- IMPORTING THE NEEDED LIBRARIES

In [1]:
```
# to read and wrangle data
import pandas as pd
import numpy as np
import seaborn as sns
from pointpats import centrography
from datetime import datetime
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from scipy.stats import ttest_ind



# to create spatial data
import geopandas as gpd
from shapely.geometry import MultiPolygon


# for basemaps
import contextily as ctx
import leafmap.kepler as leafmap

# For spatial statistics
import esda
from esda.moran import Moran, Moran_Local
from pysal.lib import weights
from libpysal.weights import Queen, KNN
from splot import esda as esdaplot

import splot
from splot.esda import moran_scatterplot, plot_moran, lisa_cluster,plot_moran_simulation
from IPython.display import display, Markdown, display_latex, display_markdown, display_html

import libpysal as lps

# Graphics
```

```
import matplotlib.pyplot as plt
import plotly.express as px
import warnings
warnings.filterwarnings("ignore")

import squarify
from matplotlib.colors import Normalize
```

In [ ]:

In [2]:
```
raw_file = "tmp75yd8t76.csv"
```

In [3]:
```
df = pd.read_csv(raw_file)
```

In [4]:
```
df.head()
```

Out[4]:

	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_CODE_GROUP	OFFENSE_DESCRIPTION	DISTRICT	REPORTING_AREA	SHOOTIN
0	232099233	3115	NaN	INVESTIGATE PERSON	C11	355	
1	242008994	1102	NaN	FRAUD - FALSE PRETENSE / SCHEME	E5	691	
2	232079828	2914	NaN	VAL - OPERATING W/O AUTHORIZATION LAWFUL	B2	321	
3	232083964	1102	NaN	FRAUD - FALSE PRETENSE / SCHEME	C6	200	
4	232022077	1001	NaN	FORGERY / COUNTERFEITING	E5	680	

- ## Data Preprocessing and EDA

In [5]:
```
df.shape
```

Out[5]: (84337, 17)

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 84337 entries, 0 to 84336
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   INCIDENT_NUMBER      84337 non-null  object 
 1   OFFENSE_CODE         84337 non-null  int64  
 2   OFFENSE_CODE_GROUP   0 non-null      float64
 3   OFFENSE_DESCRIPTION  84337 non-null  object 
 4   DISTRICT             84126 non-null  object 
 5   REPORTING_AREA       84337 non-null  object 
 6   SHOOTING             84337 non-null  int64  
 7   OCCURRED_ON_DATE     84337 non-null  object 
 8   YEAR                 84337 non-null  int64  
 9   MONTH                84337 non-null  int64  
 10  DAY_OF_WEEK          84337 non-null  object 
 11  HOUR                 84337 non-null  int64  
 12  UCR_PART             0 non-null      float64
 13  STREET               84337 non-null  object 
 14  Lat                  78340 non-null  float64
 15  Long                 78340 non-null  float64
 16  Location             78340 non-null  object 
dtypes: float64(4), int64(5), object(8)
memory usage: 10.9+ MB
```

In [7]:
```
df.isna().sum()
```

Out[7]:
```
INCIDENT_NUMBER            0
OFFENSE_CODE               0
OFFENSE_CODE_GROUP     84337
OFFENSE_DESCRIPTION        0
DISTRICT                 211
REPORTING_AREA             0
SHOOTING                   0
OCCURRED_ON_DATE           0
YEAR                       0
MONTH                      0
DAY_OF_WEEK                0
HOUR                       0
UCR_PART               84337
STREET                     0
Lat                     5997
Long                    5997
Location                5997
dtype: int64
```

In [8]:
```
df_1 = df.drop(["OFFENSE_CODE_GROUP", "UCR_PART"], axis = 1)
```

In [9]:
```
df["DISTRICT"].value_counts()
```

Out[9]:
```
DISTRICT
D4            11379
B2            11197
A1            10278
C11            9800
B3             8538
C6             7186
D14            6097
E13            5082
E18            4530
A7             4456
E5             4009
A15            1483
External         89
Outside of        2
Name: count, dtype: int64
```

In [10]:
```
map_data = df_1.dropna(subset=['Lat', 'Long'])
```

In [11]:
```
map_data_1= map_data.dropna()
```

In [13]:
```
map_data_1.shape
```

Out[13]:
```
(78284, 15)
```

In [14]:
```
map_data_1.duplicated().sum()
```

Out[14]:
```
0
```

In [16]:
```
map_data_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 78284 entries, 0 to 84336
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   INCIDENT_NUMBER      78284 non-null  object 
 1   OFFENSE_CODE         78284 non-null  int64  
 2   OFFENSE_DESCRIPTION  78284 non-null  object 
 3   DISTRICT             78284 non-null  object 
 4   REPORTING_AREA       78284 non-null  object 
 5   SHOOTING             78284 non-null  int64  
 6   OCCURRED_ON_DATE     78284 non-null  object 
 7   YEAR                 78284 non-null  int64  
 8   MONTH                78284 non-null  int64  
 9   DAY_OF_WEEK          78284 non-null  object 
 10  HOUR                 78284 non-null  int64  
 11  STREET               78284 non-null  object 
 12  Lat                  78284 non-null  float64
 13  Long                 78284 non-null  float64
 14  Location             78284 non-null  object 
dtypes: float64(2), int64(5), object(8)
memory usage: 9.6+ MB
```

In [17]:
```
data_new_1 = map_data_1.copy()
```

In [18]:
```
data_new_1.head()
```

Out[18]:

	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_DESCRIPTION	DISTRICT	REPORTING_AREA	SHOOTING	OCCURRED_ON_DATE
0	232099233	3115	INVESTIGATE PERSON	C11	355	0	2023-01-01 00:00:00+00
1	242008994	1102	FRAUD - FALSE PRETENSE / SCHEME	E5	691	0	2023-01-01 00:00:00+00
2	232079828	2914	VAL - OPERATING W/O AUTHORIZATION LAWFUL	B2	321	0	2023-01-01 00:00:00+00
3	232083964	1102	FRAUD - FALSE PRETENSE / SCHEME	C6	200	0	2023-01-01 00:00:00+00
4	232022077	1001	FORGERY / COUNTERFEITING	E5	680	0	2023-01-01 00:00:00+00

In [19]:
```
# Group by location and count crimes
crime_counts = map_data.groupby('Lat').size().reset_index(name='CRIME_PER_TRACT')
```

In [20]:
```
crime_counts.head()
```

Out[20]:

	Lat	CRIME_PER_TRACT
0	1.327335e-07	1
1	4.217278e+01	1
2	4.219160e+01	1
3	4.223266e+01	1
4	4.223309e+01	2

In [21]:
```
new_df = pd.merge(data_new_1, crime_counts, on ="Lat" , how='left')
```

In [22]:
```
new_df.head()
```

Out[22]:

	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_DESCRIPTION	DISTRICT	REPORTING_AREA	SHOOTING	OCCURRED_ON_DATE
0	232099233	3115	INVESTIGATE PERSON	C11	355	0	2023-01-01 00:00:00+00
1	242008994	1102	FRAUD - FALSE PRETENSE / SCHEME	E5	691	0	2023-01-01 00:00:00+00
2	232079828	2914	VAL - OPERATING W/O AUTHORIZATION LAWFUL	B2	321	0	2023-01-01 00:00:00+00
3	232083964	1102	FRAUD - FALSE PRETENSE / SCHEME	C6	200	0	2023-01-01 00:00:00+00
4	232022077	1001	FORGERY / COUNTERFEITING	E5	680	0	2023-01-01 00:00:00+00

In [23]:
```
new_df["DISTRICT"].value_counts()
```

Out[23]:
```
DISTRICT
D4            10902
B2            10558
C11            9314
A1             8569
B3             7942
C6             6613
D14            5910
E13            4870
E18            4298
A7             4049
E5             3822
A15            1388
External         47
Outside of        2
Name: count, dtype: int64
```

In [24]:
```
# getting rid of external district values
crime_districts = new_df[(new_df['DISTRICT'] != 'External') & (new_df["DISTRICT"]!="Outside of")]
```

In [25]:
```
crime_districts.head()
```

Out[25]:

	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_DESCRIPTION	DISTRICT	REPORTING_AREA	SHOOTING	OCCURRED_ON_DATE
0	232099233	3115	INVESTIGATE PERSON	C11	355	0	2023-01-01 00:00:00+00
1	242008994	1102	FRAUD - FALSE PRETENSE / SCHEME	E5	691	0	2023-01-01 00:00:00+00
2	232079828	2914	VAL - OPERATING W/O AUTHORIZATION LAWFUL	B2	321	0	2023-01-01 00:00:00+00
3	232083964	1102	FRAUD - FALSE PRETENSE / SCHEME	C6	200	0	2023-01-01 00:00:00+00
4	232022077	1001	FORGERY / COUNTERFEITING	E5	680	0	2023-01-01 00:00:00+00

In [26]:
```
crime_districts['OCCURRED_ON_DATE'] = pd.to_datetime(crime_districts['OCCURRED_ON_DATE'])
```

In [27]:
```
crime_districts['Date'] = crime_districts['OCCURRED_ON_DATE'].dt.date
```

In [28]:
```
crime_date_df = crime_districts['Date'].value_counts().to_frame()
```

In [29]:
```
crime_date_df.reset_index(inplace=True)
```

In [32]:
```
crime_date_df = crime_date_df.rename(columns = {'Count':'Date', 'Date':'Count'})
```

In [33]:
```
crime_date_df  = crime_date_df.sort_values(by='Date')
```

In [34]:
```
crime_date_df["count"].mean()
```

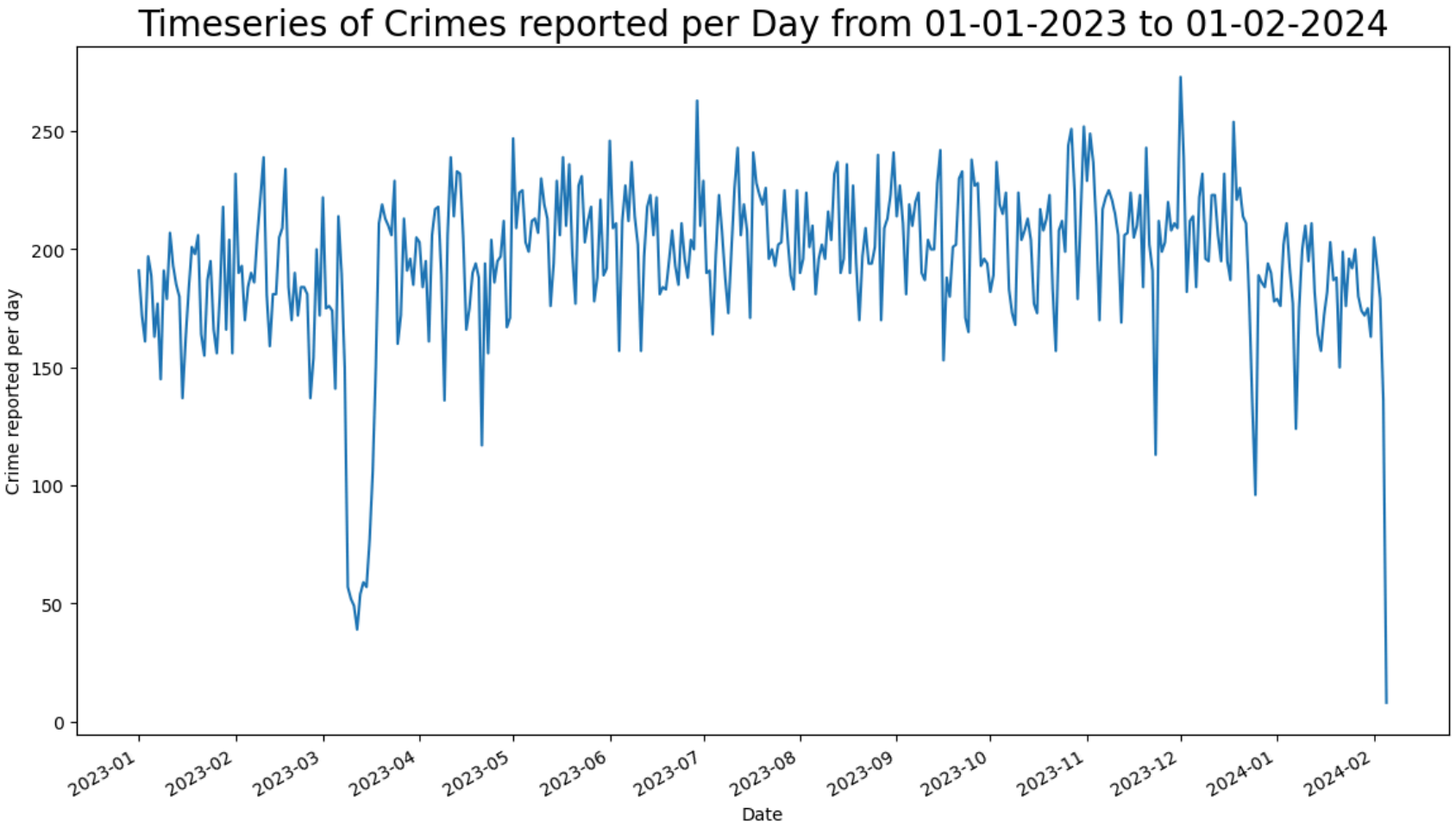
Out[34]:
```
195.0997506234414
```

- **Data Visualization**

In [35]:
```
import matplotlib.dates as mdates
```

In [36]:
```
fig, ax = plt.subplots(figsize=(14, 8))
half_year_locator = mdates.MonthLocator(interval=1)
ax.xaxis.set_major_locator(half_year_locator)
year_month_formatter = mdates.DateFormatter('%Y-%m')
ax.xaxis.set_major_formatter(year_month_formatter)
ax.plot(crime_date_df['Date'],crime_date_df['count'])

