:", roc_auc_score(y_test, y_pred_proba_norm, multi_class='ovr'))
```

Figure 22: A screenshot showing the calculations for the performance metrics.

The ‘matplotlib’ library was imported and incorporated to create the confusion matrix. It is a plotting library for Python, and with Seaborn, a heatmap confusion matrix can be produced to visualise a model’s prediction performance.

Chapter 6

Results & Discussion

6.1 Data Distribution

A distribution of discrete data is a probability distribution of the occurrence of individually countable outcomes (Investopedia, 2019). For the data used in this research project, multinomial distribution is used. The multinomial distribution is employed when an experiment has the potential to yield more than two possible outcomes. The following figures display the dataset's feature distribution among the three classes.

6.1.1 District

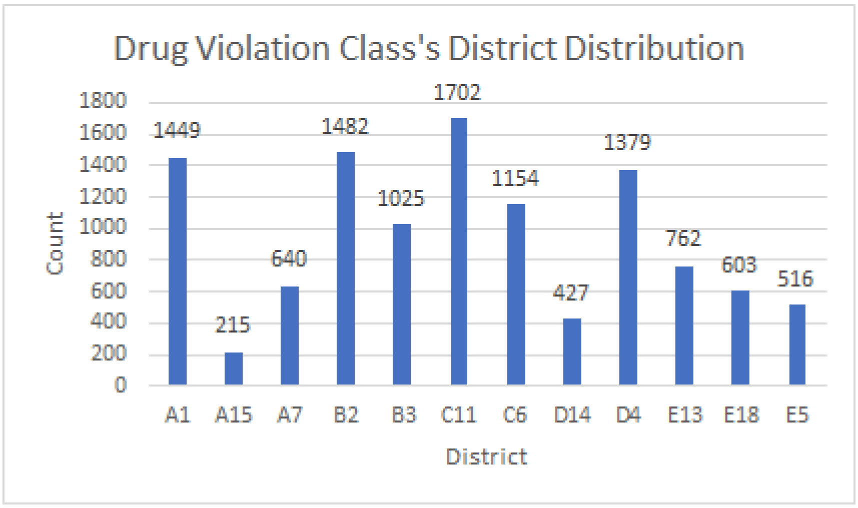


Figure 23: A bar chart showing the district feature's distribution among the Drug Violation class.

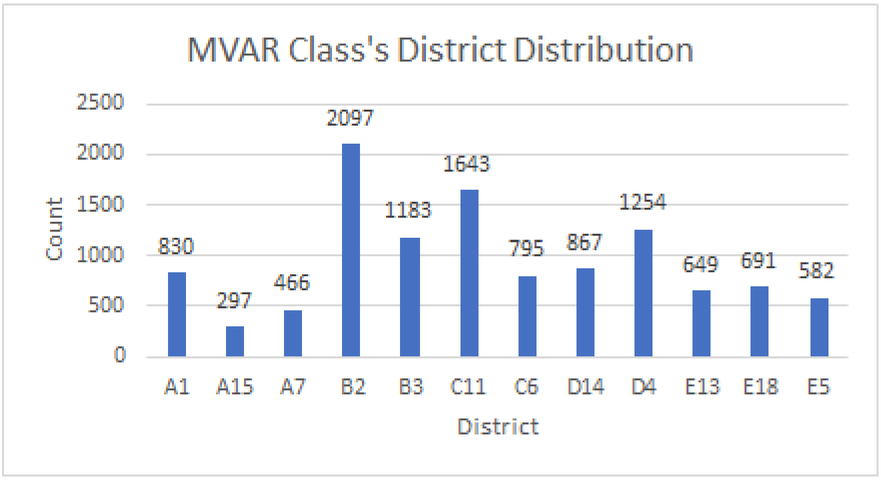


Figure 24: A bar chart showing the district feature's distribution among the MVAR class.

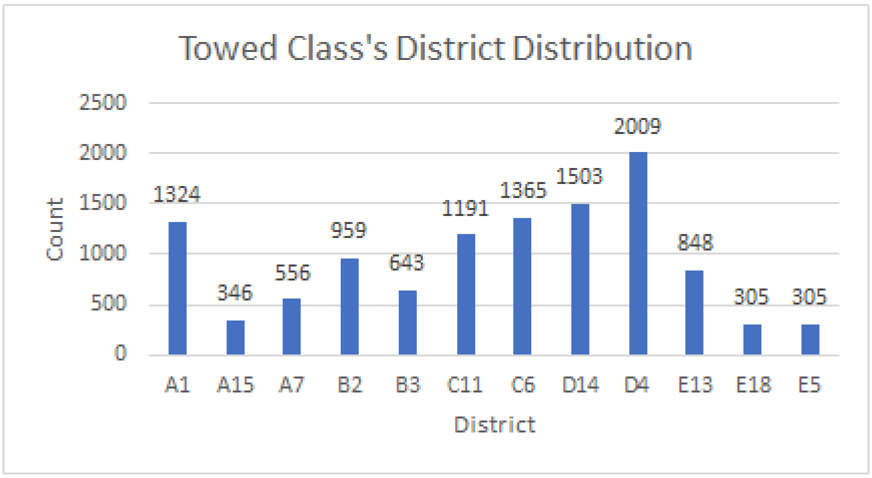


Figure 25: A bar chart showing the district feature's distribution among the Towed class.

Figure 23 shows that the most typical district in the Drug Violation class is 'C11' with 1,702 occurrences, followed closely by 'A1', 'B2' and 'D4' with counts of 1449, 1482 and 1,379, respectively. District 'A15' has the most minor occurrences, with only 215. Figure 24 shows that the MVAR class has the most variance in the districts, with a difference of 1,800 occurrences between district 'B2' (2097) and 'A15' (297). For the Towed class, as seen in Figure 25, the district 'D4' has the most counts in the data with 2,009, although 'E18' and 'E5' have the fewest with 305. A trend among these classes is that there are few district 'A15' occurrences across the dataset, but many in 'B2' and 'D4'. The number of occurrences in the other districts varied across the classes, which should help the classifiers make more accurate predictions based on the districts.

6.1.2 Year

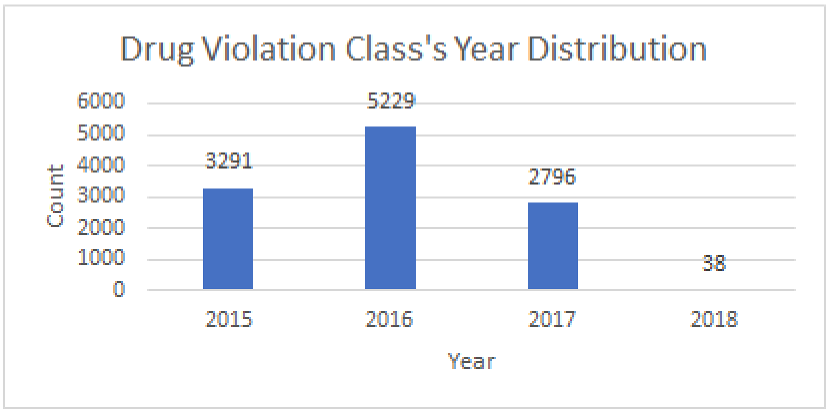


Figure 26: A bar chart showing the year feature's distribution among the Drug Violation class.

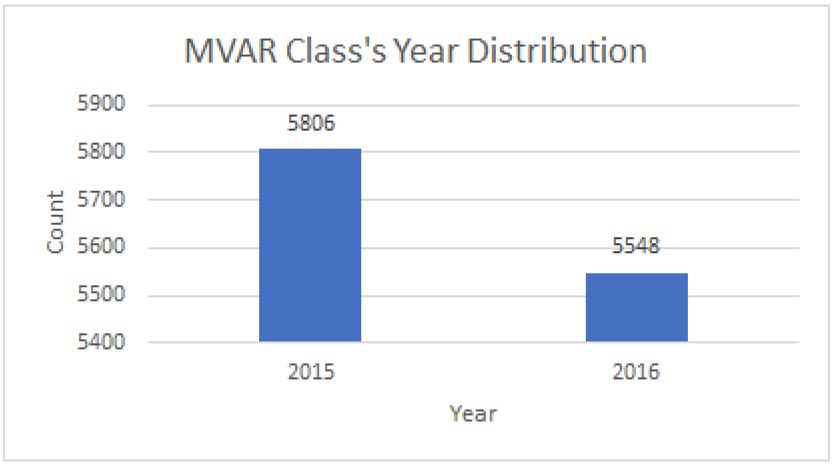


Figure 27: A bar chart showing the year feature's distribution among the MVAR class.

