PS5: Web Scraping

Maryell Abella & Sarah Kim

2024-11-09

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): Maryell Abella, maryell
    - Partner 2 (name and cnet ID): Sarah Kim, sarahk1
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: **MA** **SK**
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: **0** Late coins left after submission: **4**
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
import requests
from bs4 import BeautifulSoup
import warnings
import re
from time import sleep
from datetime import datetime
import geopandas as gpd
import matplotlib.pyplot as plt
from requests.exceptions import RequestException
import concurrent.futures
import matplotlib.colors as colors
import os
warnings.filterwarnings('ignore')
alt.renderers.enable("png")
path = r'/Users/maryell/Desktop/problem-set-5-maryell-sarah/'
```

Step 1: Develop initial scraper and crawler

1. Scraping (PARTNER 1)

1. (Partner 1) Scraping: Go to the first page of the HHS OIG's "Enforcement Actions" page and scrape and collect the following into a dataset: • Title of the enforcement action • Date • Category (e.g, "Criminal and Civil Actions") • Link associated with the enforcement action

```
url = "https://oig.hhs.gov/fraud/enforcement/"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'lxml')
```

```
"""Make an empty list for Enforcemetn Actions"""
enforcement_actions = []

"""Find li that contains div under main"""
main_tag = soup.find('main')
li_with_div = soup.find_all(lambda t: t.name == 'li' and t.find('div'))
li_with_div_content = [li.contents for li in li_with_div]
```

```
"""For Loop to gather data from each instance"""
for li in li_with_div_content[:19]:
    li_soup = BeautifulSoup(''.join(str(item) for item in li), 'html.parser')

    title_tag = li_soup.select_one('h2 a')
    title = title_tag.get_text(strip=True)
    link = title_tag['href']

    date_tag = li_soup.select_one('span')
    date = date_tag.get_text(strip=True)

    category_tag = li_soup.select_one('ul li')
    category = category_tag.get_text(strip=True)

    enforcement_actions.append({
        'Title': title,
        'Link': f"https://oig.hhs.gov{link}",
        'Date': date,
        'Category': category
    })

"""Add information into a data frame"""
enforcement_df = pd.DataFrame(enforcement_actions)
print(enforcement_df.head())
```

```
                                               Title  \
0  Pharmacist and Brother Convicted of $15M Medic...
1  Boise Nurse Practitioner Sentenced To 48 Month...
2  Former Traveling Nurse Pleads Guilty To Tamper...
3  Former Arlington Resident Sentenced To Prison ...
4  Paroled Felon Sentenced To Six Years For Fraud...

                                                Link              Date  \
0  https://oig.hhs.gov/fraud/enforcement/pharmaci...  November 8, 2024
1  https://oig.hhs.gov/fraud/enforcement/boise-nu...  November 7, 2024
2  https://oig.hhs.gov/fraud/enforcement/former-t...  November 7, 2024
3  https://oig.hhs.gov/fraud/enforcement/former-a...  November 7, 2024
4  https://oig.hhs.gov/fraud/enforcement/paroled-...  November 7, 2024

                    Category
0  Criminal and Civil Actions
1  Criminal and Civil Actions
```

```
2  Criminal and Civil Actions
3  Criminal and Civil Actions
4  Criminal and Civil Actions
```

2. Crawling (PARTNER 1)

2. (Partner 1) Crawling: Then for each enforcement action, click the link and collect the name of the agency involved (e.g., for this link, it would be U.S. Attorney's Office, Eastern District of Washington). Update your dataframe with the name of the agency and print its head.

```
url_1 =
 ↪  "https://oig.hhs.gov/fraud/enforcement/north-texas-medical-center-pays-142-million-to-res

"""Make a request for url and use BeautifulSoup"""
response_1 = requests.get(url_1)
soup_1= BeautifulSoup(response_1.content, 'html.parser')

"""Find li tags that include span to find Agency information"""
li_tags = soup_1.find_all('li')
for li in li_tags:
    agency_span = li.find('span')

    if agency_span and "Agency" in agency_span.text:
        agency_name = agency_span.find_next_sibling(text=True)

        if agency_name:
            print(agency_name.strip())
```

```
U.S. Attorney's Office, Northern District of Texas
```

```
"""Make an empty list for Agency names"""
agency_names = []

"""For loop to run through links to collect agency info and append"""
for link in enforcement_df['Link']:
    response_1 = requests.get(link)
    soup_1 = BeautifulSoup(response_1.content, 'html.parser')

    li_tags = soup_1.find_all('li')
    agency_name = None
```

```
    for li in li_tags:
        agency_span = li.find('span')

        if agency_span and "Agency" in agency_span.text:
            agency_name = agency_span.find_next_sibling(text=True)
            if agency_name:
                agency_name = agency_name.strip()

    agency_names.append(agency_name)

"""Adding column to df and printing head"""
enforcement_df['Agency'] = agency_names
print(enforcement_df.head())
```

```
                                               Title  \
0  Pharmacist and Brother Convicted of $15M Medic...
1  Boise Nurse Practitioner Sentenced To 48 Month...
2  Former Traveling Nurse Pleads Guilty To Tamper...
3  Former Arlington Resident Sentenced To Prison ...
4  Paroled Felon Sentenced To Six Years For Fraud...

