Crime_Analytics_With_TimeSeries_and_Visualizations

May 9, 2025

Big Data-Driven Crime Analytics: Scalable Data Processing and Predictive Intelligence for Urban Safety

[ ]:
```
!wget -O NYPD_Complaint_Data_Historic.csv "https://data.cityofnewyork.us/api/
↪views/qgea-i56i/rows.csv?accessType=DOWNLOAD"
```

```
--2025-05-08 14:03:05--
https://data.cityofnewyork.us/api/views/qgea-i56i/rows.csv?accessType=DOWNLOAD
Resolving data.cityofnewyork.us (data.cityofnewyork.us)… 52.206.140.205,
52.206.140.199, 52.206.68.26
Connecting to data.cityofnewyork.us
(data.cityofnewyork.us)|52.206.140.205|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: unspecified [text/csv]
Saving to: 'NYPD_Complaint_Data_Historic.csv'

NYPD_Complaint_Data     [      <=>           ]   3.01G  4.84MB/s    in 11m 49s

2025-05-08 14:14:56 (4.35 MB/s) - 'NYPD_Complaint_Data_Historic.csv' saved
[3229396846]
```

#Load and Clean NYPD Crime Data

[ ]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("CrimeAnalyticsXGBoost").getOrCreate()

from pyspark.sql.functions import col, hour, dayofweek, month, when, concat_ws,␣
↪to_timestamp
from pyspark.ml.feature import StringIndexer, VectorAssembler, Imputer
from pyspark.ml.functions import vector_to_array
from pyspark.ml.clustering import KMeans
from xgboost.spark import SparkXGBClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
import matplotlib.pyplot as plt

# Load & Clean data
df = spark.read.option("header", True).option("inferSchema", True).csv("/
↪content/NYPD_Complaint_Data_Historic.csv")
```

```
df = df.dropna(subset=["LAW_CAT_CD"])
df = df.filter(col("CMPLNT_FR_DT").isNotNull() & col("CMPLNT_FR_TM").
 ↪isNotNull())
df = df.withColumn("CMPLNT_TS", to_timestamp(concat_ws(" ",␣
 ↪col("CMPLNT_FR_DT"), col("CMPLNT_FR_TM")), "MM/dd/yyyy HH:mm:ss"))
df = df.filter(col("CMPLNT_TS").isNotNull())

df = df.withColumn("Hour", hour("CMPLNT_TS"))
df = df.withColumn("DayOfWeek", dayofweek("CMPLNT_TS"))
df = df.withColumn("Month", month("CMPLNT_TS"))
df = df.withColumn("IsWeekend", when(col("DayOfWeek").isin([1, 7]), 1).
 ↪otherwise(0))
df = StringIndexer(inputCol="LAW_CAT_CD", outputCol="label",␣
 ↪handleInvalid="skip").fit(df).transform(df)

cat_cols = ["BORO_NM", "OFNS_DESC", "CRM_ATPT_CPTD_CD", "PREM_TYP_DESC",␣
 ↪"LOC_OF_OCCUR_DESC"]
for c in cat_cols:
    df = StringIndexer(inputCol=c, outputCol=f"{c}_Idx", handleInvalid="keep").
 ↪fit(df).transform(df)
```

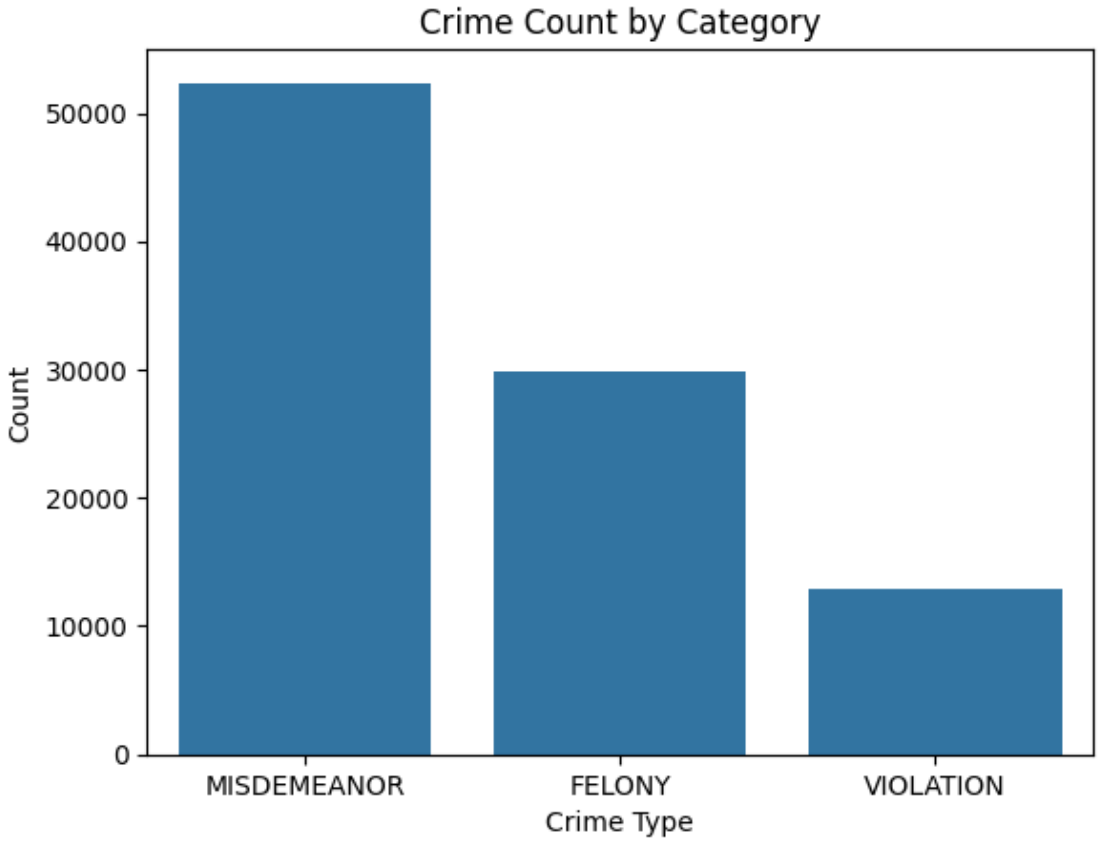
#Clustering and Vectorization for Model Training

[ ]:
```
num_cols = ["Hour", "DayOfWeek", "Month", "IsWeekend", "Latitude", "Longitude"]
df = df.filter((col("Hour").isNotNull()) & (col("DayOfWeek").isNotNull()) &␣
 ↪(col("Month").isNotNull()) &
               (col("IsWeekend").isNotNull()) & col("Latitude").isNotNull() &␣
 ↪col("Longitude").isNotNull())

df = Imputer(inputCols=num_cols, outputCols=num_cols).fit(df).transform(df)

geo_vec = VectorAssembler(inputCols=["Latitude", "Longitude"],␣
 ↪outputCol="geo_features")
df = geo_vec.transform(df)
df = KMeans(k=10, seed=42, featuresCol="geo_features",␣
 ↪predictionCol="GeoCluster").fit(df).transform(df)

features = ["Hour", "DayOfWeek", "Month", "IsWeekend", "Latitude", "Longitude",␣
 ↪"GeoCluster"] + [f"{c}_Idx" for c in cat_cols]
df = VectorAssembler(inputCols=features, outputCol="features_final").
 ↪setHandleInvalid("skip").transform(df)

df = df.withColumn("features_array", vector_to_array("features_final"))
for i in range(12):
    df = df.withColumn(f"f_{i}", col("features_array")[i])
```

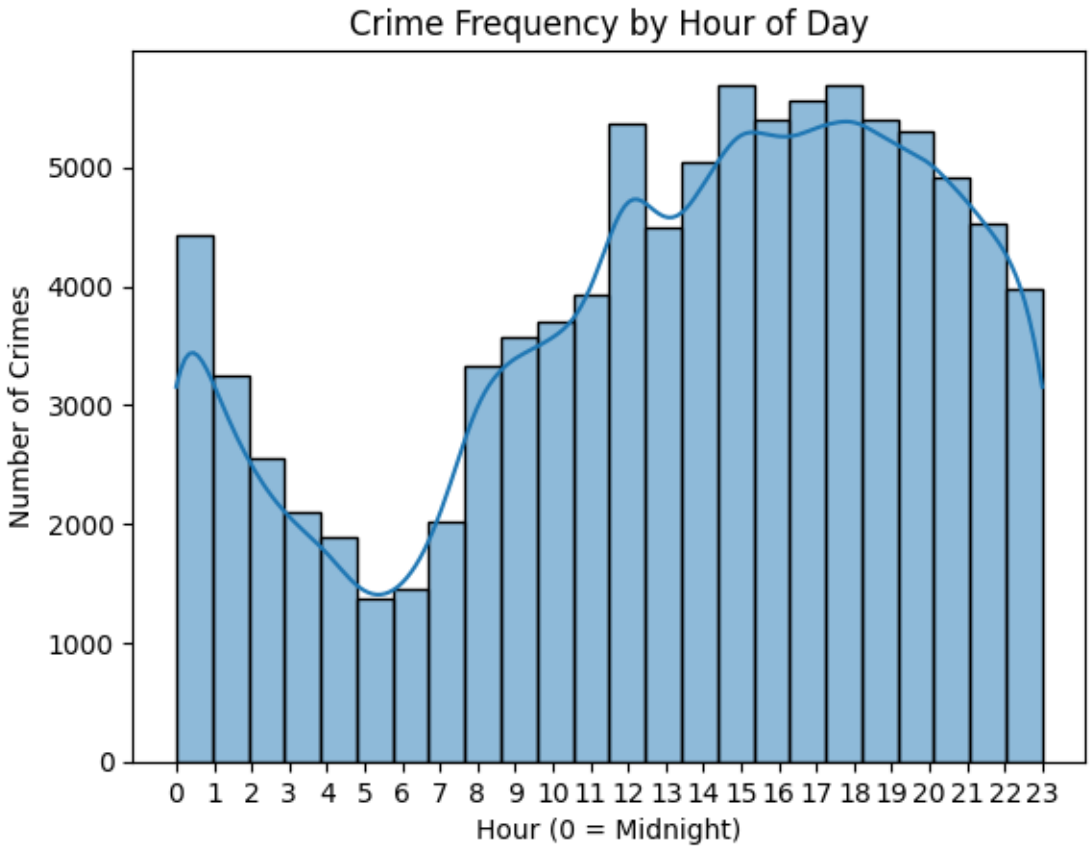
```
final_cols = [f"f_{i}" for i in range(12)] + ["label"]
df_clean = df.dropna(subset=final_cols)

