PS5

Ahona Roy and Brenda Castañeda

2024-11-09

Partner 1: Brenda Castaneda, brendac29 Partner 2: Ahona Roy, ahona1 "This submission is our work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: B.C./A.R. "I have uploaded the names of anyone else other than my partner and I worked with on the problem set **here**" (1 point) Late coins used this pset: 0/0

```
#setup
import pandas as pd
import altair as alt
import time
import requests
import lxml
from bs4 import BeautifulSoup
from urllib.parse import urljoin
from datetime import datetime
import geopandas as gpd
import matplotlib.pyplot as plt
from shapely import wkt

import warnings
warnings.filterwarnings('ignore')
alt.renderers.enable("png")
```

```
RendererRegistry.enable('png')
```

Step 1: Develop initial scraper and crawler

1. Scraping (PARTNER 1)

```
# make get request from website
url = 'https://oig.hhs.gov/fraud/enforcement/'
response = requests.get(url)

# convert into soup
soup = BeautifulSoup(response.text, 'lxml')

# scrape title of enforcement actions
# find h2 tags with specific class
titles = soup.find_all('h2', class_='usa-card__heading')
len(titles)
#scrape elements
titles_text = [item.get_text(strip=True) for item in titles]

#scrape elements of date
dates = soup.find_all('span', class_='text-base-dark padding-right-105')
dates_text = [item.get_text(strip=True) for item in dates]
```

```
#scrape elements of category
categories = soup.find_all('ul', class_="display-inline add-list-reset")
categories_text = [item.get_text(strip=True) for item in categories]

#scrape elements of links
links = soup.find_all('a', href = True)
links = [item for item in links if '/fraud/enforcement/' in item.get('href')]
filtered_links=links[3:23]
links_url = [item.get('href') for item in links]

#add baseline url to links
base_url = 'https://oig.hhs.gov/'
links_url = [urljoin(base_url, item.get('href')) for item in filtered_links]

#assemble scraped elements into dataframe
data = {
    'Title': titles_text,
    'Date': dates_text,
    'Category': categories_text,
    'Link': links_url
}

data = pd.DataFrame(data)
```

```
agencies = []
for url in links_url:
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
    agency_elements = soup.find_all('span', class_='padding-right-2
 ↪  text-base', string="Agency:")

    if agency_elements:
      for span in agency_elements:
        agency_name = span.find_next_sibling(text=True).strip()
        agencies.append(agency_name)
    else:
      agencies.append('N/A')

#update agencies on dataframe
data['agency_name'] = agencies
```

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code (PARTNER 2)

(1) start
(2) check the start_year is before 2013, if it is then it prints a message and stops execution, because only data from 2013 onward is available.
(3) start_date is set to the specified date and end_date is set to today's date
(4) Then I create an empty DataFrame to store all scraped records. base_url: Defines the base URL for scraping. 'page'Sets the page number to start at 1.
(5) Use BeautifulSoup to parse the HTML content of the page with lxml parser
(6) Then scrape tites, dates, categories, links and agency names and then create a dataframe called 'data'
(7) Convert date column to datetime format then do filtering and sorting.
(8) Implement if statement to check the earliest date on the current page. If it is earlier than start_date, it breaks out of the loop.
(9) Increment the page number to scrape the next page. Use time.sleep for a 1 second to prevent overwhelming the server.
(10) Filter the data to include only records within start_date to end_date
(11) Save filtered_data to a CSV file with this name enforcement_actions_startyear_startmonth.csv.

- b. Create Dynamic Scraper (PARTNER 2)

```
#define function
def get_enforcement_actions(start_month, start_year):
    # Check if the year is 2013 or later
    if start_year < 2013:
        print("Please enter a year >= 2013, as only enforcement actions from
         ↪  2013 onward are available.")
        return None

    # Generate start date from month and year
    start_date = datetime(start_year, start_month, 1)
    end_date = datetime.today()

    # Log the scraping start
    print(f"Starting scraping from {start_month}/{start_year} to today:
     ↪  {end_date.strftime('%m/%d/%Y')}")

    # Placeholder DataFrame to store results
    all_data = pd.DataFrame()
```

```
    base_url = 'https://oig.hhs.gov/fraud/enforcement/'

    page = 1  # Start at the first page
    while True:
        # Construct the URL with the current page number
        url = f"{base_url}?page={page}"
        response = requests.get(url)
        if response.status_code != 200:
            print("Failed to retrieve the page.")
            break

        # Convert to BeautifulSoup
        soup = BeautifulSoup(response.text, 'lxml')

        # Scrape titles
        titles = soup.find_all('h2', class_='usa-card__heading')
        titles_text = [item.get_text(strip=True) for item in titles]

        # Scrape dates
        dates = soup.find_all('span', class_='text-base-dark
↪   padding-right-105')
        dates_text = [item.get_text(strip=True) for item in dates]

        # Scrape categories
        categories = soup.find_all('ul', class_="display-inline
↪   add-list-reset")
        categories_text = [item.get_text(strip=True) for item in categories]

        # Scrape links
        links = soup.find_all('a', href=True)
        links = [item for item in links if '/fraud/enforcement/' in
↪   item.get('href')]
        links_url = [urljoin(base_url, item.get('href')) for item in
↪   links[3:23]]  # Limiting to 20 links per page for now

        # Assemble scraped elements into DataFrame
        data = pd.DataFrame({
            'Title': titles_text,
            'Date': dates_text,
            'Category': categories_text,
            'Link': links_url
        })
```

```
        # Convert 'Date' column to datetime, making sure it's in a standard
         ↪ format
        data['Date'] = pd.to_datetime(data['Date'], errors='coerce',
↪ format='%B %d, %Y')
        all_data = pd.concat([all_data, data], ignore_index=True)

        # Break if the earliest date on this page is older than `start_date`
        if data['Date'].min() < start_date:
            break

        # Increment to the next page
        page += 1

        # Sleep to prevent overwhelming the server
        time.sleep(1)

    # --- Begin agency scraping logic ---
    agencies = []  # List to store agency names
    for index, row in all_data.iterrows():
        link = row['Link']  # Get the link for the current enforcement action
        response = requests.get(link)
        soup = BeautifulSoup(response.content, 'html.parser')

        # Look for the agency name on the enforcement action page
        agency_elements = soup.find_all('span', class_='padding-right-2
↪ text-base', string="Agency:")
        if agency_elements:
            for span in agency_elements:
                agency_name = span.find_next_sibling(text=True).strip()
                agencies.append(agency_name)
        else:
            agencies.append('N/A')  # If no agency is found, append 'N/A'

    # Ensure agencies list length matches the number of enforcement actions
    if len(agencies) != len(all_data):
        print(f"Warning: Length mismatch between agencies list
         ↪ ({len(agencies)}) and all_data ({len(all_data)})")

    # Add the agencies list as a new column in the DataFrame
    all_data['Agency'] = agencies
```

```
    # Filter by start date and sort
    filtered_data = all_data[(all_data['Date'] >= start_date) &
↪   (all_data['Date'] <= end_date)]
    filtered_data =
↪   filtered_data.sort_values(by='Date').reset_index(drop=True)

    # Save to CSV
    csv_filename = f"enforcement_actions_{start_year}_{start_month}.csv"
    filtered_data.to_csv(csv_filename, index=False)
    print(f"Data saved to {csv_filename}")

    return filtered_data
```

```
get_enforcement_actions(1, 2023)
```

There were 1534 enforcement actions scraped. The earliest is from 01/03/2023. It was a Criminal and Civil Action enforcement by the U.S. Attorney's Office, Southern District of Texas titled "Podiatrist Pays $90,000 To Settle False Billing Allegations."

- c. Test Partner's Code (PARTNER 1)

```
get_enforcement_actions(1, 2021)
```

There are 3022 enforcement actions in the final dataset. The earliest enforcment action scraped was from 01/04/2024. It was a Criminal and Civil Action enforcment by the U.S. Attorney's Office, Middle District of Tennessee titled "The United States And Tennessee Resolve Claims With Three Providers For False Claims Act Liability Relating To â€˜P-Stimâ€™ Devices For A Total Of $1.72 Million."

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time (PARTNER 2)

```
#read in data
enforcement_actions = pd.read_csv('enforcement_actions_2021_1.csv')

# Convert to datetime
enforcement_actions['Date'] = pd.to_datetime(enforcement_actions['Date'],
↪   errors='coerce')
```

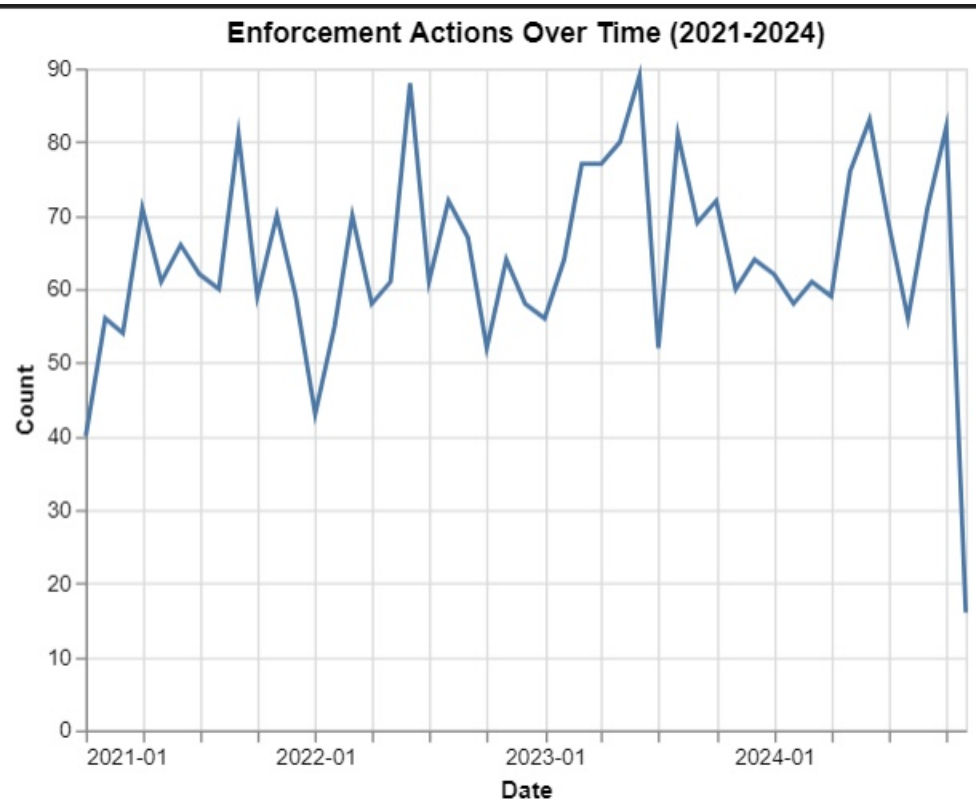
```
#create month-year column
enforcement_actions['YearMonth'] =
 ↪  enforcement_actions['Date'].dt.to_period('M')

# Convert back to datetime
enforcement_actions['YearMonth'] =
 ↪  enforcement_actions['YearMonth'].dt.to_timestamp()

# Group by 'YearMonth' and count the occurrences
enforcement_monthly =
 ↪  enforcement_actions.groupby('YearMonth').size().reset_index(name='Count')

# Set the renderer to 'default'
alt.renderers.enable('default')
#CGPT: I pasted this error I was getting that wasn't displaying my plot.
 ↪  ValueError: Saving charts in 'png' format requires the vl-convert-python
 ↪  package: see
 ↪  https://altair-viz.github.io/user_guide/saving_charts.html#png-svg-and-pdf-format

# Create the chart
enforcement_time = alt.Chart(enforcement_monthly).mark_line().encode(
    x=alt.X('YearMonth:T', title='Date', axis=alt.Axis(format='%Y-%m')),
    y='Count:Q'
).properties(
    title='Enforcement Actions Over Time (2021-2024)',
    width = 400
)

#display
enforcement_time
```

```
alt.Chart(...)
```

2. Plot the number of enforcement actions categorized: (PARTNER 1)

- based on "Criminal and Civil Actions" vs. "State Enforcement Agencies"

```
enforcement_category = enforcement_actions.groupby(['YearMonth',
 ↪  'Category']).size().reset_index(name='Count')
```

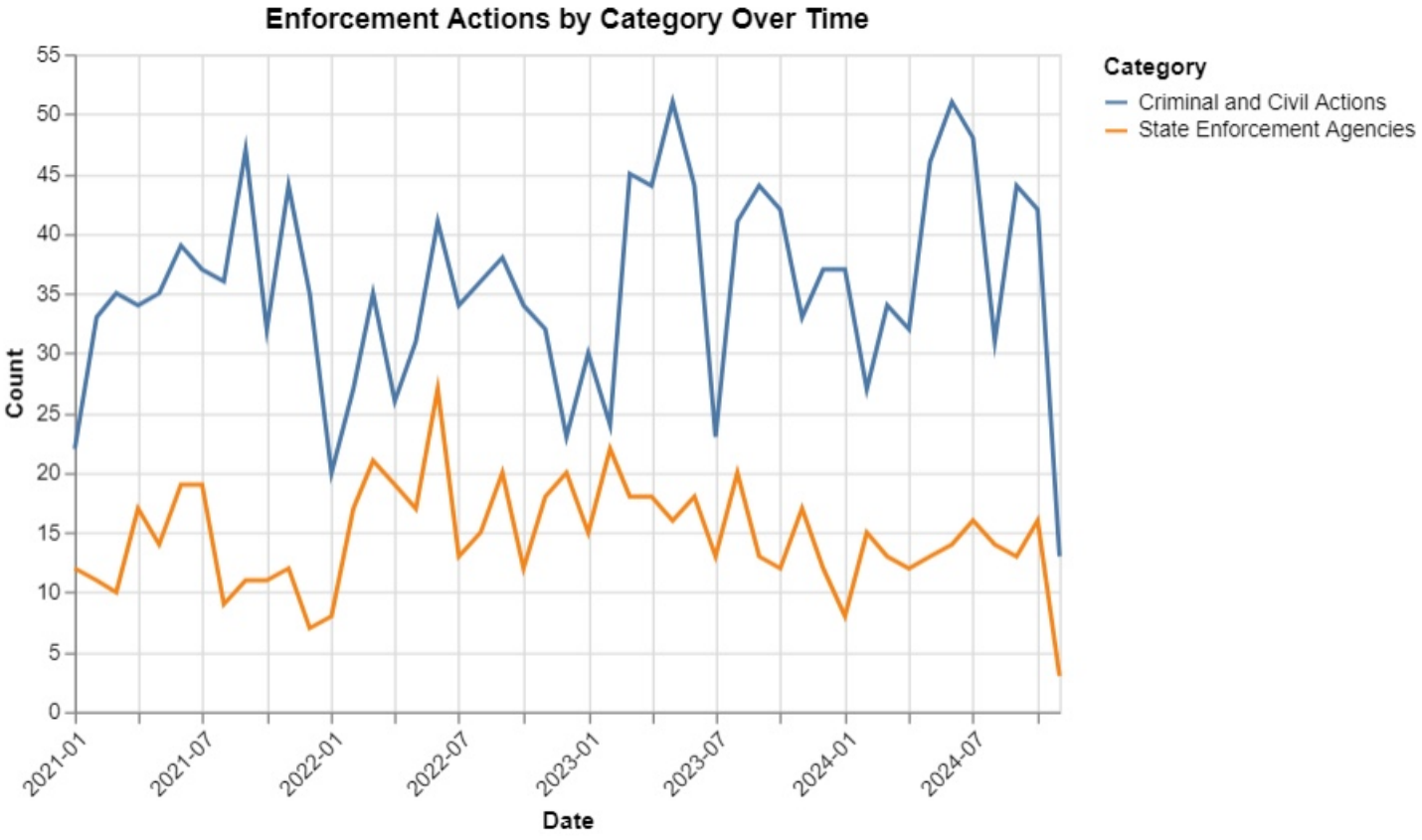

Enforcement Actions Over Time (2021-2024)
Count
90
80
70
60
50
40
30
20
10
0
2021-01
2022-01
2023-01
2024-01
Date

```
#subset criminal and state enforcement
enforcement_category = enforcement_category[
  (enforcement_category["Category"] == "Criminal and Civil Actions") |
  (enforcement_category["Category"] == "State Enforcement Agencies")]

# Create the Altair line chart
enforcement_time_category =
 ↪  alt.Chart(enforcement_category).mark_line().encode(
    x=alt.X('YearMonth:T', title='Date', axis=alt.Axis(format='%Y-%m',
 ↪  labelAngle=-45)),  # Rotate x-axis labels for readability
    y='Count:Q',
    color='Category:N'  # Different line color for each category
).properties(
    title='Enforcement Actions by Category Over Time',
    width=450  # Set chart width (adjust as needed)
)

# display
enforcement_time_category
```

`alt.Chart(...)`

- based on five topics

```
#subset to criminal and civil actions
criminal_civil =
 ↪  enforcement_actions[enforcement_actions['Category']=="Criminal and Civil
 ↪  Actions"]

#new topic column
criminal_civil['Topic'] = "Other"

#financial fraud
criminal_civil.loc[criminal_civil['Title'].str.contains("bank|financial",
 ↪  case=False, na=False), 'Topic'] = "Financial Fraud"
#health care fraud
criminal_civil.loc[criminal_civil['Title'].str.contains("health|clinic|physician|care|medica
 ↪  case=False, na=False), 'Topic'] = "Health Care Fraud"
#bribery/corruption
criminal_civil.loc[criminal_civil['Title'].str.contains("bribe|money|corruption",
 ↪  case=False, na=False), 'Topic'] = "Bribery/Corruption"
#narcotics
```


Enforcement Actions by Category Over Time
Count
Date
Category
Criminal and Civil Actions
State Enforcement Agencies
0
5
10
15
20
25
30
35
40
45
50
55
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07

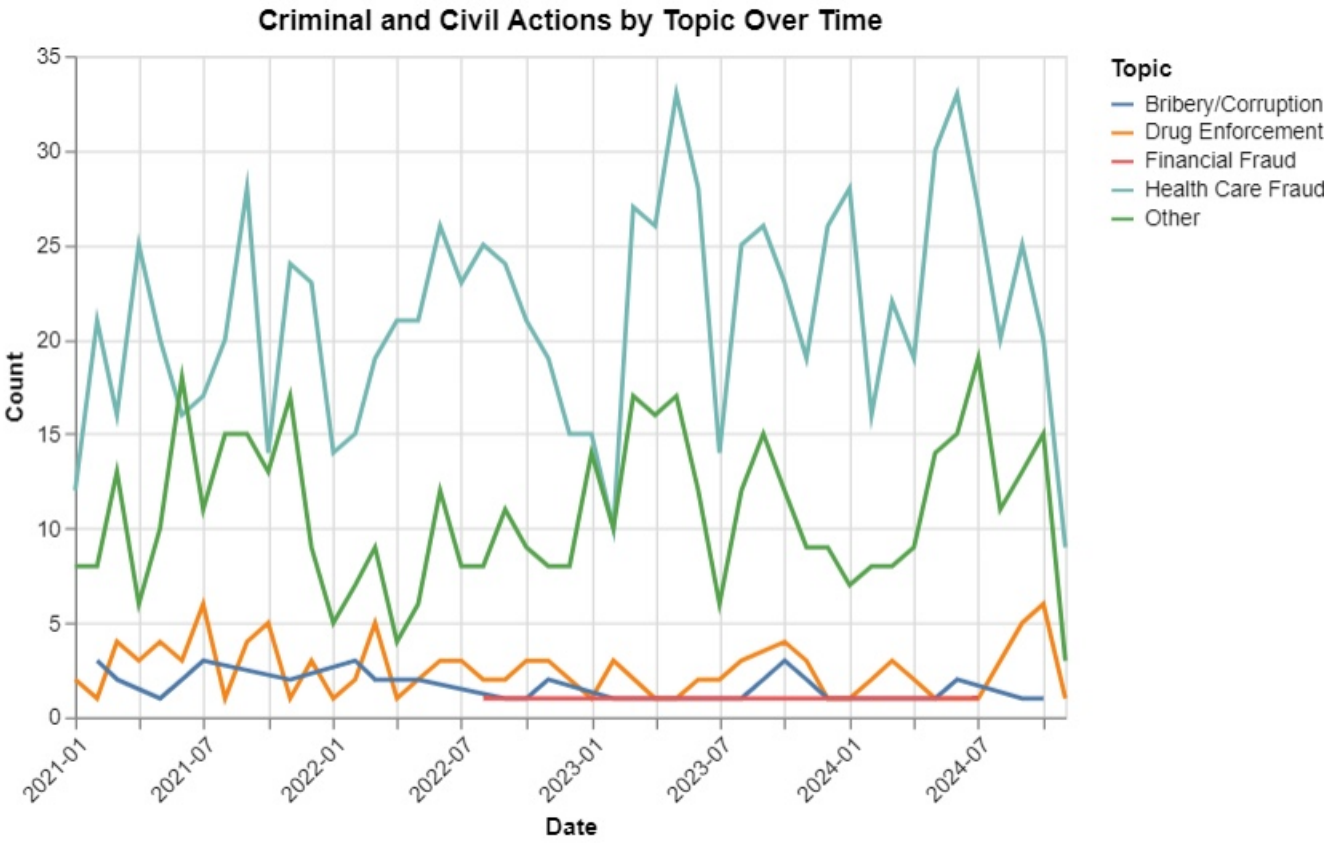
```
criminal_civil.loc[criminal_civil['Title'].str.contains("drug|narcotics|traficking",
 ↪  case=False, na=False), 'Topic'] = "Drug Enforcement"

criminal_civil_topic = criminal_civil.groupby(['YearMonth',
 ↪  'Topic']).size().reset_index(name='Count')

#plot enforcment actions by topic over time
enforcement_topic = alt.Chart(criminal_civil_topic).mark_line().encode(
    x=alt.X('YearMonth:T', title='Date', axis=alt.Axis(format='%Y-%m',
 ↪  labelAngle=-45)),
    y='Count:Q',
    color='Topic:N'
).properties(
    title='Criminal and Civil Actions by Topic Over Time',
    width=450
)

#display
enforcement_topic
```

```
alt.Chart(...)
```

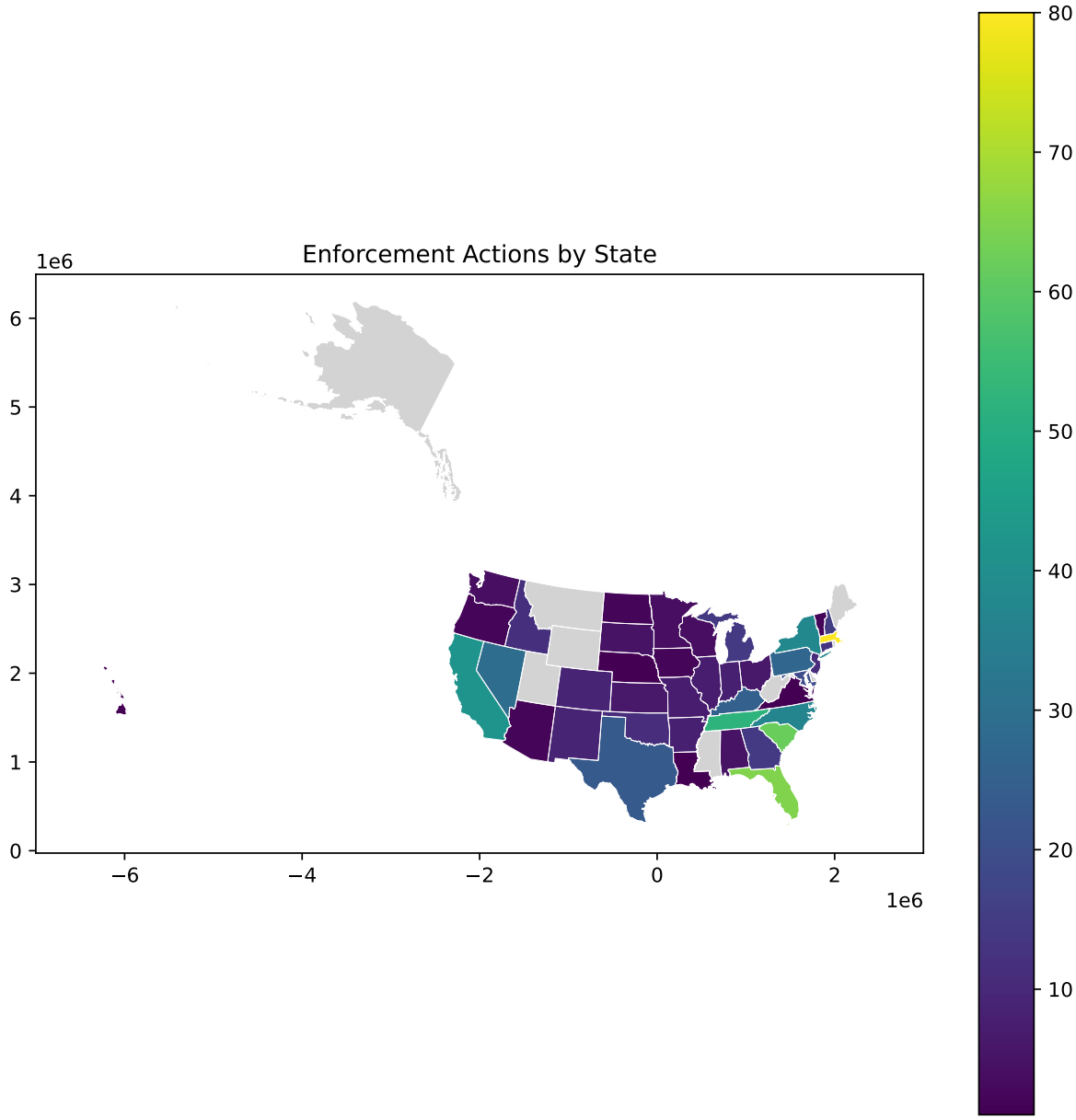
Step 4: Create maps of enforcement activity

1. Map by State (PARTNER 1)

```
#subset state enforcement actions
state_actions = enforcement_actions[enforcement_actions['Category'] == "State
 ↪  Enforcement Agencies"]

#list of 50 states
states = [
    'Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California', 'Colorado',
     ↪  'Connecticut', 'Delaware',
    'Florida', 'Georgia', "Hawai'i", 'Idaho', 'Illinois', 'Indiana', 'Iowa',
     ↪  'Kansas', 'Kentucky', 'Louisiana', 'Maine', 'Maryland',
     ↪  'Massachusetts', 'Michigan', 'Minnesota', 'Mississippi', 'Missouri',
     ↪  'Montana', 'Nebraska', 'Nevada', 'New Hampshire', 'New Jersey', 'New
     ↪  Mexico', 'New York', 'North Carolina', 'North Dakota', 'Ohio',
     ↪  'Oklahoma', 'Oregon', 'Pennsylvania', 'Rhode Island', 'South
     ↪  Carolina', 'South Dakota', 'Tennessee', 'Texas', 'Utah', 'Vermont',
     ↪  'Virginia', 'Washington', 'West Virginia', 'Wisconsin', 'Wyoming'
```


Criminal and Civil Actions by Topic Over Time
Count
Date
Topic
Bribery/Corruption
Drug Enforcement
Financial Fraud
Health Care Fraud
Other
0
5
10
15
20
25
30
35
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07

```
]
# extract state from agency name
def extract_state(agency_name):
    for state in states:
        if state in agency_name:
            return state
    return None

# Apply the function to state_actions
state_actions['NAME'] = state_actions['Agency'].apply(extract_state)

#group by state
state_actions_grouped =
 ↪  state_actions.groupby('NAME').size().reset_index(name='count')

#load state data
state = gpd.read_file('state.shp')
#remove territories from shapefile
territories = ['Puerto Rico', 'Guam', 'Commonwealth of the Northern Mariana
 ↪  Islands', 'American Samoa', 'United States Virgin Islands', 'District of
 ↪  Columbia']
state = state[~state['NAME'].isin(territories)]
#change hawaii
state['NAME'] = state['NAME'].replace('Hawaii', "Hawai'i")

#merge with state data
states_merged = state.merge(state_actions_grouped, left_on='NAME',
 ↪  right_on='NAME', how='left')

#plot map
states_merged = states_merged.to_crs(epsg=5070)

fig, ax = plt.subplots(1, 1, figsize=(10, 10))
states_merged.boundary.plot(ax=ax, color='white', linewidth = 0.2)
states_merged.plot(column='count', ax=ax, legend=True,
                   missing_kwds={'color': 'lightgrey', 'label': 'No
 ↪  closures'},
                   linewidth=0.2)
ax.set_title('Enforcement Actions by State')
ax.set_xlim(-7000000, 3000000)
# display map
plt.show()
```



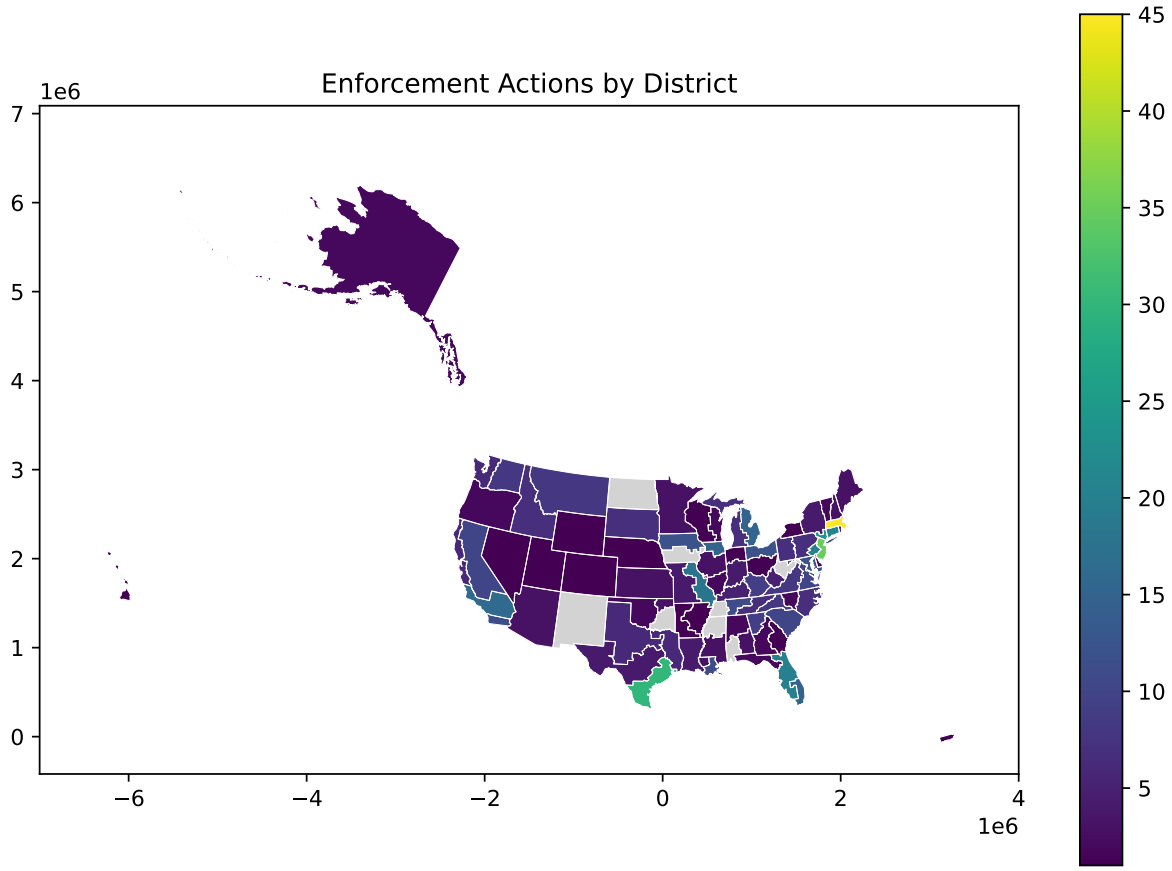
2. Map by District (PARTNER 2)

```
#function to extract district from agency name col
def extract_district(agency_name):
   # Ensure the agency name is a string and contains "U.S. Attorney's
    ↪  Office,"
```

```
    if isinstance(agency_name, str) and "U.S. Attorney's Office" in
     ↪  agency_name:
        # Split the string and get the part after "U.S. Attorney's Office"
        parts = agency_name.split("U.S. Attorney's Office,")[1].strip()  #
 ↪  Everything after "U.S. Attorney's Office"
        return parts
    else:
        return 'Not Available'  # If "U.S. Attorney's Office" is not found or
         ↪  it's not a valid string

# Apply function to extract district
enforcement_actions['District'] =
 ↪  enforcement_actions['Agency'].apply(extract_district)

#group by district
district_actions =
 ↪  enforcement_actions.groupby('District').size().reset_index(name='count')
```

```
#read in district shapefile
district = pd.read_csv('district.csv')

# Convert 'the_geom' column from WKT to actual geometry
district['geometry'] = district['the_geom'].apply(wkt.loads)

# Create a GeoDataFrame from the pandas DataFrame
district = gpd.GeoDataFrame(district, geometry='geometry')

# Set the coordinate reference system (CRS), adjust this if needed
district.set_crs('EPSG:4326', inplace=True)

#change judicial district column name
district.rename(columns={'Judicial District ': 'District'}, inplace=True)
#merge
districts_merged = district.merge(district_actions, left_on='District',
 ↪  right_on='District', how='left')


#Plot map
districts_merged = districts_merged.to_crs(epsg=5070)

fig, ax = plt.subplots(1, 1, figsize=(10, 7))
```

```
districts_merged.boundary.plot(ax=ax, color='white', linewidth=0.2)
districts_merged.plot(column='count', ax=ax, legend=True,
                      missing_kwds={'color': 'lightgrey', 'label': 'No
 ↪  closures'},
                      linewidth=0.2)
ax.set_title('Enforcement Actions by District')
ax.set_xlim(-7000000, 4000000)

# display map
plt.show()
```



Extra Credit

1. Merge zip code shapefile with population

```
#read in zips from ps4
zips = gpd.read_file('zips.shp')

#read in zip pop data
population = pd.read_csv('population.csv')
#clean population dataframe
population = population.iloc[1:].reset_index(drop=True)
population['NAME'] = population['NAME'].str.replace('ZCTA5 ', '',
 ↪  regex=False)

#merge zips with pop data
zips_population = zips.merge(population, left_on = 'ZCTA5', right_on =
 ↪  'NAME', how = 'left')
```

2. Conduct spatial join

```
#spatial join
zip_district = gpd.sjoin(
    zips_population,
    district,
    how='inner',  # or 'left'
    op='intersects'  # Use 'intersects' if you want to match zip codes inside
 ↪  districts
)

#aggregate pop by district
district_population = zip_district.groupby('District').agg(
    total_population=('P1_001N', 'sum')  # Replace 'population_column' with
 ↪  the actual column name for population
).reset_index()
```

3. Map the action ratio in each district