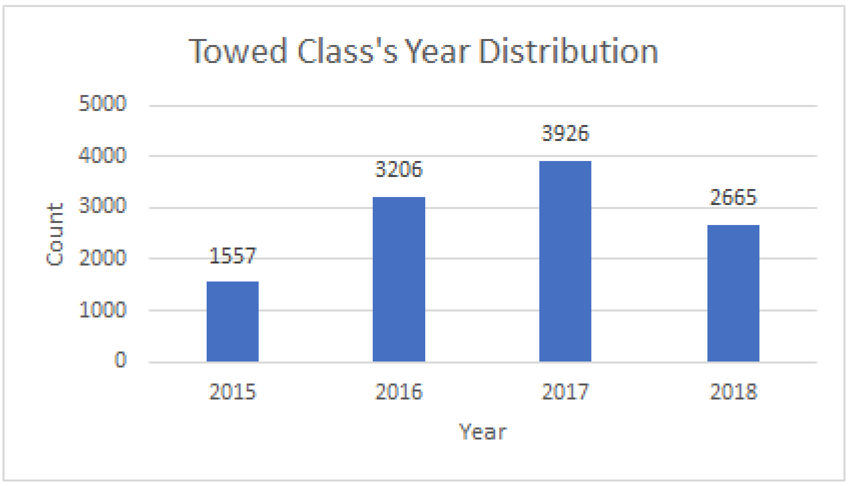


Figure 28: A bar chart showing the year feature's distribution among the Towed class.

Referring to Figure 26, 2016 has the most occurrences of drug violations with 5,229 counts. 2015 and 2017 still have many occurrences, with 3,291 and 2,796 counts, respectively. Although, the year 2018 is hugely underrepresented in this class, with only 38 total occurrences. Figure 27 shows that the MVAR class only contains crime data from 2015 and 2016, with only a difference of 258 occurrences between them. Therefore, predicting the MVAR class will be more unlikely in the prediction stage if the data contains the years 2017 or 2018. In Figure 28, it can be seen that the year 2018 is not the least occurring year, as there are 1,108 more counts compared to 2015. As a result, Towed should be predicted more often in the prediction stage when that data contains the year 2018.

6.1.3 Month

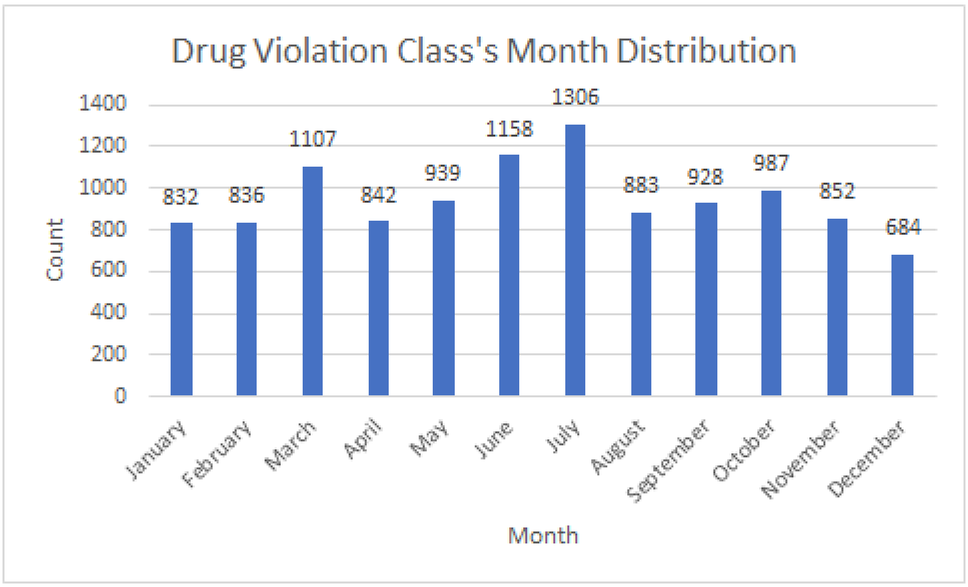


Figure 29: A bar chart showing the month feature's distribution among the Drug Violation class.

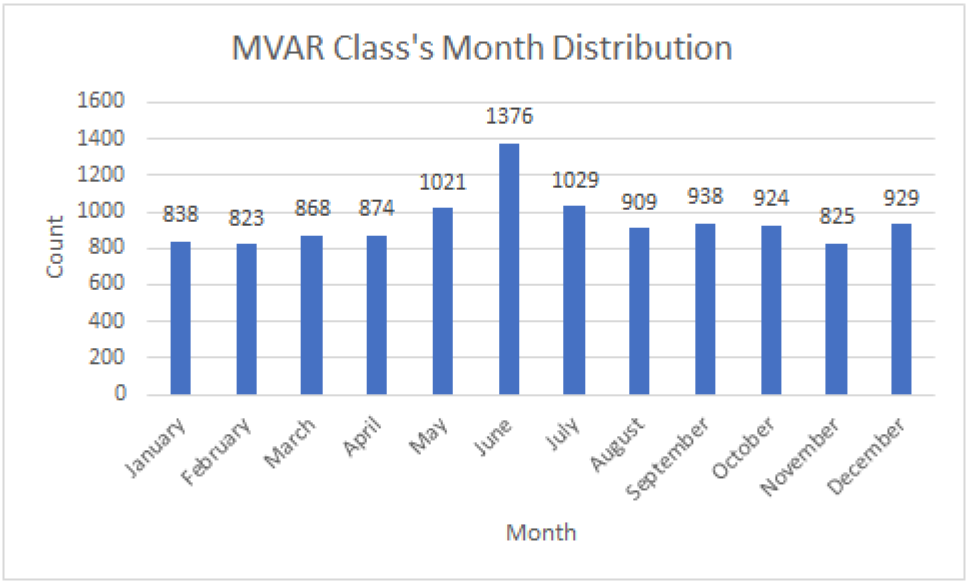


Figure 30: A bar chart showing the month feature's distribution among the MVAR class.

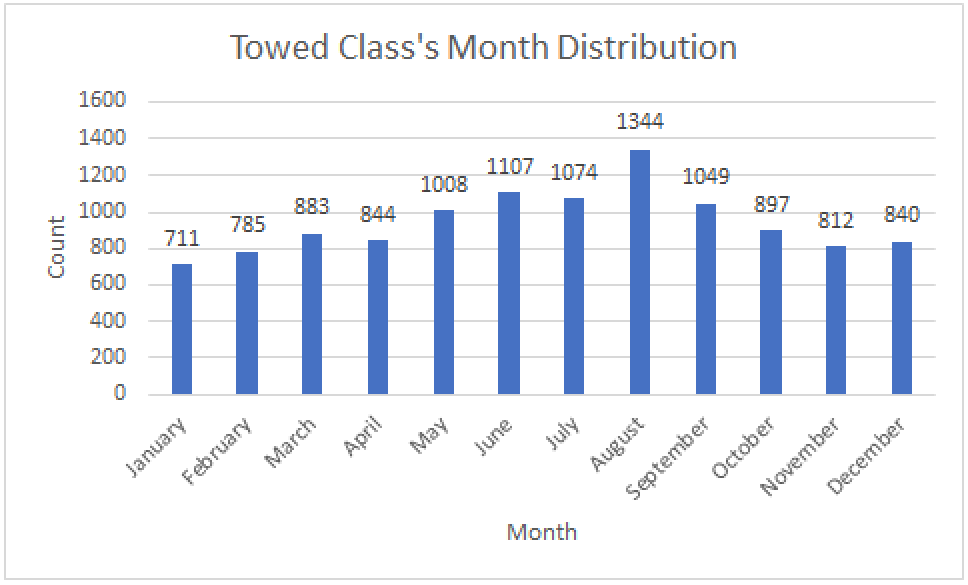


Figure 31: A bar chart showing the month feature's distribution among the Towed class.