df_sampled = df_clean.sample(fraction=0.01, seed=42)
train_df, test_df = df_sampled.randomSplit([0.8, 0.2], seed=42)
```

1 Exploratory Data Analysis (EDA)

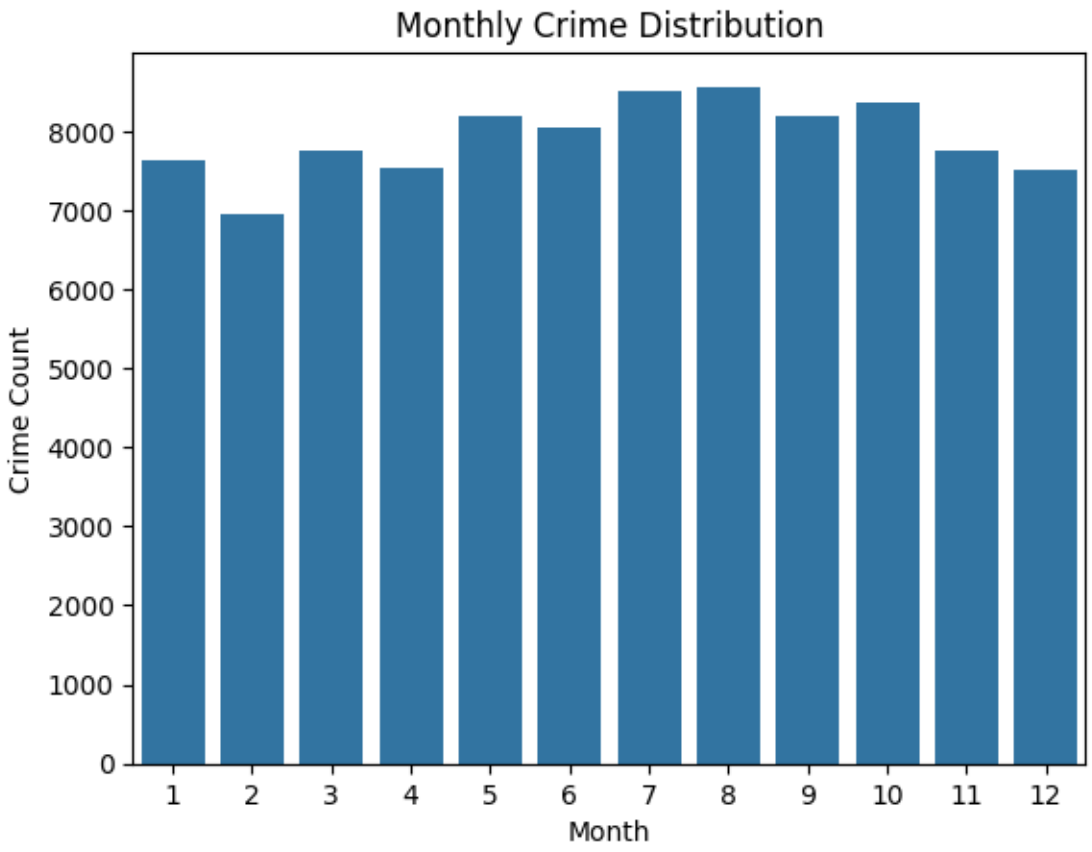
[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

pdf = df_clean.select("LAW_CAT_CD").sample(False, 0.01, seed=42).toPandas()
sns.countplot(data=pdf, x="LAW_CAT_CD", order=pdf["LAW_CAT_CD"].value_counts().
  ↪index)
plt.title("Crime Count by Category")
plt.xlabel("Crime Type")
plt.ylabel("Count")
plt.show()
```



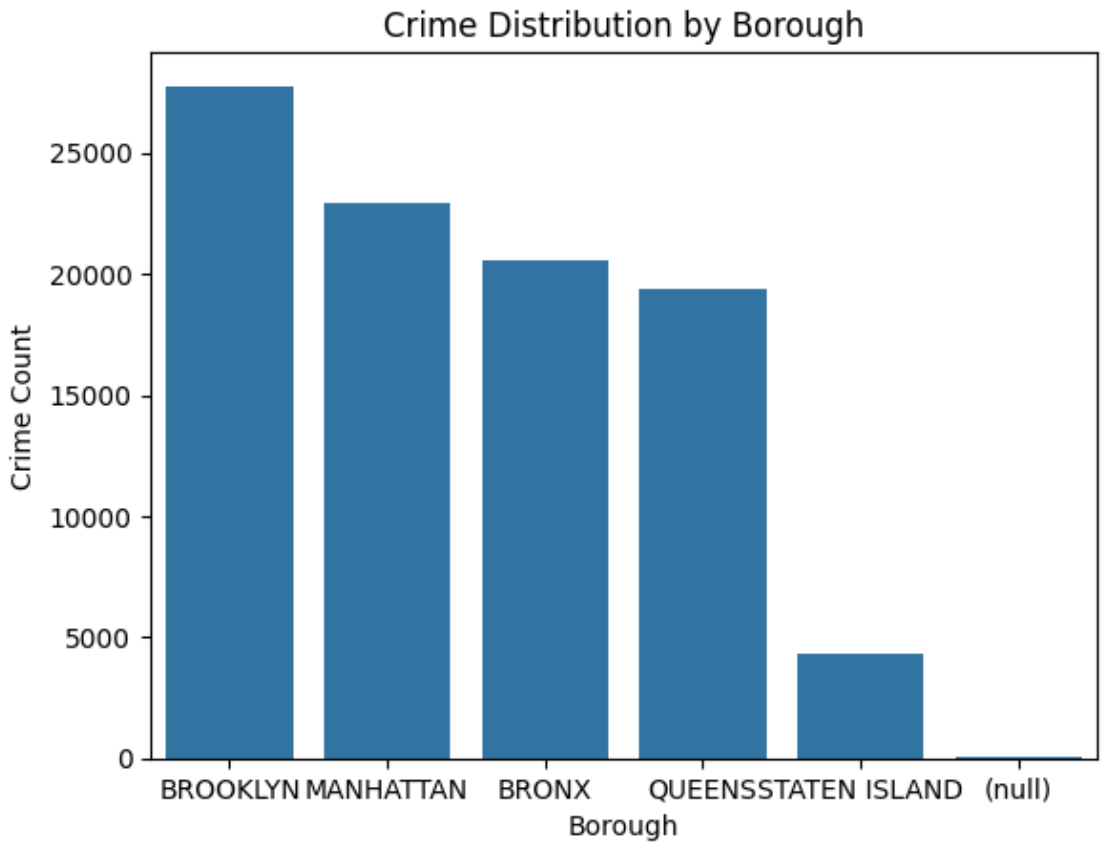
[ ]:
```
pdf = df_clean.select("Hour").dropna().sample(False, 0.01, seed=42).toPandas()
sns.histplot(pdf["Hour"], bins=24, kde=True)
plt.title("Crime Frequency by Hour of Day")
plt.xlabel("Hour (0 = Midnight)")
plt.ylabel("Number of Crimes")
plt.xticks(range(0, 24))
plt.show()
```



[ ]:
```
pdf = df_clean.select("Month").dropna().sample(False, 0.01, seed=42).toPandas()
sns.countplot(data=pdf, x="Month")
plt.title("Monthly Crime Distribution")
plt.xlabel("Month")
plt.ylabel("Crime Count")
plt.show()
```



[ ]:
```
pdf = df_clean.select("BORO_NM").dropna().sample(False, 0.01, seed=42).
  ↪toPandas()
sns.countplot(data=pdf, x="BORO_NM", order=pdf["BORO_NM"].value_counts().index)
plt.title("Crime Distribution by Borough")
plt.xlabel("Borough")
plt.ylabel("Crime Count")
plt.show()
```



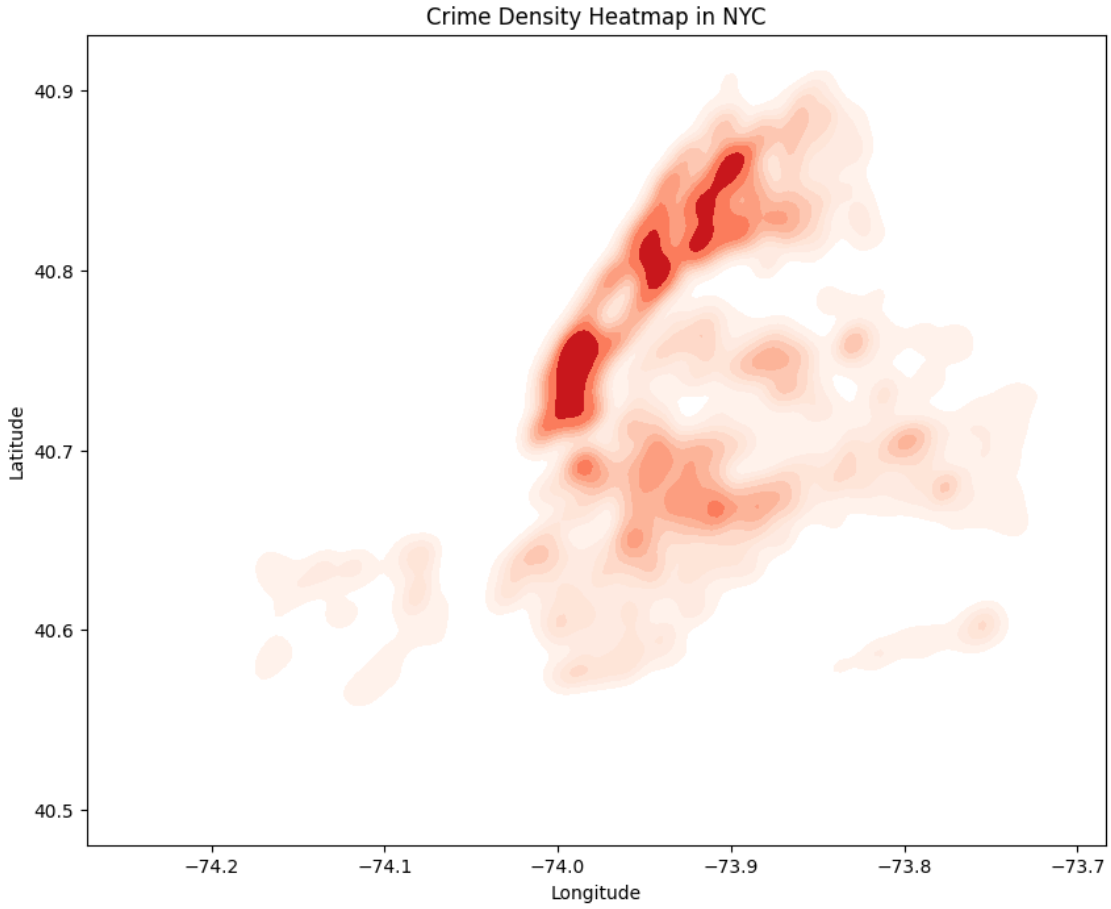
[ ]:
```
pdf = df_clean.select("Latitude", "Longitude", "LAW_CAT_CD", "BORO_NM") \
              .dropna() \
              .sample(False, 0.01, seed=42) \
              .toPandas()
```

[ ]:
```
print(pdf.columns.tolist())
# Expected: ['Latitude', 'Longitude', 'LAW_CAT_CD', 'BORO_NM']
```

```
['Latitude', 'Longitude', 'LAW_CAT_CD', 'BORO_NM']
```

[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))
sns.kdeplot(
    data=pdf, x="Longitude", y="Latitude",
    cmap="Reds", fill=True, bw_adjust=0.5, thresh=0.05
)
plt.title("Crime Density Heatmap in NYC")
plt.xlabel("Longitude")
```

```
plt.ylabel("Latitude")
plt.show()
```



[ ]:
```
import plotly.express as px

tree_data = pdf.groupby(["BORO_NM", "LAW_CAT_CD"]).size().
  ↪reset_index(name="Count")
fig = px.treemap(tree_data, path=["BORO_NM", "LAW_CAT_CD"], values="Count",
                 color="Count", color_continuous_scale="Reds",
                 title="Treemap: Crime by Borough and Category")
fig.show()
```

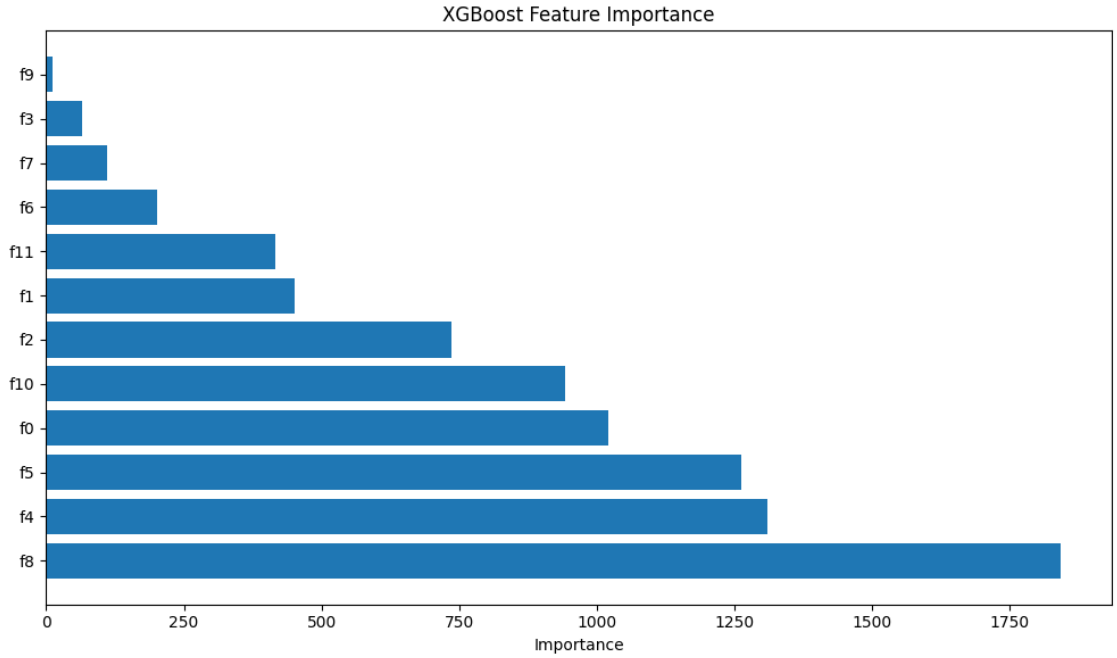
[ ]:
```
fig = px.sunburst(tree_data, path=["BORO_NM", "LAW_CAT_CD"], values="Count",
                  color="Count", color_continuous_scale="RdBu",
                  title="Sunburst: Borough → Crime Type")
fig.show()
```

1.1 XGBoost Classification and Feature Importance using PySpark

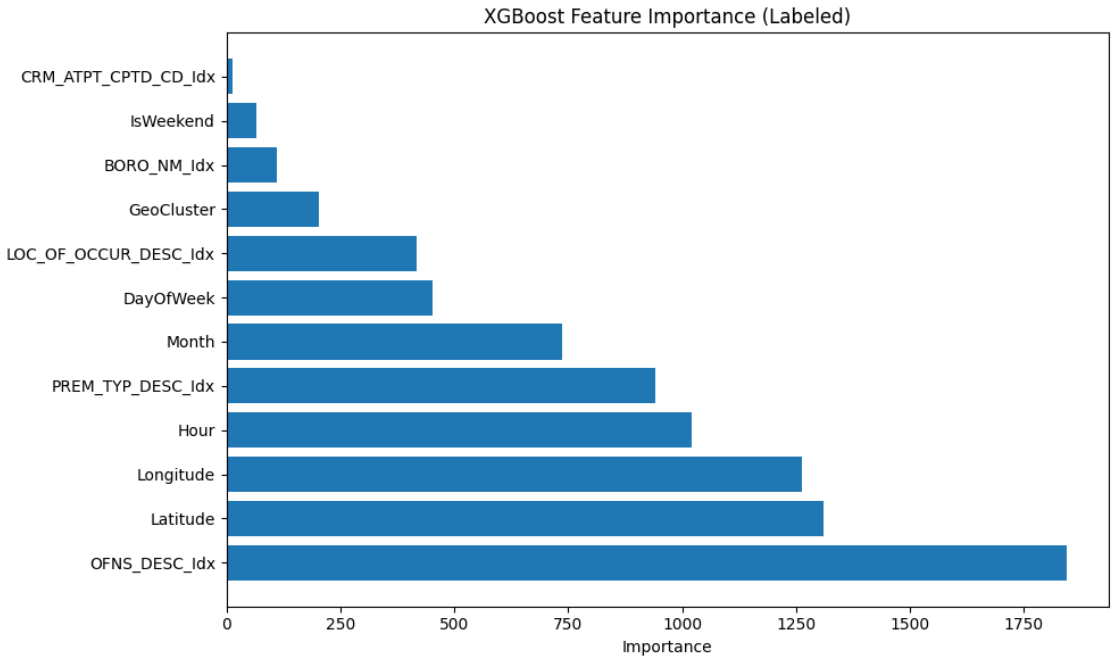
[ ]:
```
xgb = SparkXGBClassifier(
    label_col="label",
    features_col="features_final",
    prediction_col="prediction",
    num_round=50,
    max_depth=6,
    missing=0.0,
    use_gpu=False
)
xgb_model = xgb.fit(train_df)
xgb_preds = xgb_model.transform(test_df)