# rotate/align x axis and label axes
fig.autofmt_xdate()
plt.xlabel('Date')
plt.ylabel("Crime reported per day")
plt.title('Timeseries of Crimes reported per Day from 01-01-2023 to 01-02-2024', fontsize=20)
```

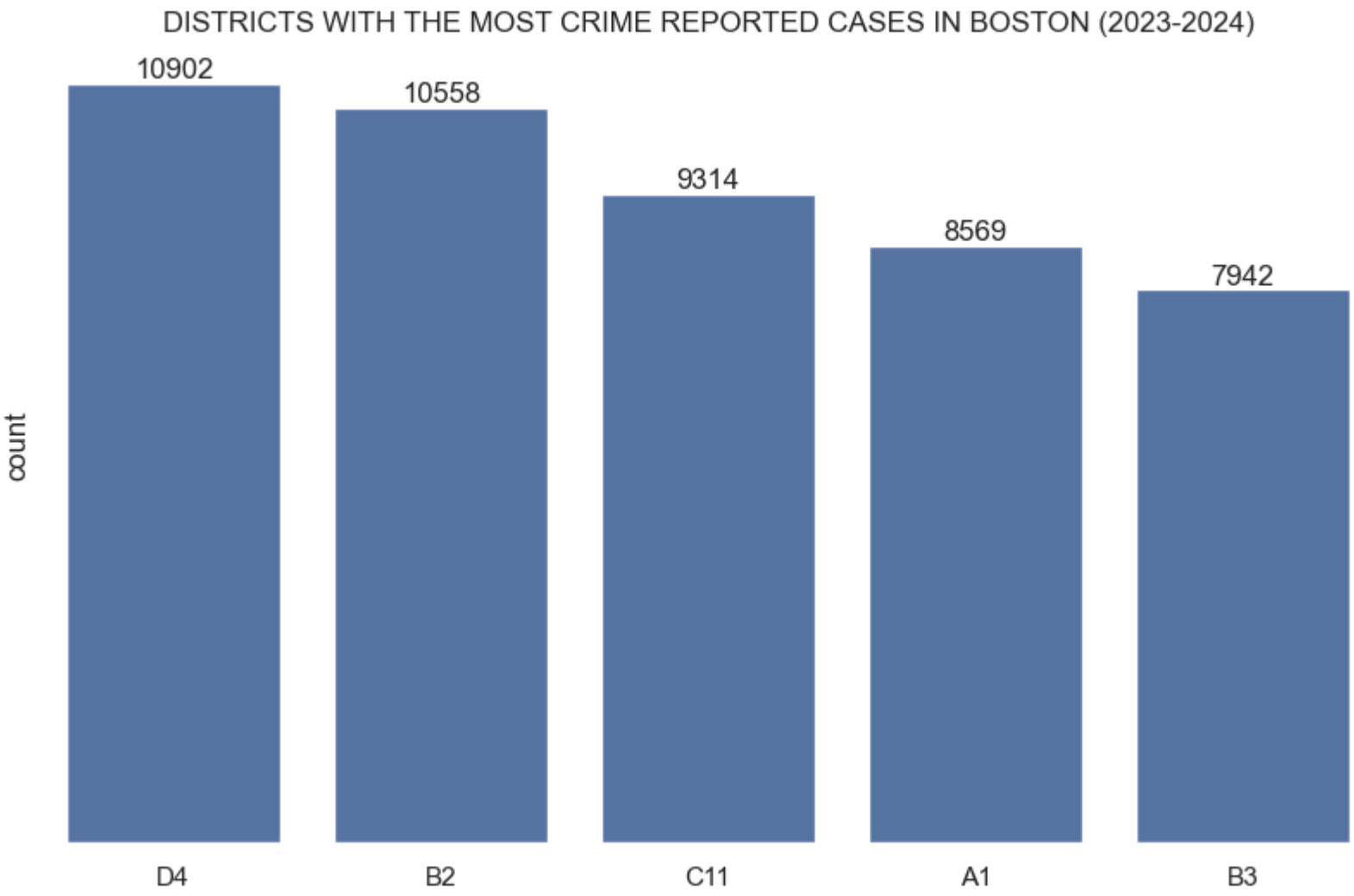
Out[36]:
```
Text(0.5, 1.0, 'Timeseries of Crimes reported per Day from 01-01-2023 to 01-02-2024')
```



- From the diagram above we could observed that reported crime in Boston decline massively at the second week of March 2023 and peak again in the last week of December 2023. Reported crime in 2024 shows a fluctuation around the mean, we can confirm this same pattern at the beginning of 2023.

In [37]:
```
plt.figure(figsize=(10, 6))
sns.set(style='darkgrid')
ax = sns.countplot(x = 'DISTRICT',
                   data = crime_districts,
                   order = crime_districts['DISTRICT'].value_counts().nlargest().index)
```

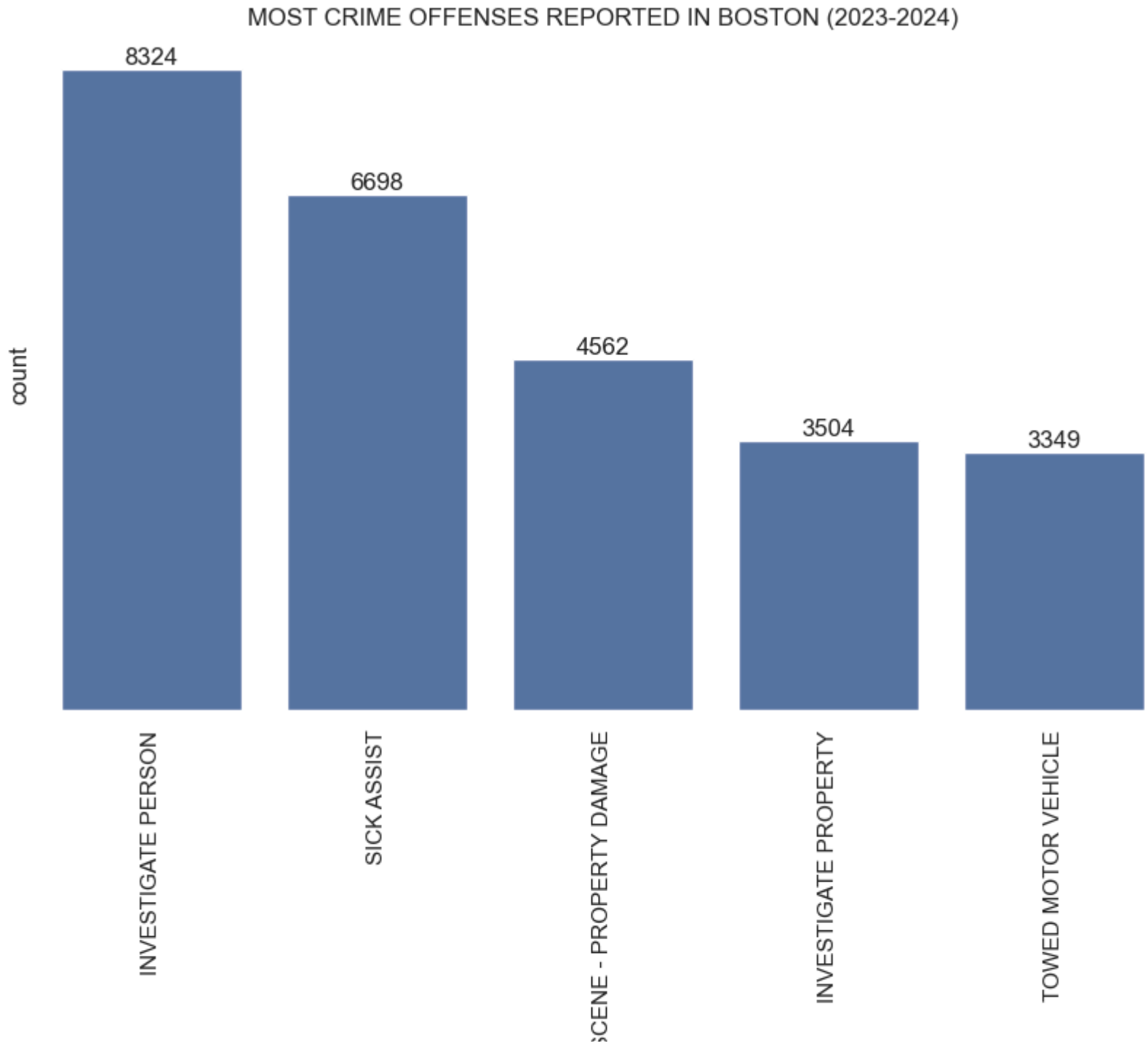
```
ax.set(xlabel='', yticks=[], title='DISTRICTS WITH THE MOST CRIME REPORTED CASES IN BOSTON (2023-2024)', frame_on=Fal
ax.bar_label(container=ax.containers[0])
plt.show()
```



- The top three crime prone districts in Boston are D4, B2 and C11 respectively. These districts account for almost half of crime reported cases in Boston.

In [38]:

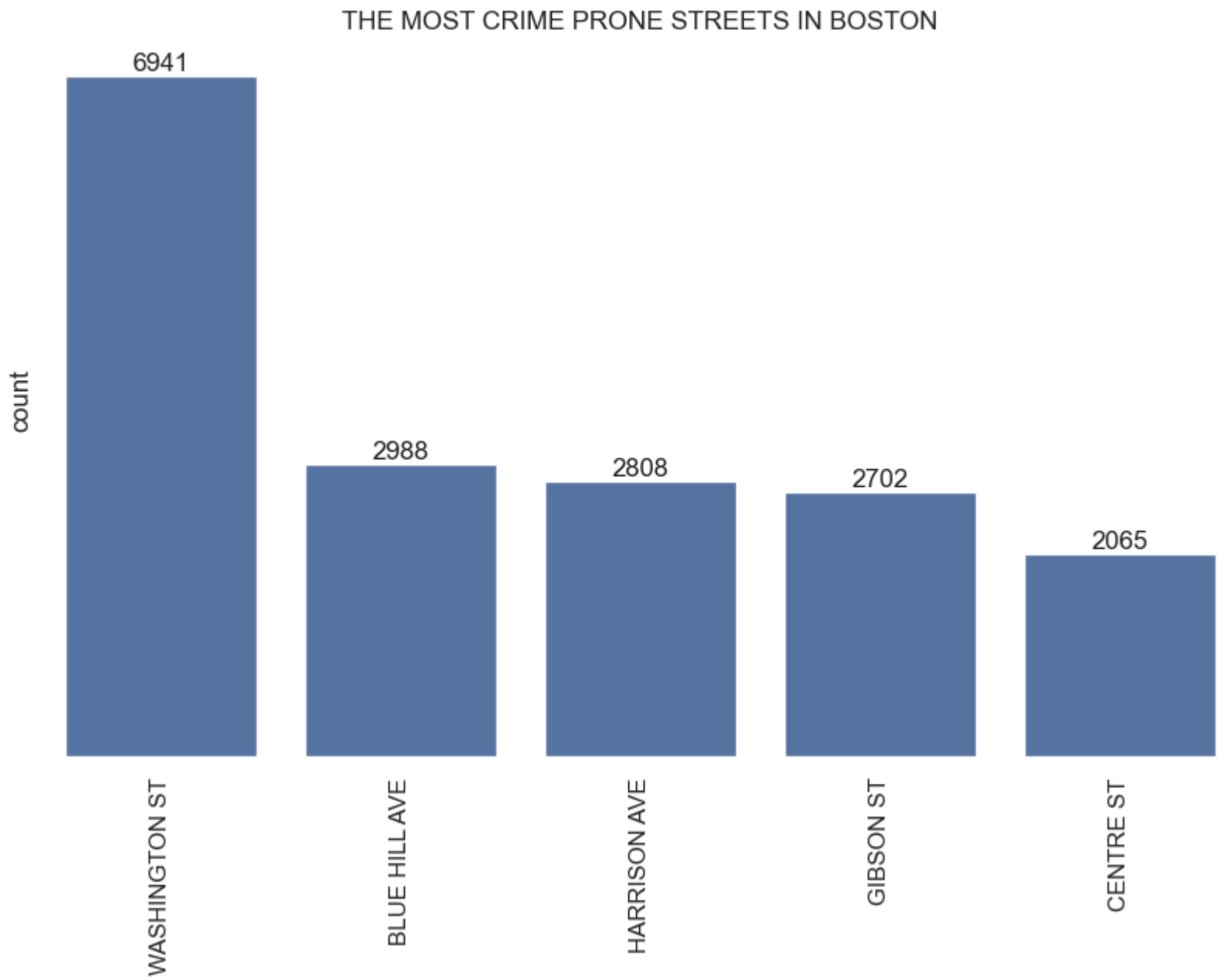
```
plt.figure(figsize=(10, 6))
sns.set(style='darkgrid')
ax = sns.countplot(x = 'OFFENSE_DESCRIPTION',
              data = crime_districts,
              order = crime_districts['OFFENSE_DESCRIPTION'].value_counts().nlargest().index)

ax.set(xlabel='', yticks=[], title='MOST CRIME OFFENSES REPORTED IN BOSTON (2023-2024)', frame_on=False) # prettify
ax.bar_label(container=ax.containers[0])
plt.xticks(rotation=90)
plt.show()
```


MOST CRIME OFFENSES REPORTED IN BOSTON (2023-2024)
count
8324
6698
4562
3504
3349
INVESTIGATE PERSON
SICK ASSIST
SCENE - PROPERTY DAMAGE
INVESTIGATE PROPERTY
TOWED MOTOR VEHICLE

M/V - LEAVING S

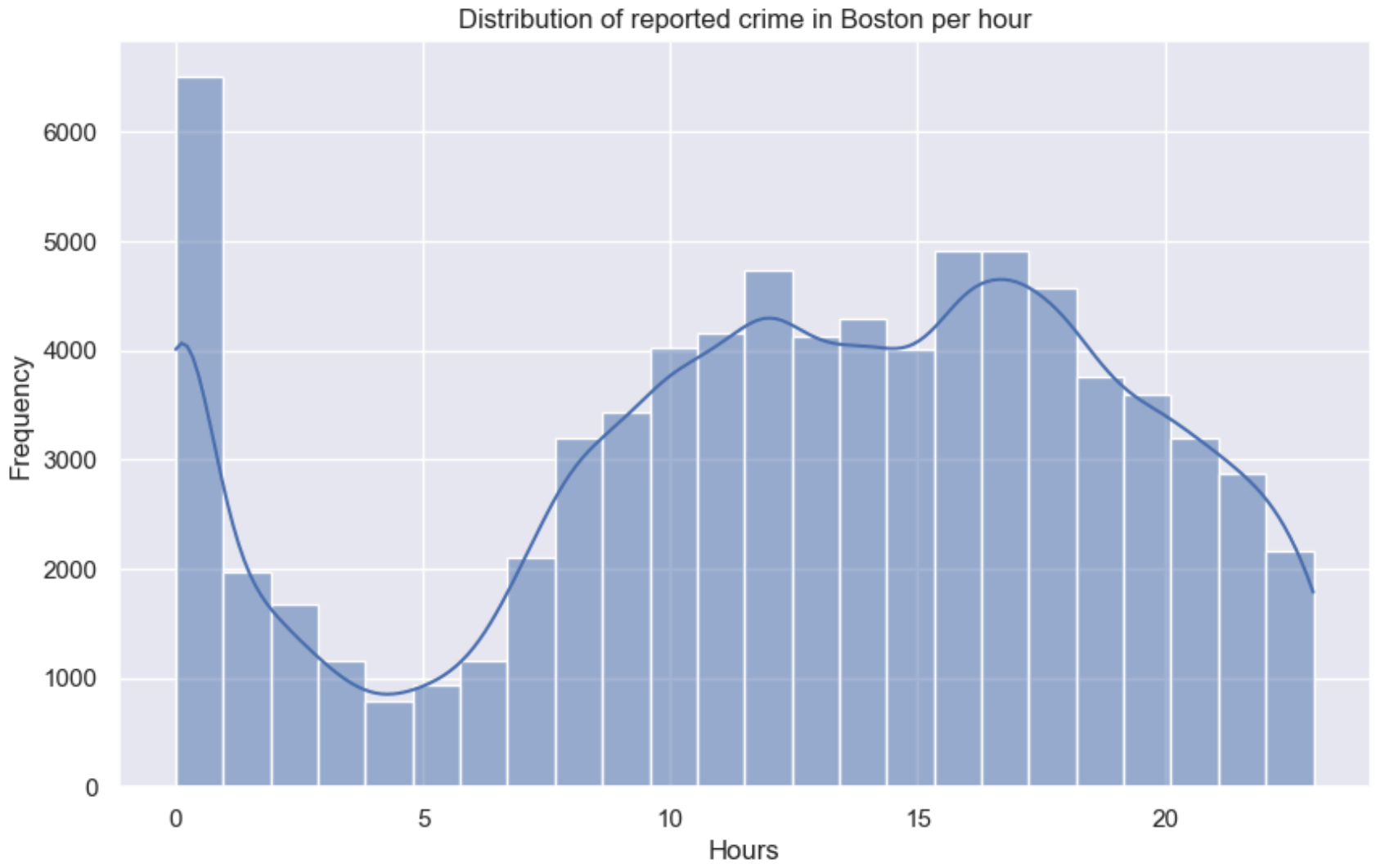
- • Investigating persons, sick assistant crime and property damage respectively are the top crime offenses in Boston.

In [39]:
```
plt.figure(figsize=(10, 6))
sns.set(style='darkgrid')
ax = sns.countplot(x = 'STREET',
              data = crime_districts,
              order = crime_districts['STREET'].value_counts().nlargest().index)

ax.set(xlabel='', yticks=[], title='THE MOST CRIME PRONE STREETS IN BOSTON', frame_on=False) # prettify
ax.bar_label(container=ax.containers[0])
plt.xticks(rotation=90)
plt.show()
```



- Street crimes are inevitable in most developed cities since such areas attract all sort of people. In Boston the most dangerous street is Washington Street. Crime reported on this street is above State average.

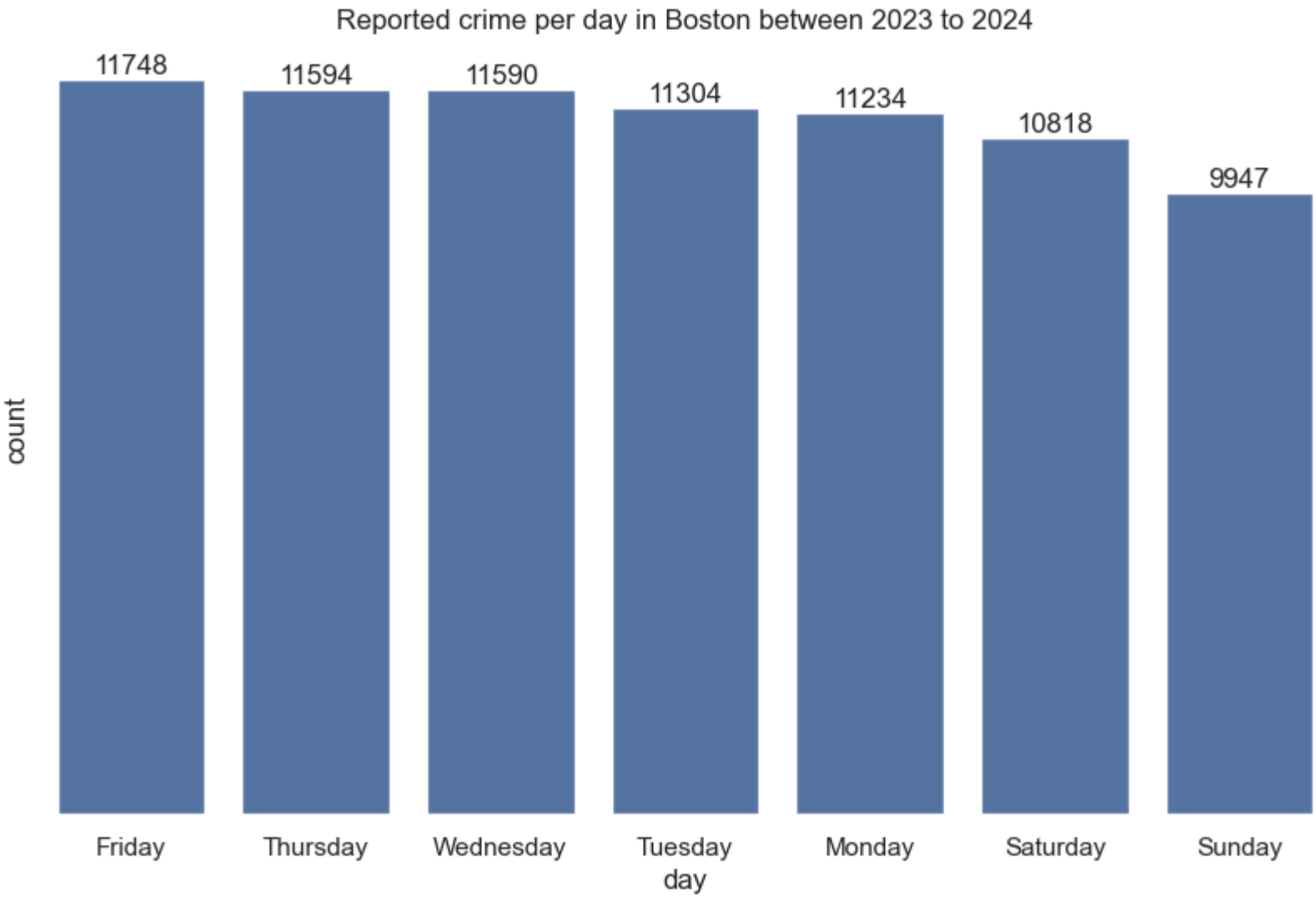
In [40]:
```
plt.figure(figsize=(10, 6))
# Create a Seaborn histogram with KDE
sns.histplot(data= crime_districts,
             x = "HOUR",
             #hue = "DISTRICT",
             bins=24,
             kde=True)

