Assignment Deliverables

Requirement #1:

- Calculate the gun crimes density in every district

```
# Recall list of the count of gun violent crimes by district, then convert list to pd.df.
gun = '%GUN%'
cursor.execute("SELECT district, count(district) from crimes where DESCRIPTION::text LIKE %s GROUP BY district",[gun])
districts_gun_violent_crimes = cursor.fetchall()
districts_gun_violent_crimes_df = pd.DataFrame(districts_gun_violent_crimes, columns = ['district_number','gun_violent_crimes_count'])
districts_gun_violent_crimes_df['district_number'] = districts_gun_violent_crimes_df['district_number'].astype(str)


# Obtain the district area in ha.
district = []
tarea = []

with open('Boundaries.geojson') as f:
    data = json.load(f)
    a = data['features']
    for i in range(len(a)):
        obj = a[i]['geometry']
        n = a[i]['properties']
        district.append(n['dist_num'])
        tarea.append(area(obj)/10000)

af = pd.DataFrame({'dist_num': district,'district_area_in_ha':tarea})
af['district_number'] = af['dist_num'].astype(str)
af.drop('dist_num', axis=1, inplace=True)


# Merge the two DataFrames.
final_data = pd.merge(af, districts_gun_violent_crimes_df, on='district_number', how='inner')
final_data = final_data[['district_number', 'gun_violent_crimes_count', 'district_area_in_ha']]

# Add a final column for the gun crime density.
final_data['gun_crime_density'] = round(final_data['gun_violent_crimes_count']/(final_data['district_area_in_ha']/100),2)
final_data.sort_values(by='gun_crime_density', ascending=False)
```

	district_number	gun_violent_crimes_count	district_area_in_ha	gun_crime_density
17	15	2015	989.631393	203.61
16	11	3175	1582.727274	200.60
5	7	2739	1688.670732	162.20
8	6	2598	2099.682124	123.73
6	3	1928	1576.063931	122.33
18	10	2188	2038.988883	107.31
21	2	1348	1949.690970	69.14
3	25	1738	2827.989237	61.46
10	5	1925	3318.613379	58.01
15	12	1334	2509.453028	53.16
4	14	822	1555.869965	52.83
20	9	1794	3505.216898	51.18
11	24	540	1406.081387	38.40
14	18	411	1215.520046	33.81
19	1	410	1214.818895	33.75

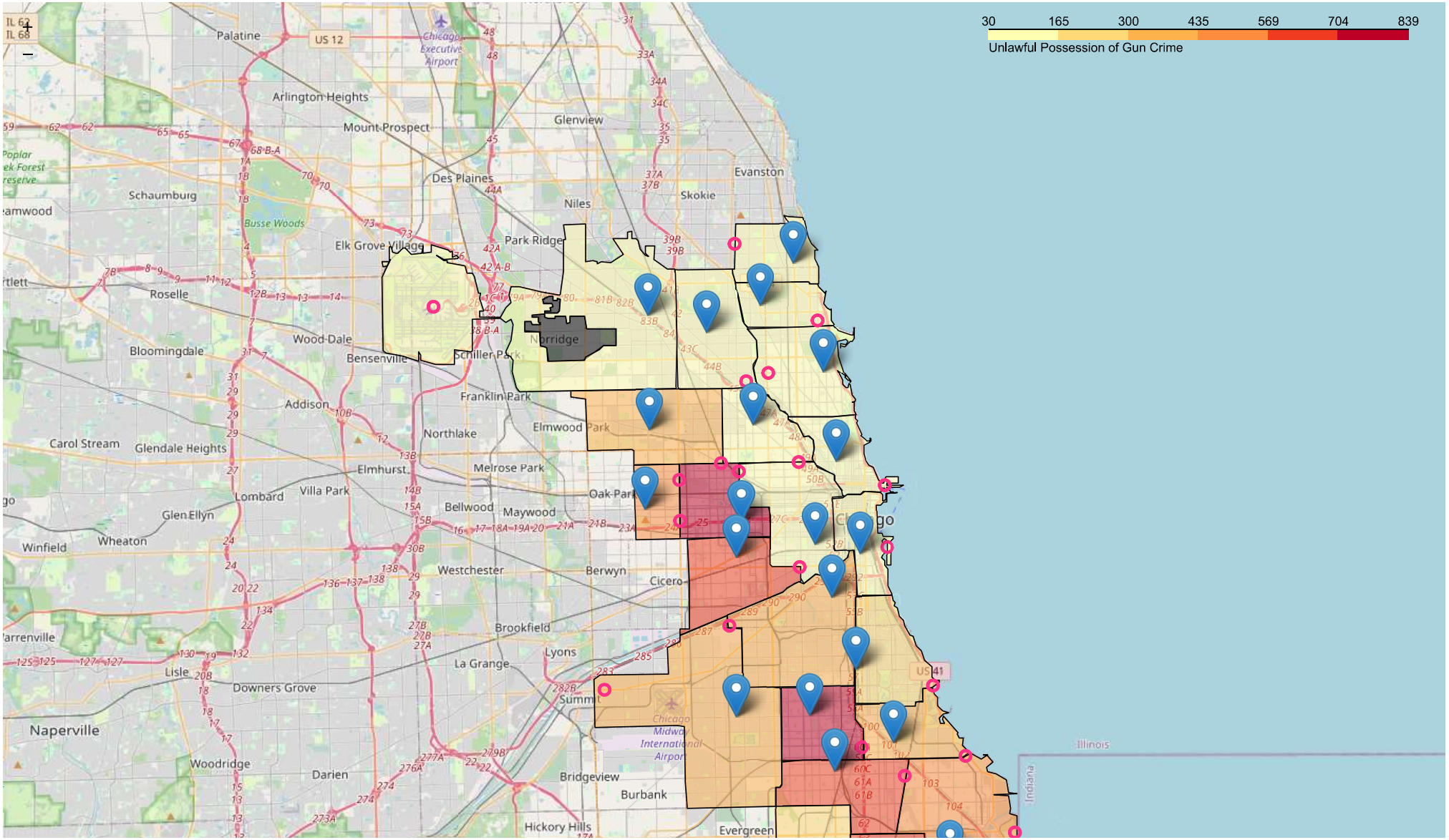
Requirement #2:

