1.0 Abstract

This report investigates 3 different Machine Learning (ML) techniques to predict whether an arrest will occur following reported incidents of crime, using a publicly available Chicago crimes dataset comprising 6 million records from 2001 until present. The primary objective is to identify potential disparities in arrest rates across crime types, locations, and police districts, using machine learning models to predict arrest outcomes. The findings aim to support fair and equitable law enforcement practices.

Feature selection was performed using the Chi-squared test, identifying the most statistically significant predictors for the arrest outcome. Three classification models, namely Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Decision Tree, were implemented and evaluated using multiple train-test split ratios. Each model was optimizedā by tuning one key hyperparameter (C for SVM, n_neighbors for KNN, and max_depth for Decision Tree). The effectiveness of each algorithm is evaluated through performance metrics, including accuracy, recall, precision, and F1 score. The report highlights the strengths and weaknesses of each algorithm and aims to determine the most appropriate method for predicting arrest. Finally the report examines the result of each approach and provides recommendations for future research such as exploring distributed solutions or optimised libraries to enhance model performance and training at scale.

2.0 Introduction of K-Nearest Neighbours (KNN), Decision Tree, and Support Vector Machine (SVM)

2.1 K-Nearest Neighbours (KNN)

The K-Nearest Neighbours (KNN) algorithm is one of the most straightforward and widely used machine learning algorithms, particularly for classification tasks. It is a non-parametric, supervised learning technique that classifies new data points based on the proximity to their nearest neighbours in the training data [1]. When predicting the label of a new instance, KNN identifies the K closest instances from the training set—typically using distance metrics such as Euclidean, Manhattan, or Minkowski distance—and assigns the most frequent label among these neighbours to the new point. One of the key strengths of KNN is that it does not involve any

model training; it stores the entire dataset and makes predictions only at runtime, a characteristic that makes it a lazy learning algorithm.

Its simplicity makes KNN particularly useful in situations where the decision boundaries are complex or nonlinear. However, this simplicity comes with trade-offs. Since the algorithm requires scanning the entire dataset during prediction, it can be computationally expensive and memory-intensive, especially for large datasets.

Additionally, KNN's performance is heavily influenced by the choice of K value and the distance metric used. Selecting an inappropriate value for K may result in underfitting or overfitting, making hyperparameter tuning a crucial step in model development [1].

2.2 Decision Tree

Decision Tree is a widely used machine learning algorithm that serves both classification and regression tasks. It represents data in the form of a tree-like model of decisions, where each internal node denotes a feature or attribute, each branch represents a decision rule or outcome of a test, and each leaf node corresponds to a predicted output or class label. The construction of a Decision Tree involves recursively splitting the dataset based on specific criteria such as Information Gain, Gini Index, or variance reduction to determine the most informative features at each step.

A major advantage of Decision Trees lies in their interpretability. The resulting model can be easily visualized and understood, making it highly suitable for domains that require transparency, such as healthcare, criminal justice, and finance. Additionally, Decision Trees are capable of handling both categorical and numerical data and do not require normalization or feature scaling. They also effectively capture nonlinear relationships and feature interactions within the data.

Despite their strengths, Decision Trees have certain limitations. One notable issue is their tendency to overfit, especially when the tree becomes too deep, capturing noise in the training data. This can lead to poor generalization on unseen data. Pruning techniques, which remove branches that do not contribute significantly to model performance, are commonly used to address overfitting. Another challenge is their instability; small changes in the input data can result in vastly different tree structures.

Overall, Decision Trees offer a balance between simplicity and predictive power, making them a foundational tool in the machine learning toolkit.

2.3 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for both classification and regression tasks. It constructs a hyperplane or set of hyperplanes in a high-dimensional space to effectively separate data points into distinct categories. The key idea behind SVM is maximizing the margin between different classes, ensuring robust generalization to unseen data. When data is not linearly separable, SVM utilizes kernel functions such as linear, radial basis function (RBF), and sigmoid to transform the input space and make classification possible.

One of the greatest advantages of SVM is its ability to handle high-dimensional data with relatively few samples, making it suitable for applications like text classification, image recognition, and bioinformatics. Unlike algorithms like Decision Trees, SVM excels in situations where data is complex and requires a clear decision boundary. It is also resistant to overfitting due to the hyperparameter C, which balances margin width with classification accuracy.

Despite its strengths, SVM has some drawbacks. It can be computationally expensive, particularly with large datasets, as the optimization process involves solving quadratic programming problems. We have only used linear kernel to sustain long training hours.Additionally, selecting the right kernel and tuning hyperparameters significantly impact performance, requiring careful experimentation. SVM models also lack interpretability compared to decision trees, making them less transparent in some critical domains.

Overall, SVM is a highly effective machine learning tool for complex classification problems, particularly when data is sparse or high-dimensional. With proper tuning, it remains a powerful choice for predictive modeling across various fields.

3.0 Objectives

The objective of this project is to determine the most effective machine learning model in distinguishing whether a reported crime will result in an arrest or not based on various attributes recorded by the Chicago Police Department. Three machine learning (ML) models will be utilised for this task, namely **K-Nearest Neighbour (KNN)**, **Decision Tree**, and **Support Vector Machine (SVM).** This report will go through the steps in preparing the data, building the models and evaluating it. The goal is to understand the process so as to tweak the code to maximise the model and minimise the disadvantages. This report is structured as a factorial experiment with 2 independent variables, namely the train-test split ratio and the number of features, N.

Factorial experimentation will see N and train-test partitions changing to explore all possible configurations and determine the best performing model. The values of N = 6, 8, 10 while the train-test split ratio is 60/40, 70/30 and 80/20. The combination of both variables made 9 possible configurations across each ML model. The rationale of such experimentation setup is to control and isolate the effect of changing the train-test split while keeping the number of features constant and vice versa.

Meanwhile, a hyperparameter is fine-tuned individually for each model and configuration to obtain its optimal value. By determining the best-performing hyperparameter for each configuration, we ensure consistency across each configuration. This style of experimentation helps in identifying the combination of split and N features that gives the best performant model based on certain metrics.

After these individual experimentations, model evaluation is performed using accuracy, weighted recall, weighted precision and weighted F1 score as primary performance indicators. Particular emphasis is placed on recall, given the potential public safety implications of misclassifying incidents that should lead to arrests. A model with high recall reduces the likelihood of false negatives, that is, situations where incidents that should result in arrests are wrongly predicted otherwise. In criminal justice and public safety contexts, ensuring that arrest-worthy incidents are correctly identified is essential for operational effectiveness and community trust in law enforcement interventions [1].

The champion model will be selected based on a balanced consideration of accuracy, recall, and F1 score, with priority given to recall and accuracy to ensure both overall reliability and critical case detection.

4.0 Justifications

4.1 K-Nearest Neighbours (KNN)

The K-Nearest Neighbours (KNN) algorithm is employed in this analysis due to its simplicity and effectiveness in supervised machine learning tasks. It classifies data points by examining the features of their nearest neighbours and assigning them the majority label among those neighbours. KNN is both intuitive and straightforward, making it an ideal choice for initial dataset analysis and serving as a foundational method before progressing to more complex algorithms [2].

The final dataset used in the analysis consists primarily of numerical data, with categorical features converted into numerical values using label encoding. The target variable, 'ARREST', is represented in boolean form, where 1 indicates an arrest and 0 indicates no arrest. KNN is well suited to handle numerical data and was used to train models with the top 10, 8, 6, and 2 features, as identified by a chi-squared feature selection test.

The objective is to predict whether an arrest would occur based on factors such as BLOCK, IUCR, FBI_CODE, WARD etc. KNN's ability to perform both classification and regression tasks makes it a versatile and effective choice for this problem.

KNN can achieve high performance when provided with sufficient data. Given the dataset contains over 6.4 million rows (nearly 140 million entries), KNN can leverage this large volume of data to achieve high performance.

Moreover, to further improve the effectiveness of the model, StandardScaler was employed. It standardized the features by removing the mean and scaling to unit variance. This ensures that all features contribute equally to the model, preventing those with larger ranges from dominating the distance calculations [3]. This is because KNN may become biased toward features with larger numerical values, leading to poor classification accuracy. By scaling the data, StandardScaler

improves the reliability and performance of the KNN algorithm and ensures fair and effective learning from all features [3][4].

Finally, KNN's nonparametric nature, meaning it does not assume any specific distribution of the input data, adds to its flexibility and makes it suitable for various practical applications, especially when working with diverse and complex datasets [2].

4.2 Decision Tree

A Decision Tree classifier was employed to analyze and predict the binary outcome variable "ARREST", which indicates whether an arrest occurred in each recorded crime incident. This algorithm was selected for its interpretability, flexibility with both categorical and numerical features, and its ability to model non-linear relationships without the need for normalization or scaling. The crime dataset used contains multiple attributes including location-based details, time-related features, and crime types—making it well-suited for a model that can naturally handle complex branching decisions like the Decision Tree. Compared to other models evaluated in this study (i.e., K-Nearest Neighbors and Support Vector Machine), the Decision Tree offers better transparency in decision-making, which is especially valuable in policy-driven domains such as law enforcement, where understanding how a prediction was made can inform public safety strategies. To ensure robust model selection, multiple experiments were conducted by varying the number of input features (6, 8, and 10) and the data split ratios (60:40, 70:30, and 80:20 for training and testing). The value of the maximum depth (max_depth) hyperparameter was determined by observing its effect on model accuracy across different configurations. Smaller depths were used to reduce overfitting and maintain simplicity, while larger depths allowed the model to capture more complex patterns in the data. A range of representative values was chosen to strike a balance between underfitting and overfitting, enabling the identification of a depth that provides the best generalization performance. Each model was then evaluated using standard classification metrics including accuracy, weighted precision, weighted recall, weighted F1-score, and the confusion matrix. This methodological approach not only enabled a structured comparison across configurations but also revealed how varying tree depth and feature richness influences the predictive capabilities of Decision Trees in crime-related classification tasks.

