```
# Visualization Libraries
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

#Preprocessing Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score, f1_score

# ML Libraries
from sklearn.ensemble import RandomForestClassifier,VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier

# Evaluation Metricsl
from yellowbrick.classifier import ClassificationReport
from sklearn import metrics
from sklearn.svm import SVC
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving data1.csv to data1.csv
Saving data2.csv to data2.csv
Saving data3.csv to data3.csv
Saving data4.csv to data4.csv
```

```
df = pd.concat([pd.read_csv('data1.csv', error_bad_lines=False)],ignore_index=True)
df = pd.concat([pd.read_csv('data2.csv', error_bad_lines=False)], ignore_index=True)
df = pd.concat([df, pd.read_csv('data3.csv', error_bad_lines=False)], ignore_index=True)
df = pd.concat([df, pd.read_csv('data4.csv', error_bad_lines=False)], ignore_index=True)
df.head()
```

```
<ipython-input-3-130af21cb0c5>:1: FutureWarning: The error_bad_lines argument has been deprecated and will be removed in a future version. Use on_bad_line


  df = pd.concat([pd.read_csv('data1.csv', error_bad_lines=False)],ignore_index=True)
<ipython-input-3-130af21cb0c5>:2: FutureWarning: The error_bad_lines argument has been deprecated and will be removed in a future version. Use on_bad_line


  df = pd.concat([pd.read_csv('data2.csv', error_bad_lines=False)], ignore_index=True)
<ipython-input-3-130af21cb0c5>:3: FutureWarning: The error_bad_lines argument has been deprecated and will be removed in a future version. Use on_bad_line


  df = pd.concat([df, pd.read_csv('data3.csv', error_bad_lines=False)], ignore_index=True)
<ipython-input-3-130af21cb0c5>:4: FutureWarning: The error_bad_lines argument has been deprecated and will be removed in a future version. Use on_bad_line


  df = pd.concat([df, pd.read_csv('data4.csv', error_bad_lines=False)], ignore_index=True)
```

	Unnamed: 0	ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	...	Ward	Community Area	FBI Code	Coordinat
0	0	4673626	HM274058	04-02-2006 13:00	055XX N MANGO AVE	2825	OTHER OFFENSE	HARASSMENT BY TELEPHONE	RESIDENCE	False	...	45.0	11.0	26	1136872.(
1	1	4673627	HM202199	02/26/2006 01:40:48 PM	065XX S RHODES AVE	2017	NARCOTICS	MANU/DELIVER:CRACK	SIDEWALK	True	...	20.0	42.0	18	1181027.(
2	2	4673628	HM113861	01-08-2006 23:16	013XX E 69TH ST	051A	ASSAULT	AGGRAVATED: HANDGUN	OTHER	False	...	5.0	69.0	04A	1186023.(
3	4	4673629	HM274049	04-05-2006 18:45	061XX W NEWPORT AVE	460	BATTERY	SIMPLE	RESIDENCE	False	...	38.0	17.0	08B	1134772.(

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175975 entries, 0 to 175974
Data columns (total 23 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Unnamed: 0            175975 non-null  int64
 1   ID                    175975 non-null  int64
 2   Case Number           175975 non-null  object
 3   Date                  175975 non-null  object
 4   Block                 175975 non-null  object
 5   IUCR                  175975 non-null  object
 6   Primary Type          175975 non-null  object
```

```
 7   Description           175975 non-null  object
 8   Location Description  175877 non-null  object
 9   Arrest                175975 non-null  bool
 10  Domestic              175975 non-null  bool
 11  Beat                  175975 non-null  int64
 12  District              175974 non-null  float64
 13  Ward                  175973 non-null  float64
 14  Community Area        175866 non-null  float64
 15  FBI Code              175975 non-null  object
 16  X Coordinate          155141 non-null  float64
 17  Y Coordinate          155141 non-null  float64
 18  Year                  175975 non-null  int64
 19  Updated On            175975 non-null  object
 20  Latitude              155141 non-null  float64
 21  Longitude             155141 non-null  float64
 22  Location              155141 non-null  object
dtypes: bool(2), float64(7), int64(4), object(10)
memory usage: 28.5+ MB
```

```
# Preprocessing
# Remove NaN Value (As Dataset is huge, the NaN row could be neglectable)
df = df.dropna()


# As the dataset is too huge is size, we would just subsampled a dataset for modelling as proof of concept
df = df.sample(n=100000)


# Remove irrelevant/not meaningfull attributes
df = df.drop(['Unnamed: 0'], axis=1)
df = df.drop(['ID'], axis=1)
df = df.drop(['Case Number'], axis=1)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100000 entries, 4739 to 167319
Data columns (total 20 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Date                  100000 non-null  object
 1   Block                 100000 non-null  object
 2   IUCR                  100000 non-null  object
 3   Primary Type          100000 non-null  object
 4   Description           100000 non-null  object
 5   Location Description  100000 non-null  object
 6   Arrest                100000 non-null  bool
```

```
 7   Domestic              100000 non-null  bool
 8   Beat                  100000 non-null  int64
 9   District              100000 non-null  float64
 10  Ward                  100000 non-null  float64
 11  Community Area        100000 non-null  float64
 12  FBI Code              100000 non-null  object
 13  X Coordinate          100000 non-null  float64
 14  Y Coordinate          100000 non-null  float64
 15  Year                  100000 non-null  int64
 16  Updated On            100000 non-null  object
 17  Latitude              100000 non-null  float64
 18  Longitude             100000 non-null  float64
 19  Location              100000 non-null  object
dtypes: bool(2), float64(7), int64(2), object(9)
memory usage: 14.7+ MB
```