                                                Link               Date  \
0  https://oig.hhs.gov/fraud/enforcement/pharmaci...  November 8, 2024
1  https://oig.hhs.gov/fraud/enforcement/boise-nu...  November 7, 2024
2  https://oig.hhs.gov/fraud/enforcement/former-t...  November 7, 2024
3  https://oig.hhs.gov/fraud/enforcement/former-a...  November 7, 2024
4  https://oig.hhs.gov/fraud/enforcement/paroled-...  November 7, 2024

                    Category  \
0  Criminal and Civil Actions
1  Criminal and Civil Actions
2  Criminal and Civil Actions
3  Criminal and Civil Actions
4  Criminal and Civil Actions

                                             Agency
0                           U.S. Department of Justice
1  November 7, 2024; U.S. Attorney's Office, Dist...
2  U.S. Attorney's Office, District of Massachusetts
3  U.S. Attorney's Office, Eastern District of Vi...
4  U.S. Attorney's Office, Middle District of Flo...
```

Step 2: Making the scraper dynamic

Turning the scraper into a function: You will write a function that takes as input a month and a year, and then pulls and formats the enforcement actions like in Step 1 starting from that month+year to today.

• This function should first check that the year inputted >= 2013 before starting to scrape. If the year inputted < 2013, it should print a statement reminding the user to restrict to year >= 2013, since only enforcement actions after 2013 are listed. • It should save the dataframe output into a .csv file named as "enforcement_actions_ year_month.csv" (do not commit this file to git) • If you're crawling multiple pages, always add 1 second wait before going to the next page to prevent potential server-side block. To implement this in Python, you may look up .sleep() function from time library.

1. Turning the scraper into a function

- a. Pseudo-Code (PARTNER 2) (Partner 2) Before writing out your function, write down pseudo-code of the steps that your function will go through. If you use a loop, discuss what kind of loop you will use and how you will define it.

1. Define the Function I would create a function called scrape_enforcement_actions(month, year) that takes month and year as inputs.
2. Input Validation Check if the year is greater than or equal to 2013: If year < 2013, print a message reminding the user that only actions after 2013 are available and return None. If the year is valid, proceed to the next steps.
3. Initialize Variables Create an empty list enforcement_actions to store dictionaries containing enforcement action data.
4. Loop through Pages Use a while loop to iterate over the pages of the enforcement actions website. Set a variable page_url to the URL of the first page (e.g., "https://oig.hhs.gov/fraud/enforcement/") or the dynamically generated URL for the current page.

Inside the Loop: 1. Scrape Page Content Fetch the page using requests and parse it with BeautifulSoup. Extract each enforcement action's title, date, category, and link. 2. Filter by Date For each action, check if its date is later than or equal to the input month and year. If the date is older, break out of the loop to avoid scraping unnecessary pages. 3. Store Data For each enforcement action that meets the date criteria, add its details to enforcement_actions. 4. Navigate to Next Page Check if there's a "Next" button or link to continue scraping additional pages. If a "Next" page exists, update page_url to the URL of the next page and wait 1 second using time.sleep(1) to prevent server overload. If no "Next" page is found, exit the loop. 5. Crawl Additional Details For each action in enforcement_actions, access its link to scrape the name of the agency involved. Append the agency name to the dictionary associated with each enforcement action. 6. Create and Save DataFrame Convert

enforcement_actions into a Pandas DataFrame. Save the DataFrame as a .csv file named enforcement_actions_year_month.csv based on the input year and month. 7. Return or Print Results Print the first few rows of the DataFrame or return it as an output of the function for verification.

- b. Create Dynamic Scraper (PARTNER 2) Now code up your dynamic scraper and run it to start collecting the enforcement actions since January 2023. How many enforcement actions do you get in your final dataframe? What is the date and details of the earliest enforcement action it scraped?

```
def scrape_enforcement_actions(month, year):
    """Scrape enforcement actions and associated agency information, and save
     ↪  to a CSV file."""

    if year < 2013:
        print("Only enforcement actions from 2013 onwards are available.
         ↪  Please enter a year >= 2013.")
        return None

    base_url = "https://oig.hhs.gov/fraud/enforcement/?page="
    start_date = datetime(year, month, 1)
    all_actions = []
    max_workers = 10
    page_num = 1
    extra_pages_after_stop = 15
    stop_scraping = False
    pages_scraped_after_stop = 0

    def scrape_page(page_num):
        page_url = base_url + str(page_num)
        actions = []
        try:
            response = requests.get(page_url, timeout=10)
            response.raise_for_status()
            soup = BeautifulSoup(response.text, 'html.parser')

            main_tag = soup.find('main')
            li_with_div = main_tag.find_all(lambda t: t.name == 'li' and
 ↪  t.find('div'))

            if not li_with_div:
                return actions, True
```

```
            for li in li_with_div:
                li_soup = BeautifulSoup(''.join(str(item) for item in
↪  li.contents), 'html.parser')

                title_tag = li_soup.select_one('h2 a')
                date_tag = li_soup.select_one('span')
                category_tag = li_soup.select_one('ul li')

                title = title_tag.get_text(strip=True) if title_tag else None
                link = title_tag['href'] if title_tag else None
                date_str = date_tag.get_text(strip=True) if date_tag else
↪  None
                category = category_tag.get_text(strip=True) if category_tag
↪  else None

                try:
                    date = datetime.strptime(date_str, "%B %d, %Y")
                except ValueError:
                    continue
                print('date', date)

                if date < start_date:
                    return actions, "STOP"

                agency_name =
↪  scrape_agency_name(f"https://oig.hhs.gov{link}") if link else "NA"

                actions.append({
                    'Title': title,
                    'Link': f"https://oig.hhs.gov{link}" if link else None,
                    'Date': date,
                    'Category': category,
                    'Agency': agency_name
                })

            return actions, "CONTINUE"
        except Exception as e:
            print(f"Error scraping page {page_num}: {e}")
            return [], "CONTINUE"

    def scrape_agency_name(link):
        """Scrape agency name from the enforcement action detail page."""
```

```
        try:
            response = requests.get(link, timeout=10)
            response.raise_for_status()
            soup = BeautifulSoup(response.content, 'html.parser')

            li_tags = soup.find_all('li')
            for li in li_tags:
                agency_span = li.find('span')
                if agency_span and "Agency" in agency_span.text:
                    agency_name = agency_span.find_next_sibling(text=True)
                    if agency_name:
                        return agency_name.strip()
            return "NA"
        except requests.exceptions.RequestException as e:
            print(f"Retrying for agency details: {e}")
            return "NA"

    while not stop_scraping:
        time.sleep(1)
        pages_to_scrape = list(range(page_num, page_num + max_workers))
        with concurrent.futures.ThreadPoolExecutor(max_workers=max_workers)
         ↪  as executor:
            future_to_page = {executor.submit(scrape_page, p): p for p in
↪   pages_to_scrape}
            for future in concurrent.futures.as_completed(future_to_page):
                page = future_to_page[future]
                try:
                    actions, stop_flag = future.result()
                    all_actions.extend(actions)

                    if stop_flag == "STOP":
                        pages_scraped_after_stop += 1
                        if pages_scraped_after_stop >=
                         ↪  extra_pages_after_stop:
                            stop_scraping = True
                            break
                except Exception as e:
                    print(f"Error processing page {page}: {e}")
        page_num += max_workers

    """Create a DataFrame from collected enforcement actions"""
    enforcement_df = pd.DataFrame(all_actions)
```

```
    """Save DataFrame to CSV"""
    current_date = datetime.now()
    year_month = current_date.strftime("%Y_%m")
    filename = f"enforcement_actions_{year_month}.csv"
    output_path = os.path.join('.', filename)
    enforcement_df.to_csv(output_path, index=False)

    print(f"CSV file saved as: {output_path}")
    return enforcement_df

enforcement_df = scrape_enforcement_actions(1, 2023)
print("Number of enforcement actions collected:", len(enforcement_df))
if not enforcement_df.empty:
    print("Earliest enforcement action:",
     ↪  enforcement_df.sort_values(by='Date').iloc[0])


#used chatgpt to debug because it was taking ages to run the function
#https://docs.python.org/3/library/concurrent.futures.html
#https://www.geeksforgeeks.org/how-to-use-threadpoolexecutor-in-python3/
#used chatgpt to fingure out concurrent.futures
```

```
"""Above 2023 Function Converted Into A CSV File"""
enforcement_df_path = os.path.join(path, 'enforcement_actions_2023_1.csv')
enforcement_df = pd.read_csv(enforcement_df_path)
print("Number of enforcement actions collected:", len(enforcement_df))
if not enforcement_df.empty:
    print("Earliest enforcement action:",
     ↪  enforcement_df.sort_values(by='Date').iloc[0])
```

```
Number of enforcement actions collected: 1534
Earliest enforcement action: Title       Podiatrist Pays $90,000 To Settle
False Billin...
Link        https://oig.hhs.gov/fraud/enforcement/podiatri...
Date                                                2023-01-03
Category                           Criminal and Civil Actions
Agency      U.S. Attorney's Office, Southern District of T...
Name: 1413, dtype: object
```

- c. Test Partner's Code (PARTNER 1) Now, let's go a little further back. Test your partner's code by collecting the actions since January 2021. Note that this can

take a while. How many enforcement actions do you get in your final dataframe? What is the date and details of the earliest enforcement action it scraped? Use the dataframe from this process for every question after this.

```
def scrape_enforcement_actions(month, year):
    """Scrape enforcement actions and associated agency information, and save
     ↪  to a CSV file."""

    if year < 2013:
        print("Only enforcement actions from 2013 onwards are available.
         ↪  Please enter a year >= 2013.")
        return None

    base_url = "https://oig.hhs.gov/fraud/enforcement/?page="
    start_date = datetime(year, month, 1)
    all_actions = []
    max_workers = 10
    page_num = 1
    extra_pages_after_stop = 15
    stop_scraping = False
    pages_scraped_after_stop = 0

    def scrape_page(page_num):
        page_url = base_url + str(page_num)
        actions = []
        try:
            response = requests.get(page_url, timeout=10)
            response.raise_for_status()
            soup = BeautifulSoup(response.text, 'html.parser')

            main_tag = soup.find('main')
            li_with_div = main_tag.find_all(lambda t: t.name == 'li' and
 ↪  t.find('div'))

            if not li_with_div:
                return actions, True

            for li in li_with_div:
                li_soup = BeautifulSoup(''.join(str(item) for item in
 ↪  li.contents), 'html.parser')

                title_tag = li_soup.select_one('h2 a')
```

```
                date_tag = li_soup.select_one('span')
                category_tag = li_soup.select_one('ul li')

                title = title_tag.get_text(strip=True) if title_tag else None
                link = title_tag['href'] if title_tag else None
                date_str = date_tag.get_text(strip=True) if date_tag else
↪ None
                category = category_tag.get_text(strip=True) if category_tag
↪ else None

                try:
                    date = datetime.strptime(date_str, "%B %d, %Y")
                except ValueError:
                    continue
                print('date', date)

                if date < start_date:
                    return actions, "STOP"

                agency_name =
↪ scrape_agency_name(f"https://oig.hhs.gov{link}") if link else "NA"

                actions.append({
                    'Title': title,
                    'Link': f"https://oig.hhs.gov{link}" if link else None,
                    'Date': date,
                    'Category': category,
                    'Agency': agency_name
                })

            return actions, "CONTINUE"
        except Exception as e:
            print(f"Error scraping page {page_num}: {e}")
            return [], "CONTINUE"

    def scrape_agency_name(link):
        """Scrape agency name from the enforcement action detail page."""
        try:
            response = requests.get(link, timeout=10)
            response.raise_for_status()
            soup = BeautifulSoup(response.content, 'html.parser')
```

```
            li_tags = soup.find_all('li')
            for li in li_tags:
                agency_span = li.find('span')
                if agency_span and "Agency" in agency_span.text:
                    agency_name = agency_span.find_next_sibling(text=True)
                    if agency_name:
                        return agency_name.strip()
            return "NA"
        except requests.exceptions.RequestException as e:
            print(f"Retrying for agency details: {e}")
            return "NA"

    while not stop_scraping:
        time.sleep(1)
        pages_to_scrape = list(range(page_num, page_num + max_workers))
        with concurrent.futures.ThreadPoolExecutor(max_workers=max_workers)
         ↪ as executor:
            future_to_page = {executor.submit(scrape_page, p): p for p in
↪   pages_to_scrape}
            for future in concurrent.futures.as_completed(future_to_page):
                page = future_to_page[future]
                try:
                    actions, stop_flag = future.result()
                    all_actions.extend(actions)

                    if stop_flag == "STOP":
                        pages_scraped_after_stop += 1
                        if pages_scraped_after_stop >=
                         ↪ extra_pages_after_stop:
                            stop_scraping = True
                            break
                except Exception as e:
                    print(f"Error processing page {page}: {e}")
        page_num += max_workers