Figure 29 shows that early to mid-summer and early spring have the most drug violations, with counts of 1,306, 1,158 and 1,107 for July, June and March, respectively. December is the least typical month for drug violations to be committed, although they are still common, with 684 occurrences. Motor Vehicle Accident Responses are also most common in June. Figure 30 shows that there are 1,376 counts of MVAR in June, although the rates are consistent throughout the year, ranging from 1,029 in July to 823 in February. In Figure 31, August counts the highest number of towed instances, with 1,344 total. Again, late spring and summer contain the most occurrences, whereas the winter months have the least; January contains only 711 counts of the Towed class.

This data suggests that these crimes are prevalent throughout the year but are even more common in late spring and summer. Perhaps this is due to the better weather when more people will be outside, rather than inside when it is cold and more rainy in autumn and winter.

6.1.4 Day

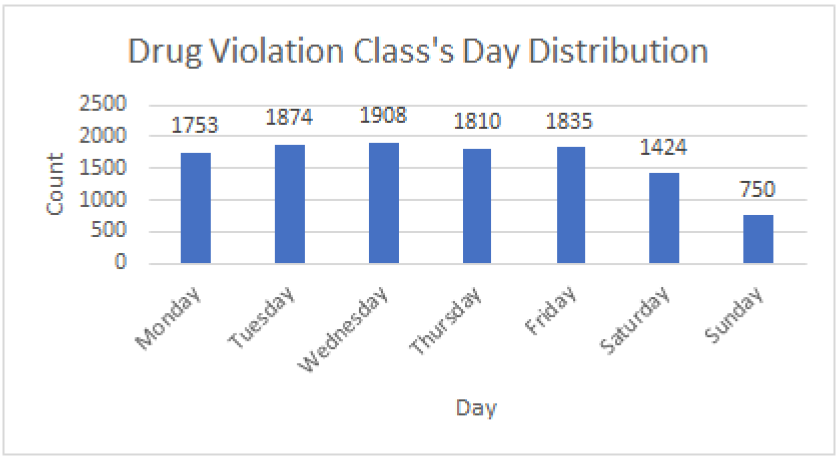


Figure 32: A bar chart showing the day feature's distribution among the Drug Violations class.

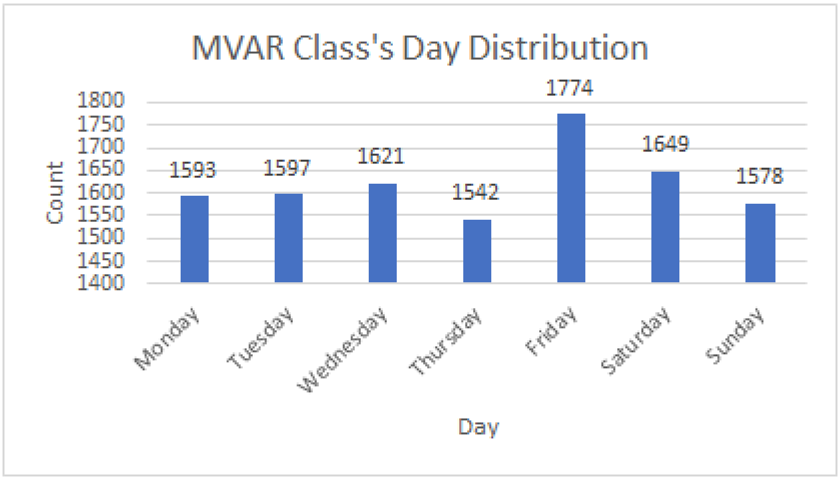


Figure 33: A bar chart showing the day feature's distribution among the MVAR class.

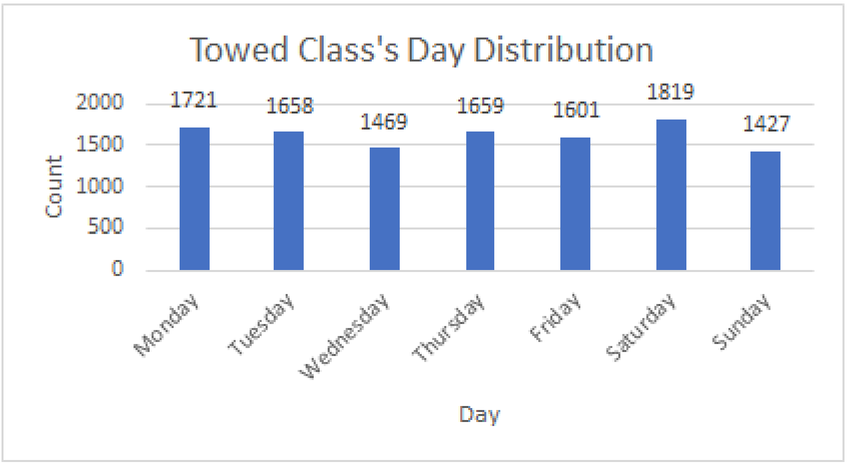


Figure 34: A bar chart showing the day feature's distribution among the Towed class.

Figure 32 shows that drug violations are committed equally on weekdays but are reduced on weekends. Wednesday has the most occurrences, with 1,908, whereas Saturday and Sunday have far fewer, with 1,424 and 750, respectively. Figure 33 displays the number of occurrences for the MVAR. The distribution shows that Friday is the most common day for this crime to be committed, with 1,774 total instances across the dataset. However, Thursday has the least number of instances, with only 1,542. In Figure 34, it can be concluded that Towes occur evenly across each day of the week. The variance only ranges from 1,819 for Saturday to 1,427 for Sunday. The models may find difficulty in distinguishing between the drug violations class and the towed class based on the day provided in the data.

6.1.5 Hour

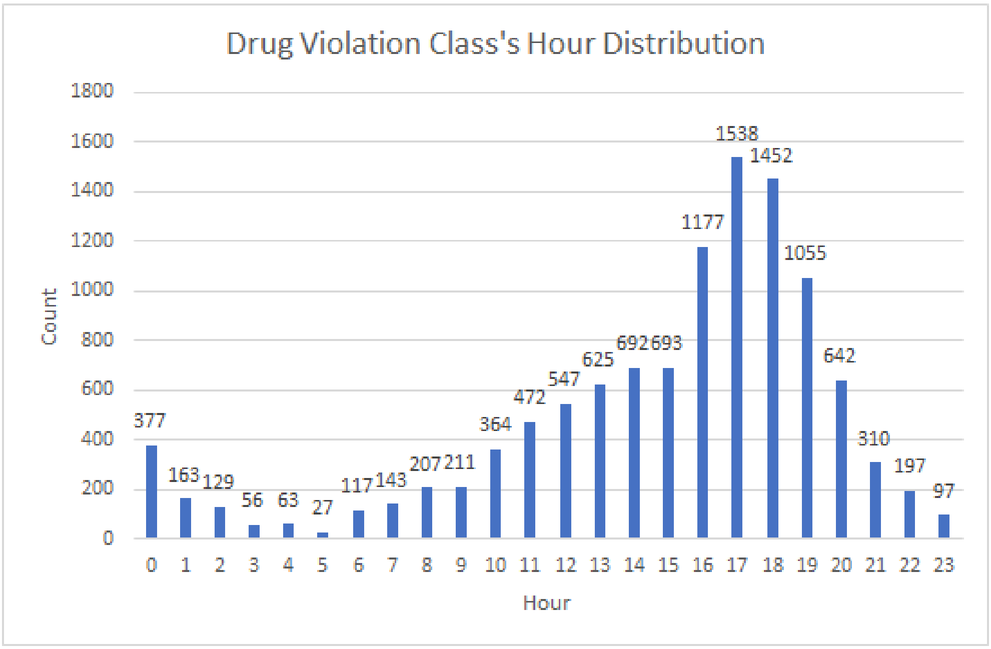


Figure 35: A bar chart showing the hour feature's distribution among the Drug violations class.

