Analyzing NYPD Historic Complaint Data

Introduction

- This presentation explores historic complaint data reported to the NYPD from 2006 to 2020, provided by NYC Open Data. The dataset includes details such as complaint type, location, date, and demographic information. It helps identify crime patterns and trends across New York City, supporting transparency and informed decision-making for policymakers, researchers, and the public.
- By analyzing this data, we can uncover insights into how crime has changed over time. It also highlights areas that may need increased attention or resources for public safety.

Dataset

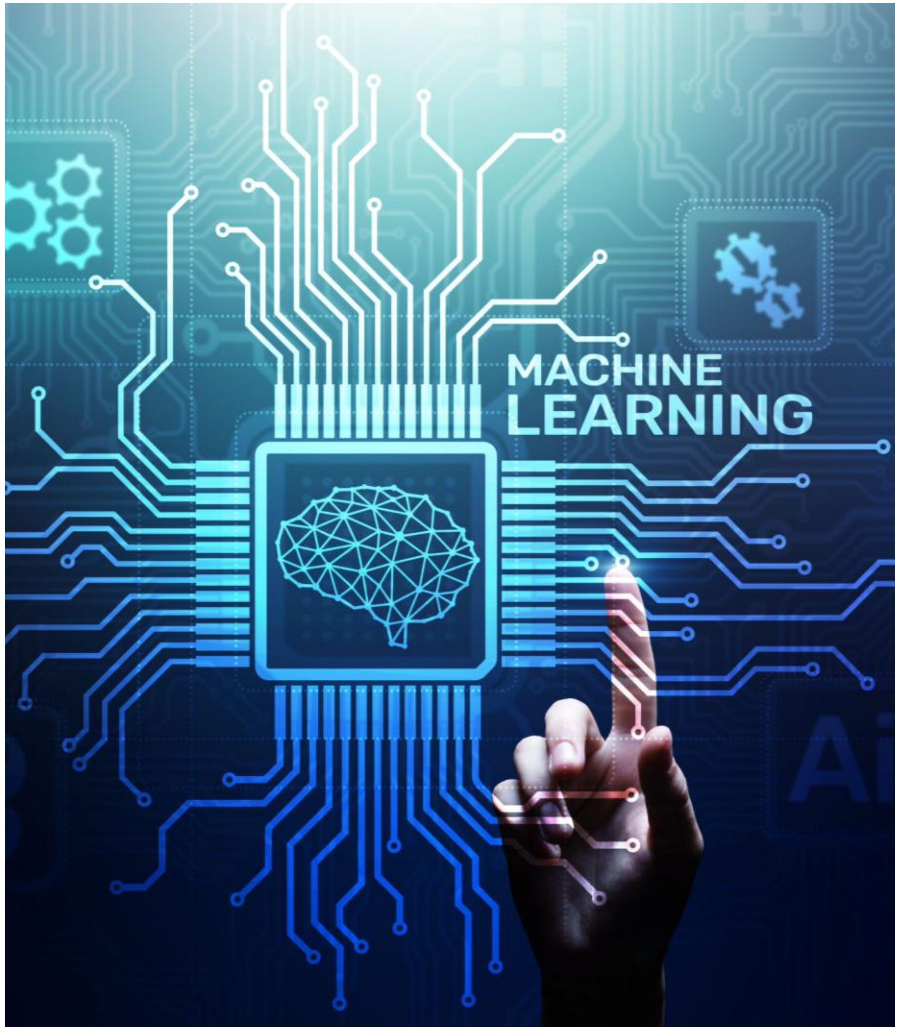
- Link: https://data.cityofnewyork.us/Public-Safety/NYPD-Complaint-Data-Historic/qgea-i56i/about_data

- The NYPD Complaint Data Historic dataset contains over **7 million rows**, covering reported crimes from **2006 to 2019**. Each row includes details such as **complaint number, offense type, location (borough, precinct, coordinates), date/time of incident, and suspect/victim demographics**. This structured data helps track crime trends across time and geography in New York City.