# Set labels and title
plt.xlabel('Hours')
plt.ylabel('Frequency')
plt.title('Distribution of reported crime in Boston per hour')
# Show the plot
plt.show()
```



- Most reported crime in Boston peak around dawn which is around 12am and decline drastically around 4am and peak again around 4pm to 5pm.

In [41]:
```
plt.figure(figsize=(10, 6))
sns.set(style='darkgrid')
ax = sns.countplot(x = 'DAY_OF_WEEK',
                   data = crime_districts,
```

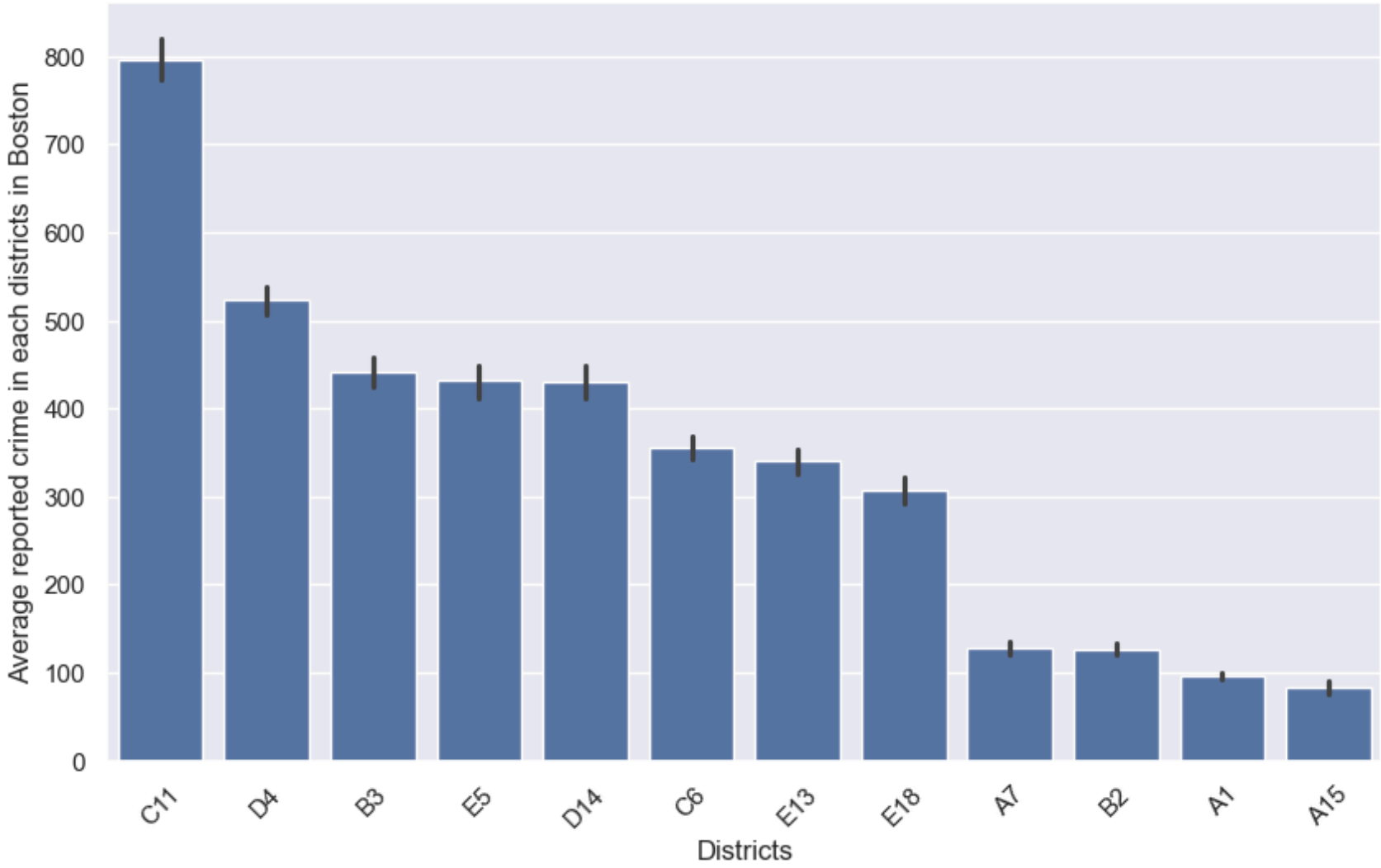
```
            order = crime_districts['DAY_OF_WEEK'].value_counts().index)
ax.set(xlabel='day', yticks=[], title='Reported crime per day in Boston between 2023 to 2024', frame_on=False) # pret
ax.bar_label(ax.containers[0]);
plt.show()
```



- Fridays and Thursdays seems the operations days for most crime offenders in Boston.

In [42]:

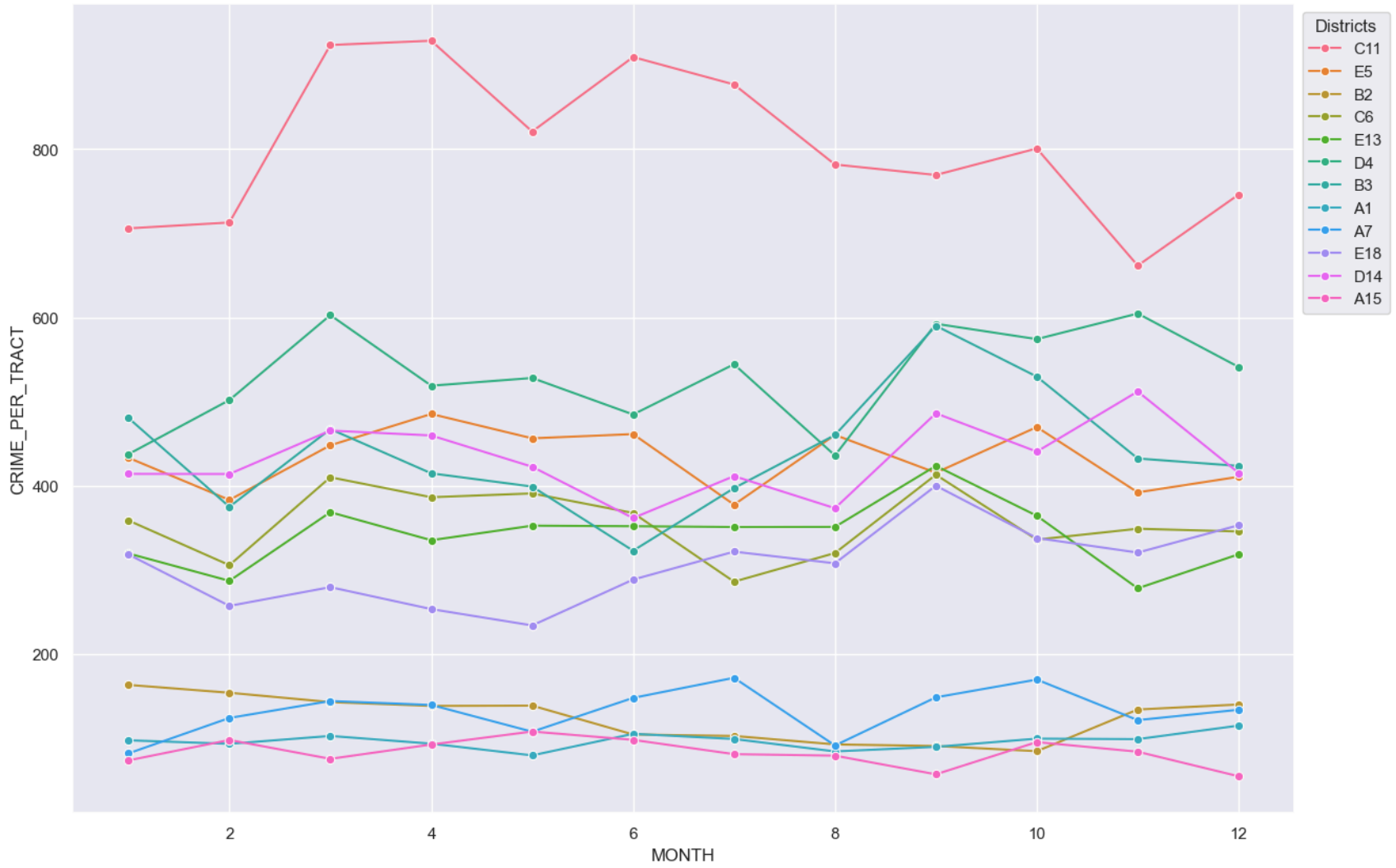
```
plt.figure(figsize=(10, 6))
# Create a Seaborn bar plot with hue
ax =sns.barplot(x="DISTRICT",
            y="CRIME_PER_TRACT",
            data=crime_districts,
            order=crime_districts.groupby("DISTRICT")["CRIME_PER_TRACT"].mean().sort_values(ascending=False).index)

# Set labels and title
plt.xlabel('Districts')
plt.ylabel('Average reported crime in each districts in Boston')
plt.title('')

# Show the plot
plt.xticks(rotation=45)
plt.show()
```



- D4 seems to be the district with the most occurrence of crime in Boston but C11 leads when it comes to numerical occurrences of reported crime in Boston.

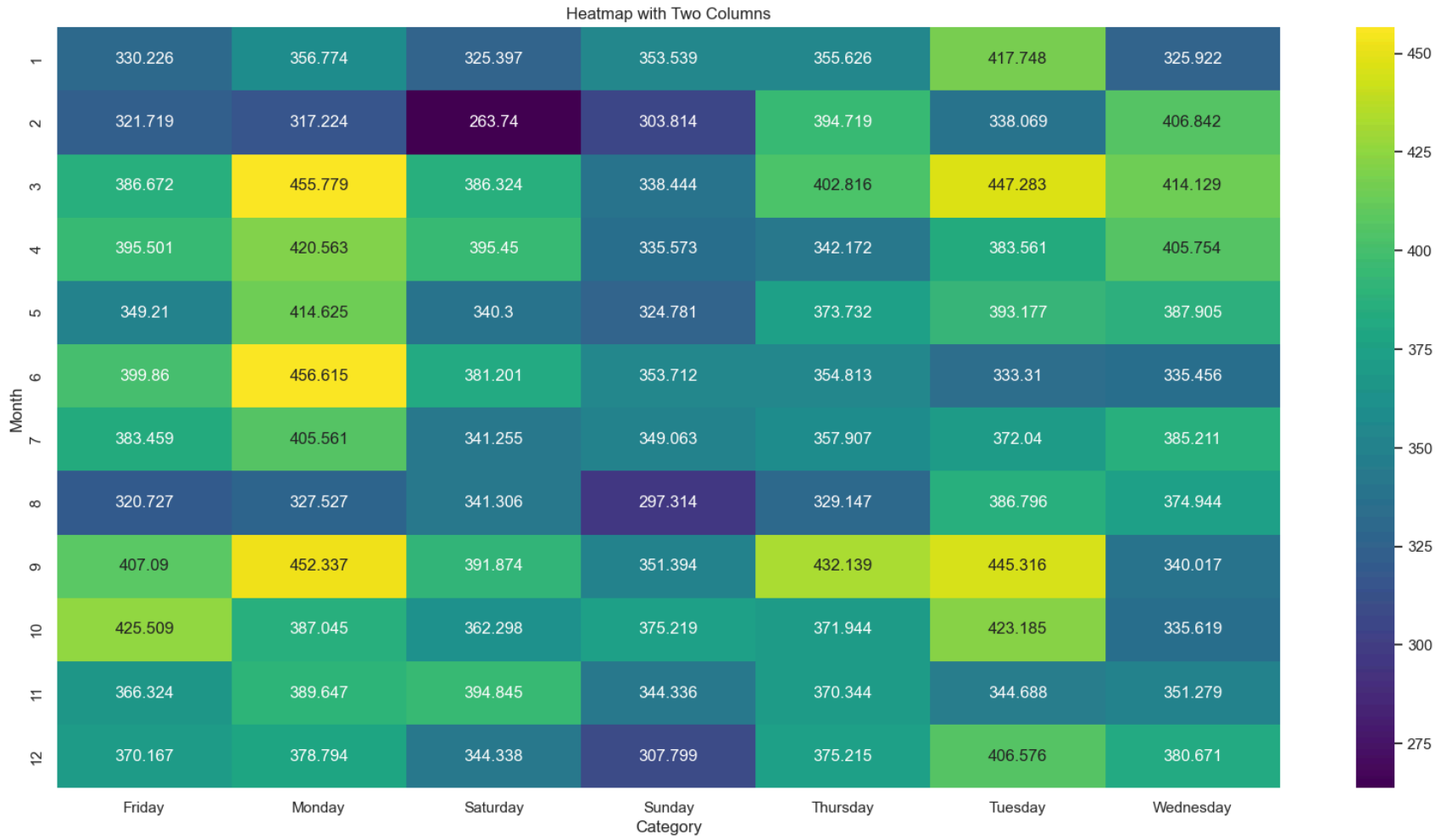
In [43]:
```
#Plot a lineplot with hue and adjust legend
plt.figure(figsize=(15, 10))
sns.lineplot(x='MONTH',
             y='CRIME_PER_TRACT',
```

```
             hue='DISTRICT', data=crime_districts,
             marker='o',
             ci =None
            )

# Adjust the legend
plt.legend(title='Districts', bbox_to_anchor=(1, 1), loc='upper left')

# Show the plot
plt.show()
```



In [44]:
```
# Pivot the DataFrame to create a matrix suitable for heatmap
heatmap_data = crime_districts.pivot_table(index='MONTH', columns='DAY_OF_WEEK', values='CRIME_PER_TRACT', aggfunc='m

# Plot heatmap using Seaborn
plt.figure(figsize=(20, 10))
sns.heatmap(heatmap_data, annot=True, cmap='viridis', fmt='g', cbar=True)

# Customize the plot
plt.title('Heatmap with Two Columns')
```

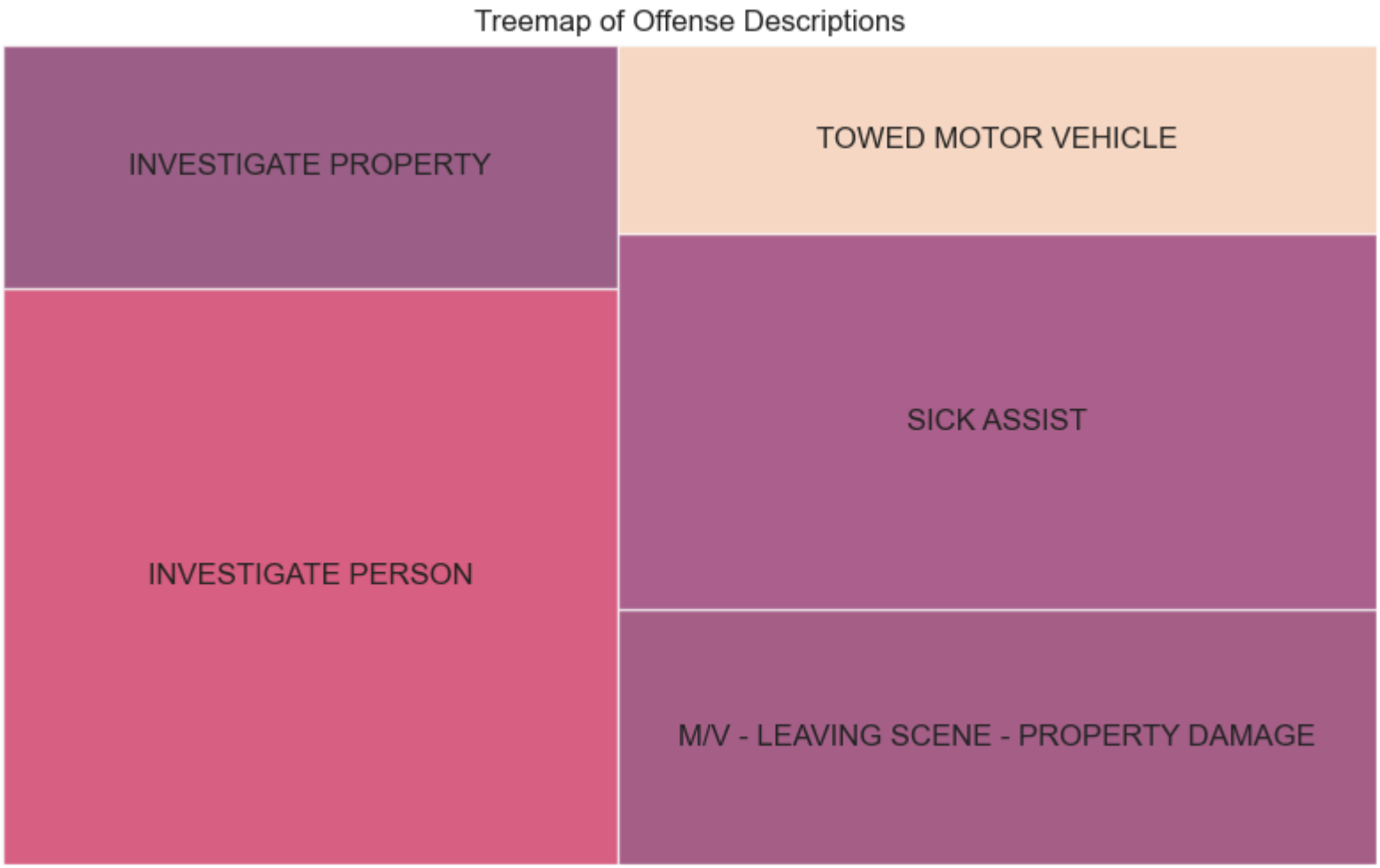
```
plt.xlabel('Category')
plt.ylabel('Month')

# Show the plot
plt.show()
```



In [ ]:

In [ ]:

In [45]:
```
# Drop NaN values from the OFFENSE_DESCRIPTION column
tmp = new_df.dropna(subset=['OFFENSE_DESCRIPTION'])

# Count the occurrences of each unique offense description
offense_counts = tmp['OFFENSE_DESCRIPTION'].value_counts().nlargest()

# Sort the unique offense descriptions
sorted_offense_counts = offense_counts.sort_index()

# Create a treemap
plt.figure(figsize=(10, 6))
squarify.plot(sizes=sorted_offense_counts.values, label=sorted_offense_counts.index, alpha=0.7)
plt.axis('off')
plt.title('Treemap of Offense Descriptions')
plt.show()
```



- **Geospatial Analysis and Visualization**

In [46]:
```
import folium
from folium.plugins import HeatMap
from matplotlib.colors import Normalize
from folium import Choropleth, Circle, Marker
from folium.plugins import HeatMap, MarkerCluster
```

In [47]:
```
# creating folium map of boston
f = folium.Figure(width=700, height=400)
```

```
m = folium.Map(location=[42.32, -71.0589], titles='openstreetmap', zoom_start=10, max_zoom=15, min_zoom=10).add_to(f)

# adding and displaying heat map of crimes with latitude/longitude
HeatMap(data=new_df[['Lat', 'Long']], radius=10).add_to(m)
m
```

Out[47]:

- From the heatmap above we can testifies that most reported crime in Boston are around the north and south eastern part of Boston.

In [48]:
```
crime_districts.reset_index(inplace = True)
```

In [49]:
```
# convert pandas dataframe to geodataframe
geo_data_1 = gpd.GeoDataFrame(crime_districts,
                          crs='EPSG:4326',
                          geometry=gpd.points_from_xy(crime_districts.Long, crime_districts.Lat))
```

In [50]:
```
geo_data_1.head()
```

Out[50]:

	index	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_DESCRIPTION	DISTRICT	REPORTING_AREA	SHOOTING	OCCURRED_ON
0	0	232099233	3115	INVESTIGATE PERSON	C11	355	0	2023 00:00:00
1	1	242008994	1102	FRAUD - FALSE PRETENSE / SCHEME	E5	691	0	2023 00:00:00
2	2	232079828	2914	VAL - OPERATING W/O AUTHORIZATION LAWFUL	B2	321	0	2023 00:00:00
3	3	232083964	1102	FRAUD - FALSE PRETENSE / SCHEME	C6	200	0	2023 00:00:00
4	4	232022077	1001	FORGERY / COUNTERFEITING	E5	680	0	2023 00:00:00

In [51]:
```
city_2 = gpd.read_file("2020_Census_Tracts_in_Boston.geojson")
```

In [52]:
```
city_2.head()
```

Out[52]:

	geoid20	countyfp20	namelsad20	statefp20	tractce20	intptlat20	name20	funcstat20	intptlon20	mtfcc20	alanc
0	25025140202	025	Census Tract	25	140202	+42.2495181	1402.02	S	-071.1175430	G5020	1538
1	25025140300	025	Census Tract	25	140300	+42.2587734	1403	S	-071.1188131	G5020	1548
2	25025140400	025	Census Tract	25	140400	+42.2692219	1404	S	-071.1118088	G5020	1874
3	25025140106	025	Census Tract	25	140106	+42.2738738	1401.06	S	-071.1371416	G5020	278
4	25025110201	025	Census Tract	25	110201	+42.2804960	1102.01	S	-071.1170508	G5020	348

In [53]:
```
new_geo_1 = gpd.sjoin(city_2, geo_data_1, how='left')
new_geo_1.head()
```

Out[53]:

	geoid20	countyfp20	namelsad20	statefp20	tractce20	intptlat20	name20	funcstat20	intptlon20	mtfcc20	...	Y
0	25025140202	025	Census Tract	25	140202	+42.2495181	1402.02	S	-071.1175430	G5020	...	2
0	25025140202	025	Census Tract	25	140202	+42.2495181	1402.02	S	-071.1175430	G5020	...	2
0	25025140202	025	Census Tract	25	140202	+42.2495181	1402.02	S	-071.1175430	G5020	...	2
0	25025140202	025	Census Tract	25	140202	+42.2495181	1402.02	S	-071.1175430	G5020	...	2
0	25025140202	025	Census Tract	25	140202	+42.2495181	1402.02	S	-071.1175430	G5020	...	2

5 rows × 33 columns

In [ ]:

In [55]:

```
new_geo_1.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
Index: 78191 entries, 0 to 206
Data columns (total 33 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   geoid20              78191 non-null  object
 1   countyfp20           78191 non-null  object
 2   namelsad20           78191 non-null  object
 3   statefp20            78191 non-null  object
 4   tractce20            78191 non-null  object
 5   intptlat20           78191 non-null  object
 6   name20               78191 non-null  object
 7   funcstat20           78191 non-null  object
 8   intptlon20           78191 non-null  object
 9   mtfcc20              78191 non-null  object
 10  aland20              78191 non-null  int64
 11  awater20             78191 non-null  int64
 12  objectid             78191 non-null  int64
 13  geometry             78191 non-null  geometry
 14  index_right          78191 non-null  int64
 15  index                78191 non-null  int64
 16  INCIDENT_NUMBER      78191 non-null  object
 17  OFFENSE_CODE         78191 non-null  int64
 18  OFFENSE_DESCRIPTION  78191 non-null  object
 19  DISTRICT             78191 non-null  object
 20  REPORTING_AREA       78191 non-null  object
 21  SHOOTING             78191 non-null  int64
 22  OCCURRED_ON_DATE     78191 non-null  datetime64[ns, UTC]
 23  YEAR                 78191 non-null  int64
 24  MONTH                78191 non-null  int64
 25  DAY_OF_WEEK          78191 non-null  object
 26  HOUR                 78191 non-null  int64
 27  STREET               78191 non-null  object
 28  Lat                  78191 non-null  float64
 29  Long                 78191 non-null  float64
 30  Location             78191 non-null  object
 31  CRIME_PER_TRACT      78191 non-null  int64
 32  Date                 78191 non-null  object
dtypes: datetime64[ns, UTC](1), float64(2), geometry(1), int64(11), object(18)
memory usage: 20.3+ MB
```

In [56]:
```
new_geo_1.isna().sum().sum()
```

Out[56]: 0

In [57]:
```
new_geo_1["DISTRICT"].value_counts()
```

Out[57]:
```
DISTRICT
D4     10898
B2     10558
C11     9314
A1      8569
B3      7942
C6      6612
D14     5871
E13     4870
E18     4298
A7      4049
E5      3822
A15     1388
Name: count, dtype: int64
```

In [58]:
```
names_to_filter = ['D4', 'B2', 'C11']
# Filter the DataFrame
filtered_df = new_geo_1[new_geo_1['DISTRICT'].isin(names_to_filter)]
```

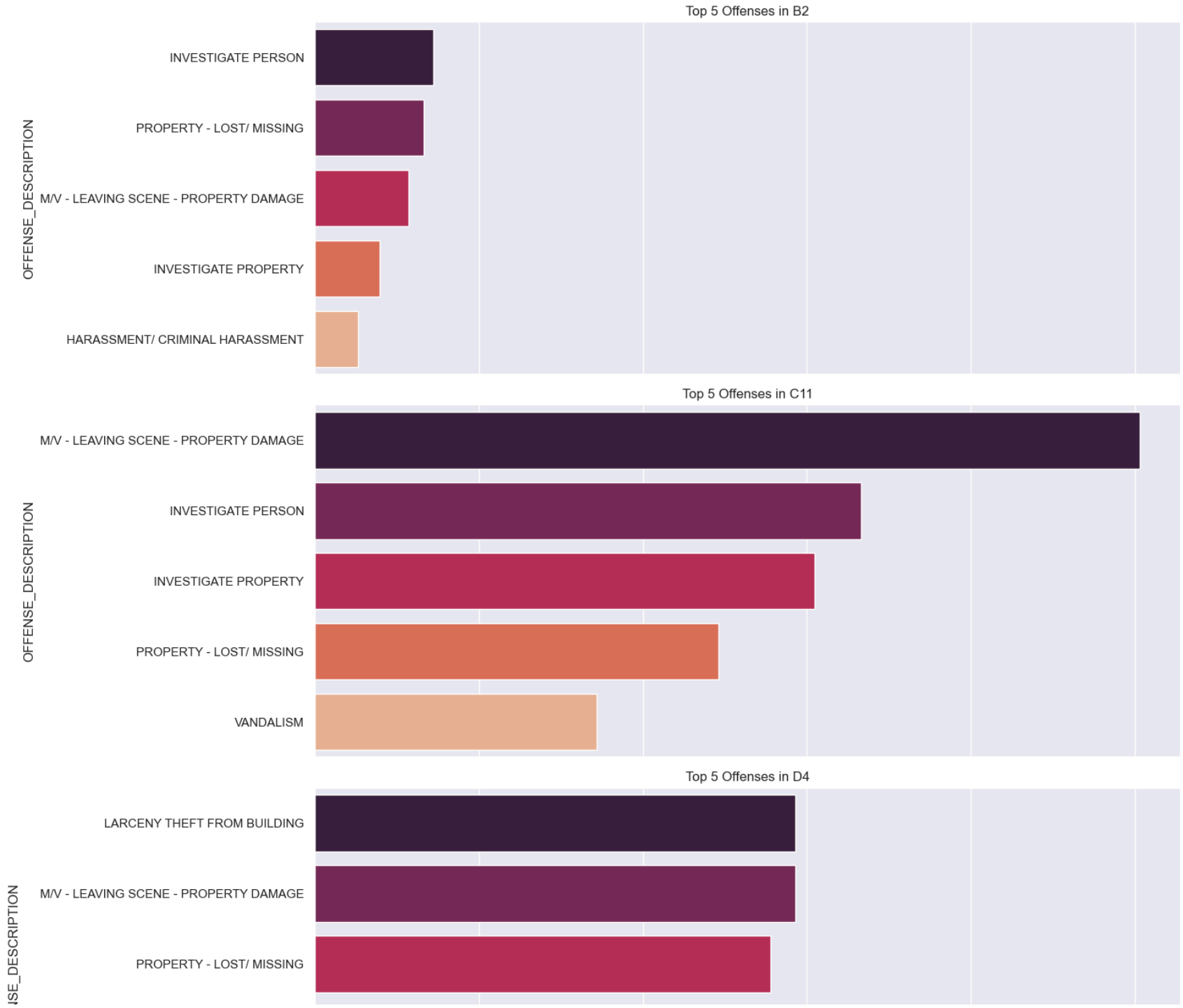
In [59]:
```
group_2 = filtered_df.groupby(["DISTRICT", "OFFENSE_DESCRIPTION"])["CRIME_PER_TRACT"].sum()
# Convert the result to a DataFrame and reset the index
group_2_df = group_2.reset_index()
group_2_df.tail()
```

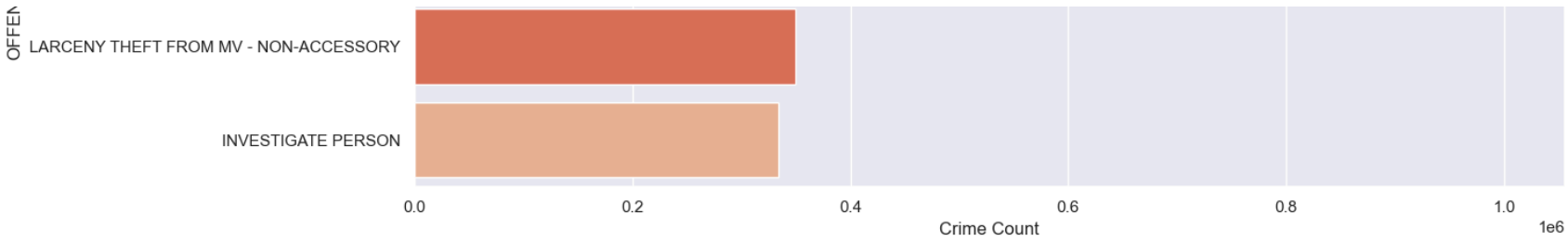
Out[59]:

	DISTRICT	OFFENSE_DESCRIPTION	CRIME_PER_TRACT
304	D4	VERBAL DISPUTE	22785
305	D4	VIOLATION - CITY ORDINANCE	51
306	D4	WARRANT ARREST - BOSTON WARRANT (MUST BE SUPPL...	329
307	D4	WARRANT ARREST - OUTSIDE OF BOSTON WARRANT	6752
308	D4	WEAPON VIOLATION - CARRY/ POSSESSING/ SALE/ TR...	1083

In [60]:

```
# Get unique districts
districts = group_2_df['DISTRICT'].unique()

# Create subplots with three rows
fig, axs = plt.subplots(3, figsize=(15, 15), sharex=True)

# Loop through each district
for i, district in enumerate(districts):
    # Get data for the current district
    district_data = group_2_df[group_2_df['DISTRICT'].isin([district])]
    # Sort by crime count and select top 5 offenses
    top_offenses = district_data.sort_values(by='CRIME_PER_TRACT', ascending=False).head(5)
    # Plot bar chart for the current district
    sns.barplot(x='CRIME_PER_TRACT', y='OFFENSE_DESCRIPTION',palette="rocket", data=top_offenses, ax=axs[i])
    axs[i].set_title(f'Top 5 Offenses in {district}')
    axs[i].set_xlabel('Crime Count')