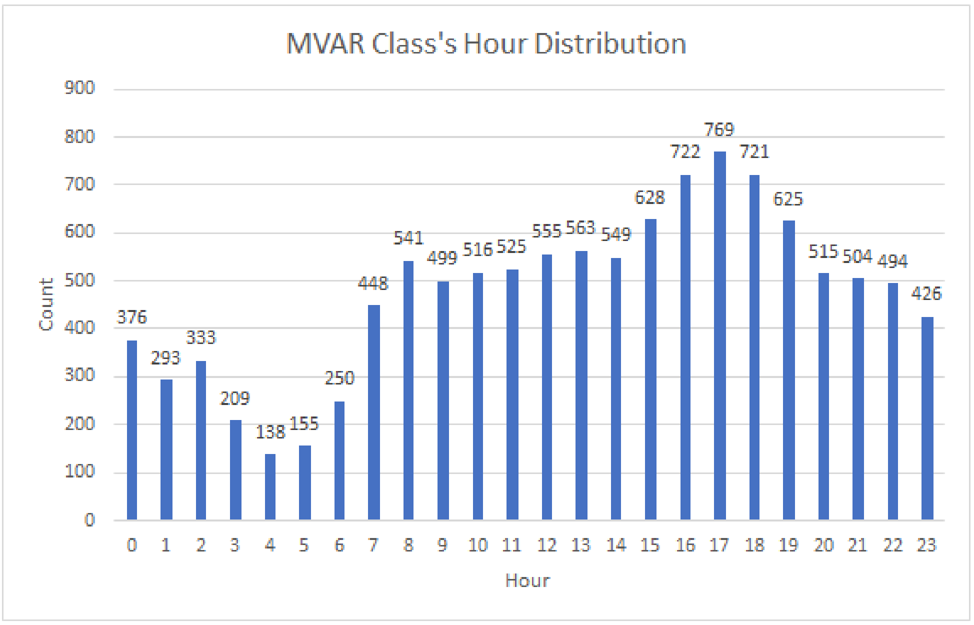


Figure 36: A bar chart showing the hour feature's distribution among the MVAR class.

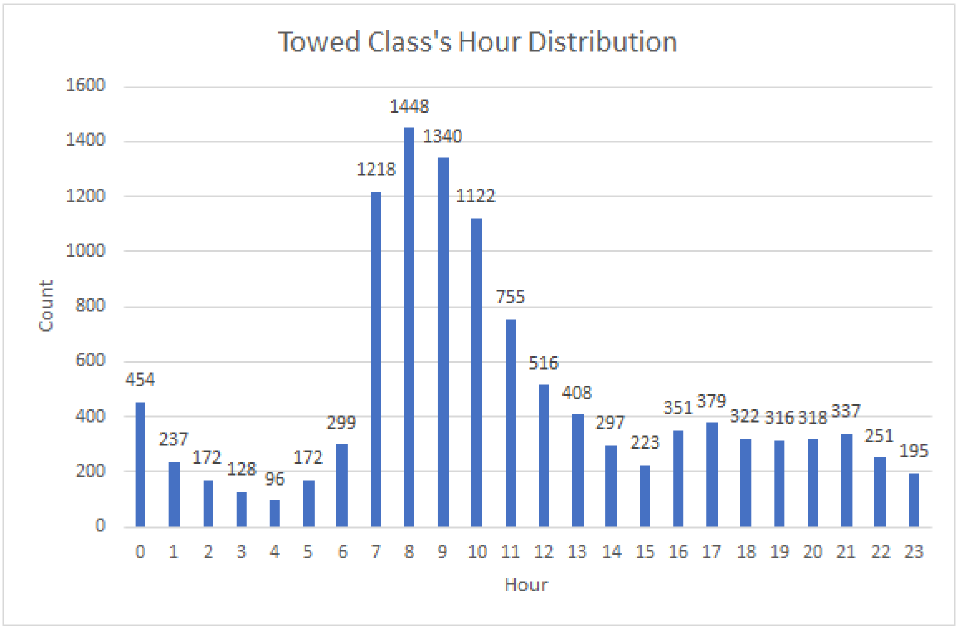


Figure 37: A bar chart showing the hour feature's distribution among the Towed class.

Figure 35 shows that drug violations occur more from 16:00 to 19:00, with the most occurring at 17:00, with 1,538 counted instances in this dataset. 05:00 is the hour when drug violations are committed the least, as only 27 counted instances are present. In Figure 36, the same trend is present, with 17:00 holding the most instances of MVAR crime happening with

769 counts. The early morning also has fewer occurrences, with 04:00 and 05:00 counted only 138 and 155 times across the dataset. Finally, Figure 37 shows the occurrences of towed crime across the dataset. Unlike the other two crimes, Towes are more likely to occur between 07:00 and 10:00, whereas 17:00 contains very few occurrences, with 223 counts. Although, this is higher than 04:00, which has only been counted 96 times across the whole dataset for this class.

As a result, predicting crime will not be an issue if the hour provided is between '7 and '10', but it may be difficult to distinguish between drug violations and MVAR if '17' is the hour provided to the model.

6.2 Model Performance

6.2.1 Naive Bayes

Table 2 summarises the performance results for the Naive Bayes Classifier on the different run types.

Run Type	Accuracy	Balanced Accuracy	Precision	Recall	F1 Score	ROC AUC
Standard	0.55438133	0.554132871	0.567152071	0.554132871	0.557332263	0.758473611
5-Fold	0.559186205	0.559237684	0.574166378	0.559237684	0.562755673	0.76272804
HPT	0.567527892	0.568047605	0.578969681	0.568047605	0.57091446	0.76903541

Table 2: The performance metrics for Naive Bayes on the different runs.

The metric scores for each run stay within a similar range, apart from the ROC AUC score, which scores higher by 0.2. This may be because accuracy, balanced accuracy, precision, recall and F1-score measure the model as a whole, whereas the ROC AUC score measures binary outputs using the 'One vs Rest' approach. A ROC AUC score of 0.758 for the standard run indicates that the model has a moderate level of discrimination between the positive and negative classes. It also means Naive Bayes performs better than random guessing but still has room for improvement. An accuracy and balanced accuracy of 0.55 and 0.56 shows that Naive Bayes only precis the correct class just over 50% of the time, indicating poor performance. As precision, recall and F1-scores also yield values of 0.55 and 0.56, the model creates an equal number of true positives and false positives or true positives and false negatives. The 5-fold run produced values that improved by 0.01, and the HPT run only improved those values by a further 0.01. These results indicate poor performance from the Naive Bayes classifier at predicting crime. However, naive Bayes performed best in the HPT run. The best-performing hyperparameter for Naive Bayes was:

- 'var_smoothing': 1e-09

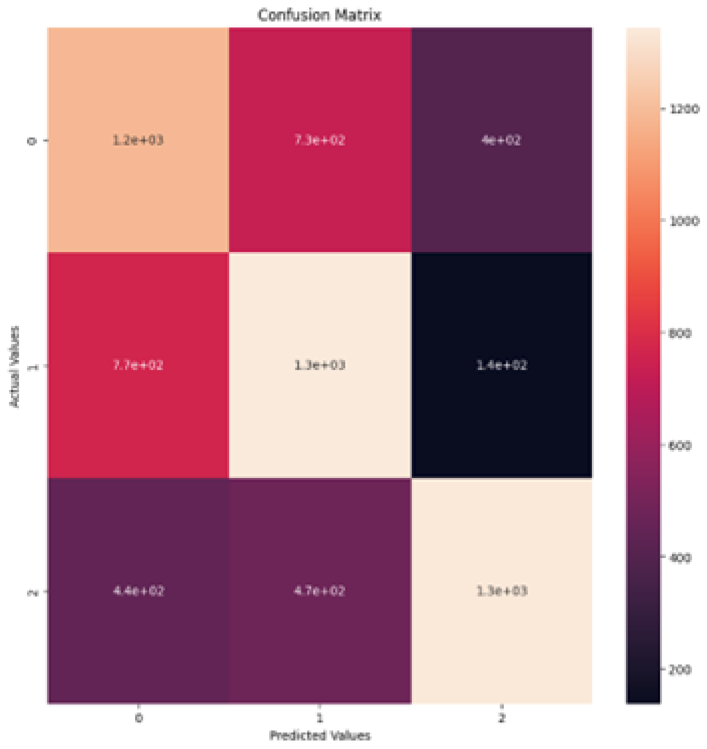


Figure 38: The confusion matrix for Naive Bayes's HPT run.

