Final Project

Lara Tamer, Ralph Valery Valiere, Jakub Budz

2024-12-07

```
# Larger dataset here: https://drive.google.com/drive/folders/1dFGQ71CesghWY5dWehwiumX57TJQiy

# Base path (EACH PERSON MUST UPDATE TO WHERE THEIR CRIME DATA IS)
base_path = 'C:/Users/Jakub/Downloads/final_data'
```

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
from shapely.geometry import Point, shape
from shapely import wkt
import matplotlib.colors as mcolors
import spacy
import altair as alt
import time
import os
import json
import numpy as np
import seaborn as sns
from spacytextblob.spacytextblob import SpacyTextBlob
nlp = spacy.load('en_core_web_sm')
nlp.add_pipe('spacytextblob')
```

```
<spacytextblob.spacytextblob.SpacyTextBlob at 0x28976951550>
```

```
# Make function to analyze sentiment
def analyze_text(file_name):
    base_name = file_name.split('/')[-1].replace('.txt', '')
    with open(file_name, 'r', encoding='utf-8') as file:
        text = file.read()
```

```
    doc = nlp(text)
    polarity = doc._.blob.polarity
    subjectivity = doc._.blob.subjectivity

    sentence_polarities = []
    for i, sentence in enumerate(doc.sents):
        polarity = sentence._.blob.polarity
        sentence_polarities.append({"n": i + 1, f"{base_name}_polarity": polarity})
    df = pd.DataFrame(sentence_polarities)

    chart = alt.Chart(df).mark_line().encode(
        x=alt.X('n', title='Sentence Number'),
        y=alt.Y(f'{base_name}_polarity', title='Polarity')
    ).properties(
        title=
            {'text':base_name.capitalize(),
            'subtitle': [
                f'Section Polarity: {polarity:.2f}',
                f'Section Subjectivity: {subjectivity:.2f}']},
        width=100,
        height=100
    )
    return chart
```

```
# Intro
intro_path = 'data/intro.txt'
intro_chart = analyze_text(intro_path)

# Results
results_path = 'data/results.txt'
results_chart = analyze_text(results_path)

# Moving Forward
future_path = 'data/future.txt'
future_chart = analyze_text(future_path)

# Full text
full_path = 'data/full.txt'
full_chart = analyze_text(full_path)

# Pillar 1: Empower and Heal
p1_path = 'data/p1.txt'
```

```
p1_chart = analyze_text(p1_path)

# Pillar 2: Protect and Secure
p2_path = 'data/p2.txt'
p2_chart = analyze_text(p2_path)

# Pillar 3: Improve and Advance
p3_path = 'data/p3.txt'
p3_chart = analyze_text(p3_path)

# Pillar 4: Affect Public Policy
p4_path = 'data/p4.txt'
p4_chart = analyze_text(p4_path)

# Pillar 5: Plan and Coordinate
p5_path = 'data/p5.txt'
p5_chart = analyze_text(p5_path)

row1 = alt.hconcat(full_chart, intro_chart, future_chart)
row2 = alt.hconcat(results_chart, p1_chart, p2_chart)
row3 = alt.hconcat(p3_chart, p4_chart, p5_chart)

combined_chart = alt.vconcat(row1, row2, row3)
combined_chart.show()
combined_chart.save('pictures/combined_chart.png')
```

```
alt.VConcatChart(...)
```

```
# Load in the CSV files
# communities has demographic info for 77 communities
communities = pd.read_csv('data/chicago-community-areas.csv')

# hyde_park is hyde park crime info from 2015 to now
hyde_park = pd.read_csv('data/Hyde_Park_Crime_20241127.csv')

# homicides is violent crime from 1991 to present
homicides = pd.read_csv(
    'data/Violence_Reduction_-_Victims_of_Homicides_and_Non-Fatal_Shootings_20241125.csv')

# crimes is the full dataset of all crime from 2015 to present (11/27)
crimes = pd.read_csv(f'{base_path}/Crimes_-_2001_to_Present_20241127.csv')
```

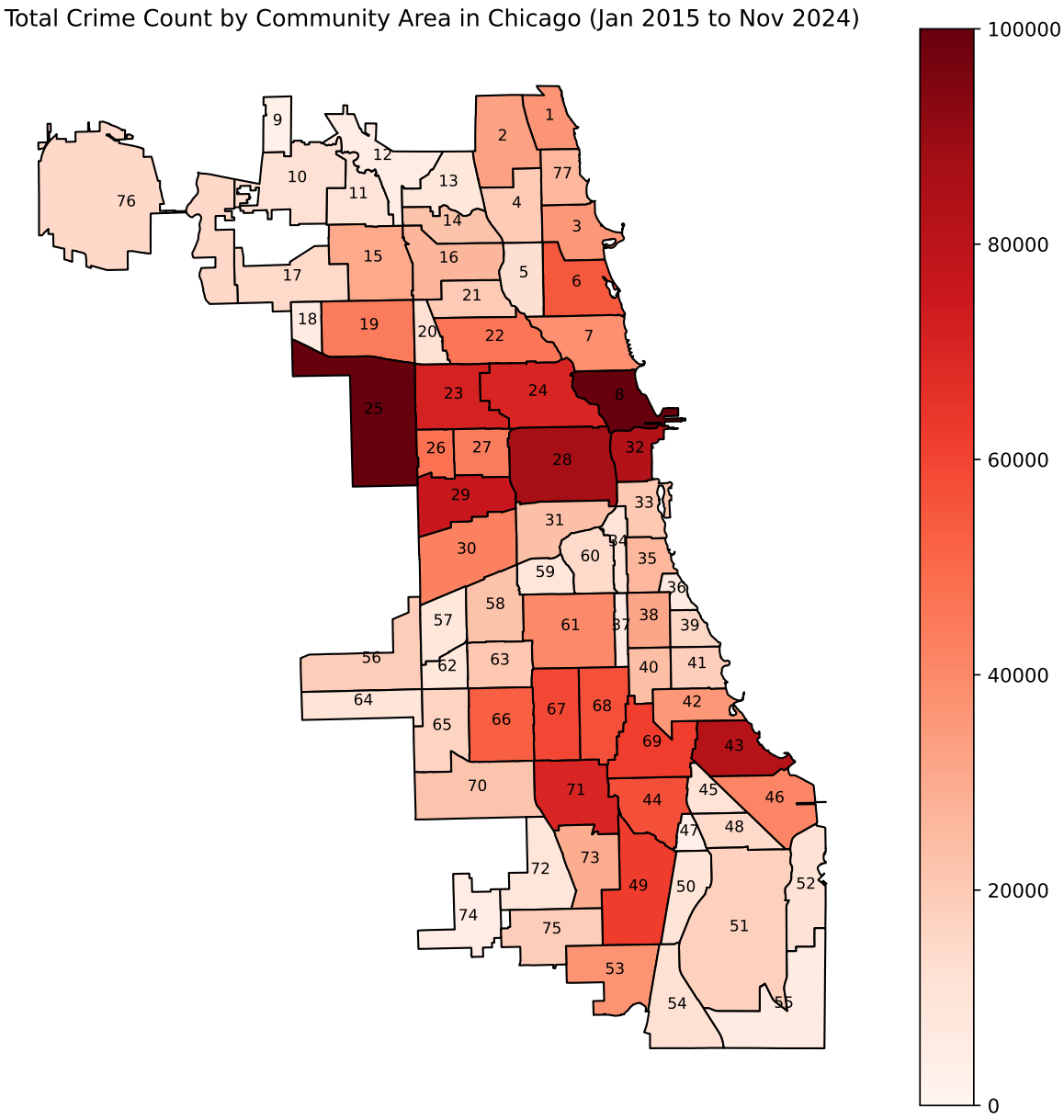
```
crimes['Date'] = pd.to_datetime(crimes['Date'])
crimes['geometry'] = crimes.apply(lambda row: Point(
    row['Longitude'], row['Latitude']), axis=1)
crimes_gdf = gpd.GeoDataFrame(crimes, geometry='geometry', crs="EPSG:4326")

# chicago has 77 community names and their associated community number
chicago = pd.read_csv('data/CommAreas_20241127.csv')
chicago['the_geom'] = chicago['the_geom'].apply(wkt.loads)
chicago_gdf = gpd.GeoDataFrame(chicago, geometry='the_geom', crs="EPSG:4326")
# Remove outliers
crimes_gdf = crimes_gdf[(crimes_gdf['Longitude'].between(-87.94011, -87.52413)) &
                        (crimes_gdf['Latitude'].between(41.64454, 42.02304))]
```

```
C:\Users\Jakub\AppData\Local\Temp\ipykernel_14352\2174377471.py:14: UserWarning: Could not in
  crimes['Date'] = pd.to_datetime(crimes['Date'])
```

```
# Spatial join crimes and community areas
joined_gdf = gpd.sjoin(crimes_gdf, chicago_gdf,
                       how="inner", predicate="within")

# Add crime counts to the geo df
crime_counts = joined_gdf.groupby(
    "index_right").size().reset_index(name="crime_count")
chicago_gdf = chicago_gdf.reset_index()
chicago_gdf = chicago_gdf.merge(
    crime_counts, left_index=True, right_on="index_right", how="left")
# replace na's with 0
chicago_gdf["crime_count"] = chicago_gdf["crime_count"].fillna(0)
```

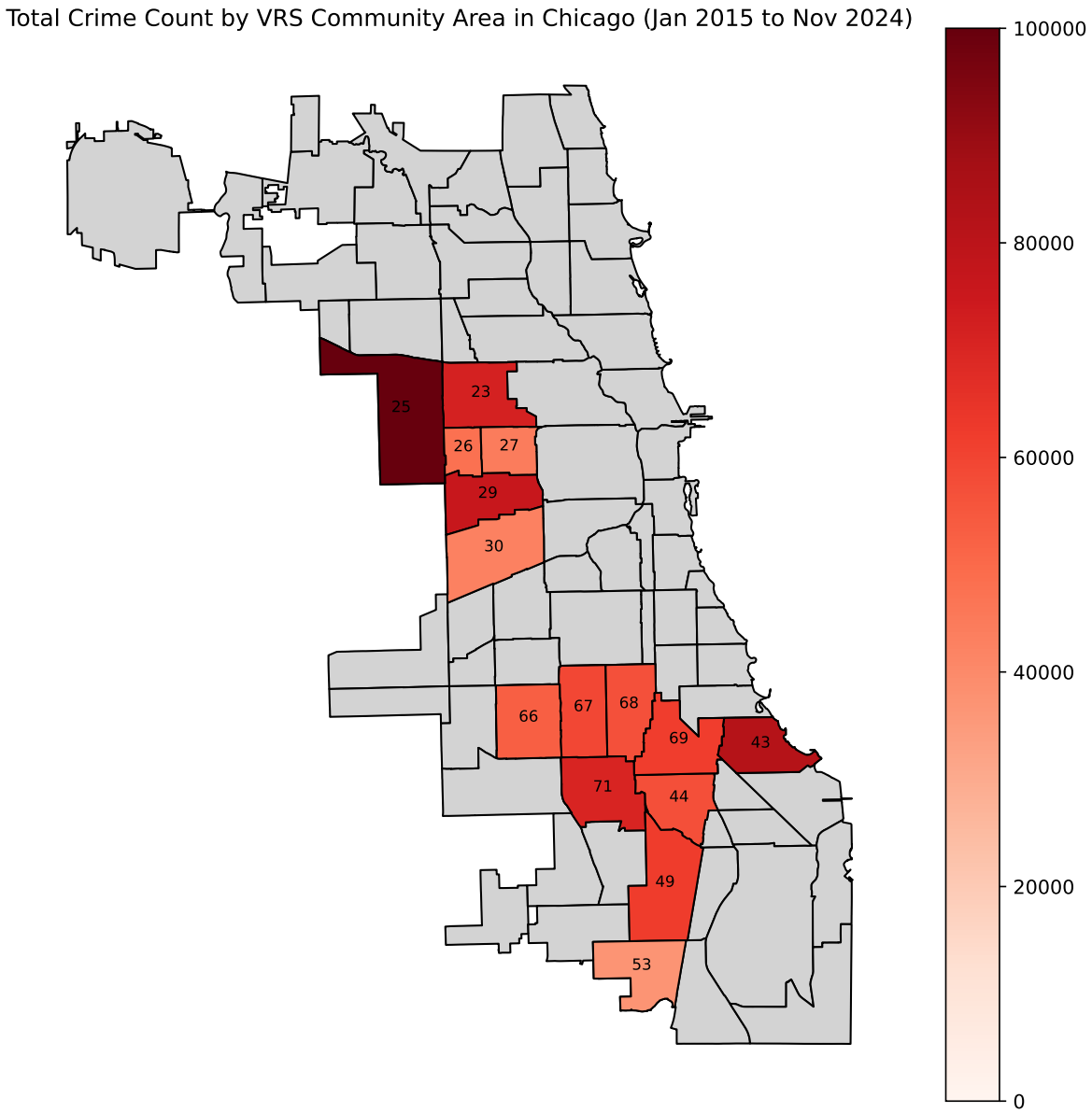
```
# Plot total crime per community area
fig, ax = plt.subplots(figsize=(10, 10))

chicago_gdf.plot(
    column="crime_count",
    cmap="Reds",
    edgecolor="black",
    legend=True,
    vmin=0,
    vmax=100000,
    ax=ax
)
```

```
# Used ChatGPT to assist in adding the community numbers to the graph
for idx, row in chicago_gdf.iterrows():
    if row["the_geom"].is_empty:
        continue
    centroid = row["the_geom"].centroid
    ax.annotate(
        text=str(row["AREA_NUM_1"]),
        xy=(centroid.x, centroid.y),
        horizontalalignment="center",
        fontsize=8,
        color="black"
    )

ax.axis("off")
for spine in ax.spines.values():
    spine.set_visible(False)

plt.title("Total Crime Count by Community Area in Chicago (Jan 2015 to Nov 2024)")
plt.savefig("pictures/comm_crime_total.png", bbox_inches='tight', dpi=300)
plt.show()
plt.close()
```




```
# Filter the data to just those 15 VRS communities
vrs_community = ["AUSTIN", "NORTH LAWNDALE", "HUMBOLDT PARK", "WEST GARFIELD PARK",
                 "ENGLEWOOD", "AUBURN GRESHAM", "WEST ENGLEWOOD", "GREATER GRAND CROSSING",
                 "ROSELAND", "EAST GARFIELD PARK", "SOUTH SHORE", "CHICAGO LAWN",
                 "SOUTH LAWNDALE", "CHATHAM", "WEST PULLMAN"]

vrs_gdf = chicago_gdf[chicago_gdf["COMMUNITY"].isin(vrs_community)]
```

```
# Plot total crime in VRS communities
fig, ax = plt.subplots(figsize=(10, 10))

chicago_gdf.plot(ax=ax, color='lightgrey', edgecolor='black')

vrs_gdf.plot(
    column="crime_count",
    cmap="Reds",
    edgecolor="black",
    legend=True,
    vmin=0,
    vmax=100000,
    ax=ax
)

for idx, row in vrs_gdf.iterrows():
    if row["the_geom"].is_empty:
        continue
    centroid = row["the_geom"].centroid
    ax.annotate(
        text=str(row["AREA_NUM_1"]),
        xy=(centroid.x, centroid.y),
        horizontalalignment="center",
        fontsize=8,
        color="black"
    )

ax.axis("off")
for spine in ax.spines.values():
    spine.set_visible(False)

plt.title("Total Crime Count by VRS Community Area in Chicago (Jan 2015 to Nov 2024)")
plt.savefig("pictures/vrs_comm_crime_total.png", bbox_inches='tight', dpi=300)
plt.show()
plt.close()
```

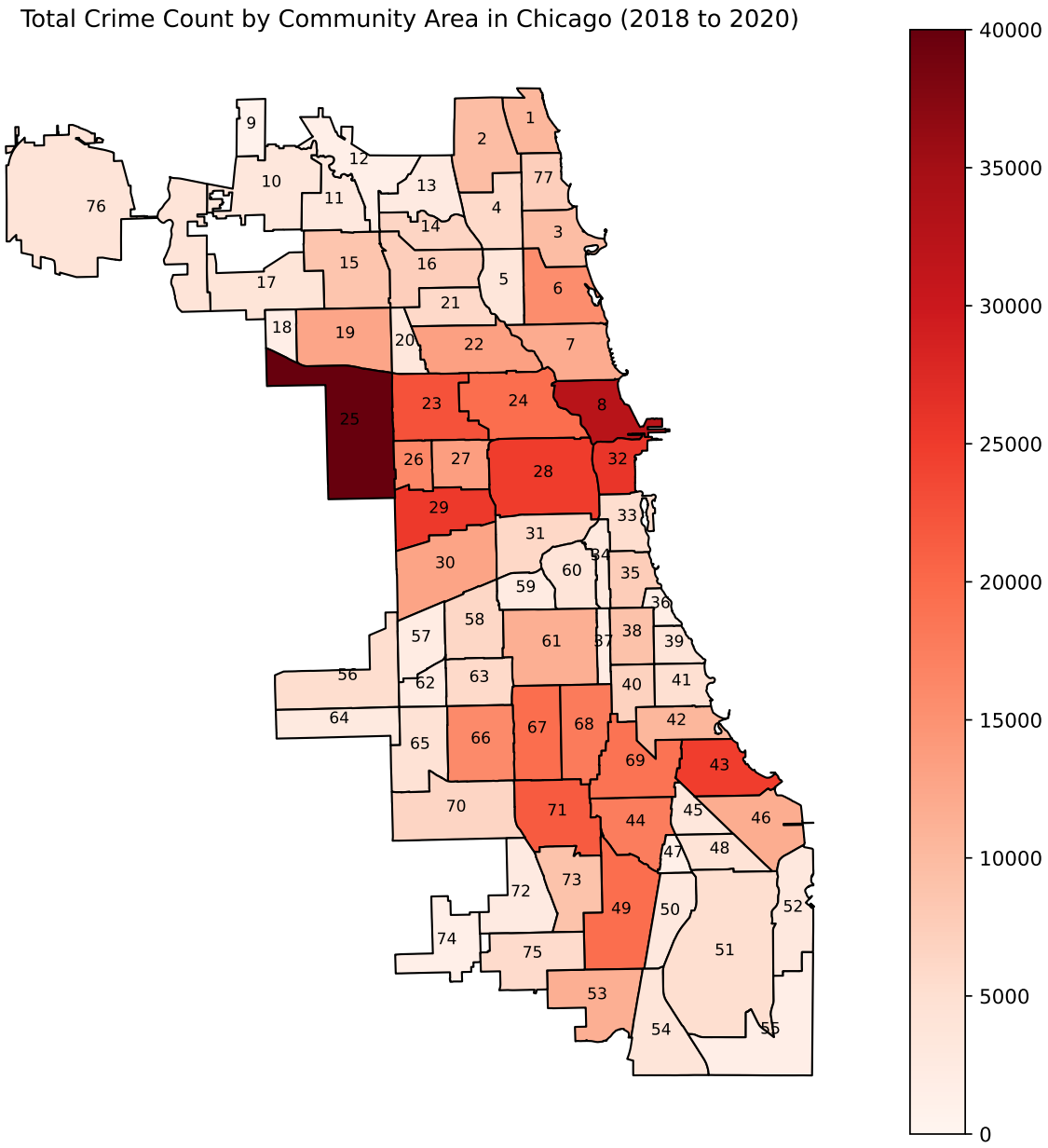



```
crimes_2018_2020 = crimes[(crimes['Date'] >= '2018-01-01') & (crimes['Date'] < '2021-01-01')]
crimes_2021_2023 = crimes[(crimes['Date'] >= '2021-01-01') & (crimes['Date'] < '2024-01-01')]

crimes_2018_2020_gdf = gpd.GeoDataFrame(crimes_2018_2020, geometry='geometry', crs="EPSG:4326
crimes_2021_2023_gdf = gpd.GeoDataFrame(crimes_2021_2023, geometry='geometry', crs="EPSG:4326

joined_2018_2020_gdf = gpd.sjoin(crimes_2018_2020_gdf, chicago_gdf,
                      how="inner", predicate="within")
```

```
joined_2021_2023_gdf = gpd.sjoin(crimes_2021_2023_gdf, chicago_gdf,
                        how="inner", predicate="within")

crime_counts_2018_2020 = joined_2018_2020_gdf.groupby(
    "index_right").size().reset_index(name="crime_count")
crime_counts_2021_2023 = joined_2021_2023_gdf.groupby(
    "index_right").size().reset_index(name="crime_count")

chicago_2018_2020_gdf = gpd.GeoDataFrame(chicago, geometry='the_geom', crs="EPSG:4326")
chicago_2021_2023_gdf = gpd.GeoDataFrame(chicago, geometry='the_geom', crs="EPSG:4326")

chicago_2018_2020_gdf = chicago_2018_2020_gdf.merge(
    crime_counts_2018_2020, left_index=True, right_on="index_right", how="left")
chicago_2021_2023_gdf = chicago_2021_2023_gdf.merge(
    crime_counts_2021_2023, left_index=True, right_on="index_right", how="left")

chicago_2018_2020_gdf["crime_count"] = chicago_2018_2020_gdf["crime_count"].fillna(0)
chicago_2021_2023_gdf["crime_count"] = chicago_2021_2023_gdf["crime_count"].fillna(0)
```

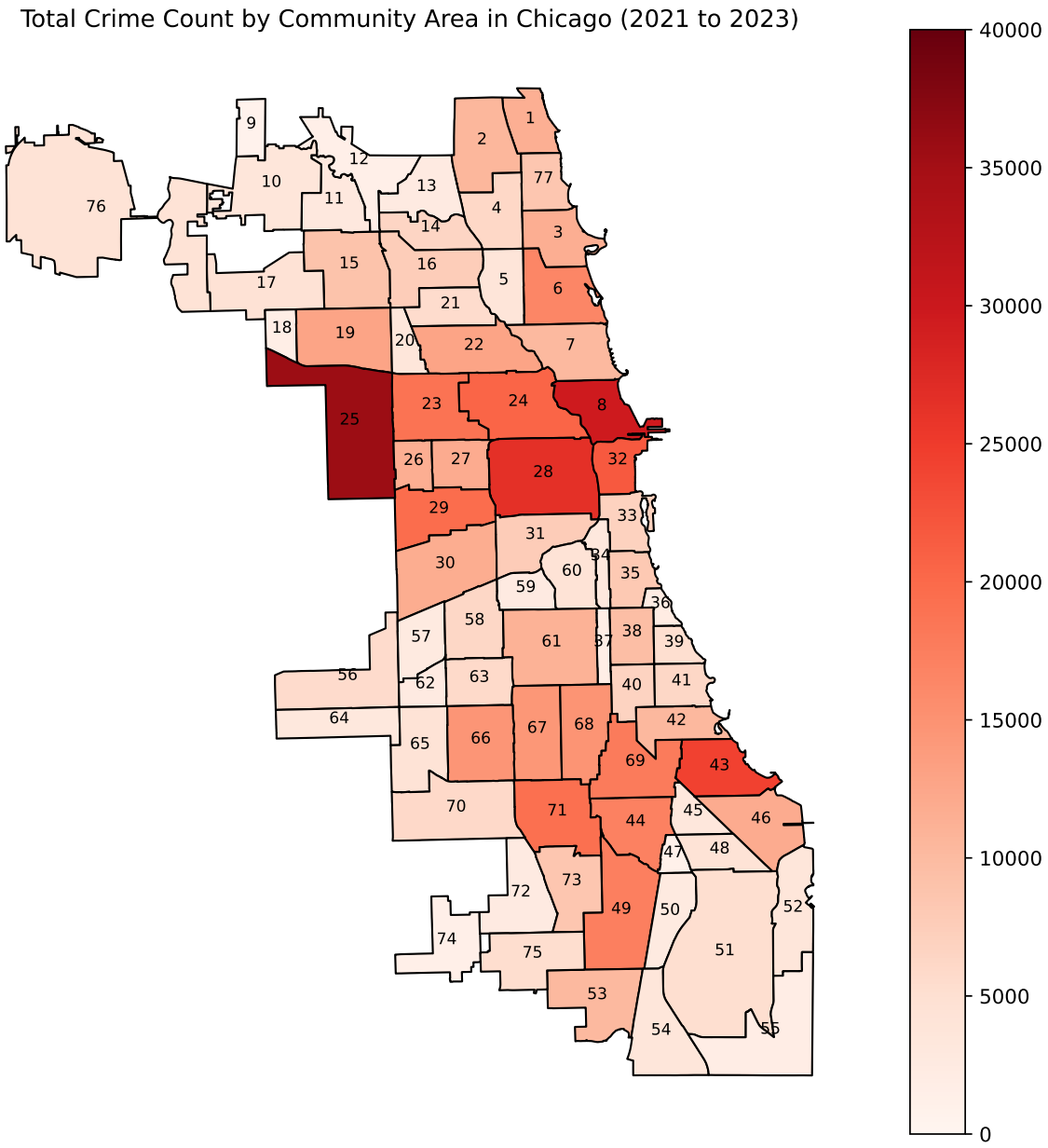
```
fig, ax = plt.subplots(figsize=(10, 10))

chicago_2018_2020_gdf.plot(
    column="crime_count",
    cmap="Reds",
    edgecolor="black",
    legend=True,
    ax=ax,
    vmin=0,
    vmax=40000
)

for idx, row in chicago_2018_2020_gdf.iterrows():
    if row["the_geom"].is_empty:
        continue
    centroid = row["the_geom"].centroid
    ax.annotate(
        text=str(row["AREA_NUM_1"]),
        xy=(centroid.x, centroid.y),
        horizontalalignment="center",
        fontsize=8,
        color="black"
```

```
    )

ax.axis("off")
for spine in ax.spines.values():
    spine.set_visible(False)

plt.title("Total Crime Count by Community Area in Chicago (2018 to 2020)")
plt.savefig("pictures/comm_crime_total_18_20.png", bbox_inches='tight', dpi=300)
plt.show()
plt.close()
```




```
fig, ax = plt.subplots(figsize=(10, 10))

chicago_2021_2023_gdf.plot(
    column="crime_count",
    cmap="Reds",
    edgecolor="black",
    legend=True,
```

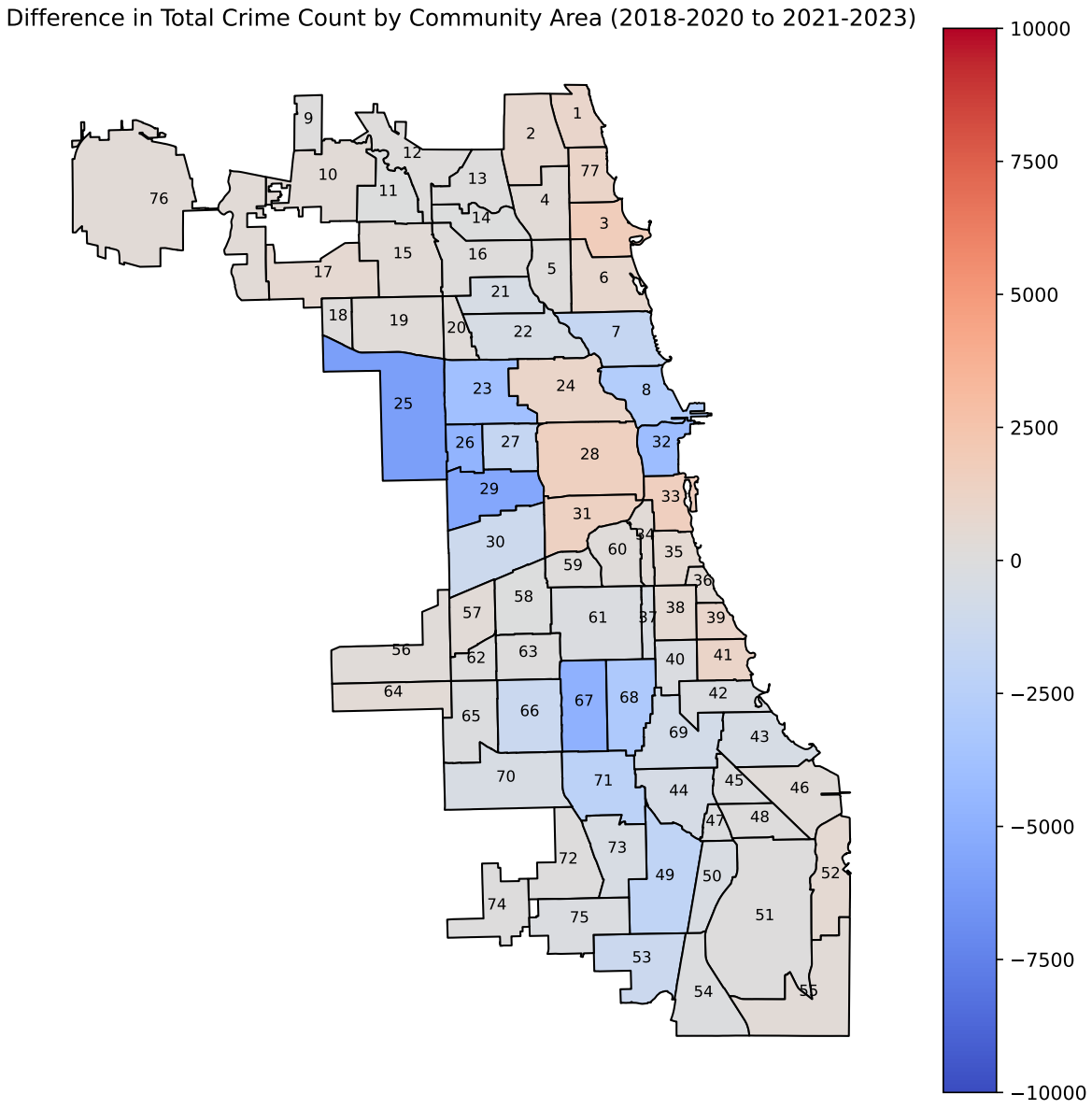
```
    ax=ax,
    vmin=0,
    vmax=40000
)

for idx, row in chicago_2021_2023_gdf.iterrows():
    if row["the_geom"].is_empty:
        continue
    centroid = row["the_geom"].centroid
    ax.annotate(
        text=str(row["AREA_NUM_1"]),
        xy=(centroid.x, centroid.y),
        horizontalalignment="center",
        fontsize=8,
        color="black"
    )

ax.axis("off")
for spine in ax.spines.values():
    spine.set_visible(False)

plt.title("Total Crime Count by Community Area in Chicago (2021 to 2023)")
plt.savefig("pictures/comm_crime_total_21_23.png", bbox_inches='tight', dpi=300)
plt.show()
plt.close()
```




```
diff_gdf = chicago_2021_2023_gdf[['AREA_NUM_1', 'crime_count', 'the_geom']].merge(
    chicago_2018_2020_gdf[['AREA_NUM_1', 'crime_count', 'the_geom']],
    on='AREA_NUM_1',
    suffixes=('_2021_2023', '_2018_2020')
)
diff_gdf['crime_diff'] = diff_gdf['crime_count_2021_2023'] - diff_gdf['crime_count_2018_2020
diff_gdf = diff_gdf.drop(columns=['the_geom_2018_2020'])
```

```
diff_gdf = diff_gdf.rename(columns={'the_geom_2021_2023':'the_geom'})
```

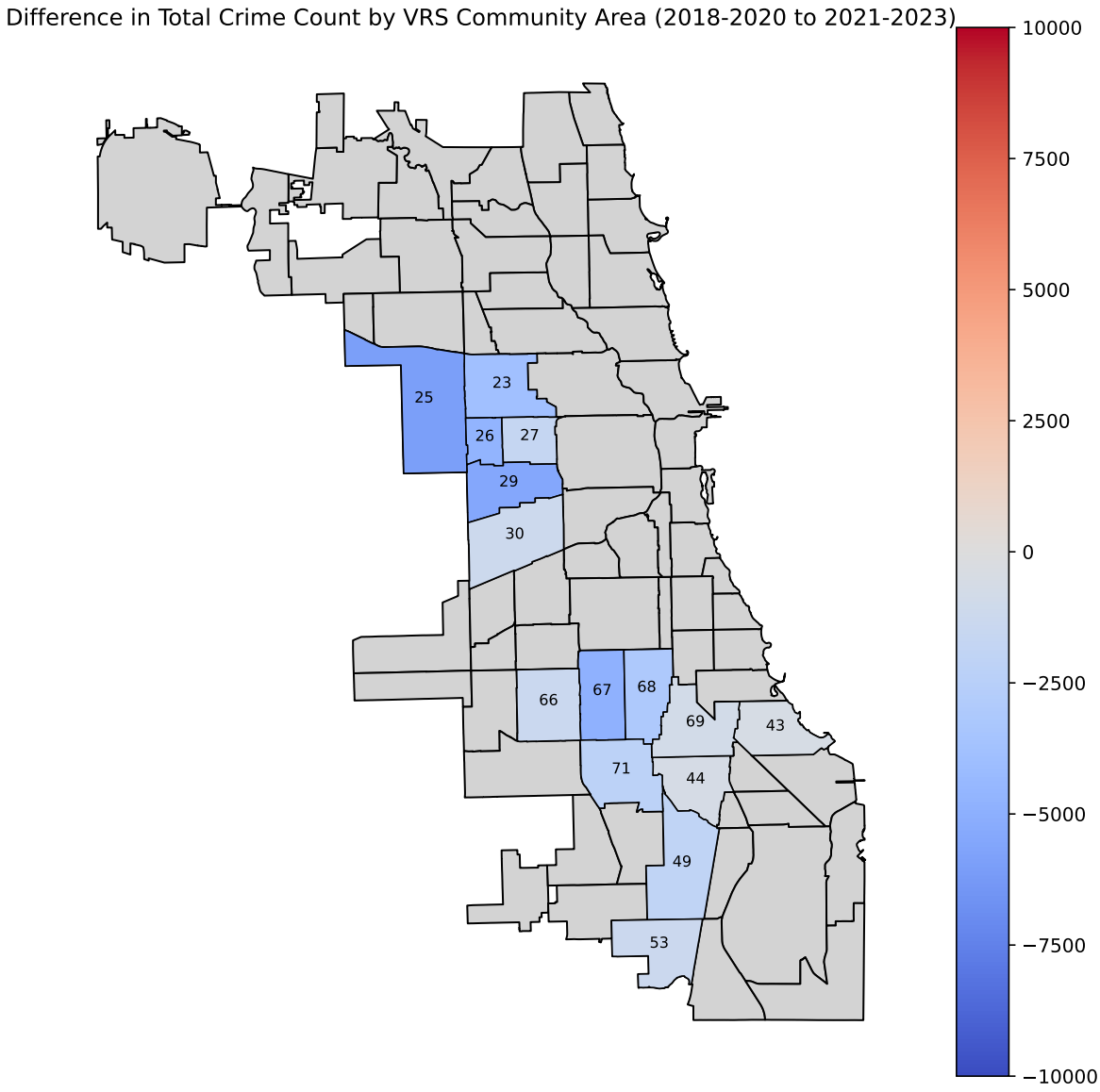
```
fig, ax = plt.subplots(figsize=(10, 10))

diff_gdf.plot(
    column="crime_diff",
    cmap="coolwarm",
    edgecolor="black",
    legend=True,
    ax=ax,
    vmin=-10000,
    vmax=10000
)
for idx, row in diff_gdf.iterrows():
    if row["the_geom"].is_empty:
        continue
    centroid = row["the_geom"].centroid
    ax.annotate(
        text=str(row["AREA_NUM_1"]),
        xy=(centroid.x, centroid.y),
        horizontalalignment="center",
        fontsize=8,
        color="black"
    )
ax.axis("off")
for spine in ax.spines.values():
    spine.set_visible(False)

plt.title("Difference in Total Crime Count by Community Area (2018-2020 to 2021-2023)")
plt.savefig("pictures/comm_crime_diff.png", bbox_inches='tight', dpi=300)
plt.show()
plt.close()
```




```
if isinstance(vrs_community, list):
    vrs_community_df = pd.DataFrame(vrs_community, columns=["COMMUNITY"])
else:
    vrs_community_df = vrs_community
comms = chicago[['AREA_NUM_1', 'COMMUNITY']]
comm_num = pd.merge(vrs_community_df, comms, on='COMMUNITY', how='inner')
vrs_diff_gdf = pd.merge(
    diff_gdf,
```

```
    comm_num[['AREA_NUM_1', 'COMMUNITY']],
    on='AREA_NUM_1',
    how='inner'
)
```

```
fig, ax = plt.subplots(figsize=(10, 10))

chicago_gdf.plot(ax=ax, color='lightgrey', edgecolor='black')

vrs_diff_gdf.plot(
    column="crime_diff",
    cmap="coolwarm",
    linewidth=0.8,
    edgecolor="black",
    legend=True,
    vmin=-10000,
    vmax=10000,
    ax=ax
)
for idx, row in vrs_diff_gdf.iterrows():
    if row["the_geom"].is_empty:
        continue
    centroid = row["the_geom"].centroid
    ax.annotate(
        text=str(row["AREA_NUM_1"]),
        xy=(centroid.x, centroid.y),
        horizontalalignment="center",
        fontsize=8,
        color="black"
    )
ax.axis("off")
for spine in ax.spines.values():
    spine.set_visible(False)

plt.title("Difference in Total Crime Count by VRS Community Area (2018-2020 to 2021-2023)")
plt.savefig("pictures/vrs_comm_crime_diff.png", bbox_inches='tight', dpi=300)
plt.show()
plt.close()
```




```
crimes_data = pd.read_csv(f'{base_path}/Crimes_-_2001_to_Present_20241127.csv')
print(crimes_data.columns)
```

```
Index(['ID', 'Case Number', 'Date', 'Block', 'IUCR', 'Primary Type',
       'Description', 'Location Description', 'Arrest', 'Domestic', 'Beat',
       'District', 'Ward', 'Community Area', 'FBI Code', 'X Coordinate',
       'Y Coordinate', 'Year', 'Updated On', 'Latitude', 'Longitude',
       'Location'],
      dtype='object')
```

```
crimes_data['Date'] = pd.to_datetime(crimes_data['Date'], errors = 'coerce')
```

```
C:\Users\Jakub\AppData\Local\Temp\ipykernel_14352\764977958.py:1: UserWarning: Could not infe
  crimes_data['Date'] = pd.to_datetime(crimes_data['Date'], errors = 'coerce')
```

```
# unique_descriptions = crimes_data['Primary Type'].unique()
# print(unique_descriptions)
```

Replace CRIM SEXUAL ASSAULT with CRIMINIAL SEXUAL ASSAULT

```
crimes_data['Primary Type'] = crimes_data['Primary Type'].replace(
    'CRIM SEXUAL ASSAULT', 'CRIMINAL SEXUAL ASSAULT'
)
```

```
# min_date = crimes_data['Date'].min()
# max_date = crimes_data['Date'].max()

# print(f"Minimum date: {min_date}")
# print(f"Maximum date: {max_date}")
```

```
# excel_file = "Crimes_-_2001_to_Present_20241127 (1).xlsx"
# max_rows = 1048575

# # Check if `crimes_data` is defined
# if 'crimes_data' in locals() or 'crimes_data' in globals():
#     # Split the DataFrame and save to multiple sheets
#     with pd.ExcelWriter(excel_file, engine='openpyxl') as writer:
#         for start_row in range(0, crimes_data.shape[0], max_rows):
#             # Extract a chunk of the DataFrame
#             end_row = min(start_row + max_rows, crimes_data.shape[0])
#             chunk = crimes_data.iloc[start_row:end_row]

#             # Write the chunk to a separate sheet
#             sheet_name = f"Sheet_{start_row // max_rows + 1}"
#             chunk.to_excel(writer, sheet_name=sheet_name, index=False)

#     print(f"Data has been saved to {excel_file} in multiple sheets.")
# else:
#     print("The DataFrame `crimes_data` is not defined. Please define it first.")
```

Added a new column to our crimes_data called Community Area Name by merging chicago-community-areas.csv with our orginal dataset

```
community_areas = pd.read_csv('data/chicago-community-areas.csv')

# Transpose the DataFrame
community_areas = community_areas.T

community_areas = community_areas.reset_index()

# Set the first row as column names
community_areas.columns = community_areas.iloc[0]

# Drop the first row as it is now the header
community_areas = community_areas[1:]

# Debug: Print the current column names
print("Columns after setting headers:", community_areas.columns)

# Select only the `Community Area` and `name` columns
if 'Community Area' in community_areas.columns and 'name' in community_areas.columns:
    community_areas = community_areas[['Community Area', 'name']]
else:
    print("Required columns are missing: 'Community Area' and 'name'")
    print("Available columns:", community_areas.columns)

# Rename columns for clarity
community_areas.rename(columns = {'Community Area': 'Community Area Number', 'name': 'Commun

# Convert "Community Area Number" to numeric
community_areas['Community Area Number'] = pd.to_numeric(community_areas['Community Area Numb

# Reset index
community_areas.reset_index(drop = True, inplace = True)

print("Processed community_areas DataFrame:")
print(community_areas.head())
```

```
Columns after setting headers: Index(['Community Area', 'name', 'population', 'income', 'requ
       'blacks', 'white', 'asian', 'other'],
      dtype='object', name=0)
Processed community_areas DataFrame:
```

```
0  Community Area Number Community Area Name
0                      1         Rogers Park
1                      2          West Ridge
2                      3              Uptown
3                      4      Lincoln Square
4                      5        North Center
```

```python
# Merge the datasets, ensuring the original column name 'Community Area' is retained
crimes_data = crimes_data.merge(
    community_areas,
    left_on = 'Community Area',         # Column in crimes_data
    right_on ='Community Area Number',  # Column in community_areas
    how = 'left'                        # Preserve all rows in crimes_data
)

# Drop the redundant 'Community Area Number' column from the merge
if 'Community Area Number' in crimes_data.columns:
    crimes_data.drop(columns = ['Community Area Number'], inplace = True)

# Debug: Verify columns after the merge
print("Columns in crimes_data after merge:", crimes_data.columns)

# Check if 'Community Area Name' exists
if 'Community Area Name' not in crimes_data.columns:
    print("Error: 'Community Area Name' column not found after merge.")
else:
    # Check for missing values in the 'Community Area Name' column
    missing_values = crimes_data['Community Area Name'].isna().sum()
    print(f"Number of missing values in 'Community Area Name': {missing_values}")

    # Exclude rows with missing 'Community Area Name'
    crimes_data = crimes_data.dropna(subset = ['Community Area Name'])

    # Verify the remaining dataset
    print(f"Number of rows after excluding missing values: {len(crimes_data)}")
    print("Sample data:")
    print(crimes_data[['Community Area', 'Community Area Name']].head())
```

```
Columns in crimes_data after merge: Index(['ID', 'Case Number', 'Date', 'Block', 'IUCR', 'Pri
       'Description', 'Location Description', 'Arrest', 'Domestic', 'Beat',
       'District', 'Ward', 'Community Area', 'FBI Code', 'X Coordinate',
       'Y Coordinate', 'Year', 'Updated On', 'Latitude', 'Longitude',
```

```
       'Location', 'Community Area Name'],
      dtype='object')
Number of missing values in 'Community Area Name': 4
Number of rows after excluding missing values: 2487958
Sample data:
   Community Area Community Area Name
0            35.0             Douglas
1            71.0      Auburn Gresham
2            14.0         Albany Park
3            40.0     Washington Park
4            71.0      Auburn Gresham
```

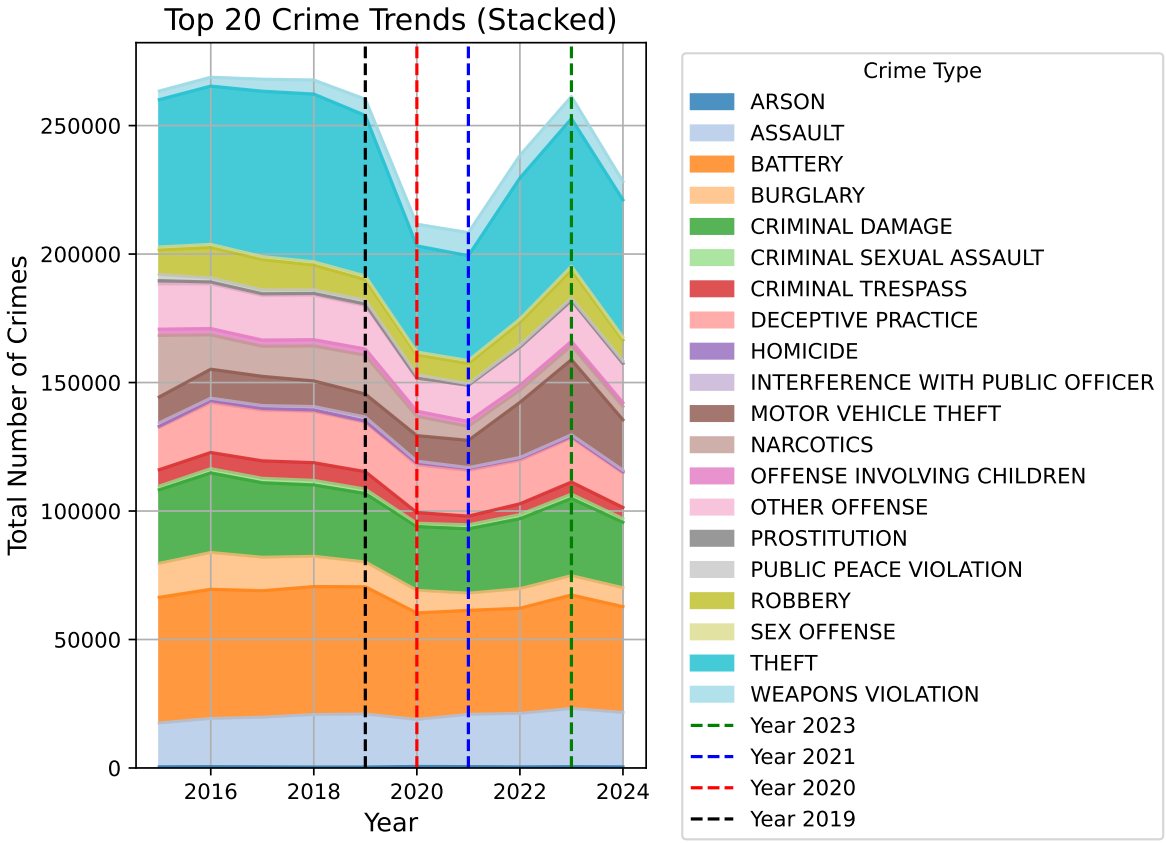
Part A: Crimes grouped by year and primary type

```
#GROUPING CRIMES BY YEAR AND PRIMARY TYPE

# Ensure the 'Date' column is in datetime format
crimes_data['Date'] = pd.to_datetime(crimes_data['Date'], errors = 'coerce')

# Create a copy of the original DataFrame to work on
crimes_grouping = crimes_data.copy()

# Extract the year from the 'Date' column
crimes_grouping['Year'] = crimes_grouping['Date'].dt.year

# Group by 'Year' and 'Primary Type' and count the number of occurrences
grouped_crimes_data = crimes_grouping.groupby(['Year', 'Primary Type']).size().reset_index(na

# View the grouped data
print(grouped_crimes_data.head())
```

```
   Year                       Primary Type  Count
0  2015                             ARSON    453
1  2015                           ASSAULT  17048
2  2015                           BATTERY  48921
3  2015                          BURGLARY  13184
4  2015  CONCEALED CARRY LICENSE VIOLATION     34
```

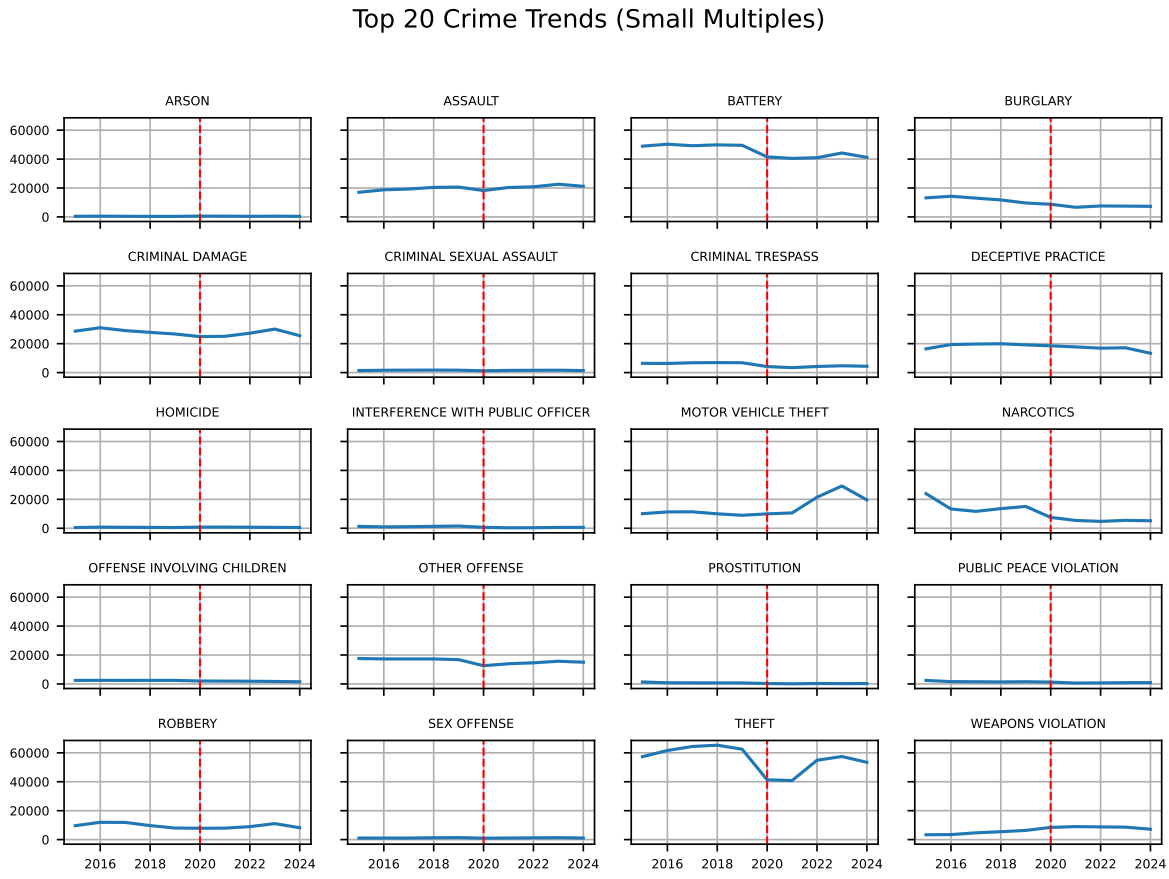
```
# CRIME TRENDS BY TOP 20 TYPES (STACKED)

# Calculate the total count for each crime type
```

```
total_crimes_by_type = grouped_crimes_data.groupby('Primary Type')[
    'Count'].sum()

# Select the top 20 crime types based on total counts
top_20_crime_types = total_crimes_by_type.nlargest(20).index

# Filter the grouped data to include only the top 20 crime types
filtered_grouped_data = grouped_crimes_data[grouped_crimes_data['Primary Type'].isin(
    top_20_crime_types)]

# Pivot the filtered data for stacked area chart
pivot_data = filtered_grouped_data.pivot(
    index='Year', columns='Primary Type', values='Count').fillna(0)

# plot stacked area chart
plt.figure(figsize=(8, 6))
pivot_data.plot.area(ax=plt.gca(), cmap='tab20', alpha=0.8)

# Added vertical lines at the year 2019, 2020, 2021
plt.axvline(x=2023, color='green', linestyle='--',
            linewidth=1.5, label='Year 2023')
plt.axvline(x=2021, color='blue', linestyle='--',
            linewidth=1.5, label='Year 2021')
plt.axvline(x=2020, color='red', linestyle='--',
            linewidth=1.5, label='Year 2020')
plt.axvline(x=2019, color='black', linestyle='--',
            linewidth=1.5, label='Year 2019')

plt.xlabel('Year', fontsize=12)
plt.ylabel('Total Number of Crimes', fontsize=12)
plt.title('Top 20 Crime Trends (Stacked)', fontsize=14)
plt.legend(title='Crime Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)

plt.tight_layout()
plt.show()
```




```
# Plot small multiple plots
fig, axes = plt.subplots(nrows = 5, ncols = 4, figsize = (8, 6), sharex = True, sharey = True
axes = axes.flatten()

for i, crime_type in enumerate(pivot_data.columns[:20]):  # only show top 20
    # Plot the crime trend
    axes[i].plot(pivot_data.index, pivot_data[crime_type], label = crime_type)

    # Add a vertical line at 2020
    axes[i].axvline(x = 2020, color = 'red', linestyle = '--', linewidth = 1, label = 'Year 2

    # Set title and tick label size
    axes[i].set_title(crime_type, fontsize = 6)
    axes[i].tick_params(axis = 'both', which = 'major', labelsize = 6)

    # Add grid
    axes[i].grid(True)
```

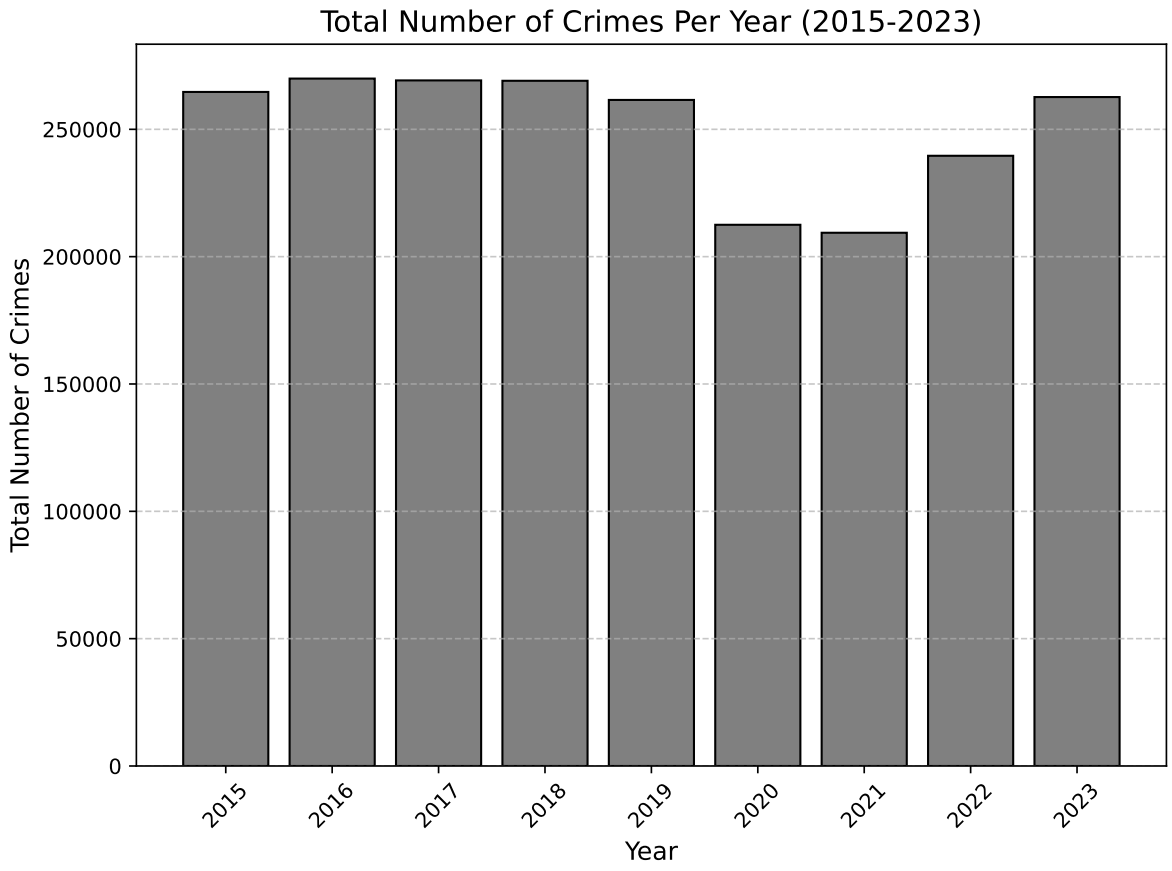
```
# Set overall plot details
plt.suptitle('Top 20 Crime Trends (Small Multiples)', fontsize = 12)
plt.tight_layout(rect = [0, 0, 1, 0.96])
plt.show()
```



Part B: Total number of crimes per year

```
# Ensure the 'Date' column is in datetime format
crimes_data['Date'] = pd.to_datetime(crimes_data['Date'], errors = 'coerce')

# Extract the year from the 'Date' column
crimes_data['Year'] = crimes_data['Date'].dt.year

# Group data by 'Year' and count total crimes
total_crimes_per_year = crimes_data.groupby('Year').size().reset_index(name='Total Crimes')

# Filter for years 2015 to 2023
```

```
filtered_crimes = total_crimes_per_year[(total_crimes_per_year['Year'] >= 2015) & (total_cri

# Plot the bar graph
plt.figure(figsize = (8, 6))
plt.bar(filtered_crimes['Year'], filtered_crimes['Total Crimes'], color = 'grey', edgecolor =

# format the plot
plt.xlabel('Year', fontsize = 12)
plt.ylabel('Total Number of Crimes', fontsize = 12)
plt.title('Total Number of Crimes Per Year (2015-2023)', fontsize = 14)
plt.xticks(filtered_crimes['Year'], rotation = 45)  # Rotate x-axis labels if needed
plt.grid(axis = 'y', linestyle = '--', alpha = 0.7)

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



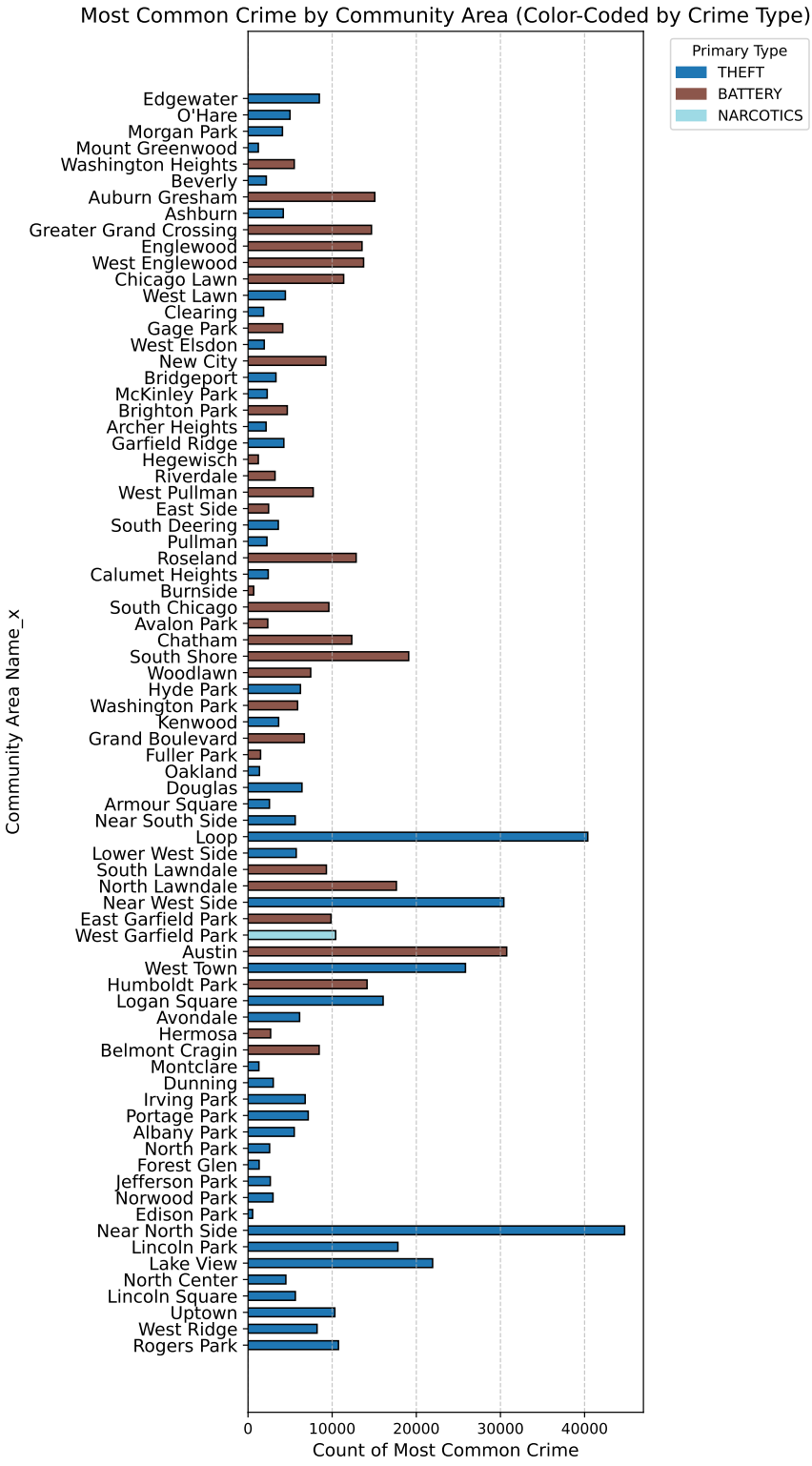
Part C: Most common crimes across community areas

```
# Group by 'Community Area', 'Community Area Name', and 'Primary Type' and count occurrences
crimes_by_area = crimes_data.groupby(['Community Area', 'Community Area Name', 'Primary Type

# Sort by 'Community Area', 'Count' in descending order
crimes_by_area_sorted = crimes_by_area.sort_values(['Community Area', 'Count'], ascending =

# Get the most common crime type for each community area
most_common_crimes = crimes_by_area_sorted.groupby('Community Area').first().reset_index()

print(most_common_crimes)
```

```
    Community Area Community Area Name Primary Type  Count
0              1.0         Rogers Park        THEFT  10749
1              2.0          West Ridge        THEFT   8207
2              3.0              Uptown        THEFT  10320
3              4.0      Lincoln Square        THEFT   5624
4              5.0        North Center        THEFT   4495
..             ...                 ...          ...    ...
72            73.0  Washington Heights      BATTERY   5494
73            74.0     Mount Greenwood        THEFT   1224
74            75.0         Morgan Park        THEFT   4092
75            76.0              O'Hare        THEFT   4992
76            77.0           Edgewater        THEFT   8474

[77 rows x 4 columns]
```

```
# Generate the y positions with spacing
y_positions = np.arange(len(most_common_crimes)) * 1.5  # Add extra spacing by multiplying po

# Assign a unique color for each Primary Type
crime_types = most_common_crimes['Primary Type'].unique()
colors = plt.cm.tab20(np.linspace(0, 1, len(crime_types)))
color_mapping = {crime: colors[i] for i, crime in enumerate(crime_types)}

# Map colors to the bars
bar_colors = most_common_crimes['Primary Type'].map(color_mapping)

# Plot the horizontal bar chart
plt.figure(figsize = (8, 14))
bars = plt.barh(
```

```
    y_positions,
    most_common_crimes['Count'],
    color = bar_colors,
    edgecolor = 'black'
)

# Set the y-ticks to match the new positions with community area names
plt.yticks(y_positions, most_common_crimes['Community Area Name'], fontsize = 12)

# Add a legend for the crime types
handles = [plt.Rectangle((0, 0), 1, 1, color = color_mapping[crime]) for crime in crime_types
plt.legend(handles, crime_types, title = 'Primary Type', bbox_to_anchor = (1.05, 1), loc = '

# Add labels and title
plt.xlabel('Count of Most Common Crime', fontsize = 12)
plt.ylabel('Community Area Name_x', fontsize = 12)
plt.title('Most Common Crime by Community Area (Color-Coded by Crime Type)', fontsize = 14)

# Add grid lines
plt.grid(axis = 'x', linestyle = '--', alpha = 0.7)

plt.tight_layout()
plt.show()
```


Most Common Crime by Community Area (Color-Coded by Crime Type)
Primary Type
THEFT
BATTERY
NARCOTICS
Community Area Name_x
Edgewater
O'Hare
Morgan Park
Mount Greenwood
Washington Heights
Beverly
Auburn Gresham
Ashburn
Greater Grand Crossing
Englewood
West Englewood
Chicago Lawn
West Lawn
Clearing
Gage Park
West Elsdon
New City
Bridgeport
McKinley Park
Brighton Park
Archer Heights
Garfield Ridge
Hegewisch
Riverdale
West Pullman
East Side
South Deering
Pullman
Roseland
Calumet Heights
Burnside
South Chicago
Avalon Park
Chatham
South Shore
Woodlawn
Hyde Park
Washington Park
Kenwood
Grand Boulevard
Fuller Park
Oakland
Douglas
Armour Square
Near South Side
Loop
Lower West Side
South Lawndale
North Lawndale
Near West Side
East Garfield Park
West Garfield Park
Austin
West Town
Humboldt Park
Logan Square
Avondale
Hermosa
Belmont Cragin
Montclare
Dunning
Irving Park
Portage Park
Albany Park
North Park
Forest Glen
Jefferson Park
Norwood Park
Edison Park
Near North Side
Lincoln Park
Lake View
North Center
Lincoln Square
Uptown
West Ridge
Rogers Park
0
10000
20000
30000
40000
Count of Most Common Crime

I want to know the 3 top crime type across community areas and then assess the top 5 crime types across all community areas

First: get top 3 crimes per community area

```
# Group by 'Community Area', 'Community Area Name', and 'Primary Type', and count occurrences
crimes_by_area = crimes_data.groupby(['Community Area', 'Community Area Name', 'Primary Type

# Sort by 'Community Area' and 'Count' in descending order
crimes_by_area_sorted = crimes_by_area.sort_values(['Community Area', 'Count'], ascending =

# Get the top 3 crimes for each community area
top_3_crimes_by_area = crimes_by_area_sorted.groupby('Community Area').head(3)

print(top_3_crimes_by_area.head(10))
```

	Community Area	Community Area Name	Primary Type	Count
29	1.0	Rogers Park	THEFT	10749
2	1.0	Rogers Park	BATTERY	7179
5	1.0	Rogers Park	CRIMINAL DAMAGE	4639
58	2.0	West Ridge	THEFT	8207
33	2.0	West Ridge	BATTERY	5798
36	2.0	West Ridge	CRIMINAL DAMAGE	4411
87	3.0	Uptown	THEFT	10320
62	3.0	Uptown	BATTERY	6722
68	3.0	Uptown	DECEPTIVE PRACTICE	3772
117	4.0	Lincoln Square	THEFT	5624

Second: count common crimes across community areas

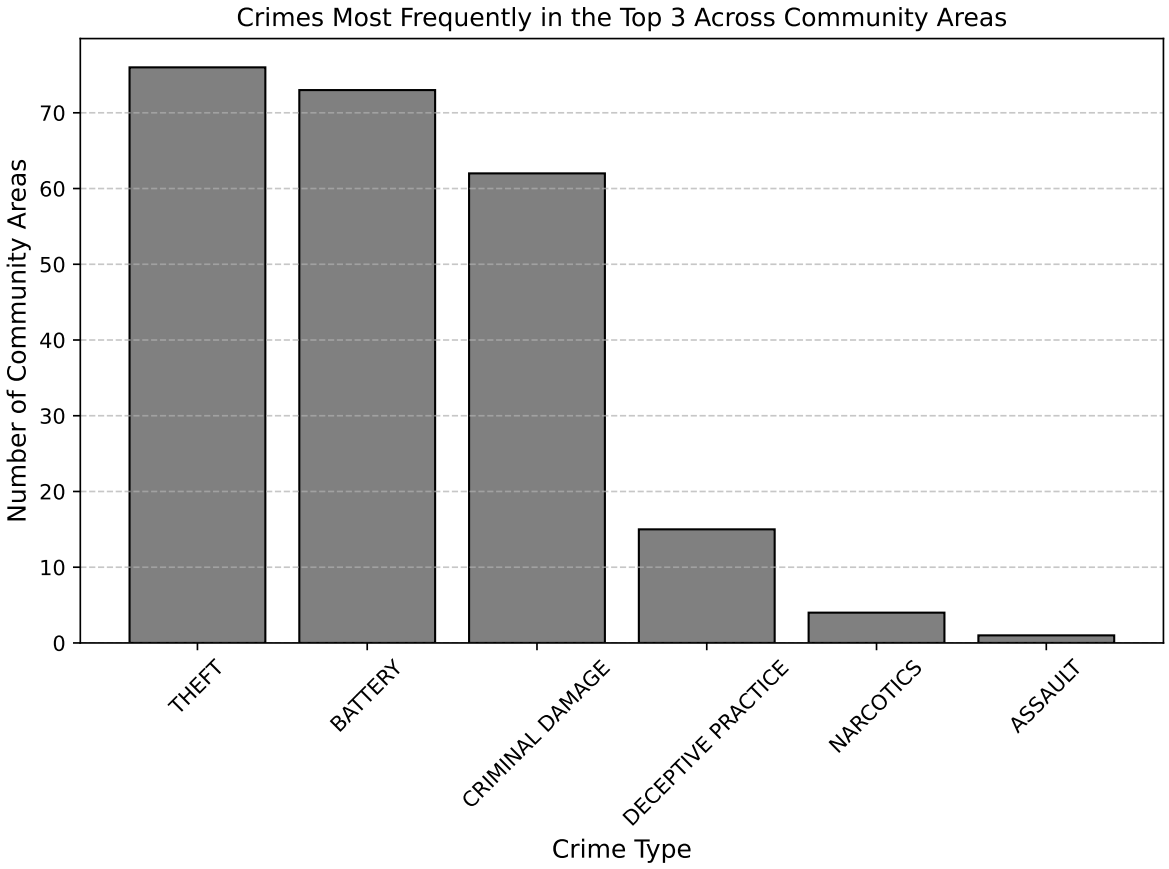
```
# Count the number of community areas in which each crime appears in the top 3
crime_counts = top_3_crimes_by_area['Primary Type'].value_counts().reset_index()
crime_counts.columns = ['Primary Type', 'Community Areas Count']

# Sort by the number of community areas
crime_counts_sorted = crime_counts.sort_values('Community Areas Count', ascending = False)

print(crime_counts_sorted.head(10))
```

	Primary Type	Community Areas Count
0	THEFT	76
1	BATTERY	73

```
2     CRIMINAL DAMAGE                    62
3  DECEPTIVE PRACTICE                    15
4           NARCOTICS                     4
5             ASSAULT                     1
```

Third: show the 5 most common crimes across community area:

```
# Plot the crimes that appear most in community areas' top 3
plt.figure(figsize = (8, 6))
plt.bar(crime_counts_sorted['Primary Type'], crime_counts_sorted['Community Areas Count'], co

# Add labels and title
plt.xlabel('Crime Type', fontsize = 12)
plt.ylabel('Number of Community Areas', fontsize = 12)
plt.title('Crimes Most Frequently in the Top 3 Across Community Areas', fontsize = 12)
plt.xticks(rotation = 45, fontsize = 10)
plt.grid(axis = 'y', linestyle = '--', alpha = 0.7)

plt.tight_layout()
plt.show()
```



Part C': Plot showing the timings with the highest crime rates in every community area

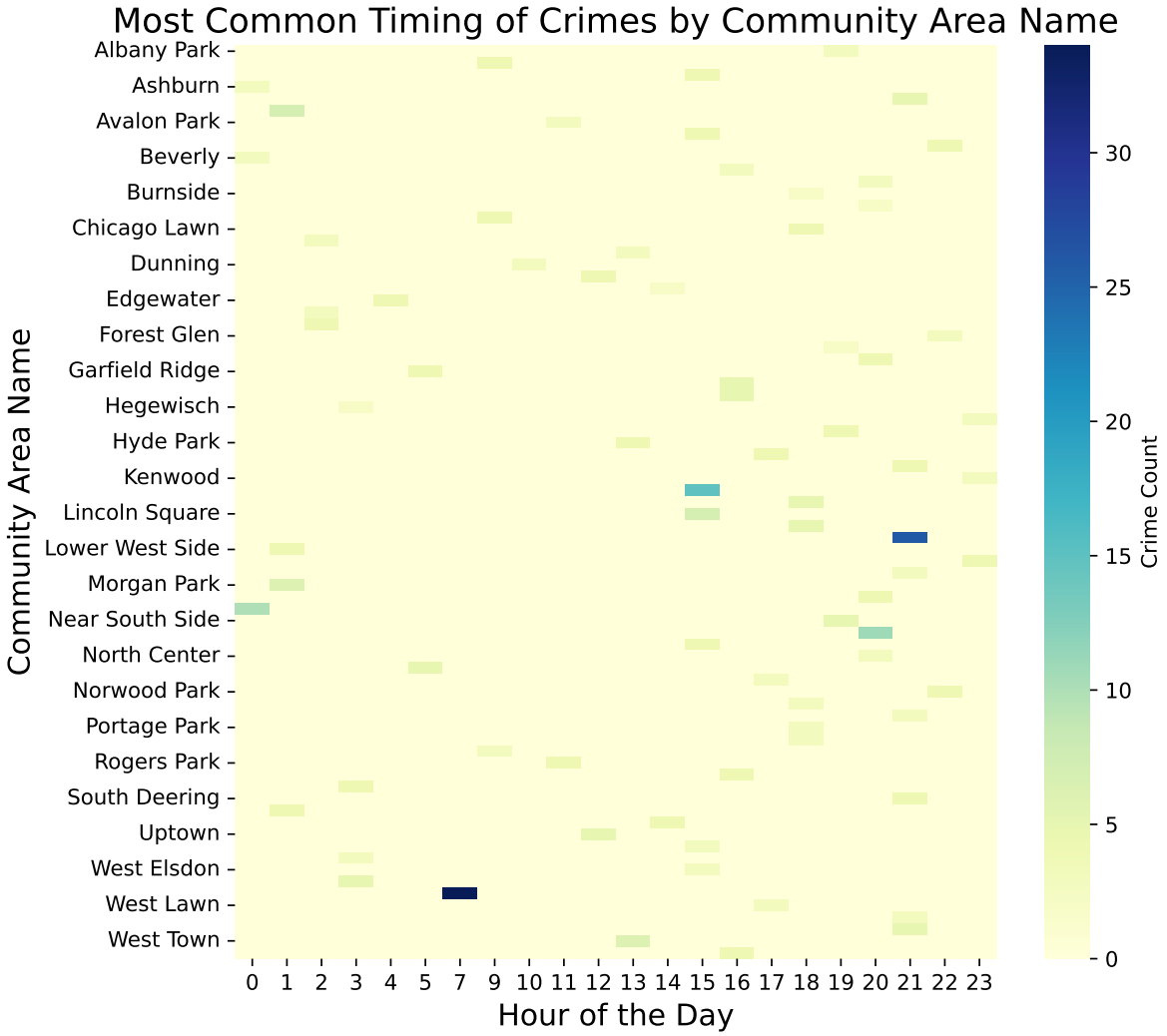
```
import seaborn as sns

# Create a copy of the original dataset
crimes_data_copy = crimes_data.copy()

# Merge the most common crimes back with the original dataset copy
most_common_crimes_with_timing = most_common_crimes.merge(
    crimes_data_copy,
    on = ['Community Area Name', 'Primary Type'],
    how = 'left'
)

# Ensure the 'Date' column is in datetime format
most_common_crimes_with_timing['Date'] = pd.to_datetime(most_common_crimes_with_timing['Date
```

```
# Extract the day, weekday, and hour from the Date column
most_common_crimes_with_timing['Day'] = most_common_crimes_with_timing['Date'].dt.date
most_common_crimes_with_timing['Weekday'] = most_common_crimes_with_timing['Date'].dt.day_nar
most_common_crimes_with_timing['Hour'] = most_common_crimes_with_timing['Date'].dt.hour

# Group by 'Community Area Name', 'Primary Type', 'Day', 'Weekday', and 'Hour' to find the mo
timing_analysis = most_common_crimes_with_timing.groupby(
    ['Community Area Name', 'Primary Type', 'Day', 'Weekday', 'Hour']
).size().reset_index(name = 'Count')

# Find the most common timing for each community area name
most_common_timing = timing_analysis.loc[
    timing_analysis.groupby(['Community Area Name', 'Primary Type'])['Count'].idxmax()
]

# View the results (Table of Most Common Timing)
print(most_common_timing[['Community Area Name', 'Primary Type', 'Day', 'Weekday', 'Hour', 'C

# Pivot data for heatmap visualization (Hour vs. Community Area Name)
heatmap_data = most_common_timing.pivot_table(
    index = 'Community Area Name', columns = 'Hour', values = 'Count', aggfunc = 'sum', fill_
)

# Plot the heatmap
plt.figure(figsize = (8, 7))
sns.heatmap(heatmap_data, cmap = 'YlGnBu', annot = False, cbar_kws = {'label': 'Crime Count'}
plt.title('Most Common Timing of Crimes by Community Area Name', fontsize = 16)
plt.xlabel('Hour of the Day', fontsize = 14)
plt.ylabel('Community Area Name', fontsize = 14)
plt.tight_layout()
plt.show()

# Pivot data for heatmap visualization (Weekday vs. Community Area Name)
heatmap_weekday_data = most_common_timing.pivot_table(
    index = 'Community Area Name', columns = 'Weekday', values = 'Count', aggfunc = 'sum', fi
)
```

	Community Area Name	Primary Type	Day	Weekday	Hour	Count
415	Albany Park	THEFT	2015-10-06	Tuesday	19	3
5374	Archer Heights	THEFT	2015-09-01	Tuesday	9	4
8352	Armour Square	THEFT	2019-01-12	Saturday	15	4
10104	Ashburn	THEFT	2015-10-06	Tuesday	0	3

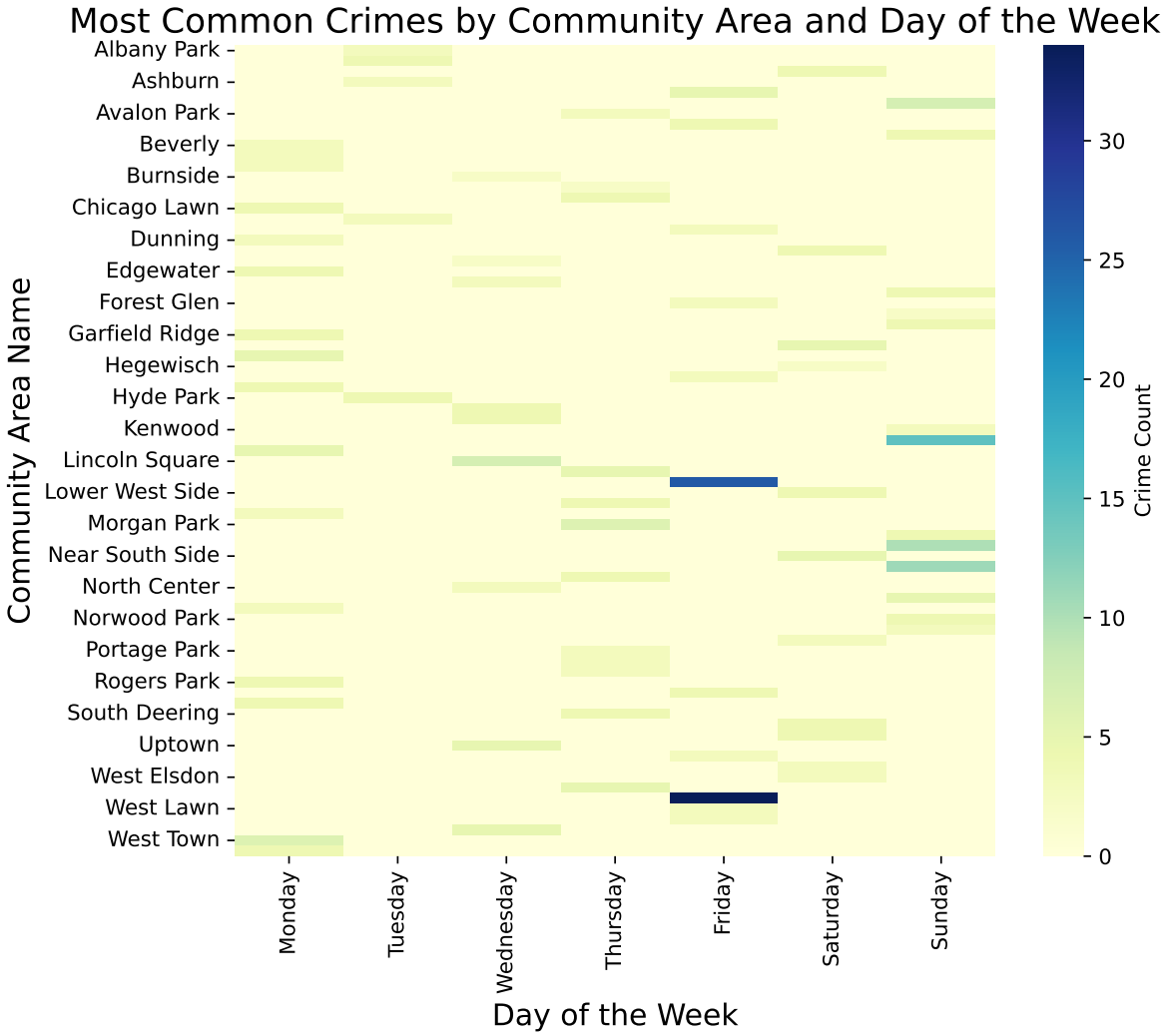
19532	Auburn Gresham	BATTERY	2018-10-26	Friday	21	5
...	...	...	...	...	...	...
520251	West Lawn	THEFT	2015-03-20	Friday	17	3
524685	West Pullman	BATTERY	2015-05-29	Friday	21	3
535513	West Ridge	THEFT	2020-01-29	Wednesday	21	5
550805	West Town	THEFT	2019-12-02	Monday	13	6
566853	Woodlawn	BATTERY	2024-02-05	Monday	16	4

[77 rows x 6 columns]



Plot showing the week days with the highest crime rates in every community area

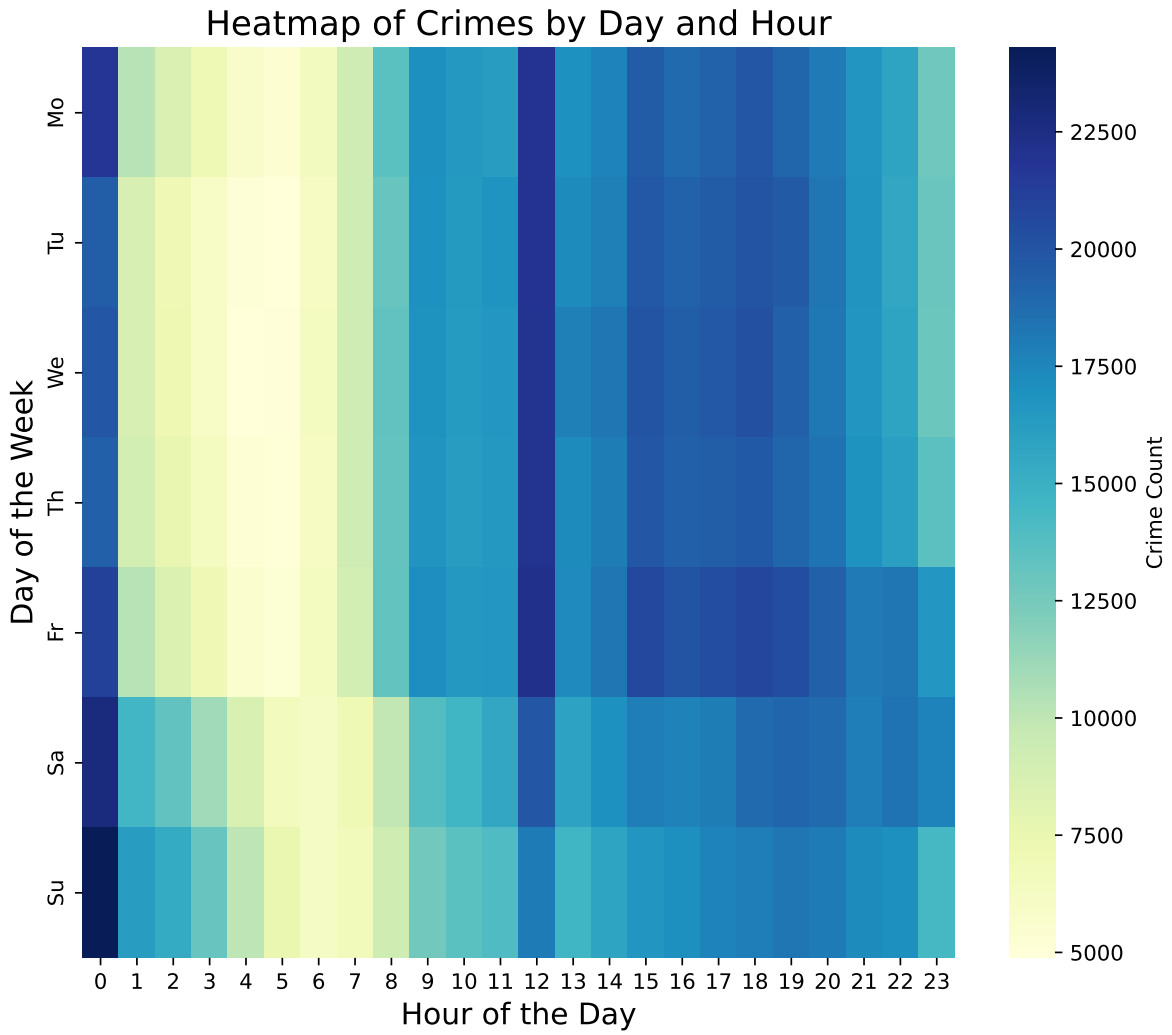
```
# Ensure weekdays are in the correct order
weekday_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday
heatmap_weekday_data = heatmap_weekday_data[weekday_order]

# Plot the weekday heatmap
plt.figure(figsize = (8, 7))
sns.heatmap(heatmap_weekday_data, cmap = 'YlGnBu', annot = False, cbar_kws = {'label': 'Crime
plt.title('Most Common Crimes by Community Area and Day of the Week', fontsize = 16)
plt.xlabel('Day of the Week', fontsize = 14)
plt.ylabel('Community Area Name', fontsize = 14)
plt.tight_layout()
plt.show()
```



Part D: Map showing the most common crime type across community areas

Part E: common timings and dates for crimes

```
# Ensure the 'Date' column is in datetime format
crimes_data['Date'] = pd.to_datetime(crimes_data['Date'], format = '%m/%d/%Y %I:%M:%S %p', er

# Extract day of the week (abbreviated) and hour
crimes_data['Day'] = crimes_data['Date'].dt.day_name().str[:2]  # Extract abbreviations (Mo,
crimes_data['Hour'] = crimes_data['Date'].dt.hour

# Group by 'Day' and 'Hour', and count occurrences
crimes_by_day_hour = crimes_data.groupby(['Day', 'Hour']).size().reset_index(name = 'Count')

# Pivot the data for a heatmap
heatmap_data = crimes_by_day_hour.pivot_table(
    index = 'Day', columns = 'Hour', values = 'Count', aggfunc = 'sum', fill_value = 0
)

# Reorder the days of the week (Mo, Tu, We, Th, Fr, Sa, Su)
days_order = ['Mo', 'Tu', 'We', 'Th', 'Fr', 'Sa', 'Su']
heatmap_data = heatmap_data.reindex(days_order)

# Plot the heatmap
plt.figure(figsize = (8, 7))
sns.heatmap(heatmap_data, cmap = 'YlGnBu', annot = False, cbar_kws = {'label': 'Crime Count'}
plt.title('Heatmap of Crimes by Day and Hour', fontsize = 16)
plt.xlabel('Hour of the Day', fontsize = 14)
plt.ylabel('Day of the Week', fontsize = 14)
plt.tight_layout()
plt.show()
```



Part F: 15 community areas before/after 2018-2020 vs 2021-2023

```
violence_reduction_data = pd.read_csv('data/Violence_Reduction_-_Victims_of_Homicides_and_Nor
print(violence_reduction_data.columns)
```

```
Index(['CASE_NUMBER', 'DATE', 'BLOCK', 'VICTIMIZATION_PRIMARY',
       'INCIDENT_PRIMARY', 'GUNSHOT_INJURY_I', 'UNIQUE_ID', 'ZIP_CODE', 'WARD',
       'COMMUNITY_AREA', 'STREET_OUTREACH_ORGANIZATION', 'AREA', 'DISTRICT',
       'BEAT', 'AGE', 'SEX', 'RACE', 'VICTIMIZATION_FBI_CD', 'INCIDENT_FBI_CD',
       'VICTIMIZATION_FBI_DESCR', 'INCIDENT_FBI_DESCR',
       'VICTIMIZATION_IUCR_CD', 'INCIDENT_IUCR_CD',
       'VICTIMIZATION_IUCR_SECONDARY', 'INCIDENT_IUCR_SECONDARY',
```

```
       'HOMICIDE_VICTIM_FIRST_NAME', 'HOMICIDE_VICTIM_MI',
       'HOMICIDE_VICTIM_LAST_NAME', 'MONTH', 'DAY_OF_WEEK', 'HOUR',
       'LOCATION_DESCRIPTION', 'STATE_HOUSE_DISTRICT', 'STATE_SENATE_DISTRICT',
       'UPDATED', 'LATITUDE', 'LONGITUDE', 'LOCATION'],
      dtype='object')
```

```
# unique_descriptions2 = violence_reduction_data['VICTIMIZATION_PRIMARY'].unique()
# print(unique_descriptions2)
```

According to the One Year Assess, OCOS Report we extracted the below 15 community areas (page 5/40). The OCOS violence reduction plan aims to prioritize violence reduction efforts in the 15 community areas most affected by violence over the three-year period 2018-20 ("priority community areas"), as defined by the City as "serious victimizations" (homicides and non-fatal shootings).

```
# Define priority areas by referring to the report
priority_areas = [
    "Auburn Gresham", "Austin", "Chatham", "Chicago Lawn", "East Garfield Park",
    "Englewood", "Greater Grand Crossing", "Humboldt Park", "North Lawndale",
    "Roseland", "South Lawndale", "South Shore", "West Englewood",
    "West Garfield Park", "West Pullman"
]

# Standardize COMMUNITY_AREA values
violence_reduction_data['COMMUNITY_AREA'] = violence_reduction_data['COMMUNITY_AREA'].str.ti

# Ensure DATE is converted to datetime
violence_reduction_data['DATE'] = pd.to_datetime(violence_reduction_data['DATE'], errors = '

# Remove rows with missing COMMUNITY_AREA
violence_reduction_data = violence_reduction_data.dropna(subset = ['COMMUNITY_AREA'])

# Filter for priority community areas only
data_filtered_all = violence_reduction_data[
    violence_reduction_data['COMMUNITY_AREA'].isin(priority_areas)
]

# Extract and categorize years into periods
data_filtered_all['Year'] = data_filtered_all['DATE'].dt.year
data_filtered_all['Period'] = data_filtered_all['Year'].apply(
    lambda x: '2018-2020' if 2018 <= x <= 2020 else '2021-2023' if 2021 <= x <= 2023 else No
)
```

```
# Remove rows outside the desired periods
data_filtered_all = data_filtered_all[data_filtered_all['Period'].notna()]

# Group and count incidents by community area and period
summary_all = data_filtered_all.groupby(['COMMUNITY_AREA', 'Period']).size().reset_index(name

# Bar chart before vs after for all values in VICTIMIZATION_PRIMARY
import altair as alt

chart_all = alt.Chart(summary_all).mark_bar().encode(
    x = alt.X('COMMUNITY_AREA:N', sort = priority_areas, title = 'Priority Community Area'),
    y = alt.Y('Count:Q', title = 'Number of Incidents'),
    color = 'Period:N',
    column = alt.Column('Period:N', title = 'Period')
).properties(
    width = 300,
    height = 400,
    title = "Homicides and Non-Fatal Shootings Across Priority Community Areas (2018-2020 vs
)

chart_all.show()
```

```
C:\Users\Jakub\AppData\Local\Temp\ipykernel_14352\2216506216.py:13: UserWarning: Could not in
  violence_reduction_data['DATE'] = pd.to_datetime(violence_reduction_data['DATE'], errors =
C:\Users\Jakub\AppData\Local\Temp\ipykernel_14352\2216506216.py:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
  data_filtered_all['Year'] = data_filtered_all['DATE'].dt.year
C:\Users\Jakub\AppData\Local\Temp\ipykernel_14352\2216506216.py:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
  data_filtered_all['Period'] = data_filtered_all['Year'].apply(


alt.Chart(...)
```

```
# Combine remaining community areas into a single dataset
priority_areas_set = set(priority_areas)
all_areas = set(violence_reduction_data['COMMUNITY_AREA'].unique())
non_priority_areas = list(all_areas - priority_areas_set)

# Filter data for non-priority community areas (no filtering for specific VICTIMIZATION_PRIMA
data_filtered_non_priority_all = violence_reduction_data[
    violence_reduction_data['COMMUNITY_AREA'].isin(non_priority_areas)
]

# Extract and categorize years into periods
data_filtered_non_priority_all['Year'] = data_filtered_non_priority_all['DATE'].dt.year
data_filtered_non_priority_all['Period'] = data_filtered_non_priority_all['Year'].apply(
    lambda x: '2018-2020' if 2018 <= x <= 2020 else '2021-2023' if 2021 <= x <= 2023 else Nor
)

# Remove rows outside the desired periods
data_filtered_non_priority_all = data_filtered_non_priority_all[data_filtered_non_priority_al

# Group and count incidents by community area and period
summary_non_priority_all = data_filtered_non_priority_all.groupby(['COMMUNITY_AREA', 'Period

# Sort community areas alphabetically for consistency
summary_non_priority_all['COMMUNITY_AREA'] = summary_non_priority_all['COMMUNITY_AREA'].astyp
summary_non_priority_all['COMMUNITY_AREA'] = pd.Categorical(
    summary_non_priority_all['COMMUNITY_AREA'],
    categories = sorted(summary_non_priority_all['COMMUNITY_AREA'].unique()),
    ordered = True
)

chart_non_priority_all = alt.Chart(summary_non_priority_all).mark_bar().encode(
    x = alt.X(
        'COMMUNITY_AREA:N',
        title = 'Community Area',
        axis = alt.Axis(labelAngle = -90, labelOverlap = True)
    ),
    y = alt.Y(
        'Count:Q',
        title = 'Number of Incidents'
    ),
    color = alt.Color(
        'Period:N',
```

```
        title = 'Time Period',
        scale = alt.Scale(domain = ['2018-2020', '2021-2023'], range = ['#4C78A8', '#F58518']
    )
).properties(
    width = 600,  # Extend width to accommodate 62+ community areas
    height = 500,
    title = "Homicides and Non-Fatal Shootings Across Non-Priority Community Areas (2018-2020
)

chart_non_priority_all.show()
```

```
C:\Users\Jakub\AppData\Local\Temp\ipykernel_14352\2758274138.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
  data_filtered_non_priority_all['Year'] = data_filtered_non_priority_all['DATE'].dt.year
C:\Users\Jakub\AppData\Local\Temp\ipykernel_14352\2758274138.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
  data_filtered_non_priority_all['Period'] = data_filtered_non_priority_all['Year'].apply(


alt.Chart(...)
```