Project - Crime Data of Boston

DESCRIPTION

Problem Statement

- Crime incident reports are provided by Boston Police Department (BPD) to document the initial details surrounding an incident to which BPD officers respond. This is a dataset containing records from the new crime incident report system, which includes a reduced set of fields focused on capturing the type of incident as well as when and where it occurred.

Goal

- The aim is to find in which city Crime count is highest
- In which year,month,day and hour Crime count is highest
- Which kind of offense is occuring for the highest number of time

Importing All the library

In [1]:

```
import sklearn
```

In [3]:

```
sklearn.__version__
```

Out[3]:

```
'0.21.3'
```

In [4]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
from statsmodels.formula.api import ols
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.decomposition import PCA
import xgboost as xgb
from IPython.display import Image
from sklearn.model_selection import train_test_split
from IPython.display import Image
import geopy
from geopy.geocoders import Nominatim
from geopy.extra.rate_limiter import RateLimiter
import matplotlib.pyplot as plt
import plotly_express as px
import tqdm
from tqdm._tqdm_notebook import tqdm_notebook
from pygeocoder import Geocoder
import reverse_geocoder as rg
import pprint
```

```
C:\Users\Subhasish Das\Anaconda3\lib\site-packages\dask\config.py:168: YAMLLo
adWarning: calling yaml.load() without Loader=... is deprecated, as the defau
lt Loader is unsafe. Please read https://msg.pyyaml.org/load for full detail
s.
  data = yaml.load(f.read()) or {}
C:\Users\Subhasish Das\Anaconda3\lib\site-packages\distributed\config.py:20:
YAMLLoadWarning: calling yaml.load() without Loader=... is deprecated, as the
default Loader is unsafe. Please read https://msg.pyyaml.org/load for full de
tails.
  defaults = yaml.load(f)
C:\Users\Subhasish Das\Anaconda3\lib\site-packages\ipykernel_launcher.py:25:
TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.*` instead of `tqdm._tqdm_notebook.*`
```

In [5]:

```
pd.set_option('display.max_row', 100000)
pd.set_option('display.max_columns',500000)
```

Loading Data Set

In [7]:

```
dt_crime=pd.read_csv("crime.csv",encoding= 'unicode_escape')
```

In [8]:

```
dt_crime.head()
```

Out[8]:

	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_CODE_GROUP	OFFENSE_DESCRIPTION	DIS
0	I182070945	619	Larceny	LARCENY ALL OTHERS	
1	I182070943	1402	Vandalism	VANDALISM	
2	I182070941	3410	Towed	TOWED MOTOR VEHICLE	
3	I182070940	3114	Investigate Property	INVESTIGATE PROPERTY	
4	I182070938	3114	Investigate Property	INVESTIGATE PROPERTY	

In [9]:

```
dt_crime.shape
```

Out[9]:

```
(319073, 17)
```

In [10]:

```
dt_crime.columns
```

Out[10]:

```
Index(['INCIDENT_NUMBER', 'OFFENSE_CODE', 'OFFENSE_CODE_GROUP',
       'OFFENSE_DESCRIPTION', 'DISTRICT', 'REPORTING_AREA', 'SHOOTING',
       'OCCURRED_ON_DATE', 'YEAR', 'MONTH', 'DAY_OF_WEEK', 'HOUR', 'UCR_PAR
T',
       'STREET', 'Lat', 'Long', 'Location'],
      dtype='object')
```

In [11]:

```
dt_crime.describe()
```

Out[11]:

	OFFENSE_CODE	YEAR	MONTH	HOUR	Lat	l
count	319073.000000	319073.000000	319073.000000	319073.000000	299074.000000	299074.00
mean	2317.546956	2016.560586	6.609719	13.118205	42.214381	-70.90
std	1185.285543	0.996344	3.273691	6.294205	2.159766	3.49
min	111.000000	2015.000000	1.000000	0.000000	-1.000000	-71.17
25%	1001.000000	2016.000000	4.000000	9.000000	42.297442	-71.09
50%	2907.000000	2017.000000	7.000000	14.000000	42.325538	-71.07
75%	3201.000000	2017.000000	9.000000	18.000000	42.348624	-71.06
max	3831.000000	2018.000000	12.000000	23.000000	42.395042	-1.00

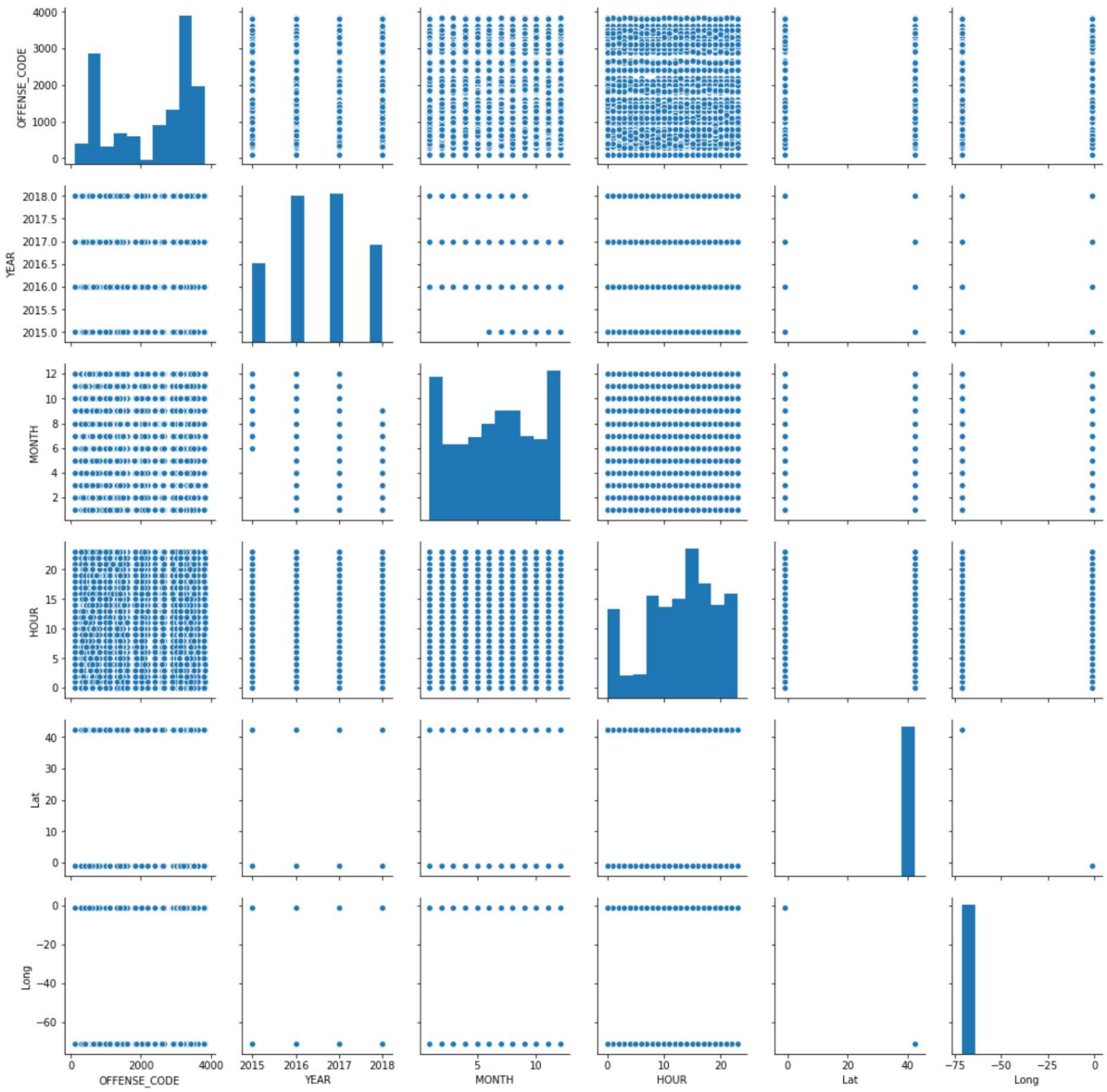
In [12]:

```
sns.pairplot(dt_crime)
```

```
C:\Users\Subhasish Das\Anaconda3\lib\site-packages\numpy\lib\histograms.py:82
4: RuntimeWarning: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
C:\Users\Subhasish Das\Anaconda3\lib\site-packages\numpy\lib\histograms.py:82
5: RuntimeWarning: invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```