evaluator = MulticlassClassificationEvaluator(labelCol="label",␣
 ↪predictionCol="prediction", metricName="accuracy")
print("XGBoost Accuracy on Full Test Set:", evaluator.evaluate(xgb_preds))

# Feature Importance
importances = xgb_model.get_booster().get_score(importance_type='weight')
importances = sorted(importances.items(), key=lambda x: x[1], reverse=True)

plt.figure(figsize=(10,6))
plt.barh([k for k, v in importances], [v for k, v in importances])
plt.xlabel("Importance")
plt.title("XGBoost Feature Importance")
plt.tight_layout()
plt.show()
```

```
INFO:XGBoost-PySpark:Running xgboost-2.1.4 on 1 workers with
        booster params: {'objective': 'multi:softprob', 'device': 'cpu',
'max_depth': 6, 'num_round': 50, 'num_class': 3, 'nthread': 1}
        train_call_kwargs_params: {'verbose_eval': True, 'num_boost_round': 100}
        dmatrix_kwargs: {'nthread': 1, 'missing': 0.0}
INFO:XGBoost-PySpark:Finished xgboost training!
```

```
XGBoost Accuracy on Full Test Set: 0.9550079491255962
```



[ ]:
```
# Define feature names
feature_names = [
    "Hour", "DayOfWeek", "Month", "IsWeekend", "Latitude", "Longitude",
    "GeoCluster", "BORO_NM_Idx", "OFNS_DESC_Idx", "CRM_ATPT_CPTD_CD_Idx",
    "PREM_TYP_DESC_Idx", "LOC_OF_OCCUR_DESC_Idx"
]

importances = xgb_model.get_booster().get_score(importance_type='weight')

mapped_importances = {feature_names[int(k[1:])]: v for k, v in importances.
 ↪items()}
sorted_importances = sorted(mapped_importances.items(), key=lambda x: x[1],␣
 ↪reverse=True)

# Plot
plt.figure(figsize=(10, 6))
plt.barh([k for k, v in sorted_importances], [v for k, v in sorted_importances])
plt.xlabel("Importance")
plt.title("XGBoost Feature Importance (Labeled)")
plt.tight_layout()
plt.show()
```



2 Time Series Forecasting

[ ]:
```
!pip install prophet
```

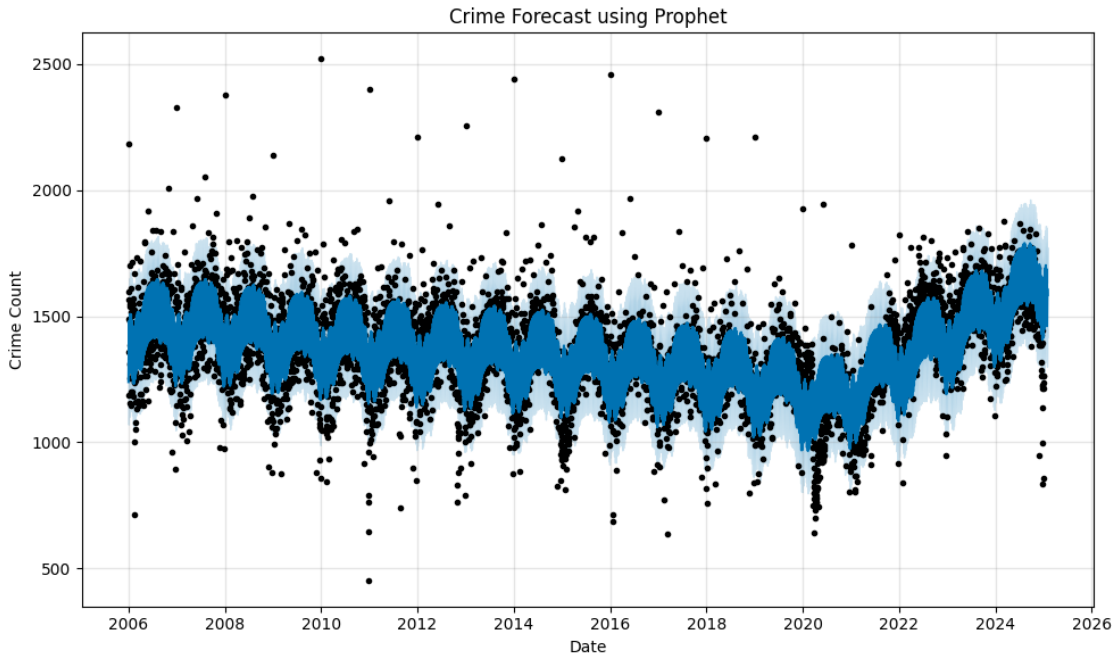
```
Requirement already satisfied: prophet in /usr/local/lib/python3.11/dist-
packages (1.1.6)
Requirement already satisfied: cmdstanpy>=1.0.4 in
/usr/local/lib/python3.11/dist-packages (from prophet) (1.2.5)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.11/dist-
packages (from prophet) (2.0.2)
Requirement already satisfied: matplotlib>=2.0.0 in
/usr/local/lib/python3.11/dist-packages (from prophet) (3.10.0)
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.11/dist-
packages (from prophet) (2.2.2)
Requirement already satisfied: holidays<1,>=0.25 in
/usr/local/lib/python3.11/dist-packages (from prophet) (0.71)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.11/dist-
packages (from prophet) (4.67.1)
Requirement already satisfied: importlib-resources in
/usr/local/lib/python3.11/dist-packages (from prophet) (6.5.2)
Requirement already satisfied: stanio<2.0.0,>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from cmdstanpy>=1.0.4->prophet) (0.5.1)
Requirement already satisfied: python-dateutil in
/usr/local/lib/python3.11/dist-packages (from holidays<1,>=0.25->prophet)
(2.9.0.post0)
```

```
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->prophet)
(1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-
packages (from matplotlib>=2.0.0->prophet) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->prophet)
(4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->prophet)
(1.4.8)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->prophet) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-
packages (from matplotlib>=2.0.0->prophet) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->prophet)
(3.2.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
packages (from pandas>=1.0.4->prophet) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
packages (from pandas>=1.0.4->prophet) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil->holidays<1,>=0.25->prophet) (1.17.0)
```

[ ]:
```python
# Filter valid dates before converting to Pandas
from pyspark.sql.functions import year

df_valid_dates = df_clean.filter((year("CMPLNT_TS") >= 2006) &␣
 ↪(year("CMPLNT_TS") <= 2025))

# Convert to Pandas
pdf = df_valid_dates.select("CMPLNT_TS").toPandas()

# Prepare for Prophet (group by day)
import pandas as pd
from prophet import Prophet
import matplotlib.pyplot as plt

pdf['CMPLNT_TS'] = pd.to_datetime(pdf['CMPLNT_TS']).dt.date
crime_counts = pdf.groupby("CMPLNT_TS").size().reset_index(name="y")
crime_counts.rename(columns={"CMPLNT_TS": "ds"}, inplace=True)

# Forecast with Prophet
model = Prophet()
model.fit(crime_counts)
```

```python
# Forecast 30 days into the future
future = model.make_future_dataframe(periods=30)
forecast = model.predict(future)

# Plot forecast
model.plot(forecast)
plt.title("Crime Forecast using Prophet")
plt.xlabel("Date")
plt.ylabel("Crime Count")
plt.tight_layout()
plt.show()

# View last 10 forecasted values
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail(10)
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with
daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpthz3hnla/wxtg2pkr.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpthz3hnla/tssmo3tg.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=30461', 'data',
'file=/tmp/tmpthz3hnla/wxtg2pkr.json', 'init=/tmp/tmpthz3hnla/tssmo3tg.json',
'output',
'file=/tmp/tmpthz3hnla/prophet_modelisixpbkw/prophet_model-20250508143154.csv',
'method=optimize', 'algorithm=lbfgs', 'iter=10000']
14:31:54 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
14:31:55 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```



[ ]:

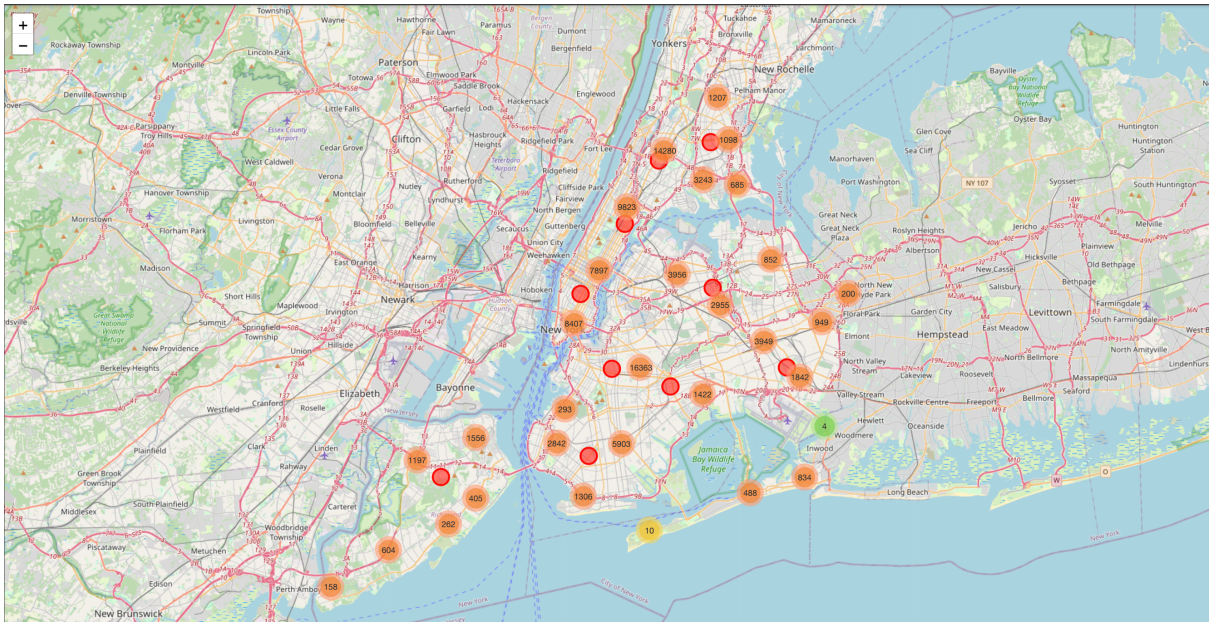
	ds	yhat	yhat_lower	yhat_upper
6960	2025-01-21	1598.418621	1449.257604	1761.828365
6961	2025-01-22	1634.348659	1473.740135	1812.182479
6962	2025-01-23	1609.503638	1454.110479	1778.529276
6963	2025-01-24	1686.558467	1515.310301	1850.452904
6964	2025-01-25	1583.246279	1427.772429	1748.901227
6965	2025-01-26	1460.606964	1300.322930	1633.491866
6966	2025-01-27	1510.618624	1350.992536	1661.650334
6967	2025-01-28	1573.878583	1418.873905	1734.202863
6968	2025-01-29	1608.802114	1449.937304	1773.169802
6969	2025-01-30	1583.472473	1418.391085	1739.670820

2.1 Crime Hotspot Visualization with Geo Clustering

[ ]:
```
from sklearn.cluster import KMeans
import numpy as np

df_sampled_geo = df_clean.select("Latitude", "Longitude").dropna().
 ↪sample(False, 0.01, seed=42).toPandas()

coords = df_sampled_geo[["Latitude", "Longitude"]].values

# Apply KMeans (e.g., 10 clusters)
kmeans = KMeans(n_clusters=10, random_state=42)
df_sampled_geo["Cluster"] = kmeans.fit_predict(coords)
```

```
# Cluster Centers
centroids = kmeans.cluster_centers_

# Plot in folium
import folium
from folium.plugins import MarkerCluster

crime_map = folium.Map(location=[40.7128, -74.0060], zoom_start=11)

# Add cluster centroids
for lat, lon in centroids:
    folium.CircleMarker(
        location=[lat, lon],
        radius=12,
        color='red',
        fill=True,
        fill_color='red',
        fill_opacity=0.6,
        popup="Hotspot Center"
    ).add_to(crime_map)

# Add actual data points
marker_cluster = MarkerCluster().add_to(crime_map)
for _, row in df_sampled_geo.iterrows():
    folium.Marker(location=[row["Latitude"], row["Longitude"]],
                  icon=folium.Icon(color="blue", icon="info-sign"),
                  popup=f"Cluster: {row['Cluster']}").add_to(marker_cluster)

crime_map
```

[ ]: <folium.folium.Map at 0x7c07c0869010>


Paterson
Clifton
Newark
Elizabeth
Bayonne
Yonkers
New Rochelle
Hempstead
Levittown
1207
1098
14280
3243
685
9823
852
7897
3956
2955
200
949
8407
3949
16363
1842
1422
293
2842
5903
1556
1197
405
262
604
158
1306
488
834
4
10