- Locate the **farthest** UNLAWFUL POSS OF HANDGUN crime from the police station in every district. The popup on Choropleth map shall display the district number and the block

```
# Read query for unlawful poss of handgun by district count. Bring values into df.
unlawful_gun='%UNLAWFUL POSS OF HANDGUN%'
cursor.execute("SELECT district, count(district) from crimes where DESCRIPTION::text LIKE %s GROUP BY district",[unlawful_gun])
districts_gun_violent_crimes = cursor.fetchall()
districts_gun_violent_crimes_df = pd.DataFrame(districts_gun_violent_crimes, columns=['dist_num','unlawful_gun'])
districts_gun_violent_crimes_df['dist_num'] = districts_gun_violent_crimes_df['dist_num'].astype(str)


# Initialize choropleth map, bring in the boundaries geojson.
farthest_block_gun_crime_map = folium.Map(location =(41.8781, -87.6298),zoom_start=11)
folium.Choropleth(geo_data="Boundaries.geojson",
              fill_color='YlOrRd',
              fill_opacity=0.5,
              line_opacity=1,
              data = districts_gun_violent_crimes_df,
              key_on='feature.properties.dist_num',
              columns = ['dist_num', 'unlawful_gun'],
              legend_name="Unlawful Possession of Gun Crime"
              ).add_to(farthest_block_gun_crime_map)
```

```
 <folium.features.Choropleth at 0x28c902548e0>
```

```
# Fetch list of police stations for district.
cursor.execute("""SELECT ST_X(ST_AsText(Where_IS)), ST_Y(ST_AsText(Where_IS)), district from police_stations where district!='Headquarters'""")
police_stations = cursor.fetchall()

unlawful_gun='%UNLAWFUL POSS OF HANDGUN%'

for police_station in police_stations:

    cursor.execute("""
                        WITH crime_distance_from_police_station AS (
                                SELECT  A.where_is as crime_where_is,
                                        B.where_is as police_where_is,
                                        A.district as crime_district,
                                        A.block as crime_block,
                                        ST_Distance(A.where_is,B.where_is) as distance
                                FROM crimes as A, police_stations as B
                                WHERE A.district=%s and DESCRIPTION::text LIKE %s and B.district= %s
                        )

                        SELECT DISTINCT on (crime_distance_from_police_station.crime_block)
                                crime_distance_from_police_station.crime_district,
                                crime_distance_from_police_station.crime_block,
                                crime_distance_from_police_station.crime_where_is,
                                crime_distance_from_police_station.distance
                        FROM crime_distance_from_police_station
                        WHERE distance = (SELECT max(distance)
                                FROM crime_distance_from_police_station )""",[police_station[2],unlawful_gun,police_station[2]])

    farthest_block_gun_crime = cursor.fetchall()

    # Check to see if farthest_block_gun_crime list is empty before proceeding with next query, otherwise this will break the for loop.
    if len(farthest_block_gun_crime)!=0:
        cursor.execute("SELECT ST_X(ST_AsText(%s)), ST_Y(ST_AsText(%s))",(farthest_block_gun_crime[0][2],farthest_block_gun_crime[0][2]))
        farthest_block_gun_crime_location = cursor.fetchall()
        folium.Marker(location=(police_station[0],police_station[1]),popup=folium.Popup(html="Police Station <br> District No.:%s <br> Farthest Gun_Crime Block:%s"%(farthest_block_gun_
        folium.CircleMarker(farthest_block_gun_crime_location[0],radius=5,color='#ff3187',popup=folium.Popup(html="District No.:%s <br> Block:%s"%(farthest_block_gun_crime[0][0],farthe
    else:
        exit()

farthest_block_gun_crime_map
```



