CSE 587 B - DIC Project Report

Predictive Policing: Data-Driven Approach to Crime Analysis

Problem Statement:

Understanding the importance of analyzing crime data involves acknowledging the multifaceted challenges it can address. It is the initial step towards realizing the full potential of crime data analysis:

1. **Prioritizing Public Safety:** A paramount concern is safeguarding the well-being and security of individuals and communities. Analyzing crime data aids in identifying areas with elevated crime rates, thereby enabling law enforcement to allocate their resources with precision, thus making our communities safer.
2. **Efficient Resource Allocation:** Law enforcement agencies operate under resource constraints. Crime data analysis serves as a guiding compass for allocating these limited resources to areas where they are most urgently needed. This results in reduced response times and increased overall operational efficiency.
3. **Rooting Out Crime Causes:** Delving into historical crime patterns and their contributing factors, crime data analysis unravels invaluable insights into the origins of criminal activities. Unearthing these underlying causes forms the bedrock for crafting effective crime prevention strategies.

4. **Adapting to Emerging Trends:** The world of crime constantly evolves, with new trends and tactics emerging regularly. The analytical examination of crime data fosters a deeper understanding of criminal behaviors and modus operandi, empowering law enforcement to respond adeptly.
5. **Optimizing Resource Efficiency:** The efficiency of resource utilization is a universal challenge. Crime data analysis helps identify and rectify inefficiencies and overlaps in resource allocation, ensuring that every resource is put to its most effective use.

Contents

Phase 1

Introduction

Crime is undoubtedly a significant concern within our society, and it casts a long shadow over the safety and well-being of individuals and entire communities. Going further into the intricate dynamics of crime, from its unique patterns to the underlying factors that give rise to it, is of paramount importance. These insights serve as the foundation for creating effective strategies to prevent and intervene in criminal activities.

The period from 2020 to 2023 has brought some significant changes in our world, driven by various social, economic, and technological forces. These shifts have inevitably left their mark on the landscape of crime. As we navigate these uncharted waters, grasping the evolving nature of criminal activities becomes increasingly vital.

In this report, our primary objective is to offer you a comprehensive analysis of the prevalent problems in the public application domains. Specifically, we aim to research and identify pertinent data sets, preferably structured data, that can help address these problems. We will not only pinpoint the issues but also work towards collecting the relevant data sets, ensuring they are meticulously cleaned and provisioned for further explorations and analytics.

By the end, we hope to provide a clear and data-driven perspective on the challenges we face, enabling informed decision-making and the development of effective solutions to the problems that lie within the selected domain

Defining the Challenge

Understanding the importance of analyzing crime data involves acknowledging the multifaceted challenges it can address. It is the initial step towards realizing the full potential of crime data analysis:

1. **Prioritizing Public Safety:** A paramount concern is safeguarding the well-being and security of individuals and communities. Analyzing crime data aids in identifying areas with elevated crime rates, thereby enabling law enforcement to allocate their resources with precision, thus making our communities safer.
2. **Efficient Resource Allocation:** Law enforcement agencies operate under resource constraints. Crime data analysis serves as a guiding compass for allocating these limited resources to areas where they are most urgently needed. This results in reduced response times and heightened overall operational efficiency.

3. **Rooting Out Crime Causes:** Delving into historical crime patterns and their contributing factors, crime data analysis unravels invaluable insights into the origins of criminal activities. Unearthing these underlying causes forms the bedrock for crafting effective crime prevention strategies.
4. **Adapting to Emerging Trends:** The world of crime constantly evolves, with new trends and tactics emerging regularly. The analytical examination of crime data fosters a deeper understanding of criminal behaviors and modus operandi, empowering law enforcement to respond adeptly.
5. **Optimizing Resource Efficiency:** The efficiency of resource utilization is a universal challenge. Crime data analysis helps identify and rectify inefficiencies and overlaps in resource allocation, ensuring that every resource is put to its most effective use.

Potential Contribution of the Project

This project holds immense potential to contribute to the problem domain in several ways.

1. **Pattern Recognition:** To systematically analyze crime data and identify recurring patterns and trends in criminal activities, including their locations, timing, and characteristics.
2. **Predictive Policing:** To develop advanced predictive models that leverage historical crime data for the anticipation of potential crime hotspots and the strategic allocation of resources to proactively deter criminal activities.
3. **Resource Optimization:** To optimize the allocation of finite law enforcement resources by focusing efforts on high-risk areas, thus maximizing their impact on crime prevention.
4. **Community Safety:** To contribute significantly to the overall safety and security of our communities by proactively addressing potential issues and creating a substantially safer environment for residents.
5. **Strategic Planning:** To empower law enforcement agencies with data-driven insights for effective strategic planning, enabling them to adapt swiftly to evolving crime patterns and make well-informed decisions.

The contribution of this project is crucial, as it directly impacts community safety, reduces crime rates, and fosters well-being. By leveraging comprehensive crime data analysis, it aims to play a pivotal role in achieving these vital societal goals.

Dataset Overview

🔗 **Source:** City of Los Angeles Open Data

Title: Crime Data from 2020 to Present

Dimensions: 811663 Rows, 28 Columns

Description:

This dataset provides comprehensive insights into incidents of crime within the City of Los Angeles from 2020 to the present day. It serves as a valuable resource for understanding and analyzing crime trends, patterns, and occurrences within this urban environment. The dataset's origin is rooted in the transcription of original crime reports, which are typically documented on paper. It's important to note that, due to the manual transcription process, there may be some inaccuracies present within the data.

Key Features and Information:

1. Temporal Scope: The dataset covers a substantial time frame, dating back to 2020 and extending into the present day. This allows for the examination of long-term crime trends and patterns.

2. Data Source: The data is sourced from original crime reports, emphasizing its real-world relevance. These reports are generated by the Los Angeles Police Department (LAPD) and are considered a reliable and authoritative source of information.

3. Privacy Considerations: To safeguard the privacy of individuals and maintain data confidentiality, address fields within the dataset are provided only up to the nearest hundred block. This ensures that specific addresses are not disclosed.

4. Data Accuracy: While the dataset strives for accuracy, it's important to acknowledge that potential inaccuracies may arise during the transcription process. Users should exercise caution and consider the dataset's limitations when conducting analyses or drawing conclusions.

The Crime Data from 2020 to Present is a valuable resource that provides an in-depth view of crime incidents within Los Angeles. While it may contain minor inaccuracies due to the data transcription process, it remains an essential tool for understanding and addressing crime-related challenges in the city. This dataset is a testament to the LAPD's commitment to transparency and the provision of vital data for public and professional use.

Data Cleaning/Processing

Let's now take a closer look at the dataset's contents, focusing on the extensive cleaning and processing stages. This intensive process is intended to ensure that the data is thoroughly processed and improved, making it ready for subsequent analysis and predictive modeling.

1. Understand the data

This Python code imports essential libraries (Pandas, Numpy, Matplotlib, and Seaborn), and then it reads a CSV file named "Crime_Data_from_2020_to_Present.csv" into a Pandas DataFrame called "crimeDF." This DataFrame likely contains crime data from 2020 to the present, which can be further analyzed and visualized using these libraries.

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

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from wordcloud import WordCloud

crimeDF = pd.read_csv("Crime_Data_from_2020_to_Present.csv")
crimeDF
```

The initial phase of data cleaning and processing involves gaining insights into the dataset. This is achieved through the following actions:

- Utilizing `crimeDF.info()` to obtain an overview of the data's characteristics.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 811663 entries, 0 to 811662
Data columns (total 28 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   DR_NO           811663 non-null  int64
 1   Date Rptd       811663 non-null  object
 2   DATE OCC        811663 non-null  object
 3   TIME OCC        811663 non-null  int64
 4   AREA            811663 non-null  int64
 5   AREA NAME       811663 non-null  object
 6   Rpt Dist No     811663 non-null  int64
 7   Part 1-2        811663 non-null  int64
 8   Crm Cd          811663 non-null  int64
 9   Crm Cd Desc     811663 non-null  object
 10  Mocodes         699639 non-null  object
 11  Vict Age        811663 non-null  int64
 12  Vict Sex        705139 non-null  object
 13  Vict Descent    705131 non-null  object
 14  Premis Cd       811654 non-null  float64
 15  Premis Desc     811184 non-null  object
 16  Weapon Used Cd  282783 non-null  float64
 17  Weapon Desc     282783 non-null  object
 18  Status          811663 non-null  object
 19  Status Desc     811663 non-null  object
...
 26  LAT             811663 non-null  float64
 27  LON             811663 non-null  float64
dtypes: float64(8), int64(7), object(13)
memory usage: 173.4+ MB
```

- Employing `crimeDF.describe()` to extract summary stats.

DR_NO	TIME OCC	AREA	Rpt Dist No	Part 1-2	Crm Cd	Vict Age	Premis Cd
count	8.116630e+05	811663.000000	811663.000000	811663.000000	811663.000000	811663.000000	811663.00000
mean	2.159652e+08	1335.588916	10.712678	1117.690694	1.413998	500.745704	29.828150
std	1.077479e+07	654.136069	6.092110	609.205985	0.492548	207.814959	21.769578
min	8.170000e+02	1.000000	1.000000	101.000000	1.000000	110.000000	-3.000000
25%	2.101207e+08	900.000000	6.000000	622.000000	1.000000	331.000000	8.000000
50%	2.201116e+08	1415.000000	11.000000	1142.000000	1.000000	442.000000	31.000000
75%	2.219102e+08	1900.000000	16.000000	1617.000000	2.000000	626.000000	45.000000
max	2.399165e+08	2359.000000	21.000000	2199.000000	2.000000	956.000000	120.000000

- Utilizing `crimeDF.isna().sum()` - count the missing values.

```
DR_NO                 0
Date Rptd             0
DATE OCC              0
TIME OCC              0
AREA                  0
AREA NAME             0
Rpt Dist No           0
Part 1-2              0
Crm Cd                0
Crm Cd Desc           0
Mocodes          112024
Vict Age              0
Vict Sex         106524
Vict Descent     106532
...
```

- `crimeDF.nunique()` to determine the count of unique values in each column.

```
DR_NO             811663
Date Rptd           1371
DATE OCC            1371
TIME OCC            1439
AREA                  21
AREA NAME             21
Rpt Dist No         1204
Part 1-2               2
Crm Cd               138
Crm Cd Desc          138
Mocodes           271218
Vict Age             103
Vict Sex               5
Vict Descent          20
Premis Cd            313
Premis Desc          306
Weapon Used Cd        79
Weapon Desc           79
Status                 6
Status Desc            6
Crm Cd 1             140
Crm Cd 2             122
Crm Cd 3              37
Crm Cd 4               6
LOCATION           63575
Cross Street        9649
LAT                 5399
LON                 4970
dtype: int64
```

- Using `crimeDF.shape` to get the dimensions of the dataset.

```
(811663, 28)
```

2. Dropping unnecessary columns

- We can eliminate the '**DR_NO**' column due to its large number of unique values, which can overly expand the data size. Similarly, '**Mocodes**' is to be dropped because it contains over 50% null values.

```
crimeDF.drop(columns=['DR_NO','Mocodes'],inplace=True)
```

Furthermore, we are also discarding several other columns, namely "**Weapon Used Cd,**" "**Weapon Desc**," "**Crm Cd 2**," "**Crm Cd 3**," "**Crm Cd 4**," and "**Cross Street**." This streamlines the dataset for more focused analysis.

```
columns_to_drop = ["Weapon Used Cd", "Weapon Desc",  "Crm Cd 2", "Crm Cd 3", "Crm Cd 4", "Cross Street"]
crimeDF.drop(columns=columns_to_drop, inplace=True)
```

3. Handling Missing Values

To address missing values in the dataset, we begin by examining the unique values in each column using a loop. This provides an understanding of the data distribution for each feature.

```
for col in crimeDF.columns:
    uniValues =crimeDF[col].unique()
    print(f"Unique values in '{col}':")
    print(uniValues)
    print("\n")
```

Output:

```
Unique values in 'Date Rptd':
['01/08/2020 12:00:00 AM' '01/02/2020 12:00:00 AM'
 '04/14/2020 12:00:00 AM' ... '09/03/2023 12:00:00 AM'
 '07/04/2023 12:00:00 AM' '07/09/2023 12:00:00 AM']

Unique values in 'DATE OCC':
['01/08/2020 12:00:00 AM' '01/01/2020 12:00:00 AM'
 '02/13/2020 12:00:00 AM' ... '09/28/2023 12:00:00 AM'
 '09/21/2023 12:00:00 AM' '03/16/2023 12:00:00 AM']

Unique values in 'TIME OCC':
[2230  330 1200 ...  559  818  841]

Unique values in 'AREA':
[ 3  1 15 19 17 11  5  9 10  8  7 14  2 12  4 18  6 13 16 20 21]

Unique values in 'AREA NAME':
['Southwest' 'Central' 'N Hollywood' 'Mission' 'Devonshire' 'Northeast'
 'Harbor' 'Van Nuys' 'West Valley' 'West LA' 'Wilshire' 'Pacific'
 'Rampart' '77th Street' 'Hollenbeck' 'Southeast' 'Hollywood' 'Newton'
 'Foothill' 'Olympic' 'Topanga']
...
Unique values in 'LON':
[-118.2978 -118.2545 -118.2474 ... -118.6606 -118.6329 -118.6506]

Output is truncated
```

We then proceed to count the number of missing values in the dataset and verify its overall dimensions.

```
crimeDF.isna().sum()

#Output
Date Rptd            0
DATE OCC             0
```

```
TIME OCC            0
AREA                0
AREA NAME           0
Rpt Dist No         0
Part 1-2            0
Crm Cd              0
Crm Cd Desc         0
Vict Age            0
Vict Sex       106524
Vict Descent   106532
Premis Cd           9
Premis Desc       479
Status              0
Status Desc         0
Crm Cd 1           10
LOCATION            0
LAT                 0
LON                 0
dtype: int64
```

For handling missing data, we adopt different strategies based on the data type of the columns. For numeric columns like "Premis Cd" and "Crm Cd 1," we fill the missing values with the mean of each respective column.

```
numericalcolumns = ["Premis Cd", "Crm Cd 1"]
crimeDF[numericalcolumns] = crimeDF[numericalcolumns].fillna(crimeDF[numericalcolumns].mean())
```

On the other hand, for object-type columns such as "Vict Sex," "Vict Descent," and "Premis Desc," we fill the missing values with the mode (most frequent value) of each column.

```
objectcolumns = ["Vict Sex", "Vict Descent", "Premis Desc"]
crimeDF[objectcolumns] = crimeDF[objectcolumns].fillna(crimeDF[objectcolumns].mode().iloc[0])
```

Finally, we assess the number of unique values in the dataset after addressing missing data, completing the data preparation process.

```
crimeDF.nunique()

#Output
Date Rptd        1371
DATE OCC         1371
TIME OCC         1439
AREA               21
AREA NAME          21
Rpt Dist No      1204
Part 1-2            2
Crm Cd            138
Crm Cd Desc       138
Vict Age          103
Vict Sex            5
Vict Descent       20
Premis Cd         314
Premis Desc       306
Status              6
Status Desc         6
Crm Cd 1          141
LOCATION        63575
LAT              5399
LON              4970
dtype: int64
```

4. Handling Duplicates

We initiate the handling of duplicate records by identifying columns with object data types in the dataset. These columns are subsequently converted to lowercase for uniformity.

```
object_columns = crimeDF.select_dtypes(include=['object']).columns

for col in object_columns:
    crimeDF[col] = crimeDF[col].str.lower()
```

We then calculate the count of duplicated entries within the dataset and record this count.

```
duplicate_count = crimeDF.duplicated().sum()
duplicate_count

# Output
3341
```

Following this assessment, we remove the duplicate records from the dataset, ensuring data cleanliness and reducing redundancy.

```
crimeDF.drop_duplicates(inplace=True)
crimeDF
```

Upon completion of this process, we reevaluate the dataset for any remaining duplicates, ultimately providing a dataset with no redundant entries.

```
crimeDF.duplicated().sum()

#Output
0
```

5. Standardizing Column Names

To ensure consistency and simplicity in column names, we employ a function named `standardize_column_names` to standardize the column names in the dataset. The steps involved are as follows:

- All column names are converted to lowercase for uniformity.
- Spaces within column names are replaced with underscores to eliminate any inconsistencies.
- Any special characters that do not adhere to standard naming conventions (letters, numbers, underscores) are removed from the column names.

This standardized naming process enhances the dataset's readability and usability.

```
def standardize_column_names(df):
    df.columns = df.columns.str.lower()  # Convert to lowercase
    df.columns = df.columns.str.replace(' ', '_')  # Replace spaces with underscores
    df.columns = df.columns.str.replace(r'[^a-zA-Z0-9_]', '')  # Remove special characters

standardize_column_names(crimeDF)
crimeDF
```

Furthermore, we define a set of new column names for specific columns, mapping them to more descriptive labels. These updated column names facilitate better understanding of the dataset.

```
new_column_names = {
    'date_rptd': 'Date_Reported',
    'date_occ': 'Date_Occurred',
    'time_occ': 'Time_Occurred',
    'area': 'Area',
    'area_name': 'Area_Name',
    'rpt_dist_no': 'Reporting_District_Number',
    'part_12': 'Part_1_or_2',
    'crm_cd': 'Crime_Code',
    'crm_cd_desc': 'Crime_Code_Description',
    'vict_age': 'Victim_Age',
```

```
    'vict_sex': 'Victim_Sex',
    'vict_descent': 'Victim_Descent',
    'premis_cd': 'Premise_Code',
    'premis_desc': 'Premise_Description',
    'status': 'Status',
    'status_desc': 'Status_Description',
    'crm_cd_1': 'Crime_Code_1',
    'location': 'Location',
    'lat': 'Latitude',
    'lon': 'Longitude'
}

crimeDF.rename(columns=new_column_names, inplace=True)
```

Upon applying these modifications, the dataset reflects standardized and more informative column names for improved data analysis and interpretation.

New column names: [Date_Reported, Date_Occurred, Time_Occurred, Area, Area_Name, Reporting_District_Number, Part_1_or_2, Crime_Code, Crime_Code_Description, Victim_Age, Victim_Sex, Victim_Descent, Premise_Code, Premise_Description, Status, Status_Description, Crime_Code_1, Location, Latitude, Longitude]

6. Data Type Conversion

Convert the "**Date_Reported**" and "**Date_Occurred**" columns to datetime data types for proper date handling.

```
crimeDF['Date_Reported'] = pd.to_datetime(crimeDF['Date_Reported'])
crimeDF['Date_Occurred'] = pd.to_datetime(crimeDF['Date_Occurred'])
```

To achieve this, we converted the 'Date_Reported' and 'Date_Occurred' columns into datetime format using Pandas. So, the dataset now comprises these new columns along with the existing ones, enhancing the data's usability and flexibility.

7. Managing low-frequency Values in Categorical Columns

Victim_Sex: We begin by examining the 'Victim_Sex' column to understand the frequency of different sex categories among victims. In this process, we identify and exclude rows with sex values 'h' and '-' due to their rarity, resulting in a more focused dataset.

```
victim_sex_counts = crimeDF['Victim_Sex'].value_counts()
victim_sex_counts

#Output
m    439911
f    298637
x     69684
h        89
-         1
Name: Victim_Sex, dtype: int64
```

```
crimeDF = crimeDF[(crimeDF['Victim_Sex'] != '-') & (crimeDF['Victim_Sex'] != 'h')]
```

Victim Descent: The 'Victim_Descent' column is analyzed for the distribution of victim descent categories. Values with a low frequency, defined as occurring fewer than 1000 times, are recategorized as 'Other' for better data management.

```
victim_descent_counts = crimeDF['Victim_Descent'].value_counts()
victim_descent_counts

#Outputs
h    354099
w    165334
b    115496
x     76947
o     64123
a     17763
k      4309
```

```
f      3338
c      3095
j      1128
v       833
i       767
z       408
p       217
u       165
d        60
g        56
l        48
s        45
-         1
Name: Victim_Descent, dtype: int64
```

```
descent_counts = crimeDF['Victim_Descent'].value_counts()
values_to_encode_as_other = descent_counts[descent_counts < 1000].index.tolist()
crimeDF.loc[crimeDF['Victim_Descent'].isin(values_to_encode_as_other), 'Victim_Descent'] = 'Other'
```

Victim_Age: Ensuring data quality, we scrutinize the 'Victim_Age' column, revealing unrealistic negative age values. These negative values are removed, preserving cases where age is specified as 0 for further analysis. This process improves the dataset's accuracy.

```
crimeDF['Victim_Age'].value_counts()

#Output
0      197694
 30      18495
 35      18057
 31      17712
 29      17671
         ...
 97         61
-1          54
-2          12
 120         1
-3           1
Name: Victim_Age, Length: 103, dtype: int64
```

```
crimeDF = crimeDF[crimeDF['Victim_Age'] >= 0]
```

Status Column: The 'Status' column is explored, and values with a frequency below 2000 are filtered out, simplifying the data and enhancing its relevance.

```
value_counts = crimeDF['Status'].value_counts()
mask = crimeDF['Status'].isin(value_counts.index[value_counts >= 2000])
crimeDF = crimeDF[mask]
```

```
crimeDF['Status'].value_counts()

#Output
ic    646940
ao     87067
aa     70191
ja      2586
Name: Status, dtype: int64
```

Status_Description: A similar procedure is applied to the 'Status_Description' column, where values with less than 5000 occurrences are omitted. Furthermore, certain status descriptions are replaced with more informative labels, contributing to a more comprehensive dataset.

```
value_counts = crimeDF['Status_Description'].value_counts()
mask = crimeDF['Status_Description'].isin(value_counts.index[value_counts >= 5000])
crimeDF = crimeDF[mask]
```

```
status_mapping = {
    'invest cont': 'investigation_continued',
    'adult other': 'adult_other',
    'adult arrest': 'adult_arrest'
}

crimeDF['Status_Description'] = crimeDF['Status_Description'].replace(status_mapping)
```

```
crimeDF['Status_Description'].value_counts()

#Output
investigation_continued    646940
adult_other                 87067
adult_arrest                70191
Name: Status_Description, dtype: int64
```

These measures ensure that the dataset contains meaningful and representative data, eliminating rare or erroneous values to facilitate more effective analysis.

8. Cleaning Textual Columns

Area Name: In the 'Area_Name' column, spaces are replaced with underscores for consistency and easier data handling.

```
crimeDF['Area_Name'] = crimeDF['Area_Name'].str.replace(' ', '_')
crimeDF['Area_Name'].unique()
```

Crime Code Description: The 'Crime_Code_Description' column undergoes several transformations:

- Conversion to lowercase for uniformity.
- Removal of punctuation and special characters to clean the text.
- Tokenization into words, splitting the descriptions into individual terms for further analysis.

```
# Cleaning the 'Crime_Code_Description' column
crimeDF['Crime_Code_Description'] = crimeDF['Crime_Code_Description'].str.lower()
crimeDF['Crime_Code_Description'] = crimeDF['Crime_Code_Description'].str.replace('[^\w\s]', '')  # Remove punctuation

# Tokenization (split into words)
crimeDF['Crime_Code_Description'] = crimeDF['Crime_Code_Description'].str.split()

crimeDF['Crime_Code_Description']
```

Tokenized data in Crime Code Description:

```
0                            [battery, -, simple, assault]
1                            [battery, -, simple, assault]
2          [sex, offender, registrant, out, of, compliance]
3           [vandalism, -, misdeameanor, ($399, or, under)]
4         [vandalism, -, felony, ($400, &, over,, all, c...
                               ...
811658    [vandalism, -, felony, ($400, &, over,, all, c...
811659    [assault, with, deadly, weapon,, aggravated, a...
811660    [assault, with, deadly, weapon,, aggravated, a...
811661                                        [pickpocket]
811662     [vandalism, -, misdeameanor, ($399, or, under)]
Name: Crime_Code_Description, Length: 804198, dtype: object
```

Premise Description: This involves the following steps:

- Elimination of rows with low-frequency values (fewer than 100 occurrences).
- Standardization and cleaning of textual descriptions, including lowercasing and stripping spaces.

```
print(crimeDF['Premise_Description'].nunique())
print(crimeDF['Premise_Description'].value_counts())
value_counts = crimeDF['Premise_Description'].value_counts()
mask = crimeDF['Premise_Description'].isin(value_counts.index[value_counts >= 100])
crimeDF = crimeDF[mask]
# Clean and standardize the 'Premise_Description' column (e.g., lowercase and strip spaces)
crimeDF['Premise_Description'] = crimeDF['Premise_Description'].str.lower().str.strip()
```

Location: The 'Location' column is cleaned through the following procedures:

- Removal of rows with uncommon abbreviations with low occurrence rates.
- Mapping common abbreviations to their full forms, such as 'st' to 'Street' and 'av' to 'Avenue'.
- Stripping special characters to ensure uniformity and consistency.
- Elimination of extra spaces and formatting.
- Splitting the 'Location' column into different columns based on spaces, creating 'Location_Number,' 'Location_Street,' and 'Location_Type.'

```
pattern = r'\b(?:[a-z]{2})\b'
two_letter_words = re.findall(pattern, ' '.join(crimeDF['Location']))
word_counts = pd.Series(two_letter_words).value_counts()
word_counts

abbreviations_to_drop = word_counts[word_counts < 2500].index.tolist()

crimeDF = crimeDF[~crimeDF['Location'].str.extract(r'([a-z]{2})', expand=False).isin(abbreviations_to_drop)]

crimeDF.shape
crimeDF['Location'] = crimeDF['Location'].str.strip()
```

Subsequently, rows containing 'Location_Number' and 'Location_Street' are removed, leaving only the 'Location_Type' for analysis. Rows with low-frequency 'Location_Type' values (less than 500 occurrences) are filtered out, streamlining the dataset for more focused analysis.

These text column cleaning processes enhance the dataset's quality and readability for improved analysis.

```
crimeDF.shape

# Output
(457935, 20)
```

We can notice the shape of our dataset so far.

9. Handling Outliers in Numerical Columns

We initiate the process by assessing the dataset's information to understand its structure. We identify numerical columns that require outlier handling:

- 'Time_Occurred'
- 'Area'
- 'Reporting_District_Number'
- 'Part_1_or_2'
- 'Crime_Code'

- 'Victim_Age'
- 'Premise_Code'
- 'Crime_Code_1'
- 'Latitude'
- 'Longitude'

```
numerical_columns = ['Time_Occurred', 'Area', 'Reporting_District_Number', 'Part_1_or_2',
                     'Crime_Code', 'Victim_Age', 'Premise_Code', 'Crime_Code_1',
                     'Latitude', 'Longitude']

def handle_outliers_iqr(column):
    Q1 = np.percentile(column, 25)
    Q3 = np.percentile(column, 75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return column.apply(lambda x: max(lower_bound, min(upper_bound, x)))
# Apply the outlier handling function to each numerical column
for column in numerical_columns:
    crimeDF[column] = handle_outliers_iqr(crimeDF[column])
```

To address outliers, we implement a function named `handle_outliers_iqr`. This function calculates the Interquartile Range (IQR) and determines lower and upper bounds for each column. Data points falling outside these bounds are adjusted to the nearest boundary value.

Subsequently, we apply this outlier-handling function to each of the numerical columns, ensuring that the dataset is devoid of extreme values that could skew analysis or modeling.

By doing this, enhancing its suitability for further analysis.

10. Encoding Categorical Columns

Commencing this phase, we begin by inspecting the dataset to understand its structure and data types. Categorical columns are identified for encoding, and the selected columns for encoding are as follows:

- 'Area_Name'
- 'Victim_Sex'
- 'Victim_Descent'
- 'Status'
- 'Status_Description'

```
categorical_columns = crimeDF.select_dtypes(include=['object']).columns
categorical_columns
encode_this = ['Area_Name', 'Victim_Sex', 'Victim_Descent', 'Status', 'Status_Description']
# Initialize a dictionary to store label mappings
label_mappings = {}
# Encode categorical columns and store label mappings
label_encoder = LabelEncoder()

for column in encode_this:
    if crimeDF[column].dtype == 'object':
        encoded_values = label_encoder.fit_transform(df[column])
        crimeDF[column] = encoded_values  # Replace the original column with encoded values
        label_mappings[column] = {index: label for index, label in enumerate(label_encoder.classes_)}
```

To encode these categorical columns, a dictionary is prepared to store label mappings. The encoding is executed using a label encoder, ensuring each category is assigned a unique numerical value. These encoded values replace the original categorical columns in the dataset.

11. Data Normalization

We initiate the data transformation process by identifying categorical columns within the dataset. These columns are selected for encoding, aiming to represent categorical data numerically. The specific columns chosen for encoding are 'Area_Name,' 'Victim_Sex,' 'Victim_Descent,' 'Status,' and 'Status_Description.' Label mappings are established to record the associations between original labels and their corresponding encoded values. A label encoder is applied to perform the encoding, and the original categorical columns are substituted with their encoded counterparts.

```
normalizing_numerical_columns = ['Time_Occurred','Reporting_District_Number', 'Crime_Code',
                                 'Premise_Code', 'Crime_Code_1',
                                 'Latitude', 'Longitude']
```

Next, we focus on data normalization, targeting numerical columns. The columns selected for normalization include 'Time_Occurred,' 'Reporting_District_Number,' 'Crime_Code,' 'Premise_Code,' 'Crime_Code_1,' 'Latitude,' and 'Longitude.' The Min-Max scaling technique is employed to transform these numerical attributes, ensuring they fall within a consistent range.

Upon completing these data transformations, the dataset is saved as 'crimeDF_cleaned_data.csv,' ready for further analysis or modeling.

```
scaler = MinMaxScaler()
crimeDF[normalizing_numerical_columns] = scaler.fit_transform(crimeDF[normalizing_numerical_columns])
```

12. Redundant Columns Removal

There are very similar values in columns Crime_Code and Crime_Code_1 hence we drop Crime_Code_1.

```
print(crimeDF['Crime_Code_1'].value_counts())
print(crimeDF['Crime_Code'].value_counts())
crimeDF.drop(columns=['Crime_Code_1'],inplace=True)
```

We check the different values and their respective counts, we concluded that it is a redundant column. Hece we remove it using drop().

In a similar way the Status and Status descriptions are redundant after the encoding was done. Status had the abbrevated values of Status_Description. Hence we drop Status.

```
print(crimeDF['Status'].value_counts())
print(crimeDF['Status_Description'].value_counts())
crimeDF.drop(columns=['Status'],inplace=True)
```

Exporting the cleaned dataframe to csv:

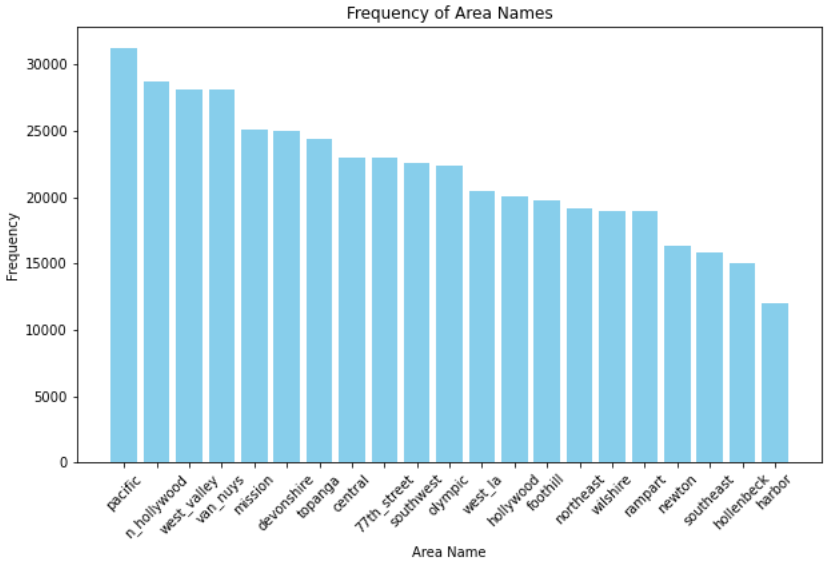
```
crimeDF.to_csv('crimeDF_cleaned_data.csv', index=False)
```

Exploratory Data Analysis

Let’s dive into Exploratory Data Analysis (EDA) phase to delve into the dataset and gain insights into its characteristics. Our primary objective is to understand the dataset's structure, distributions, and relationships, which will guide subsequent data-driven decisions.

We commence by loading the cleaned dataset, 'crimeDF_cleaned_data.csv,' and prepare it for analysis.

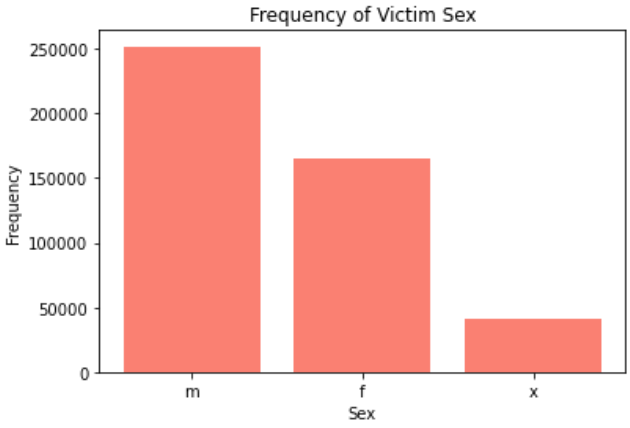
1. Univariate Analysis for 'Area_Name’



The bar plot, incorporating label mappings, reveals crucial insights into the geographical distribution of reported crimes. Key takeaways from this analysis:

- **Geographical Distribution**: The graph visually showcases how reported crimes are distributed across different areas. Some areas exhibit notably higher crime frequencies.
- **High Crime Areas**: Specific areas like"Pacific," "West Valley," and "Holywood," stand out with significantly more reported crimes, indicating them as hotspots for criminal activities.
- **Geographical Patterns**: Label mappings help us identify the specific area names associated with encoded values, enhancing our understanding of areas with higher crime rates.
- **Resource Allocation**: This analysis guides resource allocation and law enforcement efforts. Areas with high crime frequencies may require extra attention and resources for crime prevention.

2. Victim Sex Frequency



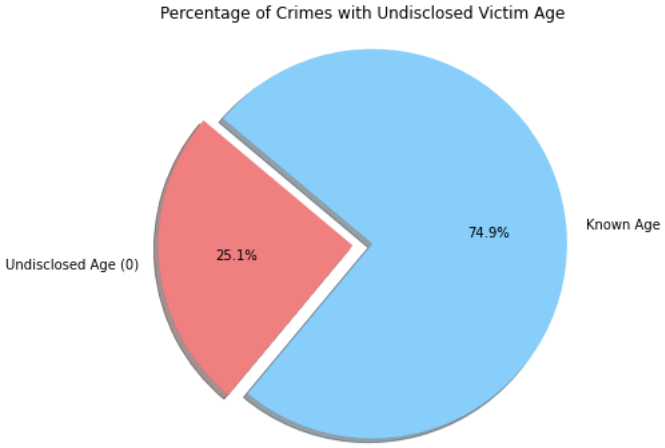
The bar plot depicting the frequency of victim sexes provides valuable insights into the distribution of victims in the dataset. From the graph, we can infer the following:

1. **Gender Distribution**: We can see the distribution of victims by gender. In this case, "Male" and "Female" are the two of the categories. An x representing non-binary or other gender classification. We observe the relative frequencies of male and female victims are higher and males are majorly affected.

2. **Gender Imbalance**: If there is a significant disparity between the frequencies of male and female victims, it suggests a gender-related pattern in the data. For instance, a large number of male victims indicate that certain types of crimes are more commonly committed against males.

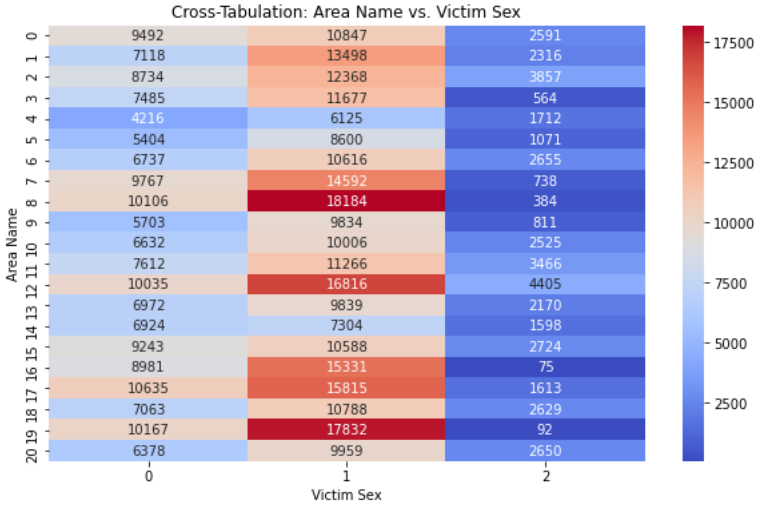
The graph provides an overview of the gender distribution among victims, helping us identify patterns and potential areas of interest for further analysis or investigation.

3. Analysis of Victim Age Disclosure



- The chart indicates that a significant portion of the crimes (approximately 25.1%) have undisclosed victim ages, where the age value is entered as 0. In contrast, the majority of crimes (approximately 74.9%) have known victim ages. This distribution highlights the prevalence of missing or undisclosed victim age data in the dataset.
- Understanding the extent of undisclosed age is crucial for data analysis and modeling since it may impact the accuracy and reliability of conclusions drawn. It may be due to the infants and crime against teenagers for privacy reasons they do not specify the ages.

4. Exploring Crime Victims by Area and Gender

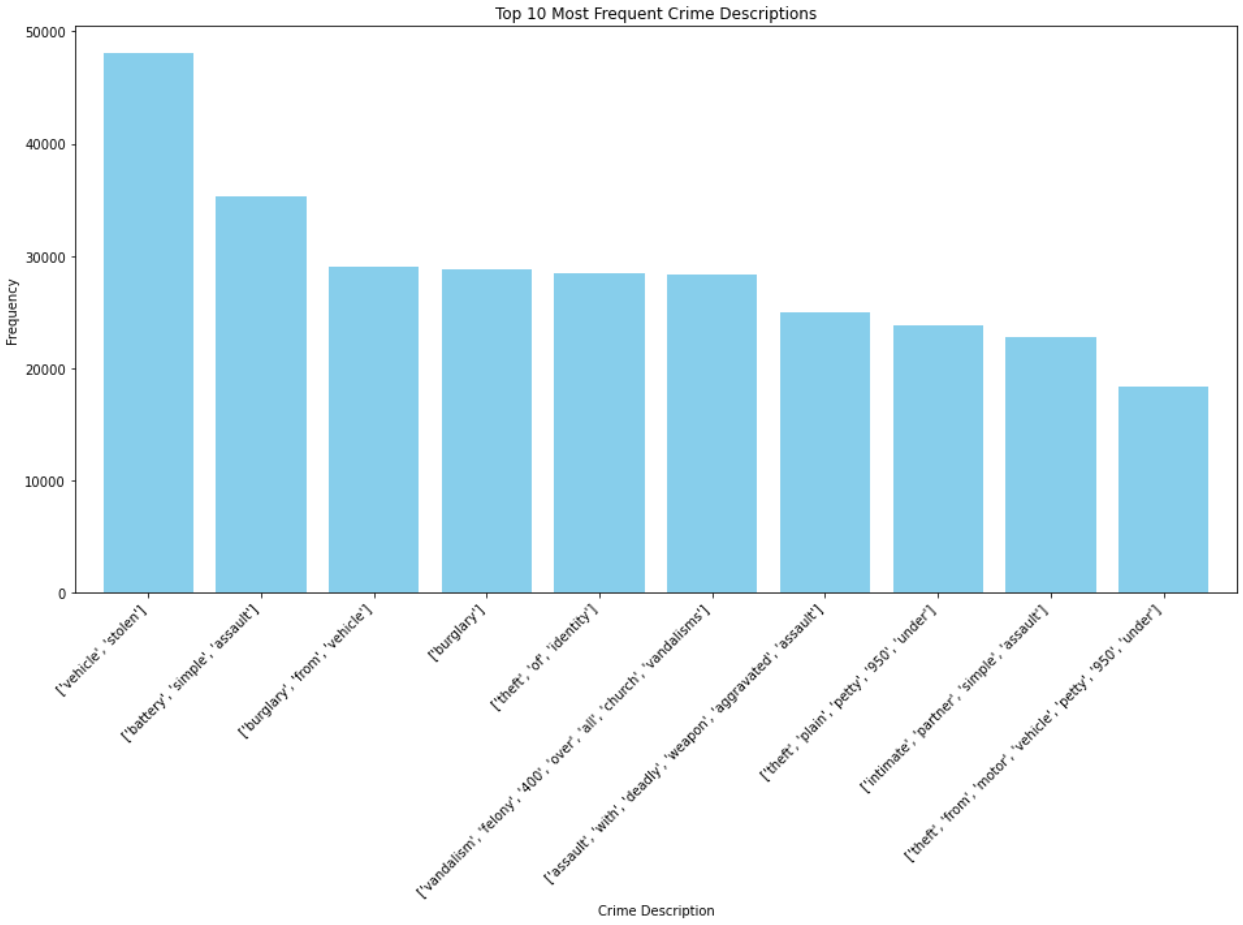


The heatmap showing the cross-tabulation between 'Area_Name' and 'Victim_Sex' provides insights into the distribution of crime victims by sex across different areas. Here's what we can infer from this visualization:

1. **Frequency of Crime Victims**: The heatmap reveals the number of crime victims (frequency) for each combination of 'Area_Name' (on the y-axis) and 'Victim_Sex' (on the x-axis). Areas with darker colors have higher victim counts, while lighter colors indicate lower counts.
2. **Gender Distribution**: We can see that both male (M) and female (F) victims are present across various areas, as indicated by the different color shades. This means that crimes are not exclusive to any particular gender.
3. **Area-Specific Patterns**: By examining the heatmap, we can identify areas where one gender is more prevalent among crime victims than the other. Some areas may have a relatively higher number of male victims compared to female victims, and vice versa. Area 8, 12, 19 have higher number of male victims.
4. **Gender-Neutral Areas**: In some areas, the distribution appears relatively balanced, suggesting that there is no significant gender bias in crime victimization in those locations.

Overall, the heatmap helps visualize the relationship between the 'Area_Name' and 'Victim_Sex' variables, allowing for a better understanding of gender-specific patterns in crime victimization across different areas.