Out[12]:

```
<seaborn.axisgrid.PairGrid at 0x2105f687f98>
```

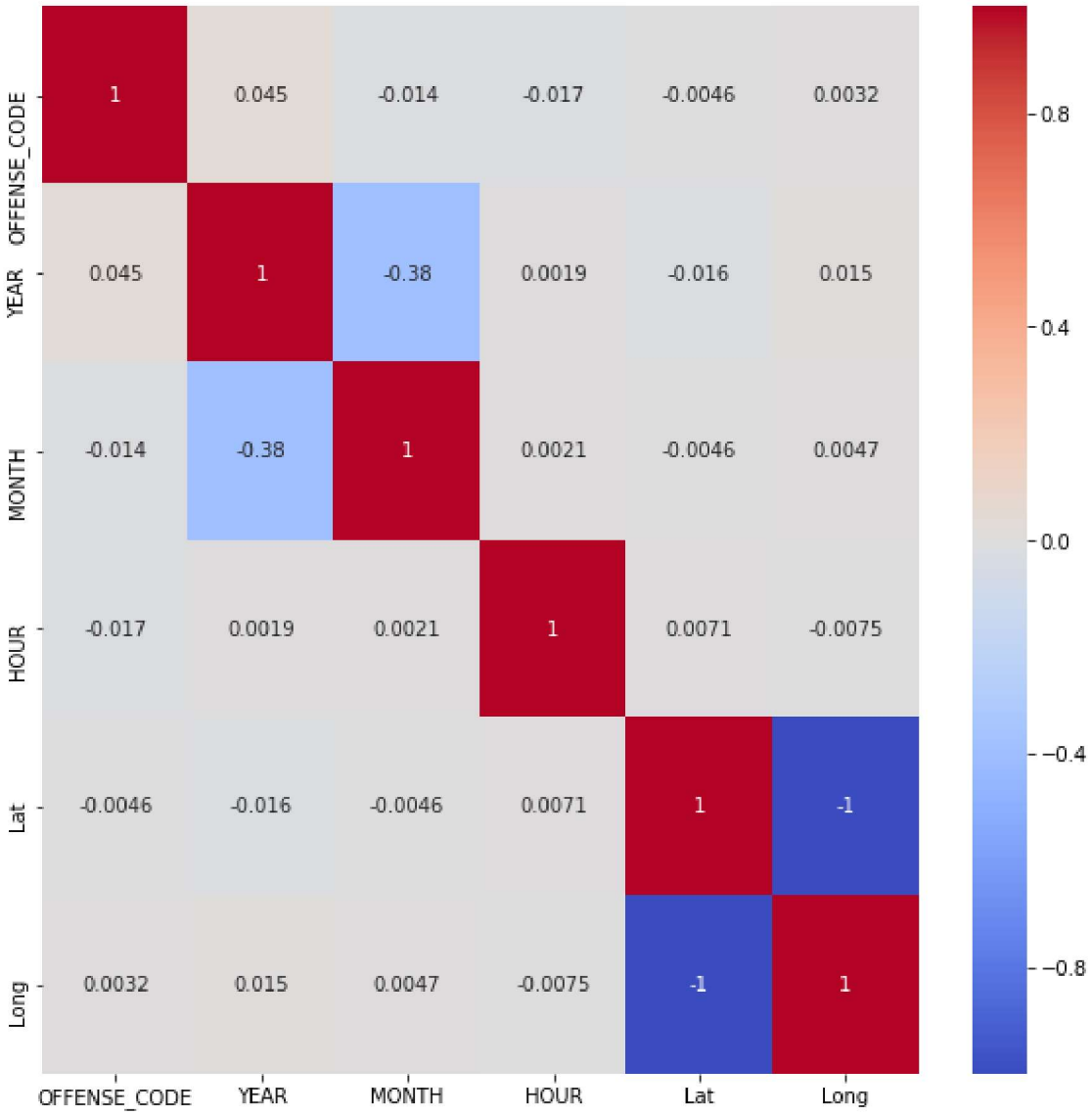

OFFENSE_CODE
YEAR
MONTH
HOUR
Lat
Long
OFFENSE_CODE
YEAR
MONTH
HOUR
Lat
Long

In [13]:

```
plt.figure(figsize=(10,10))
sns.heatmap(dt_crime.corr(),annot=True,cmap='coolwarm')
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2105fd97828>
```



Feature-Engineering

In [14]:

```
dt_crime.dtypes
```

Out[14]:

```
INCIDENT_NUMBER         object
OFFENSE_CODE             int64
OFFENSE_CODE_GROUP      object
OFFENSE_DESCRIPTION     object
DISTRICT                object
REPORTING_AREA          object
SHOOTING                object
OCCURRED_ON_DATE        object
YEAR                     int64
MONTH                    int64
DAY_OF_WEEK             object
HOUR                     int64
UCR_PART                object
STREET                  object
Lat                    float64
Long                   float64
Location                object
dtype: object
```

In [15]:

```
dt_crime.isnull().sum().sort_values(ascending=False)
```

Out[15]:

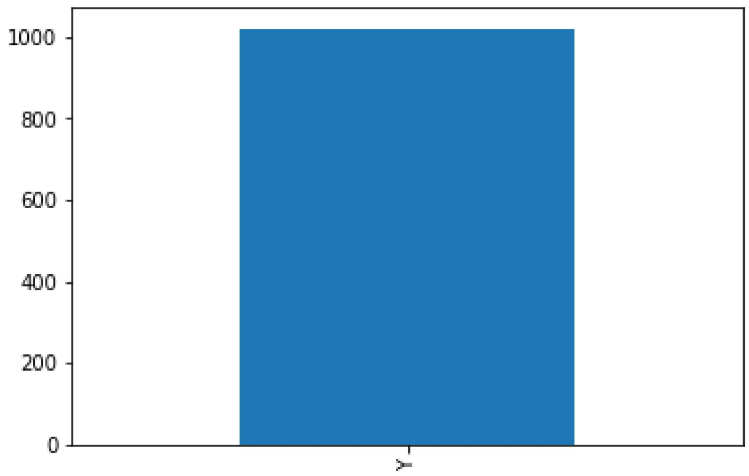
```
SHOOTING               318054
Lat                     19999
Long                    19999
STREET                  10871
DISTRICT                 1765
UCR_PART                   90
Location                    0
REPORTING_AREA              0
OFFENSE_CODE                0
OFFENSE_CODE_GROUP          0
OFFENSE_DESCRIPTION         0
YEAR                        0
OCCURRED_ON_DATE            0
MONTH                       0
DAY_OF_WEEK                 0
HOUR                        0
INCIDENT_NUMBER             0
dtype: int64
```

In [16]:

```
dt_crime['SHOOTING'].value_counts().plot.bar()
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2106614be80>
```



In [17]:

```
dt_crime.SHOOTING.unique()
```

Out[17]:

```
array([nan, 'Y'], dtype=object)
```

In [18]:

```
shooting_y=dt_crime[dt_crime['SHOOTING']=='Y']
```

In [19]:

```
print(len(shooting_y))
```

```
1019
```

In [20]:

```
print('Total:=', len(dt_crime))
```

```
Total:= 319073
```

In [21]:

```
print('Shooting as yes:', len(shooting_y)/len(dt_crime)*100)
```

```
Shooting as yes: 0.3193626536874007
```

In Shooting column % of yes is 0.3 so we can drop this column

In [22]:

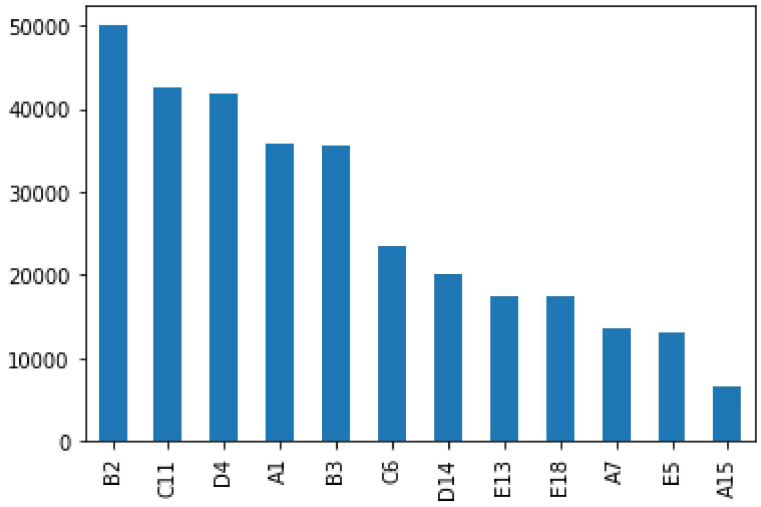
```
dt_crime.drop('SHOOTING',axis=1,inplace=True)
```

In [23]:

```
dt_crime['DISTRICT'].value_counts().plot.bar()
```

Out[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21065f17630>
```

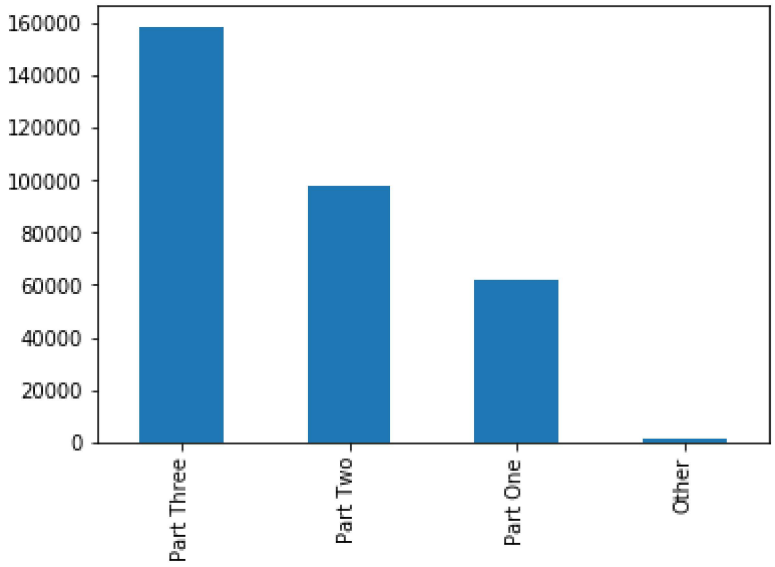


In [24]:

```
dt_crime['UCR_PART'].value_counts().plot.bar()
```

Out[24]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21065f88438>
```

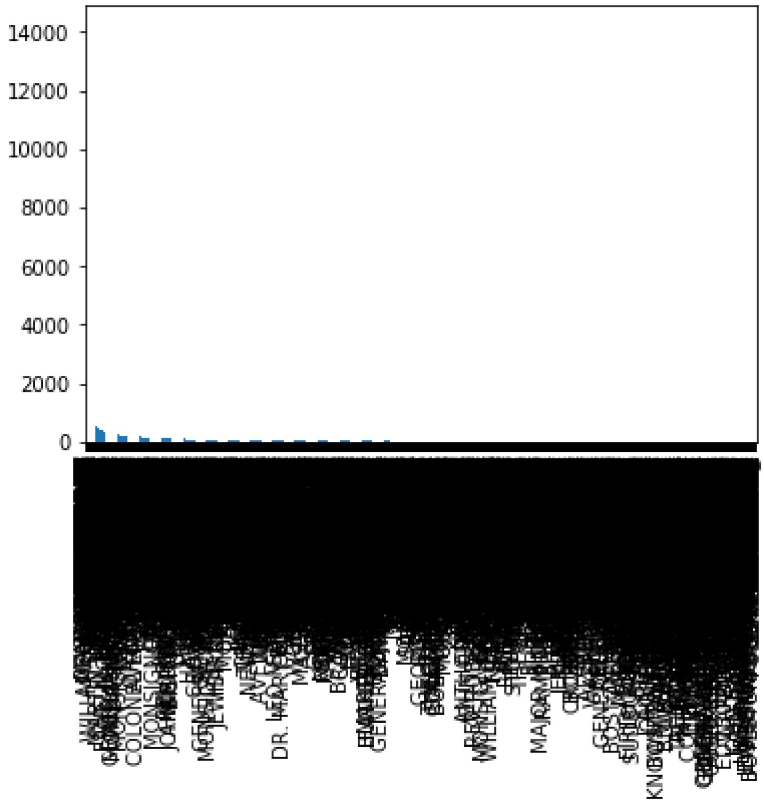


In [22]:

```
dt_crime['STREET'].value_counts().plot.bar()
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1af2c16eac8>
```