Requirement #3:

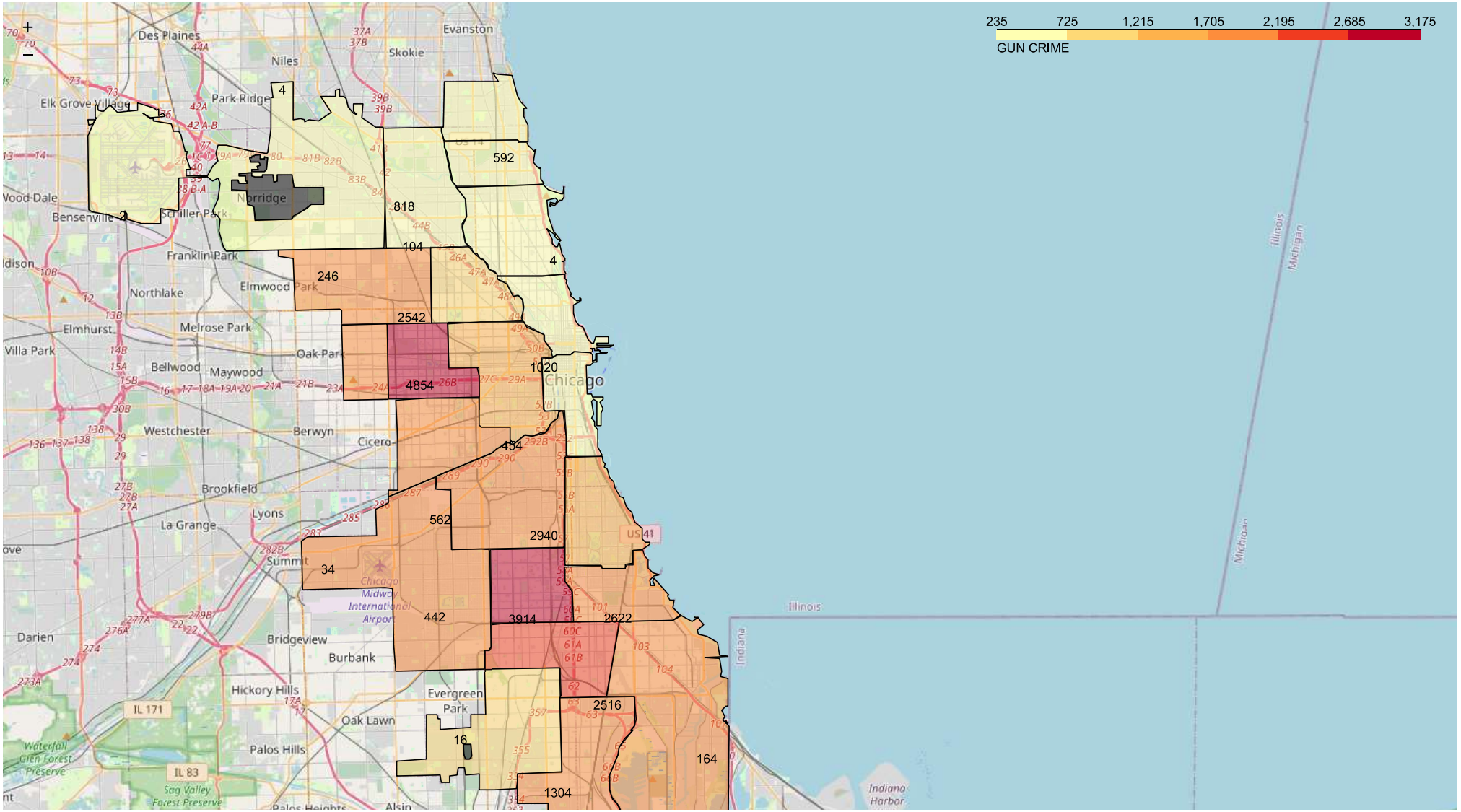
- Create **Marker Clusters** on Choropleth map for those **gun related violent crimes** that have Location Desciption as RESIDENCE in ** (green icon)** and those that have Location Desciption as STREET in **(red icon)**

```
# Count all gun crimes by district
gun='%GUN%'
cursor.execute("SELECT district, count(district) from crimes where DESCRIPTION::text LIKE %s GROUP BY district",[gun])
districts_gun_violent_crimes = cursor.fetchall()
```

```
districts_gun_violent_crimes_df = pd.DataFrame(districts_gun_violent_crimes, columns=['dist_num','gun_crimes'])
districts_gun_violent_crimes_df['dist_num'] = districts_gun_violent_crimes_df['dist_num'].astype(str)



# Initialize gun_crime_location_map choropleth map, import geojson data.
gun_crime_location_map = folium.Map(location =(41.8781, -87.6298),zoom_start=11)
folium.Choropleth(geo_data="Boundaries.geojson",
              fill_color='YlOrRd',
              fill_opacity=0.5,
              line_opacity=1,
              data = districts_gun_violent_crimes_df,
              key_on='feature.properties.dist_num',
              columns = ['dist_num', 'gun_crimes'],
              legend_name="GUN CRIME"
              ).add_to(gun_crime_location_map)
```

```
     <folium.features.Choropleth at 0x28c8fd85a60>
```

```
cursor.execute("""SELECT ST_X(ST_AsText(Where_IS)), ST_Y(ST_AsText(Where_IS)), district from police_stations where district!='Headquarters'""")
gun='%GUN%'

police_stations = cursor.fetchall()

marker_cluster = MarkerCluster().add_to(gun_crime_location_map)

# Scan each gun crime and set up green marker if location_description is RESIDENCE and red marker if it is 'STREET'. If another value, skip marker placement on map.
for police_station in police_stations:
    police_station_location = (police_station[0],police_station[1])
    cursor.execute("""SELECT DISTINCT ON(caseno) caseno, block,DESCRIPTION, count(location_description), location_description ,latitude, longitude from crimes where district=%s and DES
    crimes_per_district = cursor.fetchall()
    for crime in crimes_per_district:
        if crime[4]=='RESIDENCE':
            folium.Marker(location=(crime[5],crime[6]),popup=folium.Popup(html="District No: %s <br> Description: %s <br> Block: %s" %(police_station[2],crime[2],crime[1])),icon=folium
        elif crime[4]=='STREET':
            folium.Marker(location=(crime[5],crime[6]),popup=folium.Popup(html="District No: %s <br> Description: %s<br> Block: %s" %(police_station[2],crime[2],crime[1])),icon=folium.
        else:
            exit()

gun_crime_location_map
```