    """Create a DataFrame from collected enforcement actions"""
    enforcement_df = pd.DataFrame(all_actions)

    """Save DataFrame to CSV"""
    current_date = datetime.now()
    year_month = current_date.strftime("%Y_%m")
    filename = f"enforcement_actions_{year_month}.csv"
```

```
    output_path = os.path.join('.', filename)
    enforcement_df.to_csv(output_path, index=False)

    print(f"CSV file saved as: {output_path}")
    return enforcement_df

enforcement_df = scrape_enforcement_actions(1, 2021)
print("Number of enforcement actions collected:", len(enforcement_df))
if not enforcement_df.empty:
    print("Earliest enforcement action:",
     ↪  enforcement_df.sort_values(by='Date').iloc[0])
```

```
"""Above 2023 Function Converted Into A CSV File"""
enforcement_df_path = os.path.join(path, 'enforcement_actions_2021_1.csv')
enforcement_df = pd.read_csv(enforcement_df_path)
```

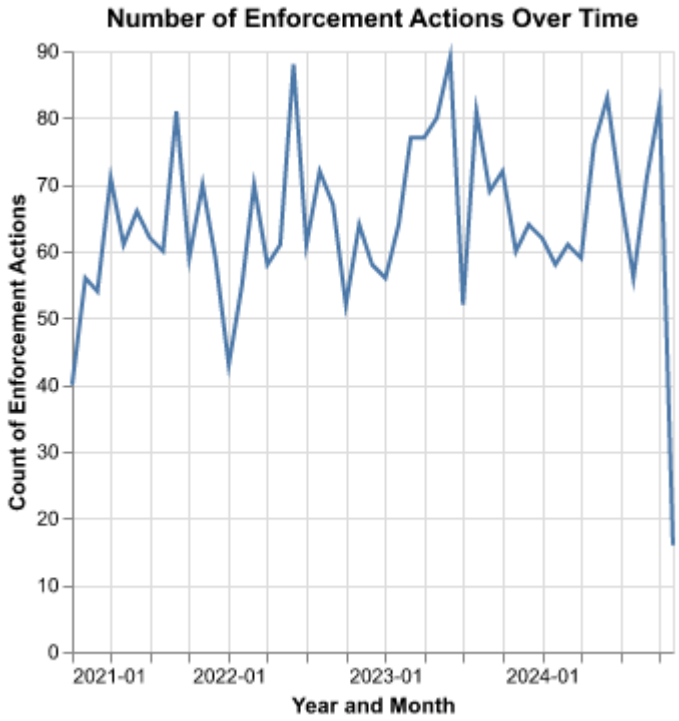
Step 3: Plot data based on scraped data

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

1. (Partner 2) Plot a line chart that shows: the number of enforcement actions over time (aggregated to each month+year) overall since January 2021,

```
enforcement_df['Date'] = pd.to_datetime(enforcement_df['Date'])

enforcement_df['YearMonth'] = enforcement_df['Date'].dt.to_period('M')
enforcement_counts =
 ↪  enforcement_df.groupby('YearMonth').size().reset_index(name='Count')

enforcement_counts['YearMonth'] =
 ↪  enforcement_counts['YearMonth'].dt.to_timestamp()

"""Plotting with Altair"""
chart = alt.Chart(enforcement_counts).mark_line().encode(
    x=alt.X('YearMonth:T', title='Year and Month',
 ↪  axis=alt.Axis(format='%Y-%m')),
    y=alt.Y('Count:Q', title='Count of Enforcement Actions')
).properties(
    title='Number of Enforcement Actions Over Time'
)
```

```
chart.show()

#source: https://docs.python.org/3/library/datetime.html
#https://stackoverflow.com/questions/70875690/change-date-axis-ticks-in-altair-to-show-years
```

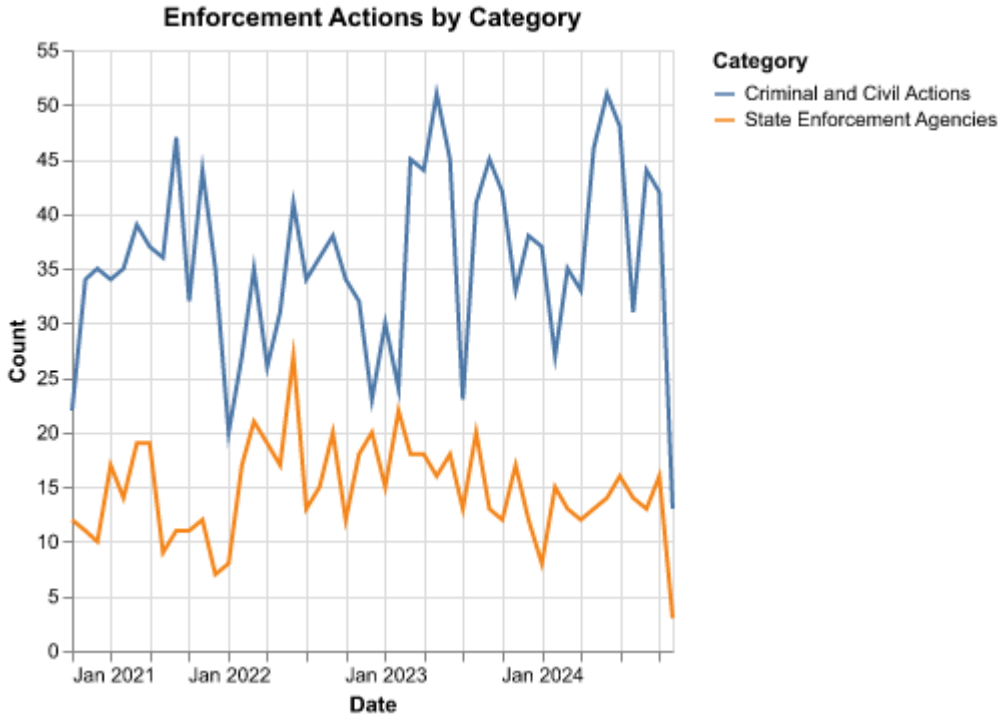


2. Plot the number of enforcement actions categorized: (PARTNER 1)

2. (Partner 1) Plot a line chart that shows: the number of enforcement actions split out by:
   - "Criminal and Civil Actions" vs. "State Enforcement Agencies"

```
"""Creating two categories for data"""
enforcement_categories = ["Criminal and Civil Actions", "State Enforcement
 ↪  Agencies"]
filtered_df =
 ↪  enforcement_df[enforcement_df['Category'].isin(enforcement_categories)]

"""Change to datetime and groupby date"""
filtered_df['Date'] = pd.to_datetime(filtered_df['Date'])
```

```
actions_by_date = filtered_df.groupby([filtered_df['Date'].dt.to_period('M'),
 ↪  'Category']).size().reset_index(name='Count')
actions_by_date['Date'] = actions_by_date['Date'].dt.to_timestamp()

"""Altair lineplot"""
chart = alt.Chart(actions_by_date).mark_line().encode(
    x=alt.X('Date:T', axis=alt.Axis(format='%b %Y')),
    y='Count:Q',
    color='Category:N',
).properties(
    title='Enforcement Actions by Category'
)

chart.show()
```



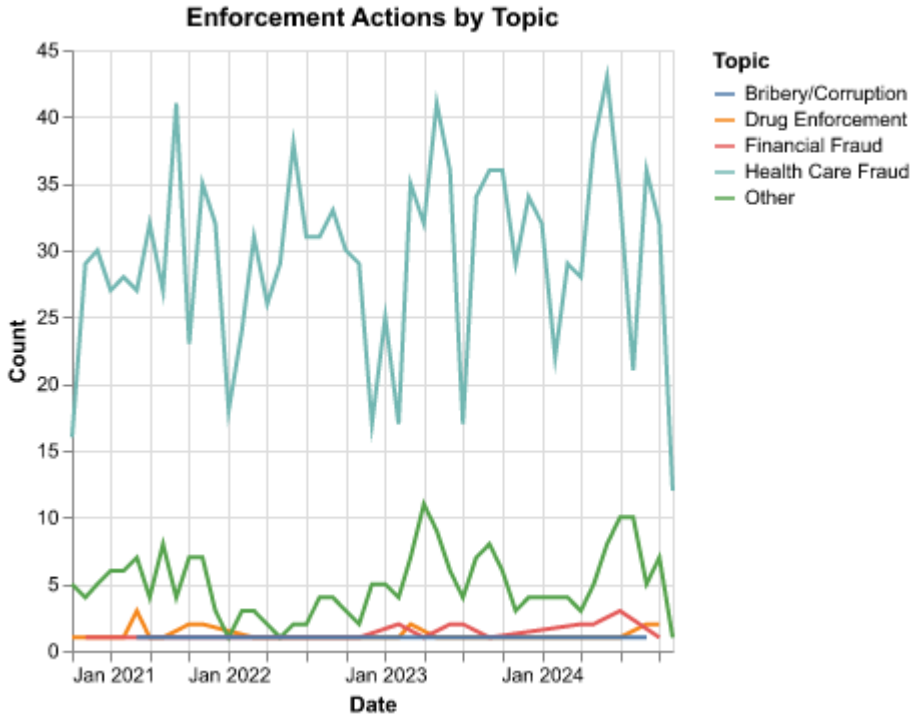
• Five topics in the "Criminal and Civil Actions" category: "Health Care Fraud", "Financial Fraud", "Drug Enforcement", "Bribery/Corruption", and "Other". Hint: You will need to divide the five topics manually by looking at the title and assigning the relevant topic. For example, if you find the word "bank" or "financial" in the title of an action, then that action should probably belong to "Financial Fraud" topic.

```
"""Filtering category"""
cc_actions = enforcement_df[enforcement_df['Category'] == 'Criminal and Civil
 ↪  Actions']

keywords = {
    "Health Care Fraud": ["health", "healthcare", "medicaid", "medicare",
     ↪  "medical", "medication", "hospital", "pharmacy", "pharmaceutical",
     ↪  "pharmacist",  "doctor", "insurance", "covid", "physician",
     ↪  "prescription", "kickback", "surgeon", "nurse", "nursing", "clinic"],
    "Financial Fraud": ["bank", "financial", "finance", "capital",
     ↪  "fraudulant", "investment", "loan", "credit", "tax evasion",
     ↪  "embezzlement", "laundering", "money laundering", "tax fraud",
     ↪  "insider trading", "grants", "assets", "identity theft", "consumer",
     ↪  "credit card", "chargeback"],
    "Drug Enforcement": ["drug", "addiction", "opioid", "narcotic",
     ↪  "fentanyl", "cocaine", "substance", "marijuana", "DEA"],
    "Bribery/Corruption": ["bribery", "bribe", "corruption", "corrupt",
     ↪  "payoff", "extort", "payment"],
    "Other": []
}

"""Function to assign topic based on title"""
def assign_topic(title):
    title = title.lower()
    for topic, words in keywords.items():
        if any(word in title for word in words):
            return topic
    return "Other"

cc_actions['Topic'] = cc_actions['Title'].apply(assign_topic)
print(cc_actions[['Title', 'Topic']].head(5))
```

```
                                               Title              Topic
0  South Dakota Surgical Hospital Agrees To Pay M...  Health Care Fraud
3  Licensed Professional Counselor Indicted In $2...  Health Care Fraud
4  California Addiction Treatment Facility Operat...  Health Care Fraud
5  Walgreens Agrees To Pay $106.8M To Resolve All...  Health Care Fraud
8  Final Medoc Defendant Sentenced In $4.4 Millio...  Health Care Fraud
```

```
"""Change to datetime"""
cc_actions['Date'] = pd.to_datetime(cc_actions['Date'])
```

```
"""Group by topics and date"""
cc_topics = ["Health Care Fraud", "Financial Fraud", "Drug Enforcement",
 ↪  "Bribery/Corruption", "Other"]
filtered_cc_actions = cc_actions[cc_actions['Topic'].isin(cc_topics)]
actions_by_date =
 ↪  filtered_cc_actions.groupby([cc_actions['Date'].dt.to_period('M'),
 ↪  'Topic']).size().reset_index(name='Count')
actions_by_date['Date'] = actions_by_date['Date'].dt.to_timestamp()

"""Altair lineplot"""
chart = alt.Chart(actions_by_date).mark_line().encode(
    x=alt.X('Date:T', axis=alt.Axis(format='%b %Y')),
    y='Count:Q',
    color='Topic:N',
).properties(
    title='Enforcement Actions by Topic'
)

chart.show()
```



Step 4: Create maps of enforcement activity

For these questions, use this US Attorney District shapefile (link) and a Census state shapefile (link)

1. Map by State (PARTNER 1)

(Partner 1) Map by state: Among actions taken by state-level agencies, clean the state names you collected and plot a choropleth of the number of enforcement actions for each state. Hint: look for "State of" in the agency info!

```
"""Load Census state shapefile"""
census_state_folder = 'cb_2018_us_state_500k'
census_state_path = os.path.join(path, census_state_folder,
 ↪  'cb_2018_us_state_500k.shp')
states_shapefile = gpd.read_file(census_state_path)

states_shapefile.head()
```

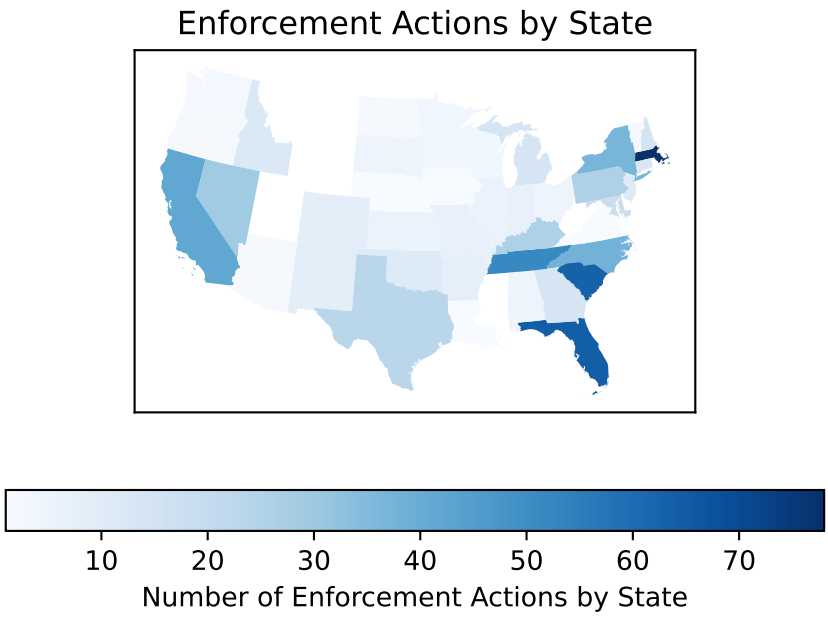
	STATEFP	STATENS	AFFGEOID	GEOID	STUSPS	NAME	LSAD	ALAND	A
0	28	01779790	0400000US28	28	MS	Mississippi	00	121533519481	3
1	37	01027616	0400000US37	37	NC	North Carolina	00	125923656064	1
2	40	01102857	0400000US40	40	OK	Oklahoma	00	177662925723	3
3	51	01779803	0400000US51	51	VA	Virginia	00	102257717110	8
4	54	01779805	0400000US54	54	WV	West Virginia	00	62266474513	4

```
state_name_to_abbr = {
    'Alabama': 'AL', 'Alaska': 'AK', 'Arizona': 'AZ', 'Arkansas': 'AR',
    'California': 'CA', 'Colorado': 'CO', 'Connecticut': 'CT', 'Delaware':
     ↪  'DE',
    'Florida': 'FL', 'Georgia': 'GA', 'Hawaii': 'HI', 'Idaho': 'ID',
    'Illinois': 'IL', 'Indiana': 'IN', 'Iowa': 'IA', 'Kansas': 'KS',
    'Kentucky': 'KY', 'Louisiana': 'LA', 'Maine': 'ME', 'Maryland': 'MD',
    'Massachusetts': 'MA', 'Michigan': 'MI', 'Minnesota': 'MN',
     ↪  'Mississippi': 'MS',
    'Missouri': 'MO', 'Montana': 'MT', 'Nebraska': 'NE', 'Nevada': 'NV',
    'New Hampshire': 'NH', 'New Jersey': 'NJ', 'New Mexico': 'NM', 'New
     ↪  York': 'NY',
    'North Carolina': 'NC', 'North Dakota': 'ND', 'Ohio': 'OH', 'Oklahoma':
     ↪  'OK',
```

```
    'Oregon': 'OR', 'Pennsylvania': 'PA', 'Rhode Island': 'RI', 'South
    ↪ Carolina': 'SC',
    'South Dakota': 'SD', 'Tennessee': 'TN', 'Texas': 'TX', 'Utah': 'UT',
    'Vermont': 'VT', 'Virginia': 'VA', 'Washington': 'WA', 'West Virginia':
    ↪ 'WV',
    'Wisconsin': 'WI', 'Wyoming': 'WY'
}

"""Extract state abbreviation from a state name found in the agency
↪ information"""
def extract_state_abbr(agency_name):
    if not isinstance(agency_name, str):
        return None
    if 'district' in agency_name.lower():
        return None
    for state_name in state_name_to_abbr:
        pattern = r'\b' + re.escape(state_name) + r'\b'
        match = re.search(pattern, agency_name, re.IGNORECASE)
        if match:
            return state_name_to_abbr[state_name]
    return None

"""Extract and set state abbreviations bassed on mapping"""
enforcement_df['State_abbr'] =
↪ enforcement_df['Agency'].apply(extract_state_abbr)
missing_states = enforcement_df[enforcement_df['State_abbr'].isna()]
enforcement_df['State_abbr'] = enforcement_df['State_abbr'].fillna('Unknown')
state_counts = enforcement_df['State_abbr'].value_counts().reset_index()

"""Rename and merge on State abbreviations"""
state_counts.columns = ['State_abbr', 'Action_Count']
states_shapefile = states_shapefile.rename(columns={'STUSPS': 'State_abbr'})
merged = states_shapefile.merge(state_counts, on='State_abbr', how='left')
```

```
"""Matplotlib choropleth"""
fig, ax = plt.subplots()
merged = merged.to_crs(epsg=5070)
merged.plot(column='Action_Count', ax=ax, legend=True,
            legend_kwds={'label': "Number of Enforcement Actions by State",
↪ 'orientation': 'horizontal'},
            cmap='Blues')
```

```
ax.set_title('Enforcement Actions by State')
ax.set_xticks([])
ax.set_yticks([])

plt.show()
```



2. Map by District (PARTNER 2)

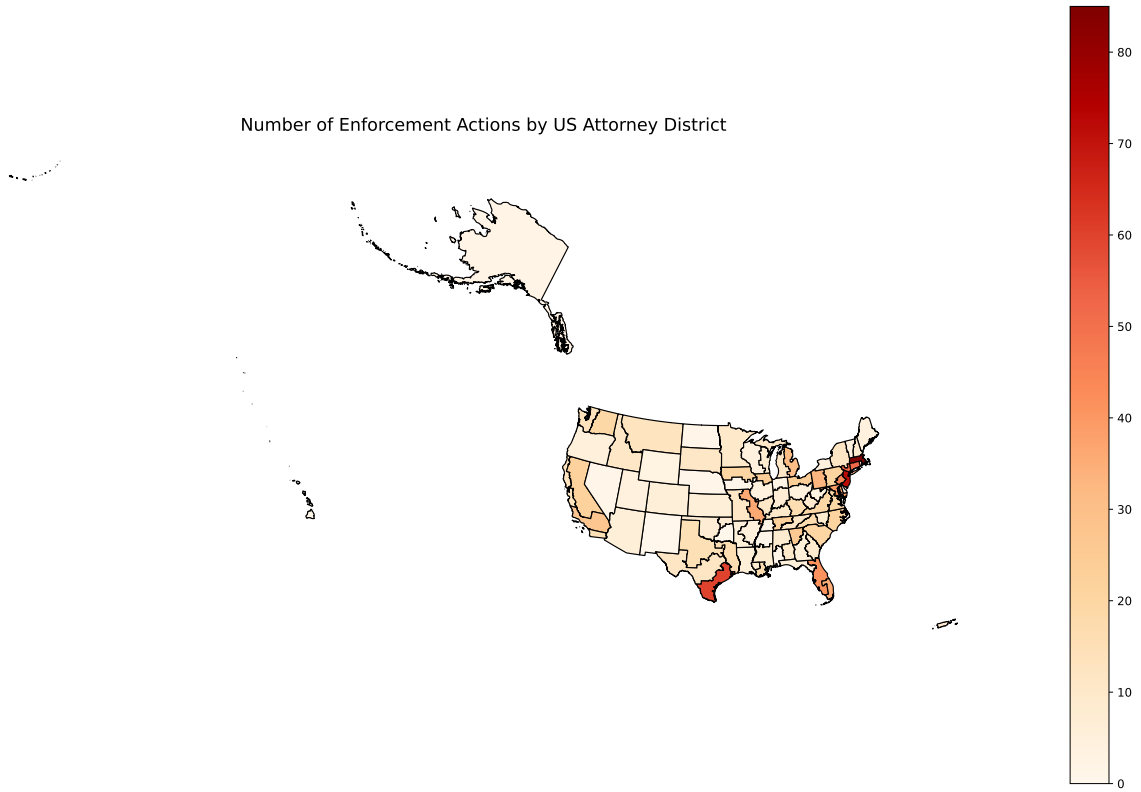
(Partner 2) Map by district: Among actions taken by US Attorney District-level agencies, clean the district names so that you can merge them with the shapefile, and then plot a choropleth of the number of enforcement actions in each US Attorney District. Hint: look for "District" in the agency info.

```
"""US Attorney District shapefile"""
districts_folder = 'US_Attorney_Districts_Shapefile_simplified_20241107'
districts_path = os.path.join(path, districts_folder,
'geo_export_a70ac548-6b1a-49a8-a51c-4c7108365975.shp')
districts_gdf = gpd.read_file(districts_path)
```

```
"""Filter actions taken by US Attorney District-level agencies"""
district_actions =
 ↪  enforcement_df[enforcement_df['Agency'].str.contains('District',
 ↪  case=False, na=False)].copy()

"""Extract and clean district names"""
def extract_district_name(agency):
    if pd.isnull(agency):
        return None
    match =
 ↪  re.search(r'([A-Za-z\s]+District\s+of\s+[A-Za-z\s]+|District\s+of\s+[A-Za-z\s]+)',
 ↪  agency)
    if match:
        district_name = match.group(0).strip()
        return district_name
    else:
        return None

district_actions['District'] =
 ↪  district_actions['Agency'].apply(extract_district_name)
district_actions = district_actions.dropna(subset=['District'])

"""Standardize district names"""
def standardize_district_name(name):
    if pd.isnull(name):
        return None
    name = name.lower().strip()
    name = re.sub(r'(u\.s\.\s+)?district\s+of\s+', '', name)
    name = name.replace('district', '').strip()
    name = re.sub(r'\s+', '_', name)
    name = re.sub(r'[^\w_]', '', name)
    return name

district_actions['District_Clean'] =
 ↪  district_actions['District'].apply(standardize_district_name)

"""Identify the correct column in the shapefile"""
district_column_name = 'judicial_d'
districts_gdf['District_Clean'] =
 ↪  districts_gdf[district_column_name].apply(standardize_district_name)

"""Aggregate counts and merge"""
```

```
district_counts =
 ↪  district_actions.groupby('District_Clean').size().reset_index(name='Count')
merged_gdf = districts_gdf.merge(district_counts, on='District_Clean',
 ↪  how='left')
merged_gdf['Count'] = merged_gdf['Count'].fillna(0).astype(int)

merged_gdf = merged_gdf.to_crs("ESRI:102003")
```

```
"""Plot the choropleth map"""
fig, ax = plt.subplots(1, 1, figsize=(20, 12))

max_count = merged_gdf['Count'].max()
norm = colors.Normalize(vmin=0, vmax=max_count)

"""Plot the merged GeoDataFrame"""
merged_gdf.plot(
    column='Count',
    ax=ax,
    legend=True,
    cmap='OrRd',
    edgecolor='black',
    norm=norm,
    missing_kwds={'color': 'lightgrey', 'hatch': '///'},
)

ax.set_title('Number of Enforcement Actions by US Attorney District',
 ↪  fontsize=15)

ax.axis('off')

plt.show()

#source: asked chatgpt to debug missing values from states since the whole
 ↪  western region was zero for me for a while
#https://stackoverflow.com/questions/20240239/python-re-search
```



Extra Credit

1. Merge zip code shapefile with population

Use the zip code shapefile from the previous problem set and merge it with zip codelevel population data. (Go to Census Data Portal, select "ZIP Code Tabulation Area", check "All 5-digit ZIP Code Tabulation Areas within United States", and under "P1 TOTAL POPULATION" select "2020: DEC Demographic and Housing Characteristics". Download the csv.).

```
"""Zipcode shapefile upload"""
census_zip_folder = 'gz_2010_us_860_00_500k'
census_zip_path = os.path.join(path, census_zip_folder,
 ↪  'gz_2010_us_860_00_500k.shp')
zip_shapefile = gpd.read_file(census_zip_path)
```

```
"""2020: DEC Demographic and Housing Characteristics upload"""
population_data_folder = 'DECENNIALDHC2020.P1_2024-11-08T190918'
population_data_path = os.path.join(path, population_data_folder,
 ↪  'DECENNIALDHC2020.P1-Data.csv')
```

```
population_data = pd.read_csv(population_data_path)
print("Columns in population data:", population_data.columns)
print(population_data.head())
```

```
Columns in population data: Index(['GEO_ID', 'NAME', 'P1_001N', 'Unnamed:
3'], dtype='object')
           GEO_ID                  NAME   P1_001N  Unnamed: 3
0       Geography  Geographic Area Name   !!Total         NaN
1  860Z200US00601            ZCTA5 00601     17242         NaN
2  860Z200US00602            ZCTA5 00602     37548         NaN
3  860Z200US00603            ZCTA5 00603     49804         NaN
4  860Z200US00606            ZCTA5 00606      5009         NaN
```

```
"""Format ZIP codes in population data and rename population column"""
population_data = population_data.drop(index=0)
population_data['ZIP'] = population_data['NAME'].str.replace('ZCTA5 ',
 ↪  '').str.zfill(5)
population_data = population_data[['ZIP', 'P1_001N']]
population_data = population_data.rename(columns={'P1_001N': 'Population'})

zip_shapefile = zip_shapefile.rename(columns={'ZCTA5': 'ZIP'})

"""Merge shapefile and population data"""
merged_zip_data = zip_shapefile.merge(population_data[['ZIP', 'Population']],
 ↪  on='ZIP', how='left')
merged_zip_data['Population'] =
 ↪  merged_zip_data['Population'].fillna(0).astype(int)


"""Display results and summary"""
print("Unmatched ZIPs after merge (first 10):")
print(merged_zip_data[merged_zip_data['Population'] == 0][['ZIP']].head(10))
print("\nSample of merged_zip_data (first 5 rows):")
print(merged_zip_data.head())
print("\nSummary of 'Population' in merged data after handling missing
 ↪  values:")
print(merged_zip_data['Population'].describe())
```

```
Unmatched ZIPs after merge (first 10):
        ZIP
490   04629
494   04637
```

```
497   04644
637   05657
1156  11351
1157  11359
1158  11371
1161  11425
1250  08808
1274  10020

Sample of merged_zip_data (first 5 rows):
           GEO_ID    ZIP   NAME   LSAD  CENSUSAREA  \
0  8600000US01040  01040  01040  ZCTA5      21.281
1  8600000US01050  01050  01050  ZCTA5      38.329
2  8600000US01053  01053  01053  ZCTA5       5.131
3  8600000US01056  01056  01056  ZCTA5      27.205
4  8600000US01057  01057  01057  ZCTA5      44.907

                                             geometry  Population
0  POLYGON ((-72.62734 42.16203, -72.62764 42.162...       38238
1  POLYGON ((-72.95393 42.34379, -72.95385 42.343...        2467
2  POLYGON ((-72.68286 42.37002, -72.68287 42.369...        2031
3  POLYGON ((-72.39529 42.18476, -72.39653 42.183...       21002
4  MULTIPOLYGON (((-72.39191 42.08066, -72.39077 ...        8152

Summary of 'Population' in merged data after handling missing values:
count     33120.000000
mean      10077.890066
std       14996.831089
min           0.000000
25%         698.750000
50%        2771.000000
75%       13787.250000
max      135256.000000
Name: Population, dtype: float64
```

2. Conduct spatial join

Conduct a spatial join between zip code shapefile and the district shapefile, then aggregate to get population in each district. by_district.head())

3. Map the action ratio in each district

Map the ratio of enforcement actions in each US Attorney District. You can calculate the ratio by aggregating the number of enforcement actions since January 2021 per district, and dividing it with the population data.