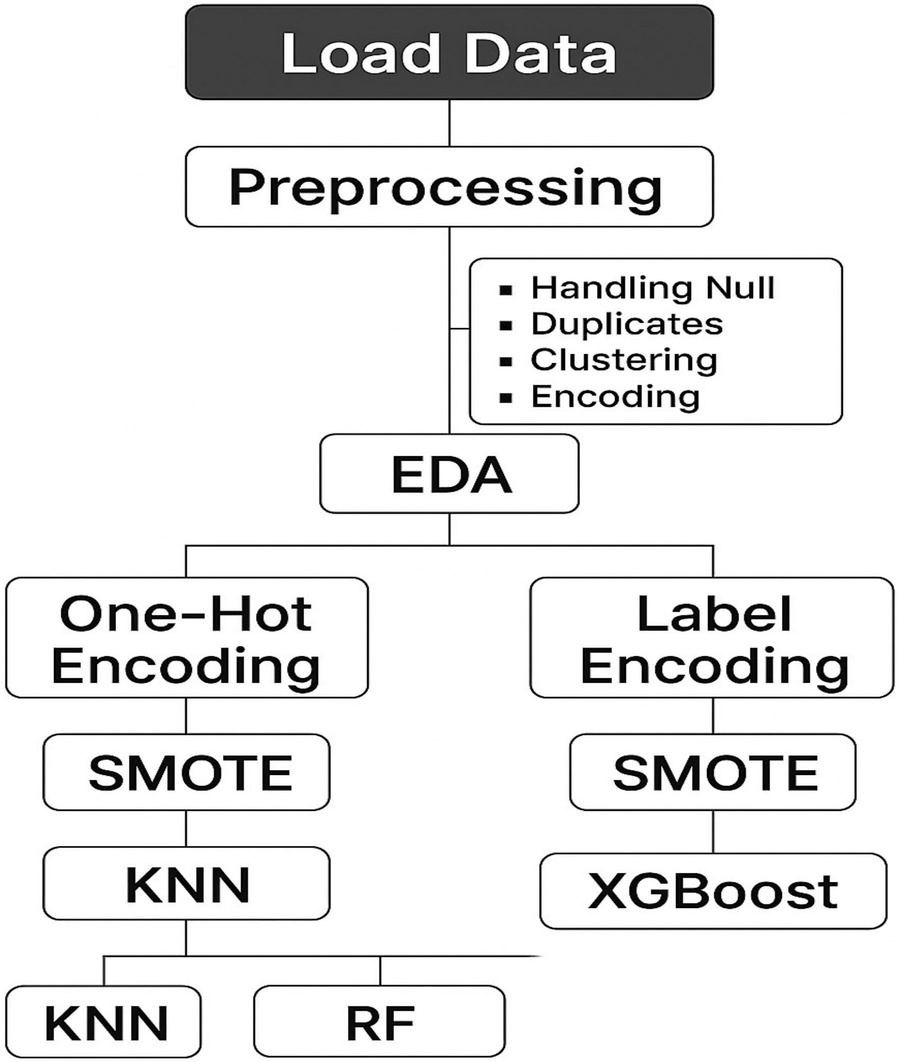
ML Models used:

Based on your notebook, here's a short, polished summary in the same style:

- In our analysis, We used KNN, Random Forest, and XGBoost models to classify NYPD crime complaints, with preprocessing steps like missing value handling, encoding, and feature scaling. EDA and visualizations using pandas, NumPy, seaborn, and matplotlib helped reveal key crime patterns. This project offers valuable insights for public safety agencies and urban planners to better understand and address crime in New York City.



Flow Diagram



EDA – Removing null and missing values

```
import pandas as pd

# Assuming your DataFrame is named df

# Replace '(null)' with NaN (missing value)
df_cleaned = df_cleaned.replace('(null)', pd.NA)

# Now drop rows that have any NaN value
df_cleaned = df_cleaned.dropna()

# (Optional) Reset the index if needed
df_cleaned = df_cleaned.reset_index(drop=True)

# Done! Now df has no '(null)' entries.
```

```
#checking null values
df_cleaned.isnull().sum()
```

```
CMPLNT_FR_DT          0
CMPLNT_FR_TM          0
ADDR_PCT_CD           0
OFNS_DESC             0
CRM_ATPT_CPTD_CD      0
LAW_CAT_CD            0
BORO_NM               0
LOC_OF_OCCUR_DESC     0
PREM_TYP_DESC         0
```

```
#dropping the duplicates
df_cleaned = df_cleaned.drop_duplicates(keep='first')
```

Exploratory Data Analysis

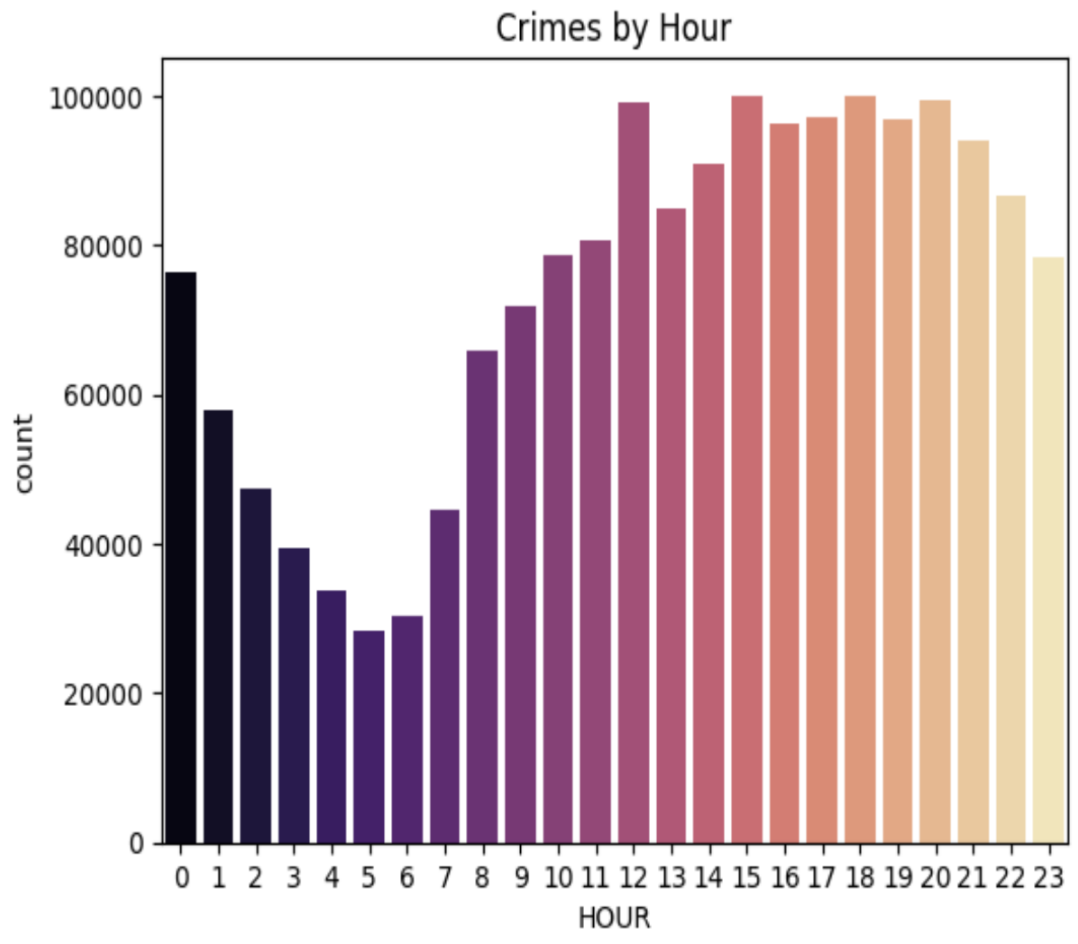
```
[ ] print(df_cleaned.shape)
    print(df_cleaned.info())
    print(df_cleaned.isnull().sum())
```

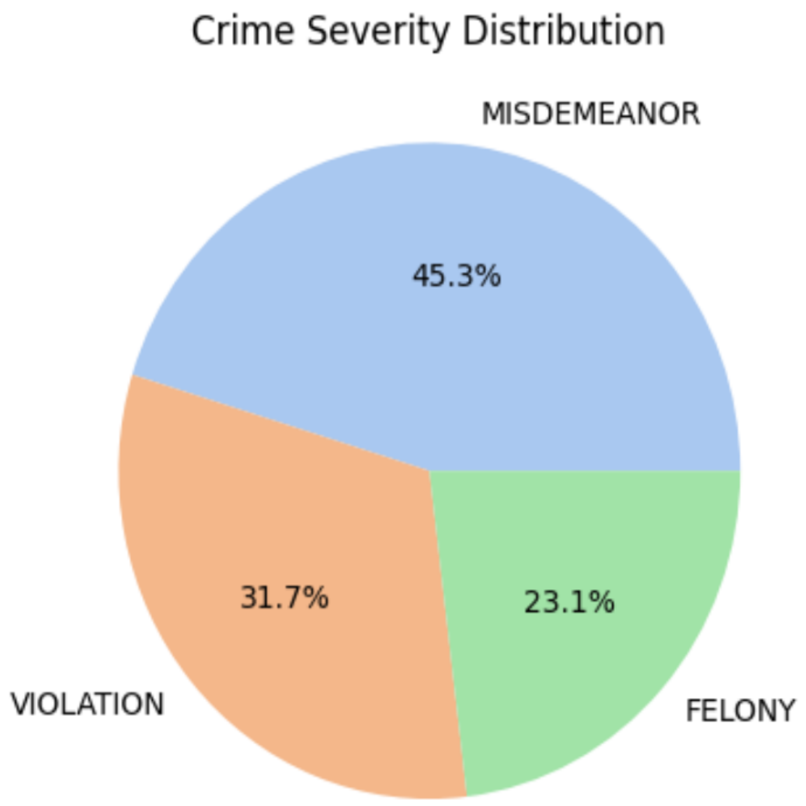
```
(1778486, 20)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1778486 entries, 0 to 1778485
Data columns (total 20 columns):
 #   Column             Dtype
---  ------             -----
 0   CMPLNT_FR_DT       datetime64[ns]
 1   CMPLNT_FR_TM       object
 2   ADDR_PCT_CD        int64
 3   OFNS_DESC          object
 4   CRM_ATPT_CPTD_CD   object
 5   LAW_CAT_CD         object
 6   BORO_NM            object
 7   LOC_OF_OCCUR_DESC  object
 8   PREM_TYP_DESC      object
 9   JURIS_DESC         object
 10  SUSP_AGE_GROUP     object
 11  SUSP_RACE          object
```

```
initial_dataset.describe()
```