Requirement #4:

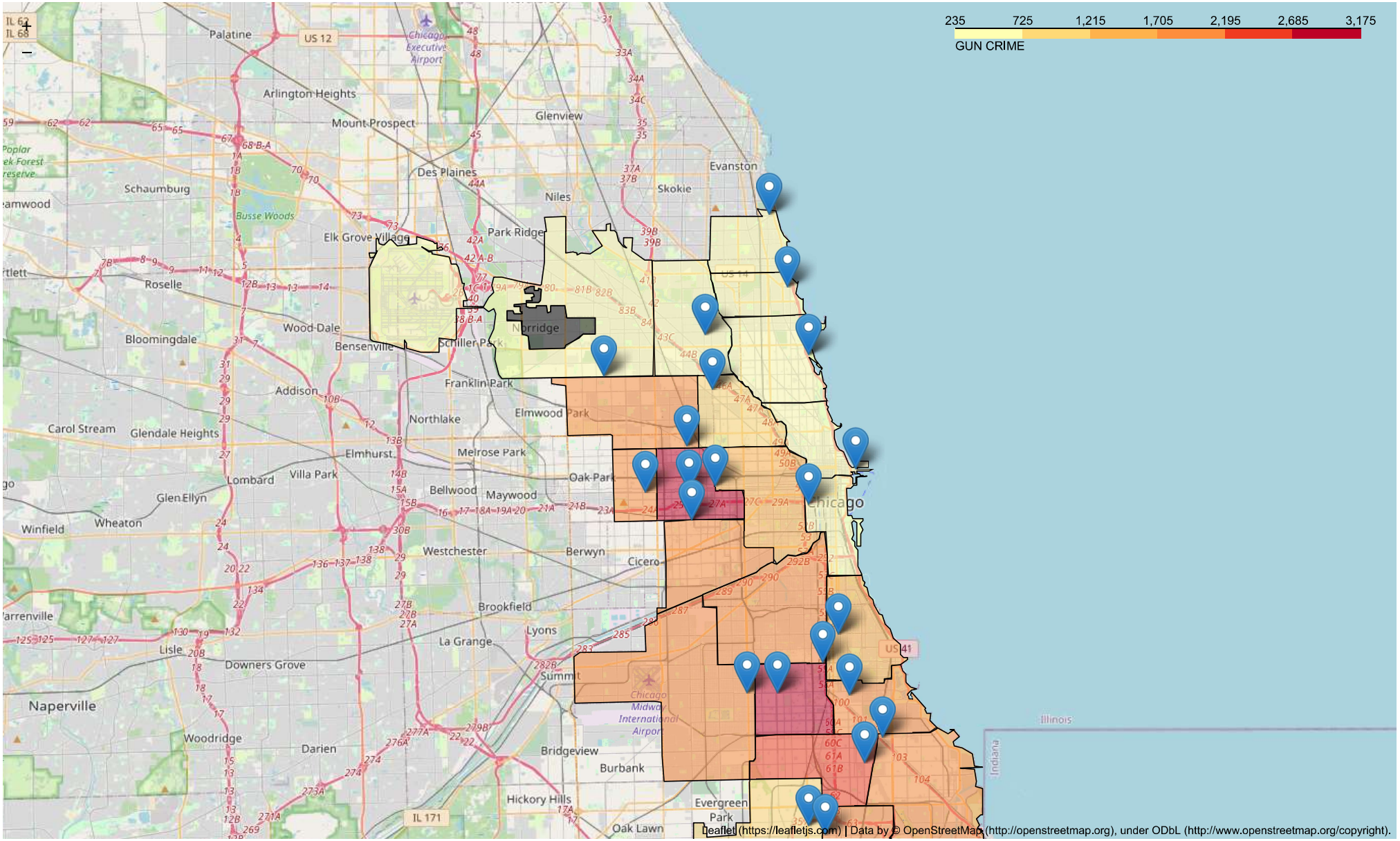
- Locate the **Block** that has the **highest number of gun crimes**. The popup on Choropleth map shall display the Block in every district along with the total number of gun crimes for that block

```
# Initialize choropleth map, import data from geojson.
highest_block_gun_crime_map = folium.Map(location =(41.8781, -87.6298),zoom_start=11)
folium.Choropleth(geo_data="Boundaries.geojson",
              fill_color='YlOrRd',
              fill_opacity=0.5,
```

```
            line_opacity=1,
            data = districts_gun_violent_crimes_df,
            key_on='feature.properties.dist_num',
            columns = ['dist_num', 'gun_crimes'],
            legend_name="GUN CRIME"
            ).add_to(highest_block_gun_crime_map)
```

```
 <folium.features.Choropleth at 0x28cc6696610>
```

```
# Obtain query of gun crimes grouped/sumed by block.

gun='%GUN%'

cursor.execute("""SELECT ST_X(ST_AsText(Where_IS)), ST_Y(ST_AsText(Where_IS)), district from police_stations where district!='Headquarters'""")
police_stations = cursor.fetchall()

for police_station in police_stations:

    cursor.execute("""
                        WITH gun_crime_by_block AS (
                                        SELECT district, block, count(block), where_is
                                        FROM crimes
                                        WHERE district=%s and DESCRIPTION::text LIKE %s
                                        GROUP BY district, block, where_is
                        )

                        SELECT DISTINCT on (gun_crime_by_block.block)
                                        gun_crime_by_block.district,
                                        gun_crime_by_block.block,
                                        gun_crime_by_block.count,
                                        gun_crime_by_block.where_is
                        FROM gun_crime_by_block
                        WHERE count = (SELECT max(count) FROM gun_crime_by_block )""",[police_station[2], gun])

    highest_block_gun_crime = cursor.fetchall()

    # Transform where_is to streets for intersection.
    cursor.execute("SELECT ST_X(ST_AsText(%s)), ST_Y(ST_AsText(%s))",(highest_block_gun_crime[0][3], highest_block_gun_crime[0][3]))
    block = cursor.fetchall()

    folium.Marker(location=(block[0][0],block[0][1]),popup=folium.Popup(html="District No. : %s <br> Highest Gun Crime Block: %s <br> Gun Crime Total: %s"%(highest_block_gun_crime[0][0

highest_block_gun_crime_map
```