I will write a function to find the mode for each column which has null values
Then I will use this function to impute the columns to remove null values

In [25]:

```
def impute_nan(df,variable):
    most_freq_cat=df[variable].mode()[0]
    df[variable].fillna(most_freq_cat,inplace=True)
```

In [26]:

```
impute_nan(dt_crime,'DISTRICT')
```

In [27]:

```
impute_nan(dt_crime,'UCR_PART')
```

In [28]:

```
impute_nan(dt_crime,'STREET')
```

I have removed the null values for 'DISTRICT','UCR_PART','STREET'

In [29]:

```
dt_crime.isnull().sum()
```

Out[29]:

```
INCIDENT_NUMBER            0
OFFENSE_CODE               0
OFFENSE_CODE_GROUP         0
OFFENSE_DESCRIPTION        0
DISTRICT                   0
REPORTING_AREA             0
OCCURRED_ON_DATE           0
YEAR                       0
MONTH                      0
DAY_OF_WEEK                0
HOUR                       0
UCR_PART                   0
STREET                     0
Lat                    19999
Long                   19999
Location                   0
dtype: int64
```

Now I will fill the null values of Lattitude and Longitude with mode

In [30]:

```
median_lat=dt_crime.Lat.median()
median_long=dt_crime.Long.median()
```

In [31]:

```
dt_crime['Lat'].fillna(median_lat,inplace=True)
```

In [32]:

```
dt_crime['Long'].fillna(median_long,inplace=True)
```

In [33]:

```
dt_crime.isnull().sum()
```

Out[33]:

```
INCIDENT_NUMBER        0
OFFENSE_CODE           0
OFFENSE_CODE_GROUP     0
OFFENSE_DESCRIPTION    0
DISTRICT               0
REPORTING_AREA         0
OCCURRED_ON_DATE       0
YEAR                   0
MONTH                  0
DAY_OF_WEEK            0
HOUR                   0
UCR_PART               0
STREET                 0
Lat                    0
Long                   0
Location               0
dtype: int64
```

Now I will combine Lattitude and Longitude columns to create a new columns LatLong Column
And I will drop the location column

In [34]:

```
dt_crime['LatLong'] = list(zip(dt_crime.Lat, dt_crime.Long))
```

In [35]:

```
dt_crime.drop('Location',axis=1,inplace=True)
```

In [36]:

```
dt_crime.head()
```

Out[36]:

	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_CODE_GROUP	OFFENSE_DESCRIPTION	DIS
0	I182070945	619	Larceny	LARCENY ALL OTHERS	
1	I182070943	1402	Vandalism	VANDALISM	
2	I182070941	3410	Towed	TOWED MOTOR VEHICLE	
3	I182070940	3114	Investigate Property	INVESTIGATE PROPERTY	
4	I182070938	3114	Investigate Property	INVESTIGATE PROPERTY	

Now I will derive new columns like city,state,county from Lat long column

In [37]:

```
def reverseGeocode(coordinates):
    result = rg.search(coordinates)
    return (result)

if __name__=="__main__":

    # Coordinates tuple.Can contain more than one pair.
    coordinates =list(zip(dt_crime['Lat'],dt_crime['Long'])) # generates pair of (lat,lon
g)
    data = reverseGeocode(coordinates)

    dt_crime['name'] = [i['name'] for i in data]
    dt_crime['admin1'] = [i['admin1'] for i in data]
    dt_crime['admin2'] = [i['admin2'] for i in data]

dt_crime.to_csv("dt_crime.csv") # write to csv # result will be saved to data_appended.csv
```

```
Loading formatted geocoded file...
```

In [38]:

```
dt_crime.rename(columns={'name':'City','admin1':'State','admin2':'County'},inplace=True)
```

In [39]:

```
dt_crime.head()
```

Out[39]:

	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_CODE_GROUP	OFFENSE_DESCRIPTION	DIS
0	I182070945	619	Larceny	LARCENY ALL OTHERS	
1	I182070943	1402	Vandalism	VANDALISM	
2	I182070941	3410	Towed	TOWED MOTOR VEHICLE	
3	I182070940	3114	Investigate Property	INVESTIGATE PROPERTY	
4	I182070938	3114	Investigate Property	INVESTIGATE PROPERTY	

In [40]:

```
dt_crime.columns
```

Out[40]:

```
Index(['INCIDENT_NUMBER', 'OFFENSE_CODE', 'OFFENSE_CODE_GROUP',
       'OFFENSE_DESCRIPTION', 'DISTRICT', 'REPORTING_AREA', 'OCCURRED_ON_DAT
E',
       'YEAR', 'MONTH', 'DAY_OF_WEEK', 'HOUR', 'UCR_PART', 'STREET', 'Lat',
       'Long', 'LatLong', 'City', 'State', 'County'],
      dtype='object')
```

In [41]:

```
dt_crime.City.unique()
```

Out[41]:

```
array(['Brookline', 'South Boston', 'Boston', 'Milton', 'Jamaica Plain',
       'Takoradi', 'Chelsea', 'Dedham', 'Watertown', 'Cambridge',
       'Somerville', 'Winthrop', 'Revere', 'Everett'], dtype=object)
```

In [42]:

```
dt_crime.OFFENSE_CODE_GROUP.unique()
```

Out[42]:

```
array(['Larceny', 'Vandalism', 'Towed', 'Investigate Property',
       'Motor Vehicle Accident Response', 'Auto Theft', 'Verbal Disputes',
       'Robbery', 'Fire Related Reports', 'Other', 'Property Lost',
       'Medical Assistance', 'Assembly or Gathering Violations',
       'Larceny From Motor Vehicle', 'Residential Burglary',
       'Simple Assault', 'Restraining Order Violations', 'Violations',
       'Harassment', 'Ballistics', 'Property Found',
       'Police Service Incidents', 'Drug Violation', 'Warrant Arrests',
       'Disorderly Conduct', 'Property Related Damage',
       'Missing Person Reported', 'Investigate Person', 'Fraud',
       'Aggravated Assault', 'License Plate Related Incidents',
       'Firearm Violations', 'Other Burglary', 'Arson', 'Bomb Hoax',
       'Harbor Related Incidents', 'Counterfeiting', 'Liquor Violation',
       'Firearm Discovery', 'Landlord/Tenant Disputes',
       'Missing Person Located', 'Auto Theft Recovery', 'Service',
       'Operating Under the Influence', 'Confidence Games',
       'Search Warrants', 'License Violation', 'Commercial Burglary',
       'HOME INVASION', 'Recovered Stolen Property',
       'Offenses Against Child / Family', 'Prostitution', 'Evading Fare',
       'Prisoner Related Incidents', 'Homicide', 'Embezzlement',
       'Explosives', 'Criminal Harassment', 'Phone Call Complaints',
       'Aircraft', 'Biological Threat', 'Manslaughter', 'Gambling',
       'INVESTIGATE PERSON', 'HUMAN TRAFFICKING',
       'HUMAN TRAFFICKING - INVOLUNTARY SERVITUDE',
       'Burglary - No Property Taken'], dtype=object)
```

Data Visualization Plots

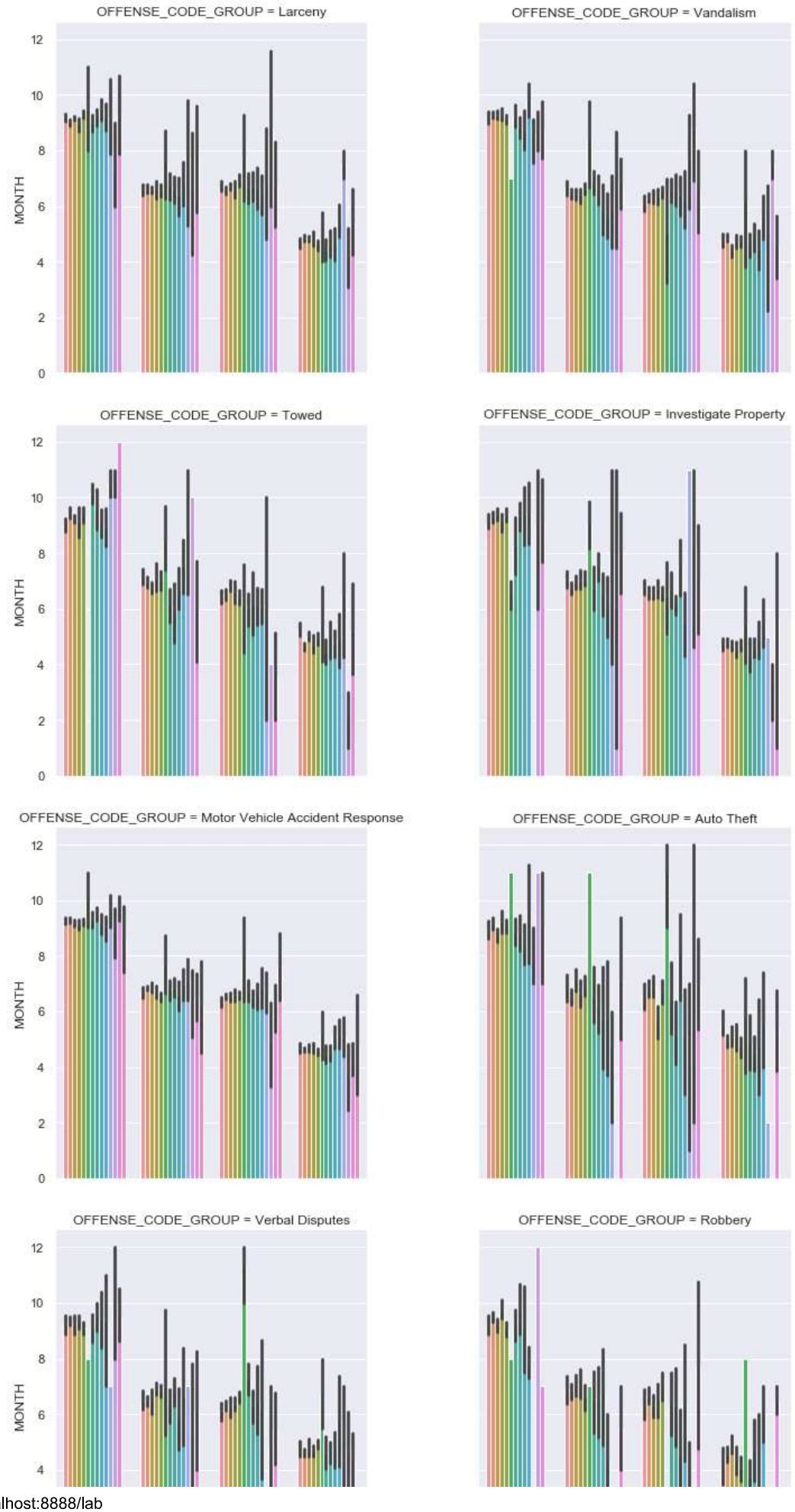
Mapping year,month,city,OFFENSE_CODE_GROUP using seaborn