4.3 Support Vector Machine (SVM)

Support Vector Machines (SVMs) have been selected as one of the machine learning algorithms for training on the "crime.csv" dataset, which contains over one million records and approximately 1.5 GB of data. The primary objective of this classification task is to accurately predict whether an arrest was made based on various features related to the nature, location, and context of a reported crime. One of the key motivations for employing SVMs, specifically the LinearSVC implementation, is their strong performance in high-dimensional spaces. The dataset includes multiple categorical variables such as PRIMARY_TYPE, DESCRIPTION, LOCATION_DESCRIPTION, and FBI_CODE, which, after one-hot encoding, significantly increase the feature space. Linear SVMs are particularly effective in such high-dimensional settings, where simpler algorithms may struggle due to the curse of dimensionality [9].
Instead of using the standard SVC class with a linear kernel which can be computationally expensive on large datasets, we opted for LinearSVC, which is specifically designed for linear classification tasks and is much more resource-efficient. LinearSVC leverages the liblinear optimization library, making it significantly faster and more scalable for large-scale datasets like crime.csv [10]. Since the kernel function is not being manipulated and is kept linear, LinearSVC offers the same decision-making capabilities as SVC(kernel='linear'), but with superior performance on high-volume data. Furthermore, the task at hand is a binary classification problem which predicts whether an arrest occurred (TRUE/FALSE), which aligns well with the core design of SVMs. SVM seeks to find the optimal separating hyperplane that maximizes the margin between the two classes, offering a theoretically grounded approach for achieving high classification accuracy, especially when the decision boundary is approximately linear. The use of class_weight='balanced' in the SVM configuration further strengthens its suitability for this task by addressing potential class imbalance in the arrest variable. This ensures that the model does not favor the majority class and maintains fair performance across both classes.
Overall, the selection of SVM via LinearSVC is justified by its suitability for high-dimensional and large-scale data, its effectiveness in binary classification tasks, its computational advantages, and its demonstrated empirical performance. These characteristics make it a practical and theoretically sound choice for accurately predicting arrest outcomes based on complex urban crime data.

5.0 Steps to Build the Machine Learning Models

5.1 Import the libraries and data

The libraries imported in the code are largely for 2 main reasons, mainly for data manipulation and training machine learning. Libraries including math, numpy, pandas, matplotlib and seaborn are used to perform data processing, data cleaning, visualisation and analysis tasks. These libraries are crucial for building a comprehensive understanding of the dataset and preparing the data for model building. Furthermore, the scikit-learn library is used for selecting the best models, building models and evaluating the models. It is useful for splitting the dataset into test and train sets, constructing different models and evaluating their performance based on several key metrics.

```
import math  # for mathematical operations
import numpy as np  # for numerical operations
import pandas as pd  # for data manipulation
import matplotlib.pyplot as plt  # for data visualisation
import seaborn as sbn  # for data visualisation

from sklearn.model_selection import train_test_split, GridSearchCV  # for splitting dataset into train and test sets and hyperparameter tuning
from sklearn.feature_selection import SelectKBest, chi2  # for selecting the best features
from sklearn.svm import LinearSVC  # for SVM model
from sklearn.tree import DecisionTreeClassifier  # for Decision Tree model
from sklearn.neighbors import KNeighborsClassifier  # for KNN model
from sklearn.preprocessing import LabelEncoder, StandardScaler  # for preprocessing data to fit models
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score  # for model evaluation
```

Screenshot 1: The libraries used in the code

5.2 EDA Analysis

5.2.1 Loading dataset

The dataset named “crimes.csv” is loaded into a dataframe. It is sourced from Kaggle, the world’s largest data science community which provides data for research and analysis purposes. The dataset is licensed under the Creative Commons Organisation (COO): Public Domain [5], ensuring the accuracy and validity of its data origin.

This dataset is extracted from the authoritative Chicago Police Department. This dataset reflects the reported incident of crimes that occured in the City of Chicago since 2001 using the CLEAR

(Citizen Law Enforcement Analysis and Reporting) reporting system. This dataset highlights the urgency and the scale of violence as the Chicago Police Department highlights approximately 10 people are shot on average day in Chicago.

Dataset: crimes.csv

Dataset Description:

- **unique_key:** Unique identifier for the record.
- **case_number:** The Chicago Police Department RD Number (Records Division Number), which is unique to the incident.
- **date:** Date when the incident occurred. This is sometimes a best estimate.
- **block:** The partially redacted address where the incident occurred, placing it on the same block as the actual address.
- **iucr:** The Illinois Uniform Crime Reporting code. This is directly linked to the Primary Type and Description.
- **primary_type:** The primary description of the IUCR code.
- **description:** The secondary description of the IUCR code, a subcategory of the primary description.
- **location_description:** Description of the location where the incident occurred.
- **domestic:** Indicates whether the incident was domestic-related as defined by the Illinois Domestic Violence Act.

Targeted column:

- **arrest:** Indicates whether an arrest was made.

5.2.2 Exploratory Data Analysis (EDA)

1. **Initial Analysis of the DataFrame**

By using *df.head()* statement, the first 5 rows are shown, allowing for easy identification of the features and target column along with some exemplary data. This indicates the successful loading of the dataset and may proceed further for data manipulation, analysis, etc.

```
df.head()  # View the first 5 rows
```

	ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	Domestic
0	3753774	HK769786	11/23/2004 10:15:00 PM	029XX S STATE ST	2024	NARCOTICS	POSS: HEROIN(WHITE)	CHA PARKING LOT/GROUNDS	True	False
1	3753775	HL121290	01/12/2005 08:15:00 PM	111XX S STATE ST	1330	CRIMINAL TRESPASS	TO LAND	GAS STATION	True	False
2	3753776	HL122737	01/13/2005 02:45:00 PM	104XX S WALLACE ST	1320	CRIMINAL DAMAGE	TO VEHICLE	PARKING LOT/GARAGE(NON.RESID.)	False	False
3	3753777	HL121030	01/12/2005 05:35:00 PM	037XX W ALTGELD ST	0486	BATTERY	DOMESTIC BATTERY SIMPLE	APARTMENT	False	True
4	3753781	HK756758	11/17/2004 04:45:00 PM	002XX S ALBANY AVE	2024	NARCOTICS	POSS: HEROIN(WHITE)	SIDEWALK	True	False

Beat	District	Ward	Community Area	FBI Code	X Coordinate	Y Coordinate	Year	Updated On	Latitude	Longitude	Location
2113	1.0	3.0	35.0	18	1176755.0	1885570.0	2004	04/15/2016 08:55:02 AM	41.841350	-87.626861	(41.841350479, -87.626860911)
522	5.0	34.0	49.0	26	1178229.0	1831287.0	2005	04/15/2016 08:55:02 AM	41.692359	-87.623097	(41.692358646, -87.623096746)
2233	22.0	34.0	49.0	14	1174150.0	1835631.0	2005	04/15/2016 08:55:02 AM	41.704371	-87.637902	(41.704370559, -87.637902149)
2524	25.0	35.0	22.0	08B	1151164.0	1916359.0	2005	04/15/2016 08:55:02 AM	41.926378	-87.719964	(41.926378017, -87.719963957)
1124	11.0	28.0	27.0	18	1155752.0	1898738.0	2004	04/15/2016 08:55:02 AM	41.877933	-87.703580	(41.877933128, -87.703580435)

Screenshot 2: Display of the first 5 rows

2. Dimension of the DataFrame

The dimension of the dataframe is a whopping 6.5 million rows and 22 features. The feature includes ‘ID , 'Case Number', 'Date', 'Block', 'IUCR', 'Primary Type', 'Description', 'Location Description', 'Arrest', 'Domestic', 'Beat', 'District', 'Ward', 'Community Area', 'FBI Code', 'X Coordinate', 'Y Coordinate', 'Year', 'Updated On', 'Latitude', 'Longitude' and 'Location'.

```
df.shape  # Look at the dimensions of the dataframe
```

```
(6446454, 22)
```

Screenshot 3: df.shape indicates the dimensions of the DataFrame

3. Feature Data Types

Using df.dtypes statement, it is shown that there are 10 objects, 3 int64, 2 boolean and 7 float64 columns. The dataset uses a sizable memory usage of 780.4 MB.

```
df.info() #provides a concise summary of a DataFrame.
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6443602 entries, 0 to 6446453
Data columns (total 22 columns):
 #   Column                Dtype
---  ------                -----
 0   BLOCK                 int32
 1   IUCR                  int32
 2   PRIMARY_TYPE          int32
 3   DESCRIPTION           int32
 4   LOCATION_DESCRIPTION  int32
 5   ARREST                bool
 6   DOMESTIC              bool
 7   BEAT                  int64
 8   DISTRICT              float64
 9   WARD                  float64
 10  COMMUNITY_AREA        float64
 11  FBI_CODE              int32
 12  YEAR                  int64
 13  LATITUDE              float64
 14  LONGITUDE             float64
 15  UPDATED_YEAR          int32
 16  UPDATED_MONTH         int32
 17  UPDATED_DAYOFWEEK     int32
 18  UPDATED_HOUR          int32
 19  UPDATED_ISWEEKEND     int64
 20  UPDATED_WEEKOFYEAR    UInt32
 21  UPDATED_SEASON        int64
dtypes: UInt32(1), bool(2), float64(5), int32(10), int64(4)
memory usage: 780.4 MB
```

Screenshot 4: The corresponding data type to its feature column and the DataFrame's memory usage.

5.3 Data Preparation

Standardise column names

```
df.columns = df.columns.str.strip().str.upper().str.replace(' ', '_')  # standardizing column names
```

Screenshot 5: The statement to standardise column names

The columns are standardized so that every letter is in uppercase and space is replaced with underscore. This is to ensure consistency which may have mixed cases and unexpected extra spaces and characters.

Convert data types

```
#necessary data type conversions
df['DATE'] = pd.to_datetime(df['DATE'], format="%m/%d/%Y %I:%M:%S %p")
df['UPDATED_ON'] = pd.to_datetime(df['UPDATED_ON'])
```

Screenshot 6: The statements to parse dates

Moving on, the dates are parsed from string representations into pandas datetime objects. This enables datetime-based operations like extracting year, month and day which is critical for crime trends over certain periods.

Check duplicates

```
df.duplicated().sum()
```

```
0
```

Screenshot 7: Checking for any duplicates

This step ensures that there is no human error during the reporting process and none were found.

Handle missing values

```
df.isnull().sum()
```

```
ID                          0
CASE_NUMBER                 4
DATE                        0
BLOCK                       0
IUCR                        0
PRIMARY_TYPE                0
DESCRIPTION                 0
LOCATION_DESCRIPTION     2799
ARREST                      0
DOMESTIC                    0
BEAT                        0
DISTRICT                   49
WARD                   614854
COMMUNITY_AREA         616030
FBI_CODE                    0
X_COORDINATE            84729
Y_COORDINATE            84729
YEAR                        0
UPDATED_ON                  0
LATITUDE                84729
LONGITUDE               84729
LOCATION                84729
dtype: int64
```

Screenshot 8: The distribution of uneven null values across different columns

```
df = df.dropna(subset=['CASE_NUMBER', 'LOCATION_DESCRIPTION', 'DISTRICT'])
```

Screenshot 9: The statement to drop the 3 columns

Based on the data presented in Screenshot 8, the columns CASE_NUMBER, LOCATION_DESCRIPTION, and DISTRICT contain an insignificant number of null values relative to the dataset size. Consequently, as shown in Screenshot 9, these columns have been removed to streamline the dataset. However, seven remaining columns still exhibit substantial amounts of missing data. Notably, coordinates, latitude, longitude, and location share similar proportions of null values, indicating a possible correlation between them, suggesting they may represent the same spatial points using different coordinate systems. To validate this, a correlation analysis is required.

```
cols = ['X_COORDINATE', 'Y_COORDINATE', 'LATITUDE', 'LONGITUDE']
print(df[cols].corr())
```

```
              X_COORDINATE  Y_COORDINATE  LATITUDE  LONGITUDE
X_COORDINATE      1.000000     -0.385476 -0.386814   0.999770
Y_COORDINATE     -0.385476      1.000000  0.999993  -0.384547
LATITUDE         -0.386814      0.999993  1.000000  -0.385957
LONGITUDE         0.999770     -0.384547 -0.385957   1.000000
```

Screenshot 10: The result of correlation between x and y coordinates, latitude and longitude

Based on screenshot 10, the correlation between X_COORDINATE with LONGITUDE and Y_COORDINATE with LATITUDE is shown to have near perfect correlation. Hence, X_COORDINATE and Y_COORDINATE are dropped along with LOCATION which store both of the coordinates information as a tuple.

```
# for each BLOCK, find the most frequent WARD value (mode).
# if there are multiple modes, select the first one.
# if no mode exists for the block, assigns null value.
ward_mode_per_block = df.groupby('BLOCK')['WARD'].apply(lambda x: x.mode().iloc[0] if not x.mode().empty else np.nan)

# for each row, if the WARD value is missing, replace it with the mode of that row's BLOCK.
# otherwise, keep the original WARD value.
df['WARD'] = df.apply(lambda row: ward_mode_per_block[row['BLOCK']] if pd.isna(row['WARD']) else row['WARD'], axis=1)
df['WARD'] = df.apply(lambda row: ward_mode_per_block[row['BLOCK']] if pd.isna(row['WARD']) else row['WARD'], axis=1)

# for each BLOCK, find the most frequent COMMUNITY_AREA value (mode).
# if there are multiple modes, select the first one.
# if no mode exists for the block, assigns null value.
community_mode_per_block = df.groupby('BLOCK')['COMMUNITY_AREA'].apply(lambda x: x.mode().iloc[0] if not x.mode().empty else np.nan)

# for each row, if the COMMUNITY_AREA value is missing, replace it with the mode of that row's BLOCK.
# otherwise, keep the original COMMUNITY_AREA value.
df['COMMUNITY_AREA'] = df.apply(lambda row: community_mode_per_block[row['BLOCK']] if pd.isna(row['COMMUNITY_AREA']) else row['COMMUNITY_AREA'], axis=1)
```

```
# Compute overall mode for WARD and COMMUNITY_AREA
overall_ward_mode = df['WARD'].mode().iloc[0]
overall_community_mode = df['COMMUNITY_AREA'].mode().iloc[0]

# replace any remaining null values in WARD and COMMUNITY_AREA with the overall mode
df['WARD'] = df['WARD'].fillna(overall_ward_mode)
df['COMMUNITY_AREA'] = df['COMMUNITY_AREA'].fillna(overall_community_mode)
```

Screenshot 11: Block-based imputation for WARD and COMMUNITY_AREA missing values

Given BLOCK has no null values, the missing values of WARD and COMMUNITY_AREA could be imputed to BLOCK. Crime trends tend to cluster by BLOCK. If a BLOCK typically belongs to a certain WARD or COMMUNITY_AREA, it's reasonable to assume the missing value should follow the mode within that BLOCK. On the other hand, it is possible that there are remaining cells where block-based imputation was impossible such as when there is no existing data on WARD and COMMUNITY_AREA for certain BLOCK. In which case, it will be filled with null.

```python
# For each BLOCK, find the most frequent LATITUDE value (mode).
# If there are multiple modes, select the first one.
# If no mode exists for the block, assign a null value.
lat_mode_per_block = df.groupby('BLOCK')['LATITUDE'].apply(lambda x: x.mode().iloc[0] if not x.mode().empty else np.nan)

# For each row, if the LATITUDE value is missing, replace it with the mode of that row's BLOCK.
# Otherwise, keep the original LATITUDE value.
df['LATITUDE'] = df.apply(lambda row: lat_mode_per_block[row['BLOCK']] if pd.isna(row['LATITUDE']) else row['LATITUDE'], axis=1)

# For each BLOCK, find the most frequent LONGITUDE value (mode).
# If there are multiple modes, select the first one.
# If no mode exists for the block, assign a null value.
lon_mode_per_block = df.groupby('BLOCK')['LONGITUDE'].apply(lambda x: x.mode().iloc[0] if not x.mode().empty else np.nan)

# For each row, if the LONGITUDE value is missing, replace it with the mode of that row's BLOCK.
# Otherwise, keep the original LONGITUDE value.
df['LONGITUDE'] = df.apply(lambda row: lon_mode_per_block[row['BLOCK']] if pd.isna(row['LONGITUDE']) else row['LONGITUDE'], axis=1)
```

```python
# Compute overall mode for LATITUDE and LONGITUDE
overall_lat_mode = df['LATITUDE'].mode().iloc[0]
overall_lon_mode = df['LONGITUDE'].mode().iloc[0]

# Replace any remaining null values in LATITUDE and LONGITUDE with the overall mode
df['LATITUDE'] = df['LATITUDE'].fillna(overall_lat_mode)
df['LONGITUDE'] = df['LONGITUDE'].fillna(overall_lon_mode)
```

```python
# fill null LOCATION values with (LATITUDE, LONGITUDE) tuple
df['LOCATION'] = df.apply(lambda row: (float(row['LATITUDE']), float(row['LONGITUDE'])) if pd.isna(row['LOCATION']) else row['LOCATION'], axis=1)
```

Screenshot 12: Block-based imputation for LATITUDE and LONGITUDE missing values

Similar concepts were applied on LATITUDE and LONGITUDE. The mode of LATITUDE and LONGITUDE were imputed based on BLOCK, else null values are filled. These data are crucial for spatial analysis which takes in a reasonable assumption that crimes on the same block are geographically close.

```
ID                    0
CASE_NUMBER           0
DATE                  0
BLOCK                 0
IUCR                  0
PRIMARY_TYPE          0
DESCRIPTION           0
LOCATION_DESCRIPTION  0
ARREST                0
DOMESTIC              0
BEAT                  0
DISTRICT              0
WARD                  0
COMMUNITY_AREA        0
FBI_CODE              0
YEAR                  0
UPDATED_ON            0
LATITUDE              0
LONGITUDE             0
LOCATION              0
```

Screenshot 13: Result of handling missing values

As per screenshot 13, missing values are non-existent after the techniques applied just now.

Normalise the categorical text data

Although LOCATION_DESCRIPTION, PRIMARY_TYPE, and DESCRIPTION are categorical fields rather than entirely unstructured text, they contain inconsistencies such as typographical errors and semantically similar values represented differently. For example, entries like "HOSPITAL" and "HOSPITAL BUILDING/GROUNDS" convey the same concept but are treated as distinct categories. Normalizing such text data enhances consistency, reduces noise, and lowers dimensionality—ultimately improving model interpretability and robustness. Additionally, generalizing niche or sparsely populated categories (e.g., "ELECTRONIC_STORE") into broader semantic groups can mitigate data sparsity issues, leading to more efficient encoding, faster training times, and reduced memory consumption.

Normalisation of LOCATION_DESCRIPTION column

```
location_mapping = {
    'ABANDONED BUILDING': 'ABANDONED_BUILDING',
    'AIRCRAFT': 'AIRPORT',
    'AIRPORT BUILDING NON-TERMINAL - NON-SECURE AREA': 'AIRPORT',
    'AIRPORT BUILDING NON-TERMINAL - SECURE AREA': 'AIRPORT',
    'AIRPORT EXTERIOR - NON-SECURE AREA': 'AIRPORT',
    'AIRPORT EXTERIOR - SECURE AREA': 'AIRPORT',
    'AIRPORT PARKING LOT': 'AIRPORT',
    'AIRPORT TERMINAL LOWER LEVEL - NON-SECURE AREA': 'AIRPORT',
    'AIRPORT TERMINAL LOWER LEVEL - SECURE AREA': 'AIRPORT',
    'AIRPORT TERMINAL MEZZANINE - NON-SECURE AREA': 'AIRPORT',
    'AIRPORT TERMINAL UPPER LEVEL - NON-SECURE AREA': 'AIRPORT',
    'AIRPORT TERMINAL UPPER LEVEL - SECURE AREA': 'AIRPORT',
    'AIRPORT TRANSPORTATION SYSTEM (ATS)': 'AIRPORT',
    'AIRPORT VENDING ESTABLISHMENT': 'AIRPORT',
    'AIRPORT/AIRCRAFT': 'AIRPORT',
```

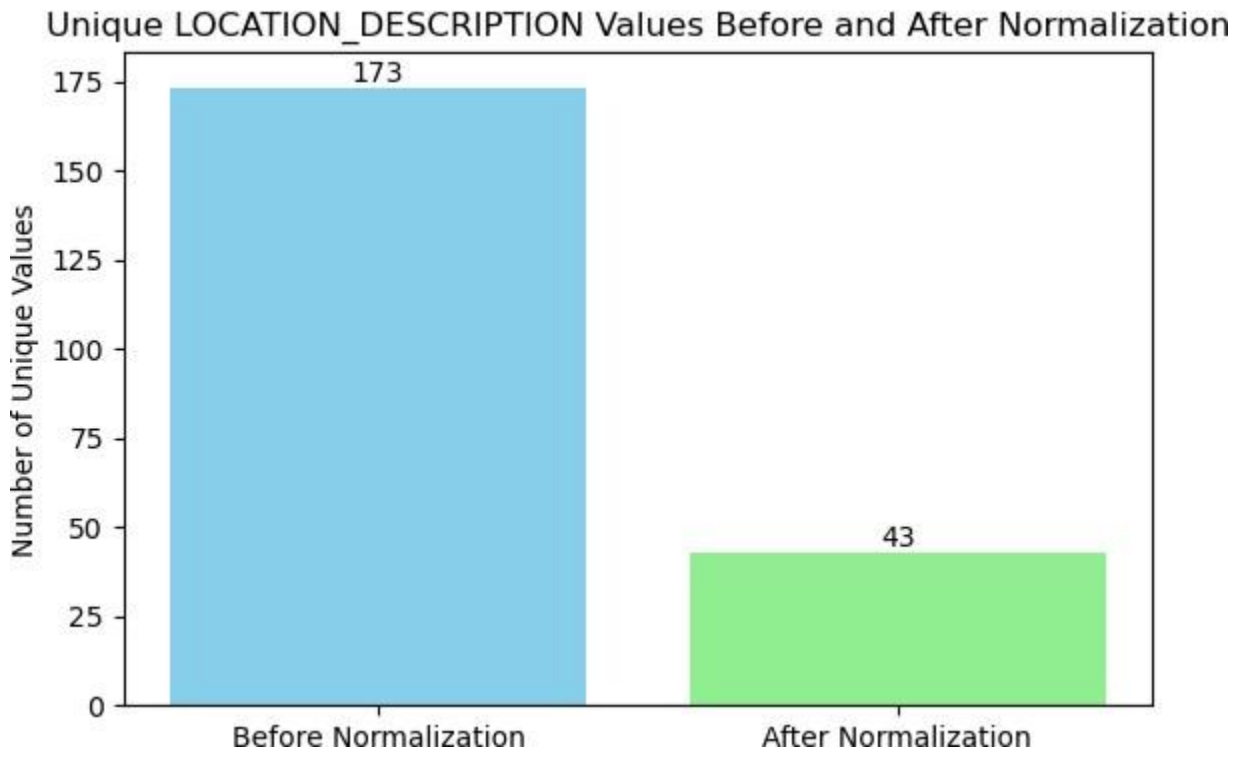
Screenshot 14: Examples of the mapping of LOCATION_DESCRIPTION values

```
# Generalise LOCATION_DESCRIPTION by mapping each value to a new value and retain original results if value not in mapping
df['LOCATION_DESCRIPTION'] = df['LOCATION_DESCRIPTION'].map(location_mapping).fillna(df['LOCATION_DESCRIPTION'])
```

```
#defining a function that normalized the values for Location Discription if any non-standardized values are left
def normalize_text(text):
    if pd.isna(text):
        return text
    text = text.upper()
    text = text.replace(' / ', '/')
    text = text.replace(' - ', '-')
    return text.strip()

#using the function
df['LOCATION_DESCRIPTION'] = df['LOCATION_DESCRIPTION'].apply(normalize_text)
```

Screenshot 15: Applying the mapping and normalise the values



Screenshot 16: The unique values before-and-after comparison for LOCATION_DESCRIPTION

The value of LOCATION_DESCRIPTION is replaced with the new generalised values. If the mapping does not include any of the original value, they are retained with the original value. The leftover non-standardised values will be turned into uppercase and standardised. The result yields 43 unique values compared to the 173 original unique values, indicating a significant reduction in dimensions and complexity.

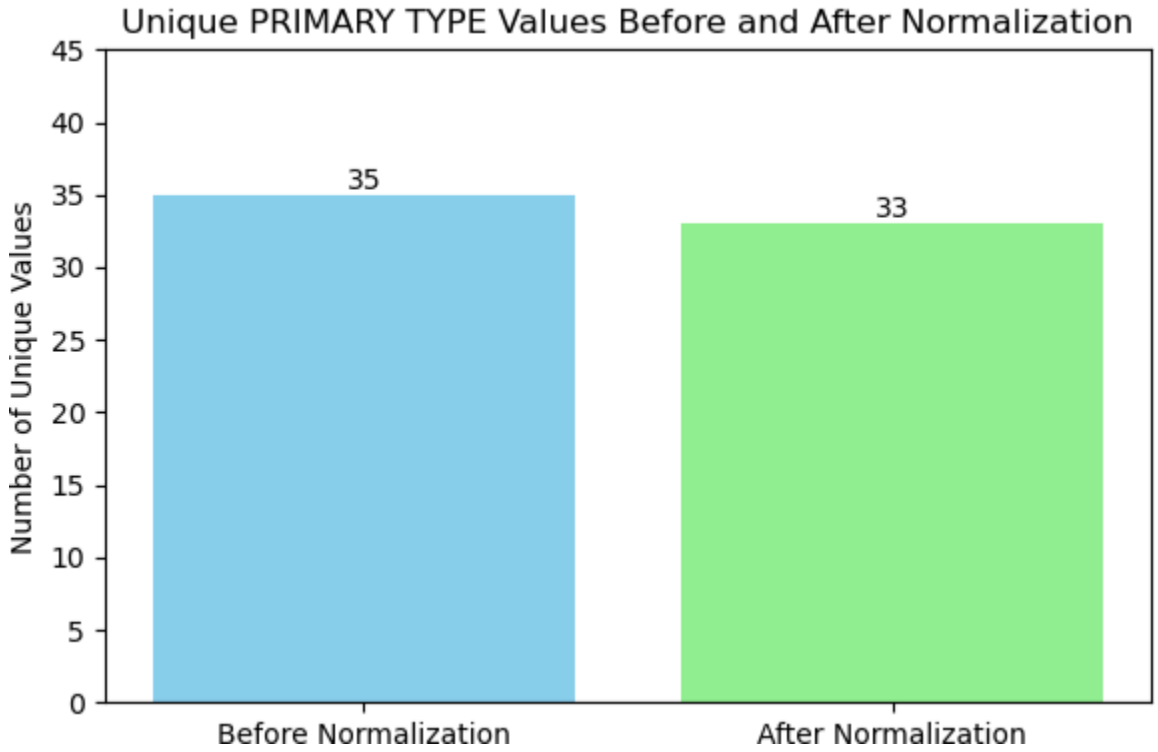
Normalisation of PRIMARY_TYPE column

```
#defining a function that normalized the values for primary type if any non-standardized values are left
def normalize_text(text):
    if pd.isna(text):
        return text
    text = text.upper()
    text = text.replace(' - ', '_')
    text = text.replace('-', '_')
    text = text.replace(' ', '_')
    text = text.replace('NON_CRIMINAL_(SUBJECT_SPECIFIED)', 'NON_CRIMINAL')

    return text.strip()

#using the function
before_primary_type = df['PRIMARY_TYPE'].nunique()
df['PRIMARY_TYPE'] = df['PRIMARY_TYPE'].apply(normalize_text)
after_primary_type = df['PRIMARY_TYPE'].nunique()
```

Screenshot 17: Applying standardisation rules on PRIMARY_TYPE column



Screenshot 18: The unique values before-and-after comparison for PRIMARY_TYPE

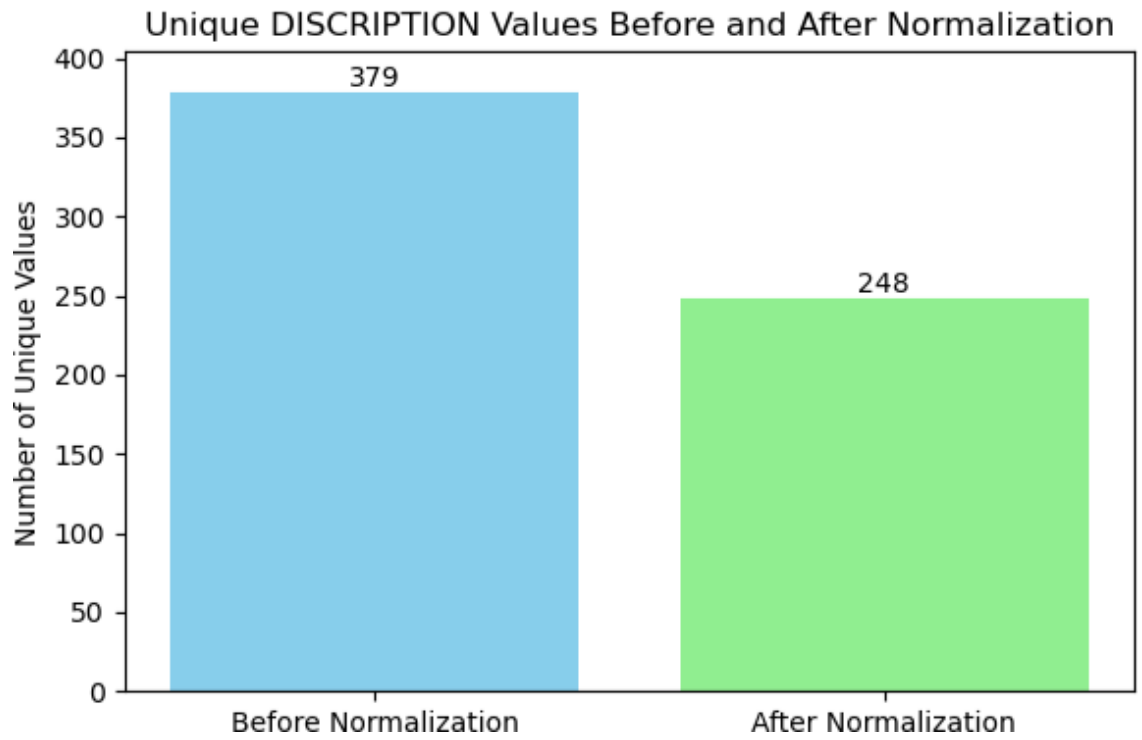
The original values of the PRIMARY_TYPE column is standardised so that every case is uppercase, the underscores, hyphens and spacings are standardised and a specific token is

transformed into a generalised category. Then the value is stripped of its spaces. The result yields 33 unique values compared to 35 unique values prior to normalisation.

Normalisation of DESCRIPTION column

```
crime_mapping = {
    # MANU/DELIVER related
    'MANU/DEL:CANNABIS 10GM OR LESS': 'MANU_DELIVER_CANNABIS',
    'MANU/DEL:CANNABIS OVER 10 GMS': 'MANU_DELIVER_CANNABIS',
    'MANU/DELIVER: HALLUCINOGEN': 'MANU_DELIVER_DRUGS',
    'MANU/DELIVER: HEROIN (WHITE)': 'MANU_DELIVER_HEROIN',
    'MANU/DELIVER: HEROIN(BRN/TAN)': 'MANU_DELIVER_HEROIN',
    'MANU/DELIVER: METHAMPHETAMINES': 'MANU_DELIVER_DRUGS',
    'MANU/DELIVER:AMPHETAMINES': 'MANU_DELIVER_DRUGS',
    'MANU/DELIVER:BARBITUATES': 'MANU_DELIVER_DRUGS',
    'MANU/DELIVER:COCAINE': 'MANU_DELIVER_COCAINE',
    'MANU/DELIVER:CRACK': 'MANU_DELIVER_CRACK',
    'MANU/DELIVER:HEROIN(BLACK TAR)': 'MANU_DELIVER_HEROIN',
    'MANU/DELIVER:LOOK-ALIKE DRUG': 'MANU_DELIVER_DRUGS',
    'MANU/DELIVER:PCP': 'MANU_DELIVER_DRUGS',
    'MANU/DELIVER:SYNTHETIC DRUGS': 'MANU_DELIVER_DRUGS',
    'MANU/POSS. W/INTENT TO DELIVER: SYNTHETIC MARIJUANA': 'MANU_DELIVER_SYNTHETIC_MARIJUANA',
```

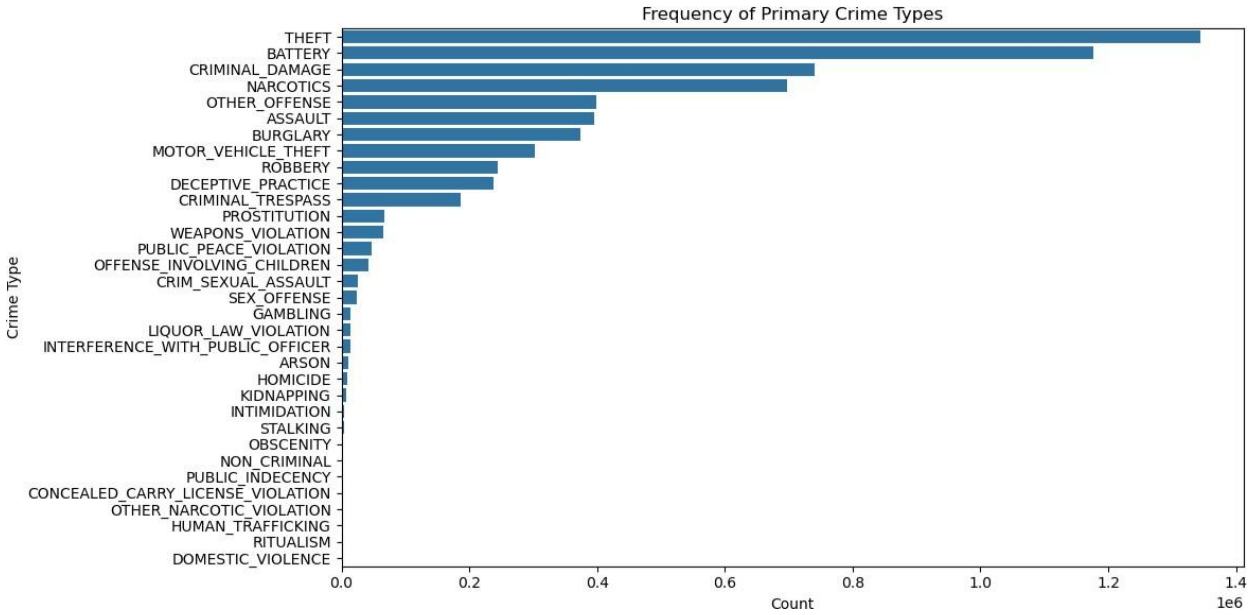
Screenshot 19: Examples of mapping of DESCRIPTION values

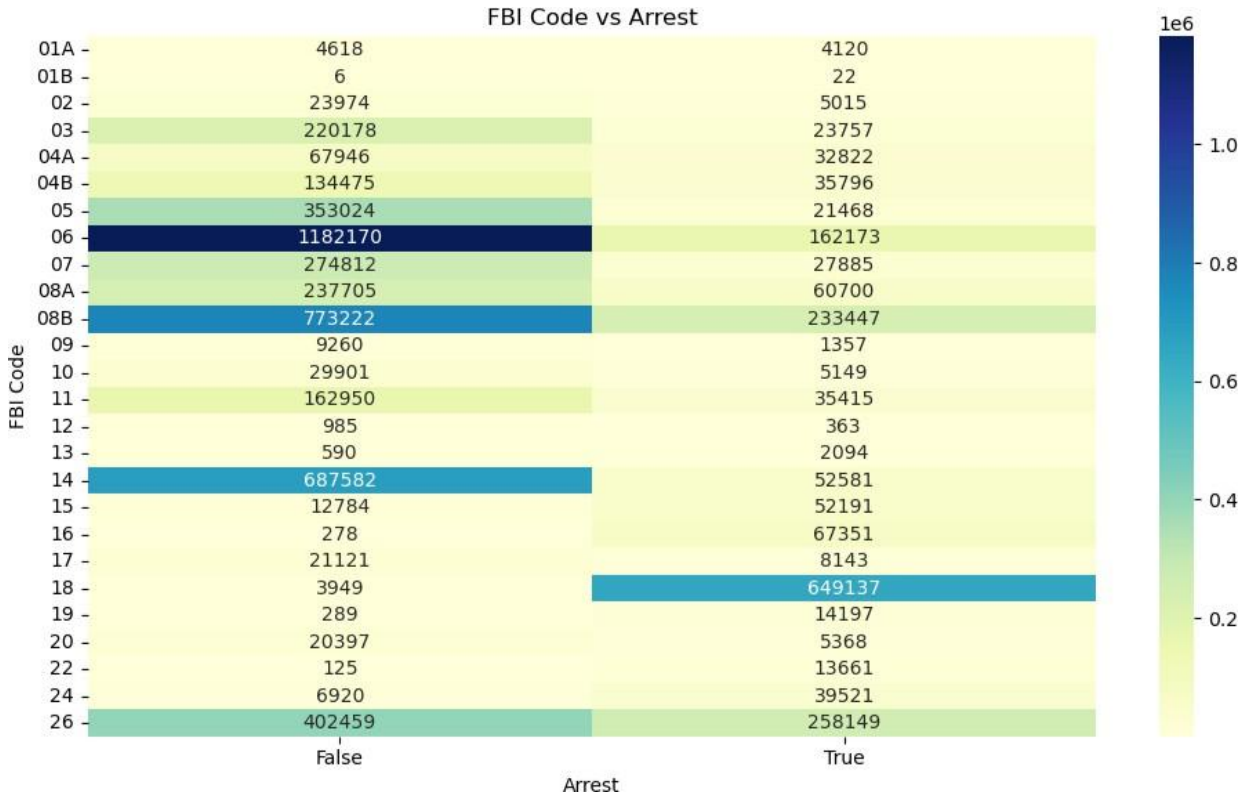


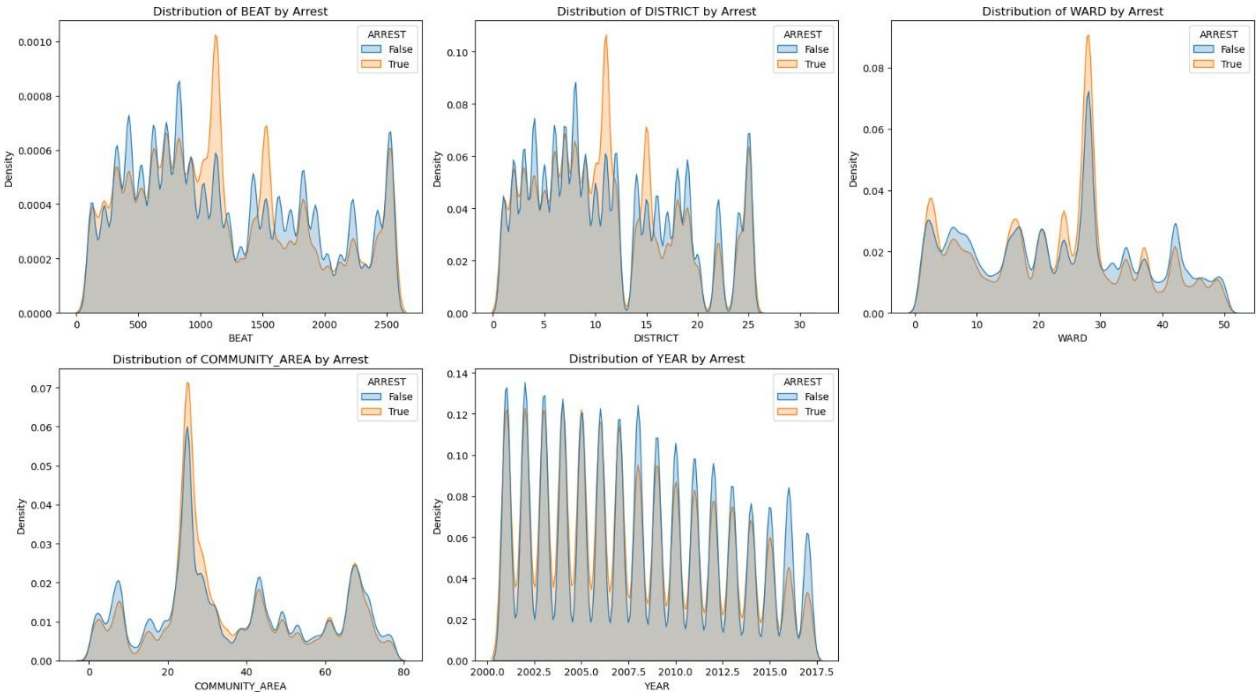
Screenshot 20: The unique values before-and-after comparison for DESCRIPTION

The values of the DESCRIPTION column are mapped to generalised crime descriptions. The result yields 248 unique values compared to 379 unique values before mapping, a 131 unique values reduction.

Visualise categorical distribution







5.4 Feature Engineering

```
# Check cardinality for all columns
for column in df.columns:
    unique_count = df[column].nunique()
    total_count = len(df[column])
    cardinality = unique_count / total_count
    print(f"Column: {column}")
    print(f"Unique values: {unique_count}")
    print(f"Total values: {total_count}")
    print(f"Cardinality ratio: {cardinality:.2%}")
    print("----------")
```

Screenshot 21: Code to find the cardinality of each feature

Column	Cardinality	Column	Cardinality	Column	Cardinality
ID	100.0%	DATE	39.78%	WARD	0.00%
CASE_NUMBER	99.99%	BLOCK	0.92%	FBI_CODE	0.00%
PRIMARY_TYPE	0.00%	IUCR	0.01%	YEAR	0.00%
DESCRIPTION	0.00%	ARREST	0.00%	LONGITUDE	13.15%

LOCATION_DESCRIPTION	0.00%	DOMESTIC	0.00%	LATITUDE	13.16%
COMMUNITY_AREA	0.00%	BEAT	0.00%		
UPDATED_ON	0.03%	DISTRICT	0.00%		

The following code calculates the cardinality for each column. The columns with 30% > cardinality rate are considered high cardinality and are removed. Hence, ID, CASE_NUMBER and DATE columns are dropped.

5.5 Model Building & Evaluation

Three machine learning algorithms — K-Nearest Neighbours (KNN), Decision Tree, and Support Vector Machine (SVM), are utilized in this study. These models are trained using the previously split training dataset. To reduce the risk of overfitting, only the top 10, 8, and 6 features, identified through a chi-squared test, are selected for model training as chi-squared tests are suited to determine whether there is a significant association between the features and the target feature, ARREST [6].

```
target_col = df['ARREST']  # our target feature
features = df.drop(columns=['ARREST', 'LATITUDE', 'LONGITUDE'])  # drop ARREST, LATITUDE and LONGITUDE columns

best_features = SelectKBest(score_func=chi2, k=10)  # selecting the best 10 features using chi square test
fit = best_features.fit(features, target_col)  # fitting the model
scores = pd.DataFrame(fit.scores_)  # getting the scores
columns = pd.DataFrame(features.columns)  # getting the columns
```

```
# fit scores and columns in a dataframe
featureScores = pd.concat([columns, scores], axis=1)  # concatenating the columns and scores
featureScores.columns = ['Feature', 'Score']  # setting the column names
featureScores = featureScores[featureScores["Feature"] != "ARREST"]  # filtering out the "quality" column
featureScores.sort_values(by='Score', ascending=False, inplace=True)  # sorting the features by score
featureScores.reset_index(drop=True, inplace=True)  # resetting the index
featureScores  # displaying the features and their scores
```

Screenshot 22: Code for the Chi-Squared used to identify the top features

```
#defining feature set
feature_sets = {
    '10_features': ['BLOCK', 'IUCR', 'FBI_CODE', 'DESCRIPTION', 'BEAT',
                    'PRIMARY_TYPE', 'LOCATION_DESCRIPTION', 'DOMESTIC', 'WARD', 'DISTRICT'],
    '8_features': ['BLOCK', 'IUCR', 'FBI_CODE', 'DESCRIPTION', 'BEAT',
                   'PRIMARY_TYPE', 'LOCATION_DESCRIPTION', 'DOMESTIC'],
    '6_features': ['BLOCK', 'IUCR', 'FBI_CODE', 'DESCRIPTION', 'BEAT',
                   'PRIMARY_TYPE'],
```

Screenshot 23: Code for a dictionary that indicates the top features used in the analysis

5.5.1 K-Nearest Neighbours (KNN)

```
neighbors_list = [1, 3, 5, 7, 9]
```

Screenshot 24: The different values chosen for *K*

The values of *K* used in the model were limited to 1, 3, 5, 7, and 9. This restriction was necessary due to the extremely large size of the dataset, which would have resulted in significantly longer processing times if a wider range of *K* values were tested across the three train-test splits and four feature sets. Additionally, odd successive values were chosen intentionally, as using an odd number for *K* is recommended to avoid ties in classification decisions.

```
#loop through k-values
for k in neighbors_list:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)
    acc = accuracy_score(y_test, y_pred)
    k_accuracy.append((k, acc))
```

Screenshot 25: Code to identify the optimal *K* value

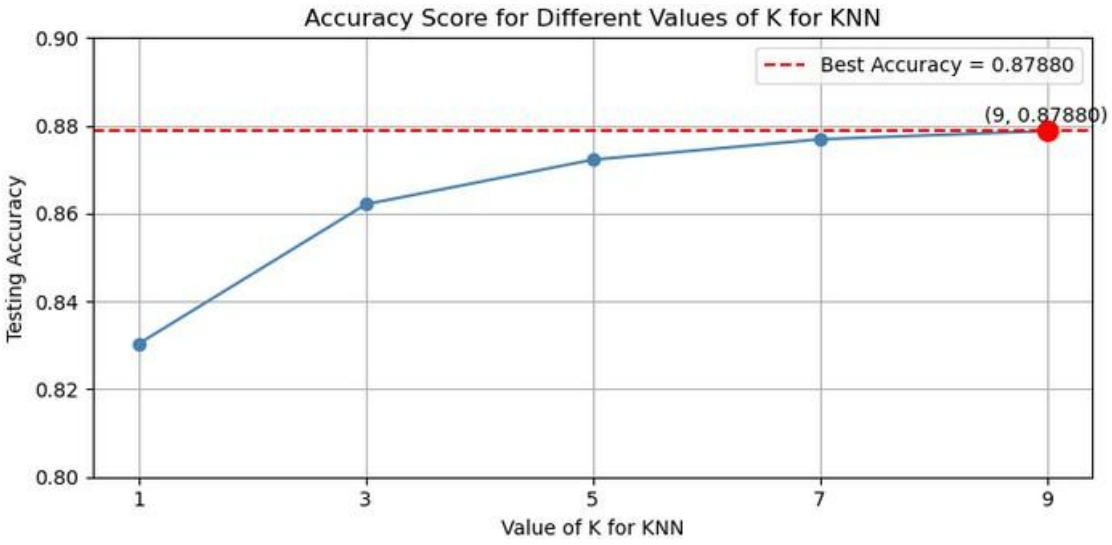
To determine the optimal *K* value for the K-Nearest Neighbours (KNN) algorithm, a loop iterates through odd *K* values ranging from 1 to 9. For each value of *K*, the model is trained and its accuracy score is calculated. The results are then plotted to visualize how accuracy changes with different *K* values. This iterative approach is essential for identifying the *K* value that provides the best performance for the KNN model.

```
#sort out the accuracy
k_accuracy.sort(key=lambda x: x[1], reverse=True)
best_k, best_acc = k_accuracy[0]

#print sorted accuracies
print("K-values and their accuracies (sorted):")
for k, acc in k_accuracy:
    print(f"k = {k}: Accuracy = {acc:.6f}")


#store the k-value with the best accuracy
bestK_feat10_60_40 = best_k
```

Screenshot 26: The image above illustrates the algorithm used to determine the best *K* value. The various *K* values are sorted by their respective accuracy, and the value with the highest accuracy is chosen.



Screenshot 27: The visualisation of the results based on different K values.

The image above showcases the optimal *K* value determined during training with a data split ratio of 60-40 and using 10 selected features, where the best *K* value is 9. This best *K* value is stored in a variable named bestK_feat<number of features trained><the train-test split>, specifically "bestK_feat10_60_40" in this instance.

The KNN model was then trained using the optimal K value, denoted as bestK_feat<number of features trained><the train-test split> which had been identified during the evaluation process. Using this optimal K, the model was applied to predict the target variable "ARREST" on the test

dataset. This step determines whether an arrest was made or not based on the model's predictions, leveraging the selected features.

```
#run the KNN model with best K as the number of neighbors
knn_final = KNeighborsClassifier(n_neighbors=bestK_feat10_60_40)
knn_final.fit(X_train_scaled, y_train)
y_pred_final = knn_final.predict(X_test_scaled)
```

Screenshot 28: An image of KNN training and inferencing

```
#store data in a df for later use
KNN_feat10_60_40 = pd.DataFrame([{
    'Feature_Set': feature_label,
    'Train_Size': train_size,
    'Test_Size': test_size,
    'Best_k': bestK_feat10_60_40,
    'Accuracy': acc_final,
    'Weighted_Precision': prec_final,
    'Weighted_Recall': rec_final,
    'Weighted_F1': f1_final,
    'Confusion_Matrix': cm_final
}])
```

Screenshot 29: An example of dataframe storing KNN's evaluation results.

Screenshot 28 and 29: The KNN model is trained using the best K value for the specific feature set and train-test split. The output metrics such as accuracy, precision, recall, and F1-score are then stored in a dataframe named KNN_feat<number of features trained><the train-test split>. In this case, the results are stored in "KNN_feat10_60_40", which corresponds to the model trained with the top 10 features using a 60/40 train-test split.

5.5.2 Support Vector Machine (SVM)

The SVM model is trained on 60/40, 70/30 and 80/20 splits. For each split, the number of features, N is tested on 6, 8 and 10. The SVM used a linear kernel to reduce the model's computational intensity. Additionally, a loop was used to train and tune the model's parameter, C for values 0.01, 0.1, 1, 10 and 100.

5.5.3 Decision Tree

A decision tree works by recursively splitting the data into subsets based on feature values, creating a tree-like model of decisions. In this there are three sets of coding by splitting the dataset into 60-40, 70-30, and 80-20 train-test ratios. For each split, it experimented with different numbers of features for the split criterion, specifically N = 6, N = 8, and N = 10. This allows us to observe the train-test split ratios and the number of features that impact the model's performance and accuracy.

```
# Config for 6 features, 60/40 split
feature_label = '6_features'
train_size = 0.6
test_size = 0.4

selected_features = feature_sets[feature_label]
features = df[selected_features]
target = df['ARREST']

X_train, X_test, y_train, y_test = train_test_split(
    features, target, train_size=train_size, random_state=42, stratify=target)
```

A loop was used to train and evaluate the model for multiple max_depth values like 3, 5, 10, and none. `depth_list = [3, 5, 10, None]`

5.5.4 Model Evaluation

Upon the construction of all the models, various metrics are utilized to evaluate their performance. These metrics are as follows: Accuracy, Precision, Recall, and F1-Score [7]. Additionally, a confusion matrix is used to visually represent the performance of the classification model by showing the number of correct predictions—True Positives (TP) and True Negatives (TN)—as well as the incorrect ones—False Positives (FP) and False Negatives (FN). This matrix offers a detailed breakdown of the model's classification results, allowing for a deeper understanding of its strengths and weaknesses beyond overall accuracy. By analyzing these values, one can assess how well the model distinguishes between classes, making it an essential tool for evaluating classification performance [8]. These evaluations offer a detailed understanding of the model's performance in predicting an arrest, highlighting how effectively it distinguishes between situations where an arrest is likely to occur and where it is not, based on the selected features

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

$$Precision_{weighted} = w_i \sum_{i=1}^{n} \frac{TP_i}{TP_i + FP_i}$$

$$Recall_{weighted} = w_i \sum_{i=1}^{n} \frac{TP_i}{TP_i + FN_i}$$

$$F1_{weighted} = w_i(2 \cdot \frac{\frac{TP_i}{TP_i + FP_i} \cdot \frac{TP_i}{TP_i + FN_i}}{\frac{TP_i}{TP_i + FP_i} \cdot \frac{TP_i}{TP_i + FN_i}})$$

Confusion Matrix:

True Negative (TN)	*False Positive (FP)*
False Negative (FN)	*True Positive (TP)*

The weighted averaging score is employed in this analysis due to the notable class imbalance in the target column “ARREST”. For instance, in all model configurations using a 80-20 train-test split, approximately 72% of the instances are labeled as 0 (indicating no arrest), while only 28% are labeled as 1 (indicating an arrest).This significant disparity can bias traditional evaluation metrics if not properly accounted for. Weighted averaging addresses this issue by calculating precision, recall, and F1-score for each class and then averaging them while assigning weights proportional to each class’s support (i.e., number of true instances). This ensures that the model's performance is evaluated fairly across both classes, preventing the majority class from disproportionately influencing the results and providing a more balanced and reliable measure of model effectiveness.

Following model accuracy, recall is especially important in the context of arrest prediction, where minimizing false negatives; cases where actual arrests are not predicted is critical. Recall measures the proportion of actual positive cases (i.e., actual arrests) that the model correctly identifies. A high recall score indicates that the model successfully captures most of the instances where an arrest truly occurred, thereby reducing the risk of overlooking real arrests. This is particularly important in law enforcement or public safety applications, where failing to flag situations that should result in an arrest could lead to serious consequences. Thus, recall plays a key role in evaluating the model's reliability in identifying true arrests.

6.0 Comparisons and Recommendations

Three experiments are conducted to achieve these set objectives:

1. **To determine the optimal value of a single parameter for 3 train-test split ratios:** Evaluate the performance by accuracy of each model type (KNN, SVM, Decision tree) with train-test split ratio of 60/40, 70/30 and 80/20 with respect to one parameter on each model.
2. **To identify the optimal number of feature selections:** Test each model on 6, 8 and 10 features sorted by the highest ranking Chi-Squared result to determine the best features that resulted in the highest performance.
3. **Select the best Machine Learning model as the champion model:** Select the best performing model out of the 9 models (3 train-test splits * 3 feature selections = 9 models) for each ML model.

There are many manipulating variables — the number of features, the value of a tuned parameter for each model and the train-test splits. Therefore, it is critical to compare the different combinations of the manipulating variables by evaluating them based on the performance metrics, particularly the accuracy and recall scores, for each model.

6.1 Comparisons and Recommendations

This part will compare the performance metrics and evaluations of the models based on:

1. The train-test split ratios (represented as column)
2. The number of features, N (represented as row)

This will help us figure out the best model configuration

Legend

Top 1	
Top 2	
Top 3	

6.1.1 K-Nearest Neighbours (KNN)

	60/40 split		**70/30 split**		**80/20 split**	
N = 6						
	Accuracy	0.8784	Accuracy	0.8788	Accuracy	0.8789
	Weighted Precision	0.8773	Weighted Precision	0.8778	Weighted Precision	0.8779
	Weighted Recall	0.8784	Weighted Recall	0.8788	Weighted Recall	0.8789
	Weighted F1	0.8734	Weighted F1	0.8737	Weighted F1	0.8739
	Best K	9	Best K	9	Best K	9
N = 8						
	Accuracy	0.8794	Accuracy	0.8799	Accuracy	0.8797
	Weighted Precision	0.8783	Weighted Precision	0.8788	Weighted Precision	0.8785
	Weighted Recall	0.8794	Weighted Recall	0.8799	Weighted Recall	0.8797
	Weighted F1	0.8745	Weighted F1	0.8751	Weighted F1	0.8749
	Best K	9	Best K	9	Best K	9
N = 10						
	Accuracy	0.8788	Accuracy	0.8791	Accuracy	0.8790

	Weighted Precision	0.8777	Weighted Precision	0.8780	Weighted Precision	0.8778
	Weighted Recall	0.8788	Weighted Recall	0.8791	Weighted Recall	0.8790
	Weighted F1	0.8738	Weighted F1	0.8742	Weighted F1	0.8741
	Best K	9	Best K	9	Best K	9

From the summary of metrics evaluation, it is found that **KNN** model with a train-test split of **70/30** and the number of top features, **N=8** achieves the highest accuracy, weighted precision, weighted recall and weighted F1 score at **K=9** amongst the 9 KNN models. Hence, this model is the champion model for KNN.

6.1.2 Support Vector Machine (SVM)

	60/40 split		**70/30 split**		**80/20 split**	
N = 6						
	Accuracy	0.7627	Accuracy	0.7621	Accuracy	0.7617
	Weighted Precision	0.7707	Weighted Precision	0.7706	Weighted Precision	0.7704
	Weighted Recall	0.7621	Weighted Recall	0.7621	Weighted Recall	0.7617
	Weighted F1	0.7657	Weighted F1	0.7656	Weighted F1	0.7653
	Best C	0.1	Best C	0.1	Best C	0.01
N = 8						
	Accuracy	0.7400	Accuracy	0.7396	Accuracy	0.7389
	Weighted Precision	0.7576	Weighted Precision	0.7572	Weighted Precision	0.7568
	Weighted Recall	0.7400	Weighted Recall	0.7396	Weighted Recall	0.7389
	Weighted	0.7464	Weighted	0.7462	Weighted	0.7455

	F1		F1		F1	
	Best C	0.1	Best C	0.1	Best C	1.0
N = 10						
	Accuracy	0.7399	Accuracy	0.7395	Accuracy	0.7388
	Weighted Precision	0.7576	Weighted Precision	0.7572	Weighted Precision	0.7567
	Weighted Recall	0.7399	Weighted Recall	0.7395	Weighted Recall	0.7388
	Weighted F1	0.7464	Weighted F1	0.7461	Weighted F1	0.7544
	Best C	0.1	Best C	10.0	Best C	0.01

From the summary of metrics evaluation, it is found that **SVM** model with a train-test split of **60/40** and the number of top features, **N=6** achieves the highest accuracy, weighted precision, weighted recall and weighted F1 score at **C=0.1** amongst the 9 SVM models. Hence, this model is the champion model for SVM.

6.1.3 Decision Tree

	60/40 split		**70/30 split**		**80/20 split**	
N = 6						
	Accuracy	0.8797	Accuracy	0.8794	Accuracy	0.8795
	Weighted Precision	0.8803	Weighted Precision	0.8799	Weighted Precision	0.8800
	Weighted Recall	0.8797	Weighted Recall	0.8794	Weighted Recall	0.8796
	Weighted F1	0.8736	Weighted F1	0.8734	Weighted F1	0.8735
	Best Depth	10	Best Depth	10	Best Depth	10
N = 8						

	Accuracy	0.8807	Accuracy	0.8804	Accuracy	0.8803
	Weighted Precision	0.8819	Weighted Precision	0.8809	Weighted Precision	0.8809
	Weighted Recall	0.8807	Weighted Recall	0.8804	Weighted Recall	0.8803
	Weighted F1	0.8745	Weighted F1	0.8744	Weighted F1	0.8744
	Best Depth	10	Best Depth	10	Best Depth	10
N = 10						
	Accuracy	0.8810	Accuracy	0.8807	Accuracy	0.8808
	Weighted Precision	0.8822	Weighted Precision	0.8818	Weighted Precision	0.8818
	Weighted Recall	0.8811	Weighted Recall	0.8807	Weighted Recall	0.8808
	Weighted F1	0.8749	Weighted F1	0.8746	Weighted F1	0.8746
	Best Depth	10	Best Depth	10	Best Depth	10

From the summary of metrics evaluation, it is found that the **Decision Tree** model with a train-test split of **60/40** and the number of top features, **N=10** achieves the highest accuracy, weighted precision, weighted recall and weighted F1 score at **depth 10** amongst the 9 Decision Tree models. Hence, this model is the champion model for Decision Tree.

6.2 Recommendations

- Currently, the models are run manually one-by-one. It could be improved by performing **systematic hyperparameter turning** using Grid Search and Bayesian optimisation rather than fixed parameters like K, C and max_depth
- Experiment with **different feature importance analysis**, other than Chi-Squared test. Random Forests and Gradient Boosted Trees could quantify feature contributions and

improve the feature selection process beyond arbitrary top N lists. The model could select which feature matters most and how many to select rather than deciding with a number.

- To improve current model accuracy, the **range of features**, N could be improved to go further beyond 10 features to include temporal features. By experimenting with these different subset of features, it will influence the overall model performance.

7.0 Results and Discussion

7.1 Results

The champion model for each Machine Learning algorithms are summarised as follows:

Performance Metrics / Algorithms	**KNN**	**SVM**	**Decision Tree**
Train test ratio	70/30	60/40	60/40
Number of features selected, N	8	6	10
Accuracy	0.8799	0.7627	0.8810
Weighted Precision	0.8788	0.7707	0.8822
Weighted Recall	0.8799	0.7621	0.8811
Weighted F1 Score	0.8751	0.7657	0.8749
Best parameter	K = 9	C = 0.1	Depth = 10

Champion Model: Decision tree of depth = 10 with 60/40 train test split and 10, number of features, N = 10.

7.2 Discussion

7.2.1 General

1. The best KNN model was trained using a 70/30 train test split while the best SVM and Decision Tree models were trained using a 60/40 train test split. The best KNN model performed better with a 70/30 split, likely due to its reliance on a larger training set for accurate neighbor-based predictions. In contrast, SVM and Decision Tree models achieved better results with a 60/40 split, suggesting they benefited from a more balanced training-test ratio to avoid overfitting.

2. The optimal KNN model was trained using 8 features with the highest score, the optimal Decision Tree model was trained using 10 features and the best SVM model was trained using 6 features.

7.2.2 Confusion Matrix Analysis

1. The Confusion Matrix is structured as follows:

   [[True Negative, False Positive],

   [False Negative, True Positive]]

2. The best KNN model achieved 1,332,195 true negatives and 57,321 false positives, along with 174,764 false negatives and 368,801 true positives. This indicates strong overall performance, especially in correctly identifying both classes, though it still misclassified a moderate number of positive instances as negative (false negatives).
3. The best Decision Tree model recorded 1,795,712 true negatives and 56,976 false positives, with 249,589 false negatives and 475,164 true positives. Compared to the KNN model, it had a slightly lower number of false positives and a significantly higher number of true positives. Further it had the highest number of true negatives across the three models, showing better balance and a stronger ability to detect positive cases. However, it did have a significantly higher number of false negatives than KNN but slightly less than SVM.
4. The best SVM model showed 1,506,670 true negatives and 346,447 false positives, along with 266,803 false negatives and 457,521 true positives. Although it had a relatively high number of true positives, and true negatives, it also had the highest number of false positives and false negatives among all three models, indicating less reliable classification performance overall.

7.2.3 Accuracy Comparison

1. Decision Tree achieved the highest accuracy score among the three models at 88.11%, followed by KNN with 87.99%, and SVM with 76.21%.
2. This suggests that the Decision Tree model slightly outperformed the other two in accurately classifying whether an arrest would occur or not.

3. Overall, both Decision Tree and KNN demonstrated strong and comparable performance, while SVM showed noticeably lower accuracy with a difference of around ±12%, indicating less reliability in this context.

7.2.4 Weighted Precision, Weighted Recall, and Weighted F1 Score Comparison

1. **Weighted Precision:** It measures the proportion of true positive predictions (correctly predicted an arrest) among all positive predictions.

   a. Decision Tree demonstrated the highest weighted precision among the three models (88.22%), followed by KNN (87.88%), and SVM (77.07%).

   b. This indicates that when the Decision Tree model predicts an arrest, it is correct 88.22% of the time, showing its strong ability to minimize false positives compared to KNN and SVM.

2. **Weighted Recall:** It measures the proportion of true positives that were correctly predicted (the actual percentage of correctly predicted arrest).

   a. KNN demonstrated the highest weighted recall among the three models (87.99%), followed by Decision Tree (88.11%), and SVM (76.21%).

   b. This indicates that KNN correctly identified 87.99% of all arrest, showing its strong ability to minimize false negatives and effectively capture true positives compared to the other models.

3. **Weighed F1 Score:** This metric evaluates the balance between precision and recall.
   - **a.** KNN achieved the highest weighted F1 score (87.51%), followed closely by Decision Tree (87.49%), and SVM (76.57%).
   - **b.** SVM had a noticeably lower score compared to the other two models, likely because it performed better in precision than in recall, meaning it was more accurate when predicting an arrest, but it missed many actual arrests.
   - **c.** KNN and Decision Tree showed strong and balanced performance in both precision and recall, resulting in similarly high weighted F1 scores when predicting arrests.

7.2.5 Model Selection

1. Based on the provided metrics, the Decision Tree model with a 60-40 train-test split and 10 features emerged as the best overall performer. It achieved the highest accuracy (88.11%) and highest precision (88.22%), along with a strong and balanced recall and F1 score, making it the most reliable model for predicting arrests.
2. The KNN model, using a 70-30 split and 8 features, also showed excellent performance, achieving the highest recall (87.99%) and highest F1 score (87.51%), indicating strong capability in correctly identifying actual arrests while maintaining balanced precision.
3. The SVM model, while achieving high precision (77.07%), had noticeably lower recall (76.21%), accuracy (76.21%), and F1 score (76.57%) compared to the other models, making it the least effective in this context despite its low false positive rate.

Champion Model:

Decision Tree of depth = 10 with 60/40 train test split, and number of features, N = 10.

8.0 Concluding Remarks

This report successfully evaluated and compared the performance of these 3 models — KNN, SVM and Decision Tree for predicting whether a crime will result in arrest when reported in Chicago. The objective of this report is to determine the optimal configuration of train test split ratio, number of features and the best hyperparameter for each model systematically.

Amongst the contested models, Decision Tree arose as the champion model trained on 60/40 train test split ratio at a maximum depth of 10. It demonstrated the highest overall performance across accuracy, weighted precision, recall and F1 score consistently outperforming KNN and SVM at an accuracy of 88.1% and recall score of 88.11%.

This report showed that model performance is sensitive to both data partitioning configuration and feature selection. A balanced features and data partition contributes significantly to the model accuracy and reducing misclassification of false negatives in arrest predictions.

9.0 Lesson Learned

This project has taught us several important lessons. Firstly, understanding the fundamentals of machine learning algorithms is essential in designing the experiments and building effective models. This helps us control the model complexity, reduce training time and increase performance.

Furthermore, deep understanding of the dataset is important during data preprocessing cannot be stressed enough. The deep comprehension of the dataset will allow us to make reasonable assumptions about the features that enabled us to clean the dataset, replace and fill in missing values and standardise certain categories. By meticulously cleaning the data, the results will reflect during model building and evaluating.

Additionally, recall is a critical metric in public safety-related applications. Prioritising recall ensures that false positives like the incidents deserving of arrest are less likely to be missed. This is vital for effectiveness of law enforcement and community safety.

Lastly, feature selection plays a crucial role in classification type model performance. The reason being that, arbitrarily increasing the number of features does not always improve model performance, but rather through meaningfully curated features based on well-grounded assumptions and effective algorithm scoring.

10.0 Conclusion

In a nutshell, KNN, SVM and Decision Tree possess distinct strengths that benefited from hyperparameter tuning. KNN is computationally and memory intensive as the dataset size scales. Optimising KNN requires selecting the optimal K value. Decision trees are simple and straightforward, however the trees must be limited or pruned to avoid becoming overly complex. SVM has the potential to excel at non-linear datasets although that would be memory and computationally too intensive for this project. It certainly benefited from selecting appropriate kernel types

.Decision Tree emerged as the champion model, delivering performance across all metrics, that is, accuracy, weighted precision, recall and F1 scores. At a configuration of depth of 10, trained on 60/40 train test split ratio and selection of top 10 number of features ranked by Chi-Squared

test. Adjusting train-test split ratio and the number of top-scoring features will improve model's prediction accuracy and other performance metrics. Fine tuning the models' parameters will enhance its accuracy too in classifying the likely outcome of an arrest. These are the important parameters to adjust in this supervised learning approach.

11.0 IEEE Referencing

[1] M. Janssen, Y. Charalabidis, and A. Zuiderwijk, "Benefits, Adoption Barriers and Myths of Open Data and Open Government," *Government Information Quarterly*, vol. 29, no. 4, pp. 498–506, Oct. 2012. [Online]. Available: https://doi.org/10.1016/j.giq.2018.01.004 [Accessed: Jun. 10, 2024]

[2] GeeksforGeeks. (2023). K-Nearest Neighbors (KNN) Algorithm. Retrieved from: https://www.geeksforgeeks.org/k-nearest-neighbours/ [Accessed: Jun. 10, 2024]

[3] Scikit-learn. (n.d.). StandardScaler — scikit-learn documentation. Retrieved from: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html [Accessed: Jun. 10, 2024]

[4] Forecastegy. (2023). Why Feature Scaling Matters in Machine Learning. Retrieved from: https://forecastegy.com/why-feature-scaling-matters [Accessed: Jun. 10, 2024]

[5] Creative Commons, CC0 1.0 Universal (CC0 1.0) Public Domain Dedication, [Online]. Available: https://creativecommons.org/publicdomain/zero/1.0/ [Accessed: Jun. 14, 2025]

[6] Z. Martin, "When to Use the Chi-Square Test," Statology, [Online]. Available: https://www.statology.org/when-to-use-chi-square-test/ [Accessed: Jun. 14, 2025]

[7] Vitalflux. (n.d.). "Micro Average & Macro Average Scoring Metrics in Multi-Class Classification using Python." Available online: https://vitalflux.com/micro-average-macro-average-scoring-metrics-multi-class-classification-python/ [Accessed: Jun. 14, 2024]

[8] Brownlee, J. (2020). Confusion Matrix for Classification. Machine Learning Mastery. Retrieved from: https://machinelearningmastery.com/confusion-matrix-for-machine-learning/ [Accessed: Jun. 15, 2024]

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://jmlr.org/papers/v12/pedregosa11a.html [Accessed: Jun. 13, 2025].

[10] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A Library for Large Linear Classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008. [Online]. Available: https://jmlr.csail.mit.edu/papers/v9/fan08a.html [Accessed: Jun. 13, 2025].