5. Top 10 Most Frequent Crime Types



The graph shows the top 10 most frequent crime descriptions in Los Angeles from 2020 to the present day. The most frequent crime descriptions are:

- Vehicle theft
- Simple assault
- Battery theft
- Burglary from vehicle

- Burglary
- Vandalism
- Aggravated assault
- Intimate partner violence
- Theft from motor vehicle under $950

This suggests that property crimes are the most common type of crime in Los Angeles, followed by violent crimes. Identity theft is also a significant problem, suggesting that criminals are increasingly targeting personal information.

The graph also shows that the frequency of crimes varies depending on the crime type. For example, vehicle theft is the most frequent crime, while aggravated assault is the least frequent crime. This suggests that some crimes are more serious and have a greater impact on victims and society than others.

Overall, the graph provides a valuable snapshot of the most common crimes in Los Angeles. This information can be used to develop crime prevention strategies and to allocate resources to law enforcement and social services.

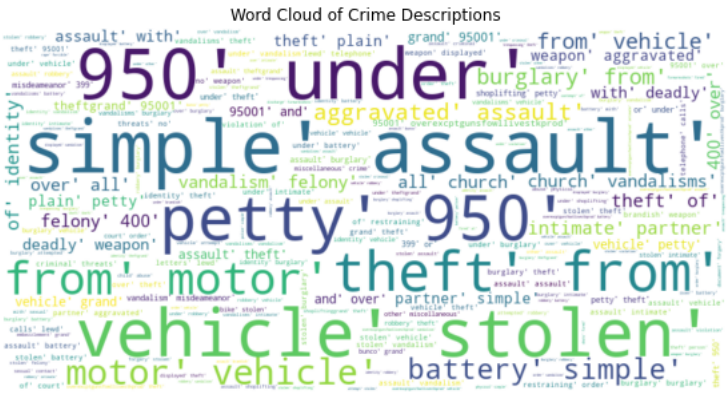
Some additional inferences that can be drawn from the graph:

- Property crimes are more common than violent crimes in Los Angeles. This is consistent with crime statistics from other major cities in the United States.
- The most common property crimes are vehicle theft and burglary. This suggests that criminals are primarily motivated by economic gain.
- Identity theft is a growing problem in Los Angeles, accounting for a significant proportion of all property crimes.
- The most common violent crime is simple assault. This suggests that violence is often impulsive and situational.
- Aggravated assault is the least common violent crime, but it is also the most serious. This suggests that aggravated assaults are more likely to be premeditated and result in serious injuries.

It is important to note that the graph only shows the most frequent crime descriptions. It does not provide information about the severity of the crimes or the impact on victims. For example, a homicide is obviously more serious than a theft. However, theft can also have a significant impact on victims, especially if it involves the loss of valuable property.

Overall, the graph provides a valuable snapshot of the most common crimes in Los Angeles. This information can be used to develop crime prevention strategies and to allocate resources to law enforcement and social services.

6. Word Cloud of Crime Descriptions



The word cloud highlights the most frequent crime descriptions in Los Angeles, revealing insights into prevalent crimes:

- Property crimes, particularly vehicle theft, burglary (including burglary from vehicles), and petty theft, dominate the dataset.
- Violent crimes like simple assault and aggravated assault are also notable.
- Identity theft stands out as a significant issue, indicating a focus on personal information by criminals.

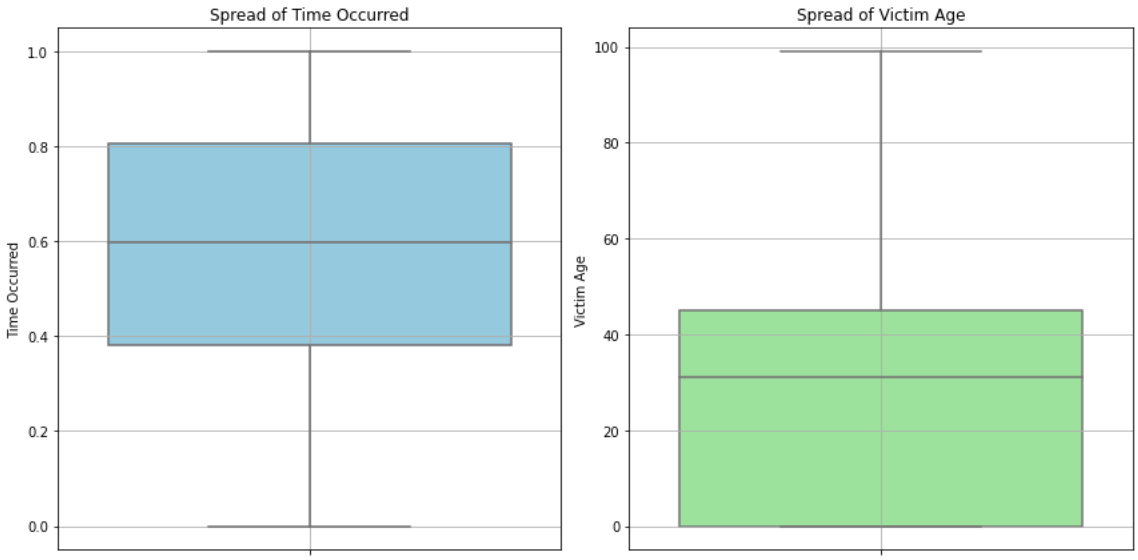
- The word cloud provides a visual representation of crime frequencies, emphasizing the prominence of property and violent crimes in Los Angeles.

It is a word cloud of crime descriptions from the City of Los Angeles dataset. The most frequent crime descriptions are:

- VEHICLE THEFT
- SIMPLE ASSAULT
- IDENTITY THEFT
- BURGLARY FROM VEHICLE
- BURGLARY
- VANDALISM
- AGGRAVATED ASSAULT
- INTIMATE PARTNER VIOLENCE
- THEFT FROM MOTOR VEHICLE UNDER $950
- PETTY THEFT UNDER $950

This suggests that property crimes are the most common type of crime in Los Angeles, followed by violent crimes. Identity theft is also a significant problem, suggesting that criminals are increasingly targeting personal information.

7. Exploring Variability



.

Based on the two box plots, the following inferences can be drawn:

Time Occurred

- The distribution of the time occurred is skewed up, meaning that most crimes occur at night.
- The interquartile range is 4 hours, meaning that the middle 50% of crimes occur between 8:00 PM and 12:00 AM.

Victim Age

- The distribution of victim age is skewed to the bottom, meaning that most victims are young.
- The median victim age is between 20 and 40 years old.
- The interquartile range is 40 years, meaning that the middle 50% of victims are between 0 and 40 years old.

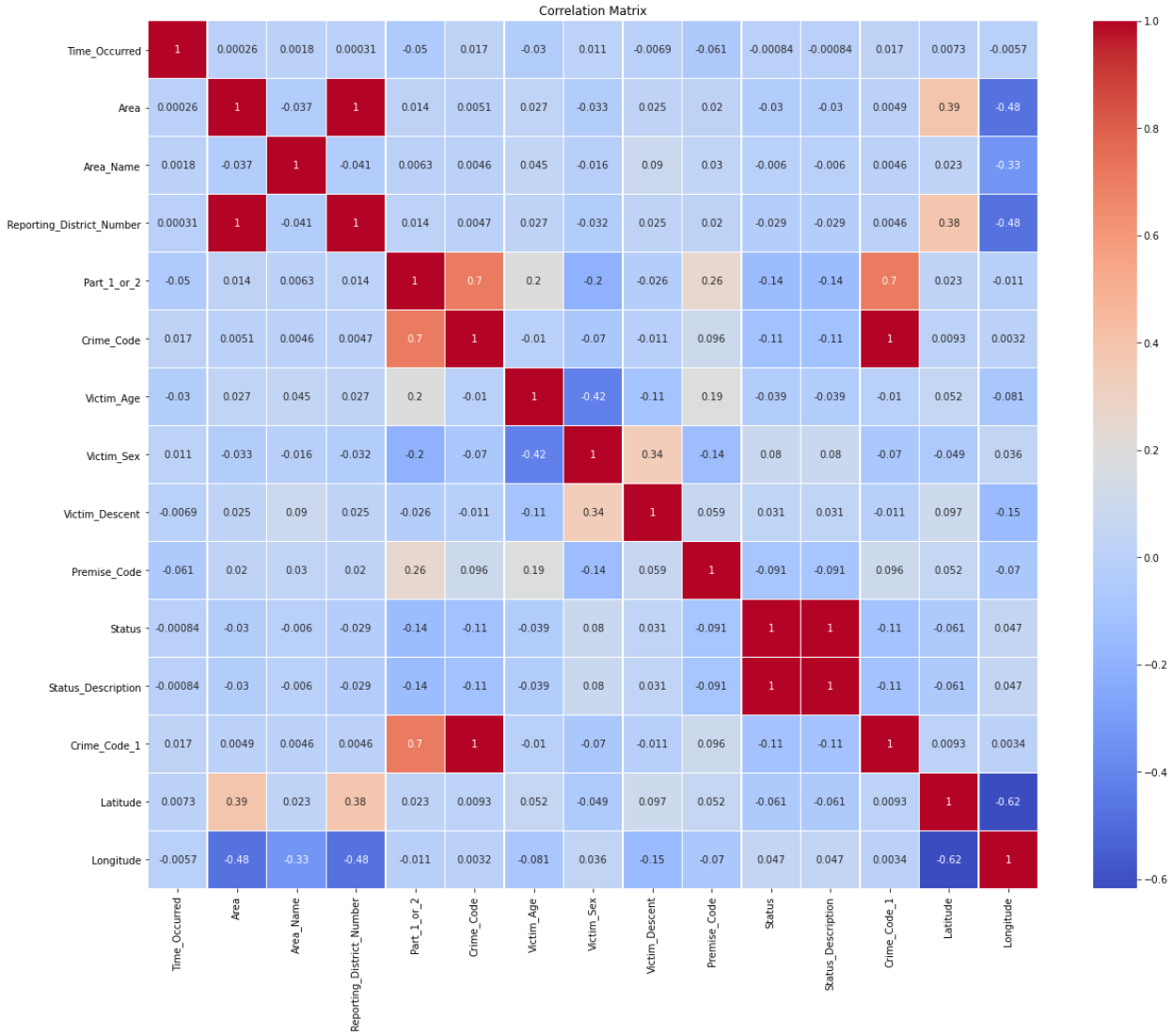
- There are a few outliers, meaning that there were a small number of victims who were very old.

Overall, the two box plots suggest that crimes are more likely to occur at night and that victims are more likely to be young.

- The spread of time occurred is greater than the spread of victim age. This suggests that there is more variability in the time of day that crimes occur than in the age of the victims.

8. Correlation Map Insights

The correlation map illustrates the relationships between 15 different variables, with correlation coefficients ranging from -0.6 to 1.0 and a median value of 0.1. This suggests a diverse range of correlation strengths among these variables, some being strongly correlated while others exhibit weaker associations.



Key takeaways from the correlation map include:

- **Strong Positive Correlation:** There exists a significant positive correlation between "Area" and "Reporting District Number." These variables tend to move in the same direction, meaning changes in one variable correspond to changes in the other.
- **Strong Negative Correlation:** "Longitude" exhibits a substantial negative correlation with both "Area" and "Reporting District Number." These relationships involve opposing movements; changes in one variable are linked to opposite changes in the other.

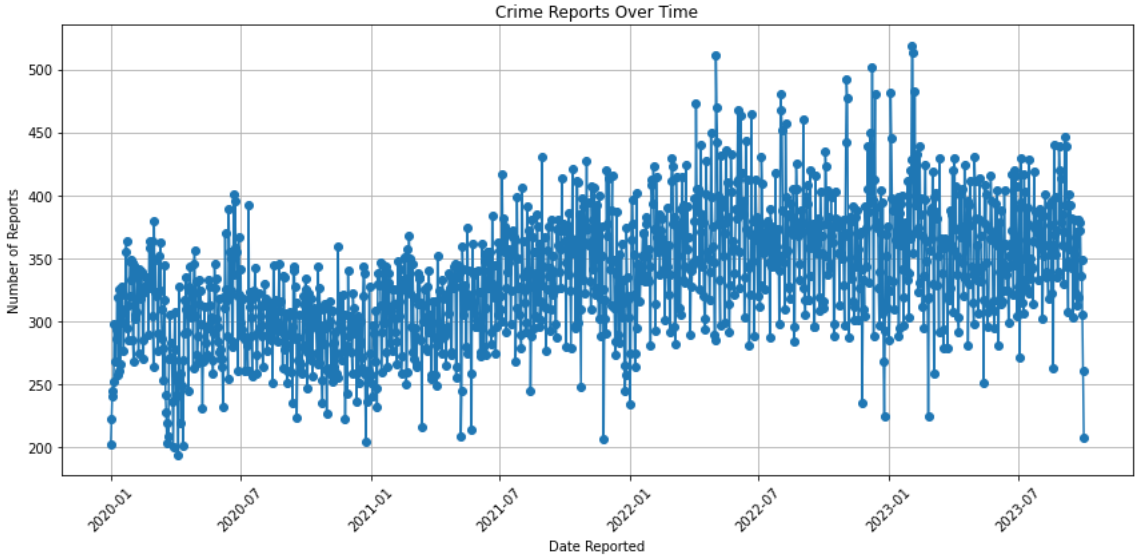
The correlation map shows the correlation coefficients between 15 different variables. The correlation coefficients ranges from -0.6 to 1.0, with a median of 0.1. This indicates that there is a wide range of correlation strengths among the variables, with some variables being strongly correlated and others being weakly correlated.

Here are some specific inferences that can be drawn from the correlation map:

- **Area and Reporting District Number are very strongly positively correlated.** This means that they tend to move in the same direction, and changes in one variable are associated with changes in the other variable.
- **Longitude with both Area and Reporting District Number strongly negatively correlated.** This means that they tend to move in opposite directions, and changes in one variable are associated with opposite changes in the other variable.

Overall, the correlation map provides a valuable summary of the relationships between the 15 variables. It can be used to identify potential relationships between variables, which can then be further investigated using other methods.

9. Crime Report Trends



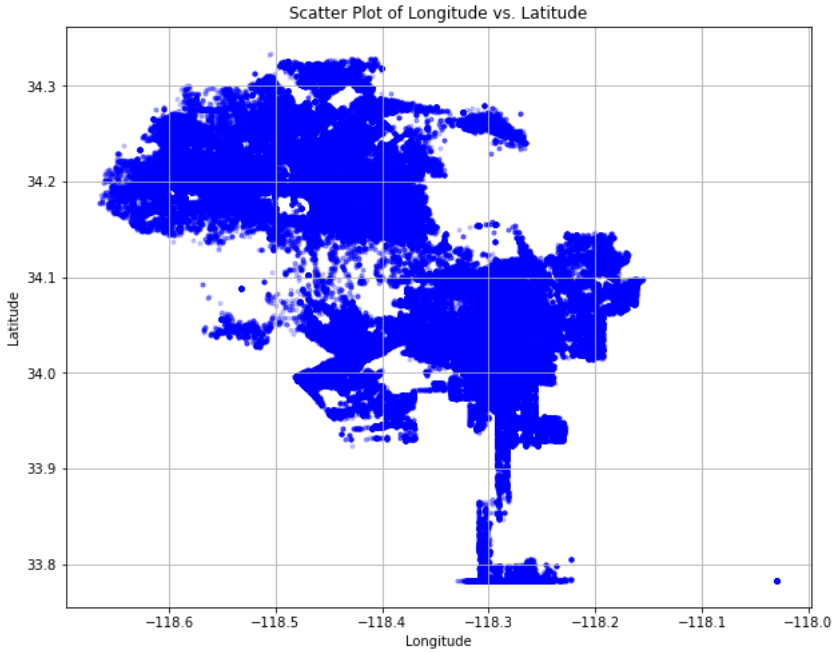
The line plot illustrates the number of reported crimes from January 2020 to July 2023. It displays a consistent increase in crime reporting with some fluctuations.

Key insights:

- **15% Increase:** Crime reporting rose by 15% between January 2020 and July 2023.
- **Surge in Late 2020:** A significant increase in crime reporting occurred in late 2020, followed by a stabilization in early 2021.
- **Continued Growth:** Crime reporting steadily increased in the latter part of 2021 and beyond.
- **March 2023 Peak:** March 2023 witnessed the highest crime reporting, surpassing 500 reported crimes.

Possible reasons for the rise include the COVID-19 pandemic's impact on financial stress and increased awareness of crime. It's important to note that the plot represents reported cases, not the actual number of crimes. Addressing the underlying causes of this trend is vital.

10. Crime Distribution Insights



The scatter plot displays the geographical distribution of crime reports, with latitude and longitude as coordinates. It highlights that crimes are spread across a wide area but exhibit concentrations in specific zones.

Key findings:

- **City Center Cluster:** A cluster of crimes is evident in the city center.
- **Southern Concentration:** Another cluster of crimes appears in the southern part of the city.
- **Suburban and Outlying Areas:** Suburban and outskirts regions show fewer reported crimes.

This helps in identifying high-crime areas that may require targeted crime prevention efforts. However, it's essential to remember that the plot solely indicates the location of crime reports, without specifying the type of crime at each site. This data can be used to further refine strategies for crime prevention.

Phase 2

In this section, we'll explore various algorithms to make valuable predictions that satisfy our problem statement which is to understand the importance of crime data to resolve various issues. Now let's look the various steps that helps us in extraction of our crime data.

Importing Required Libraries

Before we proceed, let's import various libraries that are necessary to create Statistical and analytical models. Here as you can see from the below code, we include pandas for data manipulation, numpy for numerical operations, matplotlib and seaborn for data visualization, and our Python libraries for data analysis and machine learning.

Next followed by LinearRegression, KMeans, LogisticRegression, MultinomialNB, DecisionTreeClassifier, XGBClassifier, and different evaluation metrics for regression and classification tasks were also imported from the scikit-learn library.

We also included data preprocessing libraries like train_test_split, StandardScaler, label_binarize, OneHotEncoder, and LabelEncoder. We used plotting libraries to visualize confusion matrices, ROC curves, precision-recall curves, and decision trees.

```
import numpy as np
import pandas as pd
```

```
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, label_binarize

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier

from sklearn.tree import plot_tree
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, roc_curve, confusion_matrix
```

Now after importing the required libraries, we imported the dataset as `crime_df`

```
crime_df = pd.read_csv('crimeDF_cleaned_data.csv')
crime_df
```

About the Dataset
Our dataset provides insights into crime incidents in Los Angeles from 2020 to the present, offering a valuable resource for analyzing crime trends. The data originates from transcriptions of original paper crime reports, but potential inaccuracies may exist due to the manual transcription process.

To learn more please visit the Phase 1 section of this report, where we described the dataset in more detail and analyzed various trends.

🔗 **Source:** City of Los Angeles Open Data

Deliverables 1 - Algorithms / Visualizations

Linear Regression [Algorithm 1]

Objective:

The goal of this analysis is to understand how 'Latitude' and 'Longitude' relate to the 'Area' and to create a linear regression model that can predict 'Area' based on these geographical coordinates. The visualization helps you visualize the relationship between the predictor variables and the target variable, providing insights into how well the linear regression model fits the data. The model performance metrics (MAE, MSE, R^2) give quantitative measures of the model's accuracy and predictive power.

- We started by selecting the features (Latitude and Longitude) and the target variable (Area) from our crime dataset.

```
X_linear_reg = crime_df[['Latitude','Longitude']]
y_linear_reg = crime_df['Area']
```

- Next, we split our data into training and testing sets using the train_test_split function, 80% for training and 20% for testing.

```
X_train_linear_reg, X_test_linear_reg, y_train_linear_reg, y_test_linear_reg = train_test_split(X_linear_reg, y_linear_reg, test_size=0.
2, random_state=42)
```

- We employed the Linear Regression model from scikit-learn and trained it on the training data (X_train_linear_reg, y_train_linear_reg)


```
X_train_linear_reg, X_test_linear_reg, y_train_linear_reg, y_test_linear_reg = train_test_split(X_linear_reg, y_linear_reg, test_size=0.
2, random_state=42)
```


- After training, we made predictions on the test data (X_test_linear_reg) using the trained model.

```
linear_model = LinearRegression()
linear_model.fit(X_train_linear_reg, y_train_linear_reg)
```

- We evaluated the model's performance by calculating the Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared values.

```
y_pred = linear_model.predict(X_test_linear_reg)

mae = mean_absolute_error(y_test_linear_reg, y_pred)
mse = mean_squared_error(y_test_linear_reg, y_pred)
r2 = r2_score(y_test_linear_reg, y_pred)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

- The coefficients and intercept of the linear regression model were examined.

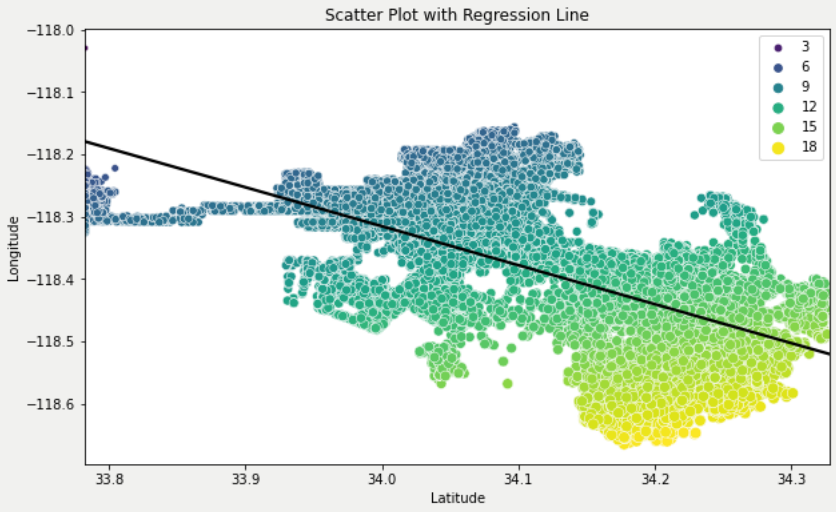
```
linear_model.coef_
linear_model.intercept_
```

- To visually represent the results, we created a scatter plot using Latitude and Longitude as axes, where the color and size of points corresponded to the predicted values.
- Additionally, we plotted the regression line on the scatter plot for better understanding.

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_test_linear_reg['Latitude'], y=X_test_linear_reg['Longitude'], hue=y_pred, palette="viridis", size=y_pred)
sns.regplot(x=X_test_linear_reg['Latitude'], y=X_test_linear_reg['Longitude'], scatter=False, color='black')
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Scatter Plot with Regression Line')
plt.legend(loc='upper right')
plt.show()
```

RESULTS

Mean Absolute Error: 4.609018309505537
Mean Squared Error: 27.14850101309127
R-squared: 0.24053047097154023



Logistic Regression [Algorithm 2]

Objective:

The goal of this analysis is to understand how well the logistic regression model can classify crimes as vehicle-related or not based on the selected features. The various performance metrics, visualizations (ROC curve, precision-recall curve, feature coefficients), and the confusion matrix helps us assess the model's accuracy and its ability to make correct predictions regarding vehicle-related crimes.

- Initially, we created a function named 'has_vehicle' that checks if the term 'vehicle' is present in the crime description and assigned a binary label (1 if present, 0 if not).

```
def has_vehicle(crime_list):
    return 1 if 'vehicle' in crime_list else 0
```

- We added a new column 'Logistic_Target' to our dataset based on whether the crime description contained the term 'vehicle' using the 'has_vehicle' function.

```
logit_df=crime_df
```

- Selected features for our logistic regression model, including 'Time_Occurred', 'Area', 'Victim_Age', and 'Premise_Code'.
- Checked the distribution of our target variable ('Logistic_Target') and printed the counts of each class (1 and 0).

```
logit_df['Logistic_Target'] = logit_df['Crime_Code_Description'].apply(has_vehicle)

selected_features_logit = ['Time_Occurred', 'Area', 'Victim_Age', 'Premise_Code']
```

```
#Output:
0    341364
1    116571
Name: Logistic_Target, dtype: int64
```

- Split the data into training and testing sets, allocating 80% for training and 20% for testing, to assess the model's performance.

```
X_train_logit, X_test_logit, y_train_logit, y_test_logit = train_test_split(X_logit, y_logit, test_size=0.2, random_state=42)

X_train_logit, X_test_logit, y_train_logit, y_test_logit = train_test_split(X_logit, y_logit, test_size=0.2, random_state=42)
```

- Applied standard scaling to normalize our feature values in both the training and testing sets.
- Utilized the Logistic Regression model from scikit-learn and trained it on the scaled training data.

```
scaler = StandardScaler()
X_train_logit = scaler.fit_transform(X_train_logit)
X_test_logit = scaler.transform(X_test_logit)

logistic_model = LogisticRegression()
logistic_model.fit(X_train_logit, y_train_logit)
```

- Made predictions on the test set using the trained logistic regression model.

```
y_pred_logit = logistic_model.predict(X_test_logit)
```

- Evaluated the model's performance using metrics such as accuracy, precision, recall, and F1 score.

```
accuracy_logit = accuracy_score(y_test_logit, y_pred_logit)
precision_logit = precision_score(y_test_logit, y_pred_logit)
recall_logit = recall_score(y_test_logit, y_pred_logit)
f1_logit = f1_score(y_test_logit, y_pred_logit)

print("Accuracy:", accuracy_logit)
print("Precision:", precision_logit)
print("Recall:", recall_logit)
print("F1 Score:", f1_logit)
```

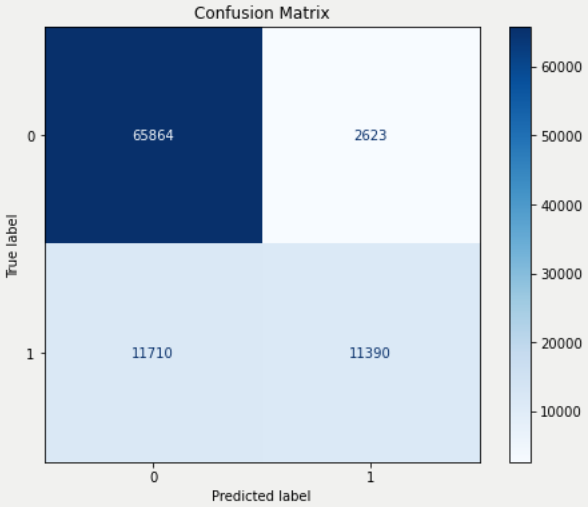
- Displayed the confusion matrix, a visual representation of the model's performance on the test set.

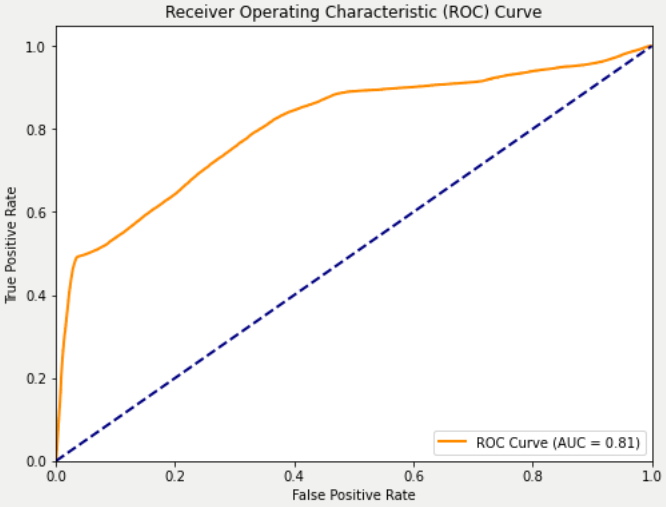
```
fig, ax = plt.subplots(figsize=(8, 6))
plot_confusion_matrix(logistic_model, X_test_logit, y_test_logit, ax=ax, cmap=plt.cm.Blues, values_format='d')
plt.title("Confusion Matrix")
plt.show()
# --------   ROC
fpr, tpr, thresholds = roc_curve(y_test_logit, logistic_model.predict_proba(X_test_logit)[:, 1])
auc = roc_auc_score(y_test_logit, logistic_model.predict_proba(X_test_logit)[:, 1])
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve (AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
# -------------- Feature Coefficients
coefficients_logit = logistic_model.coef_[0]
feature_names_logit = X_logit.columns
coef_df = pd.DataFrame({'Feature': feature_names_logit, 'Coefficient': coefficients_logit})
coef_df = coef_df.sort_values(by='Coefficient', ascending=False)
plt.figure(figsize=(8, 6))
sns.barplot(x='Coefficient', y='Feature', data=coef_df)
```

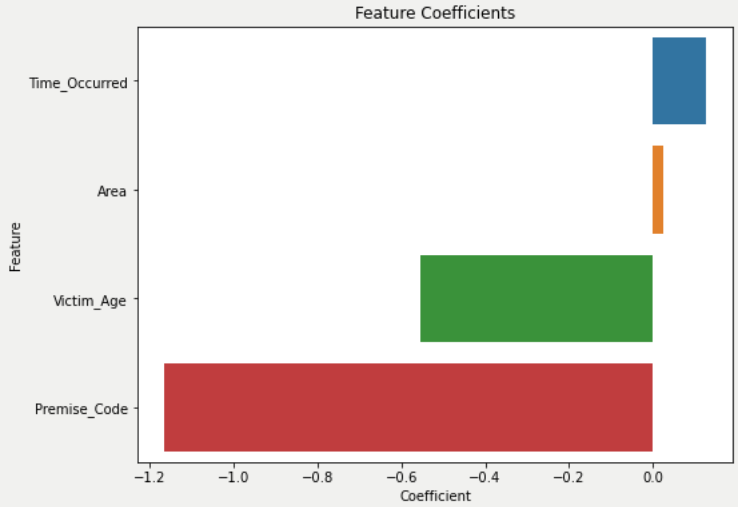
```
plt.title('Feature Coefficients')
plt.show()

# ------- Probability distribution for vehicle theft vs not vechile theft over time

sns.kdeplot(y_pred_logit[y_test_logit == 0], label='Not a vehicle theft', shade=True)
sns.kdeplot(y_pred_logit[y_test_logit == 1], label='Vehicle theft', shade=True)
plt.xlabel('Predicted Probabilities')
plt.ylabel('Density')
plt.title('Predicted Probability Distribution')
plt.legend()
plt.show()
```

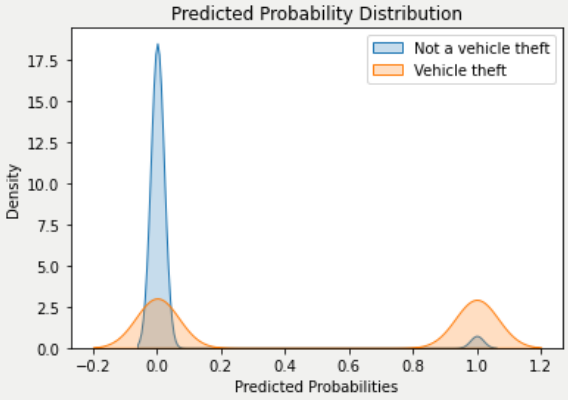
RESULTS

Accuracy: 0.843503990741044
Precision: 0.812816670234782
Recall: 0.49307359307359305
F1 Score: 0.6138010939562956









Naive Bayes [Algorithm 3]

Objective:

We are applying the Naive Bayes classifier to perform a multiclass classification task on the crime dataset, which is to be related to predicting crime areas based on various features. The goal is to classify crimes into their respective areas, using Multinomial Naive Bayes.

- Next we applied Naive Bayes classifier to conduct a multiclass classification task on our crime dataset, aiming to predict crime areas based on various features.
- Initialized a new DataFrame for Naive Bayes (nb_df) using the original crime dataset.

```
nb_df = crime_df
```

- Employed OneHotEncoder to transform the 'Area' column into a one-hot encoded format for model compatibility.

```
encoder = OneHotEncoder(sparse=False)
area_encoded = encoder.fit_transform(nb_df['Area'].values.reshape(-1, 1))
```

- Combined the transformed data with the original DataFrame, dropping unnecessary columns like 'Date_Reported,' 'Date_Occurred,' 'Crime_Code_Description,' 'Longitude,' and 'Latitude.'

```
X_nb = pd.concat([nb_df.drop(columns=['Area']), pd.DataFrame(area_encoded, columns=encoder.get_feature_names(['Area']))], axis=1)
y_nb = nb_df['Area']
```

- Utilized LabelEncoder to convert categorical columns ('Premise_Description' and 'Location_Type') into numerical format.

```
X_nb.drop(columns=['Date_Reported','Date_Occurred','Crime_Code_Description','Longitude','Latitude' ],inplace = True)

X_nb['Area_1'].value_counts()

label_encoder = LabelEncoder()
X_nb['Premise_Description'] = label_encoder.fit_transform(X_nb['Premise_Description'])
X_nb['Location_Type'] = label_encoder.fit_transform(X_nb['Location_Type'])
```

- Split the data into training and testing sets, allocating 80% for training and 20% for testing, ensuring consistent randomization.

```
X_train_nb, X_test_nb, y_train_nb, y_test_nb = train_test_split(X_nb, y_nb, test_size=0.2, random_state=42)
```

- Trained the Naive Bayes model (MultinomialNB) on the training data.

```
naive_bayes_model = MultinomialNB()
naive_bayes_model.fit(X_train_nb, y_train_nb)
```

- Made predictions and calculated prediction probabilities on the test set using the trained model.

```
y_pred_nb = naive_bayes_model.predict(X_test_nb)
y_pred_proba_nb = naive_bayes_model.predict_proba(X_test_nb)
```

- Evaluated the model's performance using metrics such as accuracy and generated a detailed classification report.

```
accuracy_nb = accuracy_score(y_test_nb, y_pred_nb)
report_nb = classification_report(y_test_nb, y_pred_nb)
print("Accuracy:", accuracy_nb)
print("Classification Report:")
print(report_nb)
```

- Displayed a visual representation of the model's performance through a confusion matrix.

```
fig, ax = plt.subplots(figsize=(20, 16))
plot_confusion_matrix(naive_bayes_model, X_test_nb, y_test_nb, ax=ax, cmap=plt.cm.Blues, values_format='d')
plt.title("Confusion Matrix")
plt.show()


# ------- Precision Curve
y_test_nb = np.array(y_test_nb)
y_test_nb = label_binarize(y_test_nb, classes=np.unique(y_test_nb))
precisions_nb = []
recalls_nb = []
avg_precisions_nb = []

plt.figure(figsize=(8, 6))
n_classes_nb = y_test_nb.shape[1]

for i in range(n_classes_nb):
    precision, recall, _ = precision_recall_curve(y_test_nb[:, i], y_pred_proba_nb[:, i])
    avg_precision = average_precision_score(y_test_nb[:, i], y_pred_proba_nb[:, i])
    precisions_nb.append(precision)
```

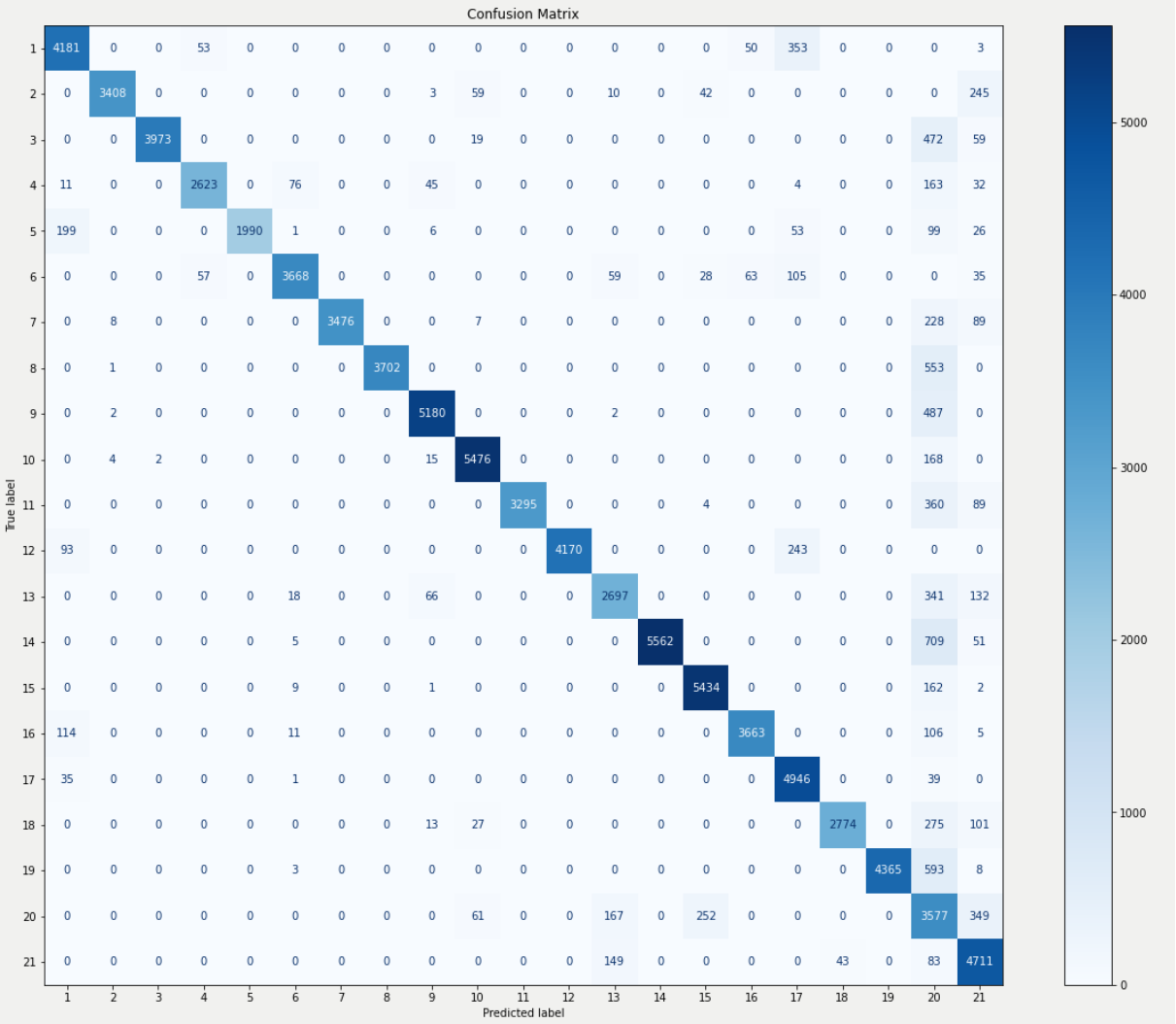
```
    recalls_nb.append(recall)
    avg_precisions_nb.append(avg_precision)

    plt.plot(recall, precision, lw=2, label=f'Class {i} (Avg. Precision = {avg_precision:.2f})')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve for Multiclass')
plt.legend(loc='lower left')
plt.show()

# ------------- Histogram
class_counts = np.bincount(y_nb)
plt.figure(figsize=(8, 6))
plt.bar(range(len(class_counts)), class_counts)
plt.xticks(range(len(class_counts)), range(len(class_counts)))
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Class Distribution')
plt.show()
```

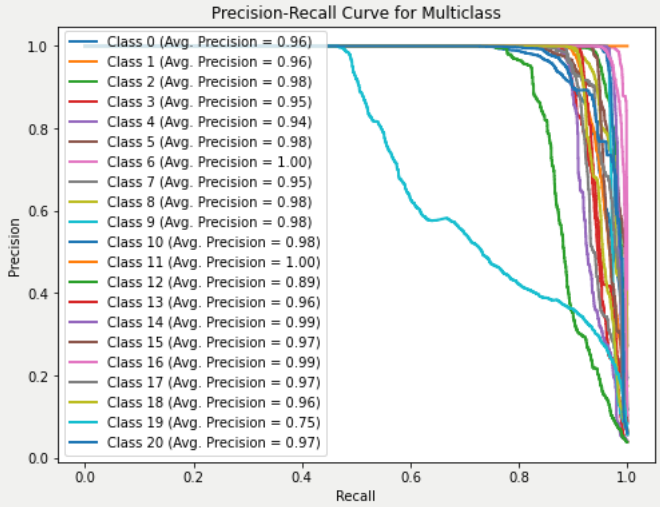
RESULTS

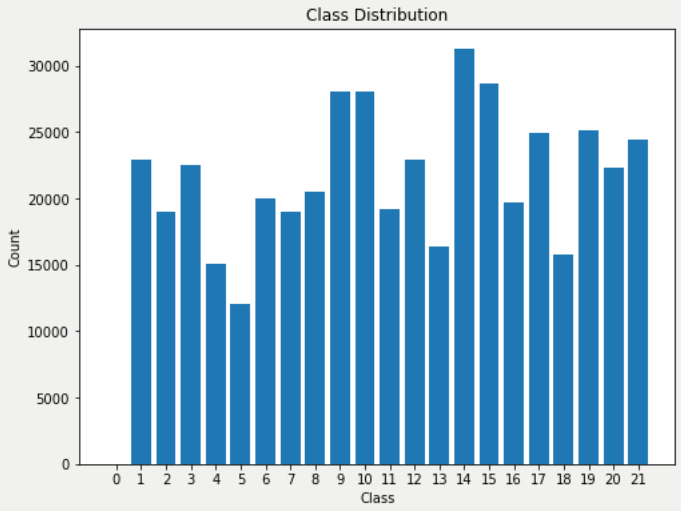
Accuracy: 0.9048336554314477
Classification Report:
precision recall f1-score support

```
       1       0.90      0.90      0.90      4640
       2       1.00      0.90      0.95      3767
       3       1.00      0.88      0.94      4523
       4       0.96      0.89      0.92      2954
       5       1.00      0.84      0.91      2374
       6       0.97      0.91      0.94      4015
       7       1.00      0.91      0.95      3808
       8       1.00      0.87      0.93      4256
       9       0.97      0.91      0.94      5671
      10       0.97      0.97      0.97      5665
      11       1.00      0.88      0.94      3748
      12       1.00      0.93      0.96      4506
      13       0.87      0.83      0.85      3254
      14       1.00      0.88      0.94      6327
      15       0.94      0.97      0.96      5608
      16       0.97      0.94      0.95      3899
      17       0.87      0.99      0.92      5021
      18       0.98      0.87      0.92      3190
      19       1.00      0.88      0.94      4969
      20       0.43      0.81      0.56      4406
      21       0.79      0.94      0.86      4986

accuracy                           0.90     91587
```

macro avg 0.93 0.90 0.91 91587
weighted avg 0.93 0.90 0.91 91587


Confusion Matrix
True label
Predicted label
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
0 1000 2000 3000 4000 5000


Precision-Recall Curve for Multiclass
Class 0 (Avg. Precision = 0.96)
Class 1 (Avg. Precision = 0.96)
Class 2 (Avg. Precision = 0.98)
Class 3 (Avg. Precision = 0.95)
Class 4 (Avg. Precision = 0.94)
Class 5 (Avg. Precision = 0.98)
Class 6 (Avg. Precision = 1.00)
Class 7 (Avg. Precision = 0.95)
Class 8 (Avg. Precision = 0.98)
Class 9 (Avg. Precision = 0.98)
Class 10 (Avg. Precision = 0.98)
Class 11 (Avg. Precision = 1.00)
Class 12 (Avg. Precision = 0.89)
Class 13 (Avg. Precision = 0.96)
Class 14 (Avg. Precision = 0.99)
Class 15 (Avg. Precision = 0.97)
Class 16 (Avg. Precision = 0.99)
Class 17 (Avg. Precision = 0.97)
Class 18 (Avg. Precision = 0.96)
Class 19 (Avg. Precision = 0.75)
Class 20 (Avg. Precision = 0.97)
Precision
Recall
0.0 0.2 0.4 0.6 0.8 1.0



KNN [Algorithm 4]

Objective:

KNN algorithm is chosen for the problem of predicting crime status descriptions. Given a new instance of crime we can use our model to get an insight on whether this can be resolved easily, or is a time consuming case to solve by the police. The status description has whether the crime was under investigation, suspect identified or whether the adult is arrested.

- We implemented the K-Nearest Neighbors (KNN) algorithm to perform a classification task on our crime dataset, specifically focusing on predicting the 'Status_Description' of reported incidents.
- Created a new DataFrame (knn_df) for KNN analysis using the original crime dataset.
- Prepared the feature set (X_knn) by excluding the 'Status_Description' column and designated the 'Status_Description' column as the target variable (y_knn).

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score

knn_df=crime_df
X_knn = knn_df.drop(columns=['Status_Description'])
y_knn = knn_df['Status_Description']

X_knn
```

- Removed unnecessary columns ('Date_Reported,' 'Date_Occurred,' 'Crime_Code_Description,' 'Premise_Description') from the feature set to refine our model input.
- Applied LabelEncoder to convert categorical values in the 'Location_Type' column into numerical format.

```
unsuitable_columns_knn = ['Date_Reported', 'Date_Occurred', 'Crime_Code_Description','Premise_Description']
X_knn.drop(columns=unsuitable_columns_knn, inplace=True)
```

- Transformed the target variable (y_knn) using LabelEncoder to make it compatible with the KNN model.

```
label_encoder = LabelEncoder()
X_knn['Location_Type'] = label_encoder.fit_transform(X_knn['Location_Type'])
y_knn = label_encoder.fit_transform(y_knn)
```

- Split the data into training and testing sets, allocating 80% for training and 20% for testing, maintaining a consistent randomization.

```
X_train_knn, X_test_knn, y_train_knn, y_test_knn = train_test_split(X_knn, y_knn, test_size=0.2, random_state=42)
```

- Configured the KNN model with a specified number of neighbors (k=5) and trained it on the training data.

```
k = 5
knn_model = KNeighborsClassifier(n_neighbors=k)

knn_model.fit(X_train_knn, y_train_knn)
```

- Generated predictions on the test set using the trained KNN model.
- Assessed the model's performance by calculating accuracy and producing a detailed classification report.

```
y_pred_knn = knn_model.predict(X_test_knn)

accuracy_knn = accuracy_score(y_test_knn, y_pred_knn)
report_knn = classification_report(y_test_knn, y_pred_knn)
```

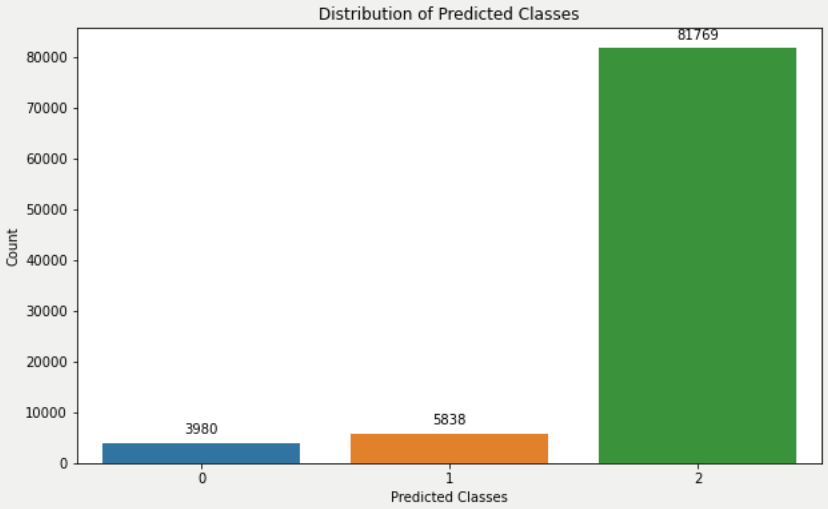
- Visualized the distribution of predicted classes through a countplot and annotated the plot with the corresponding class counts.

```
plt.figure(figsize=(10, 6))
class_counts = sns.countplot(x=y_pred_knn)
plt.xlabel('Predicted Classes')
plt.ylabel('Count')
plt.title('Distribution of Predicted Classes')
for count, p in zip(class_counts.patches, class_counts.patches):
    class_counts.annotate(f'{count.get_height()}', (p.get_x() + p.get_width() / 2., count.get_height()),
                          ha='center', va='center', xytext=(0, 10), textcoords='offset points')
plt.show()
```

RESULTS

Accuracy: 0.7627392533874895

```
Classification Report:
   precision  recall   f1-score   support
       0       0.21      0.10      0.14      8336
       1       0.27      0.16      0.20      9960
       2       0.82      0.92      0.87     73291

accuracy                           0.76     91587
macro avg       0.44      0.39      0.40     91587
weighted avg    0.71      0.76      0.73     91587
```



Decision Tree [Algorithm 5]

Objective:

The goal of this analysis is to build a Decision Tree model that can classify crime descriptions into the top 5 categories based on specific features. The model's performance is evaluated using accuracy and various classification metrics, and a visualization of the Decision Tree structure is provided to gain insights into the decision-making process of the model.

- We began by creating a new DataFrame (dtree_df) for decision tree analysis using the original crime dataset.

```
dtree_df= crime_df
```

- Identified the top five crime descriptions with the highest frequency in the dataset.

```
top_crime_descriptions_dtree = dtree_df['Crime_Code_Description'].value_counts().head(5).index
top_crime_descriptions_dtree

# Output
Index(['['vehicle', 'stolen']', '['battery', 'simple', 'assault']',
       '['burglary', 'from', 'vehicle']', '['burglary']',
       '['theft', 'of', 'identity']'],
      dtype='object')
```

- Filtered the dataset to include only rows with crime descriptions matching the top five identified earlier.

```
dtree_df = dtree_df[dtree_df['Crime_Code_Description'].isin(top_crime_descriptions_dtree)]

dtree_df
```

- Selected relevant features ('Victim_Sex,' 'Part_1_or_2,' 'Status_Description') and the target variable ('Crime_Code_Description') for the decision tree model.

```
features_dtree = ['Victim_Sex','Part_1_or_2','Status_Description']
target_dtree = 'Crime_Code_Description'
```

- Split the data into training and testing sets, allocating 80% for training and 20% for testing, with a fixed random state for consistency.

```
X_dtree = dtree_df[features_dtree]
y_dtree = dtree_df[target_dtree]

X_train_dtree, X_test_dtree, y_train_dtree, y_test_dtree = train_test_split(X_dtree, y_dtree, test_size=0.2, random_state=42)
```

- Configured and trained a Decision Tree classifier using the training data.
- Generated predictions on the test set using the trained Decision Tree model.

```
decision_tree_model = DecisionTreeClassifier(random_state=42)
decision_tree_model.fit(X_train_dtree, y_train_dtree)

y_pred_dtree = decision_tree_model.predict(X_test_dtree)
```

- Now let’s calculate the accuracy and producing a detailed classification report.

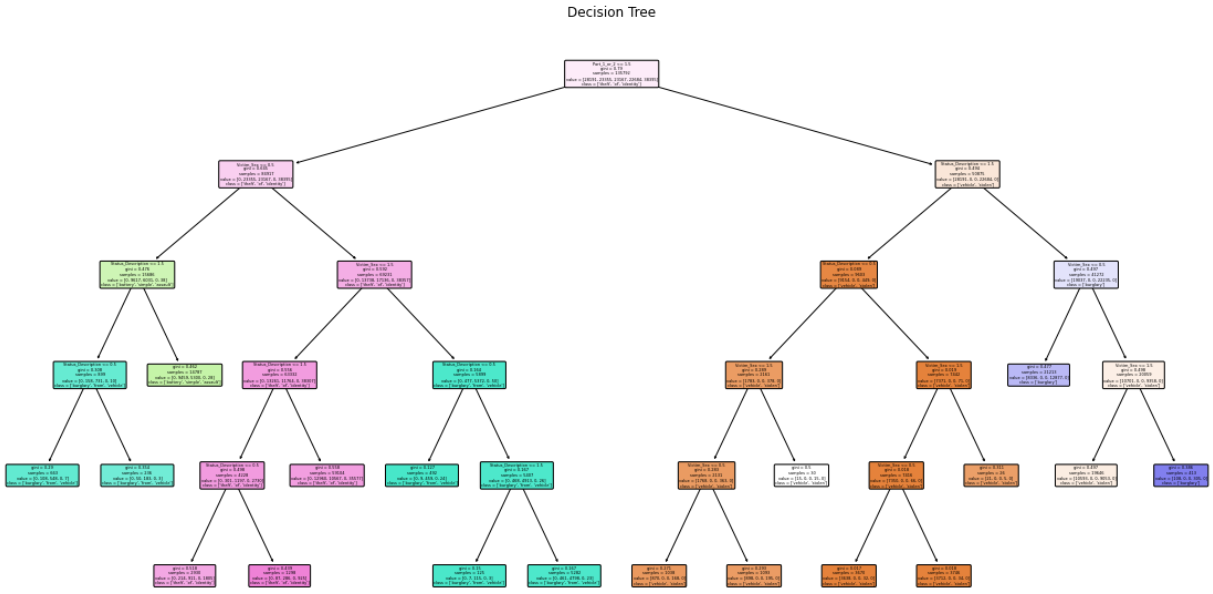
```
accuracy_dtree = accuracy_score(y_test_dtree, y_pred_dtree)
classification_rep_dtree = classification_report(y_test_dtree, y_pred_dtree)

print("Accuracy:", accuracy_dtree)
print("Classification Report:")
print(classification_rep_dtree)
```

- Created a visual representation of the Decision Tree, displaying feature names and class names, and filled nodes with color based on the top crime descriptions.

```
plt.figure(figsize=(20, 10))
plot_tree(decision_tree_model, feature_names=X_dtree.columns, class_names=top_crime_descriptions_dtree, filled=True, rounded=True)
plt.title("Decision Tree")
plt.show()
```

RESULTS

```
Accuracy: 0.6367492415093228
Classification Report:
                                precision    recall  f1-score   support

['battery', 'simple', 'assault']       0.66      0.70      0.68      7069
 ['burglary', 'from', 'vehicle']       0.62      0.39      0.48      5687
                    ['burglary']       0.89      0.26      0.41      5697
     ['theft', 'of', 'identity']       0.61      0.57      0.59      5822
           ['vehicle', 'stolen']       0.61      1.00      0.76      9674

                        accuracy                           0.64     33949
                       macro avg       0.68      0.58      0.58     33949
                    weighted avg       0.67      0.64      0.61     33949
```



XG Boost [Algorithm 6]

Objective:

Training an XGBoost model for crime Location Type classification based on several features. And then conducting a rigorous evaluation, visualizing feature importance, and assessing precision-recall characteristics for each class. We are trying to gain insights on the location type based on several features so that in future given limited information on the crime, the police can take swiftt action based on the location type and reach the crime site swiftly.

- We initiated the process by creating a new DataFrame (xgb_df) for analysis using the original crime dataset.

```
xgb_df = crime_df
```

- Focused on predicting the 'Location_Type' using an XGBoost classifier, a powerful machine learning algorithm.

- Prepared our feature set (X_xgb) by excluding columns like 'Location_Type,' 'Date_Reported,' 'Date_Occurred,' 'Crime_Code_Description,' and 'Premise_Description.'
- Designated 'Location_Type' as the target variable (y_xgb) for the XGBoost model.

```
X_xgb = xgb_df.drop(columns=['Location_Type','Date_Reported', 'Date_Occurred', 'Crime_Code_Description', 'Premise_Description'])
y_xgb = xgb_df['Location_Type']
```

- Applied LabelEncoder to transform categorical values in 'Location_Type' into numerical format, making them suitable for the model.

```
label_encoder = LabelEncoder()
y_xgb = label_encoder.fit_transform(xgb_df['Location_Type'])
```

- Split the data into training and testing sets, reserving 20% for testing, and set a fixed random state for consistency.
- Configured and trained the XGBoost classifier on the training data.
- Generated predictions on the test set using the trained XGBoost model.

```
X_train_xgb, X_test_xgb, y_train_xgb, y_test_xgb = train_test_split(X_xgb, y_xgb, test_size=0.2, random_state=42)

xgb_classifier = XGBClassifier()

xgb_classifier.fit(X_train_xgb, y_train_xgb)

#Output
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, objective='multi:softprob', ...)
```

- Assessed the model's performance by calculating accuracy, creating a detailed classification report, and constructing a confusion matrix.

```
y_pred_xgb = xgb_classifier.predict(X_test_xgb)

accuracy = accuracy_score(y_test_xgb, y_pred_xgb)
classification_rep = classification_report(y_test_xgb, y_pred_xgb)
confusion_mat = confusion_matrix(y_test_xgb, y_pred_xgb)

print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_rep)
print("Confusion Matrix:")
print(confusion_mat)
```

- Displayed the top 15 feature importances through a horizontal bar plot.
- Plotted a Precision-Recall curve for each class, illustrating the trade-off between precision and recall.

```
plt.figure(figsize=(12, 6))
feat_importances = pd.Series(xgb_classifier.feature_importances_, index=X_xgb.columns)
feat_importances.nlargest(15).plot(kind='barh')
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
```

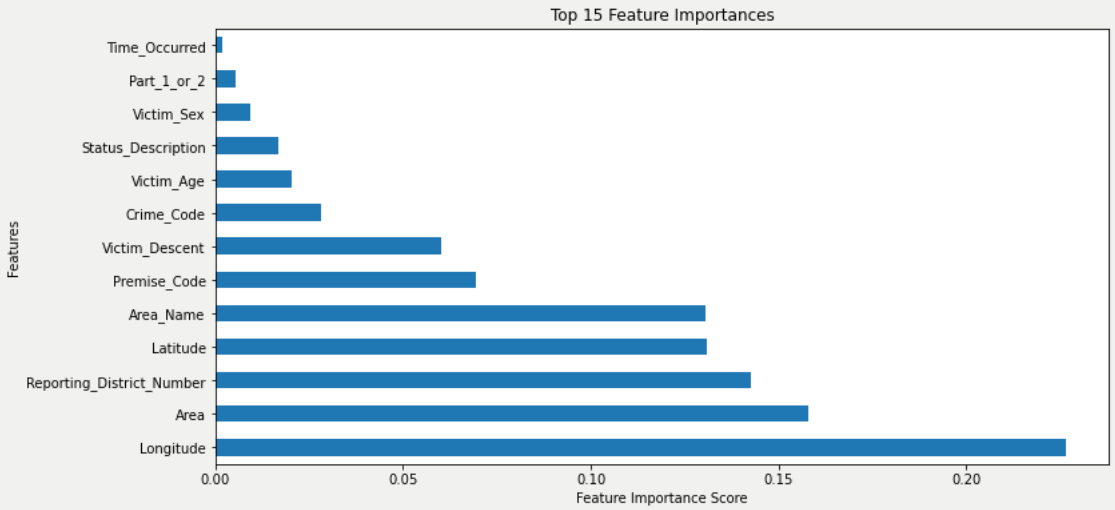
```
plt.title('Top 15 Feature Importances')
plt.show()
```

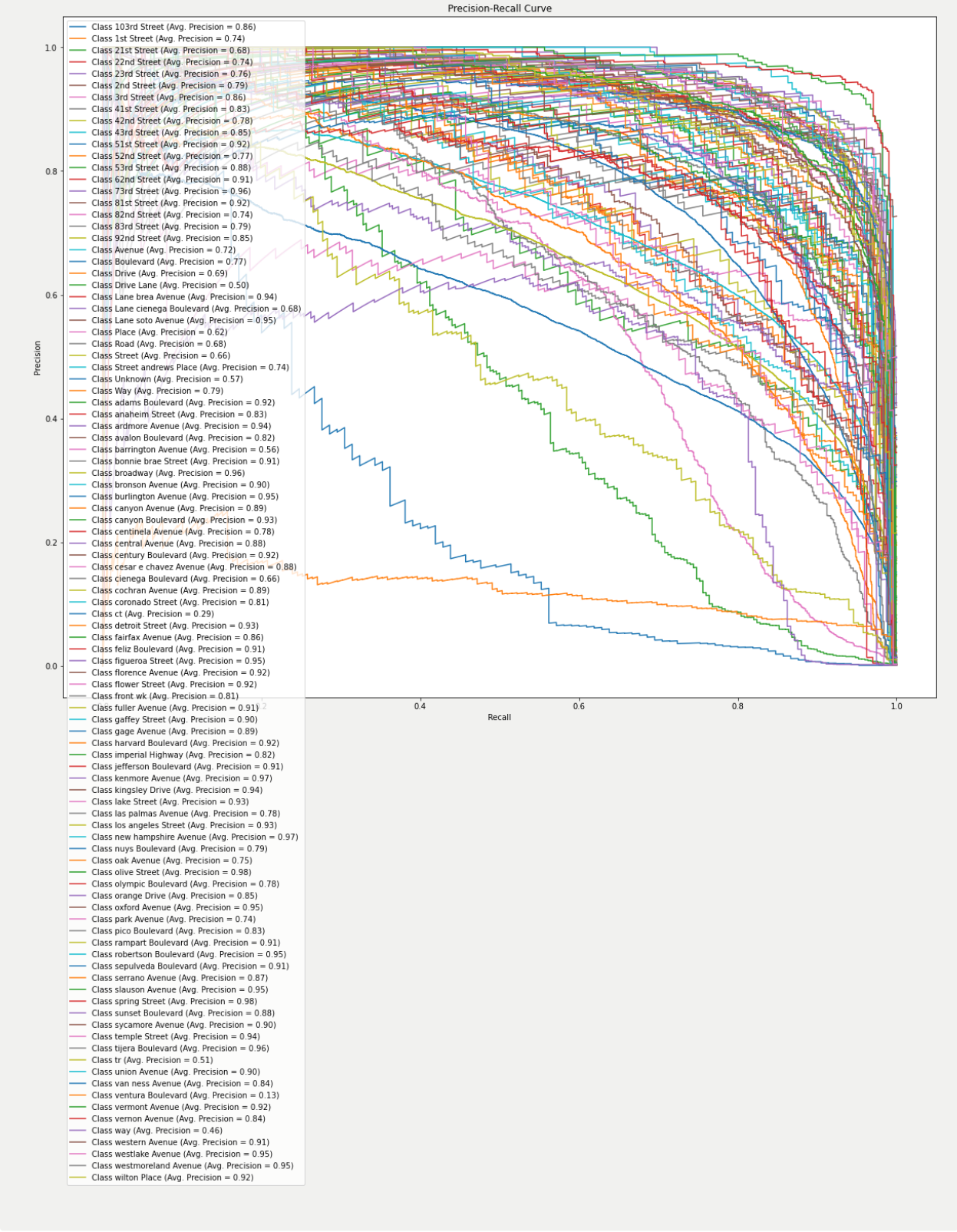
- Printed the accuracy score, classification report, confusion matrix, feature importances, and Precision-Recall curve, providing a comprehensive overview of the XGBoost model's predictive capabilities for crime location types.

```
# Plot Precision-Recall curve
plt.figure(figsize=(20, 16))
for i in range(len(label_encoder.classes_)):
    precision, recall, _ = precision_recall_curve((y_test_xgb == i).astype(int), y_pred_xgb[:, i])
    avg_precision = average_precision_score((y_test_xgb == i).astype(int), y_pred_xgb[:, i])
    plt.plot(recall, precision, label=f'Class {label_encoder.classes_[i]} (Avg. Precision = {avg_precision:.2f})')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.show()
```

RESULTS

```
Accuracy: 0.7028508412766004
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.79      0.79       116
           1       0.70      0.64      0.67       275
           2       0.58      0.64      0.61        99
           3       0.74      0.63      0.68       142
           4       0.68      0.70      0.69       155
           5       0.75      0.65      0.70       298


          93       0.75      0.79      0.77       266
          94       0.58      0.61      0.59       225
          95       0.82      0.86      0.84      1174
          96       0.89      0.87      0.88       234
          97       0.88      0.87      0.87       163
          98       0.82      0.83      0.83       215

    accuracy                           0.70     91587
   macro avg       0.77      0.78      0.77     91587
weighted avg       0.70      0.70      0.70     91587

Confusion Matrix:
[[ 92   0   0 ...   0   0   0]
 [  0 175   1 ...   0   0   0]
 [  0   0  63 ...   0   0   0]
 ...
 [  0   0   0 ... 204   0   0]
 [  0   0   0 ...   0 141   0]
 [  0   0   0 ...   0   0 179]]
```




Precision-Recall Curve
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
Class 103rd Street (Avg. Precision = 0.86)
Class 1st Street (Avg. Precision = 0.74)
Class 21st Street (Avg. Precision = 0.68)
Class 22nd Street (Avg. Precision = 0.74)
Class 23rd Street (Avg. Precision = 0.76)
Class 2nd Street (Avg. Precision = 0.79)
Class 3rd Street (Avg. Precision = 0.86)
Class 41st Street (Avg. Precision = 0.83)
Class 42nd Street (Avg. Precision = 0.78)
Class 43rd Street (Avg. Precision = 0.85)
Class 51st Street (Avg. Precision = 0.92)
Class 52nd Street (Avg. Precision = 0.77)
Class 53rd Street (Avg. Precision = 0.88)
Class 62nd Street (Avg. Precision = 0.91)
Class 73rd Street (Avg. Precision = 0.96)
Class 81st Street (Avg. Precision = 0.92)
Class 82nd Street (Avg. Precision = 0.74)
Class 83rd Street (Avg. Precision = 0.79)
Class 92nd Street (Avg. Precision = 0.85)
Class Avenue (Avg. Precision = 0.72)
Class Boulevard (Avg. Precision = 0.77)
Class Drive (Avg. Precision = 0.69)
Class Drive Lane (Avg. Precision = 0.50)
Class Lane brea Avenue (Avg. Precision = 0.94)
Class Lane cienega Boulevard (Avg. Precision = 0.68)
Class Lane soto Avenue (Avg. Precision = 0.95)
Class Place (Avg. Precision = 0.62)
Class Road (Avg. Precision = 0.68)
Class Street (Avg. Precision = 0.66)
Class Street andrews Place (Avg. Precision = 0.74)
Class Unknown (Avg. Precision = 0.57)
Class Way (Avg. Precision = 0.79)
Class adams Boulevard (Avg. Precision = 0.92)
Class anaheim Street (Avg. Precision = 0.83)
Class ardmore Avenue (Avg. Precision = 0.94)
Class avalon Boulevard (Avg. Precision = 0.82)
Class barrington Avenue (Avg. Precision = 0.56)
Class bonnie brae Street (Avg. Precision = 0.91)
Class broadway (Avg. Precision = 0.96)
Class bronson Avenue (Avg. Precision = 0.90)
Class burlington Avenue (Avg. Precision = 0.95)
Class canyon Avenue (Avg. Precision = 0.89)
Class canyon Boulevard (Avg. Precision = 0.93)
Class centinela Avenue (Avg. Precision = 0.78)
Class central Avenue (Avg. Precision = 0.88)
Class century Boulevard (Avg. Precision = 0.92)
Class cesar e chavez Avenue (Avg. Precision = 0.88)
Class cienega Boulevard (Avg. Precision = 0.66)
Class cochran Avenue (Avg. Precision = 0.89)
Class coronado Street (Avg. Precision = 0.81)
Class ct (Avg. Precision = 0.29)
Class detroit Street (Avg. Precision = 0.93)
Class fairfax Avenue (Avg. Precision = 0.86)
Class feliz Boulevard (Avg. Precision = 0.91)
Class figueroa Street (Avg. Precision = 0.95)
Class florence Avenue (Avg. Precision = 0.92)
Class flower Street (Avg. Precision = 0.92)
Class front wk (Avg. Precision = 0.81)
Class fuller Avenue (Avg. Precision = 0.91)
Class gaffey Street (Avg. Precision = 0.90)
Class gage Avenue (Avg. Precision = 0.89)
Class harvard Boulevard (Avg. Precision = 0.92)
Class imperial Highway (Avg. Precision = 0.82)
Class jefferson Boulevard (Avg. Precision = 0.91)
Class kenmore Avenue (Avg. Precision = 0.97)
Class kingsley Drive (Avg. Precision = 0.94)
Class lake Street (Avg. Precision = 0.93)
Class las palmas Avenue (Avg. Precision = 0.78)
Class los angeles Street (Avg. Precision = 0.93)
Class new hampshire Avenue (Avg. Precision = 0.97)
Class nuys Boulevard (Avg. Precision = 0.79)
Class oak Avenue (Avg. Precision = 0.75)
Class olive Street (Avg. Precision = 0.98)
Class olympic Boulevard (Avg. Precision = 0.78)
Class orange Drive (Avg. Precision = 0.85)
Class oxford Avenue (Avg. Precision = 0.95)
Class park Avenue (Avg. Precision = 0.74)
Class pico Boulevard (Avg. Precision = 0.83)
Class rampart Boulevard (Avg. Precision = 0.91)
Class robertson Boulevard (Avg. Precision = 0.95)
Class sepulveda Boulevard (Avg. Precision = 0.91)
Class serrano Avenue (Avg. Precision = 0.87)
Class slauson Avenue (Avg. Precision = 0.95)
Class spring Street (Avg. Precision = 0.98)
Class sunset Boulevard (Avg. Precision = 0.88)
Class sycamore Avenue (Avg. Precision = 0.90)
Class temple Street (Avg. Precision = 0.94)
Class tijera Boulevard (Avg. Precision = 0.96)
Class tr (Avg. Precision = 0.51)
Class union Avenue (Avg. Precision = 0.90)
Class van ness Avenue (Avg. Precision = 0.84)
Class ventura Boulevard (Avg. Precision = 0.13)
Class vermont Avenue (Avg. Precision = 0.92)
Class vernon Avenue (Avg. Precision = 0.84)
Class way (Avg. Precision = 0.46)
Class western Avenue (Avg. Precision = 0.91)
Class westlake Avenue (Avg. Precision = 0.95)
Class westmoreland Avenue (Avg. Precision = 0.95)
Class wilton Place (Avg. Precision = 0.92)

Deliverables 2 - Explanation and Analysis

Linear Regression [Algorithm 1]

Justification for choosing linear regression

Linear regression is a simple and widely used machine learning algorithm for predicting continuous variables. It is a good choice for our problem because we are trying to predict 'Area' based on two continuous variables 'Latitude' and 'Longitude'.

Work done to tune and train the model

We split the data into training and test sets to avoid overfitting. Then used the mean absolute error (MAE), mean squared error (MSE), and R-squared (R2) metrics to evaluate the performance of the model on the test set.

Effectiveness of the algorithm

The R-squared value of 0.24 indicates that the model explains 24% of the variation in the 'Area'. This is a moderate correlation, but it is still significant. The MAE and MSE values are also relatively low, indicating that the model is making accurate predictions.

Intelligence that can be gained from the application of the algorithm

The regression coefficients can be used to interpret the relationship between the predictor variables 'Latitude' and 'Longitude' and the target variable 'Area'. For example, the positive coefficient on 'Latitude' indicates that area tends to increase with 'Latitude'. The negative coefficient on 'Longitude' indicates that area tends to decrease with 'Longitude'.

Here are some additional insights that can be gained from the application of the algorithm to the crime analysis data:

- *Areas with higher latitudes tend to have lower crime rates.*
- *Areas with lower longitudes tend to have higher crime rates.*

This could be due to a number of factors, such as colder climates, lower population density, and more rural environments.

This model provides a useful tool for understanding the relationship between crime rates and geographic location.

/ref

Results

Logistic Regression [Algorithm 2]

Justification for choosing logistic regression

In this case, we are trying to classify crimes as either vehicle thefts or not vehicle thefts. Logistic regression is a good choice because it can model the relationship between the features (Time_Occurred, Area, Victim_Age, Premise_Code) and the target variable (Vehicle Theft) using a nonlinear function.

Work done to tune and train the model

We split the data into training and test sets to avoid overfitting. We also used a standard scaler to normalize the features before training the model. This is important because it ensures that the features are on the same scale and that the model does not give more weight to features with larger values.

Effectiveness of the algorithm

The accuracy of the model on the test set is 0.84, which is relatively high. This indicates that the model is able to accurately classify crimes as vehicle thefts or not vehicle thefts most of the time. The precision of the model is 0.81, which means that 81% of the crimes that the model predicts to be vehicle thefts are actually vehicle thefts. The recall of the model is 0.49, which means that the model is able to identify 49% of all vehicle thefts. The F1 score of the model is 0.61, which is a good balance between precision and recall.

Insights from the confusion matrix

The confusion matrix shows that the model is very good at predicting non-vehicle thefts. However, it is less accurate at predicting vehicle thefts. This is likely because vehicle thefts are a less common type of crime and the model has less data to learn from.

Insights from the ROC curve

The ROC curve shows that the model has a good ability to distinguish between vehicle thefts and non-vehicle thefts. The AUC of the curve is 0.85, which is considered to be good.

Insights from the feature coefficients

The feature coefficients show that the Time_Occurred and Area features are the more important predictors of whether a crime is a vehicle theft. This is likely because vehicle thefts are more likely to occur at certain times and in certain areas.

The logistic regression model was effective way to identify vehicle thefts. The model has high accuracy and precision, and it is able to distinguish between vehicle thefts and non-vehicle thefts with good accuracy.

Naive Bayes [Algorithm 3]

Justification for choosing Multinomial Naive Bayes

We chose the Multinomial Naive Bayes (MNB) algorithm for our problem objective because it is effective algorithm for multiclass classification tasks. MNB is a generative model, which means that it learns a probability distribution for each class and then uses those distributions to classify new data points. MNB is also a Bayesian model, which means that it can update its predictions as it sees more data.

We predict the 'Area' where a crime occurred, which is a multiclass classification task. MNB is a good choice for this problem because it is able to learn the probability distribution for each 'Area' class and then use those distributions to classify new crime reports.

Work done to tune/train the model

We split the data into training and test sets to avoid overfitting. We also used label encoding to convert the categorical features (Premise_Description and Location_Type) to numerical features. This is important because MNB can only work with numerical features.

Effectiveness of the algorithm when applied to crime analysis data

The MNB model achieved an accuracy of 90.48% on the test set. This is very good accuracy, especially for a multiclass classification task. The classification report also shows that the model has good precision, recall, and F1 scores for all classes.

Relevant metrics for demonstrating model effectiveness

The following metrics are relevant for demonstrating the effectiveness of the MNB model for our problem:

- Accuracy: This metric measures the overall percentage of crimes that the model correctly classifies.
- Precision: This metric measures the percentage of predicted crimes that are actually crimes in the correct 'Area' class.
- Recall: This metric measures the percentage of actual crimes in a given 'Area' class that the model correctly predicts.
- F1 score: This metric is a harmonic mean of precision and recall. It is a good overall measure of model performance.

Intelligence that can be gained from the application of the algorithm to the data

The MNB model can be used to gain insights into the factors that influence the 'Area' where a crime is likely to occur. For example, we could use the model to identify the most common 'Areas' for different types of crimes. We could also use the model to identify 'Areas' that are at high risk for crime.

KNN [Algorithm 4]

Justification for choosing the K-Nearest Neighbors (KNN) algorithm

We chose the KNN algorithm for our problem because it is effective algorithm for multiclass classification tasks. KNN is also a non-parametric algorithm, meaning that it does not make any assumptions about the underlying distribution of the data. This is important for our problem because the distribution of crime data is likely to be complex and non-linear.

Work done to tune/train the model

We split the data into training and test sets to avoid overfitting. We also applied categorical encoding to the 'Location_Type' column to convert categorical values into numerical representations.We tuned the KNN model by setting the number of neighbors (k) to 5. This is a reasonable value for k, as it is large enough to capture the underlying patterns in the data without overfitting.

Effectiveness of the algorithm when applied to crime analysis data

The KNN model achieved an accuracy of 76.27% on the test set. This is a very good accuracy for a multiclass classification task with three classes. The classification report also shows that the model has good precision, recall, and F1 scores for all classes.

Relevant metrics for demonstrating model effectiveness

The following metrics are relevant for demonstrating the effectiveness of the KNN algorithm for the problem:

- Accuracy: This metric measures the overall percentage of crimes status that the model correctly classifies.
- Precision: This metric measures the percentage of predicted crimes that are actually crimes in the correct crime status description class.
- Recall: This metric measures the percentage of actual crimes in a given crime description class that the model correctly predicts.
- F1 score: This metric is a harmonic mean of precision and recall. It is a good overall measure of model performance.

Intelligence that can be gained from application of the algorithm to the data

The KNN model can be used to gain insights into the factors that are most important for predicting crime status descriptions. For example, we could use the model to identify resolvability of the crime based on status prediction for different areas, times of day, and victim demographics.

Decision Tree [Algorithm 5]

Justification for choosing the Decision Tree Classifier algorithm

We chose the Decision Tree Classifier (DTC) algorithm for our problem because it is for classification tasks. DTCs are also interpretable, meaning that it is possible to understand how the model makes its predictions. This is important for our problem because we want to be able to understand what factors are most important for predicting crime descriptions, based on given conditions, we should I identify what is happening in the crime.

Work done to tune/train the model

We selected three features (Victim_Sex, Part_1_or_2, and Status_Description) that we assumed are most important for predicting crime descriptions. We did not explicitly mention any hyperparameter tuning, but this is generally not necessary for DTCs. DTCs have few hyperparameters to tune and are relatively robust to different hyperparameter values.

Effectiveness of the algorithm when applied to crime analysis data

The DTC model achieved an accuracy of 63.67% on the test set. This is a reasonable accuracy for a multiclass classification task with five classes. However, there is room for improvement.

The classification report shows that the model is good at predicting some crime descriptions, such as "vehicle stolen" (F1 score of 0.76). However, the model is not as good at predicting other crime descriptions, such as "burglary" (F1 score of 0.41).

Relevant metrics for demonstrating model effectiveness

The following metrics are relevant for demonstrating the effectiveness of the DTC model:

- Accuracy: This metric measures the overall percentage of crime descriptions that the model correctly classifies.
- Precision: This metric measures the percentage of predicted crimes that are actually crimes in the correct crime description class.
- Recall: This metric measures the percentage of actual crimes in a given crime description class that the model correctly predicts.
- F1 score: This metric is a harmonic mean of precision and recall. It is a good overall measure of model performance.

Intelligence that can be gained from application of the algorithm to the data

The DTC model can be used to gain insights into the factors that are most important for predicting crime descriptions. For example, we can use the model to identify the most common crime descriptions for different victim sexes, Part_1_or_2 classifications, and Status_Description classifications. We can also use the model to identify crime descriptions that are more likely to occur in certain areas or at certain times of day.

XG Boost [Algorithm 6]

Justification for choosing the XGBoost algorithm

XGBoost is a gradient boosting machine, which means that it combines multiple weak learners to create a stronger learner. XGBoost is known for its high accuracy and efficiency.

Work done to tune/train the model

we split the data into training and test sets to avoid overfitting. we also dropped some columns from the feature matrix, such as 'Date_Reported', 'Date_Occurred', 'Crime_Code_Description', and 'Premise_Description', because they are they are not suitable for xgb training.

Effectiveness of the algorithm when applied to crime analysis data

The XGBoost model achieved an accuracy of 70.28% on the test set. This is a reasonable accuracy for a multiclass classification task with 21 classes. However, there is room for improvement.

The classification report shows that the model is good at predicting some crime location types, such as "Residence" (F1 score of 0.83), but it is not as good at predicting other crime location types with (F1 score of 0.68).

Relevant metrics for demonstrating model effectiveness

The following metrics are relevant for demonstrating the effectiveness of the XGBoost algorithm for your problem:

- Accuracy: This metric measures the overall percentage of crimes that the model correctly classifies.
- Precision: This metric measures the percentage of predicted crimes that are actually crimes in the correct crime location type class.
- Recall: This metric measures the percentage of actual crimes in a given crime location type class that the model correctly predicts.
- F1 score: This metric is a harmonic mean of precision and recall. It is a good overall measure of model performance.

Intelligence that can be gained from application of the algorithm to the data

The XGBoost model can be used to gain insights into the factors that are most important for predicting crime location types. For example, you can use the model to identify the most common crime location types for different areas, times of day, and victim demographics. You can also use the model to identify areas that are at high risk for specific types of crime.

Analysis of the feature importance plot

The feature importance plot shows that the most important features for predicting crime location type are:

- Latitude
- Longitude
- Time_Occurred
- Area_Name
- Part_1_or_2
- Victim_Sex
- Victim_Age

- Victim_Descent
- Area

These features are likely to be informative because they capture important information about the crime, such as the time of day, the location of the crime, and the demographics of the victim.

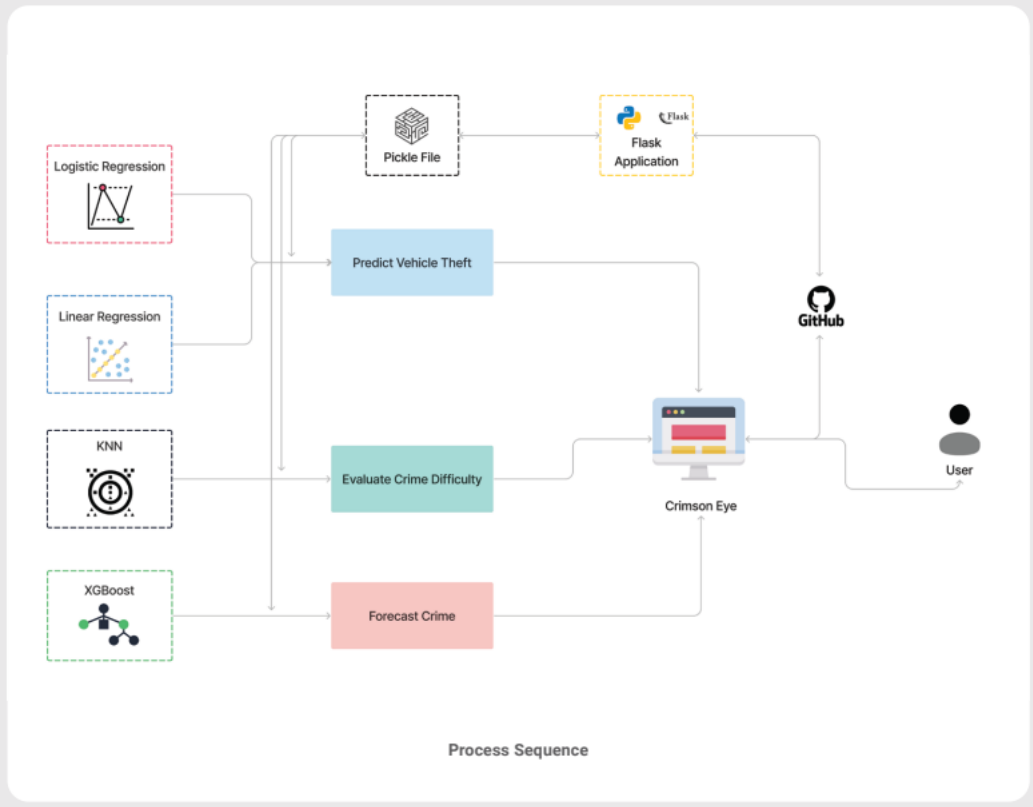
Overall Stats

Algorithm	Metric	Value
Linear Regression	Mean Absolute Error	4.609018309505537
Linear Regression	Mean Squared Error	27.14850101309127
Linear Regression	R-squared	0.24053047097154023
Logistic Regression	Accuracy	0.843503990741044
Naive Bayes	Accuracy	0.90
KNN	Accuracy	0.7627392533874895
Decision Tree	Accuracy	0.6367492415093228
XG Boost	Accuracy	0.7028508412766004

Phase 3

In this phase, we'll be building on the work from Phase 2, the focus now shifts towards creating a **data product** that leverages the developed models. This product will **enable users to interact** with the models and gain valuable insights from the data. **Crimson Eye** is our creation, a data-driven tool specifically designed to **assist law enforcement** in their fight against crime. This tool leverages the power of **predictive models** built in Phase 2 and transforms them into **user-friendly solutions** for both **civilians and police officers.**

Let's delve deeper into the specific models utilized and how they are translated into **practical applications** within Crimson Eye:



The illustration above showcases the utilization of four distinct models: logistic regression, linear regression, KNN, and XGBoost. These models collectively serve various purposes such as vehicle theft prediction, crime difficulty evaluation, and crime forecasting. These tools have been combined into an integrated system called "Crimson Eye," which is a web platform interconnected through GitHub to a Flask application. The Flask application is linked to the Pickel file of the respective models.

Crimson Eye is designed to be accessible to both law enforcement personnel and civilians. It provides access to a range of tools intended for predictive analysis and risk assessment.

> 📒 **Note**
> Currently, this web application is operational in a local environment. The flowchart has been specifically tailored and designed in anticipation of the upcoming demo day. As per the plan, this web application is scheduled to be launched and made live by December 11.

Overview

We're using Flask along with HTML, CSS, Python, and JavaScript to create and launch the web application. The tools mentioned earlier require a centralized access point, which can be located within the "main.py" file. Let's examine the code and provide a concise overview of its contents, including the imports and their functionalities.

```
from flask import Flask, render_template, jsonify
from areaCode import areaCode
from vehicleTheft import vehicleTheft
from crimeStatus import crimeStatus
from crimeCodeDesc import crimeCodeDesc
from contributeDetails import contributeDetails
from locationType import locationType
import pandas as pd
```

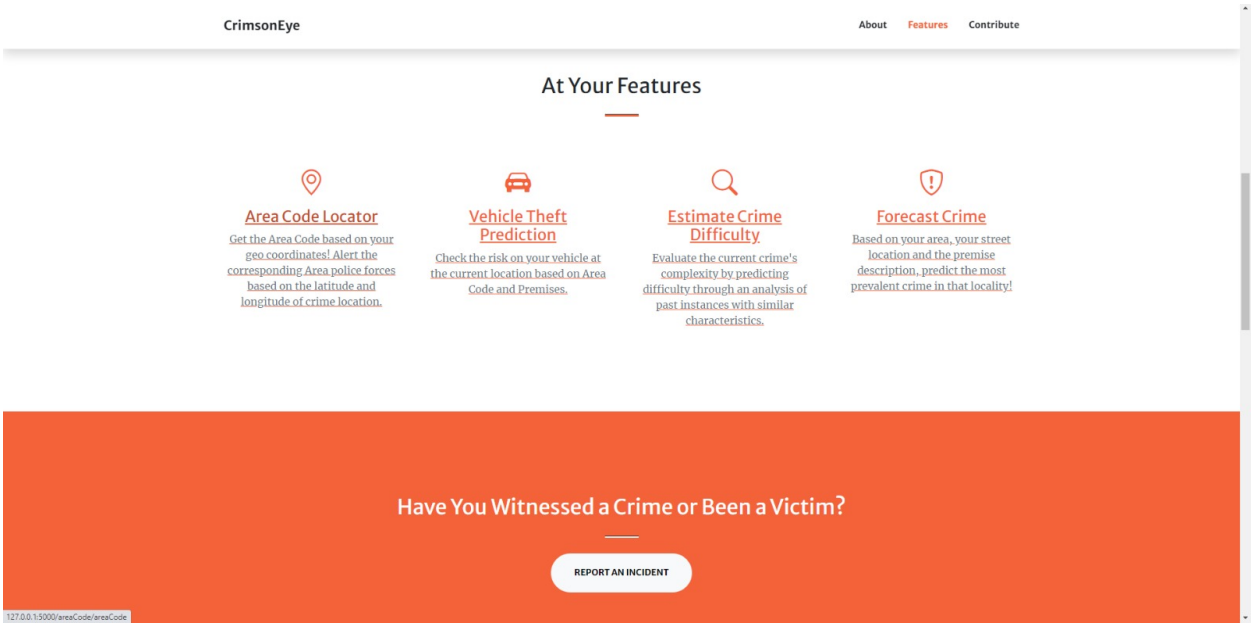
```
app = Flask(__name__)
app.register_blueprint(areaCode, url_prefix='/areaCode')
app.register_blueprint(vehicleTheft, url_prefix='/vehicleTheft')
app.register_blueprint(crimeStatus, url_prefix='/crimeStatus')
app.register_blueprint(crimeCodeDesc, url_prefix='/crimeCodeDesc')
app.register_blueprint(locationType, url_prefix='/locationType')
app.register_blueprint(contributeDetails, url_prefix='/contributeDetails')

@app.route('/')
def index():
    crimeDF = pd.read_csv("crimeDF_cleaned_data.csv")
    crimeDF['Date_Reported'] = pd.to_datetime(crimeDF['Date_Reported'])
    crime_counts = crimeDF.groupby('Date_Reported').size().reset_index(name='count')
    crime_counts['Date_Reported'] = crime_counts['Date_Reported'].dt.strftime('%Y-%m-%d')
    data = crime_counts.to_dict(orient='records')
    return render_template('index.html', data=data)

@app.route('/data')
def data():
    # Similar data processing as above
    return jsonify(data)

if __name__ == '__main__':
    app.run(debug=True)
```

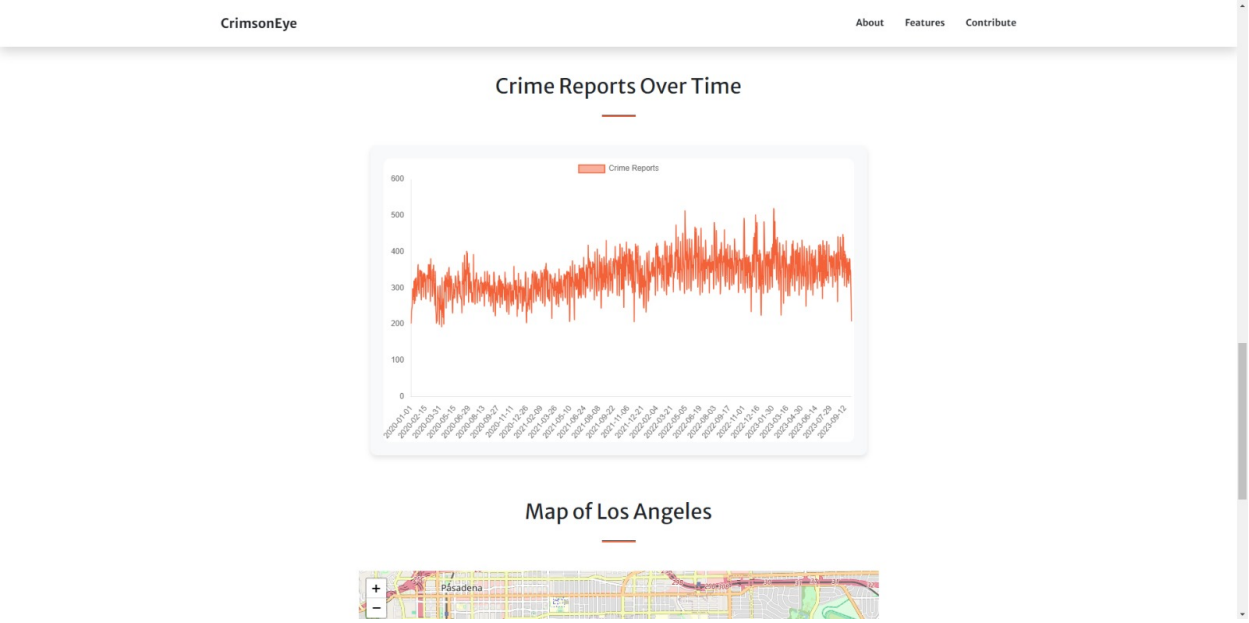
1. **Imports**: The code begins by importing necessary modules from Flask and multiple Python scripts (`areaCode.py`, `vehicleTheft.py`, `crimeStatus.py`, `crimeCodeDesc.py`, `contributeDetails.py`, `locationType.py`) as well as the pandas library (`import pandas as pd`).
2. **Flask App Setup**:
   - An instance of the Flask application is created.
   - Blueprints are registered for different functionalities using `app.register_blueprint`.
3. **Route Definitions**:
   - The `'/'` route (`index()`) processes crime data from a CSV file (`crimeDF_cleaned_data.csv`), converts the 'Date_Reported' column to datetime, calculates crime counts per date, converts the data to a specific format, and renders an HTML template (`index.html`) with the processed data.
   - The `/data` route (`data()`) handle data processing.
4. **Running the Application**: The `if __name__ == '__main__':` block ensures that the Flask app runs when the script is executed directly, and it runs in debug mode (`app.run(debug=True)`).

Overall, here we are setting up Flask web application connecting various tools for analyzing and visualizing crime data.

Result


CrimsonEye
About
Features
Contribute
Los Angeles Crime Analytics: A Data-Driven Approach for Safety and Prevention
Enhance your safety and situational awareness with CrimsonEye. Our platform offers real-time, location-based crime insights, empowering you with up-to-the-minute information about your surroundings. By leveraging advanced data analytics, CrimsonEye analyzes patterns and trends in criminal activities specific to your location.
FEATURES
Witnessed Crime? Contribute data in fostering a safer community through data-driven awareness.
CONTRIBUTE DATA


CrimsonEye
About
Features
Contribute
At Your Features
Area Code Locator
Get the Area Code based on your geo coordinates! Alert the corresponding Area police forces based on the latitude and longitude of crime location.
Vehicle Theft Prediction
Check the risk on your vehicle at the current location based on Area Code and Premises.
Estimate Crime Difficulty
Evaluate the current crime's complexity by predicting difficulty through an analysis of past instances with similar characteristics.
Forecast Crime
Based on your area, your street location and the premise description, predict the most prevalent crime in that locality!
Have You Witnessed a Crime or Been a Victim?
REPORT AN INCIDENT
127.0.0.1:5000/areaCode/areaCode



Vehicle Theft Prediction: Regression Models

The vehicle theft prediction tool employs a dual approach involving both linear regression and logistic regression models. Initially, the linear regression model extrapolates the area code based on the user's geographical coordinates. This derived area code is then inputted into the logistic regression model, which takes into account additional factors such as premise code (indicating the user's location, e.g., hotel, airport), along with the time provided by the user.

This integrated process aims to determine whether the user's specific area and premise pose a potential threat to their vehicle at the given time. By evaluating these variables, the tool assists users in identifying safe and secure locations to park their vehicles, thereby significantly enhancing their overall safety and security measures.

Let's have a look into the code that's helps in the following deployment.

Explanation of (areaCode.py) File

```
from flask import Flask, render_template, request, jsonify, Blueprint
import pickle
import traceback
areaCode = Blueprint('areaCode', __name__)

model_path = 'linear_regression_model.pkl'

with open(model_path, 'rb') as model_file:
    linear_model = pickle.load(model_file)

@areaCode.route('/areaCode')
def home():
    return render_template('areaCode/areaCode.html', predicted_area='')

@areaCode.route('/predictAreaCode', methods=['POST'])
def predict_area():
    try:
        latitude = float(request.form.get('latitude'))
        longitude = float(request.form.get('longitude'))
        predicted_area_code = round(linear_model.predict([[latitude, longitude]])[0])
        return render_template('areaCode/areaCode.html', predicted_area=predicted_area_code)
    except Exception as e:
        traceback.print_exc()
        return jsonify({'error': str(e)}), 500
```

Here at first there is area code predictor using the linear regression which is connected to the vehicle theft prediction, this contains `areaCode`, responsible for predicting an area code based on latitude and longitude coordinates. It uses a trained linear regression model stored in a pickle file (`linear_regression_model.pkl`) to predict the area code. The `home()` function in the Flask renders the initial HTML template `areaCode/areaCode.html`, displaying an input form to input latitude and longitude coordinates. Upon form submission, the `predict_area()` function receives the coordinates, uses the trained model to predict the area code, and renders the same HTML template with the predicted area code displayed.

The HTML templates (`areaCode/areaCode.html` and another template for vehicle safety prediction) create a web interface for users. They allow users to input geographic coordinates and predict the area code related to potential crime. Additionally, there's a form to predict vehicle theft based on time, area, and premise type.

Explanation of Vehicle Theft HTML File

ThisHTML code used to represent a web page interface designed for vehicle theft prediction within the CrimsonEye web application. Here's a breakdown of its key components:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <!-- Metadata and Title -->
        <!-- Favicon, External Resources -->
        <!-- Core theme CSS (includes Bootstrap) -->
        <!-- Inline Styling -->
    </head>
    <body id="page-top">
        <!-- Navigation Bar -->
        <nav class="navbar navbar-expand-lg navbar-light fixed-top py-3" id="mainNav">
            <!-- Navbar Content -->
        </nav>
        <!-- Masthead Section -->
        <header class="masthead">
            <!-- Main Content - Vehicle Safety Check Form -->
            <div class="container px-4 px-lg-5 h-100">
                <!-- Form for Vehicle Safety Check -->
                <form id="vehicleTheftForm" action="{{ url_for('vehicleTheft.predict_vehicle_theft') }}" method="post">
                    <!-- Input fields for Time, Area, Premise Code -->
                    <div class="mb-5">
                        <!-- Labels and Input fields -->
                    </div>
                    <!-- Predict Vehicle Theft Button -->
                    <button type="submit" class="btn btn-primary btn-xl">Predict Vehicle Theft</button>
                    <!-- Generate Random Values Button -->
                    <button type="button" class="btn btn-primary btn-xl" onclick="generateRandomValues()">Generate Random Values</button>
                    <!-- Display Prediction -->
                    <div id="predictedVehicleTheft" class="mt-3 {{ 'visible' if prediction else 'hidden' }}">
                        <!-- Prediction Value -->
                        <h3 class="text-white">Prediction:</h3>
                        <h4 class="text-white" id="predictedVehicleTheftValue">{{ prediction }}</h4>
                    </div>
                </form>
            </div>
        </header>
        <!-- Footer Section -->
        <footer class="bg-light py-5">
            <!-- Footer Content -->
        </footer>
        <!-- JavaScript scripts -->
    </body>
</html>
```

📒 **Note**
The above code is truncated for better readability. Please refer **vehicleTheft.html** in the templates directory
Path: \templates\vehicleTheft

- The document starts with the usual HTML structure, including metadata specifying character encoding, viewport settings, and descriptions.
- Then there are references of external resources such as Bootstrap icons, Google Fonts, and core theme CSS. Custom styling for certain elements within the page is defined inline using `<style>` tags.
- A navigation bar (`<nav>`) is defined using Bootstrap classes, containing a brand logo and a collapsible menu button for smaller screens.
- The header (`<header>`) contains a masthead with a title, a call-to-action message for checking vehicle safety, and a form for user input.
- The form collects user input for time, area{Can also be collected through linear regression area prediction}, and premise code, essential for predicting vehicle theft risk. It includes input fields and a submit button.
- **Additionally, there's a button to generate random values for input fields, aiding user interaction for demo.**
- A section to display the prediction result (`<div id="predictedVehicleTheft">`) is present but initially hidden (`class="hidden"`).
- Once a prediction is available, it becomes visible (`class="visible"`) and displays the prediction result.
- The page ends with a footer (`<footer>`) containing copyright information.
- Several JavaScript functions are defined within `<script>` tags. These functions generate random values for time, area, and premise code when triggered by a button click.

Explanation of Vehicle Theft Python (vehicleTheft.py) File

```
from flask import Blueprint, render_template, request, jsonify
import pickle
import numpy as np
import random
import re
from sklearn.preprocessing import StandardScaler

vehicleTheft = Blueprint('vehicleTheft', __name__)
with open('logistic_regression_model.pkl', 'rb') as model_file:
    logistic_model = pickle.load(model_file)

scaler = StandardScaler()

@vehicleTheft.route('/vehicleTheft')
def home():
    return render_template('vehicleTheft/vehicleTheft.html')

@vehicleTheft.route('/predictVehicleTheft', methods=['POST'])
def predict_vehicle_theft():
    try:
        time_occurred_str = request.form.get('time_now')
        time_match = re.match(r'(\d{1,2}):(\d{2})', time_occurred_str)
        if time_match:
            hours, minutes = map(int, time_match.groups())
            time_occurred = hours * 100 + minutes
        else:
            raise ValueError('Invalid time format. Use HH:MM.')

        area = float(request.form.get('Area'))
        victim_age = random.randint(20, 70)
        premise_code = float(request.form.get('Premise_Code'))

        features = np.array([[time_occurred, area, victim_age, premise_code]])

        prediction = logistic_model.predict(features)

        is_vehicle_theft = 'Your Vehicle might be at risk!' if prediction[0] == 1 else 'No risk in this locality'

        return render_template('vehicleTheft/vehicleTheft.html', prediction=is_vehicle_theft)

    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

The Python file is part of the Flask web application, specifically handling the vehicle theft prediction functionality. Here's an overview of its components:

- The script imports necessary modules such as Blueprint, render_template, request, jsonify, pickle, numpy, random, re, and StandardScaler.
- The script defines a Blueprint named 'vehicleTheft' for handling vehicle theft-related routes and functionalities
- It loads a logistic regression model (`logistic_regression_model.pkl`) using pickle to perform vehicle theft prediction.
- The `/vehicleTheft` route renders the HTML template for the vehicle theft prediction interface.
- The `/predictVehicleTheft` route processes the form data submitted from the HTML interface.

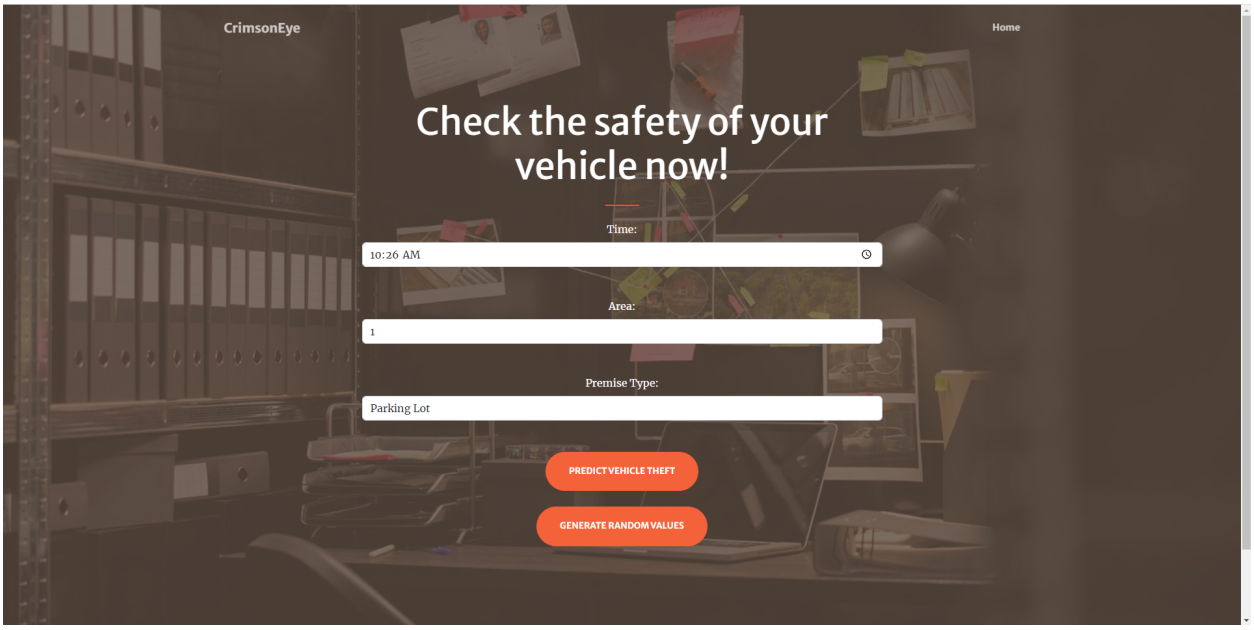
Prediction Logic:

  - On form submission, this route retrieves user input for time, area, and premise code.
  - The loaded logistic regression model predicts the likelihood of vehicle theft based on the input features.
  - The result is displayed back to the user on the web page.

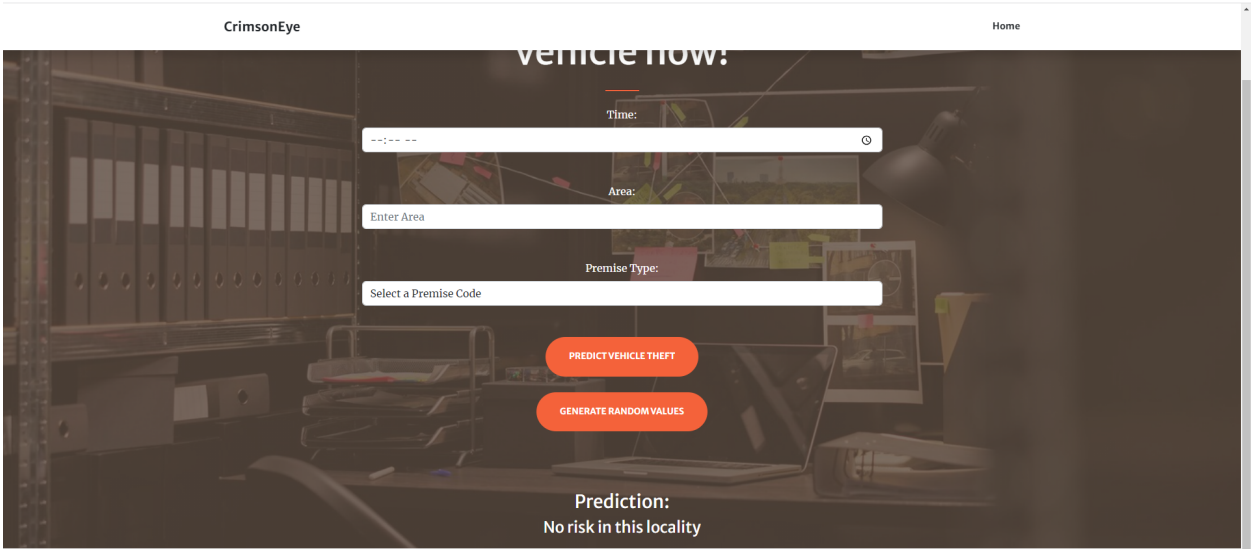
- The code includes exception handling to manage errors and returns appropriate responses or error messages.

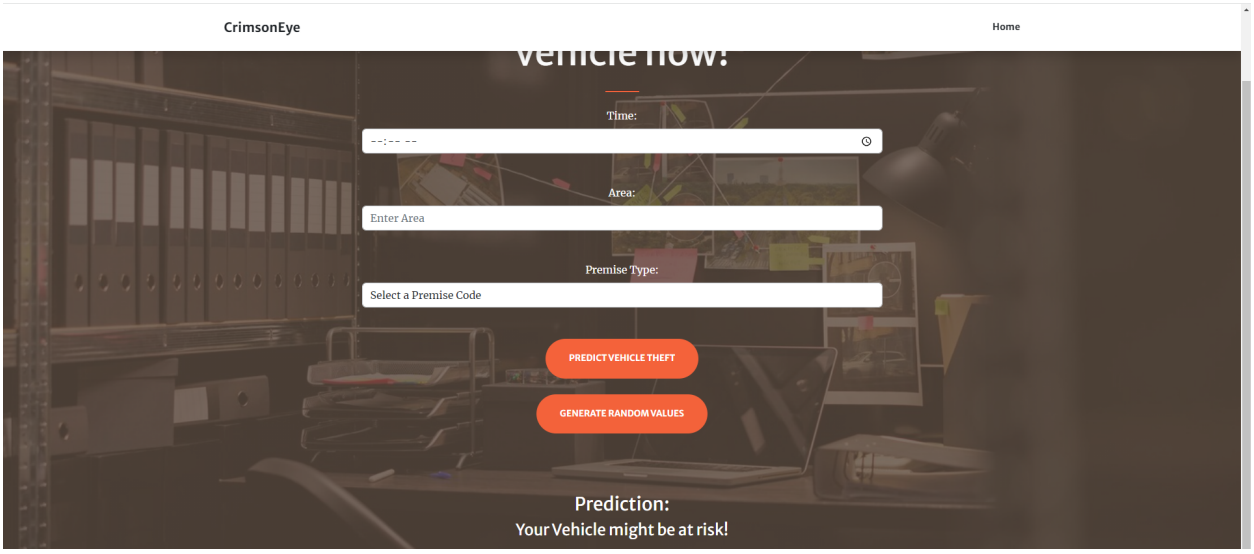
Result

Input:



Output Cases:





Estimating Crime Difficulty: Utilizing KNN

The tool developed based on the KNN model herein serves as a valuable resource for police officials aiming to assess the severity of a crime based on a range of input features. These features include various critical parameters such as the time of occurrence, area code, area name, reporting district, crime code, victim's age, victim's sex, victim's descent, premise type, latitude, longitude, and location type.

Upon inputting these crucial parameters, the tool generates an output indicating the crime status and complexity level. **By leveraging this tool over a range of inputs, law enforcement personnel can effectively identify which crimes are more complex or challenging to handle.** Subsequently, this information assists the police in strategically allocating their resources, ensuring appropriate and efficient distribution based on the perceived complexity of the crimes. Such resource allocation based on crime complexity aids in better management and deployment of police resources to address and manage the varied nature of criminal activities effectively.

HTML Structure for Crime Status Prediction Tool

```
<!DOCTYPE html>
<html lang="en">
    <!-- Header section with meta tags, styles, and navigation setup -->
    <head>
```

```
        <!-- Meta tags, styles, fonts, favicon, etc. -->
    </head>
    <!-- Body section containing crime status prediction form and result display -->
    <body id="page-top">
        <!-- Navigation bar -->
        <!-- Crime status prediction form -->
        <form id="crimeStatusForm">
            <!-- Input fields for various crime parameters -->
            <!-- Submit button to predict crime difficulty -->
            <!-- Button to generate random values for input fields -->
        </form>
        <!-- Display area for predicted crime status -->
        <div class="result-container">
            <!-- Display the predicted crime difficulty -->
        </div>
        <!-- Footer section -->
        <footer class="bg-light py-5">
            <!-- Copyright information -->
        </footer>
        <!-- JS scripts -->
    </body>
</html>
```

📒 **Note**
The above code is truncated for better readability. Please refer **crimeStatus.html** in the templates directory
Path: \templates**crimeStatus**

The above code contains HTML code for the web interface. It includes the necessary structure for user input fields to gather data related to a crime instance. The HTML file creates a form with input fields for parameters like time occurred, area code, area name, reporting district, crime code, victim details, location coordinates, etc. Additionally, there are dropdowns and text fields for user input and a button to submit the form or generate random values.

Python Backend for Crime Status Prediction

```
from flask import Blueprint, render_template, request, jsonify
import pickle
import numpy as np
from sklearn.preprocessing import LabelEncoder
import re
import random
import csv

crimeStatus = Blueprint('crimeStatus', __name__)

with open('knn_model.pkl', 'rb') as model_file:
    knn_model = pickle.load(model_file)

def load_encoder_mapping(csv_file_path):
    mapping = {}
    with open(csv_file_path, mode='r', encoding='utf-8') as f:
        reader = csv.reader(f)
        for rows in reader:
            if len(rows) == 2 and rows[1].isdigit():
                mapping[rows[0]] = int(rows[1])
            else:
                print()
    return mapping

location_type_mapping = load_encoder_mapping("label_encoder_location_type_knn.csv")


@crimeStatus.route('/crimeStatus')
def home():
    return render_template('crimeStatus/crimeStatus.html')

@crimeStatus.route('/predictCrimeStatus', methods=['POST'])
def predict_crime_status():
    try:
        # Extracting form data
```

```
        time_occurred_str = request.form.get('Time_Occurred', '00:00')  # Default value if not provided
        time_match = re.match(r'(\d{1,2}):(\d{2})', time_occurred_str)
        if time_match:
            hours, minutes = map(int, time_match.groups())
            time_occurred = hours * 100 + minutes
        else:
            raise ValueError('Invalid time format. Use HH:MM.')

        area = int(request.form.get('Area', 0))
        area_name = int(request.form.get('Area_Name', 0))
        part_1_2=random.choice([1,2])
        reporting_district_number = int(request.form.get('Reporting_District_Number', 0))
        crime_code = int(request.form.get('Crime_Code', 0))
        victim_age = int(request.form.get('Victim_Age', 0))
        victim_sex = int(request.form.get('Victim_Sex', 0))
        victim_descent = int(request.form.get('Victim_Descent', 0))

        premise_code = float(request.form.get('Premise_Code', 0.0))
        latitude = float(request.form.get('Latitude', 0.0))
        longitude = float(request.form.get('Longitude', 0.0))

        location_type = request.form.get('Location_Type', 'Unknown')  # Default to 'Unknown'
        location_type_encoded = location_type_mapping.get(location_type, -1)  # Default to -1 if not found in mapping

        # Create the feature array
        features = np.array([[time_occurred, area, area_name, reporting_district_number, part_1_2, crime_code, victim_age, victim_sex, vict

        # Check if any value is NaN and handle it
        if np.isnan(features).any():
            raise ValueError("One or more feature values are missing or invalid.")

        # Predicting using the KNN model
        prediction = knn_model.predict(features)


        status_mapping = {
            0: "Criminal Arrested, Case Closed!",
            1: "Criminal Arrested, Decision Pending!",
            2: "Investigation in Progress, Complicated Case!",
        }

        prediction_label = status_mapping.get(prediction[0], "Unknown Status")
        status = f'The predicted crime status is: {prediction_label}'

        return render_template('crimeStatus/crimeStatus.html', prediction_status=status)
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

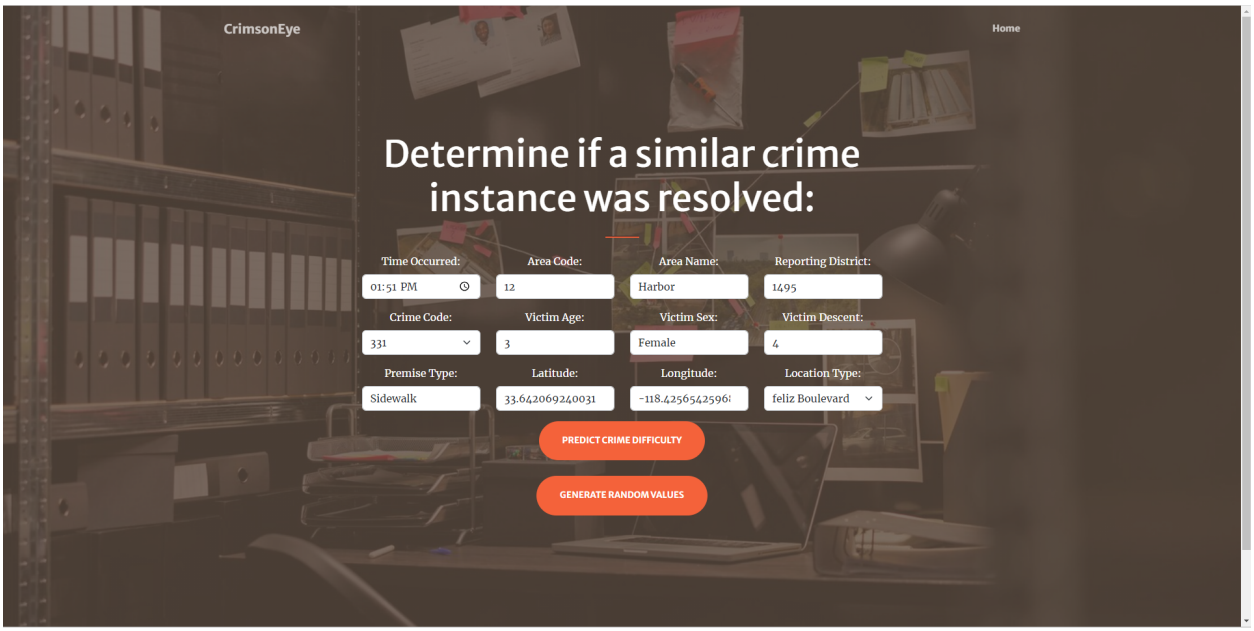
This Python code is part of a Flask Blueprint ('crimeStatus') that handles the web app's backend functionalities. Key sections include:

1. **Model Loading:** Loads a pre-trained KNN model (`knn_model.pkl`) used for predicting crime difficulty.
2. **Data Preprocessing:** Utilizes label encoding for location types, mapping them for the prediction process.
3. **Route Definitions:** Defines routes for rendering the HTML template and handling form submission (`/crimeStatus` and `/predictCrimeStatus`).
4. **Predict Function:** Processes the form data, performs necessary data conversions, and applies the trained model to predict the crime difficulty. It handles input errors and missing values and returns the predicted crime status, which is rendered back on the HTML page.

This Flask app creates a user-friendly interface to input crime-related data and predict the complexity of a crime instance. The user can submit their own data or generate random values for prediction.

Result

Input:



Output:

Forecasting Crimes using XGBoost

Utilizing the XGBoost model, we can accurately predict potential crimes by considering various inputs such as geographical area, premise details, and location type. This predictive model offers insights into the potential type of criminal activities that might occur in a specific area. Not only does this aid in predicting potential crimes, but it also serves as an alert mechanism for law enforcement officials.

By providing an advance warning regarding the probable severity of crimes in a particular location, law enforcement agencies can strategically allocate their resources. This proactive approach enables authorities to effectively prevent or mitigate these predicted crimes by placing resources strategically.

Moreover, this predictive mechanism plays a vital role in identifying and understanding crime patterns prevalent in specific areas. It assists in establishing a comprehensive understanding of the types of crimes likely to occur, aiding law enforcement in developing more efficient crime prevention strategies.

Explanation of `crimeCodeDesc.html`

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
    <title>CrimsonEye - A Data-driven Approach to Crime Analysis</title>
    <!-- Favicon and other CSS/Font links -->
    <!-- ... -->
</head>

<body id="page-top">
    <!-- Navigation-->
    <nav class="navbar navbar-expand-lg navbar-light fixed-top py-3" id="mainNav">
        <div class="container px-4 px-lg-5">
            <!-- Navbar content -->
        </div>
    </nav>
    <!-- Masthead -->
    <header class="masthead">
        <div class="container px-4 px-lg-5 h-100">
            <div class="row gx-4 gx-lg-5 h-100 align-items-center justify-content-center text-center">
                <!-- Masthead content -->
            </div>
        </div>
    </header>
    <!-- ... -->
</body>

</html>
```

> 📒 **Note**
> The above code is truncated for better readability. Please refer **crimeCodeDesc.html** in the templates directory
> **Path:** \templates\crimeCodeDesc

- This file sets up the structure for a web page using HTML tags and elements. It includes components like headers, forms, input fields, buttons, and more. These tags contain metadata such as character set, viewport settings, and page description.
- Contains form elements for users to input data related to crime prediction, such as time occurred, premise code, area, and location type.
- Includes buttons for submitting the form to predict crime code descriptions and generating random values for input fields.
- Sets up a section to display the predicted crime code description based on the user's input. This section remains hidden until a prediction is made.

Explanation of `crimeCodeDesc.py`:

```
from flask import Blueprint, render_template, request
import pandas as pd
import numpy as np
import pickle
import csv
import re

crimeCodeDesc = Blueprint('crimeCodeDesc', __name__)
with open('xgb_classifier_crime_desc.pkl', 'rb') as model_file:
    xgb_classifier = pickle.load(model_file)
```

```
def load_encoder_mapping(csv_file_path):
    mapping = {}
    with open(csv_file_path, mode='r', encoding='utf-8') as f:
        reader = csv.reader(f)
        for rows in reader:
            if len(rows) == 2 and rows[1].isdigit():
                mapping[rows[0]] = int(rows[1])
    return mapping

crime_description_mapping = load_encoder_mapping("label_encoder_crime_description.csv")
location_type_mapping = load_encoder_mapping("label_encoder_location_type_xgb.csv")

@crimeCodeDesc.route('/crimeCodeDesc')
def home():
    return render_template('crimeCodeDesc/crimeCodeDesc.html')

@crimeCodeDesc.route('/predictCrimeCodeDesc', methods=['POST'])
def predict_crime_code_desc():
    try:
        time_occurred_str = request.form.get('Time_Occurred')
        time_match = re.match(r'(\d{1,2}):(\d{2})', time_occurred_str)
        if time_match:
            hours, minutes = map(int, time_match.groups())
            time_occurred = hours * 100 + minutes
        else:
            raise ValueError('Invalid time format. Use HH:MM.')

        premise_code = float(request.form.get('Premise_Code'))
        area = int(request.form.get('Area'))
        location_type = request.form.get('Location_Type')
        location_type_encoded = location_type_mapping.get(location_type)
        features = np.array([[time_occurred, premise_code, area, location_type_encoded]])
        prediction_encoded = xgb_classifier.predict(features)
        prediction = None
        for key, value in crime_description_mapping.items():
            if value == prediction_encoded[0]:
                prediction = key
                break
        return render_template('crimeCodeDesc/crimeCodeDesc.html', prediction=prediction)
    except Exception as e:
        return str(e), 500
```

- First we define a route for `/crimeCodeDesc,` this route handles HTTP requests related to predicting crime code descriptions.
- It interacts with the HTML form in `crimeCodeDesc.html` by using the Flask `url_for` function to link the form submission action to `predict_crime_code_desc`.
- The form submission action sends input data (time occurred, premise code, area, location type) to the server for prediction.
- Upon submission, the Flask route processes the input data, likely utilizing a machine learning model or some predictive algorithm to determine and return the predicted crime code description.
- This predicted description is then displayed on the webpage under the appropriate section in the HTML file.

Result

Input:

Output:

Additional Features

We've incorporated a form within our system to efficiently gather information. This form is designed to capture various details from users, such as dates, times, crime codes, victim specifics, and more. The aim is to facilitate prompt reporting of crime incidents. For instance, consider a scenario where someone encounters a theft attempt, but due to their quick action, the actual crime is thwarted, and hence, goes unreported. Our form is intended precisely for such instances. It allows users to report these prevented crimes, providing a more comprehensive understanding of the safety landscape in a particular area.

Overview for Phase 3

Feature	Functionality	User Learning	Police Learning
Vehicle Theft Prediction Tool	Utilizes regression models to predict likelihood of vehicle theft	Identifying high-risk areas for vehicle theft	Efficiently deploying resources for targeted patrolling
Crime Difficulty Estimation Tool	Uses KNN to assess complexity of crimes based on multiple parameters	Understanding complexity levels of different crime instances	Allocating resources for handling more challenging cases
Crime Status Prediction Tool	Predicts potential crime types based on geographical and premise data	Anticipating probable crime types in specific areas	Strategically allocating resources for proactive crime prevention
Additional Reporting Form	Captures details of prevented/thwarted crime incidents	Contributing to a comprehensive safety landscape understanding	Gathering data on thwarted incidents for better crime analysis

1. **Vehicle Theft Prediction Tool:** This tool utilizes regression models to predict the likelihood of vehicle theft based on various parameters such as geographical coordinates, time, and premise code. Users can input their location details, and the tool helps identify safer places to park vehicles, enhancing overall safety measures.

   Learning: Users can learn about areas posing potential risks for vehicle theft, empowering them to make informed decisions regarding parking locations and improving their safety measures.

2. **Crime Difficulty Estimation Tool using KNN:** This tool assesses the complexity of crimes based on multiple parameters like time of occurrence, area code, victim details, and premise type. It helps law enforcement identify more challenging cases, assisting in resource allocation for effective management.

   Learning: Users, especially law enforcement, gain insights into which crimes might be more complex, aiding in resource allocation and strategic planning for crime management.

3. **Crime Status Prediction Tool with XGBoost:** By leveraging XGBoost, this tool predicts potential crime types based on geographical data, premise details, and location type. It offers an alert mechanism for law enforcement to strategically allocate resources and prevent crimes proactively.

4. **Additional Reporting Form:** This form captures various crime incident details, even those incidents that were prevented or thwarted. It allows users to report such incidents, contributing to a more comprehensive understanding of safety in specific areas.

   Learning: Users can contribute to a more holistic view of safety landscapes by reporting not only actual crimes but also prevented incidents.

To extend the project further we can implement:

- **Machine Learning Model Refinement:** Continuously refining and improving the machine learning models used for predictions to enhance accuracy and efficiency.
- **Community Engagement:** Involving the community in crime reporting or preventive measures through apps or forums, fostering a collective approach to safety.
- **Mobile Application Development:** Creating a mobile app version of the tools for convenient access and wider user reach.
- **Collaboration with Law Enforcement:** Collaborating with law enforcement agencies to integrate the tool into their systems for streamlined crime management.

These avenues could further enhance the product's utility, accuracy, and impact in addressing crime-related challenges and fostering safer communities.

Peer Evaluation Form - CSE 587B

Group Members:

- **Group Member 1:** Bhanu Chakra Sai Tarun Reddi
- **Group Member 2:** Charvi Kusuma

Evaluation Criteria:

Criteria	Group Member 1	Group Member 2
How effectively did your groupmate work with you?	5	5
Contribution in writing the report	5	5
Demonstrates a cooperative and supportive attitude	5	5
Contributes significantly to the success of the project	5	5
TOTAL	**20**	**20**

Overall Contribution:

- **Group Member 1:** 50%
- **Group Member 2:** 50%

References

- *1.1.1. What is EDA?* (n.d.). Nist.gov. Retrieved October 10, 2023, from https://www.itl.nist.gov/div898/handbook/eda/section1/eda11.htm
- Buhl, N. (n.d.). *Mastering data cleaning & data preprocessing*. Encord.com. Retrieved October 10, 2023, from https://encord.com/blog/data-cleaning-data-preprocessing/
- *Crime data from 2020 to present - catalog*. (n.d.). Data.gov. Retrieved October 10, 2023, from https://catalog.data.gov/dataset/crime-data-from-2020-to-present

- *Example gallery — seaborn 0.13.0 documentation*. (n.d.). Pydata.org. Retrieved October 10, 2023, from https://seaborn.pydata.org/examples/index.html
- *Getting started tutorials — pandas 2.1.1 documentation*. (n.d.). Pydata.org. Retrieved October 10, 2023, from https://pandas.pydata.org/docs/getting_started/intro_tutorials/index.html
- *NumPy v1.18 Manual*. (n.d.). Numpy.org. Retrieved October 10, 2023, from https://numpy.org/doc/1.18/
- *re — Regular expression operations*. (n.d.). Python Documentation. Retrieved October 10, 2023, from https://docs.python.org/3/library/re.html
- *Sklearn.Preprocessing.LabelEncoder*. (n.d.). Scikit-Learn. Retrieved October 10, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
- *Sklearn.Preprocessing.MinMaxScaler*. (n.d.). Scikit-Learn. Retrieved October 10, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
- *Using matplotlib — matplotlib 3.8.0 documentation*. (n.d.). Matplotlib.org. Retrieved October 10, 2023, from https://matplotlib.org/stable/users/index.html
- VanderPlas, J. (n.d.). *Visualization with Seaborn*. Github.Io. Retrieved October 10, 2023, from https://jakevdp.github.io/PythonDataScienceHandbook/04.14-visualization-with-seaborn.html
- *WordCloud for Python documentation — wordcloud 1.8.1 documentation*. (n.d.). Github.Io. Retrieved October 10, 2023, from https://amueller.github.io/word_cloud/
- 1.10. Decision Trees. (n.d.). Scikit-Learn. Retrieved November 12, 2023.
- Decision Trees. (n.d.). Datacamp.com. Retrieved November 12, 2023.
- Galarnyk, M. (2017, September 13). Logistic Regression using Python (scikit-learn). Towards Data Science.
- KNN. (n.d.). Datacamp.com. Retrieved November 12, 2023.
- Linear regression example. (n.d.). Scikit-Learn. Retrieved November 12, 2023.
- Python API Reference — xgboost 2.0.2 documentation. (n.d.). Readthedocs.Io. Retrieved November 12, 2023.
- Sklearn.Neighbors.KNeighborsClassifier. (n.d.). Scikit-Learn. Retrieved November 12, 2023.