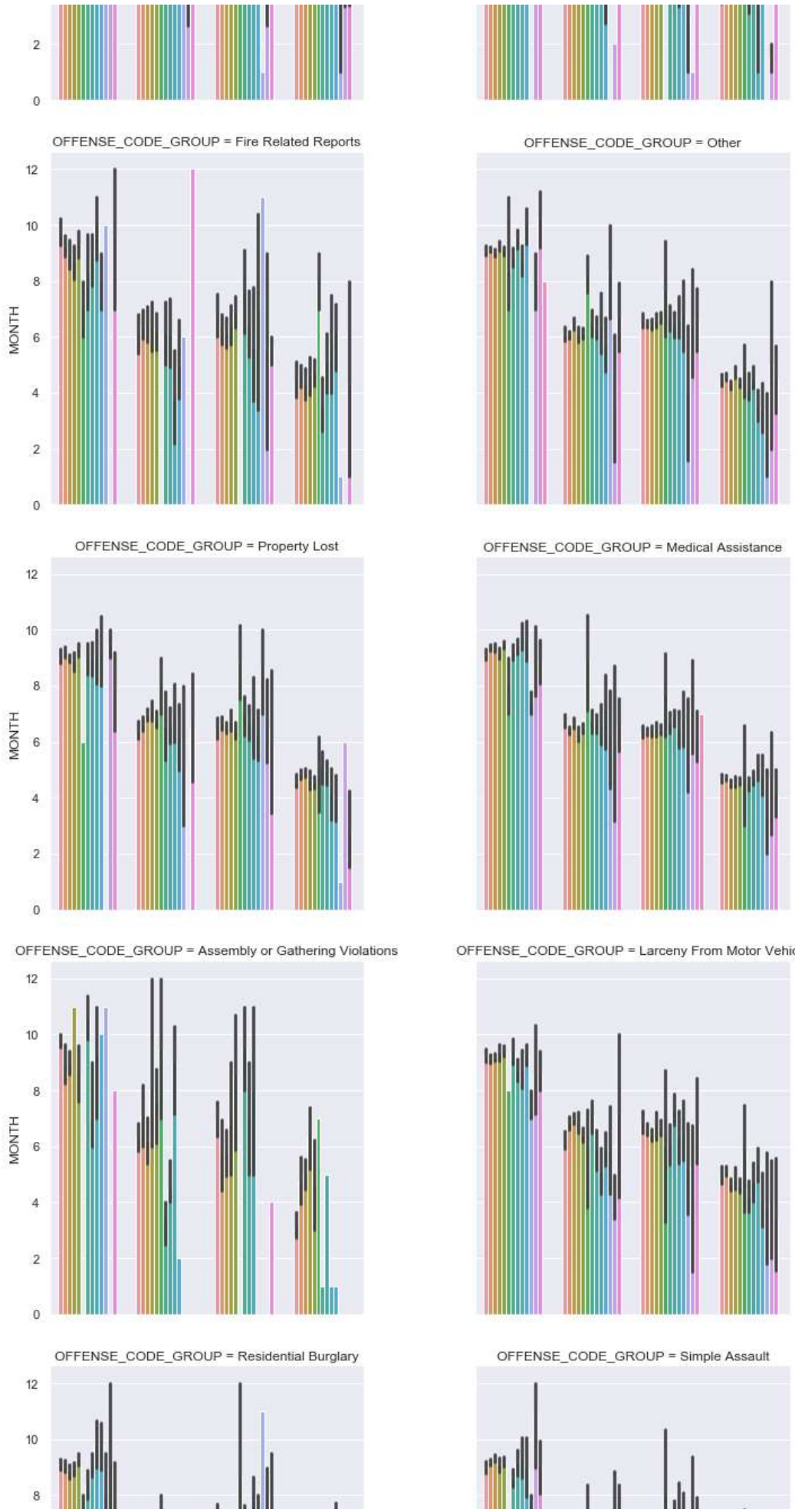
In [40]:

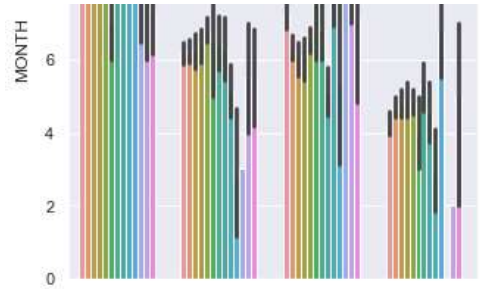
```
sns.set(style='darkgrid', font_scale=1.0)
sns.catplot(x='YEAR',
            y='MONTH',hue='City',
            col='OFFENSE_CODE_GROUP',
            col_wrap=2,
            data=dt_crime,
            kind='bar')
```

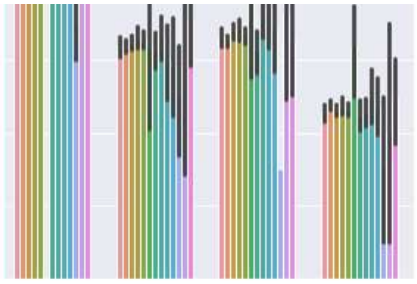
Out[40]:

```
<seaborn.axisgrid.FacetGrid at 0x207344568d0>
```

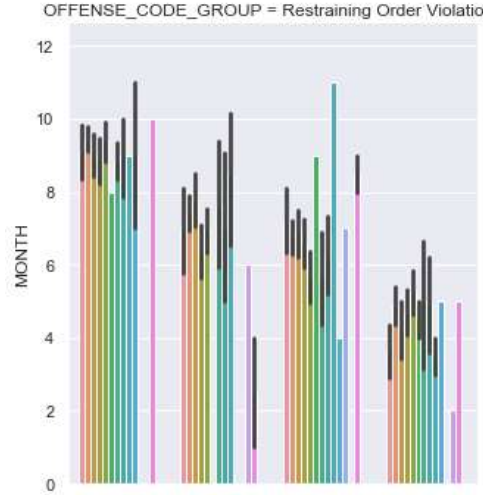

OFFENSE_CODE_GROUP = Larceny
OFFENSE_CODE_GROUP = Vandalism
OFFENSE_CODE_GROUP = Towed
OFFENSE_CODE_GROUP = Investigate Property
OFFENSE_CODE_GROUP = Motor Vehicle Accident Response
OFFENSE_CODE_GROUP = Auto Theft
OFFENSE_CODE_GROUP = Verbal Disputes
OFFENSE_CODE_GROUP = Robbery
MONTH


OFFENSE_CODE_GROUP = Fire Related Reports
OFFENSE_CODE_GROUP = Other
OFFENSE_CODE_GROUP = Property Lost
OFFENSE_CODE_GROUP = Medical Assistance
OFFENSE_CODE_GROUP = Assembly or Gathering Violations
OFFENSE_CODE_GROUP = Larceny From Motor Vehicle
OFFENSE_CODE_GROUP = Residential Burglary
OFFENSE_CODE_GROUP = Simple Assault
MONTH
0
2
4
6
8
10
12

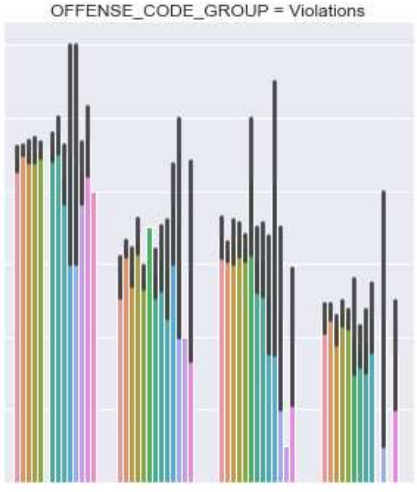




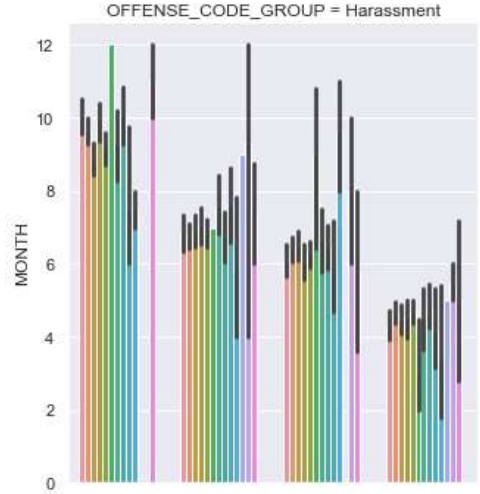
OFFENSE_CODE_GROUP = Restraining Order Violations



