Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

TITLE OF PROJECT:

Criminal Activity Hotspots Identification

Team Members:

20BDS0394 – SYED SHA SUHEB

20BDS0363 – MUTHUKRISHNAN P

20BDS0364 – SANTHOSHKUMAR S

Guided By:

DR. GUNAVATHI C

Table of the Content

ABSTRACT:

Our Crime prediction is an efficient methodology for examining patterns in crime. Our framework can foresee areas which have high likelihood for any crime event. This information can help the Law enforcement officials to accelerate the certain crime investigations and for tackling violations. Normally, 10% of the hoodlums (criminals) doing about 50% number of crimes. Notwithstanding the fact or statement that we can't anticipate who all might be the victims of crime however we can anticipate the spot that has high likelihood and we can implement more secure systems to safeguard people from crimes.

We use some Machine Leaming Algorithms like Naive Bayes, K-Means, Decision tree, Random Forest, and Logistic regression to categorize data based on their parameters. We are going to constantly train data for certain areas where the crime rate is high in our system in order to predict the future events.

Keywords:

Crime department, Decision Tree, Random Forest, K-fold Validation, Sampling (Oversampling).

INTRODUCTION:

Crime percentage is expanding now-a-days in numerous nations. In this day and age with such higher crime percentage and merciless wrongdoing occurring, there should be a few securities against this wrongdoing. Here we introduced a system by which wrongdoing rate can be diminished. Bad behaviour data should deal with into the structure.

We presented information mining calculation to anticipate wrongdoing. K-Means technique assumes a significant part in dissecting and foreseeing wrongdoings. K-Means technique will bunch co wrongdoers, joint effort and disintegration of coordinated wrongdoing gatherings, distinguishing different applicable wrongdoing designs, covered up joins, connect forecast and factual examination of wrongdoing information.

This framework will forestall wrongdoing happening in the public eye. Wrongdoing information is breaking down which is put away in the dataset. Information mining calculation will extricate data and examples from data set. Framework will bunch wrongdoing. Grouping will be done dependent on places where wrongdoing happened, pack who associated with wrongdoing and the circumstance wrongdoing occurred. This will assist with anticipating wrongdoing which will happen in future. Wrongdoing occurrence expectation relies principally upon the chronicle d wrongdoing record and different geospatial and segment data. Because of the expanding wrongdoing rate, it is of most extreme significance to have security against crimes. Information mining calculation can be acquired to group different sorts of wrongdoings presented in the public eye, which can be additionally used to examine and recognize wrongdoing.

This task utilizes K- methods bunching calculation to bunch information dependent on different factors like time, place, age, sort of wrongdoing carried out of charged. Wrongdoing information is put away in the data set to play out the examination. Information mining calculation will extricate data and examples from the information base. We accomplish grouping by places where wrongdoing has happened, blamed associated with the wrongdoing and the hour of wrongdoing occurring. Administrator will enter wrongdoing subtleties into the framework needed for expectation. Administrator can see chronicled criminal information. Wrongdoing occurrence location relies essentially upon the chronicled wrongdoing record.

DATASET DESCRIPTION:

The dataset that has been used for the particular project has been downloaded for the Kaggle website(https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2/data)

- **ID** – Unique identifier for the record.

- **Case Number** – The Chicago Police Department RD Number (Records Division Number), which is unique to the incident. Date – Date when the incident occurred. This is sometimes a best estimate.

- **Block** – The partially redacted address where the incident occurred, placing it on the same block as the actual address.

- **IUCR** – The Illinois Unifrom Crime Reporting code. This is directly linked to the Primary Type and Description. See the list of IUCR codes at https://data.cityofchicago.org/d/c7ck-438e.

- **Primary Type** – The primary description of the IUCR code.

- **Description** – The secondary description of the IUCR code, a subcategory of the primary description.

- **Location Description** – Description of the location where the incident occurred.

- **Arrest** – Indicates whether an arrest was made.

- **Domestic** – Indicates whether the incident was domestic-related as defined by the Illinois Domestic Violence Act.

- **Beat** – Indicates the beat where the incident occurred. A beat is the smallest police geographic area – each beat has dedicated police beat car. Three to five beats make up a police sector, and three sectors make up a police district. The Chicago Police Department has 22 police districts. See the beats at https://data.cityofchicago.org/d/aerh-rz74

- **District** – Indicates the police district where the incident occurred. See the districts at https://data.cityofchicago.org/d/fthy-xz3r

- **Ward** – The ward (City Council district) where the incident occurred. See the wards at https://data.cityofchicago.org/d/sp34-6z76

- **Community Area** – Indicates the community area where the incident occurred. Chicago has 77 community areas. See the community areas at https://data.cityofchicago.org/d/cauq-8yn6

- **FBI Code** – Indicates the crime classification as outlined in the FBI's National Incident-Based Reporting System (NIBRS). See the Chicago Police Department listing of these classifications at http://gis.chicagopolice.org/clearmap_crime_sums/crime_types.html

- **X Coordinate** – The x coordinate of the location where the incident occurred in State Plane Illinois East NAD 1983 projection. This location is shifted from the actual location for partial redaction but falls on the same block.

- **Y Coordinate** – The y coordinate of the location where the incident occurred in State Plane Illinois East NAD 1983 projection. This location is shifted from the actual location for partial redaction but falls on the same block.

- **Year** – Year the incident occurred.

- **Updated On** – Date and time the record was last updated.

- **Latitude** – The latitude of the location where the incident occurred. This location is shifted from the actual location for partial redaction but falls on the same block.

- **Longitude** – The longitude of the location where the incident occurred. This location is shifted from the actual location for partial redaction but falls on the same block.

- **Location** – The location where the incident occurred in a format that allows for creation of maps and other geographic operations on this data portal. This location is shifted from the actual location for partial redaction but falls on the same block.

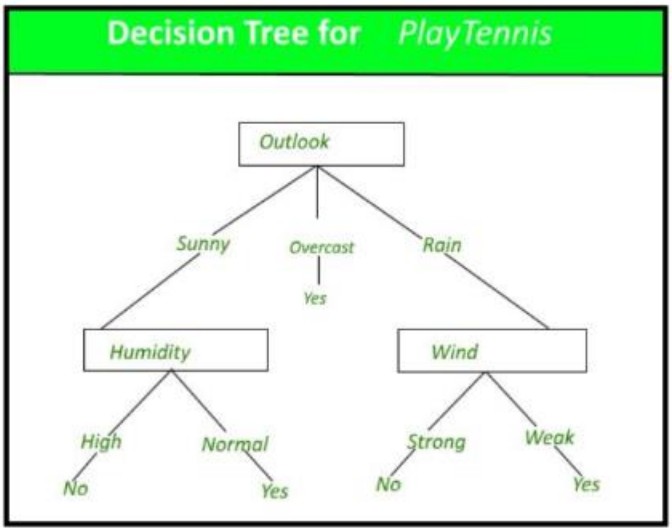
METHODOLOGY:

BASICS OF Machine Learning ALGORITHMS

There are numerous algorithms like Naive Bayes, K-Means, Decision tree, Random Forest, and Logistic regression that it can feel overpowering when algorithm names are tossed around, and we are required to simply understand what they are and where they fit.