The confusion matrix in Figure 38 represents Naive Bayes when HPT is applied. It can be seen that the classes MVAR and Towed both have equal TP scores of roughly 1,300, meaning they were predicted correctly roughly 1,300 times by the classifier. Drug Violation and MVAR have high predictions; Drug Violation was predicted roughly 2,410 times, and MVAR was correctly predicted roughly 2,500 times.

6.2.2 Decision Tree

Table 3 contains the performance matric scores for the Decision tree classifier.

Run Type	Accuracy	Balanced Accuracy	Precision	Recall	F1 Score	ROC AUC
Standard	0.661773341	0.661349843	0.668284482	0.661349843	0.662427683	0.767045071
5-Fold	0.65463006	0.654608062	0.660918485	0.6546080	0.655562393	0.762462081

	8			62		
HPT	0.664562537	0.664520955	0.675286132	0.664520955	0.66642267	0.837648295

Table 3: The performance metrics for Decision Tree on the different runs.

The metric scores for the Decision Tree classifier improve on the Naive Bayes classifier by 0.1, or 10%. The ROC AUC scores are higher than the other metrics but do not improve significantly until the HPT run, where they increase from 0.76 to 0.83. This means that hyperparameter tuning on the decision tree helps distinguish between the three classes easily. However, the other metrics still prove that the classifier only predicts the correct class around 66% of the time, which is not good enough when a police force tries to predict a crime that may be taking place or when lives are being threatened. Another point is that the 5-fold run yielded worse results than the standard run with a simple train-test split with no cross-validation. The variation among the folds may contribute to this discrepancy, as some folds may exhibit patterns or traits that do not accurately represent the entire dataset, leading to subpar performance of the classifier on those folds. Although, implementing hyperparameter tuning produced better results for the Decision Tree classifier. The combination of hyperparameters that received these scores is:

- 'criterion': 'entropy'
- 'max_depth': 50
- 'max_features': 64
- 'min_samples_leaf': 11
- 'min_samples_split': 2

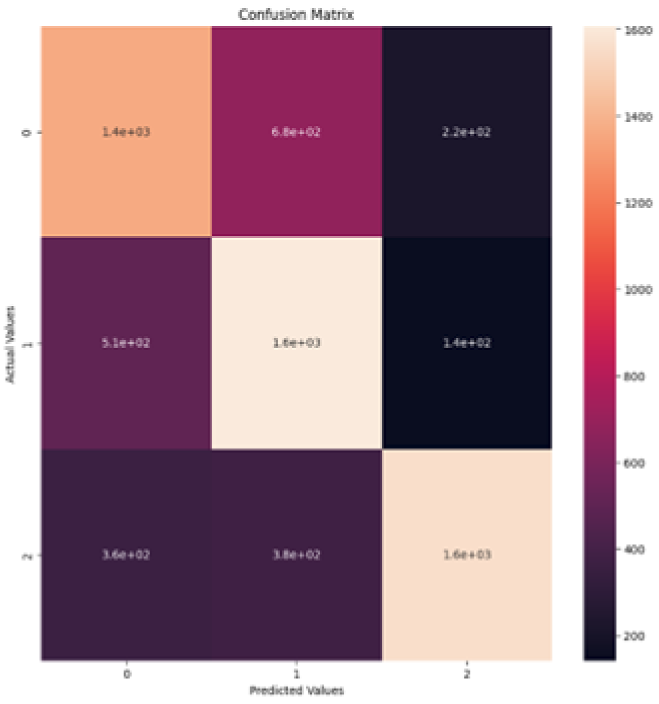


Figure 39: The confusion matrix for Decision Tree's HPT run.

Figure 39 shows that MVAR has the most TP predictions of the three classes, with roughly 1,600 predictions and is the most accessible class to predict. Even though Towed received 1,600 predictions, the darker shade from the heatmap suggests that it received fewer predictions than MVAR. Drug Violations were correctly predicted less often than the other two classes because it only received roughly 1,400 TP predictions. The values in the confusion matrices are crucial for calculating the metrics and building a picture of the classifier's performance.

6.2.3 Random Forest

Table 4 contains the performance metrics for the Random Forest classifier across the three different run types.

Run Type	Accuracy	Balanced Accuracy	Precision	Recall	F1 Score	ROC AUC
Standard	0.689371697	0.689256586	0.694735157	0.689256586	0.690710665	0.851861076
5-Fold	0.682109157	0.682096802	0.686738852	0.682096802	0.683354518	0.851218654
HPT	0.694509689	0.69425789	0.704636466	0.69425789	0.695378759	0.861751155

Table 4: The performance metrics for Random Forest on the different runs.

Based on the scores from Table 4, it is suggested that the standard run and 5-fold run predict the correct class around 68% of the time; this is an improvement over the Decision Tree classifier by 2% or 0.02. While this is an increase, it still proves significant room for improvement for the Random Forest classifier. The ROC AUC scores of 0.85 for the standard run and 5-fold run prove that the classifier improved in distinguishing between the classes and can do so to significant effect. However, the 5-fold run performed poorly against using only an 80/20 train-test split by around 0.007 for accuracy, balanced accuracy, precision, recall and F1-score. The Random Forest classifier performed best when HPT was implemented, although results only improved by 0.01. The optimal hyperparameter combination for Random Forest was:

- 'criterion': 'gini'
- 'max_depth': 50
- 'max_features': 1
- 'min_samples_leaf': 1

- 'min_samples_split': 11
- 'n_estimators': 100

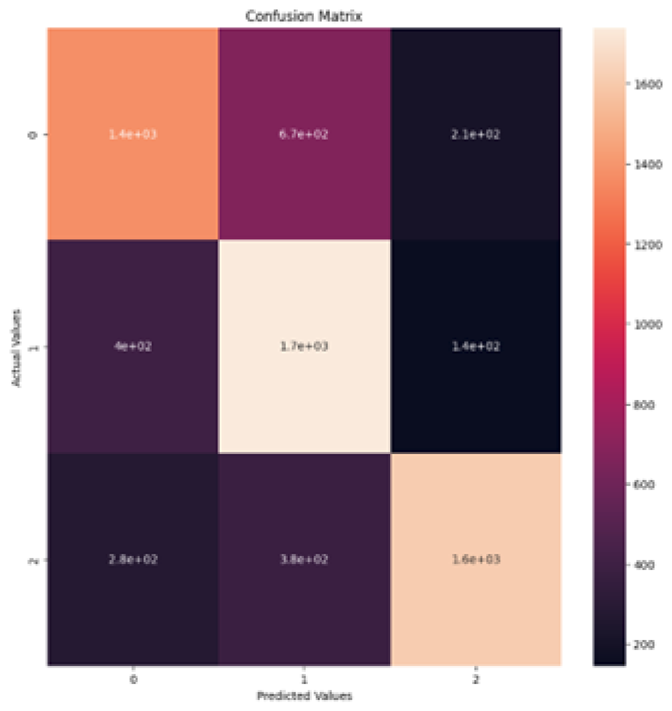


Figure 40: The confusion matrix for Random Forest's HPT run.

Figure 40 represents the confusion matrix used to calculate the performance metrics for the Random Forest classifier. MVAR received the most TP predictions, with roughly 1,700 predictions, whereas Towed and Drug Violation received fewer, with 1,600 and 1,400, respectively. These TP values are the highest among all runs in all classifiers.

6.2.4 K-Nearest Neighbours

Table 5 shows the performance metrics for each run type of the KNN classifier.

Run Type	Accuracy	Balanced Accuracy	Precision	Recall	F1 Score	ROC AUC
Standard	0.614357017	0.614618038	0.64895558	0.614618038	0.617726272	0.792172927
5-Fold	0.619752349	0.619795023	0.654666477	0.619795023	0.623216085	0.793249229
HPT	0.628156195	0.627954309	0.646105169	0.627954309	0.631094906	0.81435211

Table 5: The performance metrics for K-Nearest Neighbour on the different runs.

The metric scores for the KNN classifier are lower than Random Forest's but still above 50%. Precision scores are slightly higher across all three runs by about 0.03, meaning positive predictions have higher accuracy. ROC AUC still scores higher than the other metrics by around 0.18, or 18%, meaning that the classifier is still successful at making distinctions between the three classes. Using HPT with the KNN classifier also yields higher scores across all metrics, as seen with the three previous classifiers, besides precision. However, the difference between the performance of the HPT run and the 5-fold run is only 0.01, and the difference between the 5-fold run and the standard run is only 0.005. The best hyperparameter combination used in hyperparameter tuning for the KNN classifier was:

- 'n_neighbors': 20

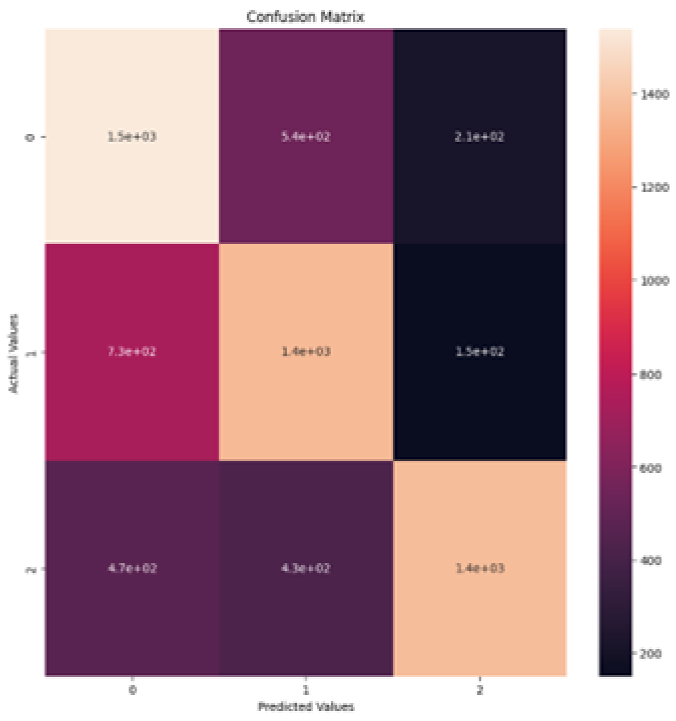


Figure 41: The confusion matrix for K-Nearest Neighbour's HPT run.

Figure 41 displays the confusion matrix used to calculate the performance metrics for the KNN classifier's HPT run performance metrics. The Drug Violation class received the most TP predictions with roughly 1,500 predictions, although MVAR and Towed only received around 100 fewer predictions. Overall, KNN consistently correctly predicted the TP variables for each class.

6.2.5 Ten Classes

Table 6 shows the metric performance scores across the four classifiers when trained and tested using the 80/20 test-train split with a dataset including ten classes. This dataset was used originally but produced scores too inaccurate to be used in a real-world setting. As a result, further research was conducted to use three classes instead.

Classifier	Accuracy	Balanced Accuracy	Precision	Recall	F1 Score	ROC AUC
Naïve Bayes	0.199841466	0.200625668	0.186361376	0.200625668	0.175431085	0.667291845
Decision Tree	0.224458341	0.225635623	0.222964812	0.225635623	0.218329707	0.599001324
Random Forest	0.230579531	0.23130134	0.228478179	0.23130134	0.229142688	0.679121176
K-Nearest Neighbours	0.221243615	0.222380138	0.224765118	0.222380138	0.215340323	0.631115207

Table 6: The performance metrics of the four classifiers with the ten-class dataset.

6.3 Discussion

For each classifier, hyperparameter tuning with 5-fold cross-validation proved more successful in each metric, despite minor improvements. Of the four classifiers implemented in the software artefact, the Random Forest classifier performed the best across each run version. Random Forest performed best overall when applied hyperparameter tuning and 5-fold cross-validation. It received an overall metric score of 0.6966081388 for accuracy, balanced accuracy, precision, recall and F1-score, and a ROC AUC score of 0.861751155.

The aim and objectives of this research project were met with fantastic results, showcasing the difference between the chosen machine learning classifiers and the impact 5-fold cross-validation and hyperparameter tuning have on their performances. Various metrics were used to represent the diverse performance of the classifiers. Each functional and non-functional requirement was met too and covered in detail to describe the effect each function had on the process and outcome of the project.

This project was a success, and the information shown can be used by any law enforcement agency or police department to predict crime in the future. This evaluation proves that to predict crime accurately, a Random Forest classifier must be used with Hyperparameter tuning and 5-fold cross-validation.

Chapter 7

Conclusion

The original aims for this project focused on developing a web application that would predict crime based on the Naive Bayes Classifier. The objectives required using a dataset containing crime from Lincoln, England and developing the Naive Bayes classifier from scratch. These objectives proved too ambitious, so the project changed to discover which classifier performed the best when predicting crime. This allowed more profound research into the field of machine learning.

Sufficient research was conducted to use the correct methodology and achieve the best outcomes. Machine learning covers many areas, so research was done to fulfil the new aims and objectives. The final software artefact implements every feature set in the requirements section and is valuable.

The results indicate that the Random Forest classifier should use hyperparameter tuning and 5-fold cross-validation for accurate crime predictions. This can help police departments reduce crime and associated costs. However, the dataset used only three classes, each with 11,354 entries. The project used a fixed number of hyperparameters per classifier, 5-fold cross-validation, and an 80/20 split. Overfitting may have occurred despite these decisions being based on previous research and analysis.

The results support the claims that the project can be used in a real-world setting, although there is room for improvement. If continued in the future, the project could implement deep learning techniques like Long Short-Term Memory (LSTM), applied to areas other than Boston or the USA, provided that appropriate police departments make the data public and exclude sensitive information.

References

Abba Babakura, Md. Nasir Sulaiman and Yusuf, M. (2014). Improved method of classification algorithms for crime prediction. *International Symposium on Biometrics and Security Technologies*. doi:https://doi.org/10.1109/isbast.2014.7013130.

Al-Masri, A. (2022). *Backpropagation in a Neural Network: Explained | Built In*. [online] builtin.com. Available at: https://builtin.com/machine-learning/backpropagation-neural-network.

Almanie, T., Mirza, R. and Lor, E., 2015. Crime prediction based on crime types and using spatial and temporal criminal hotspots. *arXiv preprint arXiv:1508.02050*.

Analytics Vidhya. (2021). *Random Forest Interview Questions | Random Forest Questions*. [online] Available at: https://www.analyticsvidhya.com/blog/2021/05/bagging-25-questions-to-test-your-skills-on-random-forest-algorithm/.

Beheshti, N. (2022). *Cross Validation and Grid Search*. [online] Medium. Available at: https://towardsdatascience.com/cross-validation-and-grid-search-efa64b127c1b.

Bhanumathi, P. and Greeshma, S. (n.d), Crime Data Analysis Using Machine Learning. *International Journal of Research in Engineering, IT and Social Sciences*, ISSN 2250-0588, Impact Factor: 6.565, Volume 12 Issue 07, July 2022, pp. 419-429.

Brownlee, J. (2017). *What is the Difference Between a Parameter and a Hyperparameter?* [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/.

Ch, R., Gadekallu, T.R., Abidi, M.H. and Al-Ahmari, A. (2020). Computational System to Classify Cyber Crime Offenses Using Machine Learning. *Sustainability*, 12(10), p.4087. doi:https://doi.org/10.3390/su12104087.