Requirement #5:

- Create the **Trend-Chart** for every crime type (use the complete dataset).
  - Index the crimes/dataframe based on the crime date_of_occurrence
  - Pivot data based on the primary_type
  - Use https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rolling.html for the moving window (365 days for the year)

```
# Query all dates for all crimes, regardless of type.
cursor.execute("SELECT date_of_occurrence, primary_type FROM crimes")
rows = cursor.fetchall()
dates_df = pd.DataFrame(rows, columns=['date','crime_type'])
dates_df['date'] = dates_df['date'].apply(str)

# Parse each string so that only the date is listed (remove time stamp).
dates_df['date'] = dates_df['date'].str.split(' ').str[0]

# Add count to each row.
dates_df['count'] = 1

# Group by and sum instances, use as=index=False to output as SQL group by query.
indexed_dates_df = dates_df.groupby(['date','crime_type'], as_index=False)['count'].sum()

# Create pivot table and fill all NaN values with zero. Set index as the date column.
pivot_df = indexed_dates_df.pivot(index='date', columns='crime_type', values='count')
pivot_df.fillna(0, inplace=True)
pivot_df.head(5)
```

crime_type	ARSON	ASSAULT	BATTERY	BURGLARY	CONCEALED CARRY LICENSE VIOLATION	CRIM SEXUAL ASSAULT	CRIMINAL DAMAGE	CRIMINAL TRESPASS	DECEPTIVE PRACTICE	GAMBLING	...	OTHER NARCOTIC VIOLATION
date												
2016-01-01	1.0	42.0	228.0	14.0	0.0	28.0	112.0	8.0	131.0	0.0	...	0.0
2016-01-02	0.0	32.0	100.0	33.0	0.0	6.0	77.0	9.0	43.0	0.0	...	0.0
2016-01-03	1.0	39.0	124.0	24.0	0.0	1.0	81.0	16.0	28.0	0.0	...	0.0
2016-01-04	2.0	35.0	108.0	55.0	0.0	3.0	73.0	23.0	49.0	0.0	...	0.0
2016-01-05	0.0	41.0	112.0	31.0	0.0	1.0	66.0	10.0	51.0	0.0	...	0.0

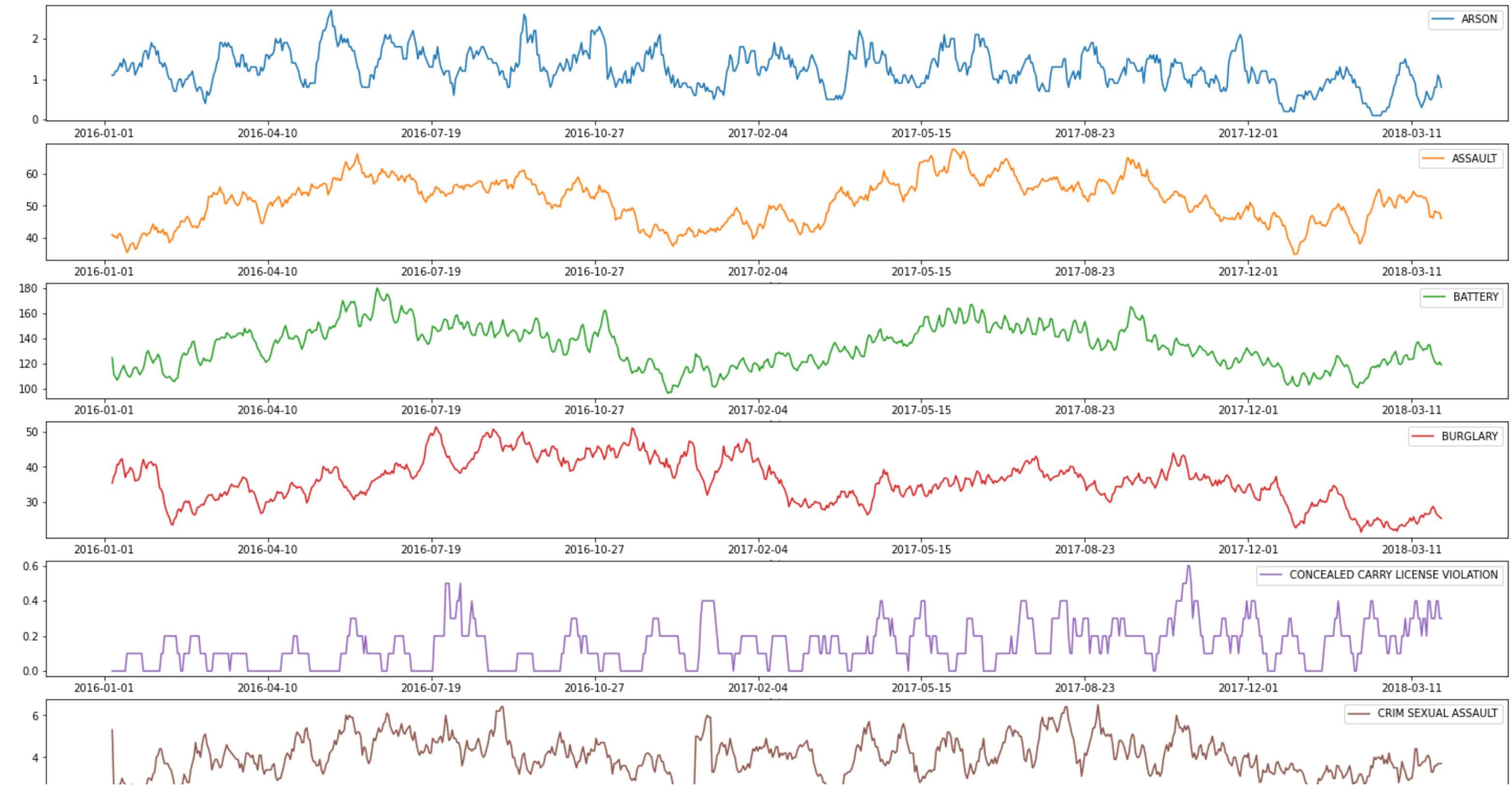
5 rows × 33 columns

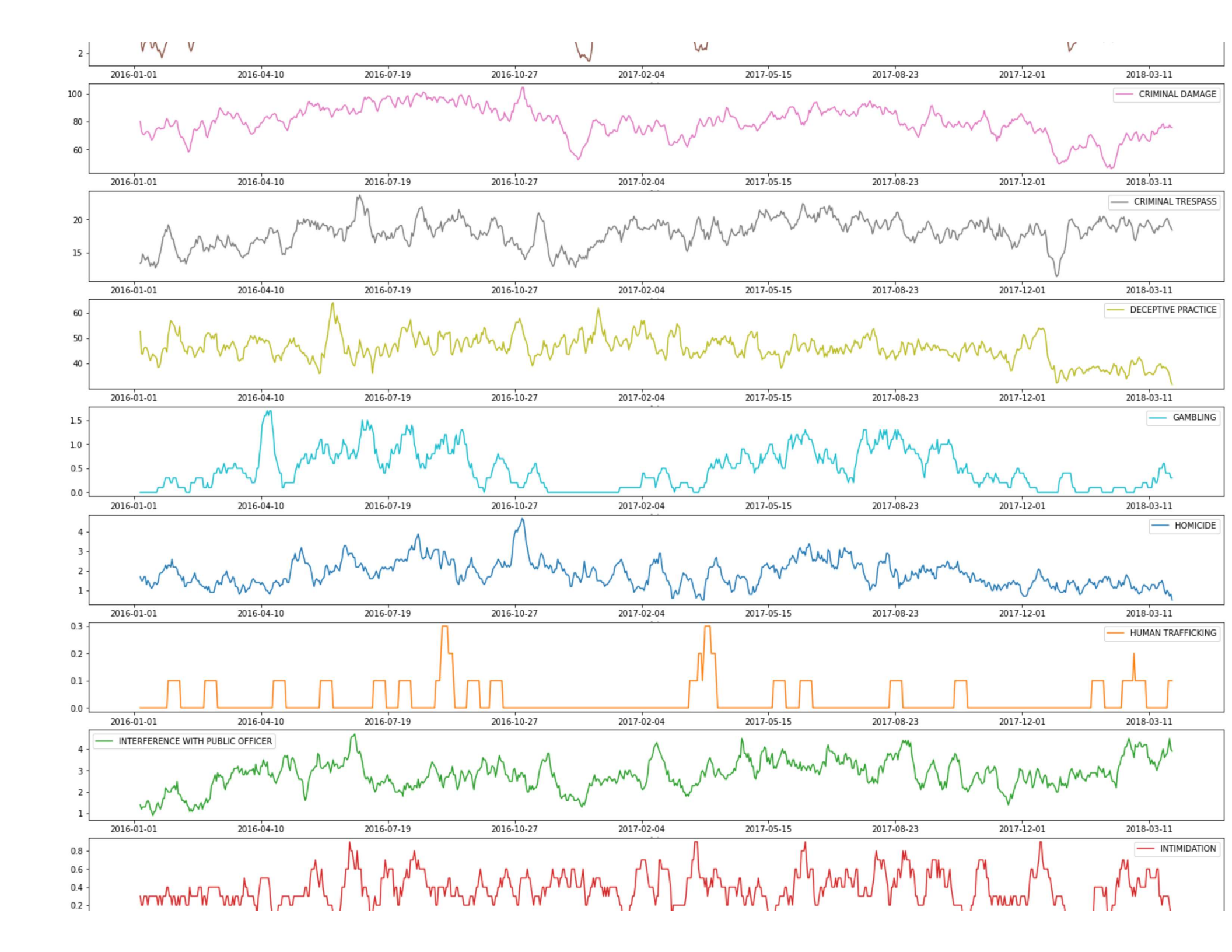
```
# Use rolling function for a centered ten day moving average
moving_average_df = pivot_df.rolling(window=10, center=True).mean()
moving_average_df.head(5)
```

crime_type	ARSON	ASSAULT	BATTERY	BURGLARY	CONCEALED CARRY LICENSE VIOLATION	CRIM SEXUAL ASSAULT	CRIMINAL DAMAGE	CRIMINAL TRESPASS	DECEPTIVE PRACTICE	GAMBLING	...	OTHER NARCOTIC VIOLATION	OTHER OFFENSE	PROSTITUTION	PUBLIC INDECENCY	PUBLIC PEACE VIOLATION	ROBBERY	OF
date																		

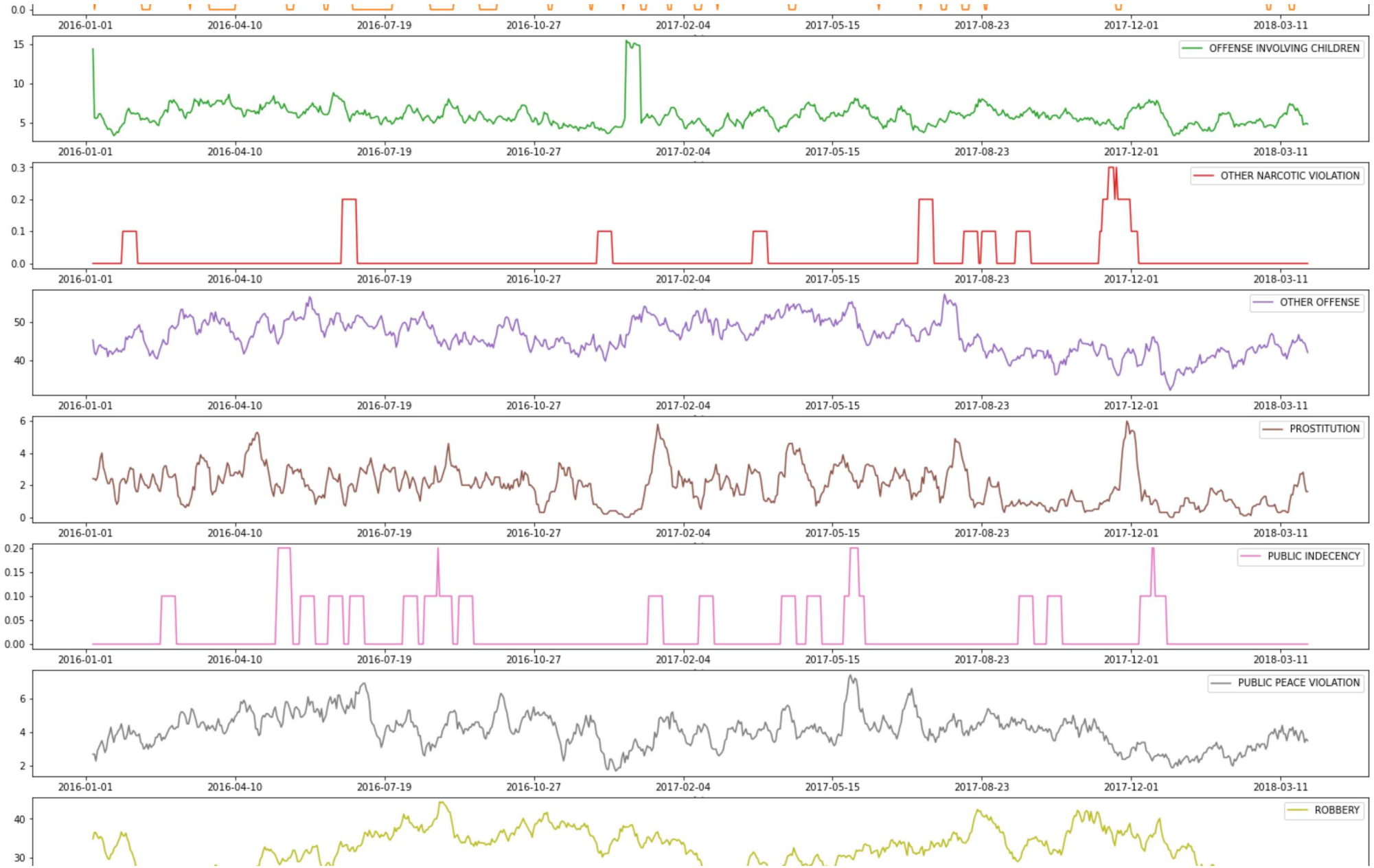
```
# Plot moving averages wrt to dates.
fig, a = plt.subplots(33, 1, figsize=(25, 80))
moving_average_df.plot(ax=a, subplots=True)
```

```
array([<AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>, <AxesSubplot:xlabel='date'>,
       <AxesSubplot:xlabel='date'>], dtype=object)
```




CRIMINAL DAMAGE
CRIMINAL TRESPASS
DECEPTIVE PRACTICE
GAMBLING
HOMICIDE
HUMAN TRAFFICKING
INTERFERENCE WITH PUBLIC OFFICER
INTIMIDATION
2016-01-01
2016-04-10
2016-07-19
2016-10-27
2017-02-04
2017-05-15
2017-08-23
2017-12-01
2018-03-11


KIDNAPPING
LIQUOR LAW VIOLATION
MOTOR VEHICLE THEFT
NARCOTICS
NON - CRIMINAL
NON-CRIMINAL
NON-CRIMINAL (SUBJECT SPECIFIED)
OBSCENITY
2016-01-01
2016-04-10
2016-07-19
2016-10-27
2017-02-04
2017-05-15
2017-08-23
2017-12-01
2018-03-11



Requirement #6:

- What is the day of the week that has the lowest number of crimes reported?
- What is the day of the week that has the highest number of crimes reported?

```
# Query for the date of occurrence for each crime.
```

```
cursor.execute("SELECT date_of_occurrence from crimes;")
rows = cursor.fetchall()
date_df = pd.DataFrame(rows, columns=['date_time_of_occurrence'])
date_df['date_time_of_occurrence'] = date_df['date_time_of_occurrence'].apply(str)

# Split date/time column to show only date.
date_df['date'] = date_df['date_time_of_occurrence'].str.split(' ').str[0]

# Determine the day of week from date.
date_df['day_of_week'] = pd.to_datetime(date_df['date'])
date_df['day_of_week'] = date_df['day_of_week'].dt.day_name()

# Count number of crimes by day of week
crime_count_by_day = date_df['day_of_week'].value_counts(dropna=True)
print(crime_count_by_day)
```

```
Friday       88586
Saturday     85319
Monday       83792
Thursday     82919
Tuesday      82384
Sunday       82216
Wednesday    81859
Name: day_of_week, dtype: int64
```