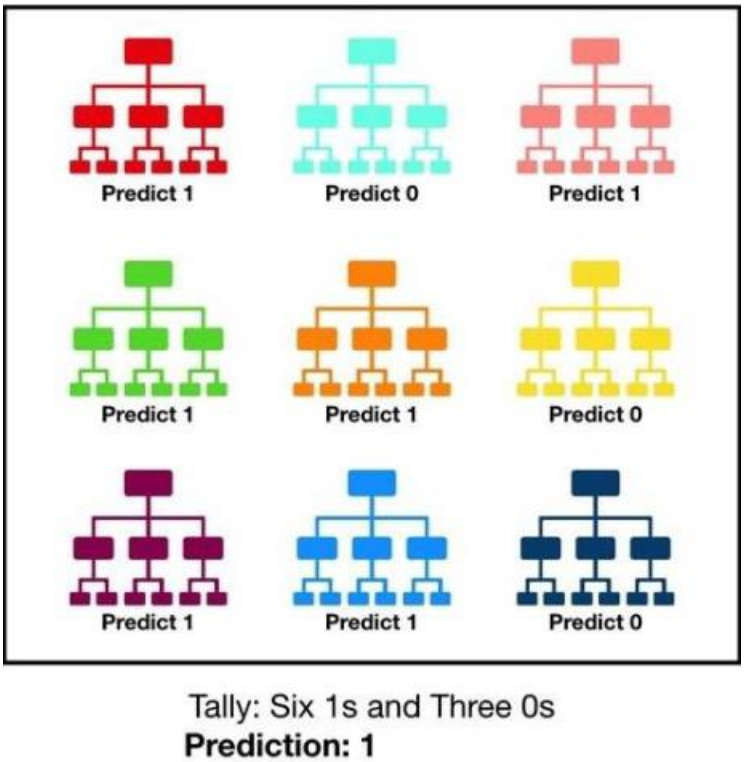
DECISION TREE

Decision Tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.



RANDOM FOREST

Random forest consists of many individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.



DATA IMBALANCE

Data imbalance is a common issue in many datasets, including crime datasets. In a crime dataset, the number of instances for different types of crimes can vary widely, with some types of crimes being much more common than others. This can create an imbalanced dataset, where the model trained on the data may perform poorly on the minority class of crimes.

To address data imbalance in a crime dataset, one approach is to use oversampling or undersampling techniques to balance the number of instances across classes. For example, if there are significantly more instances of property crimes compared to violent crimes, oversampling or undersampling techniques can be used to balance the number of instances for each type of crime.

K FOLD CROSS VALIDATION

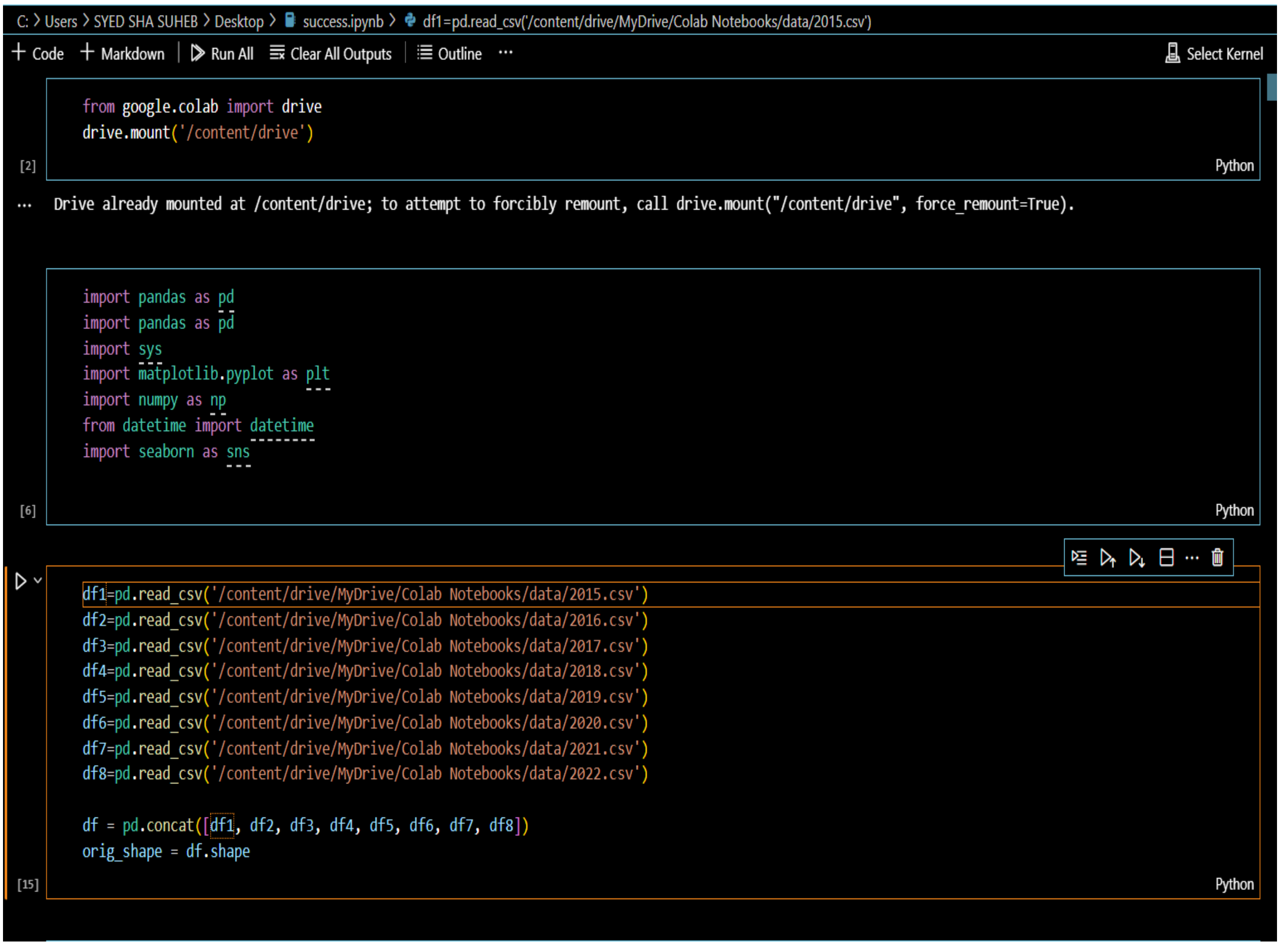
In the context of a crime dataset, k-fold cross-validation can be used to evaluate the performance of a machine learning model on predicting different types of crimes. For example, if the dataset includes multiple types of crimes, such as property crimes and violent crimes, k-fold cross-validation can be used to evaluate the performance of the model on predicting each type of crime.

To perform k-fold cross-validation on a crime dataset, the first step is to split the dataset into K equally sized folds. Next, the machine learning model is trained on K-1 of the folds, and tested on the remaining fold. This process is repeated K times, with each fold used for testing once.

After completing the K tests, the average performance of the model on all folds is calculated and reported as the final evaluation metric. This metric can be used to compare the performance of different machine learning algorithms on the crime dataset, or to evaluate the performance of a single algorithm on predicting different types of crimes.

Execution And Output:

Uploading Files:



Data frame

+ Code + Markdown | Run All Clear All Outputs | Outline ...

```
# Information about the main dataframe
df.info()
```

[17]

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1969767 entries, 0 to 220549
Data columns (total 22 columns):
 #   Column                Dtype
---  ------                -----
 0   ID                    int64
 1   Case Number           object
 2   Date                  object
 3   Block                 object
 4   IUCR                  object
 5   Primary Type          object
 6   Description           object
 7   Location Description  object
 8   Arrest                bool
 9   Domestic              bool
 10  Beat                  int64
 11  District              float64
 12  Ward                  float64
 13  Community Area        float64
 14  FBI Code              object
 15  X Coordinate          float64
 16  Y Coordinate          float64
 17  Year                  int64
 18  Updated On            object
 19  Latitude              float64
 20  Longitude             float64
 21  Location              object
dtypes: bool(2), float64(7), int64(3), object(10)
memory usage: 319.3+ MB
```

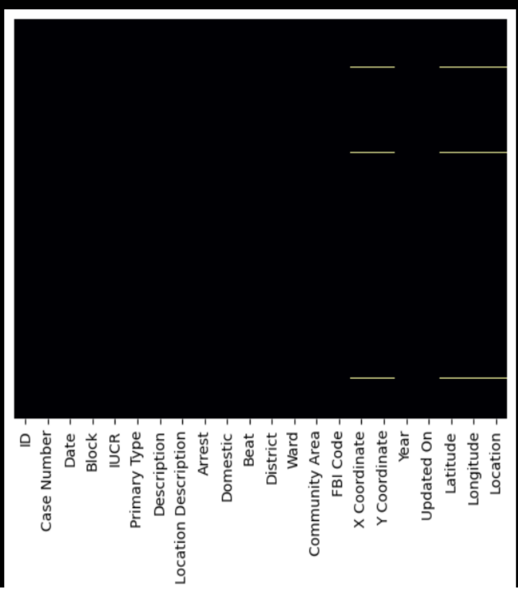
Missing Values

+ Code + Markdown | Run All Clear All Outputs | Outline ... Select Kernel

```
# Visualizing Missing Values
sns.heatmap(data = df.isna(), yticklabels=False, cbar=False, cmap='inferno')
```

[18] Python

```
<Axes: >
```



Checking Null Values

```
# To drop the rows with missing data
df = df.dropna()
df.isna().sum()
```

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

```
df.shape
```

```
(1933680, 22)
```

```
# Inspecting the loss of data after such cleaning
print("Data Retained after Cleaning:",round(((df.shape[0]/orig_shape[0]) * 100),2),"%")
```

```
Data Retained after Cleaning: 98.17 %
```

First 10 rows

+ Code + Markdown | Run All Clear All Outputs | Outline ... Select Kernel

```python
# First 10 rows (instances) of our dataset
df.head(10)
```

[23] Python

	ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	Domestic	...	Ward	Community Area	FBI Code	X Coordinate
0	10225520	HY412735	01/01/2015 12:00:00 AM	075XX S BLACKSTONE AVE	1153	DECEPTIVE PRACTICE	FINANCIAL IDENTITY THEFT OVER $ 300	RESIDENCE	False	False	...	5.0	43.0	11	1187511.0
2	10225760	HY412902	01/01/2015 12:00:00 AM	050XX N MARINE DR	0810	THEFT	OVER $500	APARTMENT	False	False	...	48.0	3.0	06	1169650.0
4	10229179	HY416572	01/01/2015 12:00:00 AM	039XX S LAKE PARK AVE	1752	OFFENSE INVOLVING CHILDREN	AGG CRIM SEX ABUSE FAM MEMBER	RESIDENCE	False	False	...	4.0	36.0	20	1183388.0
5	10231909	HY419527	01/01/2015 12:00:00 AM	047XX S CHAMPLAIN AVE	1752	OFFENSE INVOLVING CHILDREN	AGG CRIM SEX ABUSE FAM MEMBER	RESIDENCE	False	True	...	4.0	38.0	20	1181399.0
7	10245990	HY433843	01/01/2015 12:00:00 AM	045XX N WOLCOTT AVE	2825	OTHER OFFENSE	HARASSMENT BY TELEPHONE	APARTMENT	False	True	...	47.0	4.0	26	1163004.0
8	10249793	HY437344	01/01/2015 12:00:00 AM	078XX S MERRILL AVE	0266	CRIM SEXUAL ASSAULT	PREDATORY	RESIDENCE	False	True	...	8.0	43.0	02	1191894.0
9	10251294	HY438848	01/01/2015 12:00:00 AM	030XX W VAN BUREN ST	1562	SEX OFFENSE	AGG CRIMINAL SEXUAL ABUSE	RESIDENCE	False	False	...	28.0	27.0	17	1155955.0
10	10262792	HY450465	01/01/2015 12:00:00 AM	077XX S DR MARTIN LUTHER KING JR DR	1154	DECEPTIVE PRACTICE	FINANCIAL IDENTITY THEFT $300 AND UNDER	RESIDENCE	False	False	...	6.0	69.0	11	1180247.0

Dataset Columns

```python
# What are the features of our dataset?
print(df.columns)
```

[24]

```
Index(['ID', 'Case Number', 'Date', 'Block', 'IUCR', 'Primary Type',
       'Description', 'Location Description', 'Arrest', 'Domestic', 'Beat',
       'District', 'Ward', 'Community Area', 'FBI Code', 'X Coordinate',
       'Y Coordinate', 'Year', 'Updated On', 'Latitude', 'Longitude',
       'Location'],
      dtype='object')
```

Function to Clean the 'Date' Feature

```python
""" Function to Clean the 'Date' feature """

def time_convert(date_time):
    s1 = date_time[:11]
    s2 = date_time[11:]

    month = s1[:2]
    date = s1[3:5]
    year = s1[6:10]

    hr = s2[:2]
    mins = s2[3:5]
    sec = s2[6:8]
    time_frame = s2[9:]
    if(time_frame == 'PM'):
        if (int(hr) != 12):
            hr = str(int(hr) + 12)
    else:
        if(int(hr) == 12):
            hr = '00'

    final_date = datetime(int(year), int(month), int(date), int(hr), int(mins), int(sec))
    return final_date
```

```python
# Using apply() of pandas to apply time_convert on every row of the Date column
df['Date'] = df['Date'].apply(time_convert)
```

```python
# Inspect the cleaned "Date" column
df['Date'].head()
```

```
0    2015-01-01
2    2015-01-01
4    2015-01-01
5    2015-01-01
7    2015-01-01
Name: Date, dtype: datetime64[ns]
```

```python
""" Feature Engineering - Splitting the 'Date' feature into more suitable features for a Time-based analysis"""

# Feature Engineering 1 : Month
def month_col(x):
    return int(x.strftime("%m"))
df['Month'] = df['Date'].apply(month_col)

# Feature Engineering 2 : Day
def day_col(x):
    return int(x.strftime("%w"))
df['Day'] = df['Date'].apply(day_col)

# Feature Engineering 3 : Hour
def hour_col(x):
    return int(x.strftime("%H"))
df['Hour'] = df['Date'].apply(hour_col)
```

+ Code + Markdown | Run All | Clear All Outputs | Outline ··· Select Kernel

```python
# inspect the latest version of the dataset
df.head()
```

[30] Python

	ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	Domestic	...	X Coordinate	Y Coordinate	Year	Updated On	La
0	10225520	HY412735	2015-01-01	075XX S BLACKSTONE AVE	1153	DECEPTIVE PRACTICE	FINANCIAL IDENTITY THEFT OVER $ 300	RESIDENCE	False	False	...	1187511.0	1855334.0	2015	02/10/2018 03:50:01 PM	41.
2	10225760	HY412902	2015-01-01	050XX N MARINE DR	0810	THEFT	OVER $500	APARTMENT	False	False	...	1169650.0	1934124.0	2015	02/10/2018 03:50:01 PM	41.
4	10229179	HY416572	2015-01-01	039XX S LAKE PARK AVE	1752	OFFENSE INVOLVING CHILDREN	AGG CRIM SEX ABUSE FAM MEMBER	RESIDENCE	False	False	...	1183388.0	1878984.0	2015	02/10/2018 03:50:01 PM	41.
5	10231909	HY419527	2015-01-01	047XX S CHAMPLAIN AVE	1752	OFFENSE INVOLVING CHILDREN	AGG CRIM SEX ABUSE FAM MEMBER	RESIDENCE	False	True	...	1181399.0	1873687.0	2015	02/10/2018 03:50:01 PM	41.
7	10245990	HY433843	2015-01-01	045XX N WOLCOTT AVE	2825	OTHER OFFENSE	HARASSMENT BY TELEPHONE	APARTMENT	False	True	...	1163004.0	1930135.0	2015	02/10/2018 03:50:01 PM	41.

5 rows × 25 columns

+ Code + Markdown | Run All | Clear All Outputs | Outline ··· Select Kernel

```python
""" Filter the Top 10 most occuring crimes in the city of Chicago """

"""
STEPS FOLLOWED WHILE DOING THIS :

1. Take in each crime and make a dataset of it
2. Append the sub datasets to each other
"""
top_10 = list(df['Primary Type'].value_counts().head(10).index)

def filter_top_10(df0):
    df2=df0[df0['Primary Type']=='THEFT']
    for crime in top_10[1:]:
        temp=df0[df0['Primary Type']==crime]
        df2 = df2.append(temp, ignore_index=True)
    return df2

df2=filter_top_10(df) # the dataframe with all the data of only the top 10 crimes
df2.shape
```

[31] Python

```
<ipython-input-31-3cfd089e4cb7>:15: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use panda
  df2 = df2.append(temp, ignore_index=True)

(1763245, 25)
```

```python
1036588/1146382 * 100
```

[32] Python

```
90.42256420634658
```

+ Code + Markdown | Run All Clear All Outputs | Outline ··· Select Kern

```python
df2.head()
```

[33] Python

	ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	Domestic	...	X Coordinate	Y Coordinate	Year	Updated On	Latitu
0	10225760	HY412902	2015-01-01 00:00:00	050XX N MARINE DR	0810	THEFT	OVER $500	APARTMENT	False	False	...	1169650.0	1934124.0	2015	02/10/2018 03:50:01 PM	41.9747
1	10083956	HY272608	2015-01-01 00:00:00	0000X W TERMINAL ST	0820	THEFT	$500 AND UNDER	AIRPORT TERMINAL UPPER LEVEL - SECURE AREA	True	False	...	1100726.0	1934289.0	2015	02/10/2018 03:50:01 PM	41.9764
2	9988110	HY178276	2015-01-01 00:00:00	077XX S EMERALD AVE	0820	THEFT	$500 AND UNDER	RESIDENCE	False	True	...	1172642.0	1853548.0	2015	02/10/2018 03:50:01 PM	41.7535
3	10199737	HY387359	2015-01-01 00:01:00	059XX N SHERIDAN RD	0890	THEFT	FROM BUILDING	RESIDENCE	False	False	...	1168561.0	1939469.0	2015	02/10/2018 03:50:01 PM	41.9894
4	10094550	HY283318	2015-01-01 00:01:00	011XX N PARKSIDE AVE	0810	THEFT	OVER $500	RESIDENCE	True	False	...	1138472.0	1907037.0	2015	02/10/2018 03:50:01 PM	41.9010

5 rows × 25 columns

+ Code + Markdown | Run All Clear All Outputs | Outline ··· Select Kernel

```python
# Inspecting a few relevant features
df2[['Domestic', 'Beat', 'District', 'Ward', 'Community Area', 'FBI Code', 'Location', 'X Coordinate', 'Y Coordinate']].head()
```

[34] Python

	Domestic	Beat	District	Ward	Community Area	FBI Code	Location	X Coordinate	Y Coordinate
0	False	2024	20.0	48.0	3.0	06	(41.974742888, -87.651517395)	1169650.0	1934124.0
1	False	1652	16.0	41.0	76.0	06	(41.9764212, -87.904976266)	1100726.0	1934289.0
2	True	621	6.0	17.0	71.0	06	(41.753570578, -87.642897931)	1172642.0	1853548.0
3	False	2022	20.0	48.0	77.0	06	(41.989433412, -87.655366388)	1168561.0	1939469.0
4	False	1511	15.0	29.0	25.0	06	(41.90103704, -87.76682801)	1138472.0	1907037.0

```python
""" Grouping """

# Creating our explicit dataset
cri5 = df2.groupby(['Month','Day','District','Hour'], as_index=False).agg({"Primary Type":"count"})
cri5 = cri5.sort_values(by=['District'], ascending=False)
cri5.head()
```

[35] Python

	Month	Day	District	Hour	Primary Type
44403	12	6	31.0	11	1
8983	3	2	31.0	10	1
528	1	0	31.0	0	1
24841	7	4	31.0	21	1
16380	5	2	31.0	9	1

```python
# Renaming our feature
cri6=cri5.rename(index=str, columns={"Primary Type":"Crime_Count"})
cri6.head()
```

	Month	Day	District	Hour	Crime_Count
44403	12	6	31.0	11	1
8983	3	2	31.0	10	1
528	1	0	31.0	0	1
24841	7	4	31.0	21	1
16380	5	2	31.0	9	1

```python
#Exploring our Data
cri6 = cri6[['Month','Day','District','Hour','Crime_Count']]
cri6.head()
print("The shape of our final dataset is:", cri6.shape)
```

```
The shape of our final dataset is: (44404, 5)
```

```python
# Viewing the maximum and minmum crime counts
print("Highest Crime Count at any district at any time point:", cri6["Crime_Count"].max())
print("Lowest Crime Count at any district at any time point:", cri6["Crime_Count"].min())
```

```
Highest Crime Count at any district at any time point: 126
Lowest Crime Count at any district at any time point: 1
```

```python
print("Average no. of crimes per ditrict per time point :",round(cri6['Crime_Count'].sum()/cri6.shape[0], 2),".")
```

```
Average no. of crimes per ditrict per time point : 39.71 .
```

```python
# Inspecting our own lower and upper bounds to make a target feature "Alarm"
lower = np.mean(cri6['Crime_Count'])-0.75*np.std(cri6['Crime_Count'])
higher = np.mean(cri6['Crime_Count'])+0.75*np.std(cri6['Crime_Count'])
print(lower, higher)
```

```
24.794183312695825 54.62411233634481
```

Crime Count Distribution:

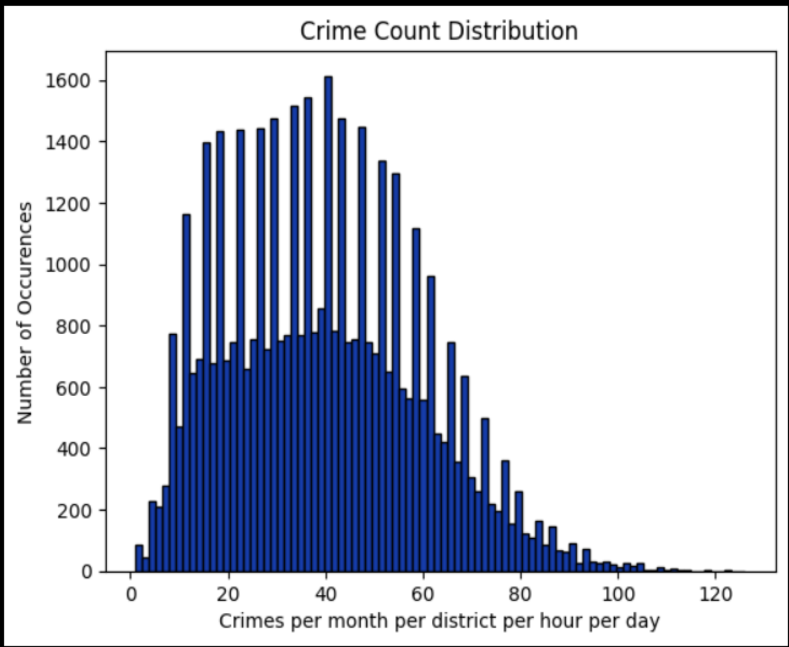
+ Code + Markdown | Run All Clear All Outputs | Outline ...

```python
# Crime Count Distribution plot (We need to be using this plot in order to devise our target feature, "Alarm")

plt.hist(x='Crime_Count', data=cri6,bins=90,linewidth=1,edgecolor='black', color='#163ca9')
#plt.title("Distribution of Crimes in Chicago", fontfamily="Agency FB", fontsize=25)
plt.xlabel("Crimes per month per district per hour per day")
plt.ylabel("Number of Occurences")
plt.title("Crime Count Distribution")
```

[41]

```
Text(0.5, 1.0, 'Crime Count Distribution')
```



+ Code + Markdown | Run All Clear All Outputs | Outline ...

```python
# 0-14 : Low Crime Rate
# 15-33 : Medium Crime Rate
# 34 and above : High Crime Rate

### The above ranges can be made better with the help of a crime analyst. As of now, we have used an intuitive way
### of generating classifications for our target feature; based on aproximating the distribution of the crime counts
### as a Normal curve

# Feature Engineer the above dataset
def crime_rate_assign(x):
    if(x<=14):
        return 0
    elif(x>14 and x<=33):
        return 1
    else:
        return 2
cri6['Alarm'] = cri6['Crime_Count'].apply(crime_rate_assign)
cri6 = cri6[['Month','Day','Hour','District','Crime_Count','Alarm']]
cri6.head()
```

[42]

	Month	Day	Hour	District	Crime_Count	Alarm
44403	12	6	11	31.0	1	0
8983	3	2	10	31.0	1	0
528	1	0	0	31.0	1	0
24841	7	4	21	31.0	1	0
16380	5	2	9	31.0	1	0

Correlation Heatmap

+ Code + Markdown | Run All Clear All Outputs | Outline ...

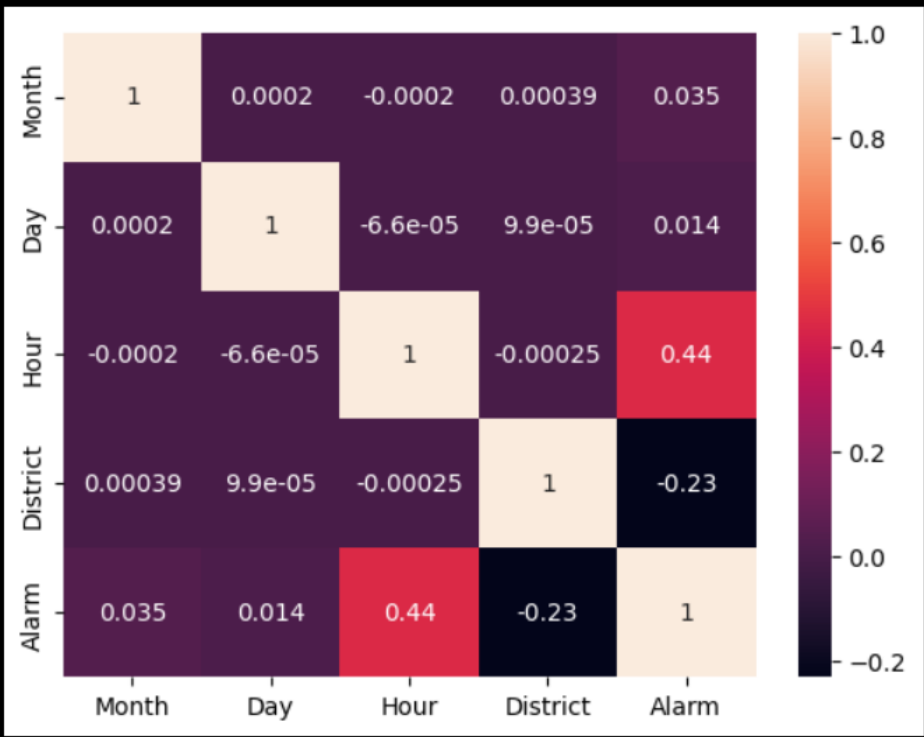
[43]

```python
# Correlation heatmap

temp = cri6[['Month', 'Day', 'Hour', 'District', 'Alarm']]
sns.heatmap(temp.corr(), annot=True)
#plt.title("Checking!", fontsize=17)
```

[44]

```
<Axes: >
```



+ Code + Markdown | Run All Clear All Outputs | Outline ...

```python
# Let's check how good our data is for classification
cri6['Alarm'].value_counts()
```

[45]

```
2    26110
1    13705
0     4589
Name: Alarm, dtype: int64
```

```python
print("Low Crime Rate Percentage:", round(cri6['Alarm'].value_counts()[0]/cri6['Alarm'].value_counts().sum()*100,2))
print("Medium Crime Rate Percentage:", round(cri6['Alarm'].value_counts()[1]/cri6['Alarm'].value_counts().sum()*100,2))
print("High Crime Rate Percentage:", round(cri6['Alarm'].value_counts()[2]/cri6['Alarm'].value_counts().sum()*100.2))
```

[46]

```
Low Crime Rate Percentage: 10.33
Medium Crime Rate Percentage: 30.86
High Crime Rate Percentage: 59
```

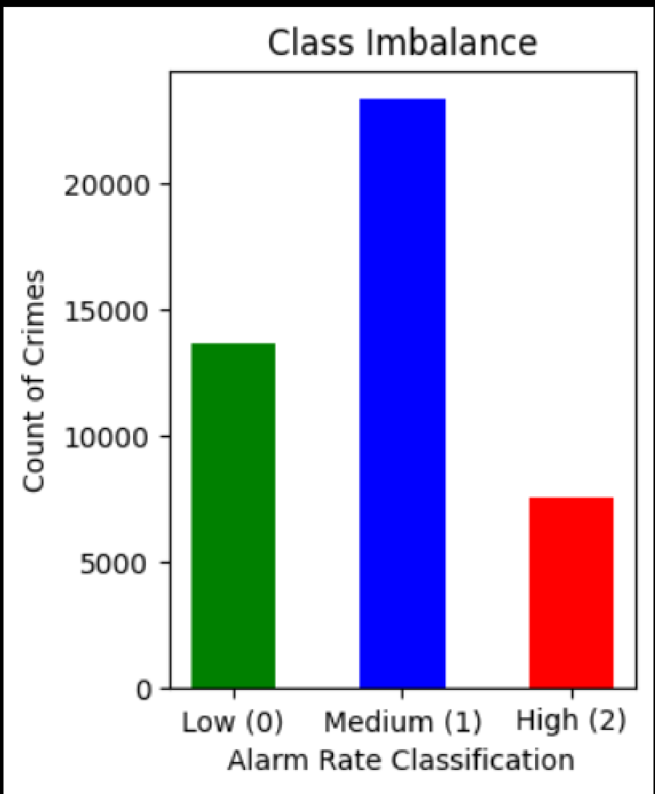
Imbalance Data Plot

```
# Plotting the Imbalance

x=['Low (0)','Medium (1)','High (2)']
y=[13600, 23273, 7488]
fig, ax = plt.subplots(figsize=(3, 4))
plt.bar(x,y, color=['green', 'blue', 'red'], width=0.5)
# plt.title('THE IMBALANCE IN THE DATASET')
plt.xlabel('Alarm Rate Classification')
plt.ylabel('Count of Crimes')
plt.title("Class Imbalance")
```

[47]

```
Text(0.5, 1.0, 'Class Imbalance')
```



+ Code + Markdown | Run All Clear All Outputs | Outline ...

```python
""" Building our completely unseen final test dataset for the 'GOD TEST 1' """

df9=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/2010.csv')
df10=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/2011.csv')
df11=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/2012.csv')
df12=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/2013.csv')
df13=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/2014.csv')

test_df = pd.concat([df9, df10, df11, df12, df13])

# Drop missing values
test_df = test_df.dropna()

# Using apply() of pandas to apply time_convert on every row of the Date column
test_df['Date'] = test_df['Date'].apply(time_convert)

# Feature Engineering our columns
test_df['Month'] = test_df['Date'].apply(month_col)
test_df['Day'] = test_df['Date'].apply(day_col)
test_df['Hour'] = test_df['Date'].apply(hour_col)

# Compressing
df7 = filter_top_10(test_df)
cri7 = df7.groupby(["Month", "Day", "District", "Hour"], as_index=False).agg({"Primary Type" : "count"})
cri7 = cri7.sort_values(by=["District"], ascending=False)
cri8 = cri7.rename(index=str, columns={"Primary Type" : "Crime_Count"})
cri8 = cri8[["Month", "Day", "District", "Hour", "Crime_Count"]]
cri8['Alarm'] = cri8['Crime_Count'].apply(crime_rate_assign)
cri8 = cri8[['Month','Day','Hour','District','Crime_Count','Alarm']]
print(cri8.head())
print("Class Imbalance\n")
print(cri8['Alarm'].value_counts())
```

[50]

```
... <ipython-input-31-3cfd089e4cb7>:15: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use panda
  df2 = df2.append(temp, ignore_index=True)
       Month  Day  Hour  District  Crime_Count  Alarm
31164      9    2     9        31            1      0
6864       2    5    12        31            1      0
29050      8    5    17        31            1      0
30107      9    0    14        31            1      0
17425      5    4    13        31            1      0
Class Imbalance

2    21540
1    15304
0     7521
Name: Alarm, dtype: int64
```

Oversampling

```
+ Code  + Markdown | Run All  Clear All Outputs | Outline ...
'''Creating the Oversampled balanced dataset'''

from sklearn.utils import resample # for upsampling

# Set individual classes
cri6_low = cri6[cri6['Alarm']==0]
cri6_medium = cri6[cri6['Alarm']==1]
cri6_high = cri6[cri6['Alarm']==2]

# Upsample the minority classes to size of class 1 (medium)
cri6_low_upsampled = resample(cri6_low,
                                  replace=True,     # sample with replacement
                                  n_samples=22640,    # to match majority class
                                  random_state=101)

cri6_high_upsampled = resample(cri6_high,
                                  replace=True,     # sample with replacement
                                  n_samples=22640,    # to match majority class
                                  random_state=101)

# Combine majority class with upsampled minority class
cri6_upsampled = pd.concat([cri6_medium, cri6_low_upsampled, cri6_high_upsampled])
[51]
```

Decision Tree

```
+ Code  + Markdown | Run All  Clear All Outputs | Outline ...
# Using Decision Trees for classification (Imbalanced Dataset)

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.utils.multiclass import unique_labels

X = cri6[['Month', 'Day', 'Hour', 'District']] # independent
y = cri6['Alarm'] # dependent

# Let's split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=101) # 75:25 split

# print(X_train)
# print('hi')
# print(X_test)
# Creating tree
d_tree = DecisionTreeClassifier(random_state=101)
# Fitting tree
d_tree = d_tree.fit(X_train, y_train)
# Predicting !
y_pred = d_tree.predict(X_test)

# Model Evaluation
# print(y_test)
# print(y_pred)
print("Accuracy:",(metrics.accuracy_score(y_test, y_pred)*100),"\n")

# Confusion Matrix for evaluating the model
cm = pd.crosstab(y_test, y_pred, rownames=['Actual Alarm'], colnames=['Predicted Alarm'])
print("\n---------Confusion Matrix------------------------------------")
print(cm)
```

Accuracy

```
...   Accuracy: 79.29916223763624


      ----------Confusion Matrix------------------------------------
      Predicted Alarm    0     1     2
      Actual Alarm
      0                750   407     3
      1                444  2316   739
      2                  5   700  5737

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

                 0       0.63      0.65      0.64      1160
                 1       0.68      0.66      0.67      3499
                 2       0.89      0.89      0.89      6442

          accuracy                           0.79     11101
         macro avg       0.73      0.73      0.73     11101
      weighted avg       0.79      0.79      0.79     11101


      UAR -> 0.7330056874653982
```

+ Code + Markdown | Run All Clear All Outputs | Outline ...

```python
# Using Decision Trees for classification (Balanced Dataset)

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.utils.multiclass import unique_labels

X = cri6_upsampled[['Month', 'Day', 'Hour', 'District']] # independent
y = cri6_upsampled['Alarm'] # dependent

# Let's split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=101) # 75:25 split

# print(X_train)
# print('hi')
# print(X_test)
# Creating tree
d_tree = DecisionTreeClassifier(random_state=101)
# Fitting tree
d_tree = d_tree.fit(X_train, y_train)
# Predicting !
y_pred = d_tree.predict(X_test)

# Model Evaluation
# print(y_test)
# print(y_pred)
print("Accuracy:",(metrics.accuracy_score(y_test, y_pred)*100),"\n")

# Confusion Matrix for evaluating the model
cm = pd.crosstab(y_test, y_pred, rownames=['Actual Alarm'], colnames=['Predicted Alarm'])
print("\n---------Confusion Matrix------------------------------------")
print(cm)

# Classification Report
```

```
+ Code  + Markdown | ▷ Run All  ≡ Clear All Outputs | ☰ Outline ···

    # Classification Report
    print("\n----------Classification Report----------------------------
    print(classification_report(y_test,y_pred))

    # Unweighted Average Recall
    print("\nUAR ->",((cm[0][0])/(cm[0][0]+cm[1][0]+cm[2][0])+(cm[1][1])/
[53]

... Accuracy: 89.40123414931851


    ----------Confusion Matrix-----------------------------------
    Predicted Alarm     0     1     2
    Actual Alarm
    0                5552    42     0
    1                 500  2351   584
    2                   6   431  5281

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

               0       0.92      0.99      0.95      5594
               1       0.83      0.68      0.75      3435
               2       0.90      0.92      0.91      5718

        accuracy                           0.89     14747
       macro avg       0.88      0.87      0.87     14747
    weighted avg       0.89      0.89      0.89     14747


    UAR -> 0.866830556172396
```

K-Fold Cross Validation

```python
# Let's try with KFold cross validation
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=100, shuffle=False)

X = cri6.iloc[:,0:4].values
y = cri6.iloc[:,5].values

i=1
scores = []
for train_index, test_index in skf.split(X, y):
    #print('{} of KFold {}'.format(i,skf.n_splits))
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    d_tree = DecisionTreeClassifier(random_state=101)
    # Fitting tree
    d_tree = d_tree.fit(X_train, y_train)
    # Predicting !
    y_pred = d_tree.predict(X_test)

    # Model Evaluation
    # print(y_test)
    # print(y_pred)
    scores.append(metrics.accuracy_score(y_test, y_pred)*100)
    #print("Accuracy:",(metrics.accuracy_score(y_test, y_pred)*100),"\n")

# Accuracy
print("Accuracy:",np.mean(scores),"\n")

# Confusion Matrix for evaluating the model
cm = pd.crosstab(y_test, y_pred, rownames=['Actual Alarm'], colnames=['Predicted Alarm'])
print("\n----------Confusion Matrix------------------------------------")
print(cm)
```

+ Code + Markdown | Run All Clear All Outputs | Outline ···

```python
# Unweighted Average Recall
print("\nUAR ->",((cm[0][0])/(cm[0][0]+cm[1][0]+cm[2][0])+(cm[1][1])/(cm[0][1]+cm[1][1]+cm[2
```

[54]

```
Accuracy: 71.15492964874987


----------Confusion Matrix------------------------------------
Predicted Alarm   0   1    2
Actual Alarm
0                27  14    4
1                30  70   37
2                 0   7  255

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

           0       0.47      0.60      0.53        45
           1       0.77      0.51      0.61       137
           2       0.86      0.97      0.91       262

    accuracy                           0.79       444
   macro avg       0.70      0.69      0.69       444
weighted avg       0.79      0.79      0.78       444


UAR -> 0.6947437826191937
```

Random Forest Classifier

```
# Using Random Forest for classification (Imbalanced Dataset)

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import joblib

X = cri6.iloc[:,0:4].values
y = cri6.iloc[:,5].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 101)

#scaler = StandardScaler()
#X_train = scaler.fit_transform(X_train)
#X_test = scaler.transform(X_test)

classifier = RandomForestClassifier(n_estimators = 1000, criterion = 'entropy', random_state = 1
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print("Accuracy:",(metrics.accuracy_score(y_test, y_pred)*100),"\n")

cm = pd.crosstab(y_test, y_pred, rownames=['Actual Alarm'], colnames=['Predicted Alarm'])
print("\n---------Confusion Matrix------------------------------------")
print(cm)

# Classification Report
print("\n---------Classification Report------------------------------------")
print(classification_report(y_test,y_pred))
```

Accuracy

+ Code + Markdown | Run All Clear All Outputs | Outline ...

```
    print("\n----------Classification Report-------------------------
    print(classification_report(y_test,y_pred))

    # Unweighted Average Recall
    print("\nUAR ->",((cm[0][0])/(cm[0][0]+cm[1][0]+cm[2][0])+(cm[1][[
[55]
```

```
...  Accuracy: 84.08251508873074


     ----------Confusion Matrix------------------------------------
     Predicted Alarm    0     1     2
     Actual Alarm
     0                781   377     2
     1                267  2594   638
     2                  2   481  5959

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

                0       0.74      0.67      0.71      1160
                1       0.75      0.74      0.75      3499
                2       0.90      0.93      0.91      6442

         accuracy                           0.84     11101
        macro avg       0.80      0.78      0.79     11101
     weighted avg       0.84      0.84      0.84     11101


     UAR -> 0.7798846065089355
```

```python
# Using Random Forest for classification (Balanced Dataset)

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import joblib

X = cri6_upsampled.iloc[:,0:4].values
y = cri6_upsampled.iloc[:,5].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 101)

#scaler = StandardScaler()
#X_train = scaler.fit_transform(X_train)
#X_test = scaler.transform(X_test)

classifier = RandomForestClassifier(n_estimators = 1000, criterion = 'entropy', random_state = 101)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print("Accuracy:",(metrics.accuracy_score(y_test, y_pred)*100),"\n")

cm = pd.crosstab(y_test, y_pred, rownames=['Actual Alarm'], colnames=['Predicted Alarm'])
print("\n----------Confusion Matrix------------------------------------")
print(cm)

# Classification Report
print("\n----------Classification Report------------------------------------")
print(classification_report(y_test,y_pred))

# Unweighted Average Recall
print("\nUAR ->",((cm[0][0])/(cm[0][0]+cm[1][0]+cm[2][0])+(cm[1][1])/(cm[0][1]+cm[1][1]+cm[2][1])+(cm[2][2]
```
[56]

+ Code + Markdown | Run All Clear All Outputs | Outline ...

```python
# Unweighted Average Recall
print("\nUAR ->",((cm[0][0])/(cm[0][0]+cm[1][0]+cm[2][0])+(cm[1][1])/(cm[0][1]+cm[1][1]+
```

[56]

```
Accuracy: 90.14036753237946


----------Confusion Matrix------------------------------------
Predicted Alarm     0     1     2
Actual Alarm
0                5554    40     0
1                 562  2362   511
2                   1   340  5377

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

           0       0.91      0.99      0.95      5594
           1       0.86      0.69      0.76      3435
           2       0.91      0.94      0.93      5718

    accuracy                           0.90     14747
   macro avg       0.89      0.87      0.88     14747
weighted avg       0.90      0.90      0.90     14747


UAR -> 0.873613536832576
```

+ Code + Markdown | Run All | Clear All Outputs | Outline ...

```python
# Using Random Forest for classification (Imbalanced Dataset) (using k-fold)

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import joblib

X = cri6.iloc[:,0:4].values
y = cri6.iloc[:,5].values

scores = []
for train_index, test_index in skf.split(X, y):
    #print('{} of KFold {}'.format(i,skf.n_splits))
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    classifier = RandomForestClassifier(n_estimators = 100, criterion = 'entropy', random_state = 101)
    classifier.fit(X_train, y_train)

    y_pred = classifier.predict(X_test)

    # Model Evaluation
    # print(y_test)
    # print(y_pred)
    scores.append(metrics.accuracy_score(y_test, y_pred)*100)
    #print("Accuracy:",(metrics.accuracy_score(y_test, y_pred)*100),"\n")

#scaler = StandardScaler()
#X_train = scaler.fit_transform(X_train)
#X_test = scaler.transform(X_test)

# Accuracy
print("Accuracy:",np.mean(scores),"\n")

cm = pd.crosstab(y_test, y_pred, rownames=['Actual Alarm'], colnames=['Predicted Alarm'])
```

+ Code + Markdown | Run All Clear All Outputs | Outline ···

```
# Unweighted Average Recall
print("\nUAR ->",((cm[0][0])/(cm[0][0]+cm[1][0]+cm[2][0])+(cm[1][1])/(cm[0][1]+cm[1][1]+cm[2
```

[57]

```
Accuracy: 75.45872051827108


----------Confusion Matrix------------------------------------
Predicted Alarm   0   1    2
Actual Alarm
0                28  17    0
1                29  86   22
2                 0   3  259

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

           0       0.49      0.62      0.55        45
           1       0.81      0.63      0.71       137
           2       0.92      0.99      0.95       262

    accuracy                           0.84       444
   macro avg       0.74      0.75      0.74       444
weighted avg       0.84      0.84      0.84       444


UAR -> 0.7461696889400683
```

Conclusion:

After pre-processing the selected crime dataset, we selected some machine learning algorithms such as Decision Tree, Random Forest. We have predicted the crime rate using each of the above selected algorithms and measured its prediction Accuracy, Precision, Recall, F1 Score. we come to know that the metrics of Random Forest is higher than the other algorithms. Random forest algorithm is the best suitable for the prediction of the crime rate.

References:

Dataset: https://www.kaggle.com/datasets/onlyrohit/crimes-in-chicago

Execution:https://colab.research.google.com/drive/1PryGgYLCA_MTEq9I4OJYmaSzZZ_5cHEx?usp=share_link