# Adjust layout
plt.tight_layout()
plt.show()
```

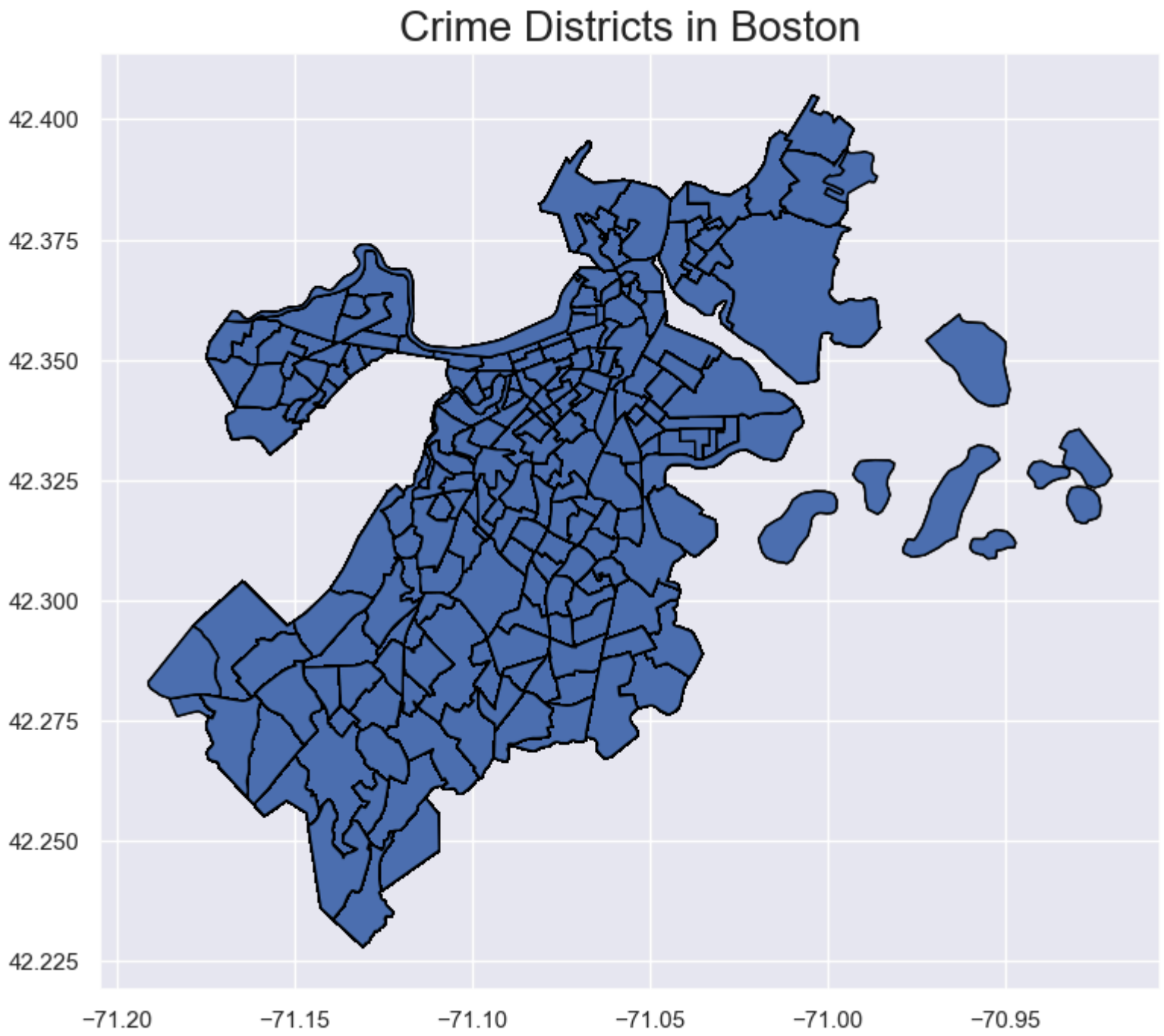

Top 5 Offenses in B2
OFFENSE_DESCRIPTION
INVESTIGATE PERSON
PROPERTY - LOST/ MISSING
M/V - LEAVING SCENE - PROPERTY DAMAGE
INVESTIGATE PROPERTY
HARASSMENT/ CRIMINAL HARASSMENT
Top 5 Offenses in C11
OFFENSE_DESCRIPTION
M/V - LEAVING SCENE - PROPERTY DAMAGE
INVESTIGATE PERSON
INVESTIGATE PROPERTY
PROPERTY - LOST/ MISSING
VANDALISM
Top 5 Offenses in D4
ISE_DESCRIPTION
LARCENY THEFT FROM BUILDING
M/V - LEAVING SCENE - PROPERTY DAMAGE
PROPERTY - LOST/ MISSING



- The graph above shows most dominating crime offenses in the top 3 Districts in Boston. Property damage is the leading crime offense in district C11, Theft from building is the topmost offense in D4 and Investigated person in district B2.

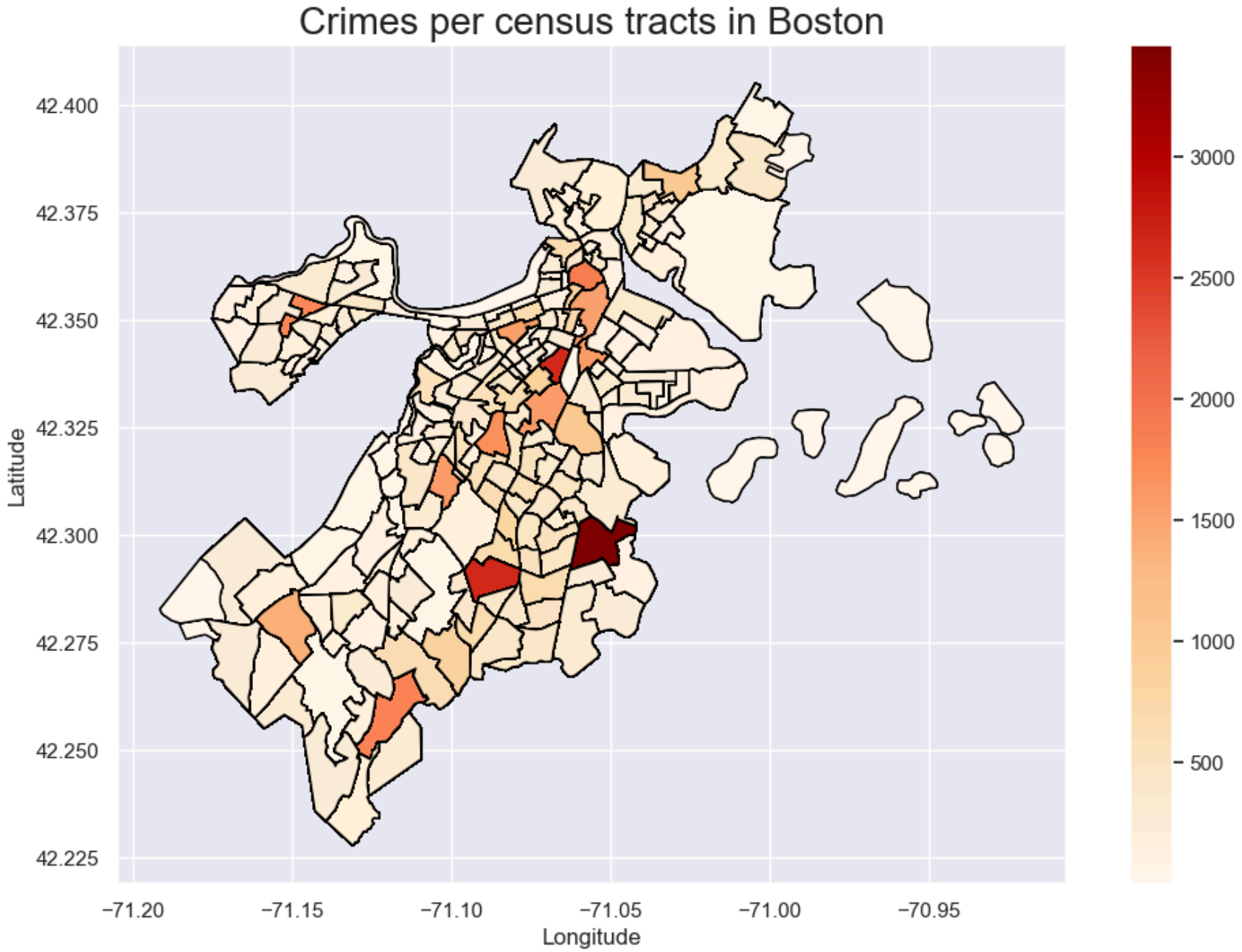
In [61]:
```
# label reference points on map
new_geo_1['point'] = new_geo_1.representative_point()
labels = new_geo_1.copy()
labels.set_geometry('point', inplace = True)

# plot districts
ax = new_geo_1.plot(figsize = (12,8), edgecolor = 'black')

# put labels in districts
#for x, y, label in zip(labels.geometry.x, labels.geometry.y, labels['DISTRICT']):
    #plt.text(x, y, label, fontsize = 10, fontweight = 'bold')


plt.title('Crime Districts in Boston', fontsize = 20)
```

Out[61]:
```
Text(0.5, 1.0, 'Crime Districts in Boston')
```


Crime Districts in Boston
42.400
42.375
42.350
42.325
42.300
42.275
42.250
42.225
−71.20
−71.15
−71.10
−71.05
−71.00
−70.95

In [62]:
```
crime_districts = new_geo_1['geoid20'].value_counts()
# mapping and plotting district crime data
new_geo_1['crimes'] = new_geo_1.geoid20.map(crime_districts)
ax = new_geo_1.plot(column = new_geo_1.crimes, cmap = 'OrRd', legend = True, edgecolor = 'black', figsize = (12,8))

# put labels in districts
#for x, y, label in zip(labels.geometry.x, labels.geometry.y, labels['DISTRICT_left']):
    #plt.text(x, y, label, fontsize = 10, fontweight = 'bold', c = 'white')


plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Crimes per census tracts in Boston', fontsize = 20)
```

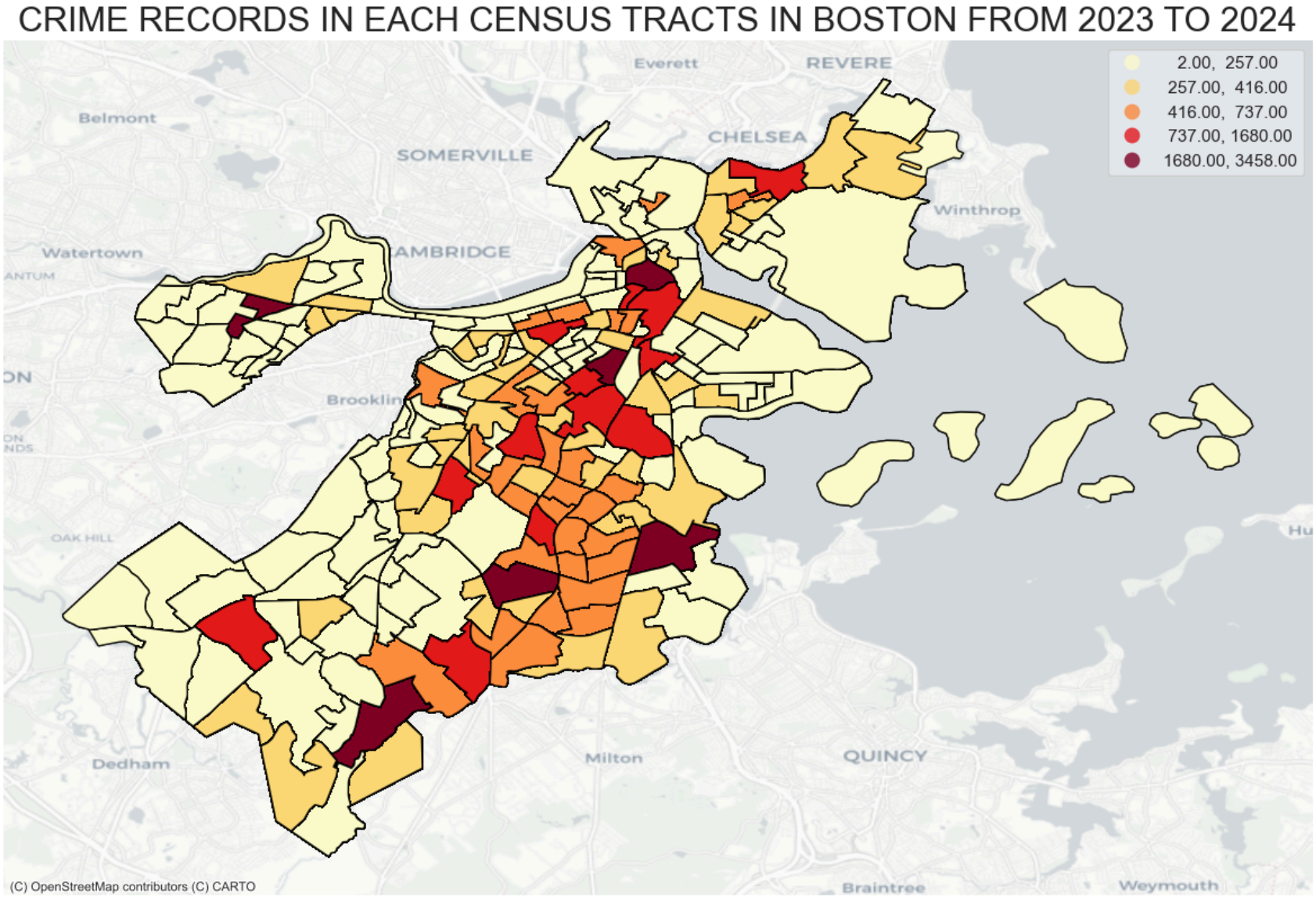
Out[62]:
```
Text(0.5, 1.0, 'Crimes per census tracts in Boston')
```



- The above map shows total number of reported crime per each census tracts in Boston.

In [63]:

```
fig,ax = plt.subplots(figsize=(15,15))

new_geo_1.plot(ax=ax,
        column='crimes', # this makes it a choropleth
        legend=True,
        alpha=0.8,
        edgecolor = 'black',
        cmap='YlOrRd', # a diverging color scheme
        scheme='quantiles') # how to break the data into bins

ax.axis('off')
ax.set_title('CRIME RECORDS IN EACH CENSUS TRACTS IN BOSTON FROM 2023 TO 2024',fontsize=22)
# Add basemap
ctx.add_basemap(
    ax,
    crs=new_geo_1.crs,
    source=ctx.providers.CartoDB.Positron,)
```



In [64]:
```
# temporary dataset with no na offense group values
tmp = new_geo_1.dropna(subset=['OFFENSE_DESCRIPTION'])

# count most offense group values and display the head
#print(tmp['OFFENSE_DESCRIPTION'].value_counts().head())
```

```python
# dataset for the 5 crimes which occur the most in the dataset
top_5_crime = tmp[(tmp['OFFENSE_DESCRIPTION'] == 'INVESTIGATE PERSON')
                        | (tmp['OFFENSE_DESCRIPTION'] == 'SICK ASSIST')
                        | (tmp['OFFENSE_DESCRIPTION'] == 'M/V - LEAVING SCENE - PROPERTY DAMAGE')
                        | (tmp['OFFENSE_DESCRIPTION'] == 'INVESTIGATE PROPERTY')
                        | (tmp['OFFENSE_DESCRIPTION'] == 'TOWED MOTOR VEHICLE')]
```

In [65]:
```python
top_5_crime.head()
```

Out[65]:

	geoid20	countyfp20	namelsad20	statefp20	tractce20	intptlat20	name20	funcstat20	intptlon20	mtfcc20	...	D
0	25025140202	025	Census Tract	25	140202	+42.2495181	1402.02	S	-071.1175430	G5020	...	
0	25025140202	025	Census Tract	25	140202	+42.2495181	1402.02	S	-071.1175430	G5020	...	
0	25025140202	025	Census Tract	25	140202	+42.2495181	1402.02	S	-071.1175430	G5020	...	
0	25025140202	025	Census Tract	25	140202	+42.2495181	1402.02	S	-071.1175430	G5020	...	
0	25025140202	025	Census Tract	25	140202	+42.2495181	1402.02	S	-071.1175430	G5020	...	

5 rows × 35 columns

In [66]:
```python
top_5_crime.columns
```

Out[66]:
```
Index(['geoid20', 'countyfp20', 'namelsad20', 'statefp20', 'tractce20',
       'intptlat20', 'name20', 'funcstat20', 'intptlon20', 'mtfcc20',
       'aland20', 'awater20', 'objectid', 'geometry', 'index_right', 'index',
       'INCIDENT_NUMBER', 'OFFENSE_CODE', 'OFFENSE_DESCRIPTION', 'DISTRICT',
       'REPORTING_AREA', 'SHOOTING', 'OCCURRED_ON_DATE', 'YEAR', 'MONTH',
       'DAY_OF_WEEK', 'HOUR', 'STREET', 'Lat', 'Long', 'Location',
       'CRIME_PER_TRACT', 'Date', 'point', 'crimes'],
      dtype='object')
```

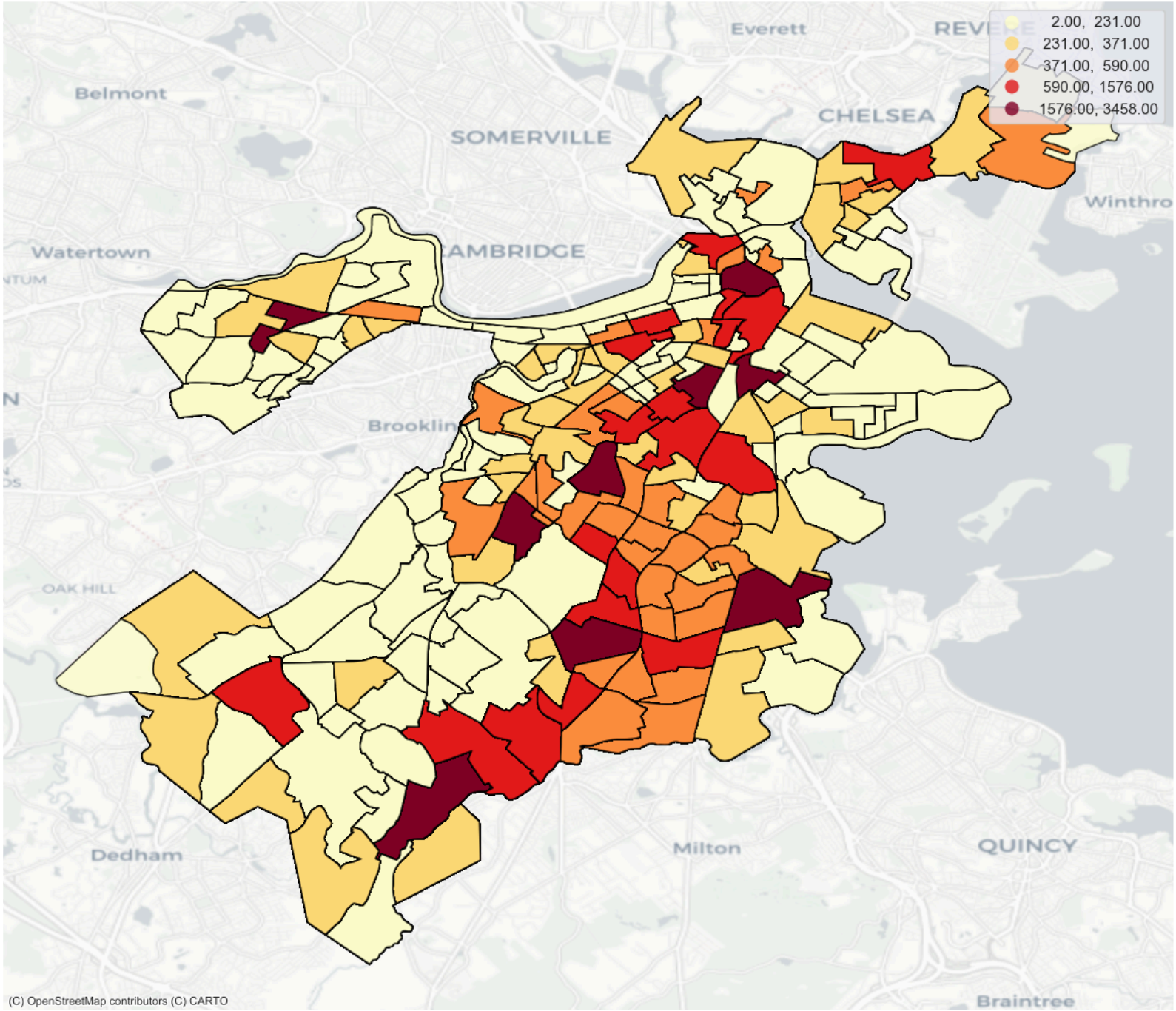
In [67]:
```
drop_column = ['namelsad20', 'statefp20', 'tractce20',
       'intptlat20', 'name20', 'funcstat20', 'intptlon20', 'mtfcc20',
       'aland20', 'awater20','index_right', 'index',
       'INCIDENT_NUMBER','CRIME_PER_TRACT']
```

In [68]:
```
gdf_2 = top_5_crime.drop(drop_column, axis =1)
```

In [69]:
```
gdf_3 = gdf_2.reset_index().rename(columns={'index': 'index_new'})
```

In [70]:
```
fig,ax = plt.subplots(figsize=(15,15))

gdf_3.plot(ax=ax,
        column='crimes', # this makes it a choropleth
        legend=True,
        alpha=0.8,
        edgecolor = 'black',
        cmap='YlOrRd', # a diverging color scheme
        scheme='quantiles') # how to break the data into bins

ax.axis('off')
ax.set_title('TOP 5 CRIME RECORDS IN EACH CENSUS TRACTS IN BOSTON FROM 2023 TO 2024',fontsize=22)
# Add basemap
ctx.add_basemap(
    ax,
    crs=gdf_3.crs,
    source=ctx.providers.CartoDB.Positron,)
```

TOP 5 CRIME RECORDS IN EACH CENSUS TRACTS IN BOSTON FROM 2023 TO 2024



- **Spatial Clustering and Regionalization**

In [71]:
```
# calculate spatial weight
wq =  lps.weights.KNN.from_dataframe(gdf_3,k=10)

# Row-standardization
wq.transform = 'r'
```

In [ ]:
```

```

In [72]:
```
# create a new column for the spatial lag
gdf_3['CRIME_LAG'] = lps.weights.lag_spatial(wq, gdf_3['crimes'])
```

In [73]:
```
gdf_3.head()
```

Out[73]:

	index_new	geoid20	countyfp20	objectid	geometry	OFFENSE_CODE	OFFENSE_DESCRIPTION	DISTRICT	REPORTING_A
0	0	25025140202	025	1	POLYGON ((-71.12623 42.24268, -71.12624 42.242...	3115	INVESTIGATE PERSON	E18	
1	0	25025140202	025	1	POLYGON ((-71.12623 42.24268, -71.12624 42.242...	3115	INVESTIGATE PERSON	E18	
2	0	25025140202	025	1	POLYGON ((-71.12623 42.24268, -71.12624 42.242...	3114	INVESTIGATE PROPERTY	E18	
3	0	25025140202	025	1	POLYGON ((-71.12623 42.24268, -71.12624 42.242...	1831	SICK ASSIST	E18	
4	0	25025140202	025	1	POLYGON ((-71.12623 42.24268, -71.12624 42.242...	3115	INVESTIGATE PERSON	E18	

5 rows × 23 columns

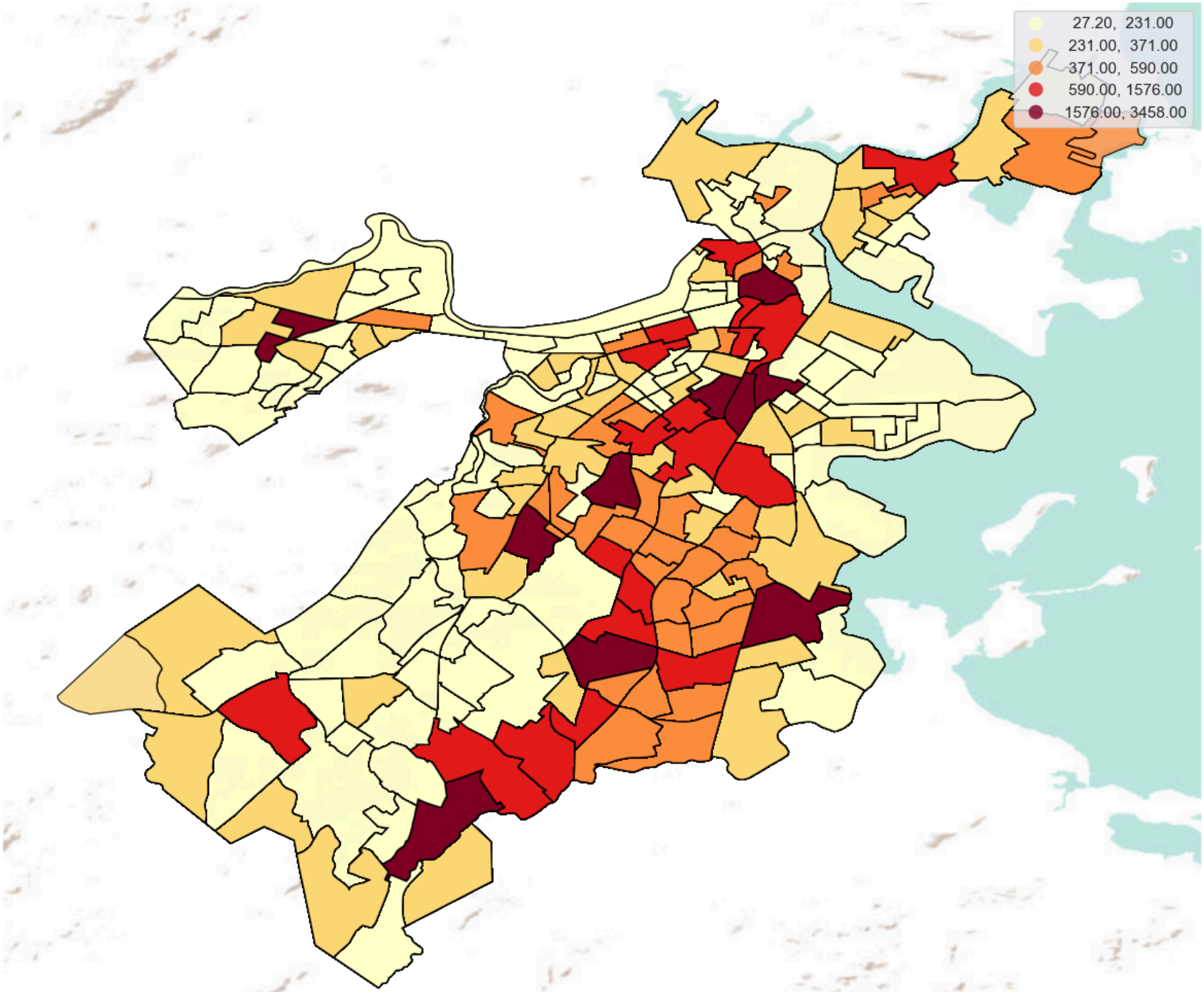
In [74]:
```
morans_stat = esda.moran.Moran(gdf_3['CRIME_LAG'], wq)
display(Markdown(f"""**Morans I:** {morans_stat.I}"""))
display(Markdown(f"""**p-value:** {morans_stat.p_sim}"""))
```

Morans I: 1.0000700351857004

p-value: 0.001

In [76]:
```
fig,ax = plt.subplots(figsize=(15,15))

gdf_3.plot(ax=ax,
        column='CRIME_LAG', # this makes it a choropleth
        legend=True,
        alpha=0.8,
        edgecolor = 'black',
        k = 5,
        cmap='YlOrRd', # a diverging color scheme
        scheme='quantiles') # how to break the data into bins

ax.axis('off')
ax.set_title('CRIME RECORDS IN EACH CENSUS TRACTS IN BOSTON FROM 2023 TO 2024',fontsize=22)
# Add basemap
ctx.add_basemap(
    ax,
    crs=gdf_3.crs,
    source=ctx.providers.Esri.WorldTerrain,)
```

CRIME RECORDS IN EACH CENSUS TRACTS IN BOSTON FROM 2023 TO 2024



- Using the spatial lag crime count, which average it surrounding to smoothen the map, we can observed that most reported crime cases in Boston are at the north and round down to south along the eastern part of Boston.

In [77]:
```
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
```

In [78]:
```
# Initialize the OneHotEncoder
encoder = OneHotEncoder(sparse=False)

# Fit and transform the data
encoded_data = encoder.fit_transform(gdf_3[['OFFENSE_DESCRIPTION']])

# Create a DataFrame with the encoded data
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(['OFFENSE_DESCRIPTION']))

# Concatenate the encoded DataFrame with the original data
final_df = pd.concat([gdf_3, encoded_df], axis=1)

final_df
```

Out[78]:

	index_new	geoid20	countyfp20	objectid	geometry	OFFENSE_CODE	OFFENSE_DESCRIPTION	DISTRICT	REPORTI
0	0	25025140202	025	1	POLYGON ((-71.12623 42.24268, -71.12624 42.242...	3115	INVESTIGATE PERSON	E18	
1	0	25025140202	025	1	POLYGON ((-71.12623 42.24268, -71.12624 42.242...	3115	INVESTIGATE PERSON	E18	
2	0	25025140202	025	1	POLYGON ((-71.12623 42.24268, -71.12624 42.242...	3114	INVESTIGATE PROPERTY	E18	
3	0	25025140202	025	1	POLYGON ((-71.12623 42.24268, -71.12624 42.242...	1831	SICK ASSIST	E18	
4	0	25025140202	025	1	POLYGON ((-71.12623 42.24268, -71.12624 42.242...	3115	INVESTIGATE PERSON	E18	
...	...	...	...	...	...	...	...	...	
26415	206	25025981501	025	207	POLYGON ((-71.16862 42.35896, -71.16947 42.358...	3115	INVESTIGATE PERSON	D14	

	index_new	geoid20	countyfp20	objectid	geometry	OFFENSE_CODE	OFFENSE_DESCRIPTION	DISTRICT	REPORTI
26416	206	25025981501	025	207	POLYGON ((-71.16862 42.35896, -71.16947 42.358...	3114	INVESTIGATE PROPERTY	D14	
26417	206	25025981501	025	207	POLYGON ((-71.16862 42.35896, -71.16947 42.358...	3410	TOWED MOTOR VEHICLE	D14	
26418	206	25025981501	025	207	POLYGON ((-71.16862 42.35896, -71.16947 42.358...	3114	INVESTIGATE PROPERTY	D14	
26419	206	25025981501	025	207	POLYGON ((-71.16862 42.35896, -71.16947 42.358...	3115	INVESTIGATE PERSON	D14	

26420 rows × 28 columns

In [79]:
```
gdf_3.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 26420 entries, 0 to 26419
Data columns (total 23 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   index_new            26420 non-null  int64
 1   geoid20              26420 non-null  object
 2   countyfp20           26420 non-null  object
 3   objectid             26420 non-null  int64
 4   geometry             26420 non-null  geometry
 5   OFFENSE_CODE         26420 non-null  int64
 6   OFFENSE_DESCRIPTION  26420 non-null  object
 7   DISTRICT             26420 non-null  object
 8   REPORTING_AREA       26420 non-null  object
 9   SHOOTING             26420 non-null  int64
 10  OCCURRED_ON_DATE     26420 non-null  datetime64[ns, UTC]
 11  YEAR                 26420 non-null  int64
 12  MONTH                26420 non-null  int64
 13  DAY_OF_WEEK          26420 non-null  object
 14  HOUR                 26420 non-null  int64
 15  STREET               26420 non-null  object
 16  Lat                  26420 non-null  float64
 17  Long                 26420 non-null  float64
 18  Location             26420 non-null  object
 19  Date                 26420 non-null  object
 20  point                26420 non-null  geometry
 21  crimes               26420 non-null  int64
 22  CRIME_LAG            26420 non-null  float64
dtypes: datetime64[ns, UTC](1), float64(3), geometry(2), int64(8), object(9)
memory usage: 4.6+ MB
```

In [80]:
```
cluster_variables = ["CRIME_LAG",
                     "OFFENSE_DESCRIPTION_INVESTIGATE PERSON",
                     "OFFENSE_DESCRIPTION_INVESTIGATE PROPERTY",
                     "OFFENSE_DESCRIPTION_M/V - LEAVING SCENE - PROPERTY DAMAGE",
                     "OFFENSE_DESCRIPTION_SICK ASSIST",
                     "OFFENSE_DESCRIPTION_TOWED MOTOR VEHICLE"]
```

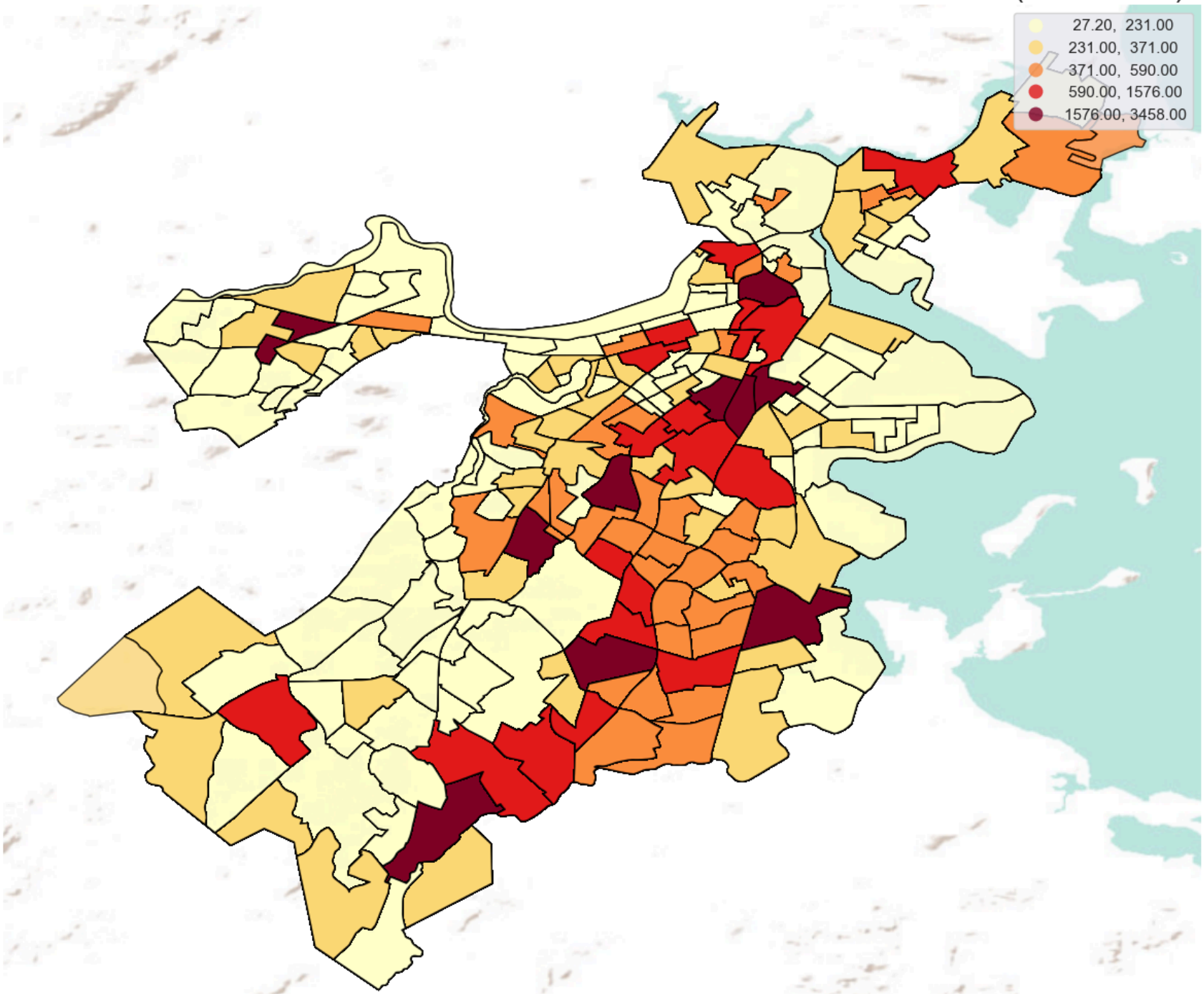
In [81]:
```
# List of new column names
new_column_names = {
     "OFFENSE_DESCRIPTION_INVESTIGATE PERSON":"INVESTIGATE PERSON",
                     "OFFENSE_DESCRIPTION_INVESTIGATE PROPERTY":"INVESTIGATE PROPERTY",
```

```
                    "OFFENSE_DESCRIPTION_M/V - LEAVING SCENE - PROPERTY DAMAGE":"PROPERTY DAMAGE",
                    "OFFENSE_DESCRIPTION_SICK ASSIST": "SICK ASSIST",
                    "OFFENSE_DESCRIPTION_TOWED MOTOR VEHICLE": "TOWED MOTOR VEHICLE"}

final_df.rename(columns=new_column_names, inplace=True)
```

In [82]:

```
fig,ax = plt.subplots(figsize=(15,15))

final_df.plot(ax=ax,
        column='CRIME_LAG', # this makes it a choropleth
        legend=True,
        alpha=0.8,
        edgecolor = 'black',
        k = 5,
        cmap='YlOrRd', # a diverging color scheme
        scheme='quantiles') # how to break the data into bins

ax.axis('off')
ax.set_title('CENSUS TRACTS WITH THE TOP 5 CRIME OFFENSES IN BOSTON (2023-2024) ',fontsize=22)
# Add basemap
ctx.add_basemap(
    ax,
    crs=final_df.crs,
    source=ctx.providers.Esri.WorldTerrain,)
```

CENSUS TRACTS WITH THE TOP 5 CRIME OFFENSES IN BOSTON (2023-2024)



- We proceed to analyzed the top 5 crime offenses in Boston. The map above shows the distibution of the top 5 crime offenses in the various census tracts.

In [83]:
```
cluster_variables = ["CRIME_LAG",
                     "INVESTIGATE PERSON",
                     "INVESTIGATE PROPERTY",
                     "PROPERTY DAMAGE",
                     "SICK ASSIST",
                     "TOWED MOTOR VEHICLE"]
```

In [84]:
```
# Set seed for reproducibility
np.random.seed(123456)
# Calculate Moran's I for each variable
mi_results = [
    Moran(final_df[variable], wq) for variable in cluster_variables
]
# Structure results as a list of tuples
mi_results = [
    (variable, res.I, res.p_sim)
    for variable, res in zip(cluster_variables, mi_results)
]
# Display on table
table = pd.DataFrame(
    mi_results, columns=["Variable", "Moran's I", "P-value"]
).set_index("Variable")
table
```

Out[84]:

Variable	Moran's I	P-value
CRIME_LAG	1.000070	0.001
INVESTIGATE PERSON	0.027744	0.001
INVESTIGATE PROPERTY	0.061826	0.001
PROPERTY DAMAGE	0.136809	0.001
SICK ASSIST	0.051797	0.001
TOWED MOTOR VEHICLE	0.109833	0.001

- We access the spatial structure of the top crime offenses in Boston, and we can testified with it p-values that each offense is statistically significant, which implies that none of the top 5 reported offenses occurred randomly.

In [85]:
```
from sklearn.preprocessing import RobustScaler
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import robust_scale
```

In [86]:
```
w = Queen.from_dataframe(final_df)
```

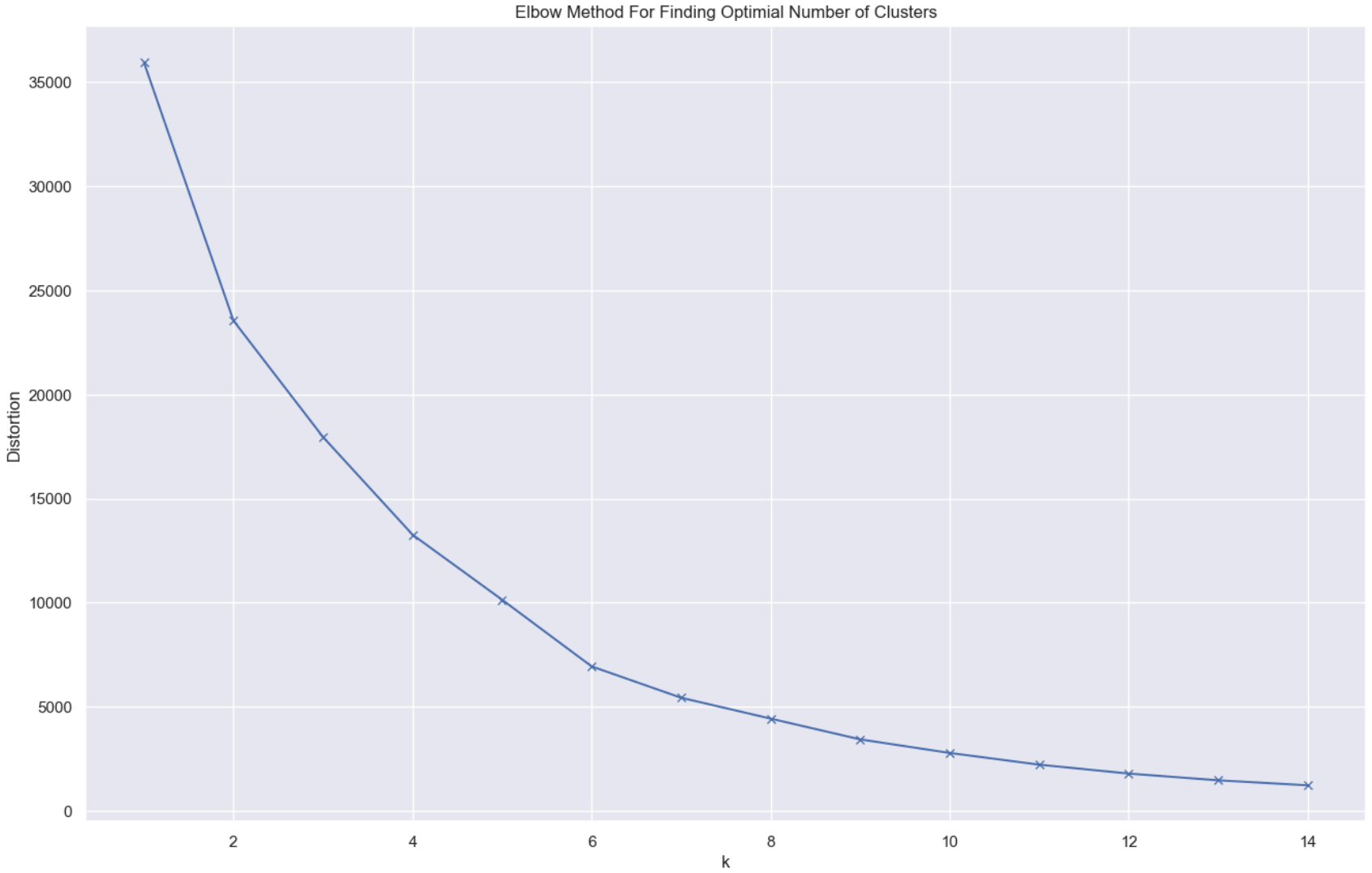
In [87]:
```
db_scaled = robust_scale(final_df[cluster_variables])
```

In [88]:
```
from sklearn.cluster import KMeans
```

In [89]:
```
np.random.seed(54321)
```

In [90]:
```
distortions = []
K = range(1,15)
for k in K:
    # Instantiating the model
    kmeans=KMeans(n_clusters=k)
    kmeans.fit(db_scaled)
    distortions.append(kmeans.inertia_)
```

```
# Setting the font size
plt.rcParams['font.size'] = '6'

plt.figure(figsize=(16,10))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('Elbow Method For Finding Optimial Number of Clusters')
plt.show()
```



- The optimal number of clusters in a k-means clustering algorithm is typically determined by identifying the “elbow point” in the plot of the within-cluster sum of squares (WCSS) against the number of clusters. The WCSS measures the total variance within each cluster. When the curve starts to level off, that point indicates a reasonable balance between minimizing the WCSS and avoiding overfitting.
- In my case, it appears that the elbow point occurs around six clusters. Therefore, we proceed to create a k-means model with this optimal number of clusters.

In [91]:
```
# Initialize KMeans instance
kmeans = KMeans(n_clusters=6)
```

In [92]:
```
# Set the seed for reproducibility
np.random.seed(1234)
# Run K-Means algorithm
k5cls = kmeans.fit(db_scaled)
```

In [93]:
```
# Print first five labels
k5cls.labels_[:5]
```

Out[93]:
```
array([2, 2, 3, 0, 2])
```

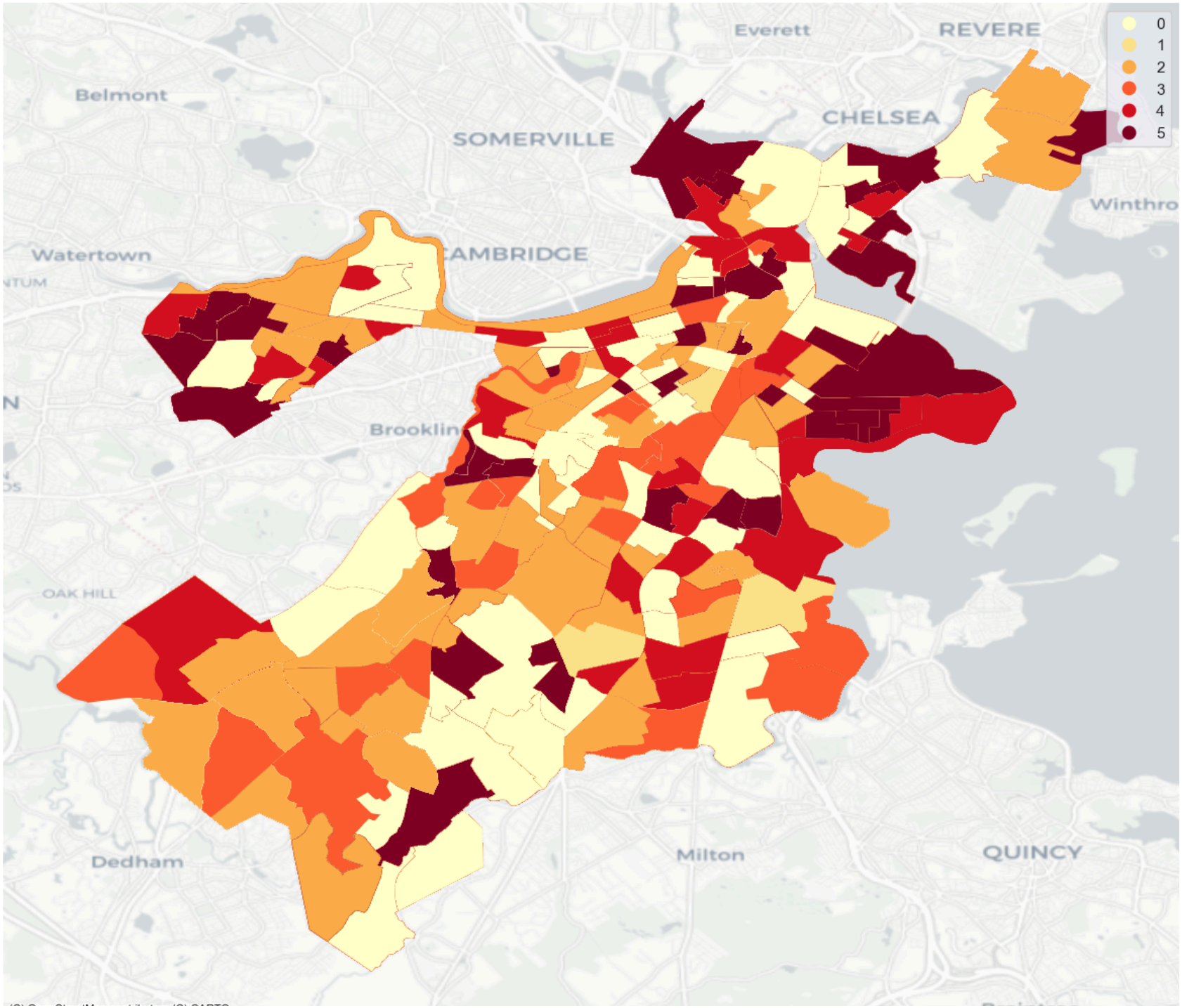
In [94]:
```
final_df["k5cls"] = k5cls.labels_
```

In [95]:
```
# Group data table by cluster label and count observations
k5sizes= final_df.groupby("k5cls").size()
k5sizes
```

Out[95]:
```
k5cls
0    6429
1    2483
2    7552
3    3087
4    3618
5    3251
dtype: int64
```

In [98]:

```
fig,ax = plt.subplots(figsize=(15,15))
# Assign labels into a column
final_df["k5cls"] = k5cls.labels_
# Set up figure and ax
#f, ax = plt.subplots(1, figsize=(9, 9))
# Plot unique values choropleth including
# a legend and with no boundary lines
final_df.plot(
    column="k5cls",
    categorical=True,
    legend=True,
    linewidth=0,
    ax=ax,
    cmap='YlOrRd',
)
# Remove axis
ax.set_axis_off()
# Add basemap
ctx.add_basemap(
    ax, crs=final_df.crs,source=ctx.providers.CartoDB.Positron)
# Display the map
plt.show()
```


Everett
REVERE
Belmont
CHELSEA
SOMERVILLE
Winthro
Watertown
CAMBRIDGE
Brooklin
OAK HILL
Dedham
Milton
QUINCY
0
1
2
3
4
5

- The clustering map provide a useful view of the crime offenses reported. We could observed some patterns emerging. This emerging pattern will be much interpretive when complement the geovisualization of the clusters with statistical properties of the cluster map.

In [99]:
```
# Group table by cluster label, keep the variables used
# for clustering, and obtain their mean
k5mean = final_df.groupby("k5cls")[cluster_variables].mean()
# Transpose the table and print it rounding each value
# to three decimals
k5mean.T.round(3)
```

Out[99]:

k5cls	0	1	2	3	4	5
CRIME_LAG	556.498	3020.670	640.055	608.593	841.596	444.348
INVESTIGATE PERSON	0.000	0.310	1.000	0.000	0.000	0.000
INVESTIGATE PROPERTY	0.000	0.166	0.000	1.000	0.000	0.000
PROPERTY DAMAGE	0.000	0.379	0.000	0.000	1.000	0.000
SICK ASSIST	1.000	0.108	0.000	0.000	0.000	0.000
TOWED MOTOR VEHICLE	0.000	0.037	0.000	0.000	0.000	1.000

- From the table above we can draw some meaning from the first clustering algorithm use, which is the kMeans. Cluster 1 had the most crime cases reported with particular offenses like property damage, towed vehicle and investigated person respectively. Cluster 4 rank after cluster 1 in terms of crime cases reported in this tracts, where property damage is the most reported crime offense in cluster 4. Cluster 1 is dominated by sick assistant offense.

In [100…
```
# Set seed for reproducibility
np.random.seed(45)
# Initialize the algorithm
model = AgglomerativeClustering(linkage="ward", n_clusters=6)
# Run clustering
model.fit(db_scaled)
```

```
# Assign labels to main data table
final_df["ward5"] = model.labels_
```

In [101…

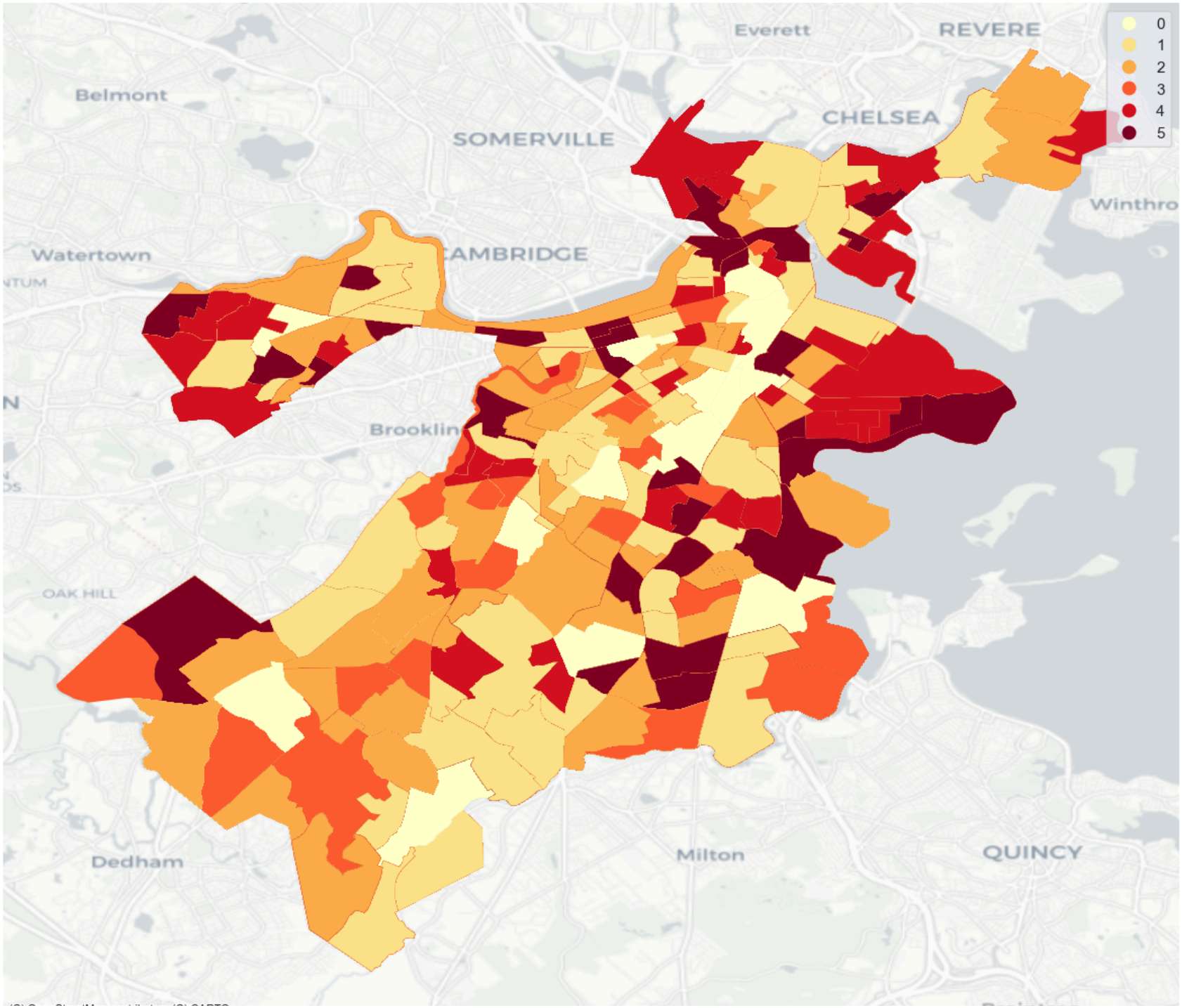
```
ward5sizes = final_df.groupby("ward5").size()
ward5sizes
```

Out[101…

```
ward5
0    6657
1    5604
2    6141
3    2619
4    3026
5    2373
dtype: int64
```

In [102…

```
fig,ax = plt.subplots(figsize=(15,15))
# Set up figure and ax
#f, ax = plt.subplots(1, figsize=(9, 9))
# Plot unique values choropleth including
# a legend and with no boundary lines
final_df.plot(
    column="ward5",
    categorical=True,
    legend=True,
    linewidth=0,
    ax=ax,
    cmap='YlOrRd',
)
# Remove axis
ax.set_axis_off()
# Add basemap
ctx.add_basemap(
    ax, crs=final_df.crs,source=ctx.providers.CartoDB.Positron)
# Display the map
plt.show()
```


0
1
2
3
4
5
Everett
REVERE
Belmont
SOMERVILLE
CHELSEA
Winthro
Watertown
CAMBRIDGE
Brooklin
OAK HILL
Dedham
Milton
QUINCY

- Another cluster algorithmn Agglomerative hierarchical was use to further investigate the top 5 crime offenses reported in Boston. Cluster 0 is has the same properties of cluster 1 in the kmeans.

In [103…
```
# Group table by cluster label, keep the variables used
# for clustering, and obtain their mean
k3mean_1 = final_df.groupby("ward5")[cluster_variables].mean()
# Transpose the table and print it rounding each value
# to three decimals
k3mean_1.T.round(3)
```

Out[103…

ward5	0	1	2	3	4	5
CRIME_LAG	2162.024	395.128	409.476	422.501	355.731	411.075
INVESTIGATE PERSON	0.327	0.000	1.000	0.000	0.000	0.000
INVESTIGATE PROPERTY	0.132	0.000	0.000	1.000	0.000	0.000
PROPERTY DAMAGE	0.328	0.000	0.000	0.000	0.000	1.000
SICK ASSIST	0.164	1.000	0.000	0.000	0.000	0.000
TOWED MOTOR VEHICLE	0.048	0.000	0.000	0.000	1.000	0.000

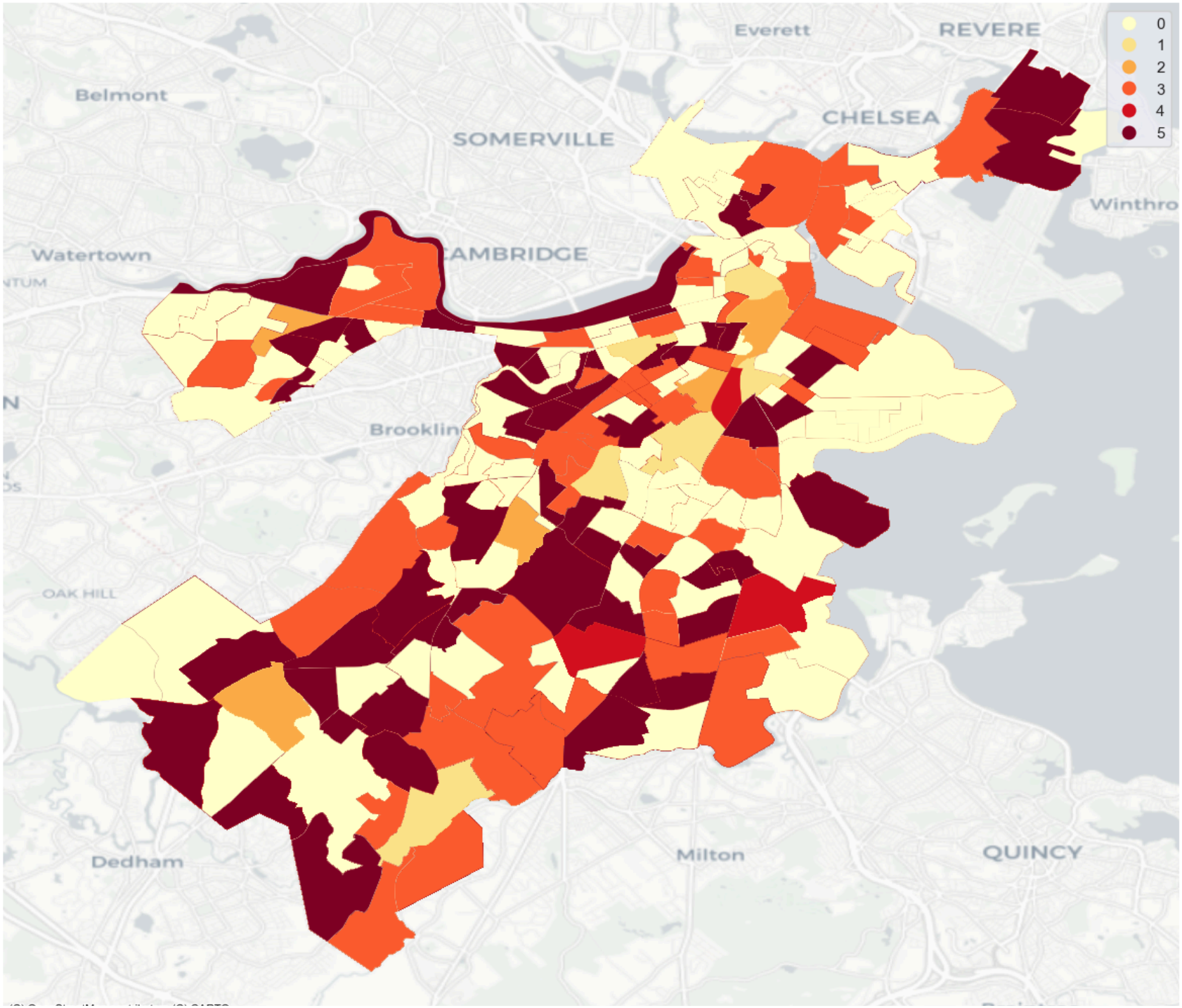
In [104…
```
np.random.seed(54321)
```

In [105…
```
model = AgglomerativeClustering(
linkage="ward", connectivity=wq.sparse, n_clusters=6
)
```

In [106…
```
# Fit the algorithm to the data
model.fit(db_scaled)
# Assign the labels to dataframe
final_df["ward5wgt_label"] = model.labels_
```

In [107…
```
fig,ax = plt.subplots(figsize=(15,15))
# Set up figure a
#f, ax = plt.subplots(1, figsize=(9, 9))
```

```
# Plot unique values choropleth including
# a legend and with no boundary lines
final_df.plot(
    column="ward5wgt_label",
    categorical=True,
    legend=True,
    linewidth=0,
    ax=ax,
    cmap='YlOrRd',
)
# Remove axis
ax.set_axis_off()
# Add basemap
ctx.add_basemap(
    ax, crs=final_df.crs,source=ctx.providers.CartoDB.Positron)
# Display the map
plt.show()
```


0
1
2
3
4
5
Everett
REVERE
Belmont
CHELSEA
SOMERVILLE
Winthro
Watertown
AMBRIDGE
Brooklin
OAK HILL
Dedham
Milton
QUINCY

- Another clustering algorithm was use which involve geography. Regionalization methods are clustering techniques that incorporate a spatial constraint into the clustering process. This constraint ensures that the resulting clusters exhibit geographical coherence along with coherent data profiles. Essentially, regionalization algorithms generate clusters where all constituent elements are internally connected, forming distinct regions. As a result, the members of a region are geographically nested within the boundaries defined by that region.
- Introducing the spatial constraint leads to fully connected clusters characterized by more concentrated spatial distributions. At first glance, it may seem that our spatial constraint has been breached, as there are tracts in both cluster 5 and cluster 2 that seem disconnected from the main body of their respective clusters. However, upon closer examination, it becomes evident that each of these tracts is connected to another tract within its own cluster via very narrow shared boundaries.

In [108…
```
ward5wgt = final_df.groupby("ward5wgt_label").size()
ward5wgt
```

Out[108…
```
ward5wgt_label
0    8065
1    3015
2    1879
3    5474
4    1782
5    6205
dtype: int64
```

In [109…
```
# Group table by cluster label, keep the variables used
# for clustering, and obtain their mean
k3mean_geo = final_df.groupby("ward5wgt_label")[cluster_variables].mean()
# Transpose the table and print it rounding each value
# to three decimals
k3mean_geo.T.round(3)
```

Out[109…

ward5wgt_label	0	1	2	3	4	5
CRIME_LAG	382.527	1737.669	1875.568	396.744	3165.430	422.082
INVESTIGATE PERSON	0.020	0.000	0.984	0.000	0.186	0.963
INVESTIGATE PROPERTY	0.309	0.143	0.016	0.006	0.235	0.015
PROPERTY DAMAGE	0.273	0.501	0.000	0.014	0.378	0.016
SICK ASSIST	0.047	0.273	0.000	0.955	0.150	0.000
TOWED MOTOR VEHICLE	0.350	0.083	0.000	0.025	0.052	0.006

- From the table above we can draw some meaning from the regional clustering algorithm use. Cluster 4 had the most crime cases reported with particular offenses like property damage and investigated property. Cluster 2 rank after cluster 4 in terms of crime cases reported in this tracts, where investigated person is the most reported crime offense in cluster 4. Cluster 1 is dominated by property damage offense.

In [113…

```
from sklearn.metrics import calinski_harabasz_score, davies_bouldin_score, silhouette_score
from sklearn import metrics
```

In [115…

```
ch_scores = []
db_scores = []
s_scores = []
```

In [116…

```
for model in ("k5cls", "ward5", "ward5wgt_label"):
    # compute the CH score
    ch_score = calinski_harabasz_score(
        final_df[cluster_variables],
        final_df[model],
    )
    ch_scores.append((model, ch_score))

    # compute the DB score
    db_score = davies_bouldin_score(
        final_df[cluster_variables],
        final_df[model],
    )
```

```
    db_scores.append((model, db_score))
    # compute the silhouette score
    s_score = silhouette_score(
    final_df[cluster_variables],
    final_df[model],
    )
    s_scores.append((model, s_score))
```

In [117…

```
# create a dataframe from the scores
ch_df = pd.DataFrame(
    ch_scores, columns=["model", "CH Score"]
).set_index("model")
db_df = pd.DataFrame(
    db_scores, columns=["model", "DB Score"]
).set_index("model")

s_df = pd.DataFrame(
s_scores, columns=["model", "Silhouettescore"]
).set_index("model")

# Merging into a combined dataframe
# Merging into a combined dataframe
scores_df = ch_df.merge(db_df, on="model")
scores_df = scores_df.merge(s_df, on="model")
# displaying the dataframe
scores_df
```

Out[117…

model	CH Score	DB Score	Silhouettescore
k5cls	10609.357963	12.482683	-0.082089
ward5	18431.665398	68.728323	0.086388
ward5wgt_label	47475.721292	12.334331	0.032158

- From the above table we can observed that the spatially constrained agglomerative hierarchical clustering performs better than the rest of the clustering algorithm, though in violent some spatial constraint.

In [ ]: