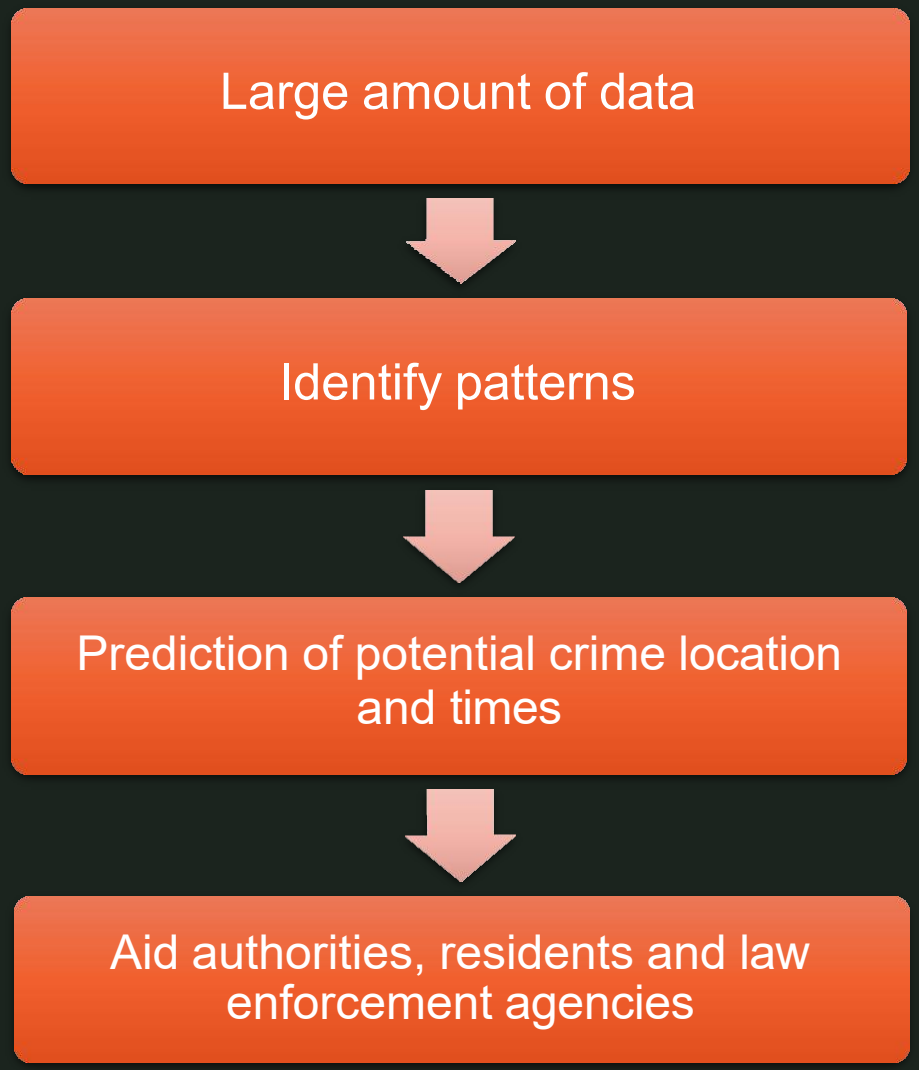
Predicting Crime and Proposing Safer Neighborhoods: Using Machine Learning Models

Agenda

- Abstract
- Chapter 1- Introduction
- Chapter 2- Data Management Plan
- Chapter 3- Data Engineering
- Chapter 4- Modelling
- Summary



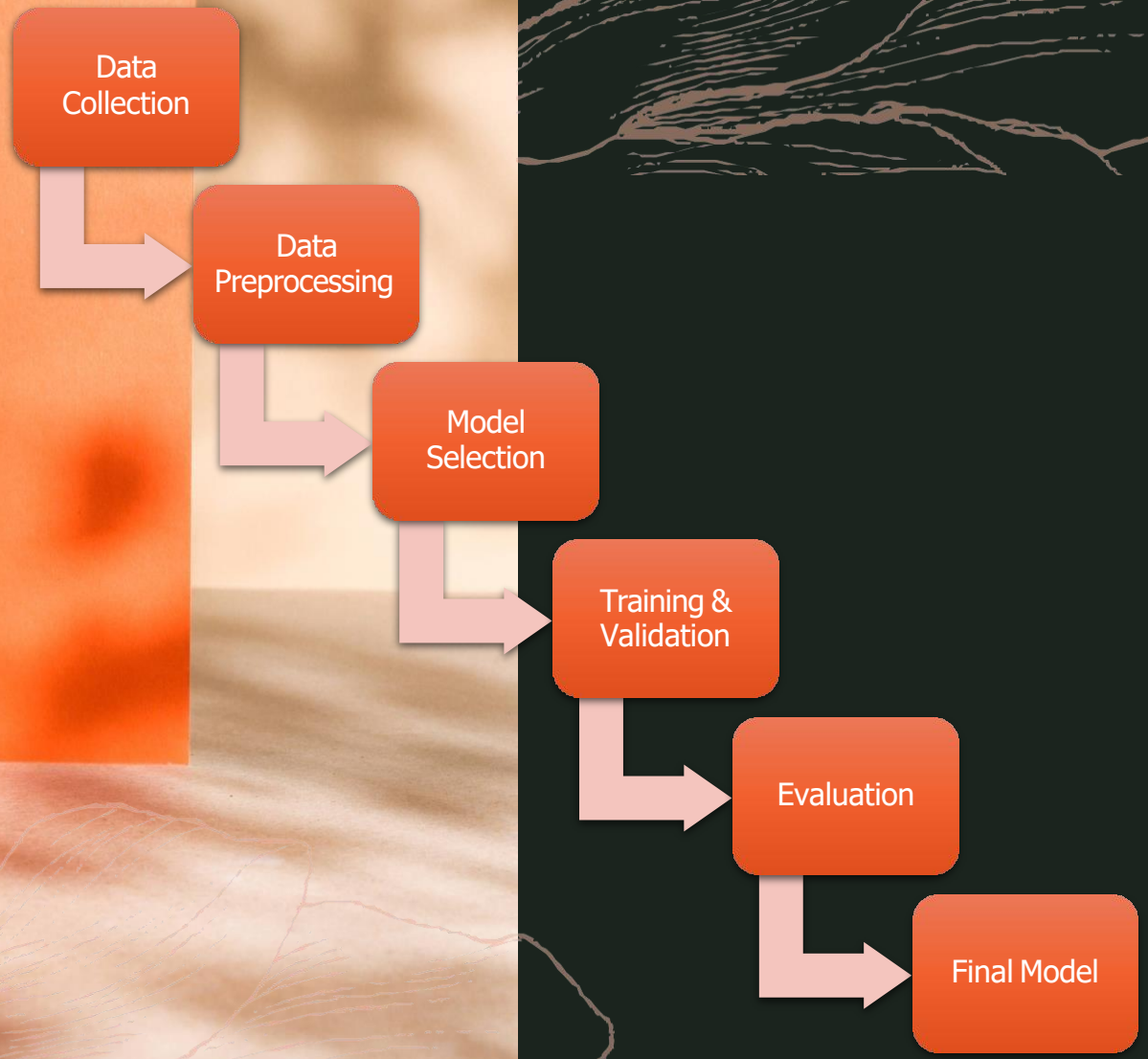
Abstract

- Leverage past crime data and by utilizing machine learning and data-driven methods, detect areas with an increased likelihood of criminal activity in advance.
- Raise public awareness about high-risk areas, and enable authorities to anticipate criminal activities for better allocation of police resources.
- Predictive policing aims to enhance public safety through preemptive actions and specific interventions, aiming to prevent incidents from happening.

Introduction

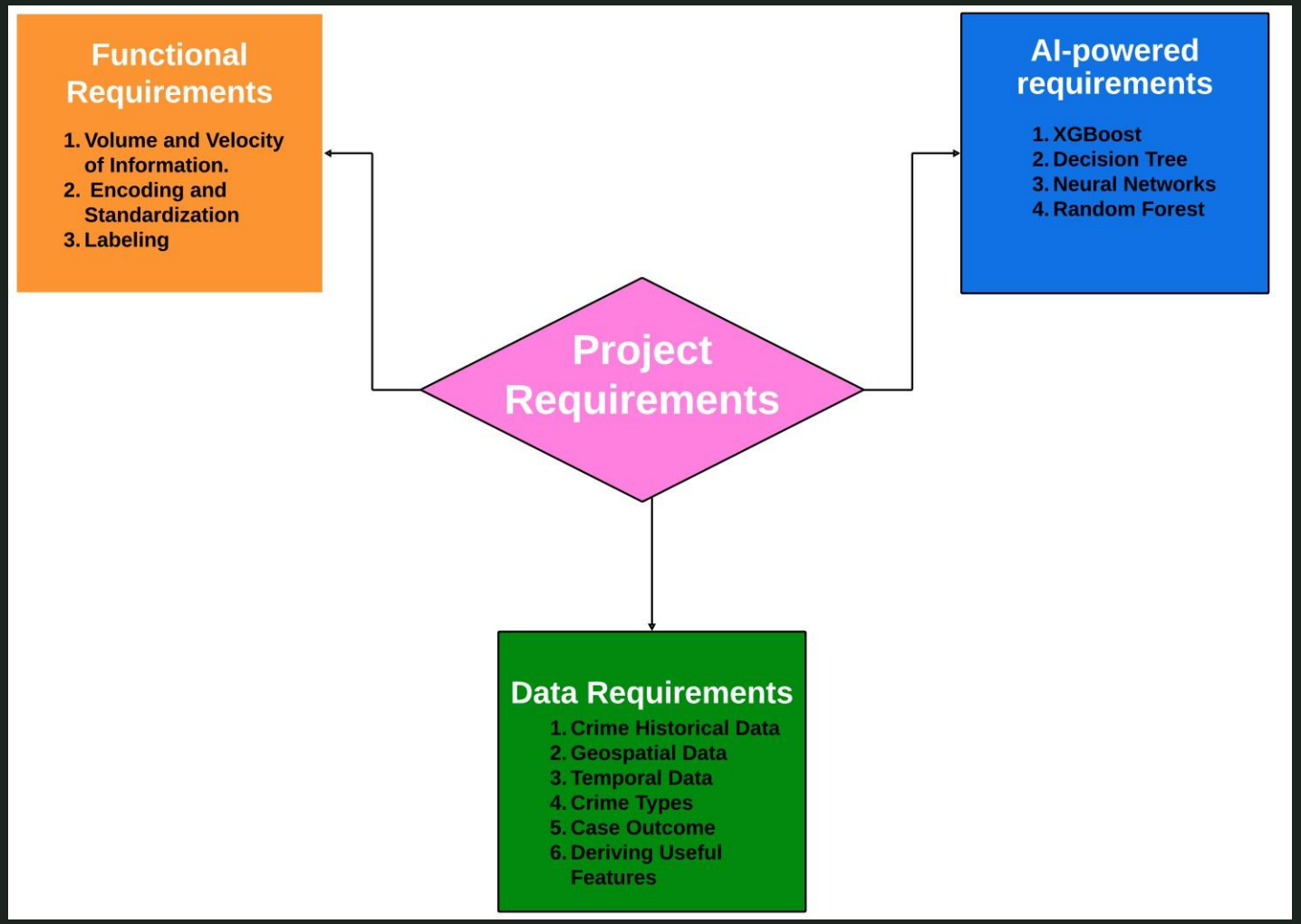
- Crime has a significant impact on individuals, families, and societies by causing economic setbacks and social disturbances.
- Factors such as socioeconomic inequalities, limited opportunities, substance abuse, and firearm availability contribute to crime in urban areas.
- Poverty and unemployment drive illegal activities while societal inequities compound challenges in marginalized neighborhoods.

Objective & Approach



- Crime Prevention Through Proactive Measures.
- Resource Allocation Optimization.
- Public Safety and Awareness.
- Reduced Social and Economic Impact.
- Making Communities Safer.

Project Requirements



Technology, Literature and Solution Survey

- **Lin et al. (2017)** proposed a data-driven approach using machine learning to combat drug-related crime in Taiwan, achieving an 87% accuracy. The strategy emphasizes addressing low-level crimes and utilizes the R programming language and the H2O package.
- **Walczak et al. (2021)** utilized neural networks to forecast crime using a dataset from Detroit, achieving 16.4% accuracy for 38 crime categories and an improved accuracy of 27.1% when categorized into seven groups.
- **Divya et al. (2022)** proposed an innovative approach to increase security measures using deep learning and achieved an accuracy pf 79%.
- **V. Balu et al. (2022)** focuses on predicting the likelihood of various crime types as their primary objective.
- **Kumar et al. (2020)** utilized The K-nearest neighbor algorithm to forecast the most common crime in a particular area, yielding a 99% precision when 'k' is configured to 3.
- **Ahishakiye et al. (2017)** predicted crime categories using the decision tree algorithm based on Violent Crimes Per Population, achieving an exceptional accuracy rate of 94.3%.

Technology, Literature and Solution Survey

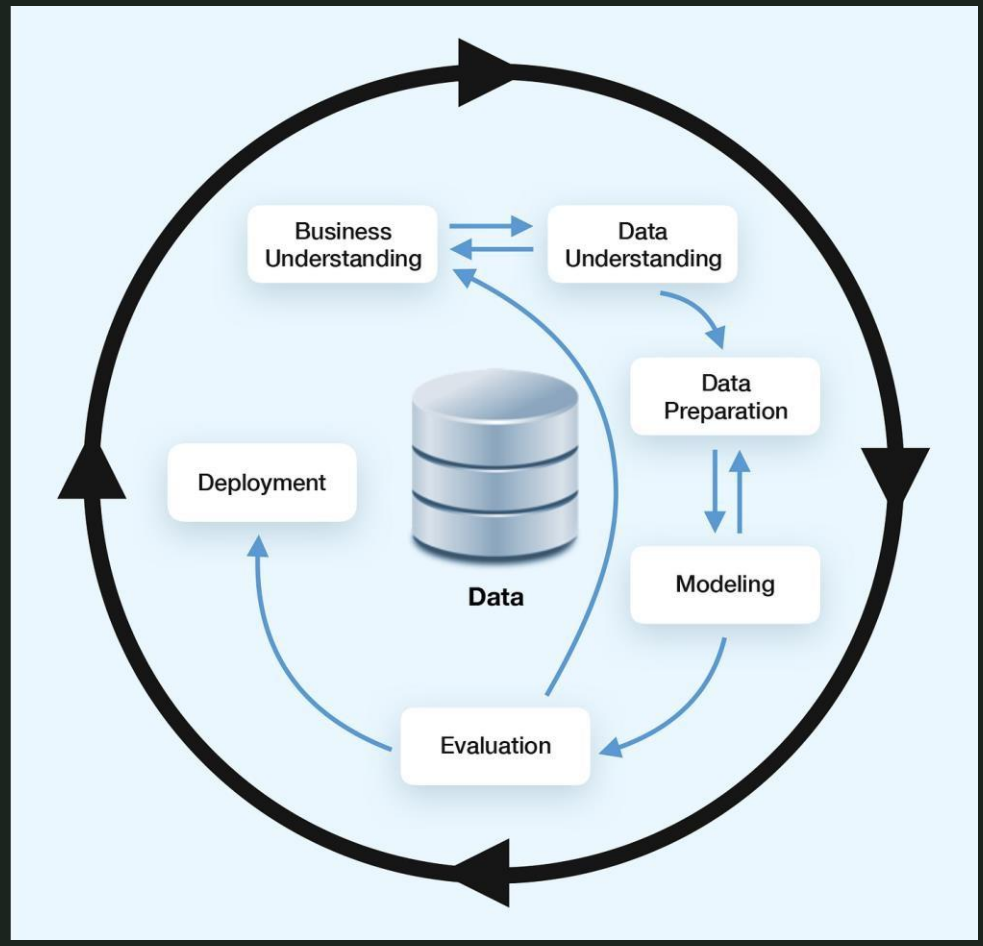
- **Bharati et al. (2018)** utilized machine learning algorithms to predict crime in Chicago. Achieved high accuracy with decision trees and random forest models.
- **Shah et al. (2021)** Explored the integration of Machine Learning and computer vision in crime prediction, emphasizing ongoing area surveillance for enhanced crime prevention.
- **Krishnaveni K.P. et al. (2022)** Explored different machine learning and deep learning methods for predicting crimes, highlighting the significance of using a variety of data sources and recognizing the difficulties in interpreting models.
- **Dr. B.S. Jadhav et al. (2021)** performed thorough analysis of machine learning methods for predicting criminal behavior, discussing challenges and exploring alternative approaches such as Natural Language Processing and Computer Vision.
- **J. Arunnehru et al.** (2021) developed a system that combines convolutional neural networks and support vector machines for real-time crime intention detection, achieving an accuracy of 95% on a dataset comprising 10,000 video clips.
- **Grega et al.** (2013) created a customized 3D CNN model to detect criminal intentions in cases of shoplifting, achieving a notable accuracy rate of 75% and demonstrating superiority over current systems.

Data Management Plan

- We chose Chicago crime dataset – 2001 to present which is of 1.86GB
- For storage we have utilized google drive, emphasized the efficient management of data, such as data manipulation, normalization due to its accessibility and collaborative capabilities.
- Scheduled to delete Google Drive files by January 01, 2024, yet will keep datasets on Kaggle for accessibility, collaboration, and visibility purposes while ensuring data integrity and openness.

Data Management Plan

- Business understanding – Project background, objective, plan
- Data understanding – Determine the data required for the targeted problem
- Data preparation – Ensure the data well-organized for modeling
- Modeling – Multiple models training, validation and testing
- Evaluation – Performance evaluation by various metrics
- Deployment – Make our work publicly accessible



Hardware and Software Requirements :

Hardware & Software	Configuration	Duration	Purpose	Cost
Data Storage	Google drive	2 Months	For storing the CSV files	Free
Local Machine	64-bit version	2 Months	For Computations	$900
CPU	Intel Core i7 or AMD Ryzen 7	2 Months	For executing instructions and processing data	$300
GPU	Nvidia GeForce RTX 3060 or AMD Radeon RX 6600	2 months	machine learning computations	$150
Python	**Version 3.12.0**	**2 Months**	**Programming language for machine learning**	**Free**
NumPy	Version 1.23.3	2 Months	Model Deployment	Free
Pandas	Version 1.5.2	2 Months	Data analysis and manipulation	Free
Scikit-learn	Version 1.1.3	2 Months	Machine learning algorithms	Free
MySQL(Community Edition)	Version 8.0.33	2 Months	Production database	Free
Tableau Desktop	2023.3	2 Months	Data visualization	Student License
Python Jupyter Notebook	Version 7.3.1	2 Months	Data exploration, Data Pre-processing (Writing python script)	Free

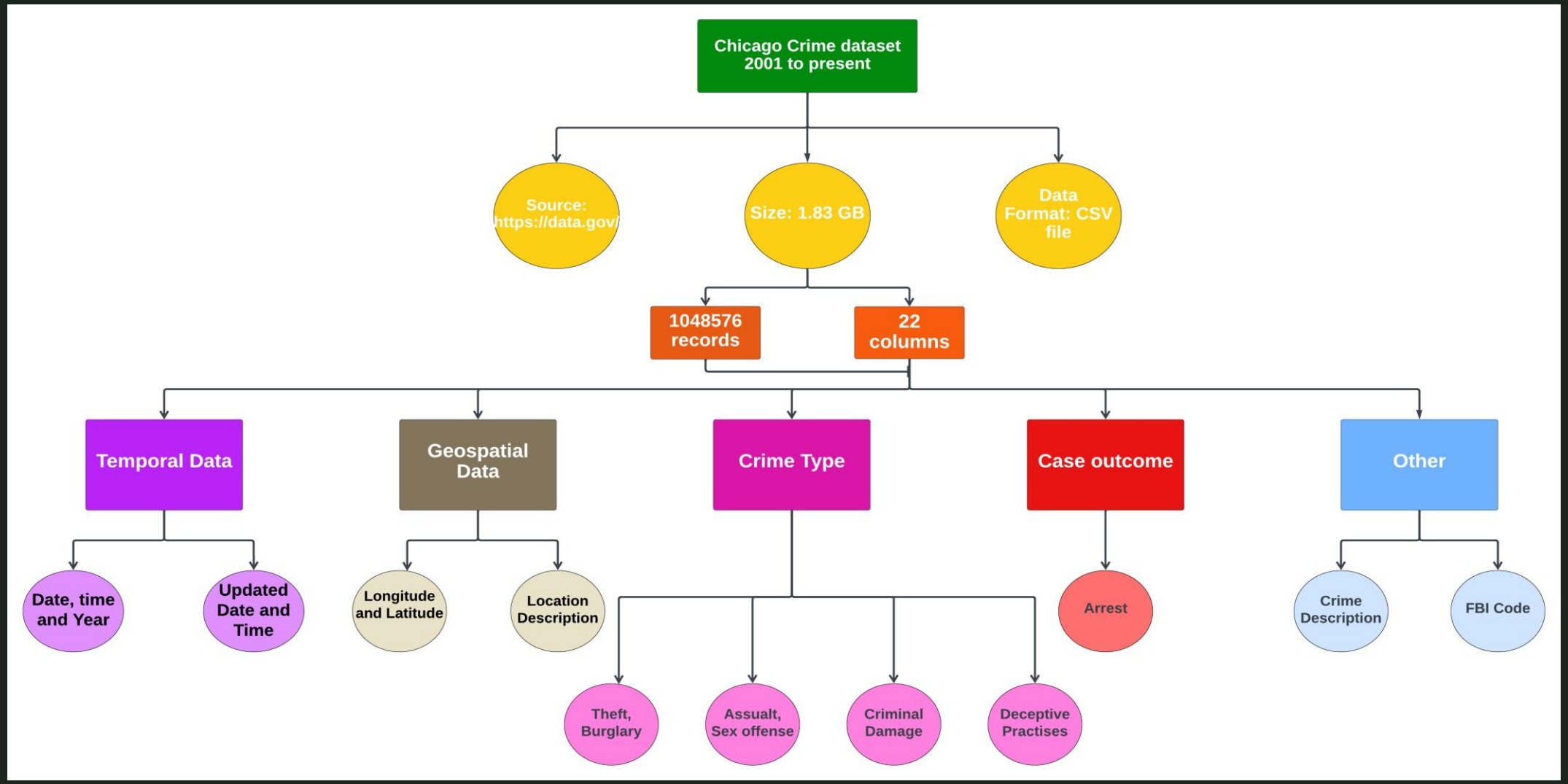
Data Engineering

- **Data Collection**
  - Utilize various sources like government databases, law enforcement agencies, and openly available datasets.
  - Obtain the dataset from the government website due to its comprehensibility and reliability.
- **Data Pre-processing**
  - Clean the data by removing duplicates, handling missing values.
- **Data Transformation**
  - Encode categorical variables and scale numerical features for consistency and accuracy.
  - Use Lasso Regression (L1 Regularization) and domain knowledge for feature extraction.
- **Data Preparation**
  - Split the dataset into training (70%), validation (20%), and test (10%) sets.
- **Data Statistics**
  - Summarize and display the results of the previous stage in visualization formats.

Data Collection-Sample Data Collected From data.gov

	ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	Domestic	...	Ward	Community Area	FBI Code	X Coordinate	Y Coordinate	Year	Updated On	Latitude	Longitude	Location
7458441	11508368	JB517219	11/14/2018 06:00:00 PM	030XX N Southport Ave	1153	DECEPTIVE PRACTICE	FINANCIAL IDENTITY THEFT OVER $ 300	NaN	False	False	...	32.0	6.0	11	NaN	NaN	2018	11/21/2018 04:14:47 PM	NaN	NaN	NaN
2136615	1428950	G151176	03/15/2001 06:00:00 PM	005XX W CORNELIA AV	0560	ASSAULT	SIMPLE	RESIDENCE	False	True	...	NaN	NaN	08A	1171891.0	1923756.0	2001	09/07/2021 03:41:02 PM	41.946244	-87.643583	(41.946243567, -87.643583496)
5404657	8161188	HT395984	07/13/2011 11:23:00 PM	103XX S ABERDEEN ST	1812	NARCOTICS	POSS: CANNABIS MORE THAN 30GMS	STREET	True	False	...	34.0	73.0	18	1170807.0	1836426.0	2011	02/10/2018 03:50:01 PM	41.706626	-87.650121	(41.70662559, -87.650120579)
1749207	6289962	HP364384	05/30/2008 04:00:00 AM	082XX S DR MARTIN LUTHER KING JR DR	0915	MOTOR VEHICLE THEFT	TRUCK, BUS, MOTOR HOME	STREET	False	False	...	6.0	44.0	07	1180335.0	1850445.0	2008	02/28/2018 03:56:25 PM	41.744883	-87.614801	(41.744882725, -87.614801011)
3160553	2614009	HJ217187	03/03/2003 07:00:00 PM	022XX N SEDGWICK ST	0313	ROBBERY	ARMED: OTHER DANGEROUS WEAPON	SIDEWALK	False	False	...	43.0	7.0	03	1173223.0	1915036.0	2003	02/10/2018 03:50:01 PM	41.922286	-87.638947	(41.922286039, -87.638947142)

Dataset description



Data Collection-Sample Data Collected From data.gov

	ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	Domestic	...	Ward	Community Area	FBI Code	X Coordinate	Y Coordinate	Year	Updated On	Latitude	Longitude	Location
7458441	11508368	JB517219	11/14/2018 06:00:00 PM	030XX N Southport Ave	1153	DECEPTIVE PRACTICE	FINANCIAL IDENTITY THEFT OVER $ 300	NaN	False	False	...	32.0	6.0	11	NaN	NaN	2018	11/21/2018 04:14:47 PM	NaN	NaN	NaN
2136615	1428950	G151176	03/15/2001 06:00:00 PM	005XX W CORNELIA AV	0560	ASSAULT	SIMPLE	RESIDENCE	False	True	...	NaN	NaN	08A	1171891.0	1923756.0	2001	09/07/2021 03:41:02 PM	41.946244	-87.643583	(41.946243567, -87.643583496)
5404657	8161188	HT395984	07/13/2011 11:23:00 PM	103XX S ABERDEEN ST	1812	NARCOTICS	POSS: CANNABIS MORE THAN 30GMS	STREET	True	False	...	34.0	73.0	18	1170807.0	1836426.0	2011	02/10/2018 03:50:01 PM	41.706626	-87.650121	(41.70662559, -87.650120579)
1749207	6289962	HP364384	05/30/2008 04:00:00 AM	082XX S DR MARTIN LUTHER KING JR DR	0915	MOTOR VEHICLE THEFT	TRUCK, BUS, MOTOR HOME	STREET	False	False	...	6.0	44.0	07	1180335.0	1850445.0	2008	02/28/2018 03:56:25 PM	41.744883	-87.614801	(41.744882725, -87.614801011)
3160553	2614009	HJ217187	03/03/2003 07:00:00 PM	022XX N SEDGWICK ST	0313	ROBBERY	ARMED: OTHER DANGEROUS WEAPON	SIDEWALK	False	False	...	43.0	7.0	03	1173223.0	1915036.0	2003	02/10/2018 03:50:01 PM	41.922286	-87.638947	(41.922286039, -87.638947142)

Data Pre-processing

Imputing the missing values using mean and mode for numerical and categorical columns respectively

```
num_cols = data.select_dtypes(include=np.number).columns
imputer = SimpleImputer(strategy='mean')
data[num_cols] = imputer.fit_transform(data[num_cols])

cat_cols = data.select_dtypes(include='object').columns
imputer = SimpleImputer(strategy='most_frequent')
data[cat_cols] = imputer.fit_transform(data[cat_cols])
```

```
# Checking for duplicate rows
print("Duplicate rows:", data.duplicated().sum())

# Dropping duplicate rows (if necessary)
data = data.drop_duplicates()
```

```
Duplicate rows: 0
```

```
Samples from the pre-processed dataset:
                ID Case Number                    Date  \
4080644  3835285.0    HL207557  02/28/2005 10:20:00 AM
4197749  3976341.0    HL334684  05/04/2005 12:13:12 PM
6427189  9820732.0    HX470166  10/11/2014 09:30:00 PM
1241169  5218217.0    HM801679  12/30/2006 10:00:00 AM
6131743  9298045.0    HW443264  09/07/2013 11:30:00 PM

                                  Block  IUCR    Primary Type  \
4080644                  025XX W MEDILL AVE  0610       BURGLARY
4197749                    032XX W 111TH ST  0860          THEFT
6427189  047XX S DR MARTIN LUTHER KING JR DR  2825  OTHER OFFENSE
1241169                  067XX N WESTERN AVE  0820          THEFT
6131743                    053XX W 53RD PL  0486        BATTERY
```

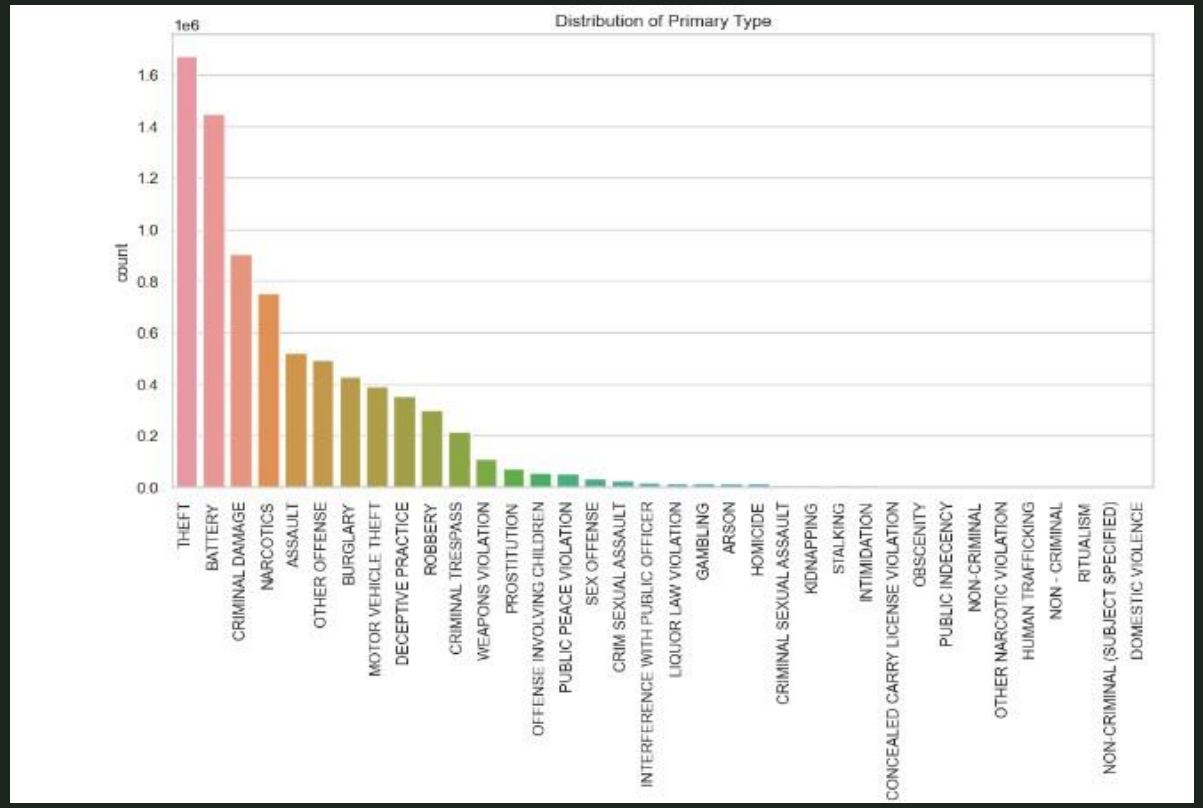
Before Imputation

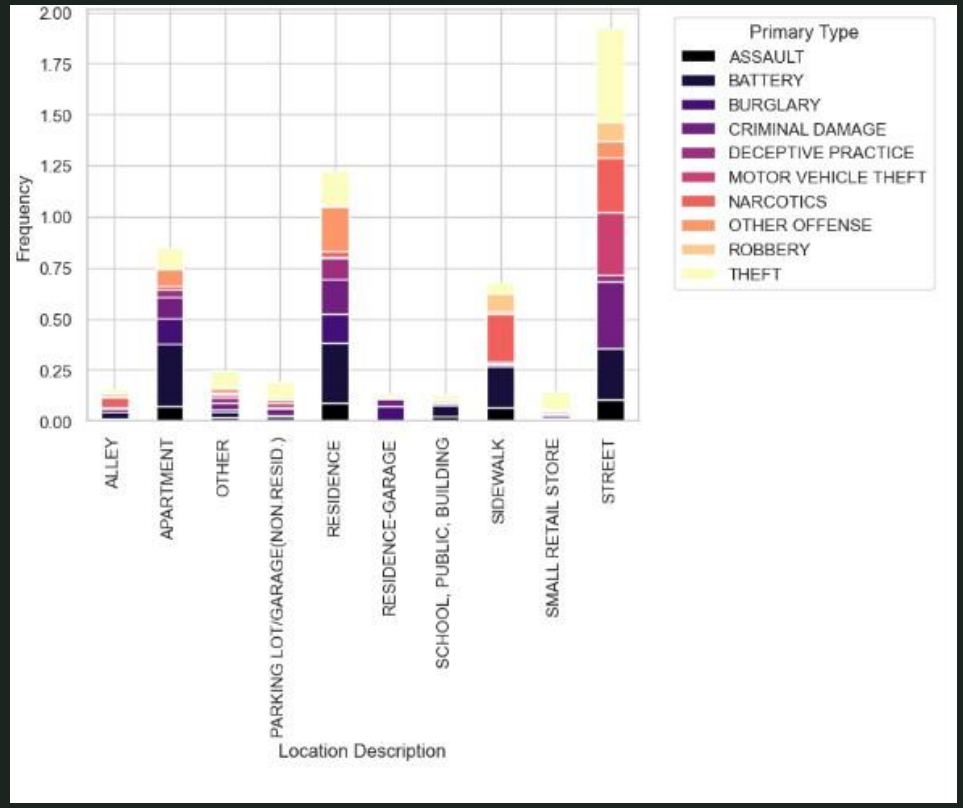
```
Missing values:
 ID                          0
Case Number                  0
Date                         0
Block                        0
IUCR                         0
Primary Type                 0
Description                  0
Location Description     11981
Arrest                       0
Domestic                     0
Beat                         0
District                    47
Ward                    614853
Community Area          613477
FBI Code                     0
X Coordinate             90298
Y Coordinate             90298
Year                         0
Updated On                   0
Latitude                 90298
Longitude                90298
Location                 90298
dtype: int64
```

After Imputation

```
Missing values:
 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
X Coordinate                 0
Y Coordinate                 0
Year                         0
Updated On                   0
Latitude                     0
Longitude                    0
Location                     0
dtype: int64
```

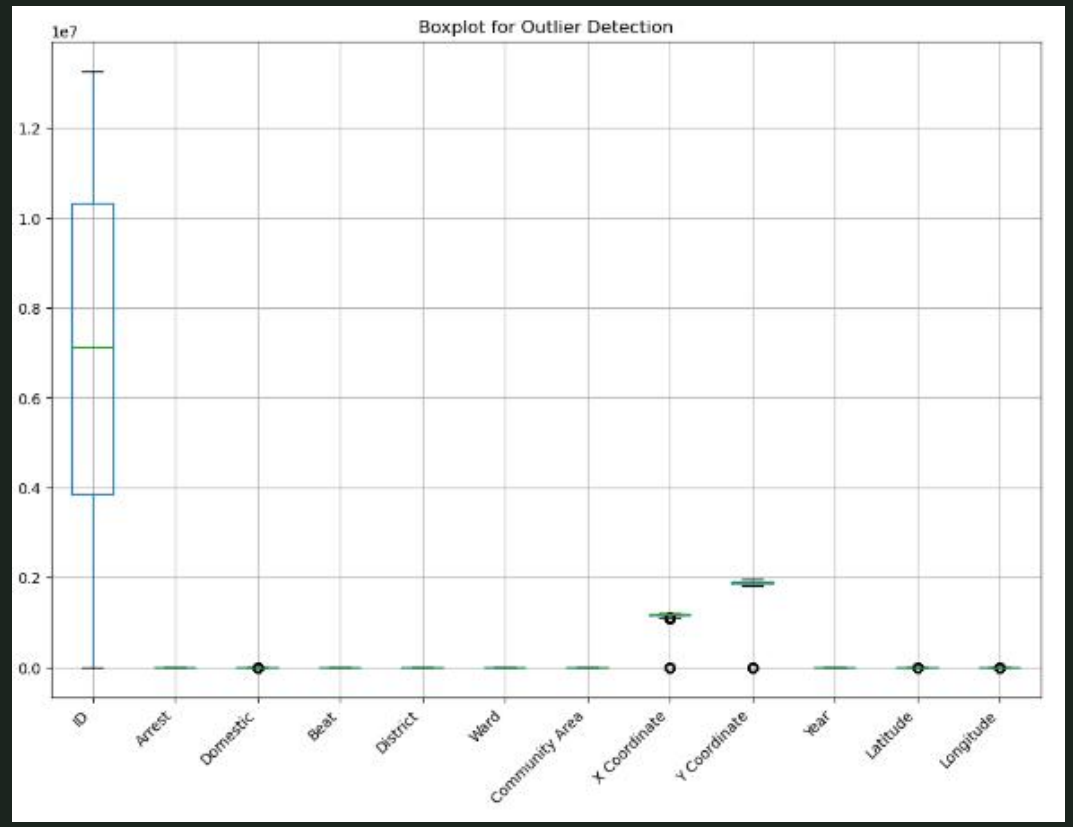
Exploratory Data Analysis



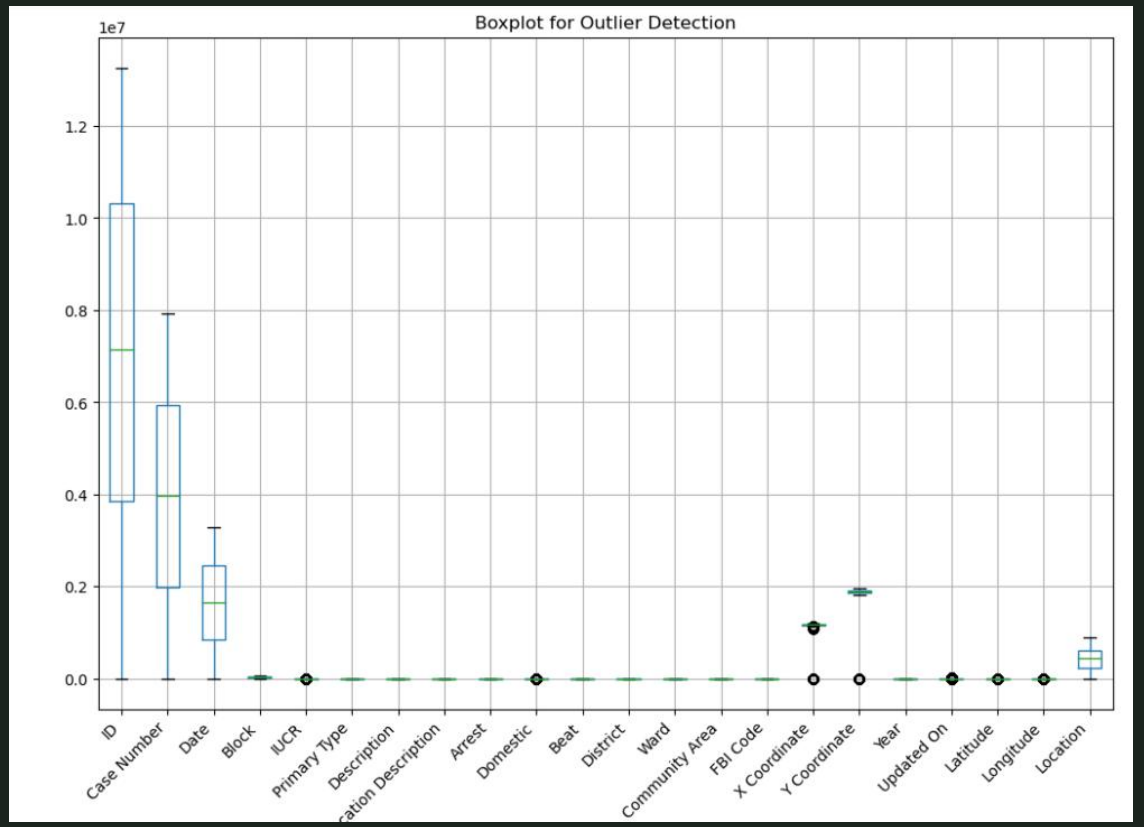


Outlier detection analysis

Outlier analysis before encoding

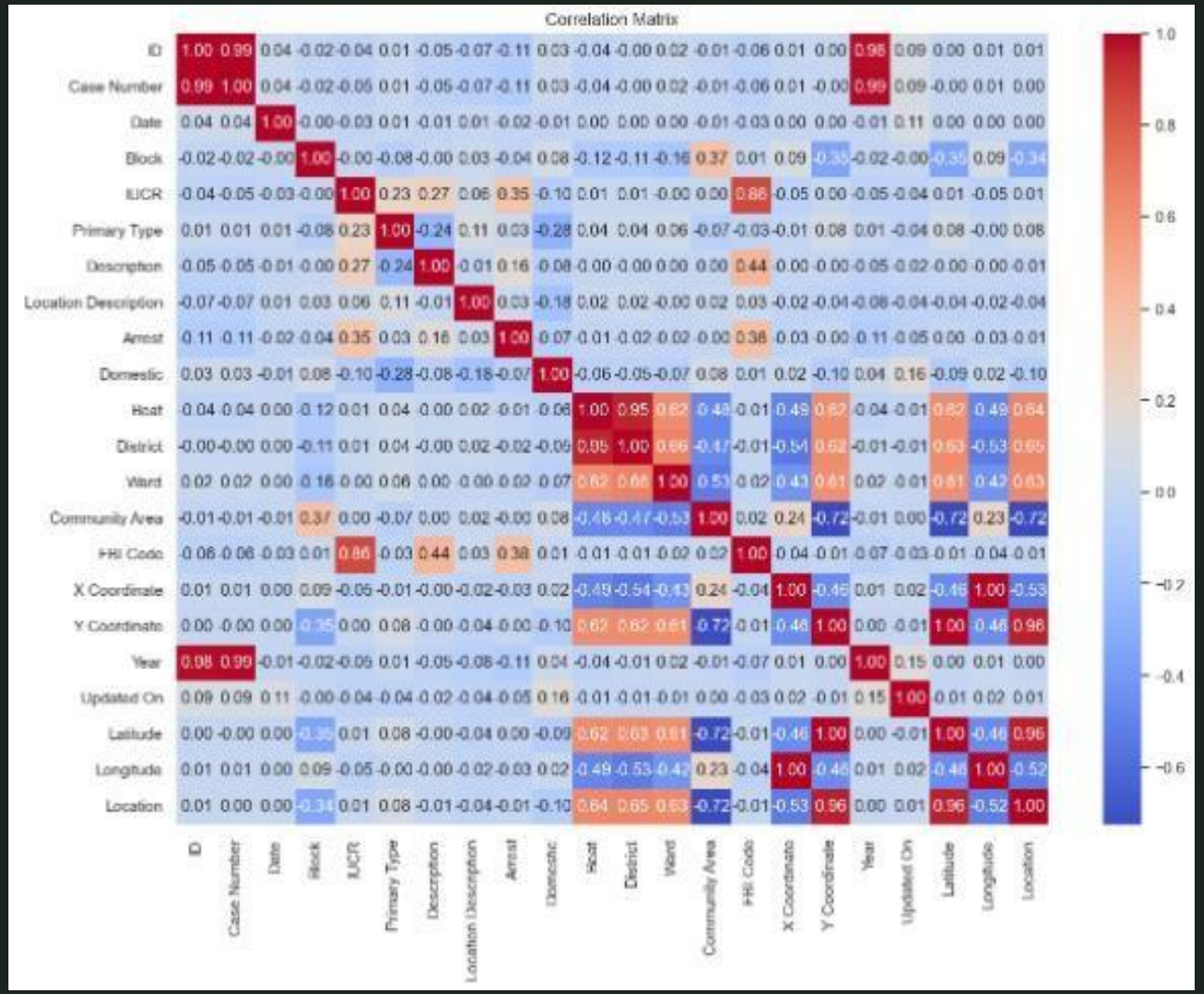


Outlier analysis after encoding



Correlation analysis

- Correlation analysis provides a basis for understanding potential relationships between the target variable 'Primary Type' and other variables.
- Guides the development of features and models for predicting crime types.



Data Transformation


```
# Label encoding for categorical variables
label_encoder = LabelEncoder()
for col in cat_cols:
    data[col] = label_encoder.fit_transform(data[col])
```

```
data.head(5)
```

	ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	Domestic	...	Ward	Community Area	FBI Code	X Coordinate	Y Coordin
0	11037294.0	6397683	653292	1219	135	9	232	23	False	False	...	42.0	32.0	13	1.164614e+06	1.885812e
1	11646293.0	6844308	3188902	21623	136	9	231	17	False	False	...	36.0	19.0	13	1.164614e+06	1.885812e
2	11645836.0	6843484	1048858	44415	135	9	232	189	False	False	...	15.0	63.0	13	1.164614e+06	1.885812e
3	11645959.0	6843050	3188928	37627	315	26	470	160	False	False	...	33.0	14.0	9	1.164614e+06	1.885812e
4	11645601.0	6843826	1338751	57489	135	9	232	160	False	False	...	21.0	71.0	13	1.164614e+06	1.885812e

5 rows x 22 columns

```
from sklearn.linear_model import LassoCV
from sklearn.preprocessing import StandardScaler

X = data.drop(['Primary Type'], axis=1)

# Target variable
y = data['Primary Type']

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

# Apply L1 regularization with cross-validated selection of the best alpha
lasso = LassoCV(cv=5)
lasso.fit(X_scaled, y)

# Getting selected features with non-zero coefficients
selected_features = X.columns[lasso.coef_ != 0]

print("Selected Features:", selected_features)
```

```
Selected Features: Index(['ID', 'Date', 'Block', 'IUCR', 'Description', 'Location Description'
       'Domestic', 'District', 'Ward', 'FBI Code', 'Updated On', 'Longitude',
       'Location'],
      dtype='object')
```

	Date	Year	Longitude	Latitude	Location Description	Crime_Type	Description
0	653292	2015.0	-87.671447	41.842264	23	9	232
1	3188902	2018.0	-87.671447	41.842264	17	9	231
2	1048858	2016.0	-87.671447	41.842264	189	9	232
3	3188928	2018.0	-87.671447	41.842264	160	26	470
4	1338751	2014.0	-87.671447	41.842264	160	9	232
5	2223711	2018.0	-87.671447	41.842264	160	34	360
6	7618	2018.0	-87.671447	41.842264	160	9	232
7	778818	2018.0	-87.671447	41.842264	160	9	232
8	276626	2015.0	-87.671447	41.842264	140	9	232
9	1084232	2012.0	-87.671447	41.842264	189	9	232

- Label encoding is used to transform categorical values into numerical values.
- Lasso Regression is used for feature selection.
- 'Primary Type' is renamed to 'Crime_Type'.

Data Preparation


```
# Split the data into training and temporary (validation and testing) sets.
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
# Split the remaining data into validation and testing sets.
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.33, random_state=42)
```

- Training dataset is used to capture underlying patterns.
- The validation dataset is for hyperparameter tuning purpose.
- The testing dataset is employed to evaluate the performance

Sample of training data

	Date	Year	Longitude	Latitude	Location Description	Description
5949304	599581	2013.0	-87.722439	41.765698	189	360
6319275	2269663	2014.0	-87.666387	41.907591	194	480
1723114	97205	2008.0	-87.688446	41.953343	189	226
849439	113340	2006.0	-87.576946	41.761597	160	498
4350847	2527213	2008.0	-87.627877	41.883500	167	246
...	...	...	...	...	...	...
6550634	485494	2015.0	-87.748024	41.878561	189	152
7705870	2879512	2019.0	-87.723777	41.860119	160	211
6423388	2279228	2014.0	-87.562069	41.760013	160	45
6962611	2502306	2016.0	-87.599101	41.750483	17	502
6413414	2756687	2014.0	-87.684843	41.778618	184	301

5548500 rows × 6 columns

Data Preparation

Sample From the Validation Dataset

	Date	Year	Longitude	Latitude	Location Description	Description
4648458	2600019	2009.0	-87.711915	41.918549	189	1
3614151	172214	2004.0	-87.587278	41.797068	140	429
6061234	1846473	2013.0	-87.602739	41.750578	189	467
1057200	2340784	2006.0	-87.701661	41.985458	189	498
5105941	2230515	2010.0	-87.653898	41.933786	25	190
...	...	...	...	...	...	...
7576310	986329	2019.0	-87.646998	41.684452	189	505
6931455	2419357	2016.0	-87.665190	41.881470	148	498
1385695	1402294	2007.0	-87.684274	41.929744	82	429
1029435	614469	2006.0	-87.645694	41.785193	184	381
7826931	716641	2020.0	-87.715892	41.961095	185	454

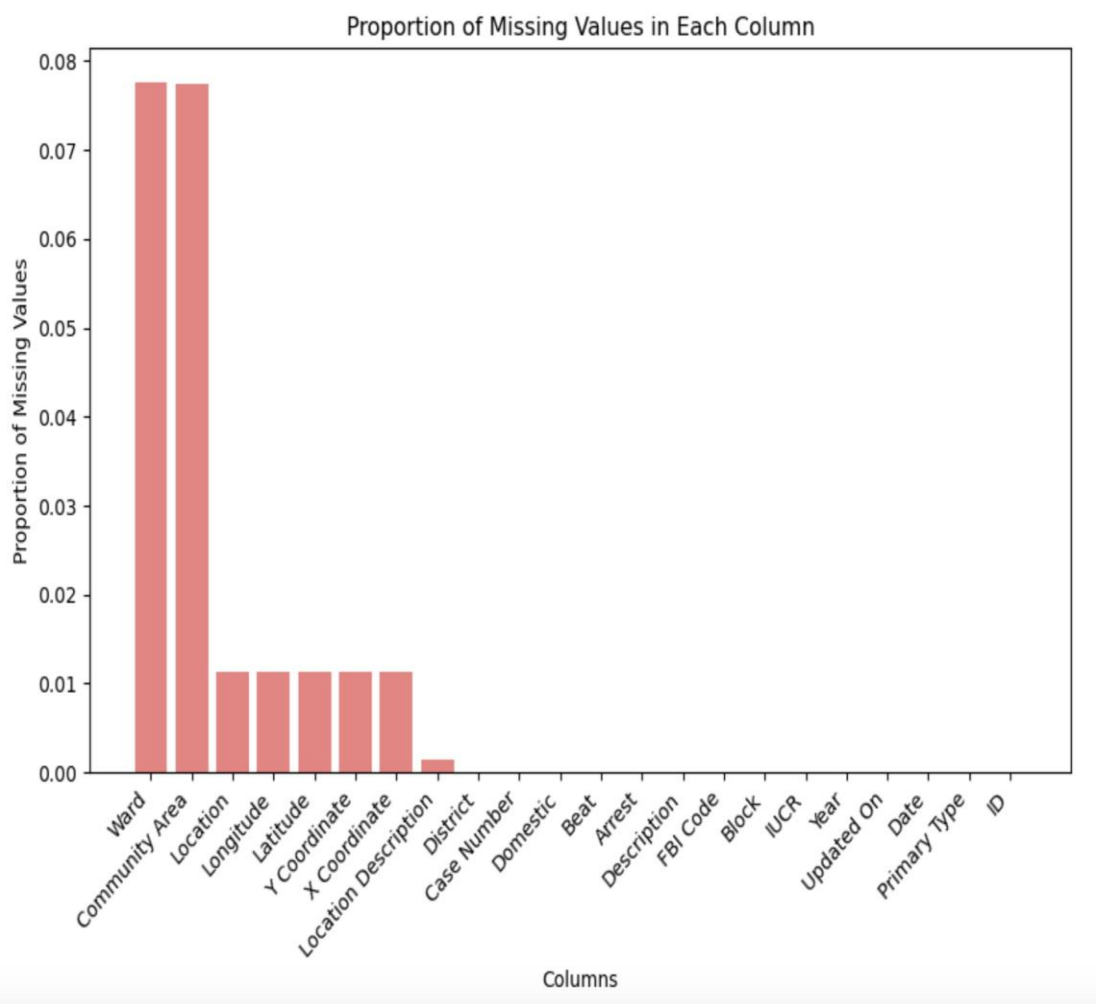
1593212 rows × 6 columns

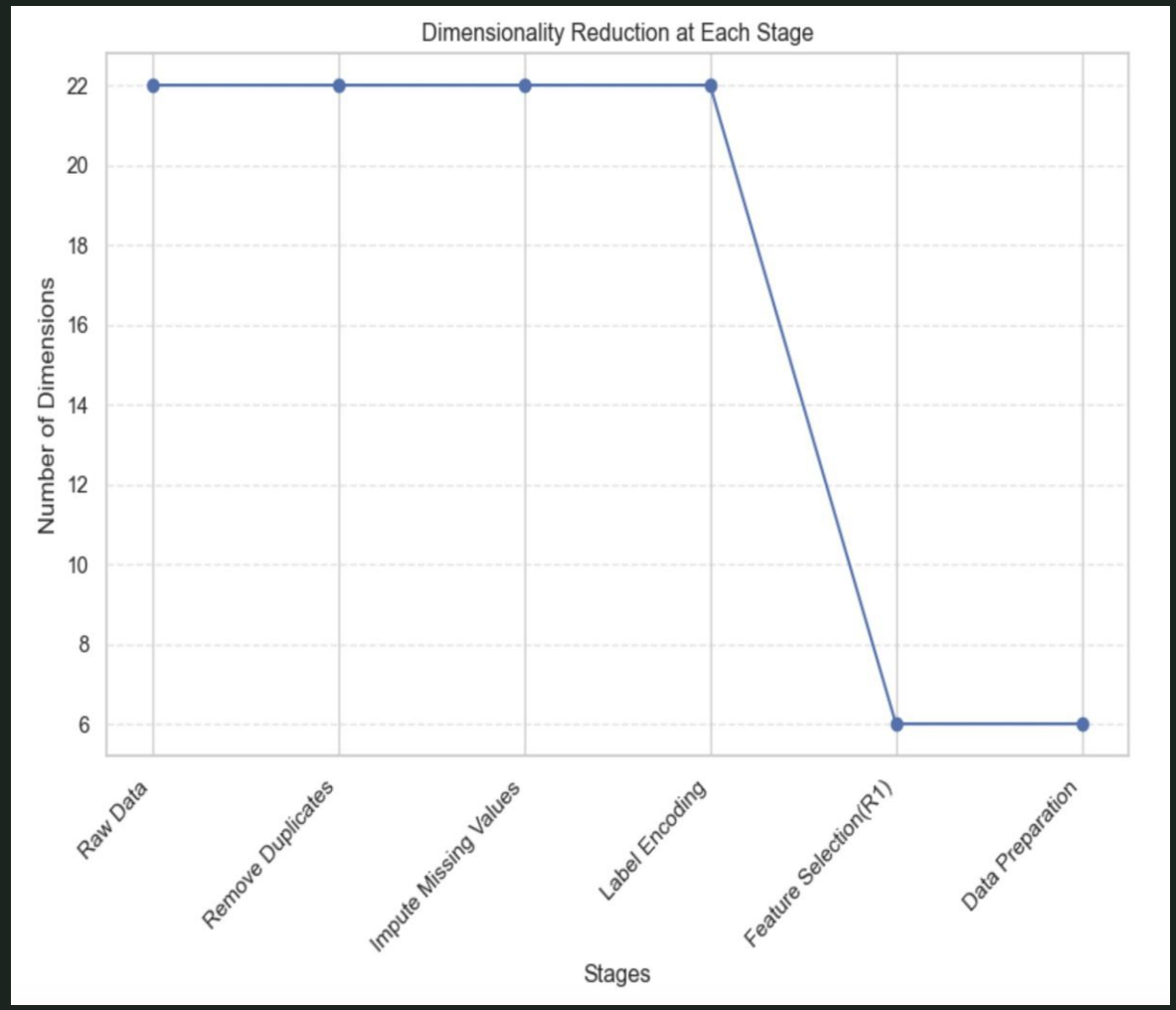
Sample From the Testing Dataset

	Date	Year	Longitude	Latitude	Location Description	Description
5121501	2564409	2010.0	-87.608062	41.816793	184	381
155447	2958659	2005.0	-87.631409	41.826839	189	152
1440864	1165282	2007.0	-87.726352	41.870012	160	26
2032759	1782732	2023.0	-87.705127	41.847467	17	493
6609629	3075823	2015.0	-87.591359	41.795511	189	498
...	...	...	...	...	...	...
17897	2346927	2023.0	-87.660173	41.783166	189	498
3338948	1200695	2003.0	-87.712122	41.899143	148	152
4097595	552334	2005.0	-87.660105	41.750404	167	492
5068135	141262	2011.0	-87.653251	41.721332	184	467
726313	2848612	2005.0	-87.624244	41.894540	208	1

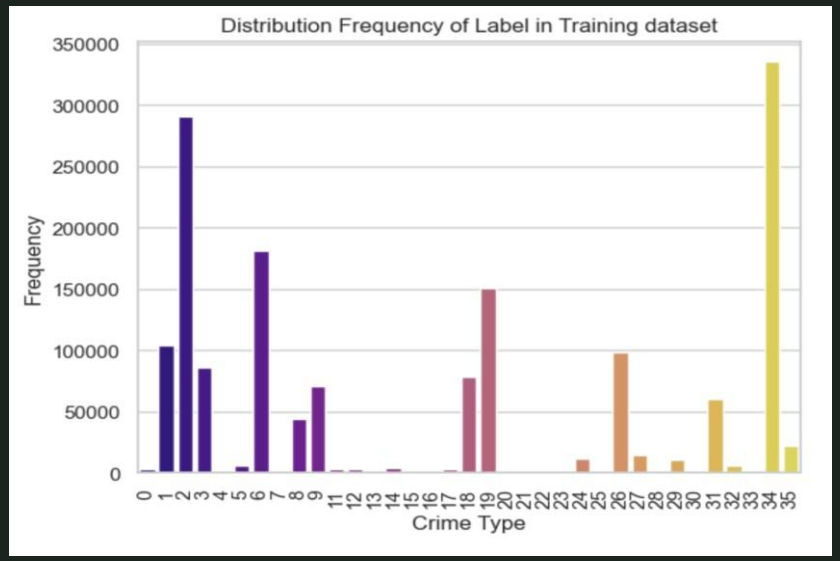
784717 rows × 6 columns

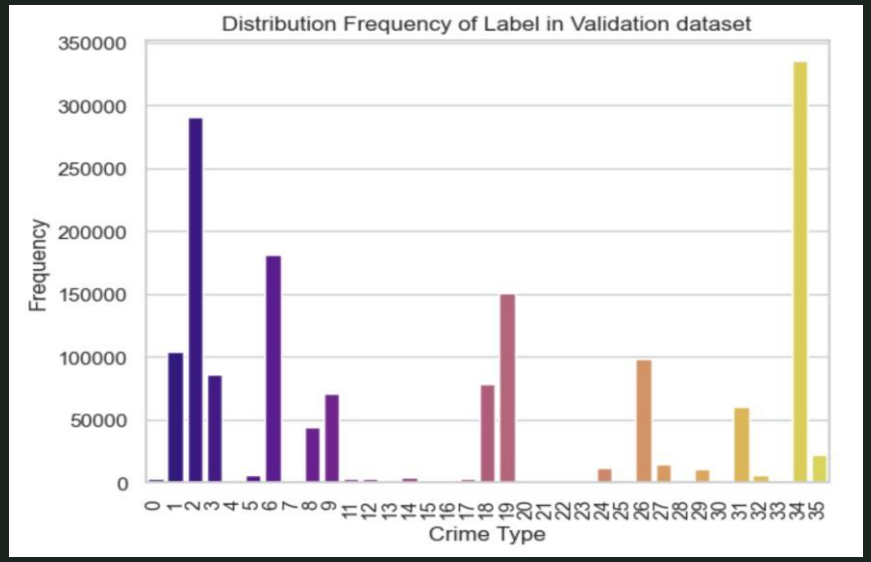
Data Statistics

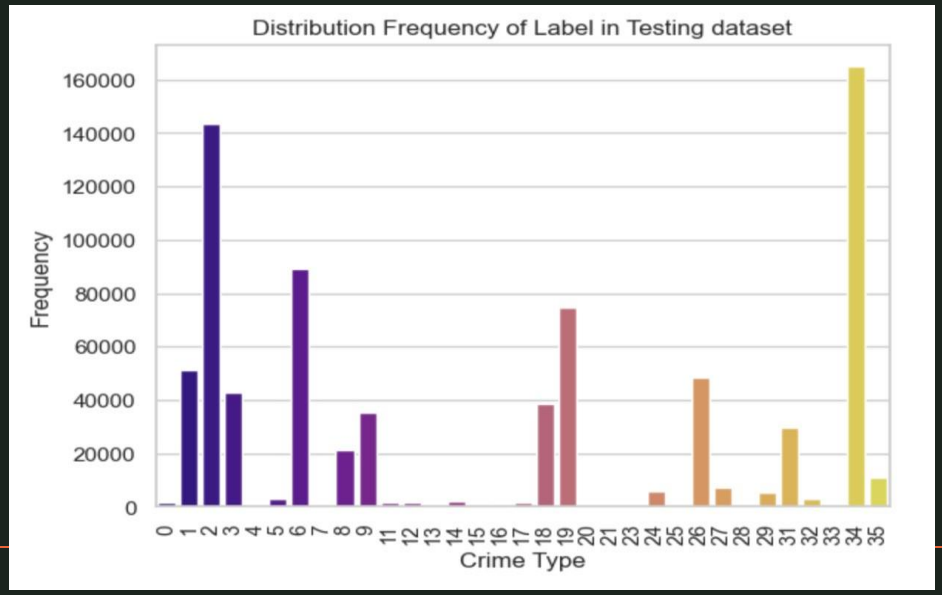




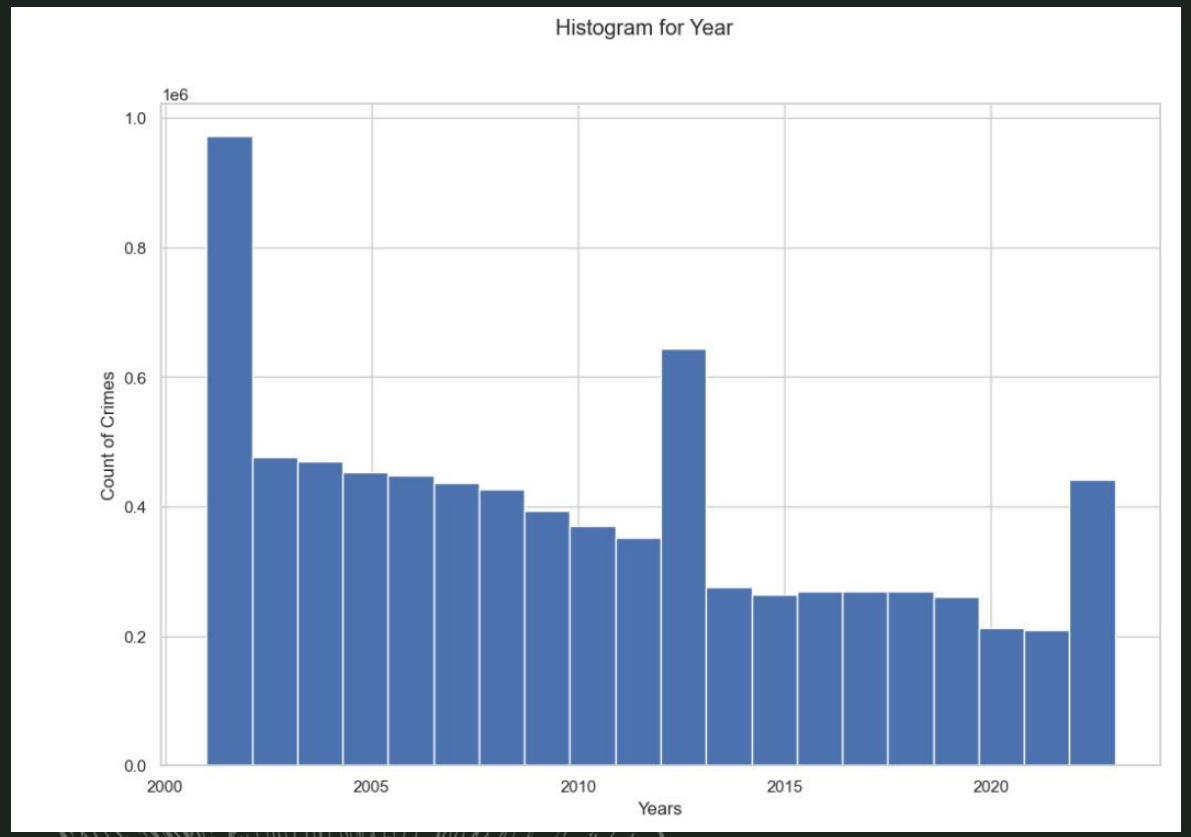
Data Statistics

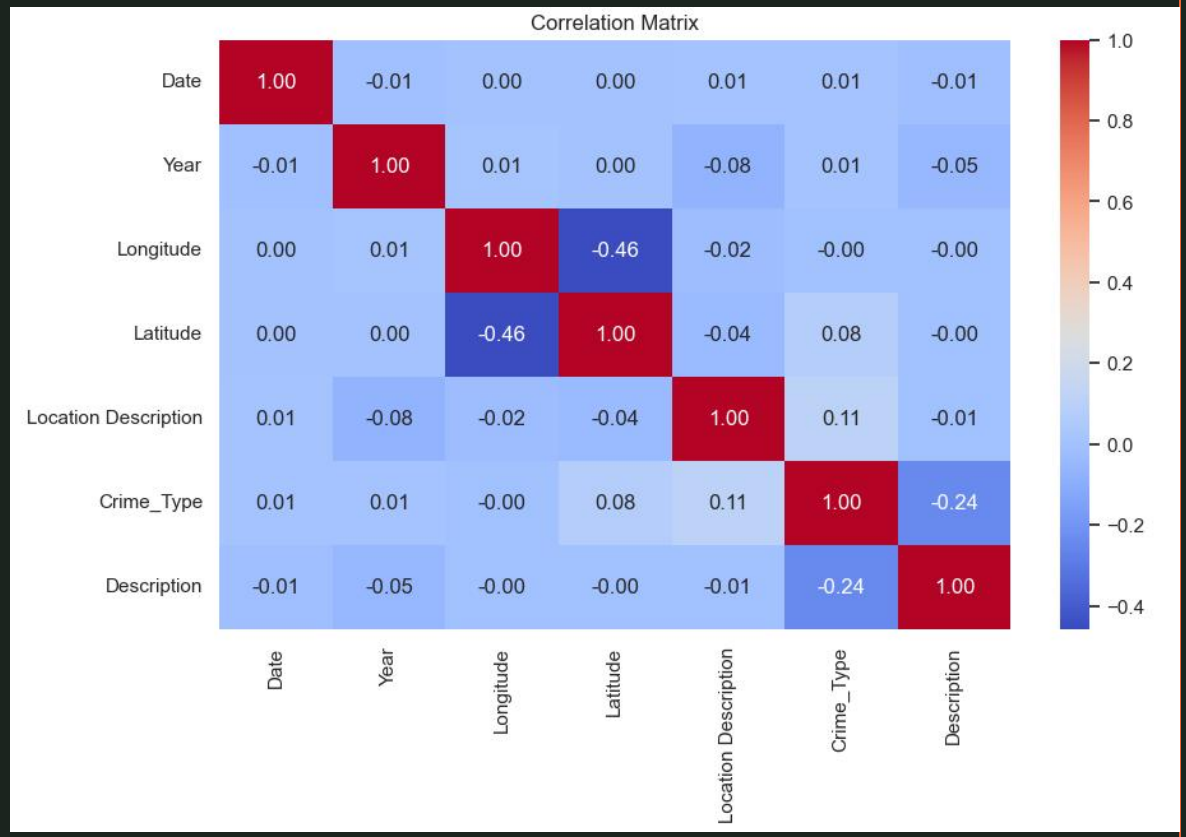




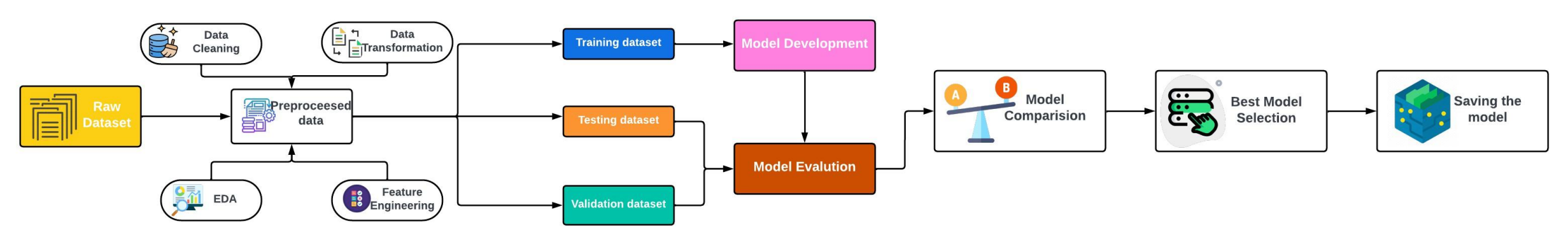


Data Analytics





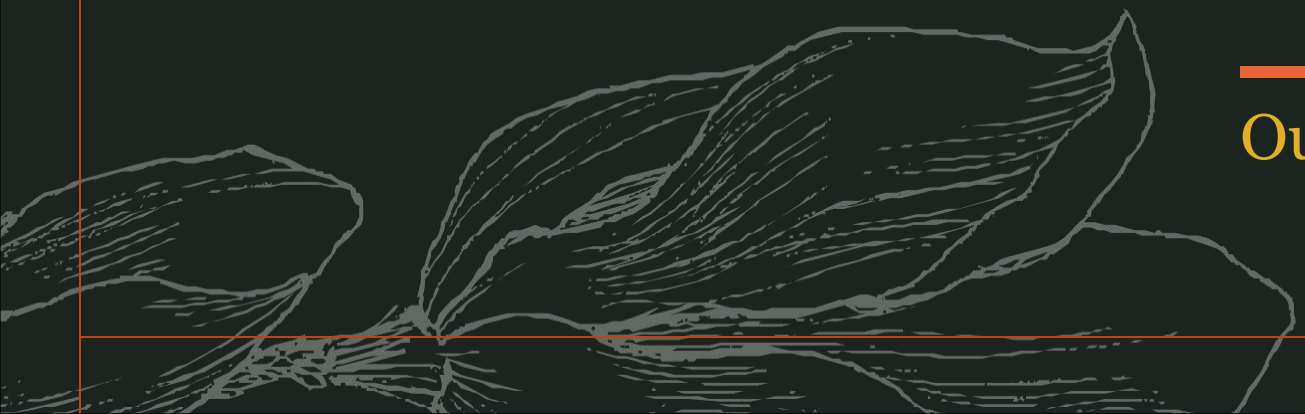
Project Architecture



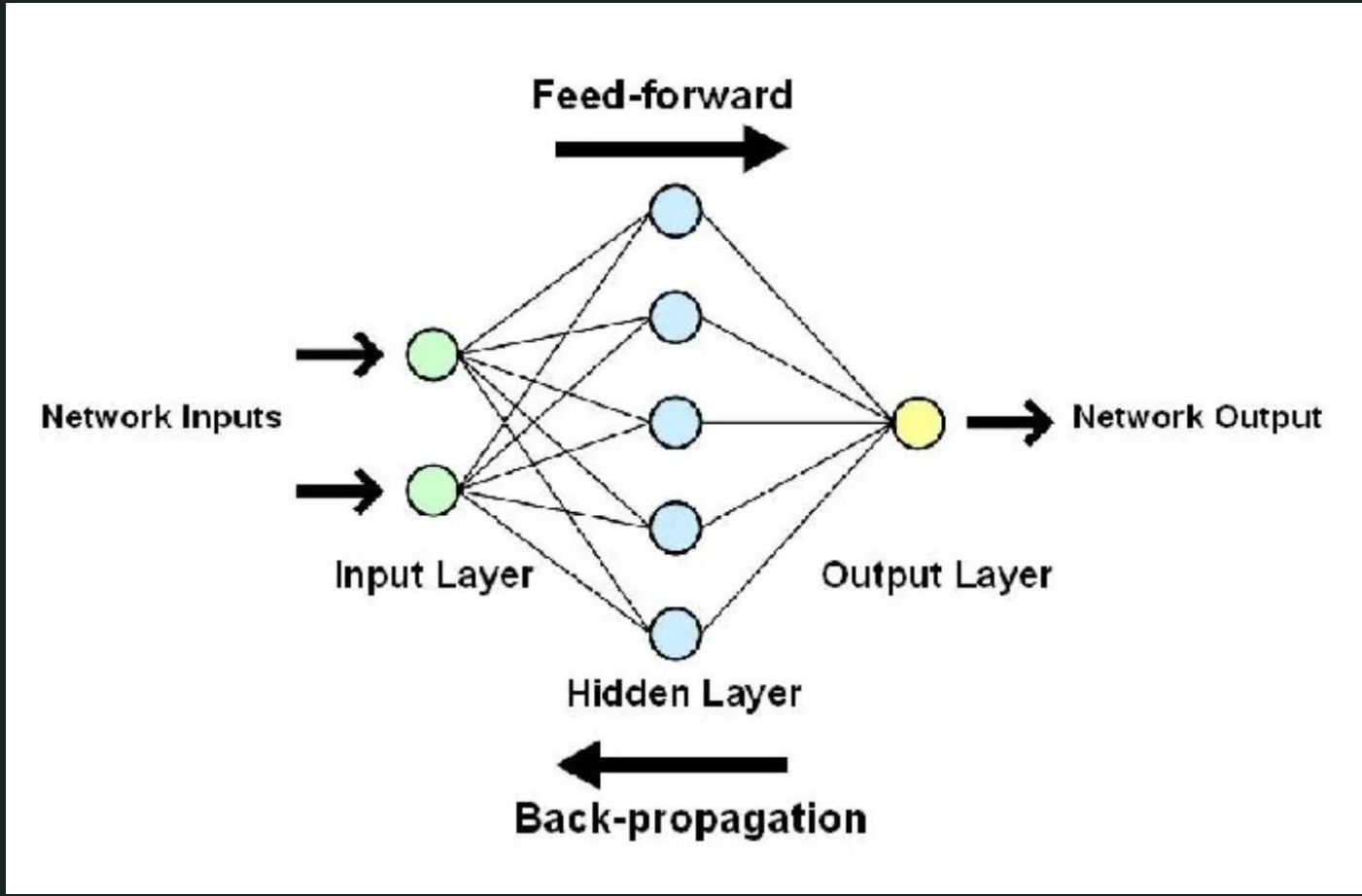
- Started with a raw dataset and preprocessed it using multiple techniques.
- Divided the preprocessed dataset into 3 parts : 70% into Training, 20% into Validation and 10% into Testing.
- Developed Machine Learning models on Training dataset.
- Evaluated the performance of the models using Validation and Testing datasets.
- Compared models based on evaluation metrics to find the best performing model.
- Saved the Baseline and Hyper tuned best model.

Modelling

- Model Proposals
- Model Support
- Model Building
- Model evaluation and comparison
- Our Best Model



Model proposal-Artificial Neural Networks



- A deep learning model inspired by the human brain's structure.
- Consists of three layers: input , hidden and output layers, connected by the neurons.
- Feedforward phase: Input data is passed through the layers, producing a prediction.
- Backward Phase: Evaluates the error and updates the weights to improve the performance.

Model Support- Artificial Neural Networks

Baseline Model Implementation

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
# add the hidden layer
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
# add the outputlayer
model.add(Dense(num_classes, activation='softmax'))
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train_onehot, epochs=5, batch_size=32, validation_data=(X_val,y_val_onehot))
```

Baseline Model Training Report

```
Epoch 1/5
173391/173391 [==============================] - 131s 752us/step - loss: 1.0410 - accuracy: 0.6813 - val_loss: 0.83
01 - val_accuracy: 0.7327
Epoch 2/5
173391/173391 [==============================] - 134s 774us/step - loss: 0.7256 - accuracy: 0.7638 - val_loss: 0.66
53 - val_accuracy: 0.7861
Epoch 3/5
173391/173391 [==============================] - 132s 763us/step - loss: 0.6398 - accuracy: 0.7886 - val_loss: 0.61
90 - val_accuracy: 0.7938
Epoch 4/5
173391/173391 [==============================] - 130s 748us/step - loss: 0.6031 - accuracy: 0.7973 - val_loss: 0.59
33 - val_accuracy: 0.7925
Epoch 5/5
173391/173391 [==============================] - 132s 763us/step - loss: 0.5795 - accuracy: 0.8035 - val_loss: 0.57
21 - val_accuracy: 0.8066
```

Hyperparameters:

- One hidden layer with 64 neurons
- Epochs: Number of iterations over the entire dataset
- Batch_size: number of instances processed before updating the weights
- Loss: Loss function used to evaluate the model

Model Building-Artificial Neural Networks

```
model = Sequential()
# add hidden layers
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train_onehot, epochs=10, batch_size=64, validation_data=(X_val,y_val_onehot))
```

```
Epoch 1/10
86696/86696 [==============================] - 89s 1ms/step - loss: 0.4676 - accuracy: 0.8262 - val_loss: 0.4179 -
val_accuracy: 0.8565
Epoch 2/10
86696/86696 [==============================] - 89s 1ms/step - loss: 0.3078 - accuracy: 0.8746 - val_loss: 0.3175 -
val_accuracy: 0.8746
Epoch 3/10
86696/86696 [==============================] - 89s 1ms/step - loss: 0.2589 - accuracy: 0.8922 - val_loss: 0.2505 -
val_accuracy: 0.8914
Epoch 4/10
86696/86696 [==============================] - 89s 1ms/step - loss: 0.2330 - accuracy: 0.9017 - val_loss: 0.2526 -
val_accuracy: 0.8940
Epoch 5/10
86696/86696 [==============================] - 87s 1ms/step - loss: 0.2147 - accuracy: 0.9084 - val_loss: 0.2307 -
val_accuracy: 0.8983
Epoch 6/10
86696/86696 [==============================] - 88s 1ms/step - loss: 0.2040 - accuracy: 0.9119 - val_loss: 0.1914 -
val_accuracy: 0.9144
Epoch 7/10
86696/86696 [==============================] - 89s 1ms/step - loss: 0.1961 - accuracy: 0.9146 - val_loss: 0.1707 -
val_accuracy: 0.9242
Epoch 8/10
86696/86696 [==============================] - 729s 8ms/step - loss: 0.1921 - accuracy: 0.9160 - val_loss: 0.1795 -
val_accuracy: 0.9180
Epoch 9/10
86696/86696 [==============================] - 89s 1ms/step - loss: 0.1891 - accuracy: 0.9171 - val_loss: 0.2326 -
val_accuracy: 0.8992
Epoch 10/10
86696/86696 [==============================] - 89s 1ms/step - loss: 0.1864 - accuracy: 0.9180 - val_loss: 0.1780 -
val_accuracy: 0.9209
```

Hyperparameters:

- Hidden Layers: 5
- Nodes in hidden layers: 64
- Epochs: 10
- Batch_size: 64
- Loss: Cross Entropy

Performance

- After Hyper tuning the model, the accuracy increased from 81% to 92% and f1 score increased from 79% to 91%.
- The running time decreases from 130s to 90s.

Model Evaluation- Artificial Neural Networks

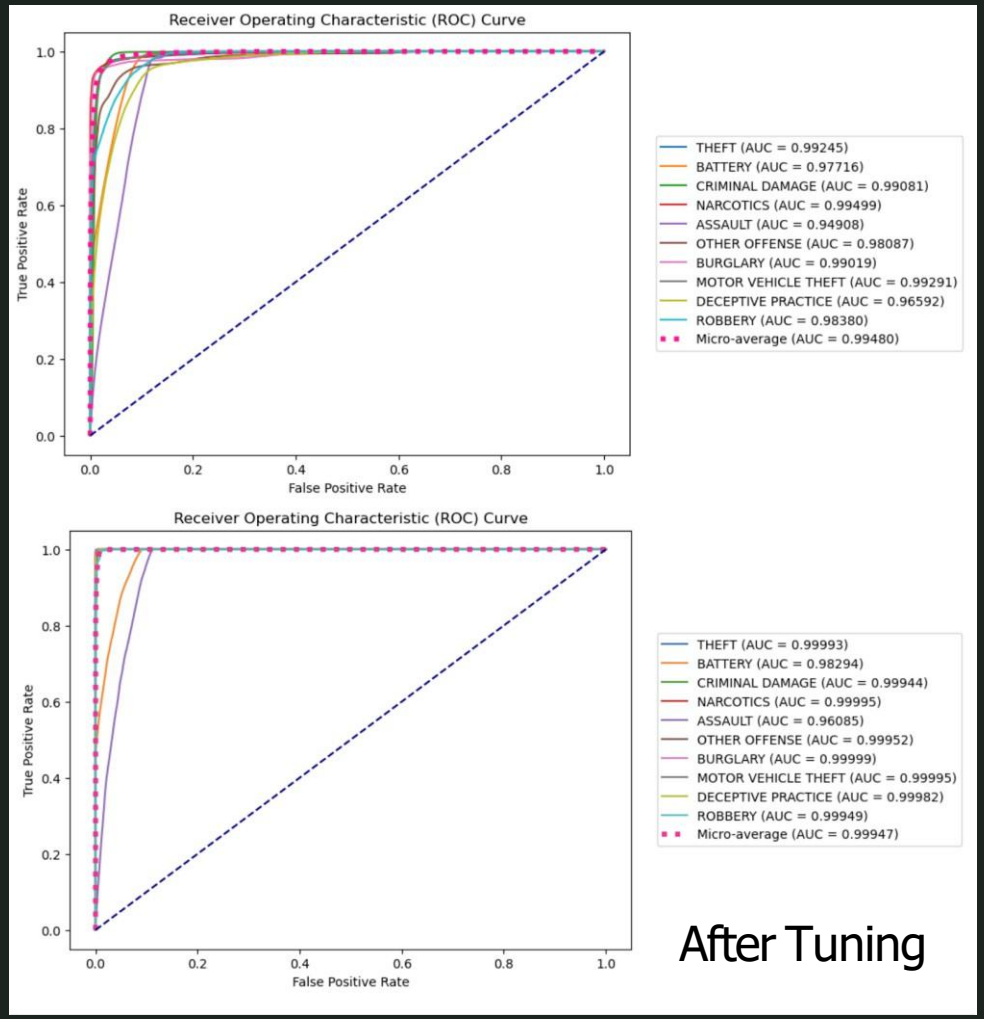
Classification report Before Tuning

	precision	recall	f1-score	support
0	0.78	0.37	0.50	1281
1	0.48	0.31	0.37	51657
2	0.74	0.87	0.80	143368
3	0.89	0.90	0.90	42516
4	0.47	0.26	0.34	103
5	0.77	0.52	0.62	2775
6	0.82	0.96	0.88	89570
7	0.61	0.52	0.56	752
8	0.82	0.20	0.32	21757
9	0.61	0.55	0.58	34693
11	0.75	0.55	0.64	1504
12	0.41	0.51	0.45	1255
13	0.00	0.00	0.00	10
14	0.71	0.33	0.46	1860
15	0.00	0.00	0.00	451
16	0.33	0.00	0.01	719
17	0.43	0.14	0.21	1514
18	0.77	0.90	0.83	38549
19	0.96	0.90	0.93	74439
20	0.00	0.00	0.00	3
21	0.00	0.00	0.00	15
22	0.00	0.00	0.00	1
23	0.00	0.00	0.00	67
24	0.74	0.28	0.40	5654
25	0.00	0.00	0.00	16
26	0.75	0.84	0.79	48629
27	0.71	0.81	0.76	6908
28	0.00	0.00	0.00	20
29	0.53	0.48	0.51	5217
30	0.00	0.00	0.00	3
31	0.83	0.69	0.75	29495
32	0.40	0.01	0.02	3131
33	0.00	0.00	0.00	503
34	0.92	0.94	0.93	165336
35	0.74	0.93	0.82	10946
accuracy			0.81	784717
macro avg	0.49	0.39	0.41	784717
weighted avg	0.80	0.81	0.79	784717

After Tuning

	precision	recall	f1-score	support
0	0.98	0.83	0.90	1281
1	0.56	0.36	0.44	51657
2	0.78	0.90	0.84	143368
3	0.99	1.00	0.99	42516
4	0.87	0.26	0.40	103
5	0.72	0.56	0.63	2775
6	0.97	1.00	0.99	89570
7	0.58	0.55	0.57	752
8	0.99	0.90	0.94	21757
9	0.99	0.98	0.98	34693
11	1.00	0.92	0.96	1504
12	0.98	1.00	0.99	1255
13	0.00	0.00	0.00	10
14	0.91	0.78	0.84	1860
15	0.75	0.97	0.84	451
16	0.69	0.74	0.71	719
17	0.92	0.84	0.88	1514
18	0.99	0.98	0.99	38549
19	1.00	0.99	1.00	74439
20	0.00	0.00	0.00	3
21	0.00	0.00	0.00	15
22	0.00	0.00	0.00	1
23	0.36	0.22	0.28	67
24	0.92	0.77	0.84	5654
25	0.00	0.00	0.00	16
26	0.96	0.97	0.96	48629
27	0.98	0.80	0.88	6908
28	0.00	0.00	0.00	20
29	0.95	0.91	0.93	5217
30	0.00	0.00	0.00	3
31	0.94	0.94	0.94	29495
32	0.81	0.66	0.73	3131
33	0.00	0.00	0.00	503
34	0.99	1.00	1.00	165336
35	0.98	0.99	0.98	10946
accuracy			0.92	784717
macro avg	0.67	0.62	0.64	784717
weighted avg	0.91	0.92	0.91	784717

ROC Curve Before Tuning



After Tuning

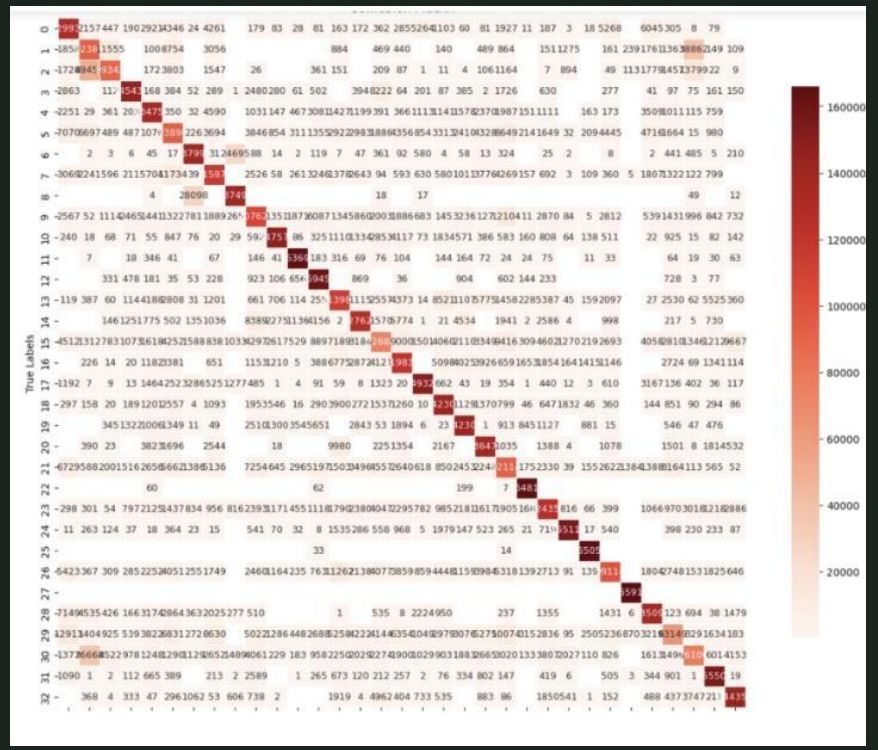
Model Evaluation- Artificial Neural Networks

Classification Report After Balancing

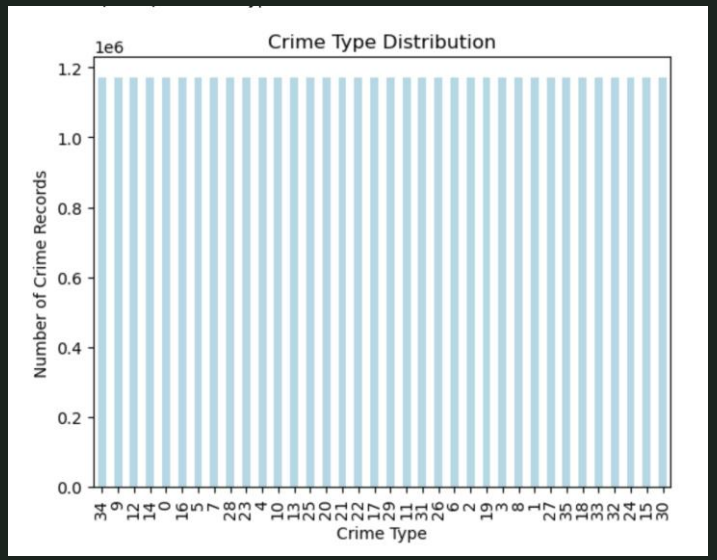
Five Hidden Layer ANN Classification Report:

	precision	recall	f1-score	support
0	0.60	0.85	0.71	165873
1	0.47	0.61	0.53	165979
2	0.75	0.59	0.66	165303
3	0.93	0.95	0.94	165641
4	0.75	0.81	0.78	165408
5	0.64	0.55	0.59	164990
6	0.80	0.93	0.86	165638
7	0.75	0.67	0.71	165640
8	0.90	0.83	0.87	165755
9	0.60	0.72	0.65	165826
10	0.90	0.89	0.90	164778
11	0.97	0.99	0.98	165768
12	0.90	0.91	0.91	165865
13	0.68	0.72	0.70	165288
14	0.77	0.78	0.77	165058
15	0.58	0.50	0.54	165795
16	0.63	0.78	0.70	165902
17	0.90	0.93	0.92	165666
18	0.80	0.86	0.83	165421
19	0.99	1.00	1.00	165304
20	0.85	0.91	0.88	166037
21	1.00	1.00	1.00	165715
22	0.74	0.83	0.78	165741
23	0.60	0.56	0.58	165892
24	0.97	1.00	0.99	166196
25	0.81	0.74	0.77	165912
26	0.96	0.94	0.95	165396
27	0.98	1.00	0.99	165517
28	0.82	0.59	0.69	165671
29	0.99	1.00	0.99	166271
30	0.87	0.77	0.82	165299
31	0.66	0.39	0.49	165516
32	0.58	0.41	0.48	165482
33	0.91	0.97	0.94	165886
34	0.87	0.88	0.88	165794
accuracy			0.80	5797223
macro avg	0.80	0.80	0.79	5797223
weighted avg	0.80	0.80	0.79	5797223

Confusion Matrix After Balancing

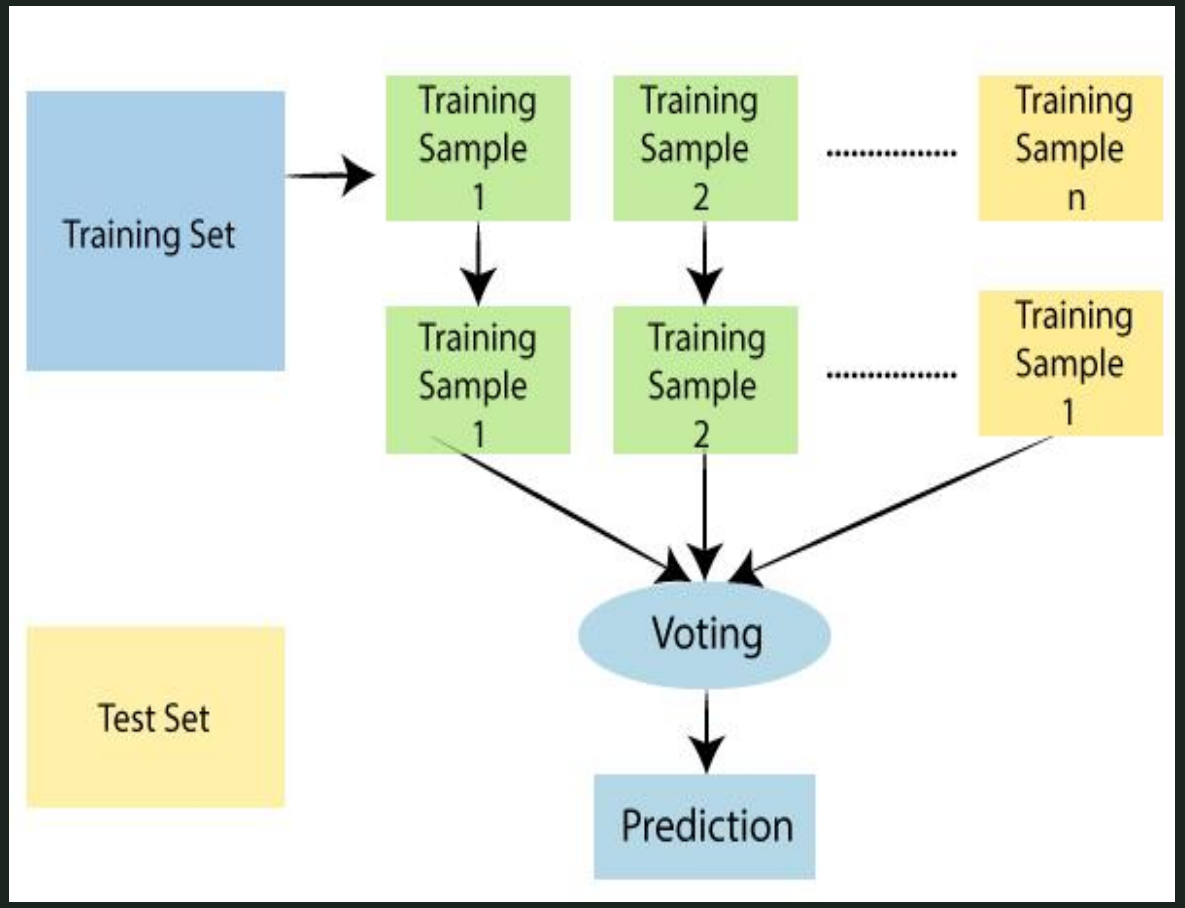


Crime Type Distribution



After Tuning

Model Proposal-Random Forest



- Random Forest uses a combination of decision trees, enabling collaborative decision-making by multiple models.
- The algorithm uses a voting system to reduce the risk of overfitting by taking into account the majority prediction from the group of trees.
- Random Forest achieves high accuracy and efficiency in predictions by consolidating forecasts from multiple trees.
- Studies, such as Abdulraheem M. et al.'s crime prediction and Guo et al.'s mining cost estimation, highlight the effectiveness of Random Forest in achieving accurate predictions and valuable insights.

Model Support- Random Forest


```
# Implementing Random Forest model
rf_model = RandomForestClassifier(
    n_estimators=150,
    max_depth=10,
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=42
)

# Model Fitting
rf_model.fit(X_train, y_train)
```

```
# Evaluate the model on the validation set
y_val_pred = rf_model.predict(X_val)

# Calculate accuracy on the validation set
accuracy = accuracy_score(y_val, y_val_pred)
print(f'Validation Accuracy: {accuracy * 100:.2f}%')

# Evaluate the model on the test set
y_test_pred = rf_model.predict(X_test)

# Calculate accuracy on the test set
accuracy_test = accuracy_score(y_test, y_test_pred)
print(f'Test Accuracy: {accuracy_test * 100:.2f}%')
```

```
Validation Accuracy: 72.97%
Test Accuracy: 73.05%
```

Baseline Model

- **n_estimators: Determines number of decision trees in each forest.**
- **max_depth: Maximum levels in each tree.**
- min_samples_split: Minimum number of branches for each tree.
- min_samples_leaf: Minimum number of samples for each leaf node.

- Based on the above parameters and their respective values, baseline model is built and its evaluation is done.
- Based on the evaluations the model performed well but in an attempt to enhance the performance hyper parameter tuning is done using Grid search.

Model Building- Random Forest


```
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold

param_dist = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

random_search = RandomizedSearchCV(
    estimator=rf_model,
    param_distributions=param_dist,
    n_iter=5,  # Adjust the number of iterations as needed
    cv=StratifiedKFold(n_splits=5),
    scoring='accuracy',
    random_state=42
)

random_search.fit(X_train, y_train)

best_params_random = random_search.best_params_
best_rf_model_random = random_search.best_estimator_

print("Best Hyperparameters (Random Search) for Random Forest:", best_params_random)
```

```
Best Hyperparameters (Random Search) for Random Forest: {'n_estimators': 50, 'min_samples_split': 2, 'min_samples_leaf': 1,
'max_depth': None}
```

```
# Evaluate the model on the validation set
y_val_pred = rf_model_tuned.predict(X_val)

# Calculate accuracy on the validation set
accuracy = accuracy_score(y_val, y_val_pred)
print(f'Validation Accuracy: {accuracy * 100:.2f}%')

# Evaluate the model on the test set
y_test_pred = rf_model_tuned.predict(X_test)

# Calculate accuracy on the test set
accuracy_test = accuracy_score(y_test, y_test_pred)
print(f'Test Accuracy: {accuracy_test * 100:.2f}%')
```

```
Validation Accuracy: 90.73%
Test Accuracy: 90.86%
```

```
# Implementing Random Forest model with best parameters
rf_model_tuned = RandomForestClassifier(
    n_estimators=50,
    min_samples_split=2,
    min_samples_leaf=1,
    max_depth=None
)

# Model Fitting
rf_model_tuned.fit(X_train, y_train)
```

Hypertuned Model

- **After Hyper tuning the model, the performance of the model progressed and the accuracy increased from 73% to 91% and f1 score from 70% to 91%.**
- **Running time decreased from 7mins to 2mins**

Model Evaluation- Random Forest

Classification report before Tuning

```
#test data classification report
y_pred = rf_model.predict(X_test)

# Generate classification report
class_report = classification_report(y_test, y_pred)
print("Classification Report:")
print(class_report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       136
           1       0.58      0.24      0.34      6106
           2       0.58      0.89      0.70     13547
           3       0.97      0.59      0.73      2913
           4       0.00      0.00      0.00        41
           5       0.00      0.00      0.00        66
           6       0.94      0.97      0.96      9104
           7       1.00      0.00      0.01       434
           8       1.00      0.68      0.81      1659
           9       0.69      0.66      0.68      5445
          11       0.00      0.00      0.00        57
          12       0.00      0.00      0.00        14
          13       0.00      0.00      0.00         2
          14       0.00      0.00      0.00       150
          15       0.00      0.00      0.00        50
          16       0.00      0.00      0.00        50
          17       0.00      0.00      0.00        81
          18       0.76      0.77      0.76      5392
          19       0.83      0.61      0.70      3271
          23       0.00      0.00      0.00        15
          24       0.82      0.10      0.18       712
          25       0.00      0.00      0.00         2
          26       0.89      0.45      0.60      4699
          27       0.95      0.40      0.56       247
          28       0.00      0.00      0.00         2
          29       0.00      0.00      0.00       284
          31       0.96      0.34      0.50      2998
          32       1.00      0.01      0.01       394
          33       0.00      0.00      0.00       118
          34       0.71      0.98      0.82     15867
          35       0.97      0.74      0.84      2491

    accuracy                           0.73     76347
   macro avg       0.44      0.27      0.30     76347
weighted avg       0.75      0.73      0.70     76347
```

```
#test data classification report
y_pred = rf_model.predict(X_test)

# Generate classification report
class_report = classification_report(y_test, y_pred)
print("Classification Report:")
print(class_report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       136
           1       0.58      0.24      0.34      6106
           2       0.58      0.89      0.70     13547
           3       0.97      0.59      0.73      2913
           4       0.00      0.00      0.00        41
           5       0.00      0.00      0.00        66
           6       0.94      0.97      0.96      9104
           7       1.00      0.00      0.01       434
           8       1.00      0.68      0.81      1659
           9       0.69      0.66      0.68      5445
          11       0.00      0.00      0.00        57
          12       0.00      0.00      0.00        14
          13       0.00      0.00      0.00         2
          14       0.00      0.00      0.00       150
          15       0.00      0.00      0.00        50
          16       0.00      0.00      0.00        50
          17       0.00      0.00      0.00        81
          18       0.76      0.77      0.76      5392
          19       0.83      0.61      0.70      3271
          23       0.00      0.00      0.00        15
          24       0.82      0.10      0.18       712
          25       0.00      0.00      0.00         2
          26       0.89      0.45      0.60      4699
          27       0.95      0.40      0.56       247
          28       0.00      0.00      0.00         2
          29       0.00      0.00      0.00       284
          31       0.96      0.34      0.50      2998
          32       1.00      0.01      0.01       394
          33       0.00      0.00      0.00       118
          34       0.71      0.98      0.82     15867
          35       0.97      0.74      0.84      2491

    accuracy                           0.73     76347
   macro avg       0.44      0.27      0.30     76347
weighted avg       0.75      0.73      0.70     76347
```

Classification report after Tuning

```
#validation data classification report
y_pred = rf_model_tuned.predict(X_val)

# Generate classification report
class_report = classification_report(y_val, y_pred)
print("Classification Report:")
print(class_report)
```

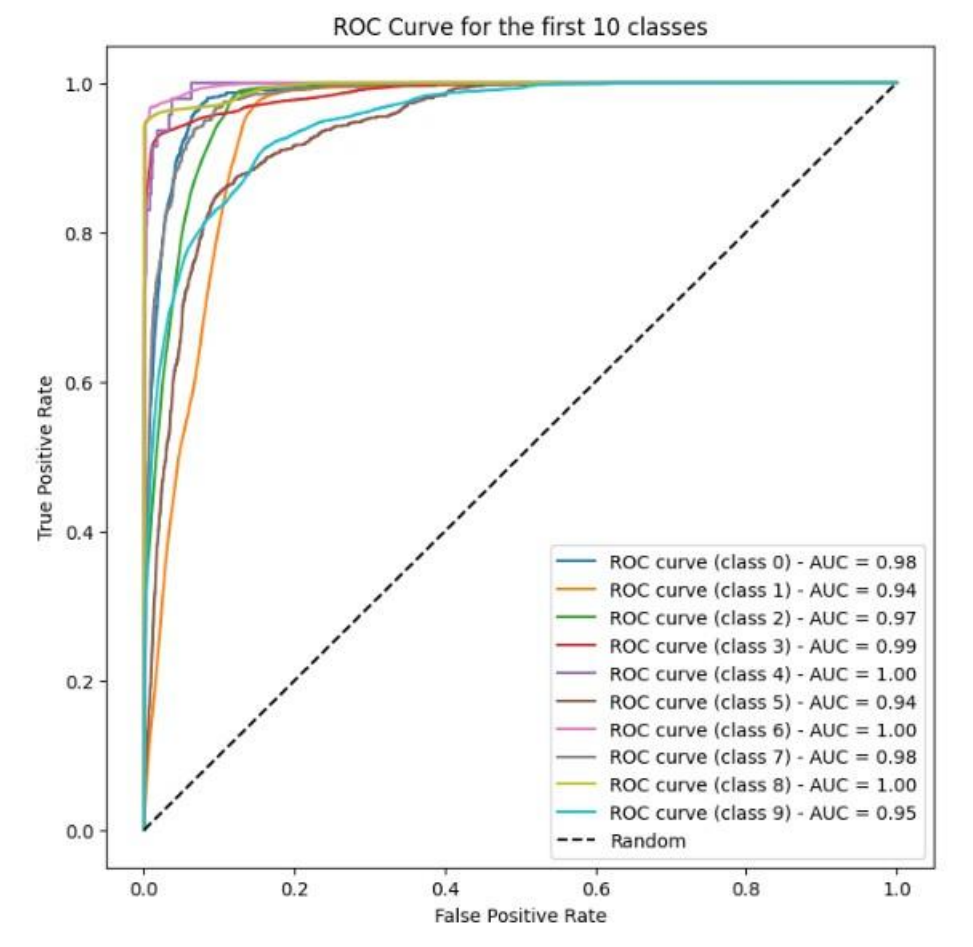
```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.65      0.75       307
           1       0.60      0.61      0.60     12807
           2       0.81      0.82      0.81     27876
           3       0.98      0.97      0.98      5944
           4       0.87      0.41      0.56        83
           5       0.55      0.18      0.27       125
           6       0.98      1.00      0.99     17997
           7       0.85      0.67      0.75       892
           8       1.00      0.90      0.94      3203
           9       0.95      0.97      0.96     11185
          11       0.86      0.76      0.80       111
          12       0.88      0.35      0.50        20
          13       0.00      0.00      0.00         7
          14       0.80      0.44      0.56       252
          15       0.90      0.31      0.46       114
          16       0.83      0.22      0.34        88
          17       0.90      0.61      0.73       135
          18       0.98      0.99      0.98     10899
          19       0.96      0.96      0.96      6491
          23       0.00      0.00      0.00        36
          24       0.90      0.79      0.84      1467
          25       0.00      0.00      0.00         1
          26       0.93      0.96      0.94      9596
          27       0.98      0.94      0.96       511
          28       0.00      0.00      0.00         3
          29       0.86      0.49      0.62       597
          30       0.00      0.00      0.00         1
          31       0.94      0.97      0.95      5930
          32       0.83      0.56      0.67       828
          33       0.72      0.18      0.29       229
          34       0.99      0.99      0.99     32284
          35       0.97      0.99      0.98      4988

    accuracy                           0.91    155007
   macro avg       0.74      0.58      0.63    155007
weighted avg       0.91      0.91      0.91    155007
```

```
#test data classification report
y_pred = rf_model_tuned.predict(X_test)

# Generate classification report
class_report = classification_report(y_test, y_pred)
print("Classification Report:")
print(class_report)
```

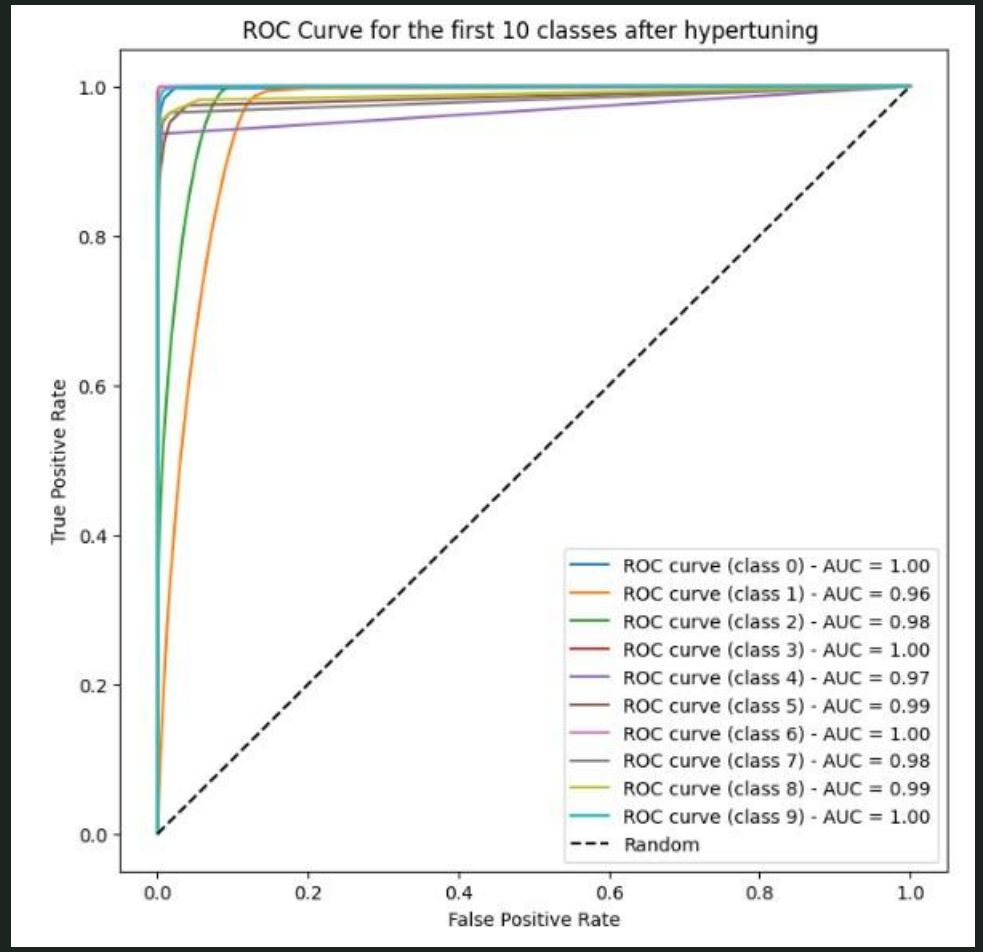
```
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.53      0.65       136
           1       0.59      0.61      0.60      6106
           2       0.81      0.82      0.81     13547
           3       0.98      0.98      0.98      2913
           4       0.88      0.68      0.77        41
           5       0.50      0.24      0.33        66
           6       0.98      1.00      0.99      9104
           7       0.85      0.70      0.77       434
           8       1.00      0.91      0.95      1659
           9       0.95      0.97      0.96      5445
          11       0.92      0.63      0.75        57
          12       1.00      0.43      0.60        14
          13       0.00      0.00      0.00         2
          14       0.89      0.34      0.49       150
          15       0.79      0.30      0.43        50
          16       0.67      0.04      0.08        50
          17       0.90      0.74      0.81        81
          18       0.98      0.99      0.98      5392
          19       0.97      0.96      0.96      3271
          23       0.00      0.00      0.00        15
          24       0.88      0.78      0.83       712
          25       0.00      0.00      0.00         2
          26       0.93      0.96      0.94      4699
          27       0.99      0.94      0.97       247
          28       0.00      0.00      0.00         2
          29       0.87      0.48      0.62       284
          31       0.94      0.97      0.96      2998
          32       0.88      0.56      0.69       394
          33       0.53      0.14      0.23       118
          34       0.99      0.99      0.99     15867
          35       0.96      0.99      0.98      2491

    accuracy                           0.91     76347
   macro avg       0.76      0.60      0.65     76347
weighted avg       0.91      0.91      0.91     76347
```

Model Evaluation-Random Forest

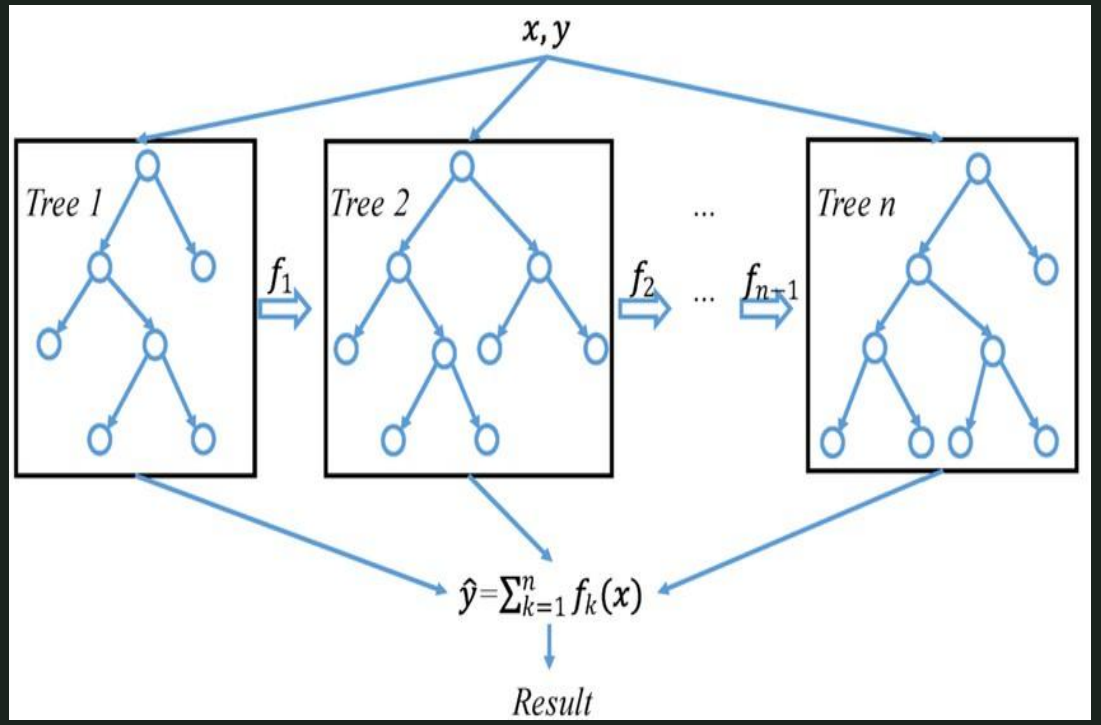
ROC curve before Tuning



ROC curve after Tuning



Model Proposal- XGBoost



- XGBoost is an ensemble learning method that combines multiple decision trees sequentially, iteratively minimizing errors.
- It employs gradient boosting, optimizing prediction accuracy by iteratively adding trees and correcting errors, using a gradient descent-based approach.
- XGBoost achieves high performance across various tasks while handling diverse data types efficiently.
- Study from research by Cheng Sheng et al and Haizheng Yu et al reveal the effectiveness of XGBoost in prediction highlights the potential of the models in tackling complex real-world problems with nonlinear relationships and multiple influencing factors.

Model Support -XGBoost


```
# Initialize XGBoost classifier
model = XGBClassifier(
    objective='multi:softmax',
    eval_metric='mlogloss',
    n_estimators=75,
    max_depth=6,
    learning_rate=0.1,
    n_jobs=-1
)
```

```
[7] #Train the model
    model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_val, y_val)], verbose=True)
```

```
[8] # Evaluate the model on the validation set
    y_val_pred = model.predict(X_val)
    accuracy = accuracy_score(y_val, y_val_pred)
    print(f'Validation Accuracy: {accuracy * 100:.2f}%')

    Validation Accuracy: 90.90%
```

```
[9] # Evaluate the model on the test set
    y_test_pred = model.predict(X_test)
    test_accuracy = accuracy_score(y_test, y_test_pred)
    print(f'Test Accuracy: {test_accuracy * 100:.2f}%')

    Test Accuracy: 90.90%
```

Baseline Model

- XGBoost baseline model performed with 75 trees ('n_estimators'), 6 levels deep ('max_depth'), 0.1 learning rate.
- Evaluated using 'mlogloss' ('eval_metrics') and 'multi:softmax' ('objective') on imbalanced data.

- Dataset split: 70% for training, 20% for validation, and 10% for testing.
- Training completed in roughly 5.86 seconds.
- Model achieved 0.91 accuracy and 0.22 loss on validation, maintaining 0.90 accuracy on the test set.

Model Building -XGBoost

Hypertuning the model

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

# Define the parameter grid to search
param_grid = {
    'n_estimators': [100, 125],  # Fixed value for number of estimators
    'learning_rate': [0.01] ,  # Fixed value for learning rate
}

# Initialize XGBoost classifier
model = XGBClassifier(objective='multi:softmax', eval_metric='mlogloss', n_jobs=-1)

# Initialize StratifiedKFold
stratified_cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

# Initialize GridSearchCV with StratifiedKFold
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
                           scoring='accuracy', cv=stratified_cv)
```

```
# Perform the grid search on the training data
grid_search.fit(X_train, y_train)

# Display the best parameters and corresponding accuracy
print(f'Best Parameters: {grid_search.best_params_}')

# Get the best model from grid search
best_model = grid_search.best_estimator_

# Make predictions on the validation set with the best model
y_val_pred = best_model.predict(X_val)
accuracy = accuracy_score(y_val, y_val_pred)
print(f'Validation Accuracy with Best Model: {accuracy * 100:.2f}%')

# Make predictions on the test set with the best model
y_test_pred = best_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f'Test Accuracy with Best Model: {test_accuracy * 100:.2f}%')

# Display the classification report for the test set
report = classification_report(y_test, y_test_pred)
print(f'{report}')
```

- Baseline model achieved 0.90 accuracy, prompting hyperparameter tuning.
- Employed GridSearchCV to optimize 'n_estimators' (100, 125) and 'learning_rate' (0.01) combinations.
- n_estimators with 125 and learning rate as 0.01 are selected as best parameter.
- Despite increased training time, both sets of hyperparameters yielded the same 90% accuracy on test data.
- Tuning n_estimators and learning_rate at 0.01 didn't enhance accuracy despite longer training duration.
- Highlighted the need for exploring other hyperparameters or techniques for potential improvement.

Model Evaluation- XGBoost

Classification report before Tuning

Classification Report:

	precision	recall	f1-score	support
0	0.82	0.58	0.68	1347
1	0.60	0.42	0.49	50928
2	0.80	0.90	0.85	142876
3	0.98	0.98	0.98	42108
4	0.93	0.36	0.52	114
5	0.68	0.61	0.65	2679
6	0.95	0.99	0.97	89189
7	0.70	0.54	0.61	726
8	0.97	0.93	0.95	21533
9	0.93	0.94	0.93	34714
11	0.90	0.70	0.79	1465
12	0.87	0.86	0.86	1244
13	0.00	0.00	0.00	12
14	0.82	0.46	0.59	1879
15	0.49	0.13	0.20	519
16	0.70	0.34	0.45	707
17	0.71	0.61	0.66	1415
18	0.97	0.93	0.95	38335
19	0.99	0.99	0.99	73904
20	0.00	0.00	0.00	1
21	0.00	0.00	0.00	14
22	0.00	0.00	0.00	1
23	1.00	0.01	0.02	86
24	0.79	0.63	0.71	5459
25	0.00	0.00	0.00	20
26	0.94	0.94	0.94	48766
27	0.95	0.79	0.86	6928
28	0.00	0.00	0.00	17
29	0.76	0.71	0.73	5148
30	0.00	0.00	0.00	1
31	0.94	0.97	0.95	29324
32	0.70	0.50	0.59	3219
33	0.00	0.00	0.00	517
34	0.98	0.99	0.99	164461
35	0.95	0.93	0.94	10865
accuracy			0.91	780521
macro avg	0.65	0.54	0.57	780521
weighted avg	0.90	0.91	0.90	780521

Classification report after Tuning

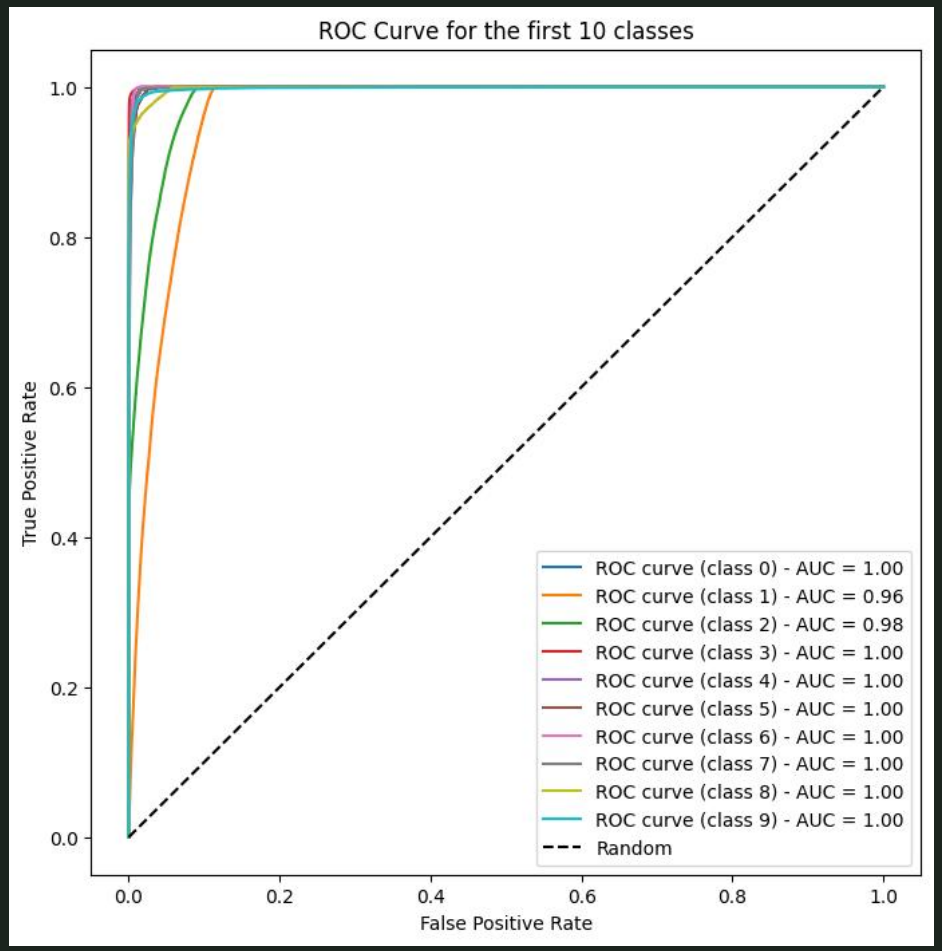
Best Parameters: {'learning_rate': 0.01, 'n_estimators': 125}
Validation Accuracy with Best Model: 90.07%
Test Accuracy with Best Model: 90.08%

	precision	recall	f1-score	support
0	0.85	0.49	0.62	2660
1	0.62	0.31	0.41	103989
2	0.76	0.93	0.84	288813
3	0.98	0.98	0.98	86312
4	0.95	0.37	0.53	232
5	0.74	0.78	0.76	5595
6	0.95	0.99	0.97	180491
7	0.74	0.54	0.62	1562
8	0.97	0.93	0.95	42994
9	0.88	0.93	0.91	70278
11	0.88	0.61	0.72	2924
12	0.91	0.52	0.66	2564
13	0.00	0.00	0.00	21
14	0.83	0.30	0.44	3691
15	0.48	0.14	0.22	961
16	0.76	0.02	0.04	1439
17	0.77	0.49	0.60	3057
18	0.97	0.91	0.94	78240
19	0.99	0.99	0.99	149855
20	0.00	0.00	0.00	7
21	0.00	0.00	0.00	45
23	0.00	0.00	0.00	175
24	0.84	0.61	0.71	11342
25	0.00	0.00	0.00	28
26	0.93	0.93	0.93	98436
27	0.93	0.72	0.81	14223
28	0.00	0.00	0.00	35
29	0.76	0.62	0.68	10516
30	0.00	0.00	0.00	5
31	0.92	0.97	0.94	59551
32	0.66	0.51	0.57	6350
33	0.00	0.00	0.00	1031
34	0.98	0.99	0.98	333728
35	0.95	0.92	0.94	22153

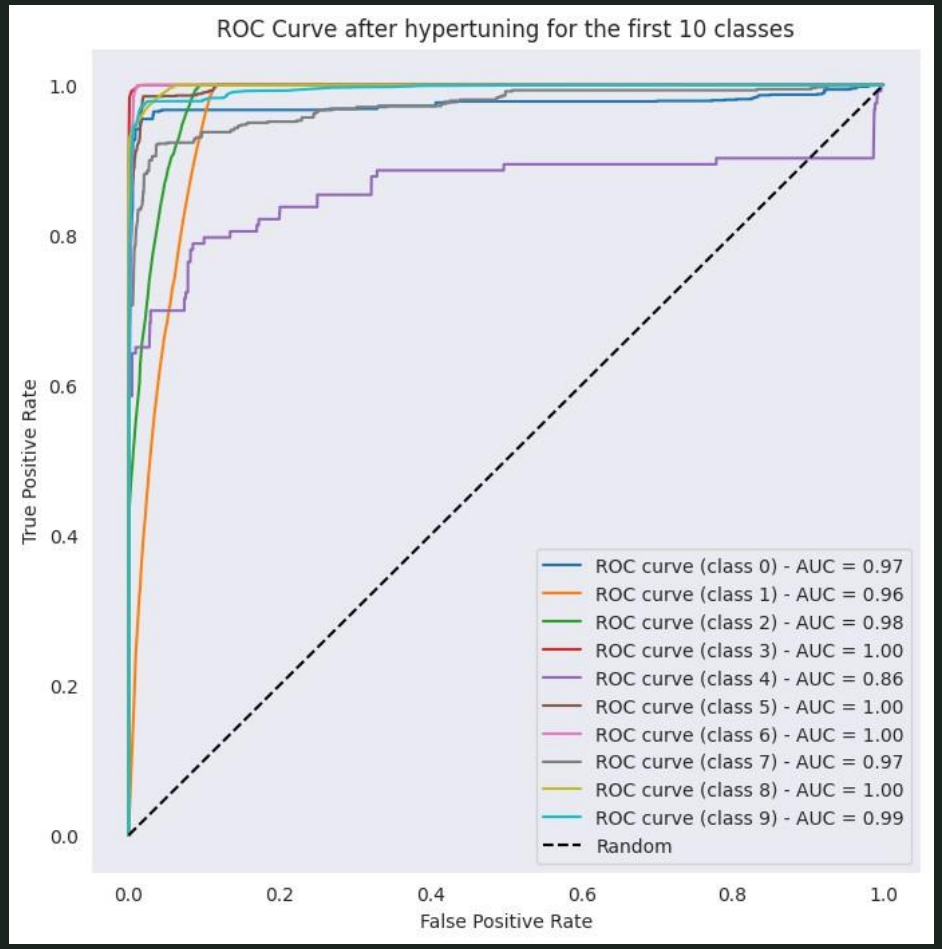
- Before tuning Accuracy is 91% and f1 score is 90%
- After tuning Accuracy is 90% and f1 score is 89%

Model Evaluation-XGBoost

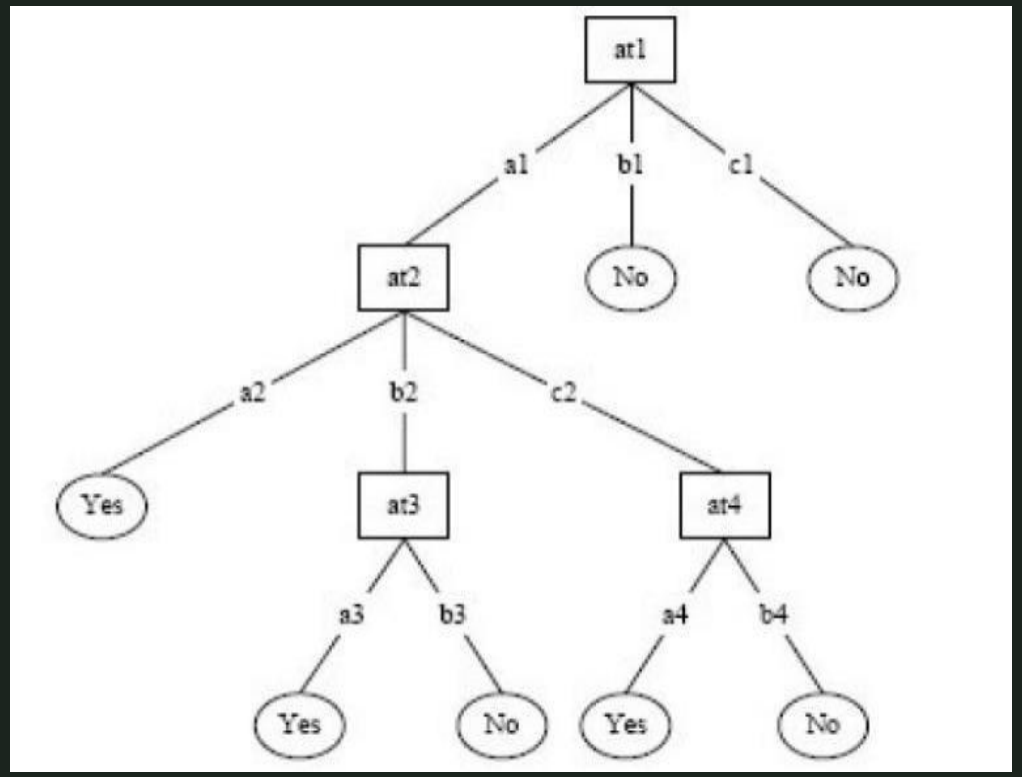
ROC curve before Tuning



ROC curve after Tuning



Model Proposal-Decision Tree



- Decision trees, a supervised learning algorithm and as a non-parametric classifier, provide a straightforward yet powerful approach with interpretability and automatic feature selection advantages.
- The process of prediction in Decision Trees entails moving from the initial node to a final node, where each node makes a decision based on a particular characteristic, creating a hierarchical series of decision guidelines.
- Insights from research by Rokach et al. and Ahishakiye et al. validate the practicality, simplicity, and transparency of decision trees in real-life situations, as well as enhancements to the decision tree algorithm.

Model Support-Decision Tree

Baseline Model

```
start_time = time.time()
classifier = DecisionTreeClassifier()

classifier.fit(X_train, y_train)

# Make predictions on the validation set
y_val_pred = classifier.predict(X_val)
accuracy = accuracy_score(y_val, y_val_pred)
print(f'Validation Accuracy: {accuracy * 100:.2f}%')

# Now make predictions on the test set
y_test_pred = classifier.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')

end_time = time.time()
print(f"Training time: {end_time - start_time} seconds")
```

```
Validation Accuracy: 92.72%
Test Accuracy: 92.78%
Training time: 62.69876670837402 seconds
```

Baseline Model:

Hyperparameters:

- Criterion: 'gini'
- Splitter strategy: 'best'
- Max depth: None (unlimited growth) -
- Min samples split: 2
- Min samples leaf: 1
- Min weighted fraction leaf: 0.0
- Max features: None (consider all features)
- Random state: None (allows non-deterministic behavior)

- Performance:
  - Accuracy: 92.77%
- Training time: 62.69 seconds

Model Building-Decision Tree

Do GridSearch to find best hyperparameters for decision tree classifier

```
dtree = DecisionTreeClassifier()

param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2']
}

# Use GridSearchCV to do grid search
grid_search = GridSearchCV(dtree, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best Hyperparameters:", grid_search.best_params_)

best_model = grid_search.best_estimator_

accuracy = best_model.score(X_test, y_test)
print("Accuracy on Test Set:", accuracy)
```

```
Best Hyperparameters: {'criterion': 'entropy', 'max_depth': 15, 'max_features': None, 'min_samples_leaf': 4, 'min_samples_sp
lit': 5}
Accuracy on Test Set: 0.9427904015726308
```

Hyperparameter tuned model

```
start_time = time.time()
best_clssifier = DecisionTreeClassifier(criterion='entropy',
                                        max_depth=15,
                                        max_features=None,
                                        min_samples_leaf=4,
                                        min_samples_split=5)

best_clssifier.fit(X_train, y_train)

# Make predictions on the validation set
y_val_pred = best_clssifier.predict(X_val)
accuracy = accuracy_score(y_val, y_val_pred)
print(f'Validation Accuracy: {accuracy * 100:.2f}%')

# Now make predictions on the test set
y_test_pred = best_clssifier.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')

end_time = time.time()
print(f"Training time: {end_time - start_time} seconds")
```

```
Validation Accuracy: 94.27%
Test Accuracy: 94.28%
Training time: 65.13338994979858 seconds
```

Hyperparameter Tuned Model

- GridSearchCV: Utilized GridSearchCV for hyperparameter optimization
- Duration: Approximately 8 hours for an extensive grid search
- Best Hyperparameters: Criterion: 'entropy', Max depth: 15 ,Max features: None ,Min samples leaf: 4 ,Min samples split: 5

Model Evaluation-Decision Tree

Classification report before Tuning

Classification Report:

	precision	recall	f1-score	support
0	0.95	0.94	0.94	2683
1	0.49	0.49	0.49	104630
2	0.82	0.81	0.82	290171
3	1.00	1.00	1.00	86378
4	0.94	0.92	0.93	238
5	0.88	0.89	0.89	5696
6	0.98	0.98	0.98	181609
7	0.80	0.79	0.79	1579
8	0.93	0.94	0.93	43429
9	1.00	1.00	1.00	70841
11	0.99	0.99	0.99	2915
12	1.00	1.00	1.00	2583
13	1.00	1.00	1.00	25
14	1.00	1.00	1.00	3696
15	1.00	1.00	1.00	935
16	0.94	0.94	0.94	1462
17	1.00	1.00	1.00	3001
18	1.00	1.00	1.00	78306
19	1.00	1.00	1.00	150383
20	1.00	1.00	1.00	7
21	0.98	0.93	0.95	45
23	0.99	1.00	1.00	171
24	1.00	1.00	1.00	11375
25	1.00	1.00	1.00	29
26	1.00	1.00	1.00	98843
27	1.00	1.00	1.00	14240
28	1.00	1.00	1.00	38
29	1.00	1.00	1.00	10502
30	1.00	0.75	0.86	4
31	1.00	1.00	1.00	59699
32	0.99	0.99	0.99	6429
33	0.17	0.19	0.18	1048
34	1.00	1.00	1.00	335975
35	1.00	1.00	1.00	22254
accuracy			0.93	1591219
macro avg	0.94	0.93	0.93	1591219
weighted avg	0.93	0.93	0.93	1591219

Accuracy Score: 92.77%

Classification report after Tuning

Classification Report:

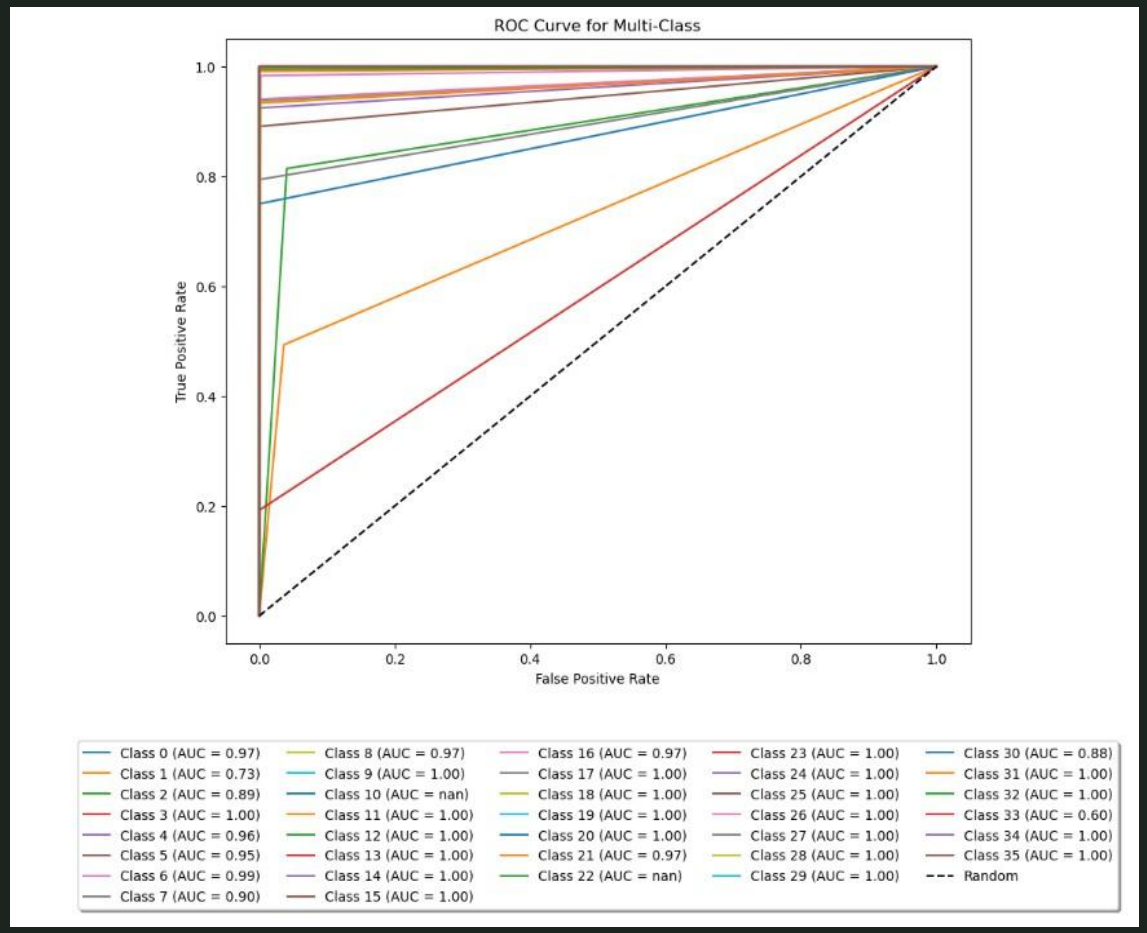
	precision	recall	f1-score	support
0	0.94	0.97	0.96	2683
1	0.62	0.46	0.52	104630
2	0.82	0.90	0.86	290171
3	1.00	1.00	1.00	86378
4	0.97	0.96	0.96	238
5	0.95	0.92	0.93	5696
6	0.98	1.00	0.99	181609
7	0.96	0.74	0.84	1579
8	1.00	0.93	0.96	43429
9	1.00	1.00	1.00	70841
11	0.99	0.99	0.99	2915
12	1.00	1.00	1.00	2583
13	1.00	1.00	1.00	25
14	1.00	1.00	1.00	3696
15	1.00	1.00	1.00	935
16	1.00	0.94	0.97	1462
17	1.00	1.00	1.00	3001
18	1.00	1.00	1.00	78306
19	1.00	1.00	1.00	150383
20	1.00	1.00	1.00	7
21	0.98	0.96	0.97	45
23	0.99	1.00	1.00	171
24	1.00	1.00	1.00	11375
25	1.00	1.00	1.00	29
26	1.00	1.00	1.00	98843
27	1.00	1.00	1.00	14240
28	1.00	1.00	1.00	38
29	1.00	1.00	1.00	10502
30	1.00	0.75	0.86	4
31	1.00	1.00	1.00	59699
32	1.00	0.99	1.00	6429
33	1.00	0.18	0.31	1048
34	1.00	1.00	1.00	335975
35	1.00	1.00	1.00	22254
accuracy			0.94	1591219
macro avg	0.98	0.93	0.94	1591219
weighted avg	0.94	0.94	0.94	1591219

Accuracy Score: 94.28%

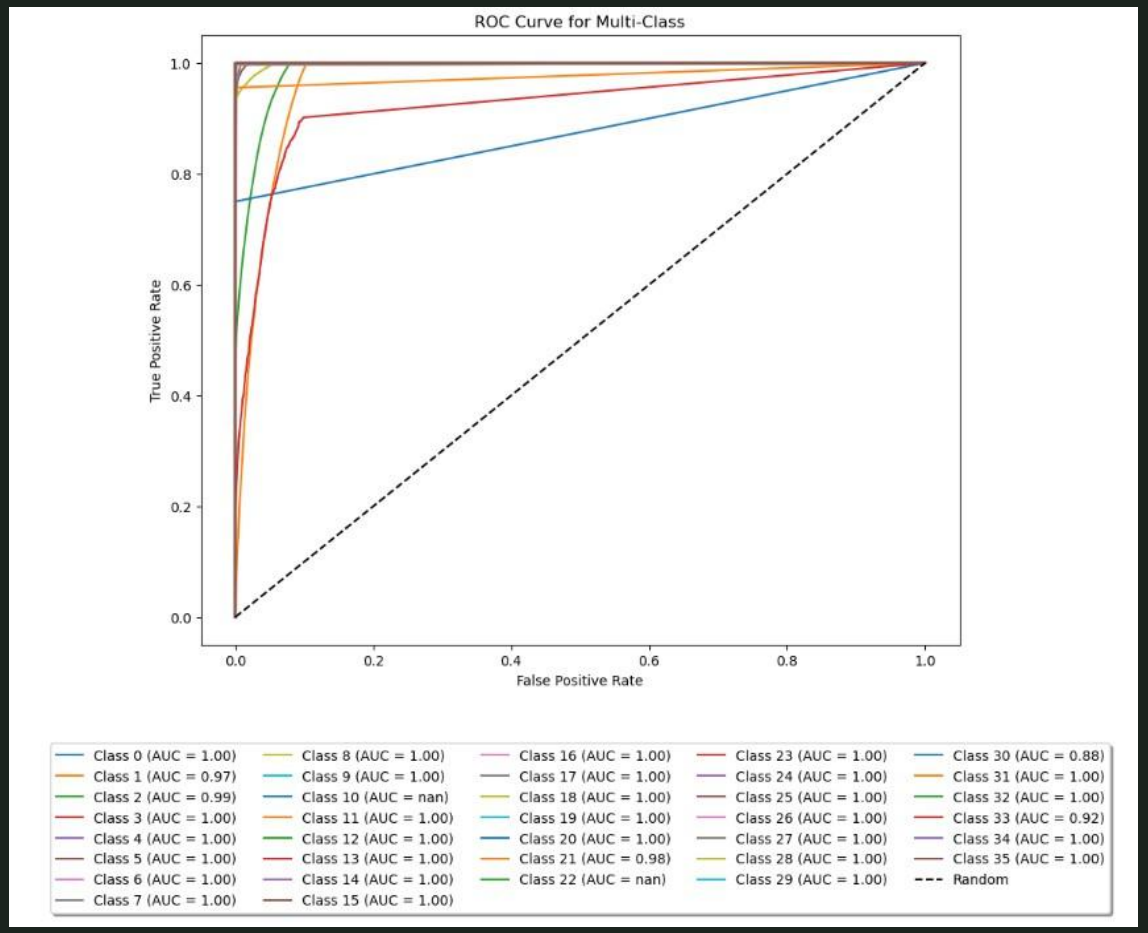
- Accuracy: 93%, f1 score: 93%, before tuning.
- Accuracy: 94% , f1 score:94%, after tuning.
- Training time: 65.13 seconds
- Generalization Assessment: Validated on an independent test set
- Insights into effectiveness and reliability of the trained decision tree

Model Evaluation-Decision Tree

ROC curve before Tuning



ROC curve after Tuning



Model Comparison

Model	Advantages	Disadvantages	Perfromance
Random Forest	• Ensemble method contributes to model robustness • Different trees can be built simultaneously • Provides high accuracy • Works with both categorical and numerical variables	• Less Interpretable than DT • Computationally expensive on large dataset • Biased toward the dominant class	• After Hyper tuning the model, the performance of the model progressed and the accuracy increased from 73% to 91% and f1 score from 70% to 91%. • Running time decreased from 7mins to 2mins.
Decision Tree（C4.5）	• Provides human-readable prediction • Adapt to both categorical and numerical attributes • Can handle missing values • Scalable to large datasets	• Sensitive to noisy data and outliers • Designed for classification, limit support for regression • Easily lead to overfitting	• Accuracy: 93%, f1 score: 93%, before tuning. • Accuracy: 94% , f1 score:94%, after tuning. • Training time: 65.13 seconds
XGBoost	• High accuracy and performance on various datasets • Faster than traditional algorithm using parallel processing • Includes regularization terms to reduce overfitting • Handling data imbalance and missing values	• Hard to find the optimal set of hyperparameters • Computationally expensive, especially for large datasets • Require large amount of memory • Sensitive to outliers and noises	• Increased training time • Before tuning Accuracy is 91% and f1 score is 90% • After tuning Accuracy is 90% and f1 score is 89%
ANN	• Achieves high accuracy by learning autonomously from data • Ability to capture the complex patterns of data • Can handle incomplete or noisy data • Generalizes well to unseen data	• Require large amount of data for training • Computationally intensive for deep neural networks • Model performance affected by data preprocessing • Black box nature makes it hard to interpret	• After Hyper tuning the model, the accuracy increased from 81% to 92% and f1 score increased from 79% to 91%. • The running time decreases from 130s to 90s.

Our best model

- The results show that the decision tree model is the best model for this task.
- It has the highest accuracy, precision, recall, and F1-score.
- The random forest model is the second best model, followed by the XGBoost and ANN models.
- We selected Decision Tree instead of Random Forest because DTs is more interpretable and easier to visualize than Random Forest also Random Forest is computationally intensive and is taking more time for training when compared to decision tree.
- The decision tree model was also saved using the joblib library, so that it can be used to make predictions on new data in the future.

```
# Save the trained model
model_filename = 'dt_hyperparameter_tuning_model.joblib'
joblib.dump(best_clssifier, model_filename)
print(f'Model saved to {model_filename}')
```

```
Model saved to dt_hyperparameter_tuning_model.joblib
```

Summary

The project's goal is to use machine learning and historical crime data to proactively identify high-risk areas for criminal behavior. Traditional methods of reactive crime prevention have limitations in adapting to changing threats, so the project aims to optimize law enforcement resources and increase public safety using predictive models such as ANN, Decision Tree, XGBoost, and Random Forest. By analyzing crime patterns and demographics, the initiative seeks to foster safer communities through early intervention strategies while minimizing societal and economic impacts.

Future Scope

- Integrating diverse dataset from all over the world and not limited to the city of Chicago.
- Big data integration such as including public records and sensor data.
- Implementing Geographic information system (GIS) to visualize crime pattern.

References

https://www.researchgate.net/publication/340969457_Crime_Prediction_Using_K-Nearest_Neighboring_Algorithm

https://www.ijcit.com/archives/volume6/issue3/Paper060308.pdf

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8529125/

https://www.irjet.net/archives/V5/i9/IRJET-V5I9192.pdf

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8081790/pdf/42492_2021_Article_75.pdf

https://www.irjmets.com/uploadedfiles/paper/issue_6_june_2022/27431/final/fin_irjmets1657108227.pdf

https://ieeexplore-ieee-org.libaccess.sjlibrary.org/stamp/stamp.jsp?tp=&arnumber=8821168

https://ieeexplore.ieee.org/abstract/document/8113405

https://www.mdpi.com/2079-3197/9/2/24

https://www-sciencedirect-com.libaccess.sjlibrary.org/science/article/pii/S0301420718306901?via%3Dihub

https://core.ac.uk/reader/560325700

https://link.springer.com/content/pdf/10.1023/A:1010933404324.pdf

https://ieeexplore-ieee-org.libaccess.sjlibrary.org/stamp/stamp.jsp?tp=&arnumber=9985004

https://ieeexplore-ieee-org.libaccess.sjlibrary.org/stamp/stamp.jsp?tp=&arnumber=8734193

https://arxiv.org/pdf/1603.02754.pdf

https://www.sciencedirect.com/science/article/abs/pii/S0927025619307712?via%3Dihub

Thank you