```
# Splitting the Date to Day, Month, Year, Hour, Minute, Second
df['date2'] = pd.to_datetime(df['Date'])
df['Year'] = df['date2'].dt.year
df['Month'] = df['date2'].dt.month
df['Day'] = df['date2'].dt.day
df['Hour'] = df['date2'].dt.hour
df['Minute'] = df['date2'].dt.minute
df['Second'] = df['date2'].dt.second
df = df.drop(['Date'], axis=1)
df = df.drop(['date2'], axis=1)
df = df.drop(['Updated On'], axis=1)
df.head()
```

	Block	IUCR	Primary Type	Description	Location Description	Arrest	Domestic	Beat	District	Ward	...	Y Coordinate	Year	Latitu
4739	063XX N HOYNE AVE	810	THEFT	OVER $500	PARKING LOT/GARAGE(NON.RESID.)	False	False	2413	24.0	50.0	...	1942168.0	2006	41.9969
14717	017XX W JUNEWAY TER	1350	CRIMINAL TRESPASS	TO STATE SUP LAND	CHA HALLWAY/STAIRWELL/ELEVATOR	True	False	2422	24.0	49.0	...	1951493.0	2006	42.0225
	026XX S													

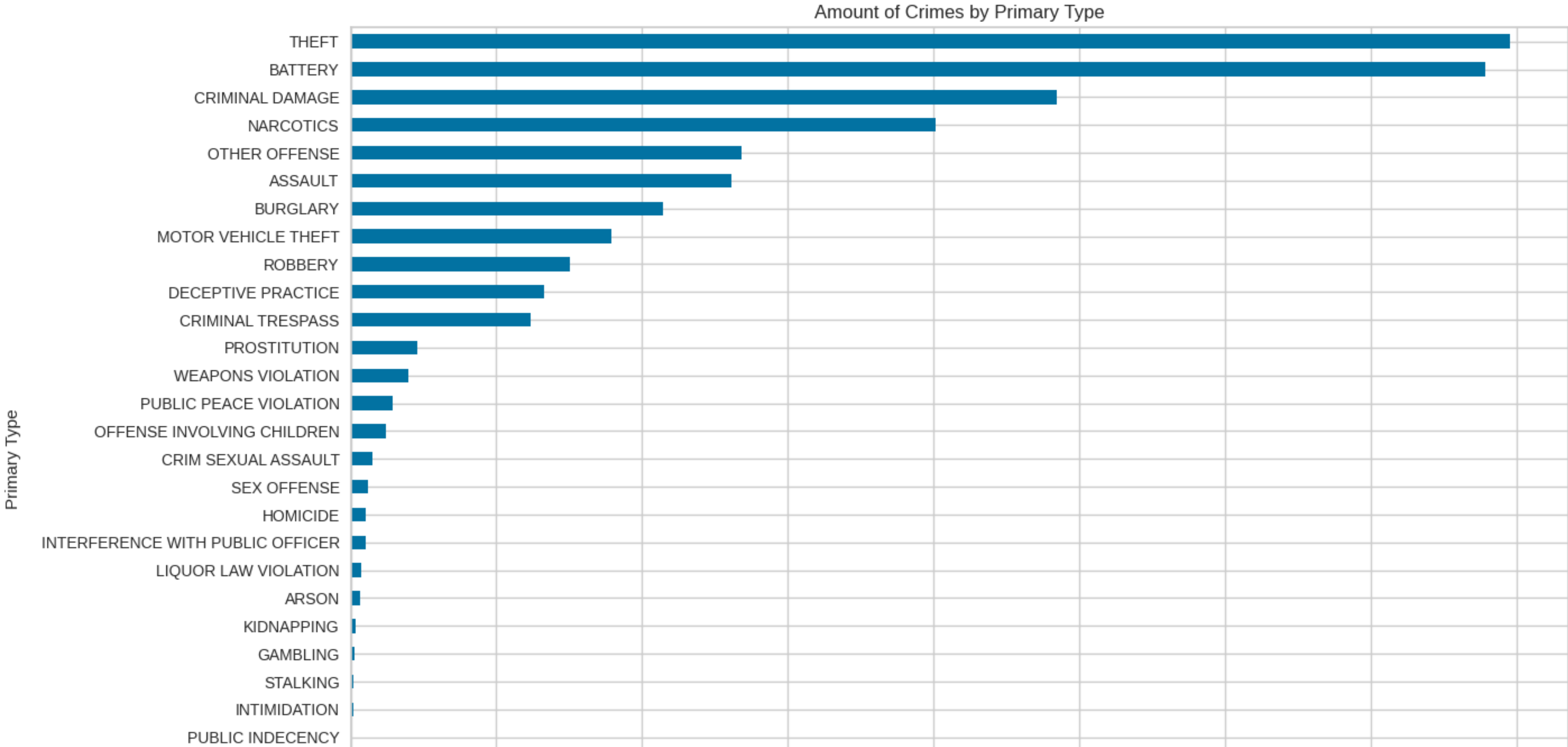
```
# Convert Categorical Attributes to Numerical
df['Block'] = pd.factorize(df["Block"])[0]
df['IUCR'] = pd.factorize(df["IUCR"])[0]
df['Description'] = pd.factorize(df["Description"])[0]
df['Location Description'] = pd.factorize(df["Location Description"])[0]
df['FBI Code'] = pd.factorize(df["FBI Code"])[0]
df['Location'] = pd.factorize(df["Location"])[0]
```

134931	WASHINGTON	1153	DECEPTIVE	IDENTITY	RESIDENCE	False	False	1123	11.0	28.0	...	1900496.0	2015	41.8827

```
Target = 'Primary Type'
print('Target: ', Target)
```

```
Target:  Primary Type
```

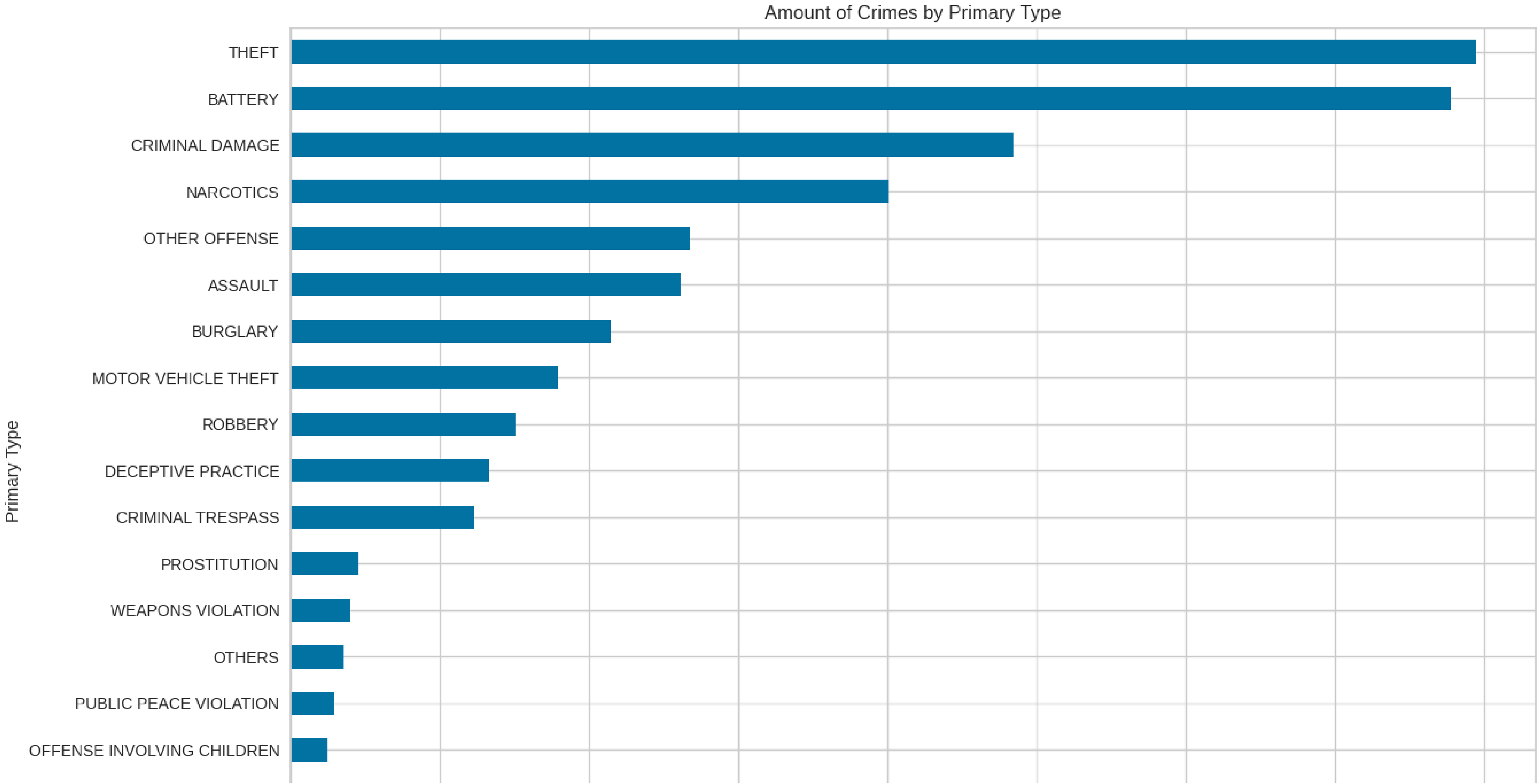
```
# Plot Bar Chart visualize Primary Types
plt.figure(figsize=(14,10))
plt.title('Amount of Crimes by Primary Type')
plt.ylabel('Crime Type')
plt.xlabel('Amount of Crimes')

df.groupby([df['Primary Type']]).size().sort_values(ascending=True).plot(kind='barh')

plt.show()
```




```
# At previous plot, we could see that the classes is quite imbalance
# Therefore, we are going to group several less occured Crime Type into 'Others' to reduce the Target Class amount

# First, we sum up the amount of Crime Type happened and select the last 13 classes
all_classes = df.groupby(['Primary Type'])['Block'].size().reset_index()
all_classes['Amt'] = all_classes['Block']
all_classes = all_classes.drop(['Block'], axis=1)
all_classes = all_classes.sort_values(['Amt'], ascending=[False])

unwanted_classes = all_classes.tail(13)
unwanted_classes
```

	Primary Type	Amt
12	INTERFERENCE WITH PUBLIC OFFICER	254
15	LIQUOR LAW VIOLATION	187
0	ARSON	163
14	KIDNAPPING	95
9	GAMBLING	70
28	STALKING	48
13	INTIMIDATION	46
24	PUBLIC INDECENCY	5
20	OBSCENITY	4
19	NON-CRIMINAL	2
18	NON - CRIMINAL	1
11	HUMAN TRAFFICKING	1

```
# After that, we replaced it with label 'OTHERS'
df.loc[df['Primary Type'].isin(unwanted_classes['Primary Type']), 'Primary Type'] = 'OTHERS'

# Plot Bar Chart visualize Primary Types
plt.figure(figsize=(14,10))
plt.title('Amount of Crimes by Primary Type')
plt.ylabel('Crime Type')
plt.xlabel('Amount of Crimes')

df.groupby([df['Primary Type']]).size().sort_values(ascending=True).plot(kind='barh')

plt.show()
```



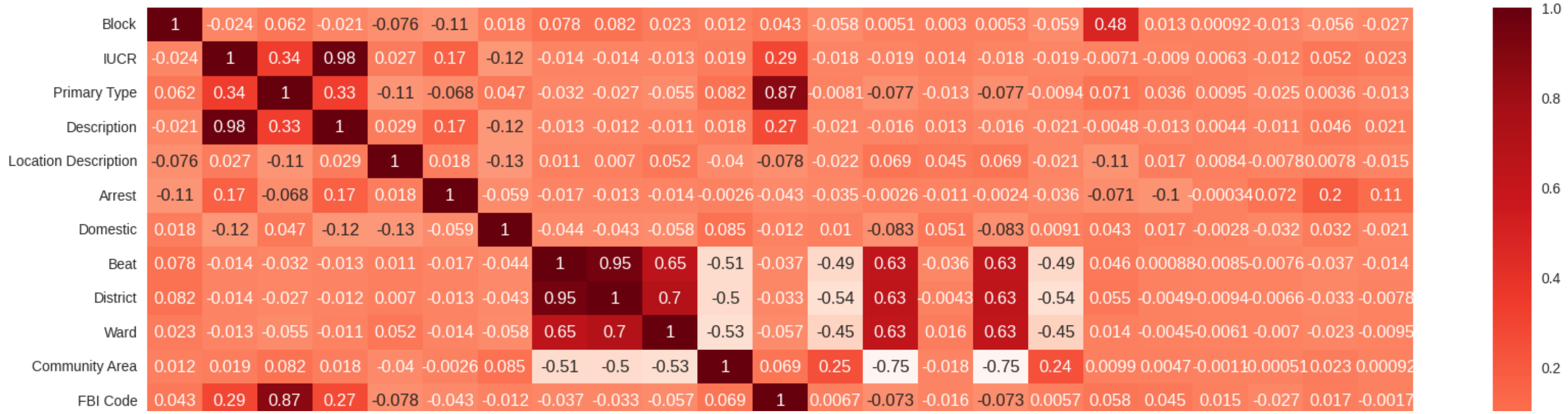

```
# Now we are left with 14 Class as our predictive class
Classes = df['Primary Type'].unique()
Classes
```

```
array(['THEFT', 'CRIMINAL TRESPASS', 'NARCOTICS', 'DECEPTIVE PRACTICE',
       'BATTERY', 'BURGLARY', 'PUBLIC PEACE VIOLATION', 'ROBBERY',
       'OTHER OFFENSE', 'CRIMINAL DAMAGE', 'SEX OFFENSE', 'ASSAULT',
       'CRIM SEXUAL ASSAULT', 'MOTOR VEHICLE THEFT', 'OTHERS',
       'OFFENSE INVOLVING CHILDREN', 'WEAPONS VIOLATION', 'PROSTITUTION',
       'HOMICIDE'], dtype=object)
```

```
#Encode target labels into categorical variables:
df['Primary Type'] = pd.factorize(df["Primary Type"])[0]
df['Primary Type'].unique()
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18])
```

```
# Feature Selection using Filter Method
# Split Dataframe to target class and features
X_fs = df.drop(['Primary Type'], axis=1)
Y_fs = df['Primary Type']

#Using Pearson Correlation
plt.figure(figsize=(20,10))
cor = df.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
```



Further Elaboration of Correlation

The correlation coefficient has values between -1 to 1

- A value closer to 0 implies weaker correlation (exact 0 implying no correlation)
- A value closer to 1 implies stronger positive correlation
- A value closer to -1 implies stronger negative correlation

```
#Correlation with output variable
cor_target = abs(cor['Primary Type'])
#Selecting highly correlated features
relevant_features = cor_target[cor_target>0.2]
relevant_features
```

```
IUCR            0.340309
Primary Type    1.000000
Description     0.333508
FBI Code        0.873328
Name: Primary Type, dtype: float64
```

```
# At Current Point, the attributes is select manually based on Feature Selection Part.
Features = ["IUCR", "Description", "FBI Code"]
print('Full Features: ', Features)
```

```
Full Features:  ['IUCR', 'Description', 'FBI Code']
```

```
#Split dataset to Training Set & Test Set
x, y = train_test_split(df,
```

```
                        test_size = 0.2,
                        train_size = 0.8,
                        random_state= 3)

x1 = x[Features]    #Features to train
x2 = x[Target]      #Target Class to train
y1 = y[Features]    #Features to test
y2 = y[Target]      #Target Class to test

print('Feature Set Used    : ', Features)
print('Target Class        : ', Target)
print('Training Set Size   : ', x.shape)
print('Test Set Size       : ', y.shape)
```

```
Feature Set Used    :  ['IUCR', 'Description', 'FBI Code']
Target Class        :  Primary Type
Training Set Size   :  (80000, 23)
Test Set Size       :  (20000, 23)
```

Machine Learning Modelling

```
from sklearn.svm import SVC
from sklearn import metrics
svc=SVC() #Default hyperparameters
#svc.fit(X_train,y_train)
# Create Model with configuration
rf_model = RandomForestClassifier(n_estimators=70, # Number of trees
                                  min_samples_split = 30,
                                  bootstrap = True,
                                  max_depth = 50,
                                  min_samples_leaf = 25)

# Model Training
rf_model.fit(X=x1,
             y=x2)

# Prediction
result = rf_model.predict(y[Features])


# Model Evaluation
ac_sc = accuracy_score(y2, result)
rc_sc = recall_score(y2, result, average="weighted")
pr_sc = precision_score(y2, result, average="weighted")
```

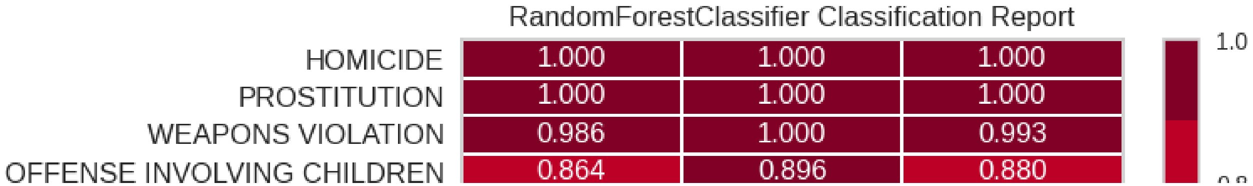
```
f1_sc = f1_score(y2, result, average='micro')
confusion_m = confusion_matrix(y2, result)

print("========== SVM Results ==========")
print("Accuracy    : ", ac_sc)
print("Recall      : ", rc_sc)
print("Precision   : ", pr_sc)
print("F1 Score    : ", f1_sc)
print("Confusion Matrix: ")
print(confusion_m)
```

```
========== SVM Results ==========
Accuracy    :  0.99485
Recall      :  0.99485
Precision   :  0.9949102531584498
F1 Score    :  0.99485
Confusion Matrix:
[[3914    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0]
 [   0  594    0    0    0    0    0    0    3    0    0    0    0    0
     0    0    0    0    0]
 [   0    0 2001    0    0    0    0    0    0    0    0    0    0    0
     0    4    0    0    0]
 [   0    0    0  701    0    0    0    0    0    0    0    0    0    0
     5    0    0    0    0]
 [   0    0    0    1 3913    0    0    0    0    0    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    0 1026    0    0    0    0    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    0    0  159    0    6    0    0    0    0    0
     1    0    0    0    0]
 [   0    0    0    0    0    0    0  762    0    0    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    0    0   11    9 1339    0    2    0    0    0
     3    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0 2429    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0   59    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0 1251    0    0
     0    0    0    0    0]
 [   0    0    0    0    2    0    0    0    0    0    0    6   73    0
     0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0  947
     0    0    0    0    0]
 [   0    2    3    0    0    0   12    0    0    0    4    7    0    0
   130   11    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    2    0    0    0
```

```
     6   95    3    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0  205    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0  238    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0   61]]
```

```
# Classification Report
# Instantiate the classification model and visualizer
target_names = Classes
visualizer = ClassificationReport(rf_model, classes=target_names)
visualizer.fit(X=x1, y=x2)     # Fit the training data to the visualizer
visualizer.score(y1, y2)       # Evaluate the model on the test data

print('================= Classification Report =================')
print('')
print(classification_report(y2, result, target_names=target_names))

g = visualizer.poof()             # Draw/show/poof the data
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with
  warnings.warn(
================= Classification Report =================

                            precision    recall  f1-score   support

                     THEFT       1.00      1.00      1.00      3914
         CRIMINAL TRESPASS       1.00      0.99      1.00       597
                 NARCOTICS       1.00      1.00      1.00      2005
        DECEPTIVE PRACTICE       1.00      0.99      1.00       706
                   BATTERY       1.00      1.00      1.00      3914
                  BURGLARY       1.00      1.00      1.00      1026
     PUBLIC PEACE VIOLATION       0.87      0.96      0.91       166
                   ROBBERY       0.99      1.00      0.99       762
             OTHER OFFENSE       0.99      0.98      0.99      1364
           CRIMINAL DAMAGE       1.00      1.00      1.00      2429
               SEX OFFENSE       0.88      1.00      0.94        59
                   ASSAULT       0.99      1.00      0.99      1251
        CRIM SEXUAL ASSAULT       1.00      0.90      0.95        81
        MOTOR VEHICLE THEFT       1.00      1.00      1.00       947
                    OTHERS       0.90      0.77      0.83       169
   OFFENSE INVOLVING CHILDREN       0.86      0.90      0.88       106
          WEAPONS VIOLATION       0.99      1.00      0.99       205
              PROSTITUTION       1.00      1.00      1.00       238
                  HOMICIDE       1.00      1.00      1.00        61

                  accuracy                           0.99     20000
                 macro avg       0.97      0.97      0.97     20000
              weighted avg       0.99      0.99      0.99     20000
```




```
# Neural Network(Convolution Neural Network)
# Create Model with configuration
nn_model = MLPClassifier(solver='adam',
                         alpha=1e-5,
                         hidden_layer_sizes=(40,),
                         random_state=1,
                         max_iter=1000
                        )


# Model Training
```

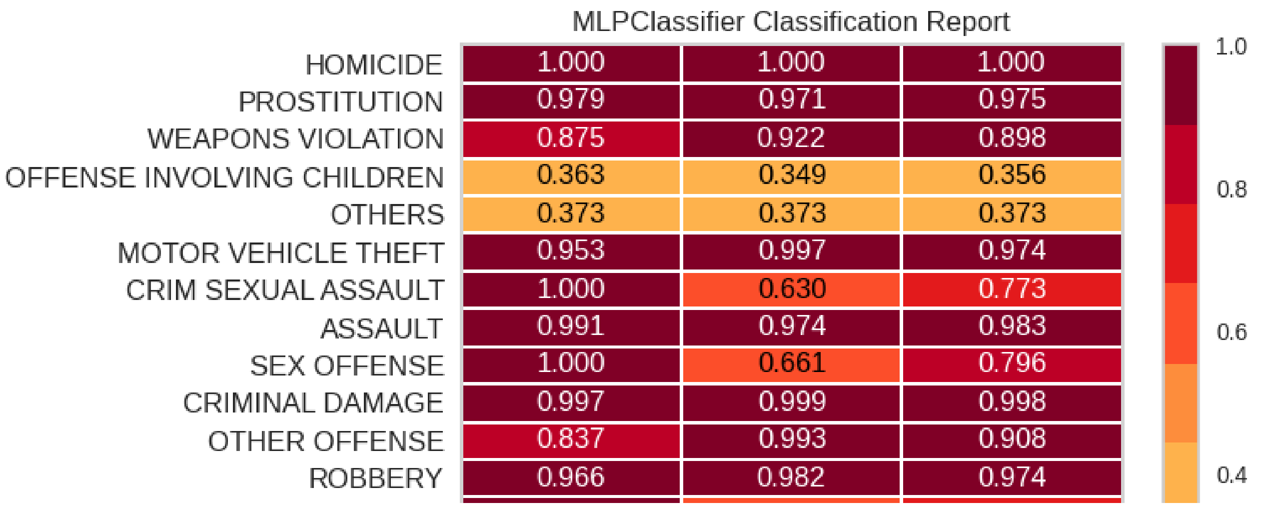
```
    nn_model.fit(X=x1,
                 y=x2)

    # Prediction
    result = nn_model.predict(y[Features])

    # Model Evaluation
    ac_sc = accuracy_score(y2, result)
    rc_sc = recall_score(y2, result, average="weighted")
    pr_sc = precision_score(y2, result, average="weighted")
    f1_sc = f1_score(y2, result, average='micro')
    confusion_m = confusion_matrix(y2, result)

    print("========== RCNN Neural Network Results ==========")
    print("Accuracy    : ", ac_sc)
    print("Recall      : ", rc_sc)
    print("Precision   : ", pr_sc)
    print("F1 Score    : ", f1_sc)
    print("Confusion Matrix: ")
    print(confusion_m)
```

```
========== RCNN Neural Network Results ==========
Accuracy    :  0.96945
Recall      :  0.96945
Precision   :  0.9715802635814953
F1 Score    :  0.96945
Confusion Matrix: 
[[3914    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0]
 [   0  523    0    0    0    0    0    0   74    0    0    0    0    0
     0    0    0    0    0]
 [   0    0 1895    0    0    0    0    0  107    0    0    0    0    0
     3    0    0    0    0]
 [   0    0    0  679    0    0    0    0    0    0    0    0    0   18
     0    0    4    5    0]
 [   0    0    0    0 3885    0    0   26    0    0    0    0    0    0
     3    0    0    0    0]
 [   0    0    0    0    0 1026    0    0    0    0    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    2    0  103    0   43    0    0    0    0    0
    18    0    0    0    0]
 [   0    0    0    0   14    0    0  748    0    0    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    0    0    0    0 1355    0    0    0    0    0
     1    8    0    0    0]
 [   0    0    0    0    0    0    0    0    0 2427    0    2    0    0
```

```
     0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0   39    0    0    0
    20    0    0    0    0]
 [   0    0    0    3    0    0    0    0    0    0    0 1219    0   11
    12    0    6    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    8   51    0
    22    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0  944
     3    0    0    0    0]
 [   0    0    0    0   18   10    0    0   11    8    0    0    0   11
    63   48    0    0    0]
 [   0    0    0    0    0    0    0    0   29    0    0    1    0    0
    24   37   15    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    7
     0    9  189    0    0]
 [   0    0    0    5    0    0    0    0    0    0    0    0    0    0
     0    0    2  231    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0   61]]
```

```
# Classification Report
# Instantiate the classification model and visualizer
target_names = Classes
visualizer = ClassificationReport(nn_model, classes=target_names)
visualizer.fit(X=x1, y=x2)     # Fit the training data to the visualizer
visualizer.score(y1, y2)       # Evaluate the model on the test data

print('================= Classification Report =================')
print('')
print(classification_report(y2, result, target_names=target_names))

g = visualizer.poof()             # Draw/show/poof the data
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but MLPClassifier was fitted with featur
  warnings.warn(
================= Classification Report =================

                           precision    recall  f1-score   support

                    THEFT       1.00      1.00      1.00      3914
        CRIMINAL TRESPASS       1.00      0.88      0.93       597
                NARCOTICS       1.00      0.95      0.97      2005
       DECEPTIVE PRACTICE       0.99      0.96      0.97       706
                  BATTERY       0.99      0.99      0.99      3914
                 BURGLARY       0.99      1.00      1.00      1026
    PUBLIC PEACE VIOLATION       1.00      0.62      0.77       166
                  ROBBERY       0.97      0.98      0.97       762
            OTHER OFFENSE       0.84      0.99      0.91      1364
          CRIMINAL DAMAGE       1.00      1.00      1.00      2429
              SEX OFFENSE       1.00      0.66      0.80        59
                  ASSAULT       0.99      0.97      0.98      1251
      CRIM SEXUAL ASSAULT       1.00      0.63      0.77        81
      MOTOR VEHICLE THEFT       0.95      1.00      0.97       947
                   OTHERS       0.37      0.37      0.37       169
OFFENSE INVOLVING CHILDREN       0.36      0.35      0.36       106
        WEAPONS VIOLATION       0.88      0.92      0.90       205
             PROSTITUTION       0.98      0.97      0.97       238
                 HOMICIDE       1.00      1.00      1.00        61

                 accuracy                           0.97     20000
                macro avg       0.91      0.86      0.88     20000
             weighted avg       0.97      0.97      0.97     20000
```



```
# K-Nearest Neighbors
# Create Model with configuration
knn_model = KNeighborsClassifier(n_neighbors=3)

# Model Training
knn_model.fit(X=x1,
             y=x2)

# Prediction
result = knn_model.predict(y[Features])
```

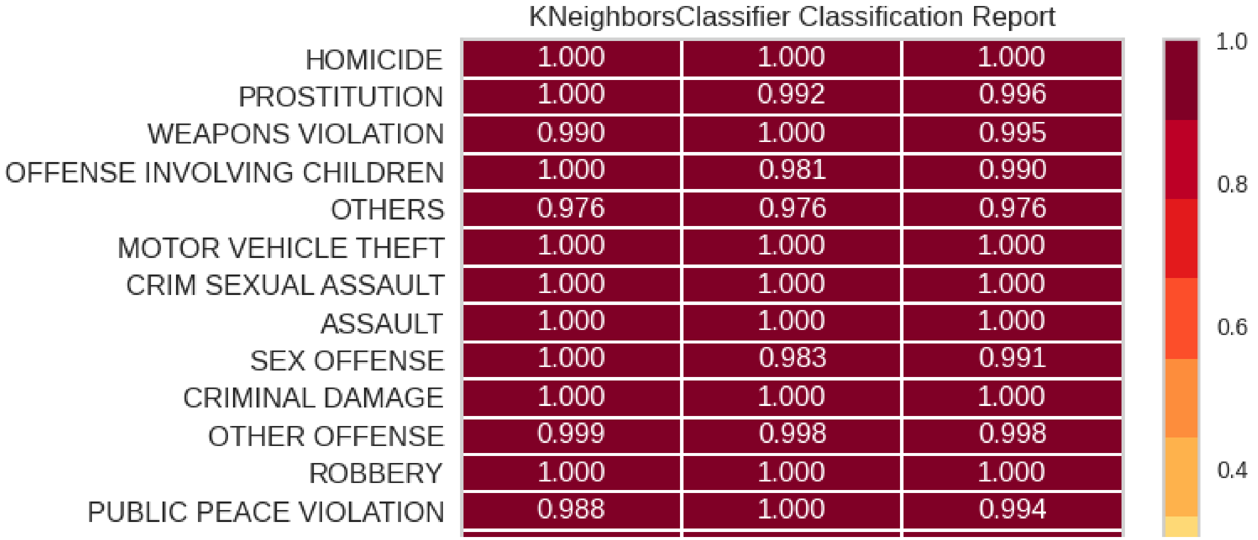
```
# Model Evaluation
ac_sc = accuracy_score(y2, result)
rc_sc = recall_score(y2, result, average="weighted")
pr_sc = precision_score(y2, result, average="weighted")
f1_sc = f1_score(y2, result, average='micro')
confusion_m = confusion_matrix(y2, result)

print("========== K-Nearest Neighbors Results ==========")
print("Accuracy    : ", ac_sc)
print("Recall      : ", rc_sc)
print("Precision   : ", pr_sc)
print("F1 Score    : ", f1_sc)
print("Confusion Matrix: ")
print(confusion_m)
```

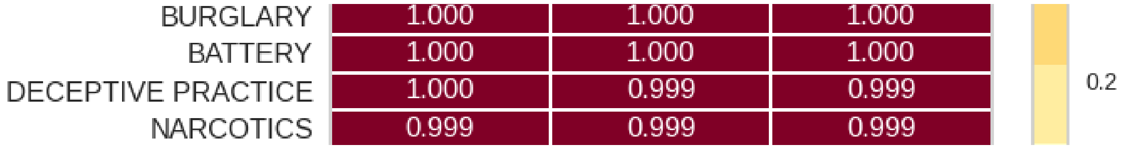
```
========== K-Nearest Neighbors Results ==========
Accuracy    :  0.9992
Recall      :  0.9992
Precision   :  0.9992021914153906
F1 Score    :  0.9992
Confusion Matrix:
[[3914    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0]
 [   0  597    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0]
 [   0    0 2003    0    0    0    0    0    1    0    0    0    0    0
     1    0    0    0    0]
 [   0    0    0  705    0    0    0    0    1    0    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0 3913    0    0    0    0    1    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    0 1026    0    0    0    0    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    0    0  166    0    0    0    0    0    0    0
     0    0    0    0    0]
```

```
 [   0    0    0    0    0    0    0  762    0    0    0    0    0    0
     0    0    0    0    0]
 [   1    0    1    0    0    0    0    0 1361    0    0    0    0    0
     1    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0 2429    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    1    0    0    0    0    0   58    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0 1251    0    0
     0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0   81    0
     0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0  947
     0    0    0    0    0]
 [   0    0    2    0    0    0    0    0    0    0    0    0    0    0
   165    0    2    0    0]
 [   0    0    0    0    0    0    2    0    0    0    0    0    0    0
     0  104    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0  205    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     2    0    0  236    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0   61]]
```

```
# Classification Report
# Instantiate the classification model and visualizer
target_names = Classes
visualizer = ClassificationReport(knn_model, classes=target_names)
visualizer.fit(X=x1, y=x2)     # Fit the training data to the visualizer
visualizer.score(y1, y2)       # Evaluate the model on the test data

print('================= Classification Report =================')
print('')
print(classification_report(y2, result, target_names=target_names))

g = visualizer.poof()             # Draw/show/poof the data
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted with fe
  warnings.warn(
================= Classification Report =================

                          precision    recall  f1-score   support

                   THEFT       1.00      1.00      1.00      3914
       CRIMINAL TRESPASS       1.00      1.00      1.00       597
               NARCOTICS       1.00      1.00      1.00      2005
      DECEPTIVE PRACTICE       1.00      1.00      1.00       706
                 BATTERY       1.00      1.00      1.00      3914
                BURGLARY       1.00      1.00      1.00      1026
   PUBLIC PEACE VIOLATION       0.99      1.00      0.99       166
                 ROBBERY       1.00      1.00      1.00       762
           OTHER OFFENSE       1.00      1.00      1.00      1364
         CRIMINAL DAMAGE       1.00      1.00      1.00      2429
             SEX OFFENSE       1.00      0.98      0.99        59
                 ASSAULT       1.00      1.00      1.00      1251
     CRIM SEXUAL ASSAULT       1.00      1.00      1.00        81
     MOTOR VEHICLE THEFT       1.00      1.00      1.00       947
                  OTHERS       0.98      0.98      0.98       169
OFFENSE INVOLVING CHILDREN       1.00      0.98      0.99       106
       WEAPONS VIOLATION       0.99      1.00      1.00       205
            PROSTITUTION       1.00      0.99      1.00       238
                HOMICIDE       1.00      1.00      1.00        61

                accuracy                           1.00     20000
               macro avg       1.00      1.00      1.00     20000
            weighted avg       1.00      1.00      1.00     20000
```






```
# Ensemble LSTM Model
import tensorflow as tf
from keras.layers import Dense, BatchNormalization, Dropout, LSTM, Bidirectional
from keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score, f1_score
from keras import callbacks
from tensorflow.keras.callbacks import EarlyStopping
# Combine 3 Models to create an Ensemble Model
```

```
# Define and compile model
from tensorflow import keras
model = keras.Sequential()
model.add(Dense(28 , input_shape=(56,) , activation="relu" , name="Hidden_Layer_1"))
model.add(Dense(10 , activation="relu" , name="Hidden_Layer_2"))
model.add(Dense(1 , activation="sigmoid" , name="Output_Layer"))
opt = keras.optimizers.Adam(learning_rate=0.01)
model.compile( optimizer=opt, loss="binary_crossentropy", metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Hidden_Layer_1 (Dense)      (None, 28)                1596

 Hidden_Layer_2 (Dense)      (None, 10)                290

 Output_Layer (Dense)        (None, 1)                 11

=================================================================
Total params: 1,897
Trainable params: 1,897
Non-trainable params: 0
_________________________________________________________________
```

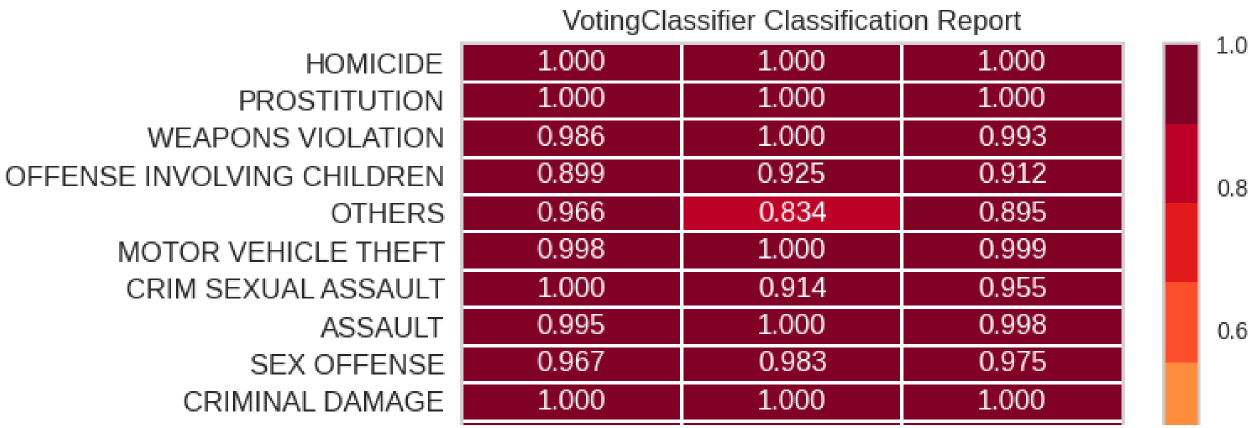
```
# Create Model with configuration
eclf1 = VotingClassifier(estimators=[('knn', knn_model), ('rf', rf_model), ('nn', nn_model)],
                         weights=[1,1,1],
                         flatten_transform=True)
eclf1 = eclf1.fit(X=x1, y=x2)

# Prediction
result = eclf1.predict(y[Features])


# Model Evaluation
ac_sc = accuracy_score(y2, result)
rc_sc = recall_score(y2, result, average="weighted")
pr_sc = precision_score(y2, result, average="weighted")
f1_sc = f1_score(y2, result, average='micro')
confusion_m = confusion_matrix(y2, result)

print("============= LSTM Results =============")
print("Accuracy    : ", ac_sc)
print("Recall      : ", rc_sc)
print("Precision   : ", pr_sc)
print("F1 Score    : ", f1_sc)
print("Confusion Matrix: ")
print(confusion_m)
```

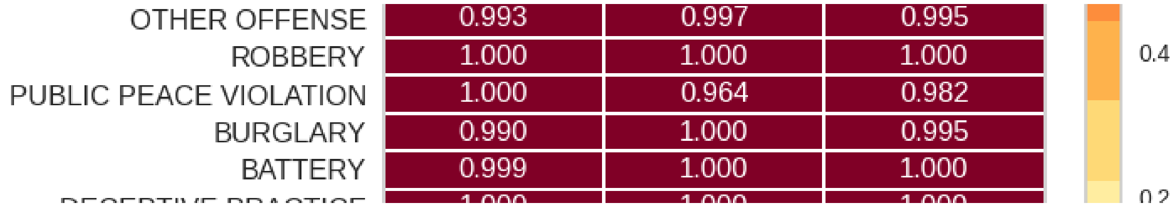
```
============= LSTM Results =============
Accuracy    :  0.9971
Recall      :  0.9971
Precision   :  0.9970885706244451
F1 Score    :  0.9971
Confusion Matrix:
[[3914    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0]
 [   0  594    0    0    0    0    0    0    3    0    0    0    0    0
     0    0    0    0    0]
 [   0    0 2004    0    0    0    0    0    0    0    0    0    0    0
     1    0    0    0    0]
 [   0    0    0  706    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0 3914    0    0    0    0    0    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    0 1026    0    0    0    0    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    0    0  160    0    6    0    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    0    0    0  762    0    0    0    0    0    0
     0    0    0    0    0]
```

```
 [   1    0    1    0    0    0    0    0 1360    0    0    0    0    0
     2    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0 2429    0    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    1    0    0    0    0    0   58    0    0    0
     0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0 1251    0    0
     0    0    0    0    0]
 [   0    0    0    0    2    0    0    0    0    0    0    5   74    0
     0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0  947
     0    0    0    0    0]
 [   0    0    5    0    0   10    0    0    0    0    0    0    0    2
   141   11    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    2    1    0    0
     2   98    3    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0  205    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0  238    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0   61]]
```

```
# Classification Report
# Instantiate the classification model and visualizer
target_names = Classes
visualizer = ClassificationReport(eclf1, classes=target_names)
visualizer.fit(X=x1, y=x2)     # Fit the training data to the visualizer
visualizer.score(y1, y2)       # Evaluate the model on the test data

print('================ Classification Report ================')
print('')
print(classification_report(y2, result, target_names=target_names))

g = visualizer.poof()             # Draw/show/poof the data
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted with fe
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but MLPClassifier was fitted with feature r
  warnings.warn(
================= Classification Report =================

                            precision    recall  f1-score   support

                     THEFT       1.00      1.00      1.00      3914
         CRIMINAL TRESPASS       1.00      0.99      1.00       597
                 NARCOTICS       1.00      1.00      1.00      2005
        DECEPTIVE PRACTICE       1.00      1.00      1.00       706
                   BATTERY       1.00      1.00      1.00      3914
                  BURGLARY       0.99      1.00      1.00      1026
      PUBLIC PEACE VIOLATION     1.00      0.96      0.98       166
                   ROBBERY       1.00      1.00      1.00       762
             OTHER OFFENSE       0.99      1.00      1.00      1364
           CRIMINAL DAMAGE       1.00      1.00      1.00      2429
               SEX OFFENSE       0.97      0.98      0.97        59
                   ASSAULT       1.00      1.00      1.00      1251
        CRIM SEXUAL ASSAULT       1.00      0.91      0.95        81
        MOTOR VEHICLE THEFT       1.00      1.00      1.00       947
                    OTHERS       0.97      0.83      0.90       169
OFFENSE INVOLVING CHILDREN       0.90      0.92      0.91       106
          WEAPONS VIOLATION      0.99      1.00      0.99       205
              PROSTITUTION       1.00      1.00      1.00       238
                  HOMICIDE       1.00      1.00      1.00        61

                  accuracy                           1.00     20000
                 macro avg       0.99      0.98      0.98     20000
              weighted avg       1.00      1.00      1.00     20000
```






```
from google.colab import files
uploaded = files.upload()
```

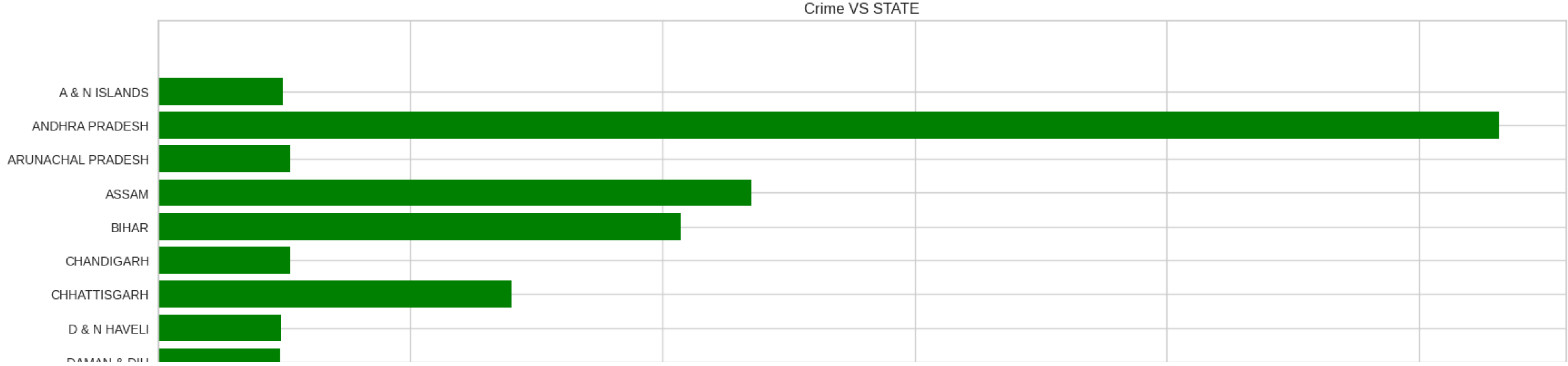
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

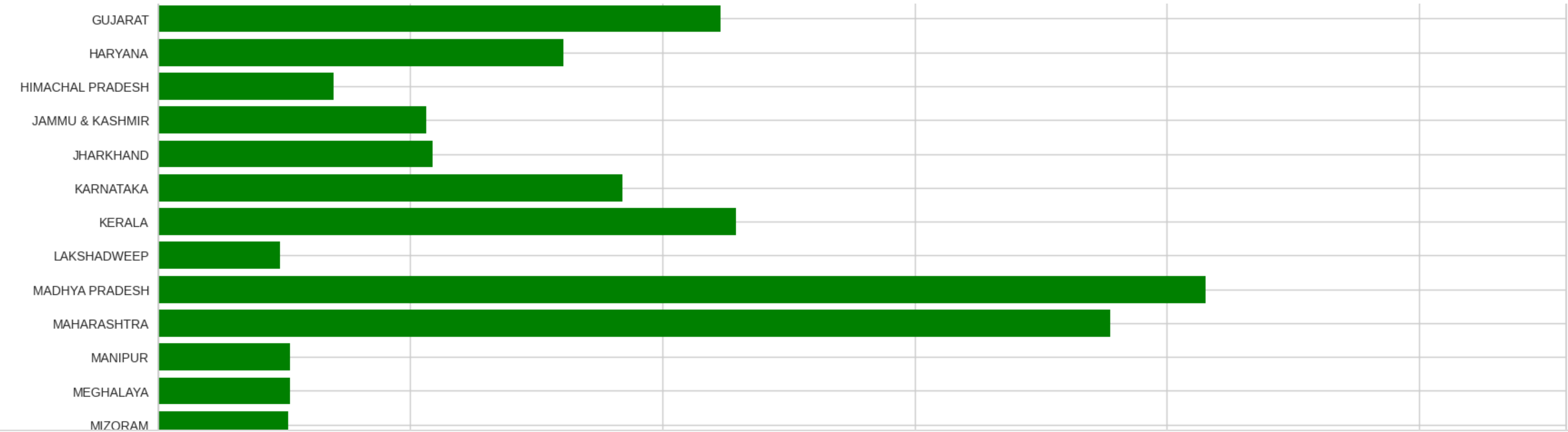
```
Saving District_wise_crimes2022.csv to District_wise_crimes2022.csv
Saving District2001_2020.csv to District2001_2020.csv
```

```
import numpy as np
crimes_total_women1 = pd.read_csv('District2001_2020.csv')
crimes_total_women2= pd.read_csv('District_wise_crimes2022.csv')

crimes_total_women = pd.concat([crimes_total_women1,crimes_total_women2],  ignore_index=False, axis=0)
crimes_total_women.rename(columns={'STATE/UT':'STATE'}, inplace=True)

del crimes_total_women1
del crimes_total_women2

# calculating total crimes of all kinds in each state from 2001 to 2013
crimes_total_women = crimes_total_women[crimes_total_women['DISTRICT'] == 'TOTAL']
crimes_total_women.drop('DISTRICT', axis=1, inplace=True)

crimes_total_women['Total Crimes']= crimes_total_women.iloc[:, -9:-1].sum(axis=1)

crimes_total_women = crimes_total_women.groupby(['STATE'])['Total Crimes'].sum()

# plot graph of crimes committed on women since 2001-2013 in each state/ UT
fig1, ax1 = plt.subplots()
states = crimes_total_women.index.tolist()
y_pos = np.arange(len(states))
performance = crimes_total_women.tolist()
ax1.barh(y_pos, performance, align='center',color='green', ecolor='black')
ax1.set_yticks(y_pos)
ax1.set_yticklabels(states)
ax1.invert_yaxis()  # labels read top-to-bottom
ax1.set_xlabel('Overall districtwise Crimerate')
ax1.set_title('Crime VS STATE')
```

```
fig1.set_size_inches(20, 18, forward=True)
plt.show()
```

```
<ipython-input-35-c1f36bf0c25e>:15: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a fut
  crimes_total_women['Total Crimes']= crimes_total_women.iloc[:, -9:-1].sum(axis=1)
```





✓ 0s completed at 1:15 PM