coderzcolumn.com. (n.d.). *Scikit-Learn - Naive Bayes Classifiers by Sunny Solanki*. [online] Available at: https://coderzcolumn.com/tutorials/machine-learning/scikit-learn-sklearn-naive-bayes#4 [Accessed 3 Mar 2023].

CrimeRate (2022). *UK Crime and Safety Statistics | CrimeRate*. [online] crimerate.co.uk. Available at: https://crimerate.co.uk/.

Gerber, M.S. (2014). Predicting crime using Twitter and kernel density estimation. *Decision Support Systems*, 61, pp.115–125. doi:https://doi.org/10.1016/j.dss.2014.02.003.

Google Cloud. (n.d.). *Overview of hyperparameter tuning | AI Platform Training*. [online] Available at: https://cloud.google.com/ai-platform/training/docs/hyperparameter-tuning-overview.

Google Developers. (n.d.). *Classification: Check Your Understanding (ROC and AUC) | Machine Learning*. [online] Available at: https://developers.google.com/machine-learning/crash-course/classification/check-your-understanding-roc-and-auc [Accessed 10 Feb 2023].

Grandini, M., Bagli, E. and Visani, G. (2020). Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756.*

Guerette, R. T., & Bowers, K. J. (2009). Assessing the extent of crime displacement and diffusion of benefits: A review of situational crime prevention evaluations. *Criminology*, [online] 47(4), pp.1331–1368. doi:https://doi.org/10.1111/j.1745-9125.2009.00177.x.

Investopedia. (2019). *How Discrete Distribution Works*. [online] Available at: https://www.investopedia.com/terms/d/discrete-distribution.asp.

Iqbal, R., Murad, M.A.A., Mustapha, A., Panahy, P.H.S. and Khanahmadliravi, N., 2013. An experimental study of classification algorithms for crime prediction. *Indian Journal of Science and Technology*, *6*(3), pp.4219-4225. doi:https://doi.org/10.17485/ijst/2013/v6i3.6).

Isafiade, O. (n.d.). *Artificial intelligence is used for predictive policing in the US and UK – South Africa should embrace it, too*. [online] The Conversation. Available at: https://theconversation.com/artificial-intelligence-is-used-for-predictive-policing-in-the-us-and-uk-south-africa-should-embrace-it-too-191266.

Jain, A. (n.d.). *Crimes in Boston*. [online] Available at: https://www.kaggle.com/datasets/ankkur13/boston-crime-data [Accessed 5 Jan 2023].

Joseph, V.R. (2022). Optimal ratio for data splitting. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 15(4), pp.531–538. doi:https://doi.org/10.1002/sam.11583.

Lau, T. (2020). *Predictive Policing Explained*. [online] Brennan Center for Justice. Available at: https://www.brennancenter.org/our-work/research-reports/predictive-policing-explained.

Liberty. (n.d.). *Predictive policing*. [online] Available at: https://www.libertyhumanrights.org.uk/fundamental/predictive-policing/.

Maheshwari, V. (2019). *KNN ALGORITHM AND IMPLEMENTATION FROM SCRATCH*. [online] Medium. Available at: https://medium.datadriveninvestor.com/knn-algorithm-and-implementation-from-scratch-b9f9b739c28f.

Mohler, G.O., Short, M.B., Brantingham, P.J., Schoenberg, F.P. and Tita, G.E. (2011). Self-Exciting Point Process Modeling of Crime. *Journal of the American Statistical Association*, 106(493), pp.100–108. doi:https://doi.org/10.1198/jasa.2011.ap09546.

Mujtaba, H. (2020). *An Introduction to Grid Search CV | What is Grid Search*. [online] GreatLearning. Available at: https://www.mygreatlearning.com/blog/gridsearchcv/.

Narkhede, S. (2018). *Understanding AUC - ROC Curve*. [online] Medium. Available at: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

neptune.ai. (2021). *Balanced Accuracy: When Should You Use It?* [online] Available at: https://neptune.ai/blog/balanced-accuracy.

Nyuytiymbiy, K. (2021). *Parameters and Hyperparameters in Machine Learning and Deep Learning*. [online] Medium. Available at: https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac.

Office for National Statistics (2023). *Crime in England and Wales*. [online] Ons.gov.uk. Available at: https://www.ons.gov.uk/peoplepopulationandcommunity/crimeandjustice/bulletins/crimeinenglandandwales/yearendingseptember2022.

OpenGenus IQ. (2023). *50+ Key Terms/ Topics in Deep Learning [Complete DL revision]*. [online] Available at: https://iq.opengenus.org/key-terms-in-deep-learning/ [Accessed 8 Feb 2023].

Pahwa, R. (2017). *Micro-Macro Precision,Recall and F-Score*. [online] Medium. Available at: https://medium.com/@ramit.singh.pahwa/micro-macro-precision-recall-and-f-score-44439de1a044 [Accessed 10 Feb 2023].

pramod, om (2023). *Cross Validation*. [online] Medium. Available at: https://medium.com/@ompramod9921/cross-validation-623620ff84c2.

Safat, W., Asghar, S. and Gillani, S.A. (2021). Empirical Analysis for Crime Prediction and Forecasting Using Machine Learning and Deep Learning Techniques. *IEEE Access*, pp.1–1. doi:https://doi.org/10.1109/access.2021.3078117.

Sakkaf, Y. (2020). *Decision Trees for Classification: ID3 Algorithm Explained*. [online] Medium. Available at: https://towardsdatascience.com/decision-trees-for-classification-id3-algorithm-explained-89df76e72df1.

Schratz, P., Muenchow, J., Iturritxa, E., Richter, J. and Brenning, A. (2019). Hyperparameter tuning and performance assessment of statistical and machine-learning algorithms using spatial data. *Ecological Modelling*, 406, pp.109–120. doi:https://doi.org/10.1016/j.ecolmodel.2019.06.002.

Sciencedirect.com. (2019). *Confusion Matrix - an overview | ScienceDirect Topics*. [online] Available at: https://www.sciencedirect.com/topics/engineering/confusion-matrix.

SciKit-Learn (2009). *3.1. Cross-validation: evaluating estimator performance — scikit-learn 0.21.3 documentation*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/cross_validation.html.

Scikit-learn.org. (2010). *sklearn.metrics.balanced_accuracy_score — scikit-learn 0.21.3 documentation*. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html.

Sherrer, K. (2022). *Google Colab vs Jupyter Notebook: Compare data science software*. [online] TechRepublic. Available at: https://www.techrepublic.com/article/google-colab-vs-jupyter-notebook/.

Stripe, N. (2022). *The impact of crime on victims and society - Office for National Statistics*. [online] www.ons.gov.uk. Available at: https://www.ons.gov.uk/peoplepopulationandcommunity/crimeandjustice/articles/theimpactofcrimeonvictimsandsociety/march2022).

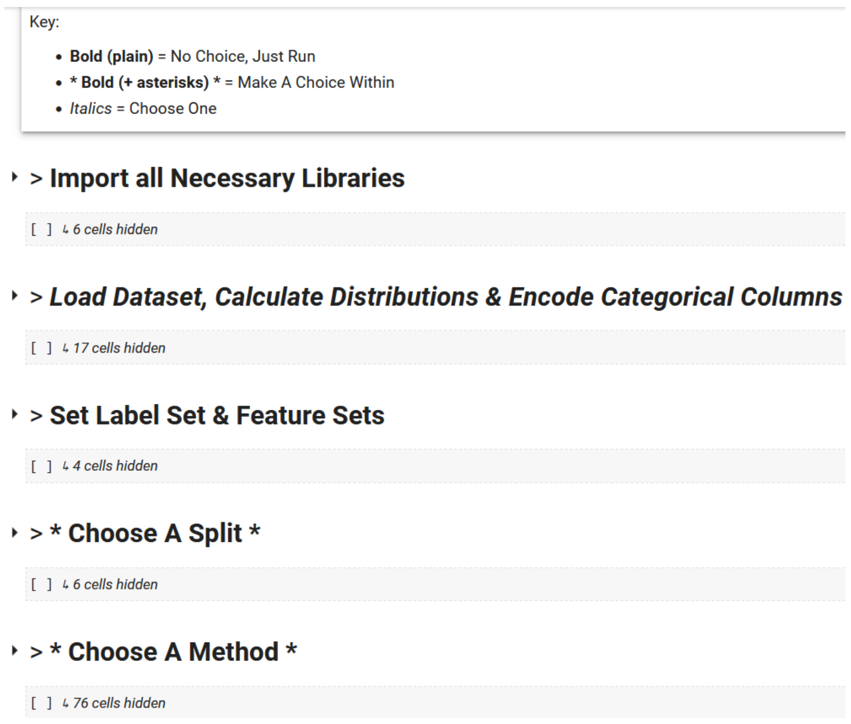
Tharwat, A. (2021). Classification assessment methods. *Applied computing and informatics*, *17*(1), pp.168-192.

www.ibm.com. (n.d.). *What is Machine Learning? | IBM*. [online] Available at: https://www.ibm.com/in-en/topics/machine-learning.

www.kaggle.com. (n.d.). *What is a kaggle? | Data Science and Machine Learning*. [online] Available at: https://www.kaggle.com/general/328265.

Yang, L. and Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415, pp.295–316. doi:https://doi.org/10.1016/j.neucom.2020.07.061.

Appendices



Key:

- **Bold (plain)** = No Choice, Just Run
- *** Bold (+ asterisks) *** = Make A Choice Within
- *Italics* = Choose One

> Import all Necessary Libraries

[ ] ↳ *6 cells hidden*

> Load Dataset, Calculate Distributions & Encode Categorical Columns

[ ] ↳ *17 cells hidden*

> Set Label Set & Feature Sets

[ ] ↳ *4 cells hidden*

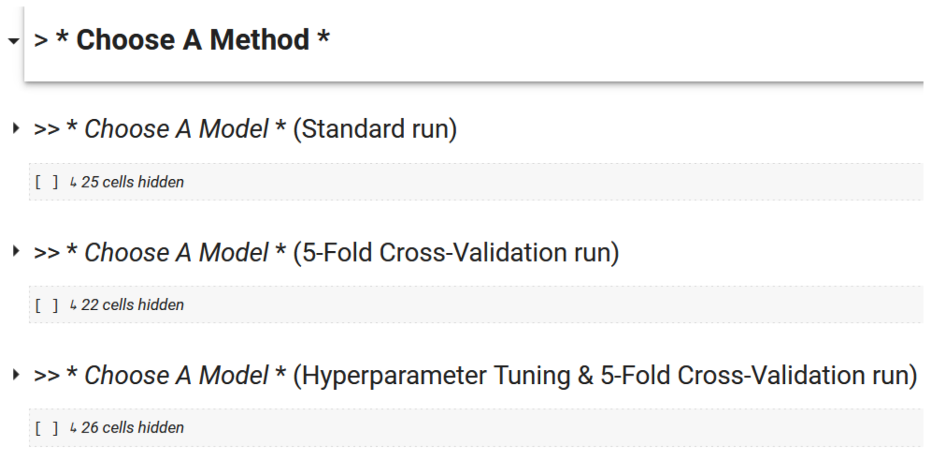
> * Choose A Split *

[ ] ↳ *6 cells hidden*

> * Choose A Method *

[ ] ↳ *76 cells hidden*

Appendix 1: A zoomed-out screenshot of the whole software artefact.



> * Choose A Method *

>> * *Choose A Model* * (Standard run)

[ ] ↳ *25 cells hidden*

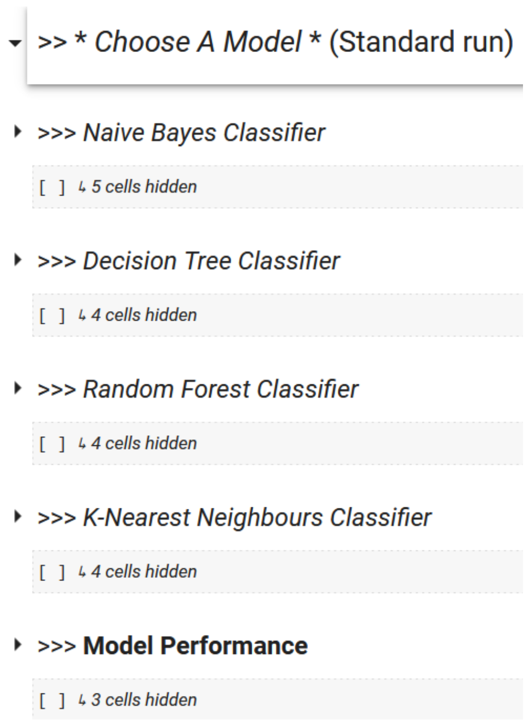
>> * *Choose A Model* * (5-Fold Cross-Validation run)

[ ] ↳ *22 cells hidden*

>> * *Choose A Model* * (Hyperparameter Tuning & 5-Fold Cross-Validation run)

[ ] ↳ *26 cells hidden*

Appendix 2: A screenshot of the inside of the 'Choose A Method' step.


>> * Choose A Model * (Standard run)

>>> Naive Bayes Classifier

[ ] ↳ 5 cells hidden

>>> Decision Tree Classifier

[ ] ↳ 4 cells hidden

>>> Random Forest Classifier

[ ] ↳ 4 cells hidden

>>> K-Nearest Neighbours Classifier

[ ] ↳ 4 cells hidden

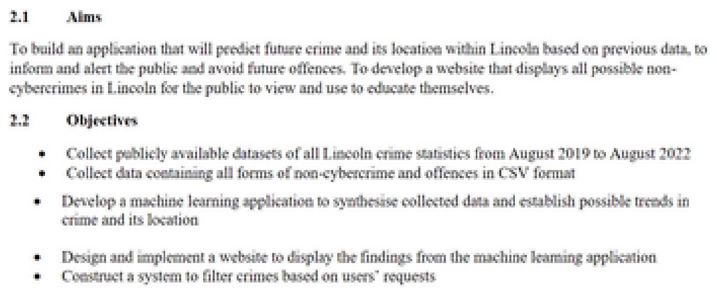
>>> **Model Performance**

[ ] ↳ 3 cells hidden

Appendix 3: A screenshot inside the 'Choose A Model' step.

2.1 Aims, Objectives and Results

The aims and objectives that were set out in the original project proposal can be seen in Figure 1.


2.1 Aims

To build an application that will predict future crime and its location within Lincoln based on previous data, to inform and alert the public and avoid future offences. To develop a website that displays all possible non-cybercrimes in Lincoln for the public to view and use to educate themselves.

2.2 Objectives

- Collect publicly available datasets of all Lincoln crime statistics from August 2019 to August 2022
- Collect data containing all forms of non-cybercrime and offences in CSV format
- Develop a machine learning application to synthesise collected data and establish possible trends in crime and its location
- Design and implement a website to display the findings from the machine learning application
- Construct a system to filter crimes based on users' requests

Fig. 1. Original aims and objectives.

The aims of this project still hold true except rather than predicting the location of a crime, the crime will be predicted based on a location given by the user; it must also be stated that the project is no longer a "Crime Rate Predictor for Lincoln", but rather a "Crime Predictor for Lincoln". There has been a deviation from the first objective, as instead of gathering crime data from August 2019 to August 2022, the data that has been gathered ranges from December 2019 to November 2022; the amount of data is one month lesser in the data collected for this project, but the impact will be negligible. In reference to Figure 1, the machine learning algorithm that was originally chosen and developed was a Decision Tree. However, based on further reading and research, it was concluded that using the Naive Bayes classifier was the superior option to create the outcome that was needed as it has shown to be best suited for the needs of this project. The final objectives outlining the web application are still relevant and have not been changed.

Appendix 4: The original aims and objectives with discussion from the interim report.