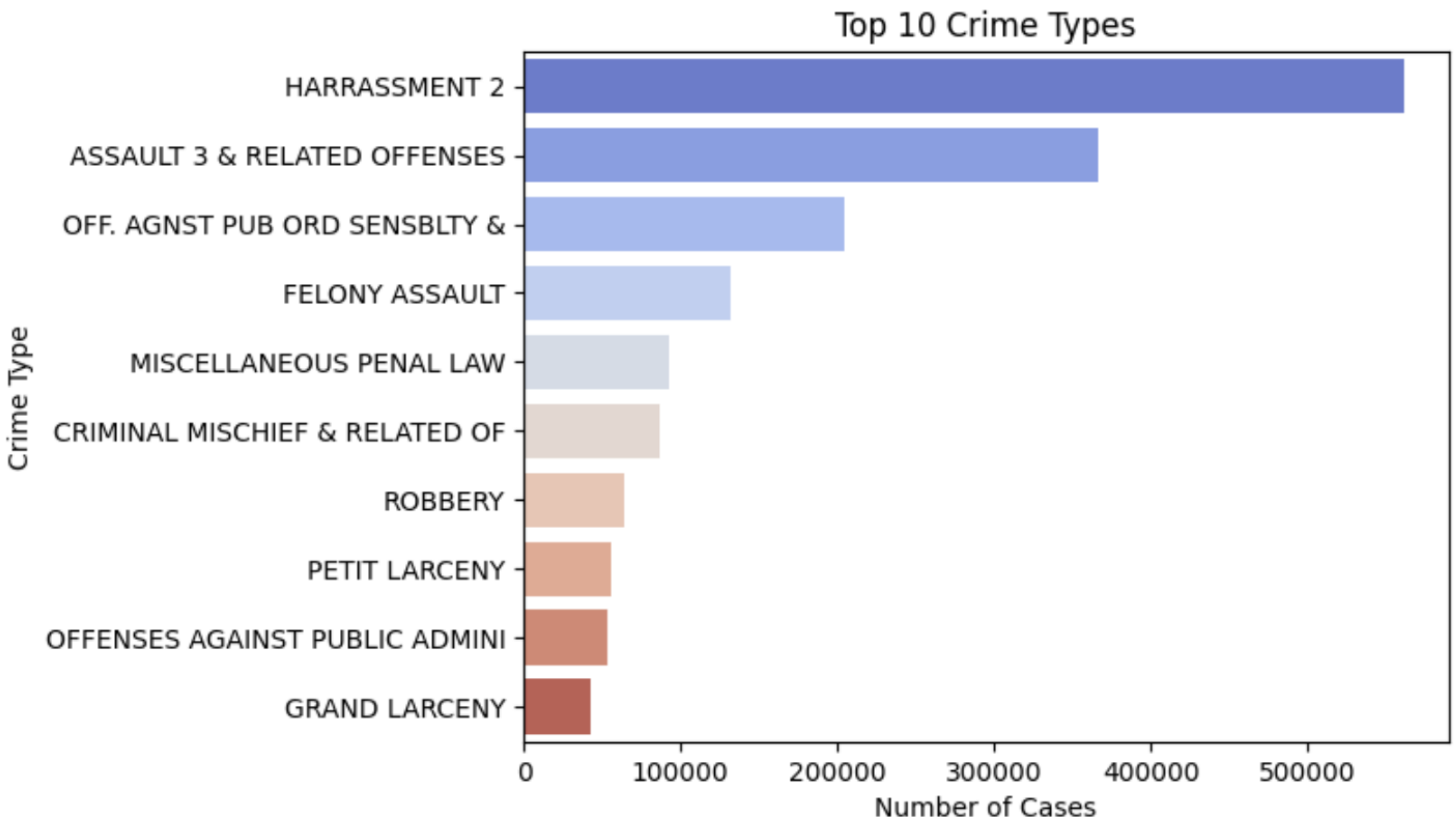
	ADDR_PCT_CD	KY_CD	PD_CD
count	8.914067e+06	8.914838e+06	8.907245e+06
mean	6.336865e+01	2.972099e+02	4.124253e+02
std	3.459529e+01	1.521695e+02	2.181523e+02
min	1.000000e+00	1.010000e+02	1.000000e+02
25%	4.000000e+01	1.170000e+02	2.540000e+02
50%	6.300000e+01	3.410000e+02	3.610000e+02
75%	9.400000e+01	3.510000e+02	6.370000e+02
max	1.230000e+02	8.810000e+02	9.750000e+02

Visualizations

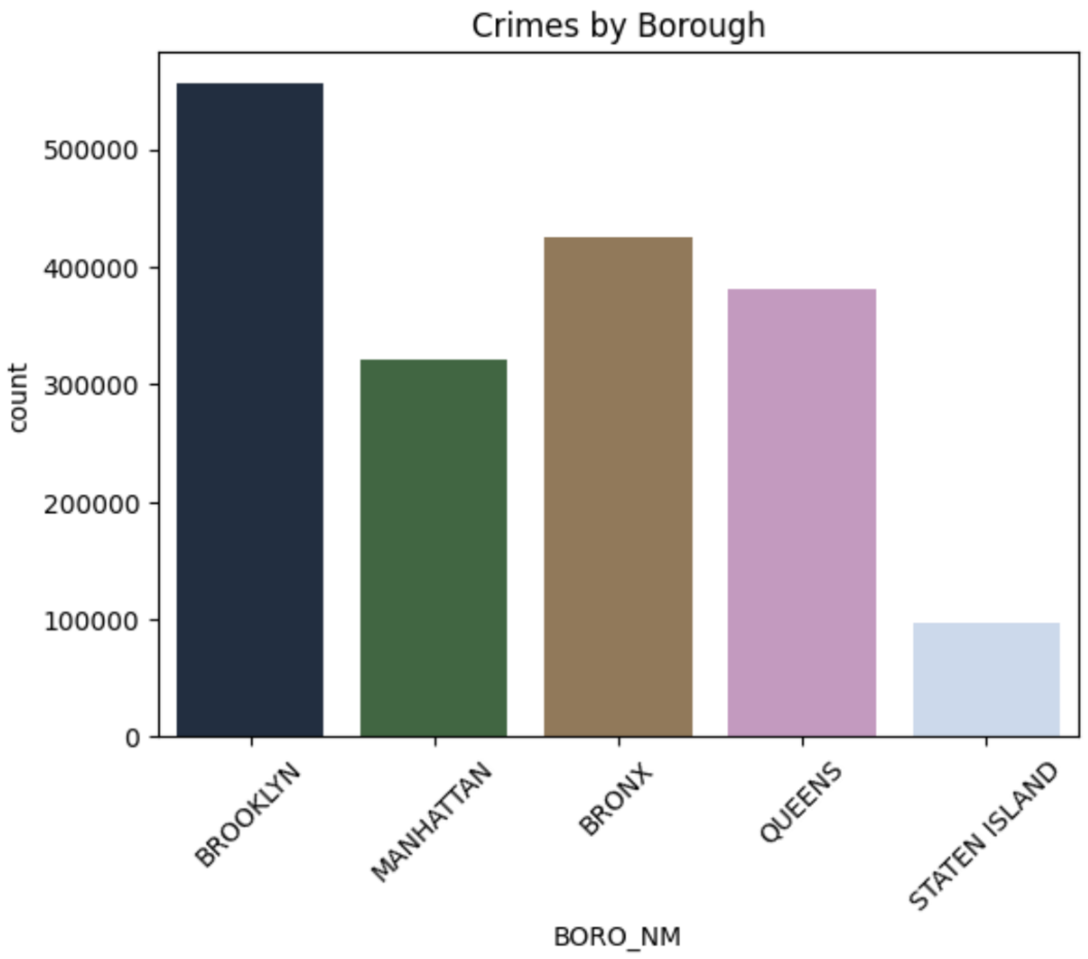


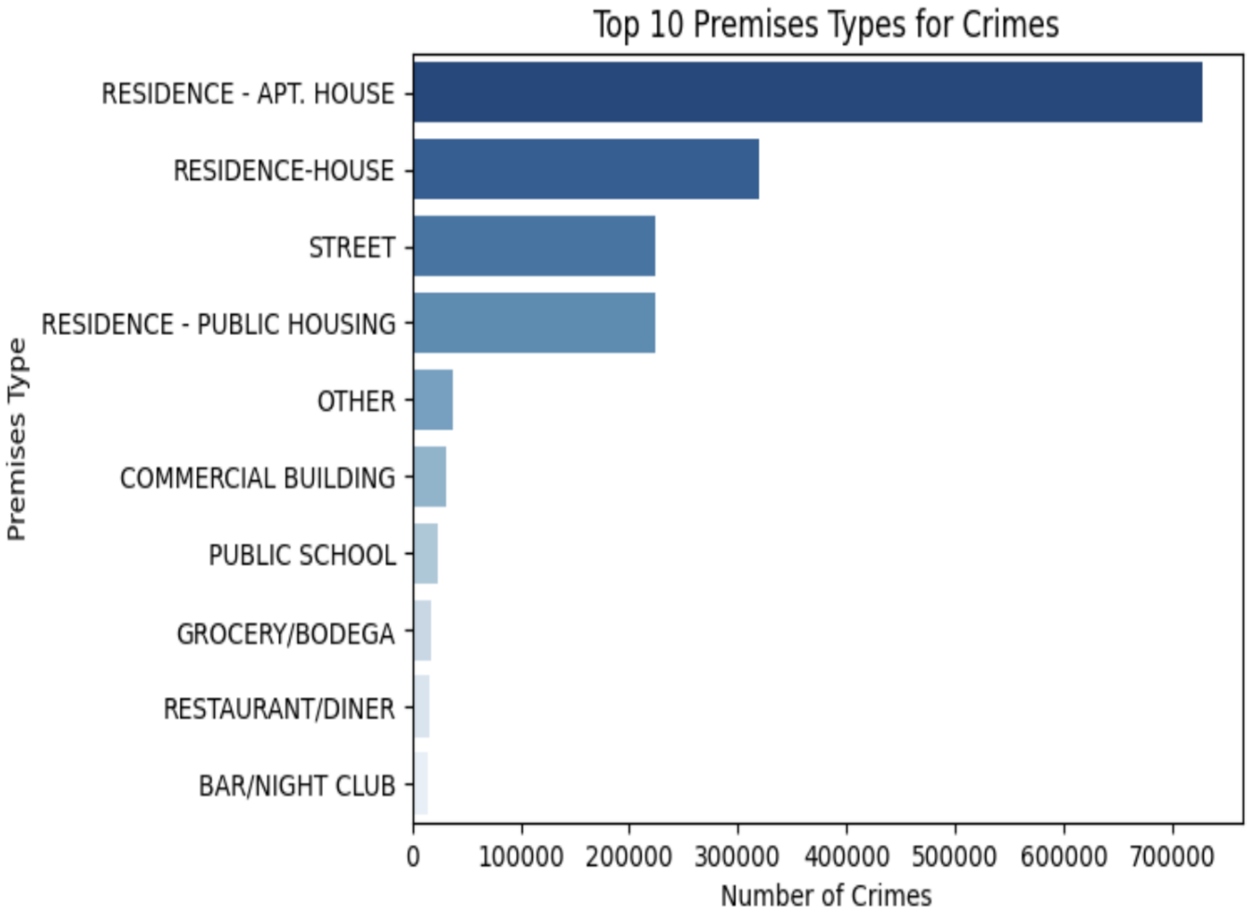


Visualizations

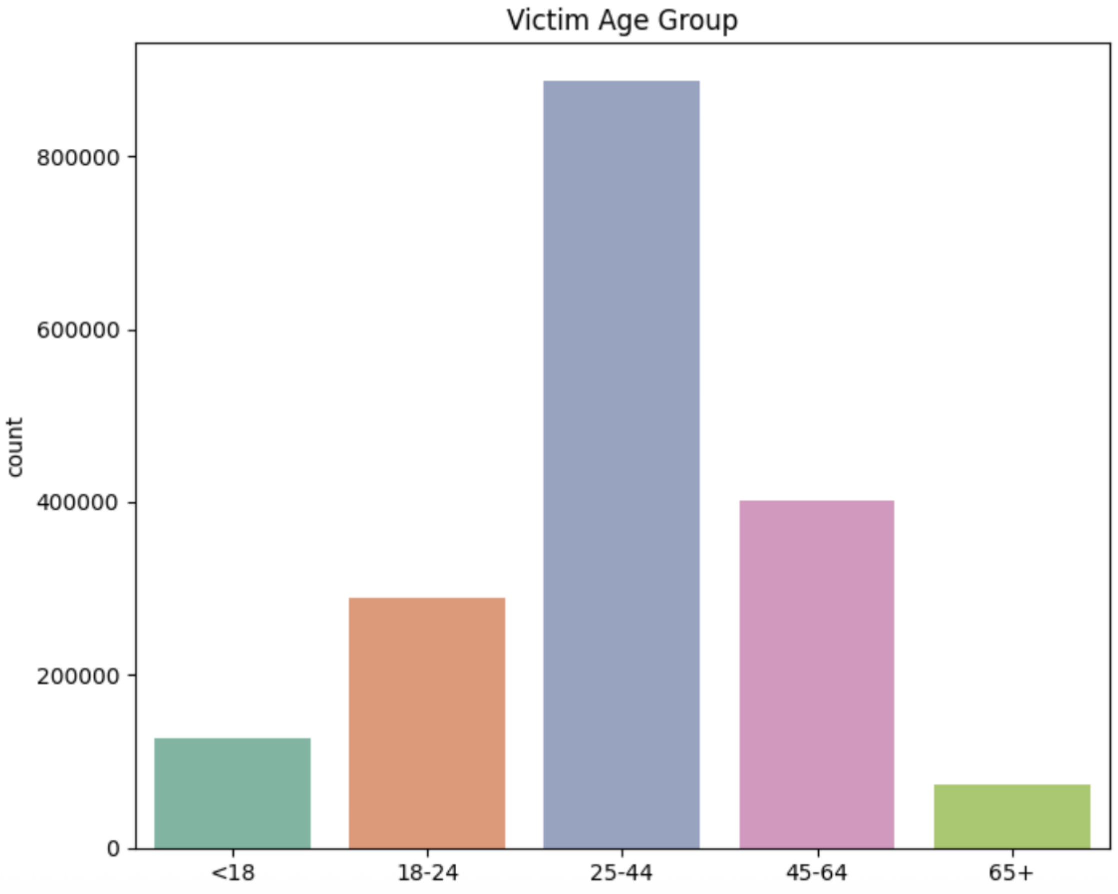


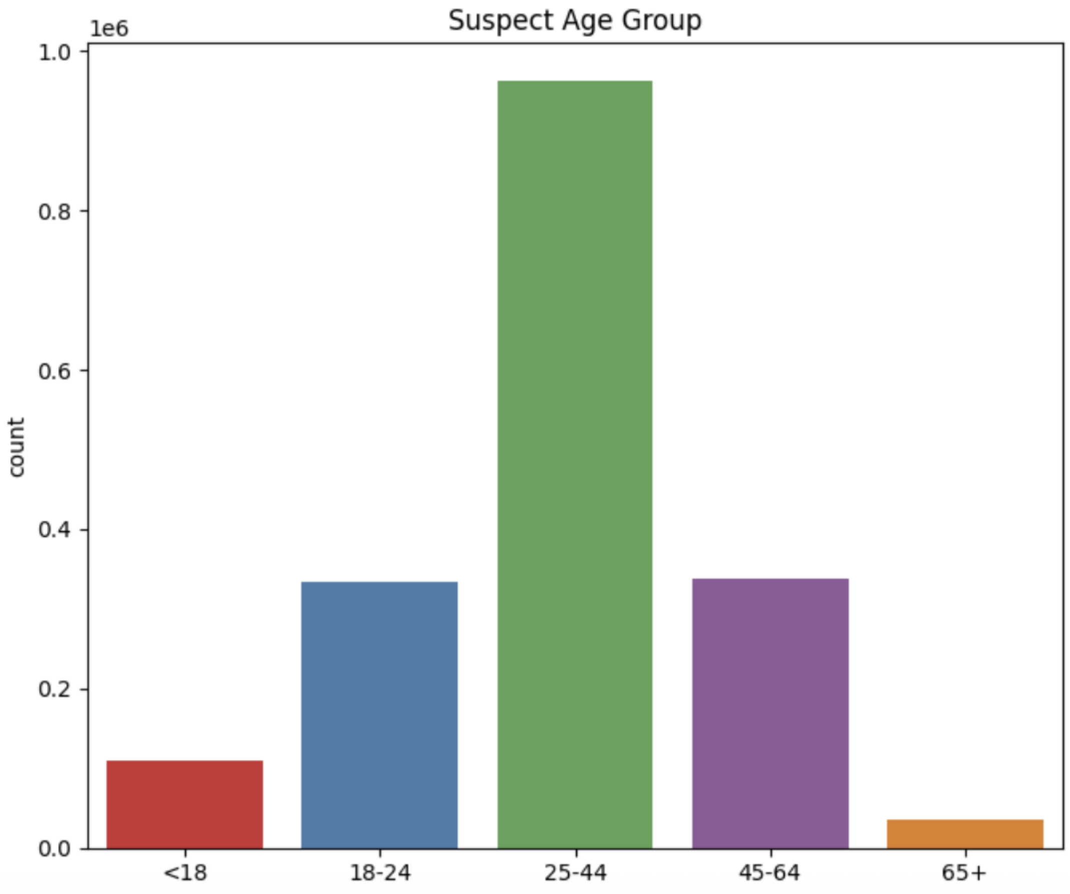
Visualizations





Visualizations





Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

rf_10 = RandomForestClassifier(max_depth=10, random_state=42, n_jobs=-1)
rf_10.fit(X_train, y_train)

y_pred = rf_10.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9761475374195859
              precision    recall  f1-score   support

      FELONY       0.96      0.97      0.96    160989
 MISDEMEANOR       0.97      0.96      0.96    160990
   VIOLATION       1.00      1.00      1.00    160990

    accuracy                           0.98    482969
   macro avg       0.98      0.98      0.98    482969
weighted avg       0.98      0.98      0.98    482969
```

KNN

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

knn = KNeighborsClassifier(n_neighbors=5, n_jobs=-1)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9517173980110525
              precision    recall  f1-score   support

      FELONY       0.93      0.95      0.94    160989
 MISDEMEANOR       0.96      0.90      0.93    160990
   VIOLATION       0.96      1.00      0.98    160990

    accuracy                           0.95    482969
   macro avg       0.95      0.95      0.95    482969
weighted avg       0.95      0.95      0.95    482969
```

XGBoost Model

```python
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, accuracy_score

# Initialize XGBoost with basic parameters
xgb_model = XGBClassifier(
    n_estimators=100,
    max_depth=6,
    learning_rate=0.1,
    objective='multi:softmax',  # for multi-class classification
    num_class=len(y_train.unique()),  # important for multi-class
    use_label_encoder=False,
    eval_metric='mlogloss',
    random_state=42
)

# Fit model (use X_smote and y_smote if using balanced data)
xgb_model.fit(X_train, y_train)

# Predict and evaluate
y_pred = xgb_model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9821334288536118

Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.96      0.97    160989
           1       0.96      0.98      0.97    160990
           2       1.00      1.00      1.00    160990

    accuracy                           0.98    482969
   macro avg       0.98      0.98      0.98    482969
weighted avg       0.98      0.98      0.98    482969
```

Comparison

Model	Accuracy	Precision	Recall	F1-Score
Random Forest	0.9761	0.98	0.98	0.98
KNN	0.9517	0.95	0.95	0.95
XGBoost	0.9821	0.98	0.98	0.98

Results & Analysis

In this project, we applied three machine learning models — **Random Forest**, **XGBoost**, and **K-Nearest Neighbors (KNN)** — to classify crime categories based on NYPD data.
After evaluating each model using accuracy, precision, recall, and F1-score, **XGBoost emerged as the best performer** with an accuracy of **98.21%**.

This high performance is due to XGBoost's ability to handle complex patterns and optimize classification through boosting techniques.

Conclusion

We analyzed NYPD crime data to uncover trends in offense types and time patterns, with 2018 showing the highest crime rate. The most common crimes in the dataset are **Grand Larceny**, **Felony Assault**, **Petit Larceny**, **Robbery**, and **Burglary**, with **Grand Larceny** being the most frequent. These highlight major focus areas for crime prevention.

Using machine learning models — **KNN, Random Forest, and XGBoost** — we classified crimes effectively. **XGBoost performed best** with **98.21% accuracy**, making it ideal for predicting crime categories and aiding public safety planning.

Thank You