CHICAGO CRIME DATA ANALYSIS

Submitted By

Team - 04

Bhashitha Siddareddy

Lalith Chandra Attaluri

Udaya Byreddy

Team Contribution:

Member Name	Email	Class ID	Contribution
Bhashitha Siddareddy	bstkg@mail.umkc.edu	04	Machine Learning model Analysis, Model evaluation, Voting Classifier model, Front end Development
Lalith Chandra Attaluri	la4kf@mail.umkc.edu	11	Data cleaning, Geographical Visualization, Classification Report, MLP Classifier Model, Documentation
Udaya Byredy	ubx35@mail.umkc.edu	25	Bar graph Visualization, Documentation, Random Forest Classifier Model, Power point presentation

Technologies Used:

Jupyter Notebook

Approach:

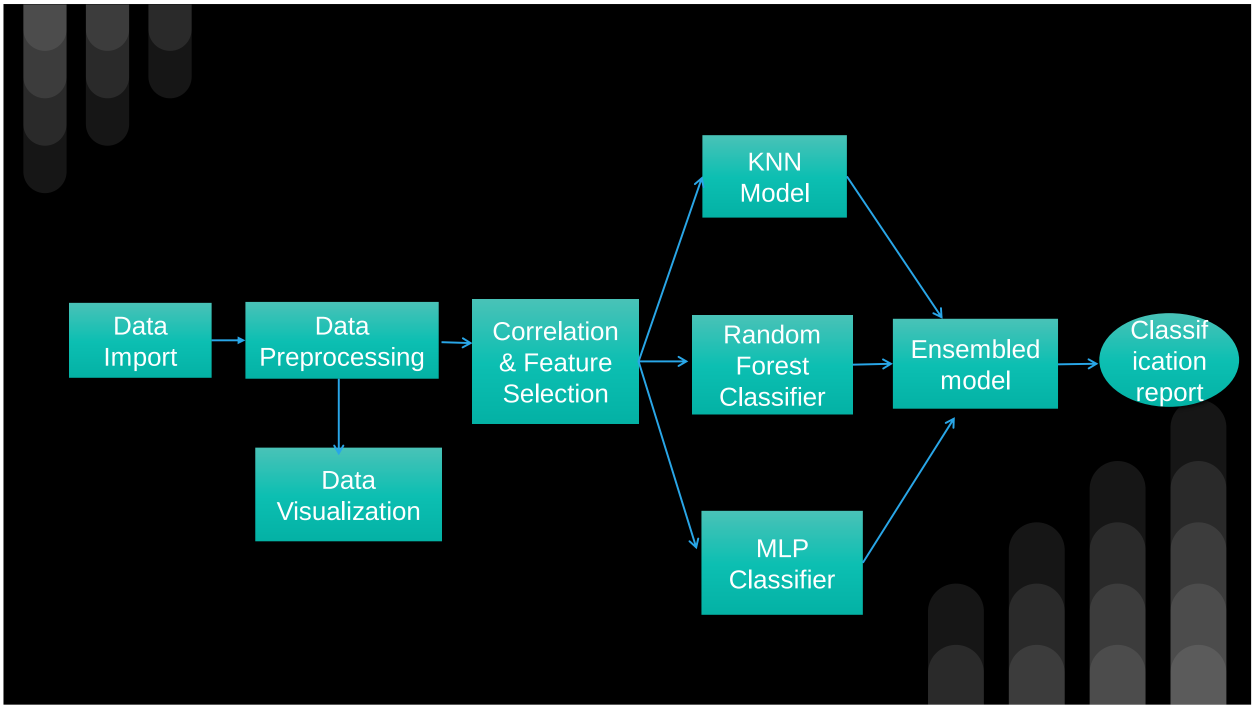
- We have chosen Chicago crimes dataset that shows the crime records over the span of 16 years from 2001 to 2017.
- We have implemented ***Classification*** approach in ***Supervised Learning*** to classify the crimes based on their ***primary type.***

Description of Data Set:

The dataset we used is "Chicago Crimes Dataset" that comprises of reported incidents of crime (with the exception of murders where data exists for each victim) that occurred in the City of Chicago from 2001 to 2017.
Please find the description of each column below

- **ID** - Unique identifier for the record
- **Case Number** - The Chicago Police Department RD Number which is unique to the incident
- **Date** - Date when the incident occurred
- **Block** - The partially redacted address where the incident occurred, placing it on the same block as the actual address
- **IUCR** - The Illinois Uniform Crime Reporting code. This is directly linked to the Primary Type and Description.
- **Primary Type** - The primary description of the IUCR code.
- **Description** - The secondary description of the IUCR code, a subcategory of the primary description.
- **Location Description** - Description of the location where the incident occurred.
- **Arrest** - Indicates whether an arrest was made.
- **Domestic** - Indicates whether the incident was domestic related as defined by the Illinois Domestic Violence Act.
- **District** - Indicates the police district where the incident occurred.
- **Ward** - The ward (City Council district) where the incident occurred.
- **Community Area** - Indicates the community area where the incident occurred. Chicago has 77 community areas.
- **FBI Code** - Indicates the crime classification as outlined in the FBI's National Incident-Based Reporting System.
- **X Coordinate** - The x coordinate of the location where the incident occurred in State Plane Illinois East NAD 1983 projection.
- **Y Coordinate** - The y coordinate of the location where the incident occurred in State Plane Illinois East NAD 1983 projection.
- **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.
- **Longitude** - The longitude of the location where the incident occurred.
- **Location** - The location where the incident occurred in a format that allows for creation of maps and other geographic operations on this data portal.