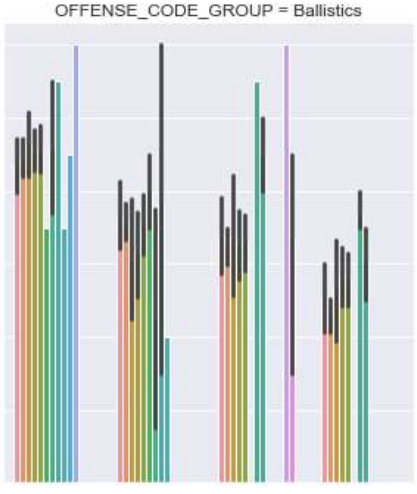
OFFENSE_CODE_GROUP = Violations



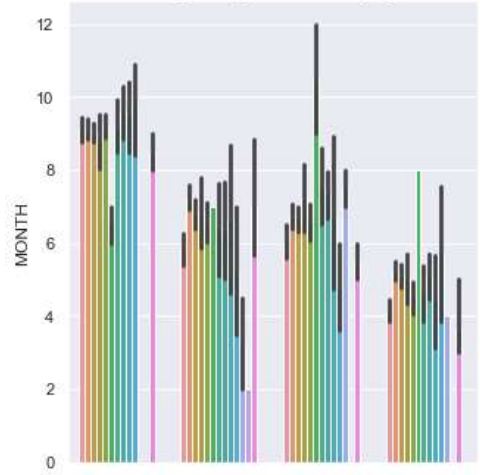
OFFENSE_CODE_GROUP = Harassment



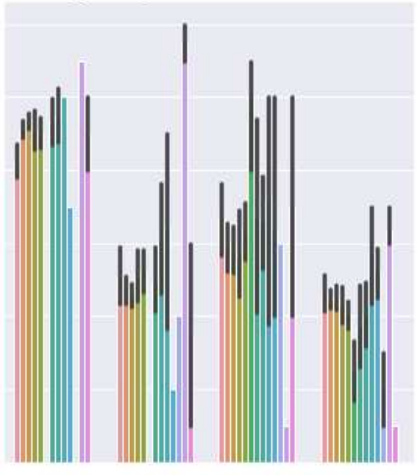
OFFENSE_CODE_GROUP = Ballistics



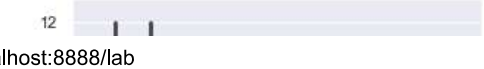
OFFENSE_CODE_GROUP = Property Found



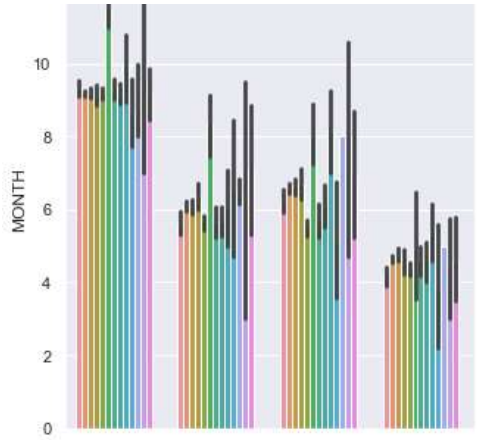
OFFENSE_CODE_GROUP = Police Service Incidents

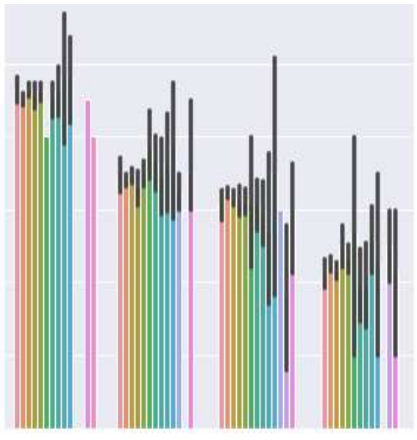


OFFENSE_CODE_GROUP = Drug Violation

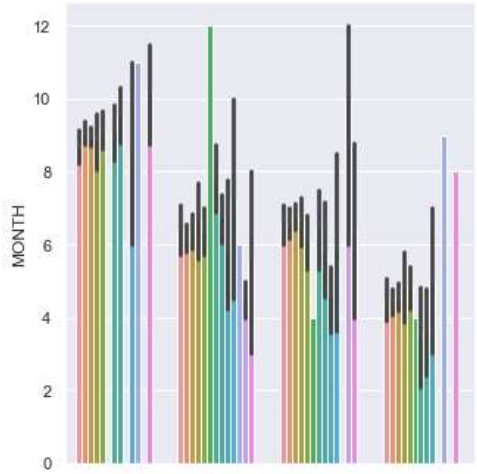


OFFENSE_CODE_GROUP = Warrant Arrests

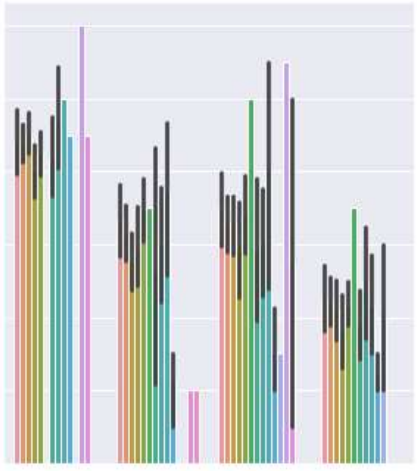




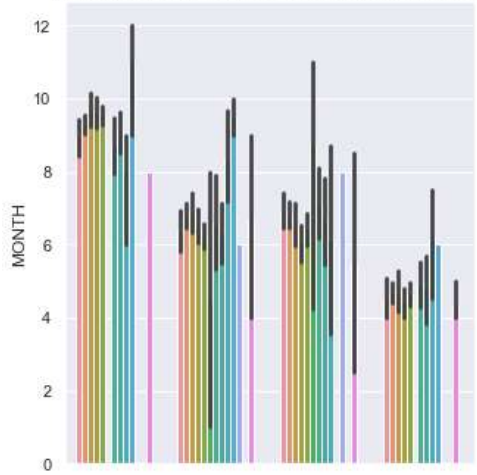
OFFENSE_CODE_GROUP = Disorderly Conduct



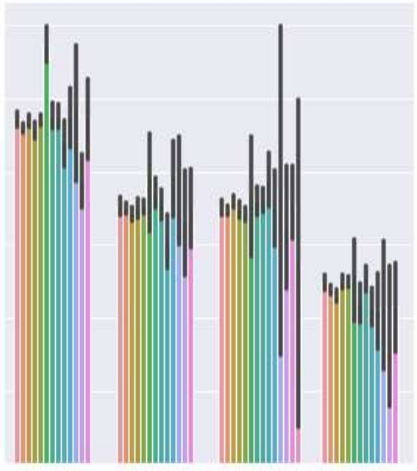
OFFENSE_CODE_GROUP = Property Related Damage



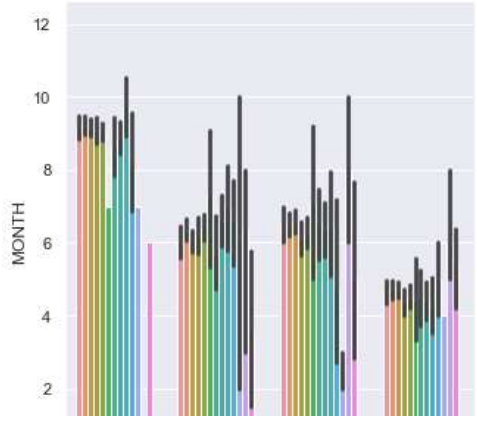
OFFENSE_CODE_GROUP = Missing Person Reported



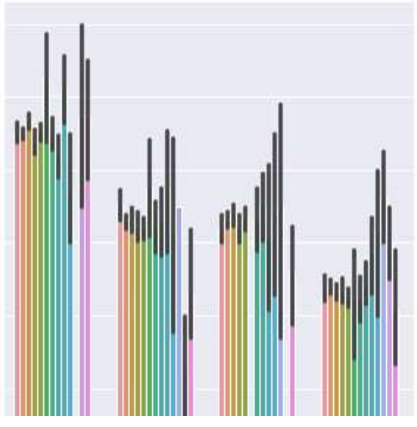
OFFENSE_CODE_GROUP = Investigate Person

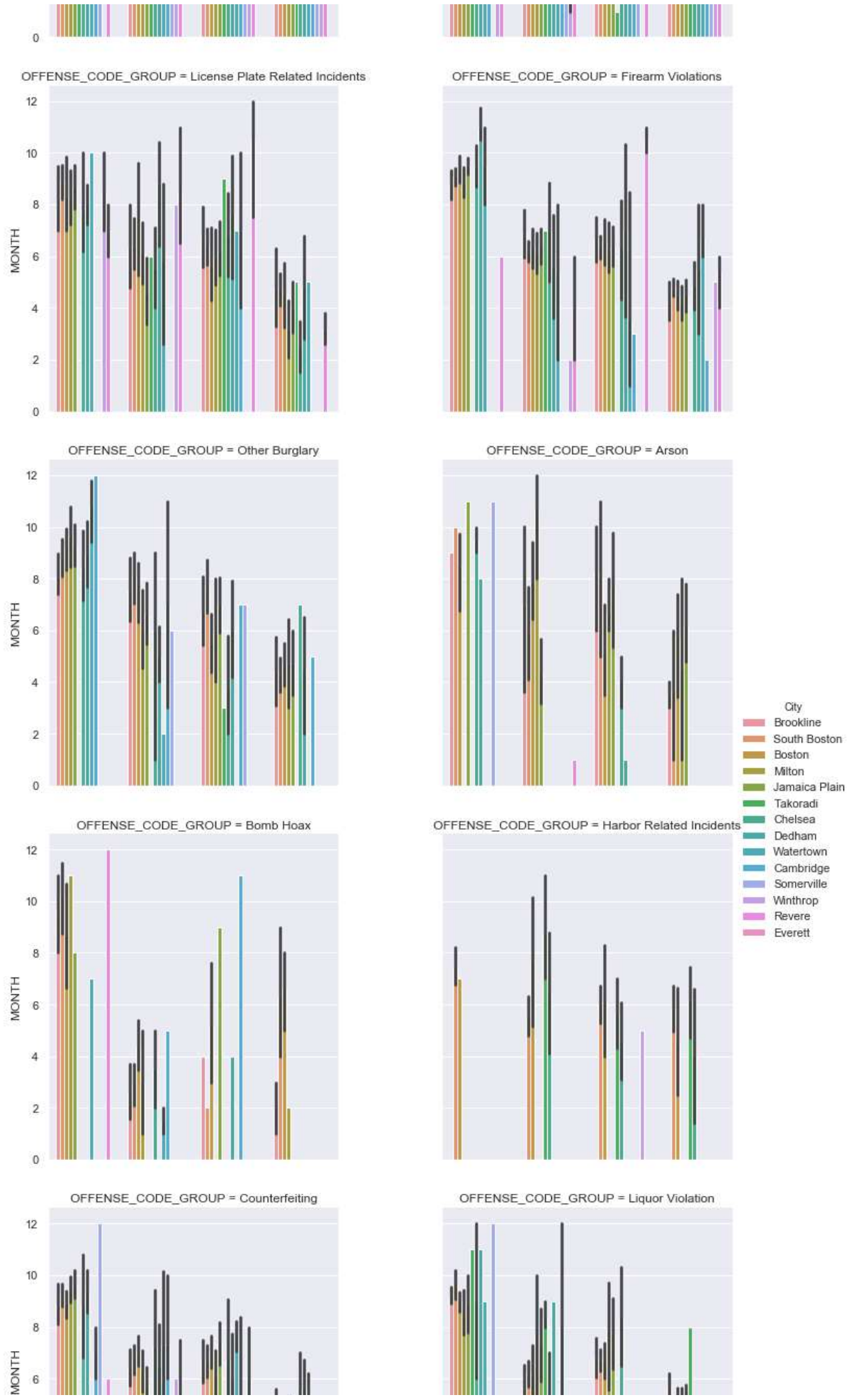


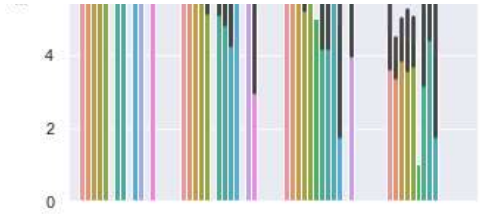
OFFENSE_CODE_GROUP = Fraud

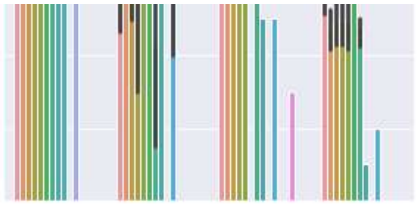


OFFENSE_CODE_GROUP = Aggravated Assault

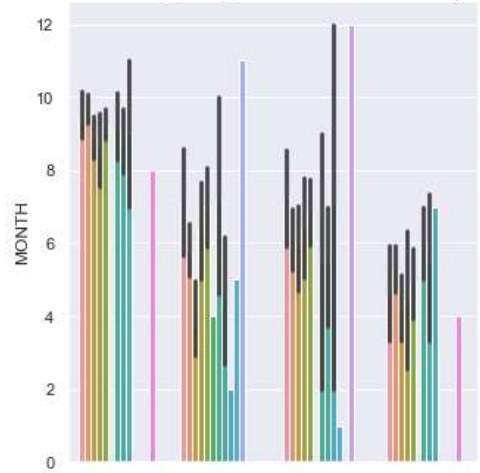



OFFENSE_CODE_GROUP = License Plate Related Incidents
OFFENSE_CODE_GROUP = Firearm Violations
OFFENSE_CODE_GROUP = Other Burglary
OFFENSE_CODE_GROUP = Arson
OFFENSE_CODE_GROUP = Bomb Hoax
OFFENSE_CODE_GROUP = Harbor Related Incidents
OFFENSE_CODE_GROUP = Counterfeiting
OFFENSE_CODE_GROUP = Liquor Violation
MONTH
City
Brookline
South Boston
Boston
Milton
Jamaica Plain
Takoradi
Chelsea
Dedham
Watertown
Cambridge
Somerville
Winthrop
Revere
Everett

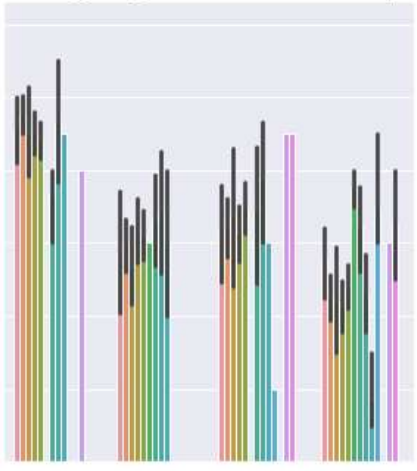




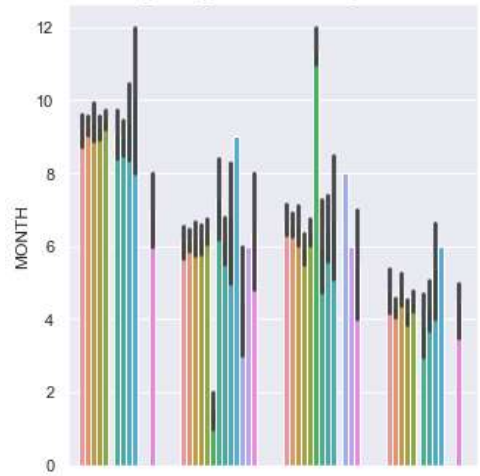
OFFENSE_CODE_GROUP = Firearm Discovery



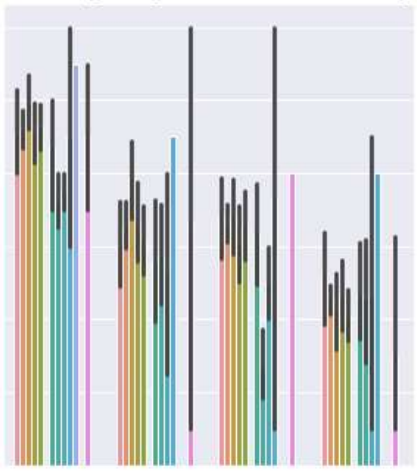
OFFENSE_CODE_GROUP = Landlord/Tenant Disputes



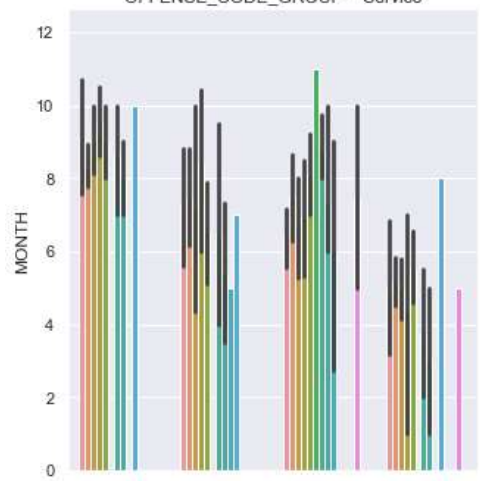
OFFENSE_CODE_GROUP = Missing Person Located



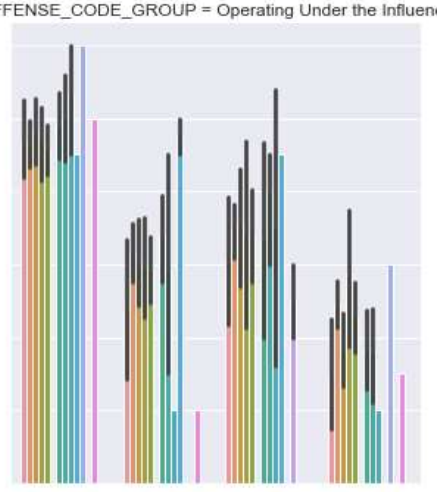
OFFENSE_CODE_GROUP = Auto Theft Recovery



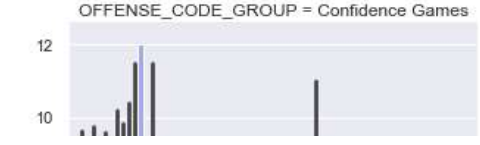
OFFENSE_CODE_GROUP = Service



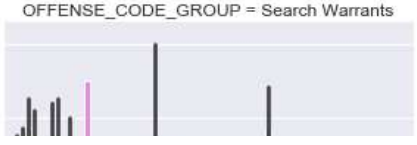
OFFENSE_CODE_GROUP = Operating Under the Influence

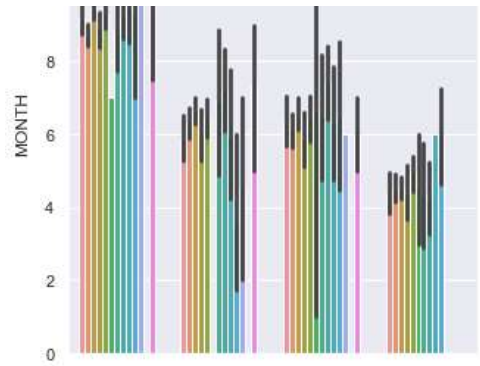


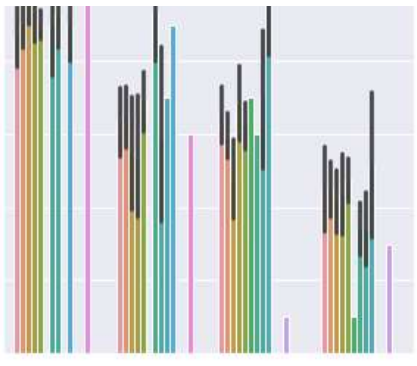
OFFENSE_CODE_GROUP = Confidence Games

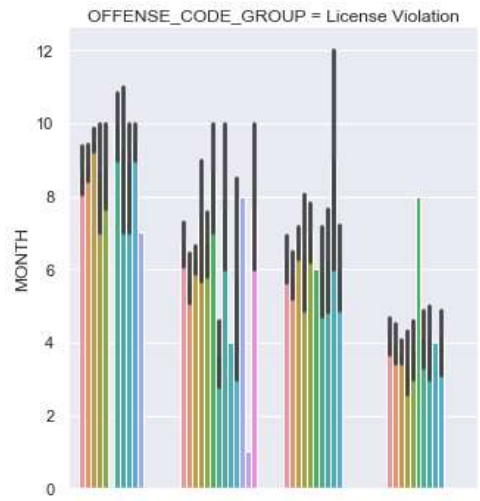


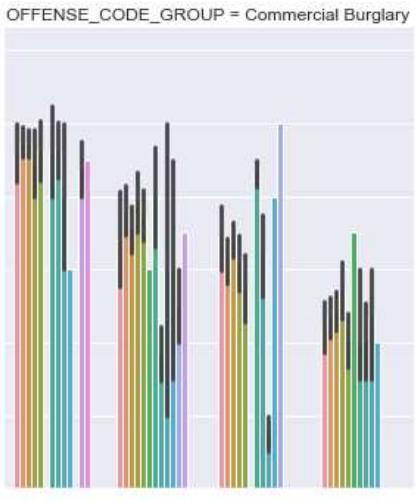
OFFENSE_CODE_GROUP = Search Warrants

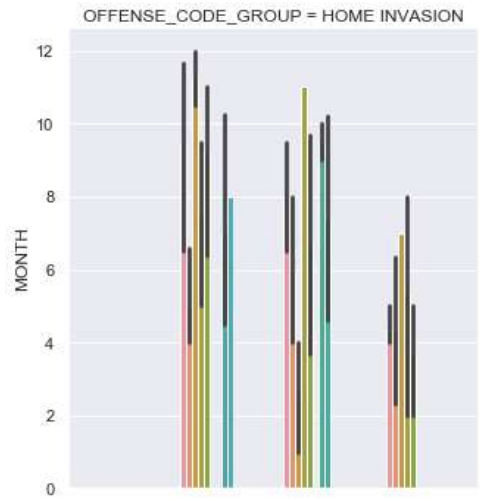


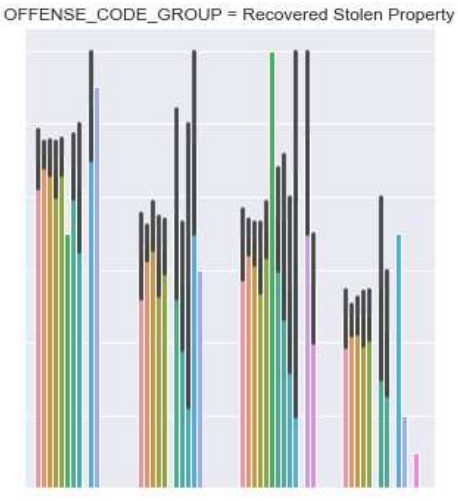

MONTH
0
2
4
6
8

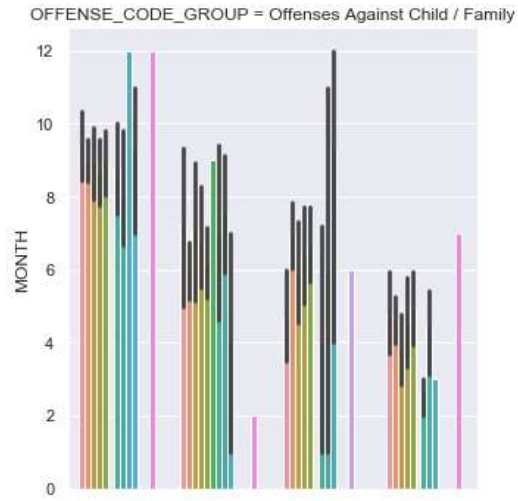


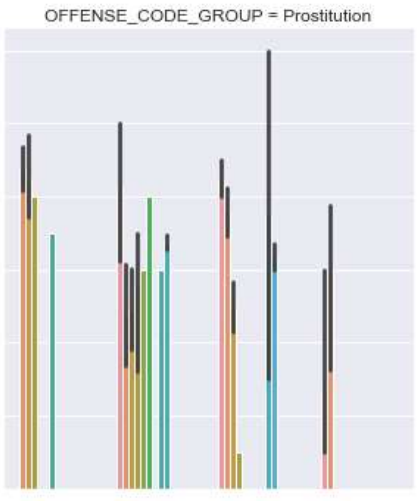

OFFENSE_CODE_GROUP = License Violation
MONTH
0
2
4
6
8
10
12

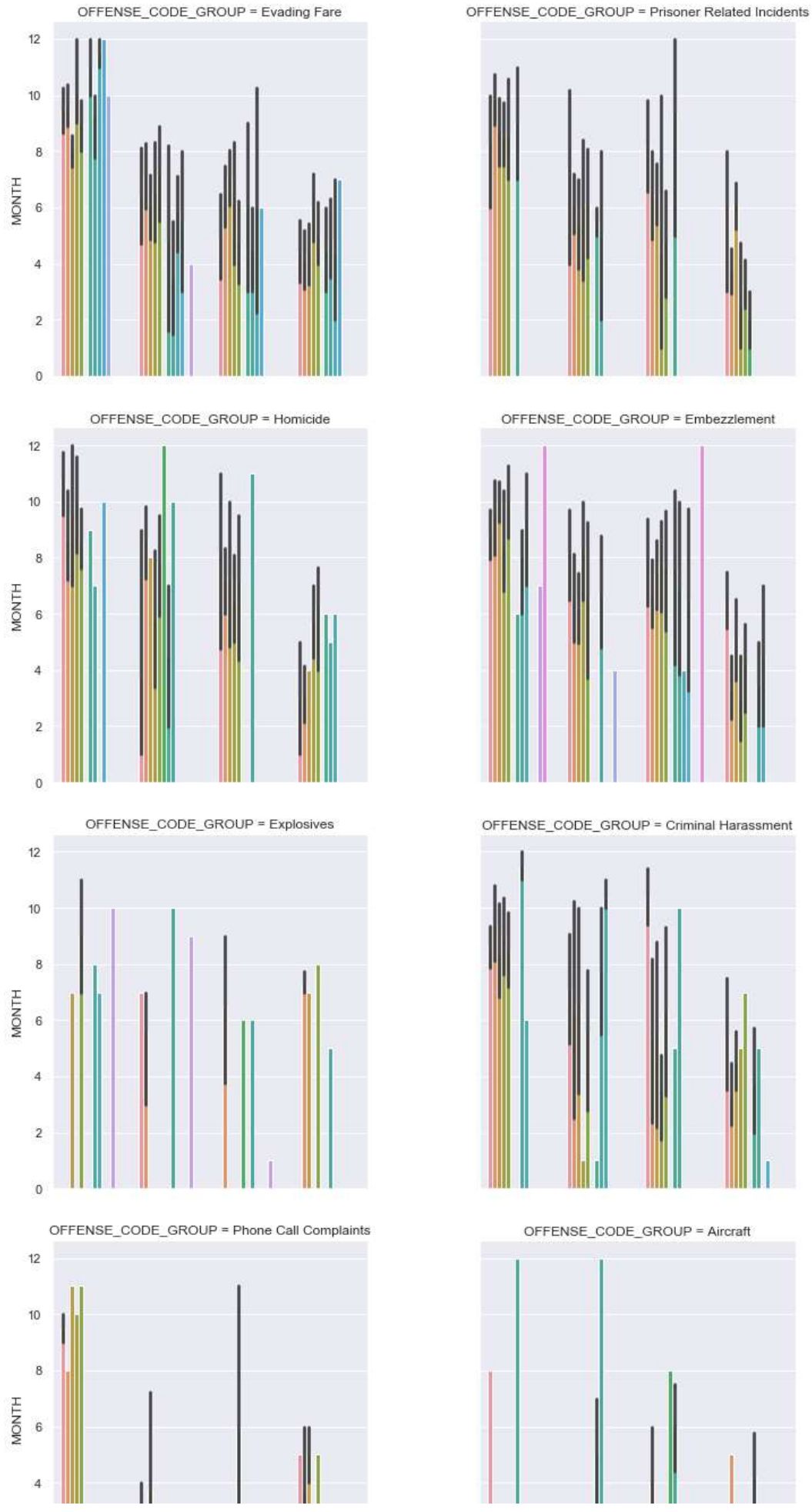

OFFENSE_CODE_GROUP = Commercial Burglary

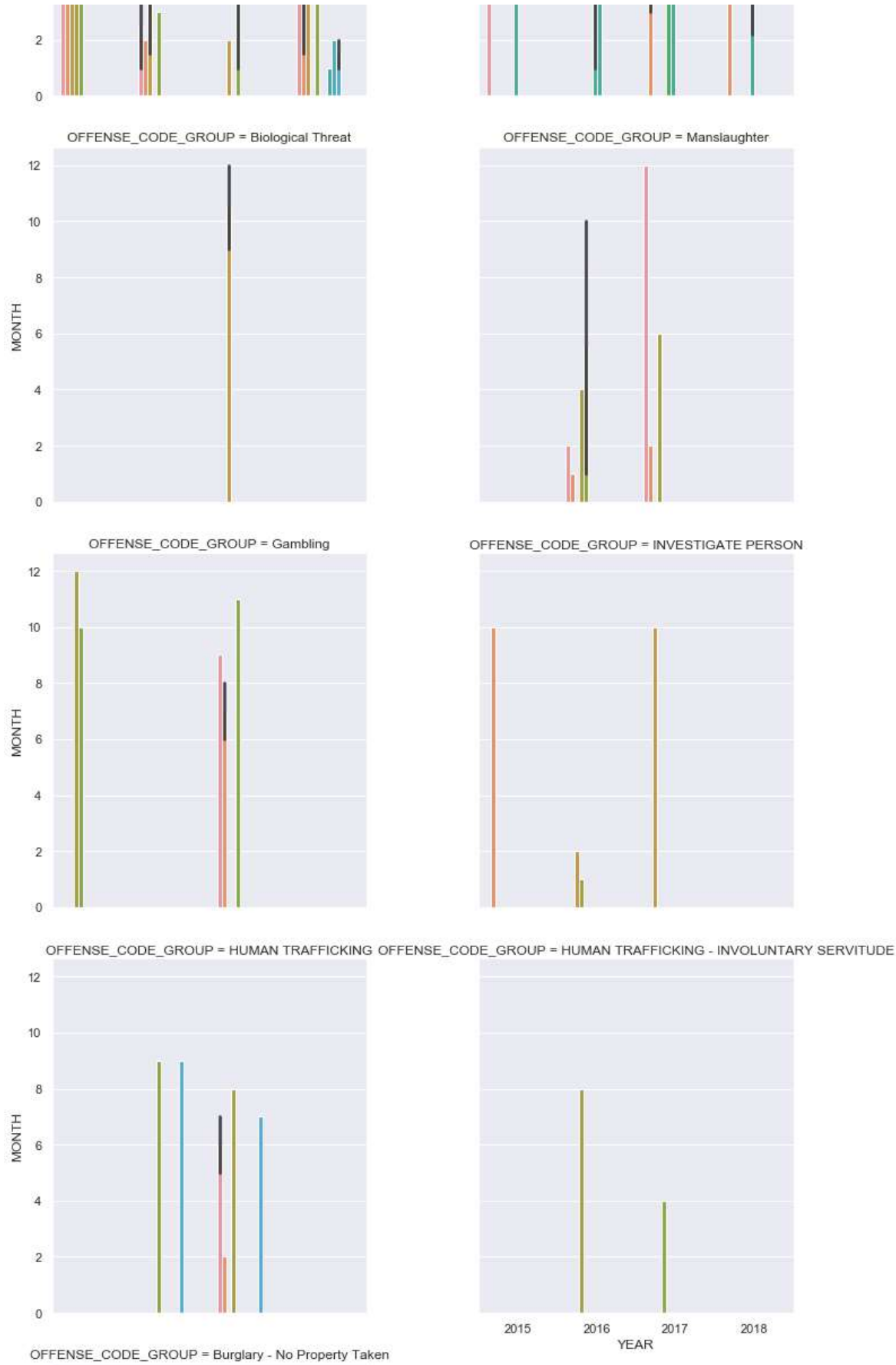

OFFENSE_CODE_GROUP = HOME INVASION
MONTH
0
2
4
6
8
10
12

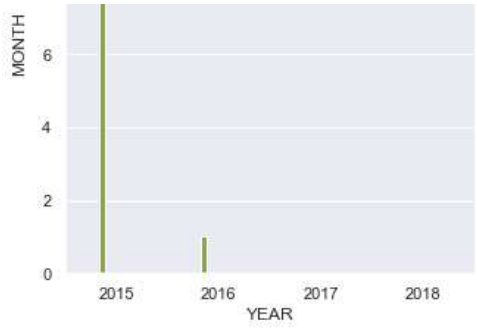

OFFENSE_CODE_GROUP = Recovered Stolen Property


OFFENSE_CODE_GROUP = Offenses Against Child / Family
MONTH
0
2
4
6
8
10
12


OFFENSE_CODE_GROUP = Prostitution


OFFENSE_CODE_GROUP = Evading Fare
OFFENSE_CODE_GROUP = Prisoner Related Incidents
OFFENSE_CODE_GROUP = Homicide
OFFENSE_CODE_GROUP = Embezzlement
OFFENSE_CODE_GROUP = Explosives
OFFENSE_CODE_GROUP = Criminal Harassment
OFFENSE_CODE_GROUP = Phone Call Complaints
OFFENSE_CODE_GROUP = Aircraft
MONTH
0
2
4
6
8
10
12


OFFENSE_CODE_GROUP = Biological Threat
OFFENSE_CODE_GROUP = Manslaughter
OFFENSE_CODE_GROUP = Gambling
OFFENSE_CODE_GROUP = INVESTIGATE PERSON
OFFENSE_CODE_GROUP = HUMAN TRAFFICKING
OFFENSE_CODE_GROUP = HUMAN TRAFFICKING - INVOLUNTARY SERVITUDE
OFFENSE_CODE_GROUP = Burglary - No Property Taken
MONTH
YEAR
2015
2016
2017
2018



In [42]:

```
import folium
```

In [43]:

```
dt_crime.shape
```

Out[43]:

```
(319073, 19)
```

Now I will plot the lattitude and Longitude with city and OFFENSE_CODE_GROUP using google map
As we can see the shape of the dataset is 319073. So we will plot first 50 rows and last 50 rows

In [58]:

```
import folium
```

In [59]:

```
dt_1=dt_crime.head(50)
```

In [60]:

```
latitude = 42.30682138
longitude = -71.06030035
traffic_map = folium.Map(location=[latitude, longitude], zoom_start=5)
```

In [61]:

```
colordict = {0: 'lightblue', 1: 'lightgreen', 2: 'orange', 3: 'red'}
```

In [62]:

```
for lat, lon, city,OFFENSE_CODE_GROUP in zip(dt_1['Lat'], dt_1['Long'], dt_1['City'], dt_1
['OFFENSE_CODE_GROUP']):

    folium.CircleMarker(
        [lat, lon],

        popup = ('City: ' + str(city).capitalize() + '<br>'
                 'OFFENSE_CODE_GROUP: ' + str(OFFENSE_CODE_GROUP)

                ),
        color='b',


        fill=True,
        fill_opacity=0.7
        ).add_to(traffic_map)

display(traffic_map)
```

Make this Notebook Trusted to load map: File -> Trust Notebook

In [63]:

```
dt_2=dt_crime.tail(50)
```

In [64]:

```
for lat, lon, city,OFFENSE_CODE_GROUP in zip(dt_2['Lat'], dt_2['Long'], dt_2['City'], dt_2
['OFFENSE_CODE_GROUP']):

    folium.CircleMarker(
        [lat, lon],

        popup = ('City: ' + str(city).capitalize() + '<br>'
                 'OFFENSE_CODE_GROUP: ' + str(OFFENSE_CODE_GROUP)

                ),
        color='b',


        fill=True,
        fill_opacity=0.7
        ).add_to(traffic_map)

display(traffic_map)
```

Make this Notebook Trusted to load map: File -> Trust Notebook

In [65]:

```
dt = dt_crime.groupby(['YEAR','City','Lat','Long','MONTH',])['OFFENSE_CODE_GROUP'].count()
```

In [66]:

```
def histogram(data,path,color,title,xaxis,yaxis):
    fig = px.histogram(data, x=path,color=color)
    fig.update_layout(
        title_text=title,
        xaxis_title_text=xaxis,
        yaxis_title_text=yaxis,
        bargap=0.2,
        bargroupgap=0.1
    )
    fig.show()
```

In [67]:

```
histogram(dt_crime,"OFFENSE_CODE_GROUP","OFFENSE_CODE_GROUP",'Major Crimes in Boston','Cri
me','Count')
```

In [68]:

```
Number_crimes = dt_crime['OFFENSE_CODE_GROUP'].value_counts()
values = Number_crimes.values
categories = pd.DataFrame(data=Number_crimes.index, columns=["OFFENSE_CODE_GROUP"])
categories['values'] = values
```

In [69]:

```
def treemap(categories,title,path,values):
    fig = px.treemap(categories, path=path, values=values, height=700,
                 title=title, color_discrete_sequence = px.colors.sequential.RdBu)
    fig.data[0].textinfo = 'label+text+value'
    fig.show()
```

In [70]:

```
treemap(categories,'Major Crimes in Boston',['OFFENSE_CODE_GROUP'],categories['values'])
```

In [71]:

```
def bar(categories,x,y,color,title,xlab,ylab):
    fig = px.bar(categories, x=x, y=y,
             color=color,
             height=400)
    fig.update_layout(
    title_text=title,
    xaxis_title_text=xlab,
    yaxis_title_text=ylab,
    bargap=0.2,
    bargroupgap=0.1
    )
    fig.show()
```

In [72]:

```
bar(categories,categories['OFFENSE_CODE_GROUP'][0:10],categories['values'][0:10]
    ,categories['OFFENSE_CODE_GROUP'][0:10],'Top 10 Major Crimes in Boston','Crime','Coun
t')
```

Plotting Number of Crime Per year

In [73]:

```
Number_crimes_year = dt_crime['YEAR'].value_counts()
years = pd.DataFrame(data=Number_crimes_year.index, columns=["YEAR"])
years['values'] = Number_crimes_year.values
```

In [74]:

```
import plotly.express as px
```

In [75]:

```
fig = px.pie(years, values='values', names='YEAR', color_discrete_sequence=px.colors.seque
ntial.RdBu)
fig.show()
```

In [76]:

```
import plotly.graph_objects as go
```

In [77]:

```
Number_crimes_month = dt_crime['MONTH'].value_counts()
months = pd.DataFrame(data=Number_crimes_month.index, columns=["MONTH"])
months['values'] = Number_crimes_month.values
```

In [78]:

```
fig = go.Figure(go.Bar(
            x=months['values'],
            y=months['MONTH'],
        marker=dict(
            color='rgb(13,143,129)',

        ),
            orientation='h'))
fig.update_layout(
    title_text='Crimes in Boston as per month',
    xaxis_title_text='Count',
    yaxis_title_text='Month',
    bargap=0.2,
    bargroupgap=0.1
)
fig.show()
```

In [79]:

```
Number_crimes_days = dt_crime['DAY_OF_WEEK'].value_counts()
days = pd.DataFrame(data=Number_crimes_days.index, columns=["DAY_OF_WEEK"])
days['values'] = Number_crimes_days.values
```

In [80]:

```
fig = px.histogram(dt_crime, y="DAY_OF_WEEK",color="DAY_OF_WEEK")
fig.update_layout(
    title_text='Crime count per day',
    xaxis_title_text='Day',
    yaxis_title_text='Crimes Count',
    bargap=0.2,
    bargroupgap=0.1
)
fig.show()
```

In [81]:

```
fig = go.Figure(data=[go.Pie(labels=days['DAY_OF_WEEK'], values=days['values'], hole=.4)])
fig.update_layout(
    title_text='Crime count on each day',
)
fig.show()
```

In [82]:

```
histogram(dt_crime,"HOUR","HOUR",'Crime count on each Hour','Hour','Count')
```

In [83]:

```
histogram(dt_crime,"YEAR","YEAR",'Crime count on each Hour','Hour','Count')
```

In [84]:

```
histogram(dt_crime,"YEAR","MONTH",'Crime count on each year per month','Year','Crimes Coun
t')
```

In [85]:

```
Number_crimes_city = dt_crime['City'].value_counts()
city = pd.DataFrame(data=Number_crimes_city.index, columns=["City"])
city['values'] = Number_crimes_city.values
```

In [86]:

```
bar(city,city['City'][0:10],city['values'][0:10]
    ,city['City'][0:10],'Top 10 Crime count on each City','City',' Crime Count')
```

In [87]:

```
histogram(dt_crime,"OFFENSE_CODE_GROUP","YEAR",'Crime count per Category on each Year','Ca
tegory','Crimes Count on each Year')
```

In [88]:

```
histogram(dt_crime,"OFFENSE_CODE_GROUP","MONTH",'Crime count per Category on each Month',
'Category','Crimes Count on each Month')
```

In [89]:

```
histogram(dt_crime,"MONTH","DAY_OF_WEEK",'Crime count per Month on each Day','Month','Crim
es Count on each Day')
```

In [90]:

```
histogram(dt_crime,"DAY_OF_WEEK","HOUR",'Crime count per Day on each Hour','Day','Crimes C
ount on each Hour')
```

In [91]:

```
histogram(dt_crime,"DAY_OF_WEEK","HOUR",'Crime count per Day on each Hour','Day','Crimes C
ount on each Hour')
```

Conclusion

From this above graph we can conclude

1. South Boston has the highest crime count
2. On Friday the claim count is highest
3. On 17th Hour claim count is highest
4. In 2016 and 2017 crime count is highest and almost same
5. 'Motor Vehicle Accident Response' is most occured Offence Code

In [ ]:

In [ ]: