Problem set 5

Cristian Bancayan & Sol Rivas

2024-11-10

Due 11/9 at 5:00PM Central. Worth 100 points + 10 points extra credit.

Submission Steps (10 pts)

1. This problem set is a paired problem set.
2. Play paper, scissors, rock to determine who goes first. Call that person *Partner 1*.
    - Partner 1 (name and cnet ID): Sol Rivas Lopes - 12218930
    - Partner 2 (name and cnet ID): Cristian Bancayan - 12403034
3. Partner 1 will accept the `ps5` and then share the link it creates with their partner. You can only share it with one partner so you will not be able to change it after your partner has accepted.
4. "This submission is our work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: **SCRL** **___**
5. "I have uploaded the names of anyone else other than my partner and I worked with on the problem set **here**" (1 point)
6. Late coins used this pset: 1 **___** Late coins left after submission: 2. **___**
7. Knit your `ps5.qmd` to an PDF file to make `ps5.pdf`,
    - The PDF should not be more than 25 pages. Use `head()` and re-size figures when appropriate.
8. (Partner 1): push `ps5.qmd` and `ps5.pdf` to your github repo.
9. (Partner 1): submit `ps5.pdf` via Gradescope. Add your partner on Gradescope.
10. (Partner 1): tag your submission in Gradescope

```
import pandas as pd
import altair as alt
import time
from bs4 import BeautifulSoup
import requests
from datetime import datetime
import os
import geopandas as gpd
import matplotlib.pyplot as plt
import re
import warnings
warnings.filterwarnings('ignore')
alt.renderers.enable("png")

# Working directory
#Cris
os.chdir(r"C:\Users\Cristian\Documents\GitHub\ppha30538_fall2024\problem-set-5-sol-cristian")
# Sol
#os.chdir(r"C:\Users\solch\OneDrive\Documentos\2024 - autumn quarter\python
 ↪  II\problem-set-5-sol-cristian")
```

Step 1: Develop initial scraper and crawler

1. Scraping (PARTNER 1)

```
# To make the PDF knitting more efficient, we did not run the scraping since
 ↪  we had already obtained the database beforehand.

# Creating empty vector for data
all_enforcement_data = []

# Looping through each page
for page_num in range(1, 481):  # 480 pages total
    # Construct the URL for each page
    url = f"https://oig.hhs.gov/fraud/enforcement/?page={page_num}"
    response = requests.get(url)

    # Check if the request was successful
    # Chat GPT told me to add this as a general good measure when I fed it
     ↪  this code and asked how could I improve it
```

```
    if response.status_code != 200:
        print(f"Failed to retrieve page {page_num}")
        continue

    # Parse the page content
    soup = BeautifulSoup(response.text, "html.parser")

    # Extract each enforcement action on the page
    for action in soup.find_all("li", class_="usa-card card--list
    ↪  pep-card--minimal mobile:grid-col-12"):
        # Extract title
        title_tag = action.find("h2", class_="usa-card__heading")
        title = title_tag.get_text(strip=True)

        # Extract link
        link = title_tag.find("a")["href"]
        full_link = f"https://oig.hhs.gov{link}"  # The paths were relative

        # Extract date
        date = action.find("span", class_="text-base-dark
↪  padding-right-105").get_text(strip=True)

        # Extract category
        category = action.find("li", class_="display-inline-block usa-tag
↪  text-no-lowercase text-base-darkest bg-base-lightest
↪  margin-right-1").get_text(strip=True)

        # Create a dictionary and append it to items
        all_enforcement_data.append({
            "Title": title,
            "Date": date,
            "Category": category,
            "Link": full_link
        })

    # Track progress
    print(f"Completed page {page_num}")

    # Delay to avoid being blocked
    time.sleep(1)

# Convert the list to a DataFrame
```

```
df_enforcement = pd.DataFrame(all_enforcement_data)

# Print the first few rows of the dataset
print(df.head())

# Writing a csv to store this locally
df_enforcement.to_csv("enforcement_actions.csv", index=False)
```

2. Crawling (PARTNER 1)

```
# To make the PDF knitting more efficient, we did not run the crawling since
 ↪  we had already obtained the database beforehand.

# Since data for this problem set is only needed starting from January 2021
# I am dropping older entries from this df
df_enforcement = pd.read_csv("enforcement_actions.csv")
df_enforcement = df_enforcement[0:2995]

# putting links in a list
links = df_enforcement["Link"]

# Empty list
agency_info = []

# Loop through each link to get agency info
for index in range(0, 2995):
    url = links[index]
    response = requests.get(url)

    # Check if the request was successful
    if response.status_code != 200:
        print(f"Failed to retrieve agency info for {url}")
        agency_info.append(None)  # Append None if there's an error
        continue

    # Parse the page content
    soup = BeautifulSoup(response.text, "html.parser")

    # Find the <li> element, where <span> is
    agency_tag = None
```

```
    for li in soup.find_all("li"):
      #find <span> subelement where agency information is
        if li.find("span", class_="padding-right-2 text-base") and "Agency:"
         ↪  in li.get_text():
            agency_tag = li
            break  # Stop once we find the first matching <li> with "Agency:"

    # Extract the agency information if found
    if agency_tag:
        agency = agency_tag.get_text(strip=True).replace("Agency:",
↪   "").strip()
    # If it doesn't find any info on agency, say none
    else:
        agency = None

    # Append the agency info to agency object
    agency_info.append(agency)

    # Track progress
    # print(f"Completed retrieving agency info for row {index}")
    # Omit the tracking for the pdf

    # Delay to avoid being blocked
    time.sleep(1)

# Add the agency information to the DataFrame as a new column
df_enforcement["Agency"] = agency_info

# Writing a csv to store this locally
df_enforcement.to_csv("enforcement_actions_with_agency.csv", index=False)
```

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code (PARTNER 2)

The steps of the pseudo-code for turning the scraper into a function are:

1. Define the function with the two arguments for year and month:

def scraper_enforcement(year, month)

2. Set up the the input validation

In this step, we restrict the function to accept only years from 2013 onwards. If the user enters a year before 2013, the function should return the message: "Please enter a year from 2013 onwards!" If the year is 2013 or later, the function proceeds with its intended purpose. The validation check should be:

if year $<$ 2013 return "Please enter a year from 2013 onwards!"

3. Set the period for scraping:

We start with an empty list to store the results of each scraped enforcement action. Then, we set the current month to determine the number of years to scrape.

4. Set up the loop through pages:

We will create a loop to iterate through the pages (unitl the specified input month and year) from page 1. This step also constructs the URL for each page. Additionally, it makes a request to each URL, following the procedures from step 1, which includes scraping data from the website and crawling: clicking on each page and collecting the name of the agency involved. The loop uses while and stops for dates older than the user-entered year and month.

5. Save the dataframe

The final step converts the list into a DataFrame and saves it as a CSV file, with specifications for the file name.

- b. Create Dynamic Scraper (PARTNER 2)

```
def scrape_enforcement_actions(year, month):
    if year < 2013:
        print("Please enter a year from 2013 onwards!")
        year = 2013  # Start year

    all_enforcement_data = []

    # Base URL
    base_url = "https://oig.hhs.gov/fraud/enforcement/?page="
    page_num = 1
    is_more_pages = True  # Flag to check if there are more pages
    start_date = datetime(year, month, 1)

    # Looping through the pages
    while is_more_pages:
        url = base_url + str(page_num)
```

```
        response = requests.get(url)

        if response.status_code != 200:
            print(f"Failed to retrieve page {page_num}")
            break

        soup = BeautifulSoup(response.text, "html.parser")
        actions = soup.find_all("li", class_="usa-card card--list
↪  pep-card--minimal mobile:grid-col-12")

        # Break the loop if no actions are found
        if not actions:
            is_more_pages = False
            break

        # Extract data for each enforcement action
        for action in actions:
            title_tag = action.find("h2", class_="usa-card__heading")
            title = title_tag.get_text(strip=True) if title_tag else "N/A"
            link = title_tag.find("a")["href"] if title_tag else "#"
            full_link = f"https://oig.hhs.gov{link}"
            date_str = action.find("span", class_="text-base-dark
↪  padding-right-105").get_text(strip=True) if action.find("span",
↪  class_="text-base-dark padding-right-105") else "N/A"

            # Converting the date string to a datetime object for comparison
            date = datetime.strptime(date_str, "%B %d, %Y") if date_str !=
↪  "N/A" else None

            # Skip actions that are older than the start date
            if date and date < start_date:
                is_more_pages = False
                break

            category = action.find("li", class_="display-inline-block usa-tag
↪  text-no-lowercase text-base-darkest bg-base-lightest
↪  margin-right-1").get_text(strip=True) if action.find("li",
↪  class_="display-inline-block usa-tag text-no-lowercase text-base-darkest
↪  bg-base-lightest margin-right-1") else "N/A"

            # Append to the list
            all_enforcement_data.append({
```

```
                "Title": title,
                "Date": date_str,
                "Category": category,
                "Link": full_link
            })

        # print(f"Completed page {page_num}")  # Track progress. Omitted for
        ↪  knitting the pdf
        page_num += 1
        time.sleep(1)

    # Convert the data to a DataFrame
    df_enforcement = pd.DataFrame(all_enforcement_data)

    # Putting links in a list
    links = df_enforcement["Link"]

    # Empty list to store agency information
    agency_info = []

    # Function to retrieve agency info for a given URL
    def get_agency_info(url):
        try:
            response = requests.get(url)
            if response.status_code != 200:
                print(f"Failed to retrieve agency info for {url}")
                return None

            soup = BeautifulSoup(response.text, "html.parser")
            agency_tag = None
            for li in soup.find_all("li"):
                if li.find("span", class_="padding-right-2 text-base") and
                ↪  "Agency:" in li.get_text():
                    agency_tag = li
                    break

            if agency_tag:
                agency = agency_tag.get_text(strip=True).replace("Agency:",
↪  "").strip()
                return agency
            else:
                return None
```

```
        except Exception as e:
            print(f"Error retrieving agency info for {url}: {e}")
            return None

    # Loop through each enforcement action link to retrieve the agency info
    for index, url in enumerate(links):
        agency = get_agency_info(url)
        agency_info.append(agency)

        # Track progress. Omitted for knitting the pdf
        # print(f"Completed retrieving agency info for row {index}")
        time.sleep(0.5)

    # Add the agency information to the DataFrame as a new column
    df_enforcement["Agency"] = agency_info

    # Save the DataFrame to a CSV file
    file_name = f"enforcement_actions_{year}_{month}.csv"
    df_enforcement.to_csv(file_name, index=False)
    print(f"Data saved to {file_name}")

    return df_enforcement
```

```
df_2023 = scrape_enforcement_actions(2023, 1)
print(f"Total number of actions since January 2023: {len(df_2023)}")
print("Earliest action details:", df_2023.iloc[-1])
```

```
Data saved to enforcement_actions_2023_1.csv
Total number of actions since January 2023: 1534
Earliest action details: Title       Podiatrist Pays $90,000 To Settle False
Billin...
Date                                            January 3, 2023
Category                              Criminal and Civil Actions
Link        https://oig.hhs.gov/fraud/enforcement/podiatri...
Agency      U.S. Attorney's Office, Southern District of T...
Name: 1533, dtype: object
```

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

```
# Storing years and moths in lists
df_final = scrape_enforcement_actions(2021, 1)
```

```
print(f"Total number of actions between January 2021 and 2023:
 ↪  {len(df_final)}")
print("Earliest action details:", df_final.iloc[-1])
```

```
Data saved to enforcement_actions_2021_1.csv
Total number of actions between January 2021 and 2023: 3022
Earliest action details: Title       The United States And Tennessee Resolve
Claims...
Date                                            January 4, 2021
Category                              Criminal and Civil Actions
Link        https://oig.hhs.gov/fraud/enforcement/the-unit...
Agency      U.S. Attorney's Office, Middle District of Ten...
Name: 3021, dtype: object
```

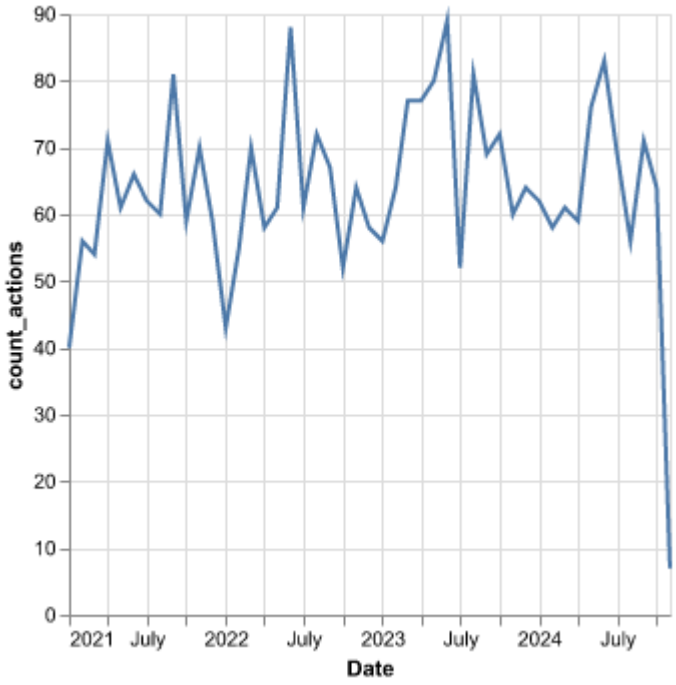
The number of rows is the same as the step 1.

Step 3: Plot data based on scraped data

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

```
enforcement_actions_with_agency =
 ↪  pd.read_csv("enforcement_actions_with_agency.csv")
# Make sure "Date" column is in dt form
enforcement_actions_with_agency["Date"] =
 ↪  pd.to_datetime(enforcement_actions_with_agency["Date"])
enforcement_actions_with_agency["Year"] =
 ↪  enforcement_actions_with_agency["Date"].dt.year
enforcement_actions_with_agency["Month"] =
 ↪  enforcement_actions_with_agency["Date"].dt.month

df_enforcement_ym = enforcement_actions_with_agency.groupby(["Year",
 ↪  "Month"]).aggregate(
    count_actions=("Title", "count")
).reset_index()

df_enforcement_ym['Date'] = pd.to_datetime(
    df_enforcement_ym[['Year', 'Month']].assign(DAY=1))

alt.Chart(df_enforcement_ym).mark_line().encode(
    alt.X('Date:T'),
```

```
    alt.Y('count_actions:Q')
)
```

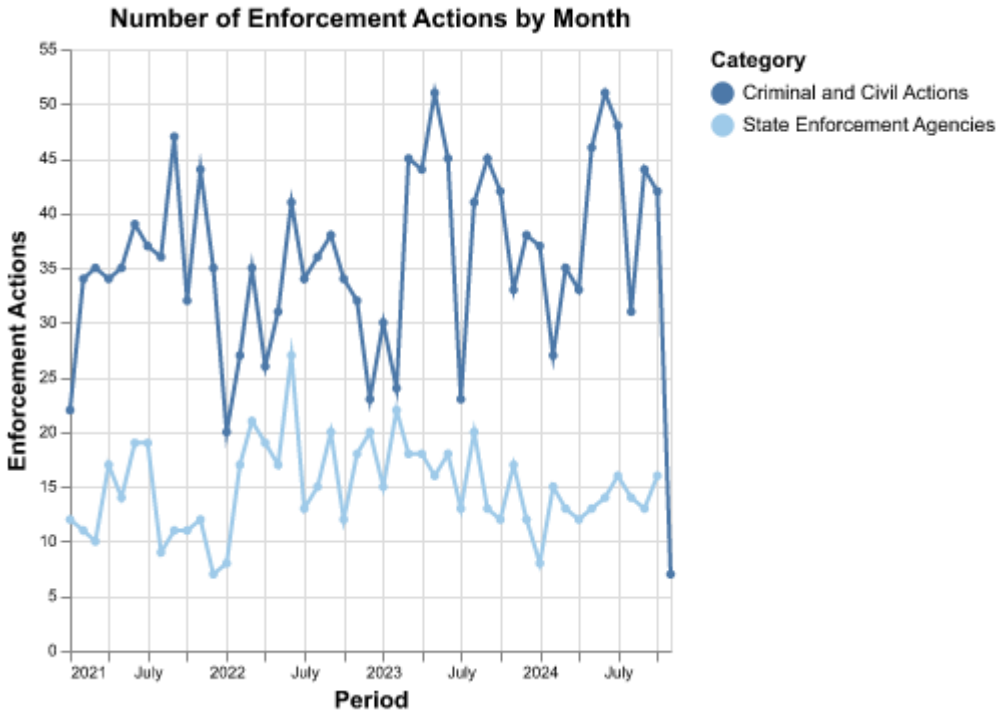


2. Plot the number of enforcement actions categorized: (PARTNER 1)

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

```
# Read in csv
df_enforcement = pd.read_csv("enforcement_actions_with_agency.csv")

# Make sure "Date" column is in dt form
df_enforcement["Date"] = pd.to_datetime(df_enforcement["Date"])
# Extract "period", aka month-year info
df_enforcement["Period"] = df_enforcement["Date"].dt.to_period("M")
# Make sure it's in dt form again
df_enforcement["Period"] = df_enforcement["Period"].dt.to_timestamp()

# Subset categories we want
df_enforcement_by_cat = df_enforcement[
    (df_enforcement["Category"] == "Criminal and Civil Actions") |
```

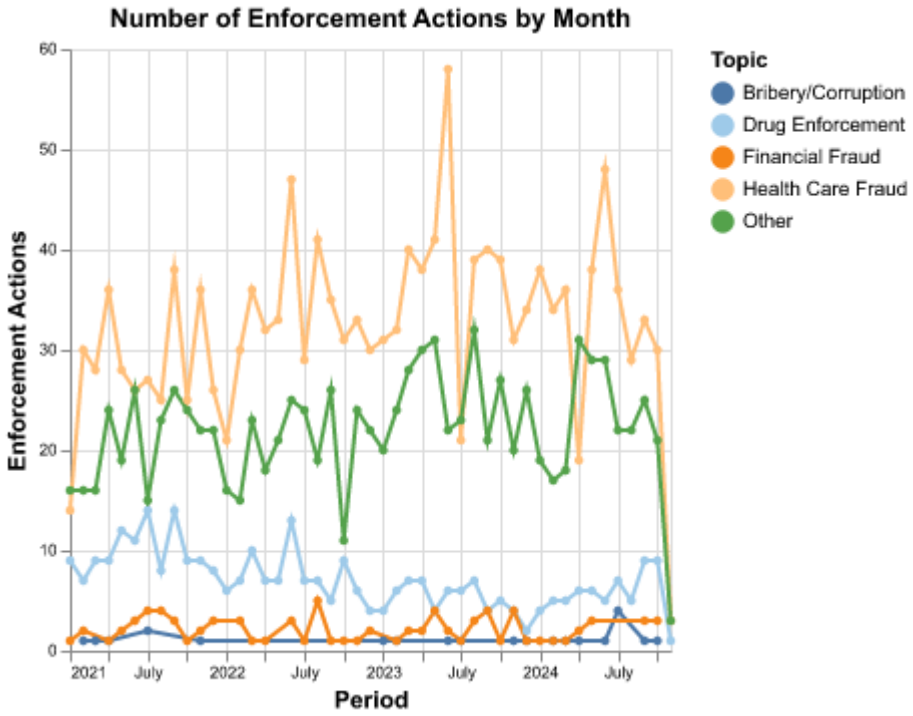
```
    (df_enforcement["Category"] == "State Enforcement Agencies")
]

# Count enforcement actions by category, for each period
df_enforcement_by_cat = df_enforcement_by_cat.groupby(["Category",
 ↪  "Period"]).size().reset_index()
# Rename columns
df_enforcement_by_cat.columns = ["Category", "Period", "Count"]

# Plot
alt.Chart(df_enforcement_by_cat, title="Number of Enforcement Actions by
 ↪  Month"
    ).mark_line(point=alt.OverlayMarkDef(color="white", size=20)
    ).encode(
    alt.X("Period:T", title="Period"),
    alt.Y("Count:Q", title="Enforcement Actions"),
    color=alt.Color("Category:N", title="Category",
 ↪  scale=alt.Scale(scheme="tableau20"))
).configure_axis(
    labelFontSize=8,
    titleFontSize=12
)
```



- based on five topics

```
# Creating a dictionary with keywords that match to each topic
# Source: ChatGPT
# Query: How can I use a dictionary to find keywords in observations of data
 ↪  frame?
topics = {
    "Health Care Fraud":
     ↪  r"\bHealth|Insurance|Care|Medicaid|Medicare|Healthcare|Medical\b",
    "Financial Fraud": r"\bBank|Financial|Investment|Business|Tax|Evasion\b",
    "Drug Enforcement":
     ↪  r"\bDrug|Narcotics|Pills|Opioid|Substance|Fentanyl\b",
    "Bribery/Corruption": r"\bBribery|Corruption|Extortion|Embezzlement\b",
}

# Create a "Topic" column
# Set to other so it will stay that way if keywords don't match to any type
df_enforcement["Topic"] = "Other"

# Use for loop to search for key words and assign topic
for topic, keywords in topics.items():
```

```
    df_enforcement.loc[df_enforcement["Title"].str.contains(keywords,
 ↪  case=False, na=False), "Topic"] = topic

# Count enforcement actions by topic, for each period
df_enforcement_by_topic = df_enforcement.groupby(["Topic",
 ↪  "Period"]).size().reset_index()
# Renaming columns
df_enforcement_by_topic.columns = ["Topic", "Period", "Count"]

# Plotting
alt.Chart(df_enforcement_by_topic, title="Number of Enforcement Actions by
 ↪  Month"
    ).mark_line(point=alt.OverlayMarkDef(color="white", size=20)
    ).encode(
    alt.X("Period:T", title="Period"),
    alt.Y("Count:Q", title="Enforcement Actions"),
    color=alt.Color("Topic:N", title="Topic",
 ↪  scale=alt.Scale(scheme="tableau20"))
).configure_axis(
    labelFontSize=8,
    titleFontSize=12
)
```



Step 4: Create maps of enforcement activity

1. Map by State (PARTNER 1)

```
# First, we will clean data by state

# Subsetting to state agencies only
df_enforcement_by_state =
 ↪  df_enforcement[df_enforcement["Agency"].str.contains("State of",
 ↪  na=False)]

# Creating a dictionary that matches each state to itself
states = {
    "Alabama": r"\bAlabama\b",
    "Alaska": r"\bAlaska\b",
    "Arizona": r"\bArizona\b",
    "Arkansas": r"\bArkansas\b",
    "California": r"\bCalifornia\b",
    "Colorado": r"\bColorado\b",
```

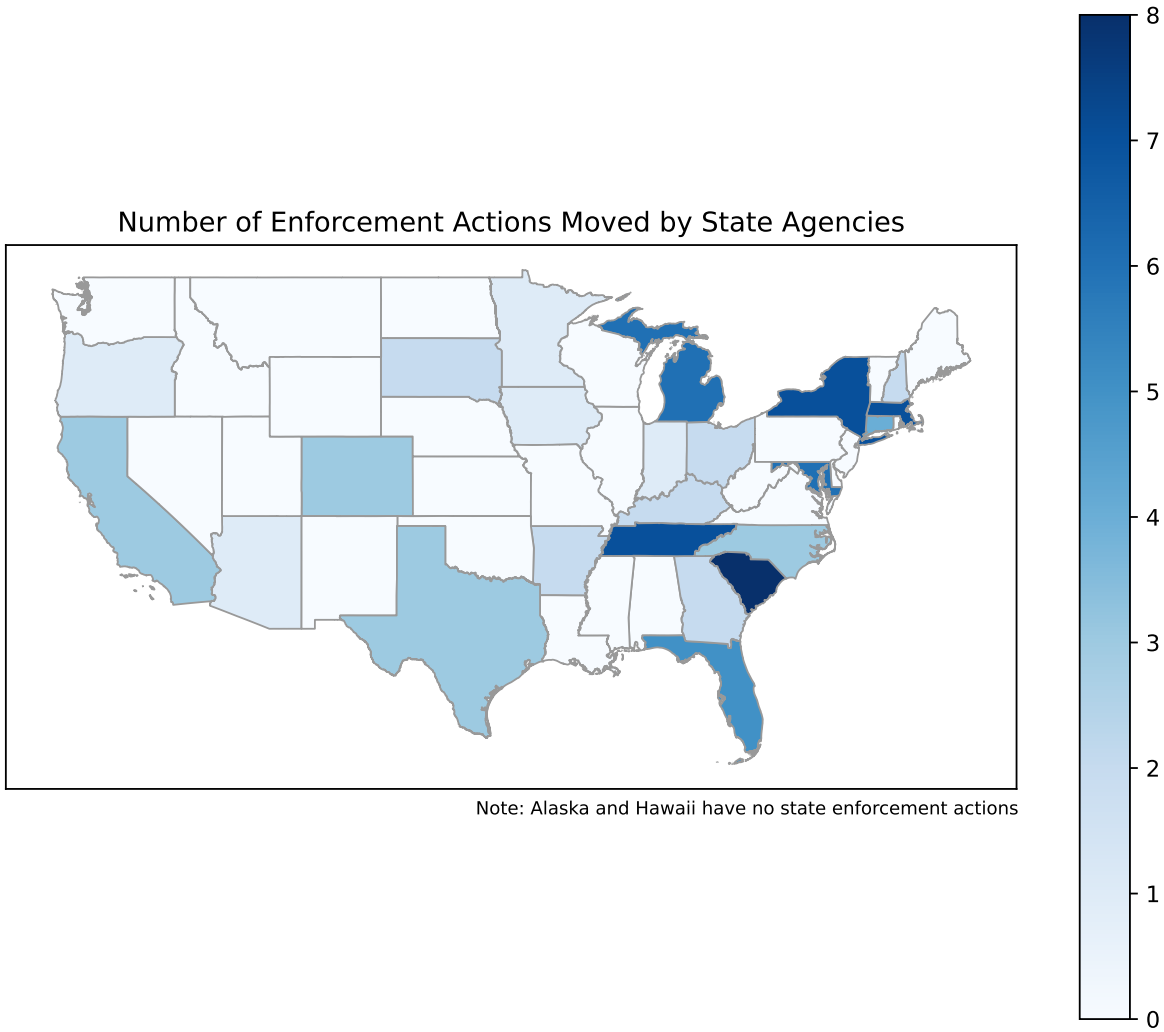
```
    "Connecticut": r"\bConnecticut\b",
    "Delaware": r"\bDelaware\b",
    "Florida": r"\bFlorida\b",
    "Georgia": r"\bGeorgia\b",
    "Hawaii": r"\bHawaii\b",
    "Idaho": r"\bIdaho\b",
    "Illinois": r"\bIllinois\b",
    "Indiana": r"\bIndiana\b",
    "Iowa": r"\bIowa\b",
    "Kansas": r"\bKansas\b",
    "Kentucky": r"\bKentucky\b",
    "Louisiana": r"\bLouisiana\b",
    "Maine": r"\bMaine\b",
    "Maryland": r"\bMaryland\b",
    "Massachusetts": r"\bMassachusetts\b",
    "Michigan": r"\bMichigan\b",
    "Minnesota": r"\bMinnesota\b",
    "Mississippi": r"\bMississippi\b",
    "Missouri": r"\bMissouri\b",
    "Montana": r"\bMontana\b",
    "Nebraska": r"\bNebraska\b",
    "Nevada": r"\bNevada\b",
    "New Hampshire": r"\bNew Hampshire\b",
    "New Jersey": r"\bNew Jersey\b",
    "New Mexico": r"\bNew Mexico\b",
    "New York": r"\bNew York\b",
    "North Carolina": r"\bNorth Carolina\b",
    "North Dakota": r"\bNorth Dakota\b",
    "Ohio": r"\bOhio\b",
    "Oklahoma": r"\bOklahoma\b",
    "Oregon": r"\bOregon\b",
    "Pennsylvania": r"\bPennsylvania\b",
    "Rhode Island": r"\bRhode Island\b",
    "South Carolina": r"\bSouth Carolina\b",
    "South Dakota": r"\bSouth Dakota\b",
    "Tennessee": r"\bTennessee\b",
    "Texas": r"\bTexas\b",
    "Utah": r"\bUtah\b",
    "Vermont": r"\bVermont\b",
    "Virginia": r"\bVirginia\b",
    "Washington": r"\bWashington\b",
    "West Virginia": r"\bWest Virginia\b",
```

```
    "Wisconsin": r"\bWisconsin\b",
    "Wyoming": r"\bWyoming\b"
}


# Creating a "State" column
df_enforcement_by_state["State"] = "Placeholder"

# Use a for loop to search for states in "Agency" column using dictionary
for state, name in states.items():
    # Use str.contains with the correct regex pattern and na=False to avoid
     ↪  errors with NaN values

 ↪  df_enforcement_by_state.loc[df_enforcement_by_state["Agency"].str.contains(name,
 ↪  case=False, na=False), "State"] = state

# Count enforcement actions by state, for each period
df_enforcement_by_state =
 ↪  df_enforcement_by_state.groupby("State").size().reset_index()
# Renaming columns
df_enforcement_by_state.columns = ["State", "Count"]
```

```
# Read in shp file, merge, and clean


# Reading in file
shp_state = gpd.read_file("cb_2018_us_state_500k.shp")

# Checking columns and getting a sense of how shp is structured
shp_state.columns
shp_state.head()
# Renaming columns to be able to merge
shp_state.rename(columns={"NAME": "State"}, inplace=True)

# Merging
gdf_by_state = shp_state.merge(df_enforcement_by_state, on="State",
 ↪  how="outer")

# Making sure "Count" NaNs are set to 0 (which is what they mean)
gdf_by_state["Count"] = gdf_by_state["Count"].fillna(0)
```

```
# Dropping territories
gdf_by_state = gdf_by_state[gdf_by_state["State"].isin(states.keys())]
# Since Alaska and Hawaii both have 0 enforcement actions
# I'm dropping them because otherwise the choropleth does not look very nice
gdf_by_state = gdf_by_state[~gdf_by_state["State"].isin(["Alaska",
 ↪  "Hawaii"])]
```

```
# Plotting
#---------

fig, ax = plt.subplots(1, 1, figsize=(10, 8))
gdf_by_state.plot(column="Count", cmap="Blues",
                    linewidth=0.8, ax=ax, edgecolor="0.6", legend=True)
ax.set_title("Number of Enforcement Actions Moved by State Agencies")

# Asked ChatGPT to help me add a note about dropping Alaska and Hawaii
# Query: How do I add a note to this graph? [Fed it my code]
ax.text(
    -81, 22,  # Coordinates for the note (adjust as necessary)
    "Note: Alaska and Hawaii have no state enforcement actions",
    fontsize=8,
    color='black',
    ha='center'
)

# Remove axis and ticks
ax.set_xticks([])
ax.set_yticks([])

plt.show()
```



2. Map by District (PARTNER 2)

```
# Import the shape file
shp_file_district = pd.read_csv(
    "US_Attorney_Districts_Shapefile_simplified_20241107.csv")
shp_file_district.columns
shp_file_district.head(3)

shp_file_district = gpd.GeoDataFrame(
    shp_file_district,
 ↪  geometry=gpd.GeoSeries.from_wkt(shp_file_district['the_geom']))
```

```
shp_file_district.head(3)

# Clean the district name of the enforcement actions
def extract_district(agency):
    if isinstance(agency, str):
        # Look for patterns like "Eastern District of Washington" or
        ↪  "District of Connecticut"
        match = re.search(r"(Eastern|Western|Northern|Southern)? ?District of
↪  [\w\s]+", agency, re.IGNORECASE)
        if match:
            return match.group(0).strip()  # Return the matched district name
    return None  # Return None if no match is found

enforcement_actions_with_agency["District"] =
↪  enforcement_actions_with_agency["Agency"].apply(extract_district)

# Keep districts from the enforcement data
enforcement_districts =
↪  enforcement_actions_with_agency[enforcement_actions_with_agency["Agency"].str.contains(
    "District", case=False, na=False)]
enforcement_districts = enforcement_districts.groupby("District").aggregate(
    freq = ("Title", "count")
)
enforcement_districts.head(3)

# Merge
enforcement_districts_shp = shp_file_district.merge(enforcement_districts,
                                                    left_on="Judicial
↪  District ",
                                                    right_on="District",
                                                    how="inner")
```

```
# Plotting
fig, ax = plt.subplots(figsize=(8, 8))
enforcement_districts_shp.plot(
    column="freq",
    cmap="Greens",
    legend=False,
    linewidth=0.5,
    edgecolor='grey',
    ax=ax
```

```
)

# Set the x-axis limit to focus on the desired area
ax.set_xlim(-180, -50)  # Adjust the limits as needed

# Add title and axis labels
ax.set_title("Number of enforcement actions in each US Attorney District",
 ↪  fontsize=16)
ax.set_xlabel("Longitude", fontsize=10)
ax.set_ylabel("Latitude", fontsize=10)

sm = plt.cm.ScalarMappable(cmap="Greens",
 ↪  norm=plt.Normalize(vmin=enforcement_districts_shp["freq"].min(),
 ↪  vmax=enforcement_districts_shp["freq"].max()))
cbar = fig.colorbar(sm, ax=ax, orientation="horizontal", pad=0.04,
 ↪  fraction=0.046, aspect=40)
cbar.set_label("Frequency", fontsize=12)

plt.show()
```