Architecture:



Algorithm Flow:

- Data Import
- Data Preprocessing
- Data Visualization
- Correlation & Feature Selection
- Implementing Machine leaning, Neural networks & Ensembled model
- Extraction of classification report

Algorithm:

1. **Data Import:**
   Initially we have imported the dataset that comprises of 4 csv files. As the dataset is too large for fast processing we have limited the records in each csv file to 50K samples. Hence our data consists of 200K samples.

```
#Converting our multiple CSV files to dataframes
crimes_2005_2007=pd.read_csv('Chicago_Crimes_2005_to_2007.csv',error_bad_lines=False)
crimes_2001_2004=pd.read_csv('Chicago_Crimes_2001_to_2004.csv',error_bad_lines=False)
crimes_2008_2011=pd.read_csv('Chicago_Crimes_2008_to_2011.csv',error_bad_lines=False)
crimes_2012_2017=pd.read_csv('Chicago_Crimes_2012_to_2017.csv',error_bad_lines=False)
#Sampling each data frame to 50K samples
crimes_2001_2004 = crimes_2001_2004.sample(n=50000)
crimes_2005_2007 = crimes_2005_2007.sample(n=50000)
crimes_2008_2011 = crimes_2008_2011.sample(n=50000)
crimes_2012_2017 = crimes_2012_2017.sample(n=50000)
#Concatenating all the dataframes to a single dataframe
data_frames=[crimes_2001_2004, crimes_2005_2007, crimes_2008_2011, crimes_2012_2017]
crimes=pd.concat(data_frames)
```

Please find the dataframe description below

jupyter Source_code_Increment1 Last Checkpoint: 3 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
In [72]: crimes.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 199182 entries, 2004-11-13 01:23:04 to 2013-10-22 11:47:00
Data columns (total 23 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Unnamed: 0            199182 non-null  int64
 1   ID                    199182 non-null  int64
 2   Case Number           199182 non-null  object
 3   Date                  199182 non-null  datetime64[ns]
 4   Block                 199182 non-null  object
 5   IUCR                  199182 non-null  object
 6   Primary Type          199182 non-null  object
 7   Description           199182 non-null  object
 8   Location Description  199124 non-null  object
 9   Arrest                199182 non-null  bool
 10  Domestic              199182 non-null  bool
 11  Beat                  199182 non-null  int64
 12  District              199181 non-null  float64
 13  Ward                  181091 non-null  float64
 14  Community Area        181055 non-null  float64
 15  FBI Code              199182 non-null  object
 16  X Coordinate          196331 non-null  float64
 17  Y Coordinate          196331 non-null  object
 18  Year                  199182 non-null  float64
 19  Updated On            199182 non-null  object
 20  Latitude              196331 non-null  object
 21  Longitude             196331 non-null  float64
 22  Location              196331 non-null  object
dtypes: bool(2), datetime64[ns](1), float64(6), int64(3), object(11)
memory usage: 33.8+ MB
```

2. Data Preprocessing:

- Initially we have removed the duplicate records in our dataset.

```
#Removing Duplicate Records
print('Dataset ready..')
print('Dataset Shape before drop_duplicate : ', crimes.shape)
crimes.drop_duplicates(subset=['ID', 'Case Number'], inplace=True)
print('Dataset Shape after drop_duplicate: ', crimes.shape)

Dataset ready..
Dataset Shape before drop_duplicate :  (200000, 23)
Dataset Shape after drop_duplicate:  (199080, 23)
```

- Calculating the percentage of null values in our dataset

```
#Calaculationg percentage of Null values
percent_missing = crimes.isnull().sum()/ len(crimes) * 100
percent_missing
```

```
Unnamed: 0              0.000000
ID                      0.000000
Case Number             0.000000
Date                    0.000000
Block                   0.000000
IUCR                    0.000000
Primary Type            0.000000
Description             0.000000
Location Description    0.026120
Arrest                  0.000000
Domestic                0.000000
Beat                    0.000000
District                0.001507
Ward                    9.166164
Community Area          9.184248
FBI Code                0.000000
X Coordinate            1.445148
Y Coordinate            1.445148
Year                    0.000000
Updated On              0.000000
Latitude                1.445148
Longitude               1.445148
Location                1.445148
dtype: float64
```

- Dropping null values from the dataset.

```
#Dropping Null values
crimes = crimes.dropna()
crimes.isnull().sum()
```

```
Unnamed: 0              0
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
```

- The final step our data preprocessing is to replace the infrequent columns in Description and Location Description column with OTHERS. Here we have decided the category as infrequent if it is repeated in our dataset less than 20 times.

```
#Transforming the least used cateogiries to a single cateogory "OTHER"
loc_to_change  = list(crimes['Location Description'].value_counts()[20:].index)
crimes.loc[crimes['Location Description'].isin(loc_to_change) , crimes.columns=='Location Description'] = 'OTHER'
desc_to_change = list(crimes['Description'].value_counts()[20:].index)
crimes.loc[crimes['Description'].isin(desc_to_change) , crimes.columns=='Description'] = 'OTHER'
```

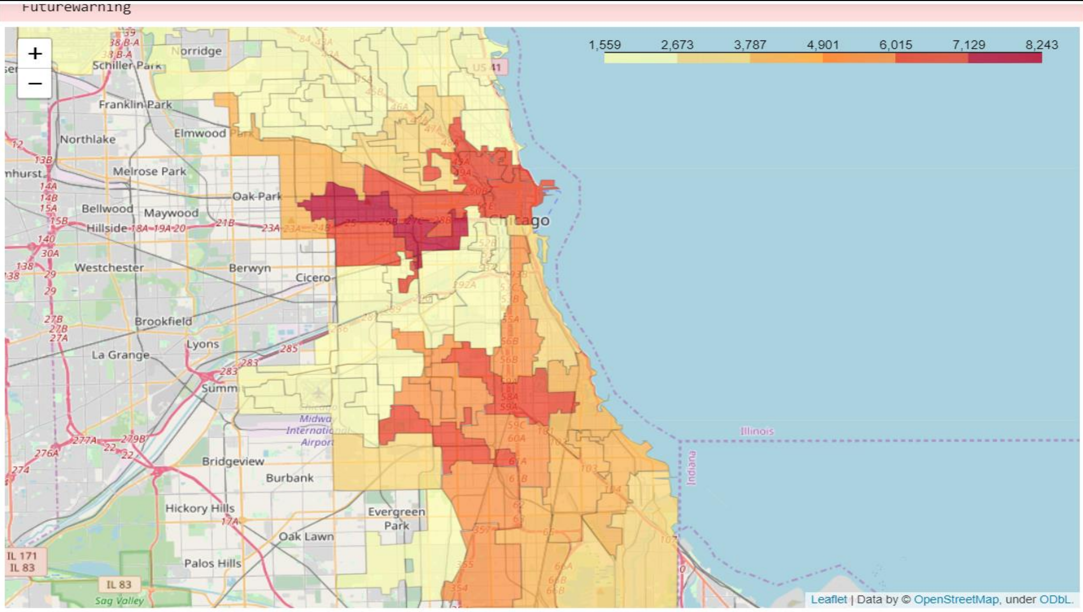
3. Data Visualization:

- **Visualizing crime data at ward level:**

```
#Plotting geographical map of crimes commited with respect to wards
#definition of the boundaries in the map
district_geo = r'Boundaries-Wards.geojson'
Chicago_COORDINATES = (41.895140898, -87.624255632)
import folium
#calculating total number of incidents per district for 2016
WardData2016 = pd.DataFrame(crimes['ward'].value_counts().astype(float))
WardData2016.to_json('Ward_Map.json')
WardData2016 = WardData2016.reset_index()
WardData2016.columns = ['ward', 'Crime_Count']

#creating choropleth map for Chicago District 2016
map1 = folium.Map(location=Chicago_COORDINATES, zoom_start=11)
map1.choropleth(geo_data = district_geo,
                #data_out = 'Ward_Map.json',
                data = WardData2016,
                columns = ['ward', 'Crime_Count'],
                key_on = 'feature.properties.ward',
                fill_color = 'YlOrRd',
                fill_opacity = 0.7,
                line_opacity = 0.2)
map1
```

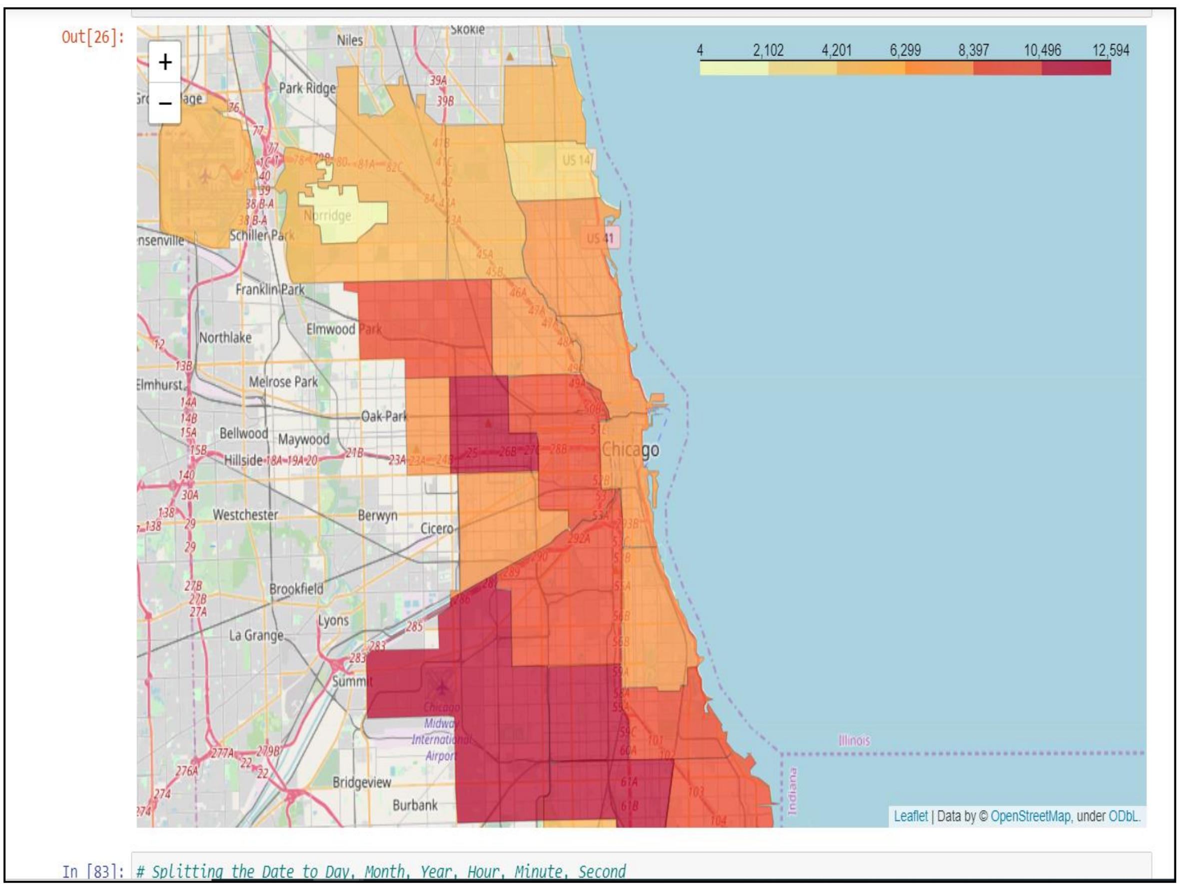
FutureWarning

Out[82]:



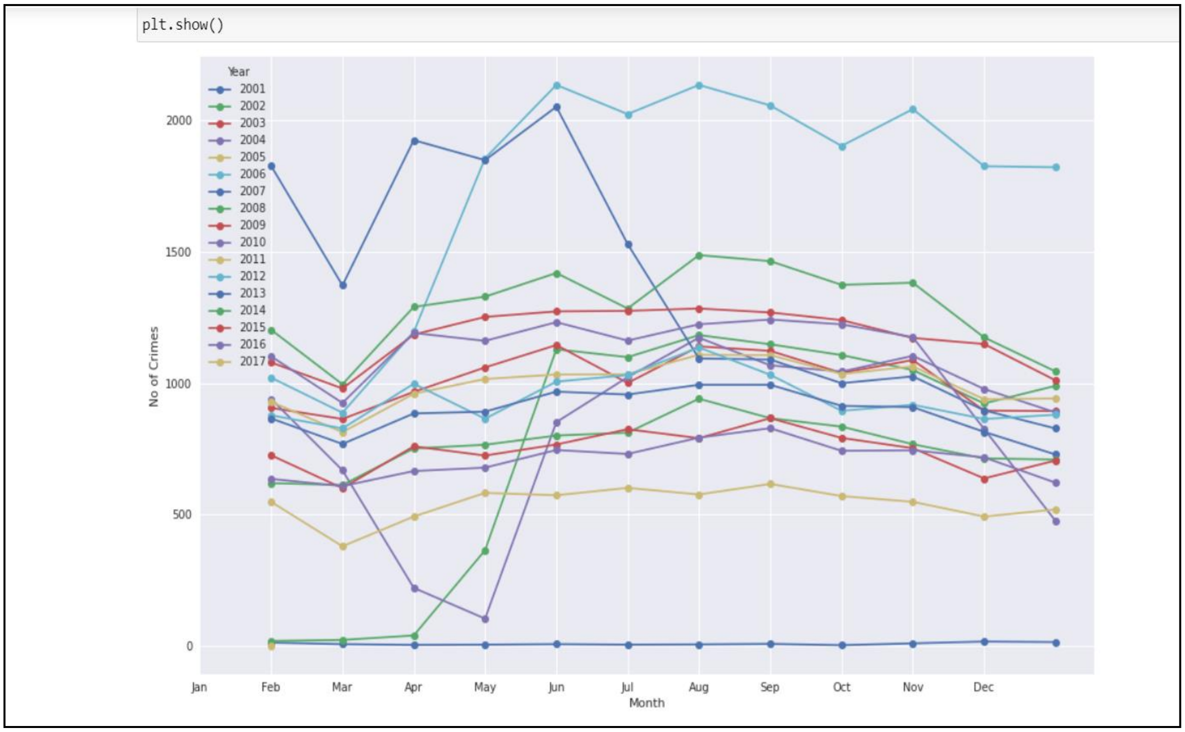
- **Visualizing crime data at district level:**

```
#Plotting geographicl map of Crimes with respect to Districts
#definition of the boundaries in the map
district_geo = r'Boundaries_Police_Districts.geojson'

district_data = pd.DataFrame(crimes['district'].value_counts().astype(float))
district_data.to_json('District_Map.json')
district_data = district_data.reset_index()
district_data.columns = ['district', 'Crime_Count']

#creation of the choropleth
map2 = folium.Map(location=Chicago_COORDINATES, zoom_start=11)
map2.choropleth(geo_data = district_geo,
                data = district_data,
                columns = ['district', 'Crime_Count'],
                key_on = "feature.properties.dist_num",
                fill_color = 'YlOrRd',
                fill_opacity = 0.7,
                line_opacity = 0.2)
map2
```



- **Visualization of monthly crime data over different years**

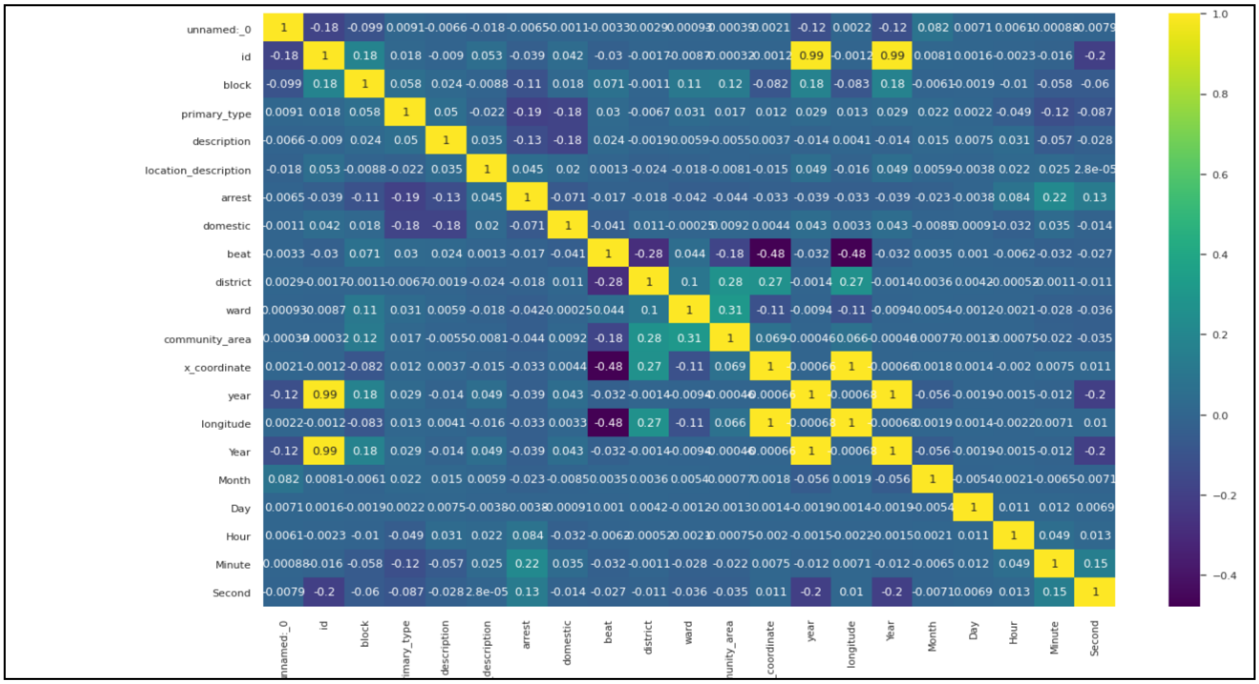
```
#Plotting monthly crimes commited every year
crimes.groupby(['Month','Year'])['id'].count().unstack().plot(marker='o', figsize=(15,10))
months=['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']
plt.xticks(np.arange(12),months)
plt.ylabel('No of Crimes')

plt.show()
```



4. Correlation and selecting relevant features:

- Generating heat map to find the correlation between features

```
#Finding Correlation
x = crimes.drop(['primary_type'], axis=1)
y = crimes['primary_type']
import seaborn as sns; sns.set(color_codes=True)
plt.figure(figsize=(20,12))
correlation = crimes.corr()
sns.heatmap(correlation, annot=True, cmap='viridis')
plt.show()
```



- **Finding the correlation with target**

In [97]:
```python
#Finding Correlation with Target
correlation_target=abs(correlation['primary_type'])
print(correlation_target)
```

```
unnamed:_0              0.009119
id                      0.018411
block                   0.057744
primary_type            1.000000
description             0.050013
location_description    0.022171
arrest                  0.194701
domestic                0.180980
beat                    0.030042
district                0.006689
ward                    0.030571
community_area          0.017368
x_coordinate            0.012311
year                    0.029319
longitude               0.012795
Year                    0.029319
Month                   0.022015
Day                     0.002208
Hour                    0.049076
Minute                  0.122226
Second                  0.086636
Name: primary_type, dtype: float64
```

- The features that have correlation with target > 0.1 as relevant features which are **Description and Arrest**

```
#Selecting highly correlated features
relevant_features = correlation_target[correlation_target>0.1]
relevant_features
```

```
primary_type    1.0
Name: primary_type, dtype: float64
```

```
features=["description", "arrest"]
print('The features that are more correlated to the model are: ', features)
```

```
The features that are more correlated to the model are:  ['description', 'arrest']
```

5. Implementing KNN Model:

- Splitting the data into test and training data

```
#Splitting data into train and test
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(crimes ,test_size=0.4, random_state=0)
```

```
target='primary_type'
x_train=train_data[features]
y_train=train_data[target]
x_test=test_data[features]
y_test=test_data[target]
```

- Building the model and making predictions.

```
#K-nearest neighbours Model
from sklearn.neighbors import KNeighborsClassifier
model_knn=KNeighborsClassifier(n_neighbors=4)
model_knn.fit(x_train,y_train)
#Predicting the result
predicted_result=model_knn.predict(x_test)
# Evaluating the model
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score, f1_score
accuracy = accuracy_score(y_test, predicted_result)
recall = recall_score(y_test, predicted_result, average="weighted")
percision = precision_score(y_test, predicted_result, average="weighted")
f1score = f1_score(y_test, predicted_result, average='micro')
confusionmatrix = confusion_matrix(y_test, predicted_result)

print("========== Evaluation results of KNN Model ==========")
print("Accuracy    : ", accuracy)
print("Recall      : ", recall)
print("Precision   : ", percision)
print("F1 Score    : ", f1score)
print("Confusion Matrix: ")
print(confusionmatrix)
```

- **Evaluation Report:**

```
print(confusionmatrix)

========== Evaluation results of KNN Model ==========
Accuracy    :  0.7278851540616247
Recall      :  0.7278851540616247
Precision   :  0.7788827284133902
F1 Score    :  0.7278851540616247
Confusion Matrix:
[[ 7942     0     0     0     0     0     0     0    20     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [  405  2460   805     0     0     0     0     0   670     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [  505  3851  7633     0     0     0     0     0  1010     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [  468     0     0  1277   109     0     0     0   236     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [  125     0     0     0  7517     0     0     0   534     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [  713     0     0     0     0  2530     0     0  1160     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [   66     0     0     0     0     0 13450     0  1295     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [  446     0     0     0     0     0     0     0  2109     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [  102     0     0     0     0     0     0     0  2722     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [  106     0     0     0     0     0     0     0   663  2444     0     0
      0     0     0     0     0     0     0     0     0     0]
 [   41     0     2     0     0     0     0     0   213     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [   21     0     0     0     0     0     0     0   200     0     0  3996
      0     0     0     0     0     0     0     0     0     0]
 [  720     0     0     0     0     0     0     0     1     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [   95     0     0     0     0     0     0     0   391     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [  131     0     0     0     0     0     0     0     1     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [  408     0     0     0     0     0     0     0   168     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [  555     0     0     0     0     0     0     0   157     0     0     0
      0     0     0     0     0     0     0     0     0     0]
 [   75     0     0     0     0     0     0     0   164     0     0     0
```

- **Classification report:**

```
Visualizer score is:  0.7278851540616247
                                 precision    recall  f1-score   support

                     NARCOTICS       0.60      1.00      0.75      7962
                       ASSAULT       0.39      0.57      0.46      4340
                       BATTERY       0.90      0.59      0.71     12999
             CRIMINAL TRESPASS       1.00      0.61      0.76      2090
               CRIMINAL DAMAGE       0.99      0.92      0.95      8176
                 OTHER OFFENSE       1.00      0.57      0.73      4403
                         THEFT       1.00      0.91      0.95     14811
            DECEPTIVE PRACTICE       0.00      0.00      0.00      2555
                       ROBBERY       0.23      0.96      0.37      2824
           MOTOR VEHICLE THEFT       1.00      0.76      0.86      3213
           CRIM SEXUAL ASSAULT       0.00      0.00      0.00       256
                      BURGLARY       1.00      0.95      0.97      4217
                  PROSTITUTION       0.00      0.00      0.00       721
    OFFENSE INVOLVING CHILDREN       0.00      0.00      0.00       486
          LIQUOR LAW VIOLATION       0.00      0.00      0.00       132
          PUBLIC PEACE VIOLATION       0.00      0.00      0.00       576
             WEAPONS VIOLATION       0.00      0.00      0.00       712
                   SEX OFFENSE       0.00      0.00      0.00       239
                         ARSON       0.00      0.00      0.00       120
                      GAMBLING       0.00      0.00      0.00       186
                        OTHERS       0.00      0.00      0.00       222
INTERFERENCE WITH PUBLIC OFFICER       0.00      0.00      0.00       160

                      accuracy                           0.73     71400
                     macro avg       0.37      0.36      0.34     71400
                  weighted avg       0.78      0.73      0.73     71400
```

6. Implementing Random forest Classifier:

- Building the model and making predictions

```python
from sklearn.ensemble import RandomForestClassifier
#Using Random forest Classifier
model_rforest = RandomForestClassifier(n_estimators=70, # Number of trees
                                  min_samples_split = 30,
                                  bootstrap = True,
                                  max_depth = 50,
                                  min_samples_leaf = 25)
# Training the data
model_rforest.fit(x_train, y_train)
# Predicting the result using test data
predicted_result = model_rforest.predict(x_test)
# Evaluating the model
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score, f1_score
accuracy = accuracy_score(y_test, predicted_result)
recall = recall_score(y_test, predicted_result, average="weighted")
percision = precision_score(y_test, predicted_result, average="weighted")
f1score = f1_score(y_test, predicted_result, average='micro')
confusionmatrix = confusion_matrix(y_test, predicted_result)

print("========== Evaluation results of Random Forest Classifier Model ==========")
print("Accuracy    : ", accuracy)
print("Recall      : ", recall)
print("Precision   : ", percision)
print("F1 Score    : ", f1score)
print("Confusion Matrix: ")
print(confusionmatrix)
#Classification report
target_names = a
visualizer = ClassificationReport(model_rforest, classes=target_names, size=(1080, 720))
visualizer.fit(X=x_train, y=y_train)     # Fit the training data to the visualizer
print("Visualizer score is: ",visualizer.score(x_test, y_test))      # Evaluate the model on the test data
print(classification_report(y_test, predicted_result,target_names=a))
g = visualizer.poof()
```

- **Evaluation Report:**

```
========== Evaluation results of Random Forest Classifier Model ==========
Accuracy    :  0.766769532893814
Recall      :  0.766769532893814
Precision   :  0.7695011493776878
F1 Score    :  0.766769532893814
Confusion Matrix:
[[ 7904     0     0     0     0     0     0     0     0     0     0    23
      0     0     0     0     0     0     0     0     0]
 [  479 11374     0     0     0     0     0     0     0     0     0  1047
      0     0     0     0     0     0     0     0     0]
 [   36     0  4009     0     0     0     0     0     0     0     0   201
      0     0     0     0     0     0     0     0     0]
 [   52     0     0 13388     0     0     0     0     0     0     0  1361
      0     0     0     0     0     0     0     0     0]
 [   82     0     0     0  2551     0     0     0     0     0     0   629
      0     0     0     0     0     0     0     0     0]
 [   87     0     0     0     0  1919     0     0     0     0     0   716
      0     0     0     0     0     0     0     0     0]
 [  454     0     0     0     0     0  1270     0   113     0     0   207
      0     0     0     0     0     0     0     0     0]
 [  410  3284     0     0     0     0     0     0     0     0     0   657
      0     0     0     0     0     0     0     0     0]
 [  132     0     0     0     0     0     0     0  7575     0     0   520
      0     0     0     0     0     0     0     0     0]
 [  716     0     0     0     0     0     0     0     0  2514     0  1165
      0     0     0     0     0     0     0     0     0]
```

- **Classification report:**

```
Visualizer score is:  0.7666993968298499
                                 precision    recall  f1-score   support

                     NARCOTICS       0.60      1.00      0.75      7927
                       BATTERY       0.77      0.88      0.82     12900
                      BURGLARY       1.00      0.94      0.97      4246
                         THEFT       1.00      0.90      0.95     14801
           MOTOR VEHICLE THEFT       1.00      0.78      0.88      3262
                       ROBBERY       1.00      0.70      0.83      2722
             CRIMINAL TRESPASS       1.00      0.62      0.77      2044
                       ASSAULT       0.00      0.00      0.00      4351
               CRIMINAL DAMAGE       0.99      0.92      0.95      8227
                 OTHER OFFENSE       1.00      0.57      0.73      4395
                  PROSTITUTION       0.00      0.00      0.00       707
            DECEPTIVE PRACTICE       0.22      0.83      0.34      2604
                   SEX OFFENSE       0.00      0.00      0.00       238
     OFFENSE INVOLVING CHILDREN       0.00      0.00      0.00       463
              WEAPONS VIOLATION       0.00      0.00      0.00       709
INTERFERENCE WITH PUBLIC OFFICER       0.00      0.00      0.00       160
           CRIM SEXUAL ASSAULT       0.00      0.00      0.00       271
                      GAMBLING       0.00      0.00      0.00       165
                        OTHERS       0.00      0.00      0.00       352
          PUBLIC PEACE VIOLATION       0.00      0.00      0.00       573
            LIQUOR LAW VIOLATION       0.00      0.00      0.00       173

                      accuracy                           0.77     71290
                     macro avg       0.41      0.39      0.38     71290
                  weighted avg       0.77      0.77      0.75     71290
```

7. Implementing Neural Network model (MLP Classifier):

- **Building the model**

```
#Model with Neural networks
from sklearn.neural_network import MLPClassifier
neural_model = MLPClassifier(solver='adam', alpha=1e-5, hidden_layer_sizes=(100,100,100), activation='relu', random_state=1 ,
# Training the model with the data
neural_model.fit(x_train,y_train)
# Predicting the result using test data
predicted_result = neural_model.predict(x_test)
# Evaluating the model
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score, f1_score
accuracy = accuracy_score(y_test, predicted_result)
recall = recall_score(y_test, predicted_result, average="weighted")
percision = precision_score(y_test, predicted_result, average="weighted")
f1score = f1_score(y_test, predicted_result, average='micro')
confusionmatrix = confusion_matrix(y_test, predicted_result)

print("========== Evaluation results of MLP Classifier Model ==========")
print("Accuracy    : ", accuracy)
print("Recall      : ", recall)
print("Precision   : ", percision)
print("F1 Score    : ", f1score)
print("Confusion Matrix: ")
print(confusionmatrix)
target_names = a
visualizer = ClassificationReport(neural_model, classes=target_names, size=(1080, 720))
visualizer.fit(X=x_train, y=y_train)     # Fit the training data to the visualizer
print("Visualizer score is: ",visualizer.score(x_test, y_test))      # Evaluate the model on the test data
print(classification_report(y_test, predicted_result,target_names=a))
g = visualizer.poof()
```

- **Evaluation Report:**

```
========== Evaluation results of MLP Classifier Model ==========
Accuracy    :  0.766769532893814
Recall      :  0.766769532893814
Precision   :  0.7695011493776878
F1 Score    :  0.766769532893814
Confusion Matrix:
[[ 7904     0     0     0     0     0     0     0     0     0     0    23
      0     0     0     0     0     0     0     0     0]
 [  479 11374     0     0     0     0     0     0     0     0     0  1047
      0     0     0     0     0     0     0     0     0]
 [   36     0  4009     0     0     0     0     0     0     0     0   201
      0     0     0     0     0     0     0     0     0]
 [   52     0     0 13388     0     0     0     0     0     0     0  1361
      0     0     0     0     0     0     0     0     0]
 [   82     0     0     0  2551     0     0     0     0     0     0   629
      0     0     0     0     0     0     0     0     0]
 [   87     0     0     0     0  1919     0     0     0     0     0   716
      0     0     0     0     0     0     0     0     0]
 [  454     0     0     0     0     0  1270     0   113     0     0   207
      0     0     0     0     0     0     0     0     0]
 [  410  3284     0     0     0     0     0     0     0     0     0   657
      0     0     0     0     0     0     0     0     0]
 [  132     0     0     0     0     0     0     0  7575     0     0   520
      0     0     0     0     0     0     0     0     0]
 [  716     0     0     0     0     0     0     0     0  2514     0  1165
      0     0     0     0     0     0     0     0     0]
 [  703     0     0     0     0     0     0     0     0     0     0     4
```

- **Classification Report:**

```
Visualizer score is:  0.766769532893814
                                 precision    recall  f1-score   support

                       NARCOTICS       0.60      1.00      0.75      7927
                         BATTERY       0.77      0.88      0.82     12900
                        BURGLARY       1.00      0.94      0.97      4246
                           THEFT       1.00      0.90      0.95     14801
             MOTOR VEHICLE THEFT       1.00      0.78      0.88      3262
                         ROBBERY       1.00      0.70      0.83      2722
               CRIMINAL TRESPASS       1.00      0.62      0.77      2044
                         ASSAULT       0.00      0.00      0.00      4351
                 CRIMINAL DAMAGE       0.99      0.92      0.95      8227
                   OTHER OFFENSE       1.00      0.57      0.73      4395
                    PROSTITUTION       0.00      0.00      0.00       707
              DECEPTIVE PRACTICE       0.22      0.83      0.34      2604
                     SEX OFFENSE       0.00      0.00      0.00       238
      OFFENSE INVOLVING CHILDREN       0.00      0.00      0.00       463
               WEAPONS VIOLATION       0.00      0.00      0.00       709
INTERFERENCE WITH PUBLIC OFFICER       0.00      0.00      0.00       160
              CRIM SEXUAL ASSAULT       0.00      0.00      0.00       271
                        GAMBLING       0.00      0.00      0.00       165
                          OTHERS       0.00      0.00      0.00       352
           PUBLIC PEACE VIOLATION       0.00      0.00      0.00       573
              LIQUOR LAW VIOLATION       0.00      0.00      0.00       173

                        accuracy                           0.77     71290
                       macro avg       0.41      0.39      0.38     71290
                    weighted avg       0.77      0.77      0.75     71290
```

8. Implementing an ensemble model:

Ensemble model we have used is Voting Classifier, which takes the above three models as input and combines the features of three models based on the highest probability.

- **Building the model:**

```python
from sklearn.ensemble import VotingClassifier
#Creating an ensemble model to combine 3 models using votingclassifier
model_ensemble = VotingClassifier(estimators=[('knn', model_knn), ('rf', model_rforest), ('nn', neural_model)],weights=[1,1,1],f
#Training the model
model_ensemble.fit(x_train,y_train)
# Predicting the result using test data
predicted_result = model_ensemble.predict(x_test)
# Evaluating the model
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score, f1_score
accuracy = accuracy_score(y_test, predicted_result)
recall = recall_score(y_test, predicted_result, average="weighted")
percision = precision_score(y_test, predicted_result, average="weighted")
f1score = f1_score(y_test, predicted_result, average='micro')
confusionmatrix = confusion_matrix(y_test, predicted_result)

print("========== Evaluation results of Ensemble Model ==========")
print("Accuracy    : ", accuracy)
print("Recall      : ", recall)
print("Precision   : ", percision)
print("F1 Score    : ", f1score)
print("Confusion Matrix: ")
print(confusionmatrix)
#Classification report
target_names = a
visualizer = ClassificationReport(model_ensemble, classes=target_names, size=(1080, 720))
visualizer.fit(X=x_train, y=y_train)     # Fit the training data to the visualizer
print("Visualizer score is: ",visualizer.score(x_test, y_test))      # Evaluate the model on the test data
print(classification_report(y_test, predicted_result,target_names=a))
g = visualizer.poof()
```

- **Evaluation Report:**

```
========== Evaluation results of Ensemble Model ==========
Accuracy    :  0.766769532893814
Recall      :  0.766769532893814
Precision   :  0.7695011493776878
F1 Score    :  0.766769532893814
Confusion Matrix:
[[ 7904     0     0     0     0     0     0     0     0     0     0    23
      0     0     0     0     0     0     0     0     0]
 [  479 11374     0     0     0     0     0     0     0     0     0  1047
      0     0     0     0     0     0     0     0     0]
 [   36     0  4009     0     0     0     0     0     0     0     0   201
      0     0     0     0     0     0     0     0     0]
 [   52     0     0 13388     0     0     0     0     0     0     0  1361
      0     0     0     0     0     0     0     0     0]
 [   82     0     0     0  2551     0     0     0     0     0     0   629
      0     0     0     0     0     0     0     0     0]
 [   87     0     0     0     0  1919     0     0     0     0     0   716
      0     0     0     0     0     0     0     0     0]
 [  454     0     0     0     0     0  1270     0   113     0     0   207
      0     0     0     0     0     0     0     0     0]
 [  410  3284     0     0     0     0     0     0     0     0     0   657
      0     0     0     0     0     0     0     0     0]
 [  132     0     0     0     0     0     0     0  7575     0     0   520
      0     0     0     0     0     0     0     0     0]
 [  716     0     0     0     0     0     0     0     0  2514     0  1165
      0     0     0     0     0     0     0     0     0]
```

- **Classification report:**

Visualizer score is: 0.766769532893814

	precision	recall	f1-score	support
NARCOTICS	0.60	1.00	0.75	7927
BATTERY	0.77	0.88	0.82	12900
BURGLARY	1.00	0.94	0.97	4246
THEFT	1.00	0.90	0.95	14801
MOTOR VEHICLE THEFT	1.00	0.78	0.88	3262
ROBBERY	1.00	0.70	0.83	2722
CRIMINAL TRESPASS	1.00	0.62	0.77	2044
ASSAULT	0.00	0.00	0.00	4351
CRIMINAL DAMAGE	0.99	0.92	0.95	8227
OTHER OFFENSE	1.00	0.57	0.73	4395
PROSTITUTION	0.00	0.00	0.00	707
DECEPTIVE PRACTICE	0.22	0.83	0.34	2604
SEX OFFENSE	0.00	0.00	0.00	238
OFFENSE INVOLVING CHILDREN	0.00	0.00	0.00	463
WEAPONS VIOLATION	0.00	0.00	0.00	709
INTERFERENCE WITH PUBLIC OFFICER	0.00	0.00	0.00	160
CRIM SEXUAL ASSAULT	0.00	0.00	0.00	271
GAMBLING	0.00	0.00	0.00	165
OTHERS	0.00	0.00	0.00	352
PUBLIC PEACE VIOLATION	0.00	0.00	0.00	573
LIQUOR LAW VIOLATION	0.00	0.00	0.00	173
accuracy			0.77	71290
macro avg	0.41	0.39	0.38	71290
weighted avg	0.77	0.77	0.75	71290

Front End:

Our project had been hosted in the below mentioned link

https://myfirstapp132.herokuapp.com/

Advantages and Disadvantages of each Approach:

- In KNN classifier approach accuracy is just 72.7% but the execution speed is high.
- Whereas for MLP classifier (Neural Network model) the accuracy is improved to 76.67% but the execution speed had be dropped to a drastic level.
- Finally ensemble model, even though it had less execution speed it combines the features of all the earlier models to improve the accuracy.

Video link:

https://youtu.be/cVnEVwHOvFw

Source Code Github link:
https://github.com/LalithChandraAttaluri/Python_ML_Masters_Project/blob/master/Source/Final_Source_code.py

PPT Github link:

https://github.com/LalithChandraAttaluri/Python_ML_Masters_Project/blob/master/Documentation/Chicago_Crime_Data_Analysis_Team4.pptx