30538 Problem Set 5: Web Scraping

Alison Filbey and Claire Conzelmann

2024-11-09

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): Alison Filbey afilbey • Partner 2 (name and cnet ID): Claire Conzelmann cconzelmann
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: **AF CC**
5. "I have uploaded the names of anyone else other than my partner and I worked with on the problem set here" (1 point)**AF CC**
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 Gradescop

Step 1: Develop initial scraper and crawler

```
#libraries
import pandas as pd
import altair as alt
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin
```

```
import shapely
import geopandas as gpd
import time
from datetime import datetime
from concurrent.futures import ThreadPoolExecutor
import numpy as np
import matplotlib.pyplot as plt
alt.renderers.enable("png")
```

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

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 Collect your output into a tidy dataframe and print its head.

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

#Scraping the title and appending it to a list, subsetting to headers we need
h2 = soup.find_all('h2')
title = []
for h2 in h2:
   title.append(h2.get_text(strip=True))
title=title[2:22]

#Scraping the date and appending it to a list
span = soup.find_all('span', class_='text-base-dark padding-right-105')
date = []
for span in span:
   date.append(span.get_text(strip=True))

#Scraping the classification and appending it to a list
listed = soup.find_all('li', class_='display-inline-block usa-tag
 ↪  text-no-lowercase text-base-darkest bg-base-lightest margin-right-1')
classification = []
for element in listed:
    text = element.get_text(strip=True)
    classification.append(str(text))
```

```
#Scraping the link and appending it to a list, focusing on links we need
a_tags = soup.find_all('a')
links = []
for a in a_tags:
  links.append(a['href'])

links=links[74:94]
short_url = 'https://oig.hhs.gov'
links = [short_url + link for link in links]

#Creating a dataframe from the lists
enforcement_actions = {
    'Title': title,
    'Date': date,
    'Classification': classification,
    'URL': links
}

enforcement_actions = pd.DataFrame(enforcement_actions)
print(enforcement_actions.head())
```

```
                                               Title              Date  \
0  Pharmacist and Brother Convicted of $15M Medic...  November 8, 2024
1  Boise Nurse Practitioner Sentenced To 48 Month...  November 7, 2024
2  Former Traveling Nurse Pleads Guilty To Tamper...  November 7, 2024
3  Former Arlington Resident Sentenced To Prison ...  November 7, 2024
4  Paroled Felon Sentenced To Six Years For Fraud...  November 7, 2024

              Classification  \
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

                                                 URL
0  https://oig.hhs.gov/fraud/enforcement/pharmaci...
1  https://oig.hhs.gov/fraud/enforcement/boise-nu...
2  https://oig.hhs.gov/fraud/enforcement/former-t...
3  https://oig.hhs.gov/fraud/enforcement/former-a...
4  https://oig.hhs.gov/fraud/enforcement/paroled-...
```

2. 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

```
#Creating a loop that goes through each URL and scrapes the agency
agency = []
for crawl_url in enforcement_actions['URL']:
#Retreving the URL and finding the Agency span
  response = requests.get(crawl_url)
  soup = BeautifulSoup(response.text, 'lxml')
  span = soup.find('span', string=lambda text: text and 'Agency:' in text)
#For the agency span, find the test that is next to it and append it
#Asked chatgpt for help finding a function that would find the next tag next
 ↪  to
# "Agency"
  if span:
    agency_name = span.find_next_sibling(text=True).strip()
    agency.append(agency_name)
  else:
    agency_name = None
    agency.append(agency_name)

enforcement_actions['Agency'] = agency
print(enforcement_actions.head())
```

```
C:\Users\aliso\AppData\Local\Temp\ipykernel_4164\1179023401.py:12:
DeprecationWarning: The 'text' argument to find()-type methods is deprecated.
Use 'string' instead.
  agency_name = span.find_next_sibling(text=True).strip()
```

	Title	Date	Classification
0	Pharmacist and Brother Convicted of $15M Medic...	November 8, 2024	Criminal and Civil Actions
1	Boise Nurse Practitioner Sentenced To 48 Month...	November 7, 2024	Criminal and Civil Actions
2	Former Traveling Nurse Pleads Guilty To Tamper...	November 7, 2024	Criminal and Civil Actions
3	Former Arlington Resident Sentenced To Prison ...	November 7, 2024	Criminal and Civil Actions
4	Paroled Felon Sentenced To Six Years For Fraud...	November 7, 2024	Criminal and Civil Actions

```
                                                  URL  \
0  https://oig.hhs.gov/fraud/enforcement/pharmaci...
1  https://oig.hhs.gov/fraud/enforcement/boise-nu...
2  https://oig.hhs.gov/fraud/enforcement/former-t...
3  https://oig.hhs.gov/fraud/enforcement/former-a...
4  https://oig.hhs.gov/fraud/enforcement/paroled-...

                                              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

1. Turning the scraper into a function

- a. Pseudo-Code (PARTNER 2)

1. check if year inputted is < 2013

- if <2013, print "Invalid year." (use an if statement here)

2. Set base url (without the page info), base page=1, and empty lists to be appended with each iteration
3. Begin a while loop

- while loop will continue looping through pages and exit once there are no more pages to scrape

4. Go through first page, extract soup and all relavant tags
5. Use for loop to extract each item in the find_all tag lists and append to empty lists created before
6. Check if date of given item is outside of the range inputted in the function

- if it is, exit out of the function

7. Add 1 to page and repeat

- b. Create Dynamic Scraper (PARTNER 2) I originally included the crawling in my for loop within the function. This was taking a really long time to run. I pasted my code into ChatGPT and asked how to make it run faster. It suggested I create a separate function for the crawling and use ThreadPoolExecutor.

```
#function to crawl extracted links and get agency name
def fetch_agency_name(link):
    """
    Takes a link as an input and fetches the agency name from the given link
    """

    #get url to craawl
    short_url = "https://oig.hhs.gov"
    if not link.startswith("http"):
        full_link = short_url + link
    else:
        full_link = link

    response = requests.get(full_link)

    #crawl the link that is extracted on the main page and extract agency
     ↪ name
    soup_crawl = BeautifulSoup(response.text, "lxml")
    span_crawl = soup_crawl.find("span",
      string=lambda text: text and "Agency:" in text)
    return span_crawl.find_next_sibling(string=True).strip() if span_crawl
     ↪ else ""

#main scraping function to scrape oig website through specified date
def dynamic_scraper(year, month):

  #create datetime object from year and month
  date_obj = datetime(year, month, 1)

  #first confirm year is in range
  if year < 2013:
    print("Invalid year. Please enter a year that is 2013 or later.")
    return

  #set base url and first page
  base_url = "https://oig.hhs.gov/fraud/enforcement/"
  page = 1
```

```
#set empty lists to store results
title = []
date = []
classification = []
links = []
agency = []

#set short url to append to url in loop
short_url = 'https://oig.hhs.gov'

#create soup object from given page
while True:
  url = base_url + "?page=" + str(page)
  response = requests.get(url)
  soup = BeautifulSoup(response.content, 'html.parser')

  #get out of loop if there is no page to scrape
  if not soup:
    break

  #find all necessary tags on given page
  h2_tags = soup.find_all("h2")[2:]
  span_tags = soup.find_all("span", class_="text-base-dark
↪ padding-right-105")
  li_tags = soup.find_all("li",
    class_="display-inline-block usa-tag text-no-lowercase
↪ text-base-darkest bg-base-lightest margin-right-1")
  a_tags = soup.find_all("a")[74:74+len(span_tags)]

  for i in range(len(span_tags)):
    #turn date string into datetime object
    full_date = datetime.strptime(span_tags[i].get_text(strip=True), "%B
↪ %d, %Y")

    #stop scraping if date is outside the range
    if full_date < date_obj:

      #fetch agency names before exiting the loop
      with ThreadPoolExecutor(max_workers=5) as executor:
        agency = list(executor.map(fetch_agency_name, links))
```

```
        #write dataframe
        df = pd.DataFrame({"Title": title,
                           "Date": date,
                           "Classification": classification,
                           "URL": links,
                           "Agency" : agency})

        #write csv
        file_name = "Data/enforcement_actions_" + str(year) + "_" +
↪   str(month) + ".csv"
        df.to_csv(file_name)

        #exit function
        return df

      #extract date and append to date column
      date.append(span_tags[i].get_text(strip=True))

      #extract data from other tags and append to respective columns
      title.append(h2_tags[i].get_text(strip=True))

      text = li_tags[i].get_text(strip=True)
      classification.append(str(text))

      if 'href' in a_tags[i].attrs and a_tags[i]['href']:
        link = a_tags[i]['href']
        link = short_url + link
        links.append(link)

    page += 1

    #delay
    time.sleep(1)

  #fetch agency names in parallel
  with ThreadPoolExecutor(max_workers=5) as executor:
    agency = list(executor.map(fetch_agency_name, links))

  #write dataframe
  df = pd.DataFrame({"Title": title,
                     "Date": date,
                     "Classification": classification,
```

```
                    "URL": links,
                    "Agency" : agency})

  #write csv
  file_name = "Data/enforcement_actions_" + str(year) + "_" + str(month) +
↪   ".csv"
  df.to_csv(file_name)

  return df
```

```
enforcement_actions_012023 = dynamic_scraper(2023, 1)
```

```
#get info on tested function
print(len(enforcement_actions_012023))

#convert date to datetime so we can calculate the minimum
enforcement_actions_012023["Date_obj"] = pd.to_datetime(
  enforcement_actions_012023["Date"], format="%B %d, %Y")

min_date = min(enforcement_actions_012023["Date_obj"])

print(enforcement_actions_012023.loc[enforcement_actions_012023["Date_obj"]
↪   == min_date])
```

```
1534
                                                  Title             Date  \
1533  Podiatrist Pays $90,000 To Settle False Billin...  January 3, 2023

                Classification  \
1533  Criminal and Civil Actions

                                                    URL  \
1533  https://oig.hhs.gov/fraud/enforcement/podiatri...

                                                 Agency   Date_obj
1533  U.S. Attorney's Office, Southern District of T... 2023-01-03
```

The number of enforcement actions in the final dataframe is shown above. The earliest enforcement action scraped was from January 3, 2023, and the title, classification, agency, and url that was scraped for this enforcement action is listed above.

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

```
#test dynamic scraper
enforcement_actions_012021 = dynamic_scraper(2021, 1)
```

```
enforcement_actions_012021 =
 ↪  pd.read_csv('Data/enforcement_actions_2021_1.csv')
```

```
#get info on tested function
print(len(enforcement_actions_012021))

enforcement_actions_012021["Date_obj"] = pd.to_datetime(
  enforcement_actions_012021["Date"], format="%B %d, %Y")

min_date = min(enforcement_actions_012021["Date_obj"])

print(enforcement_actions_012021.loc[enforcement_actions_012021["Date_obj"]
 ↪  == min_date])
```

```
3022
      Unnamed: 0                                              Title  \
3021        3021  The United States And Tennessee Resolve Claims...

                 Date              Classification  \
3021  January 4, 2021  Criminal and Civil Actions

                                                   URL  \
3021  https://oig.hhs.gov/fraud/enforcement/the-unit...

                                                Agency   Date_obj
3021  U.S. Attorney's Office, Middle District of Ten... 2021-01-04
```

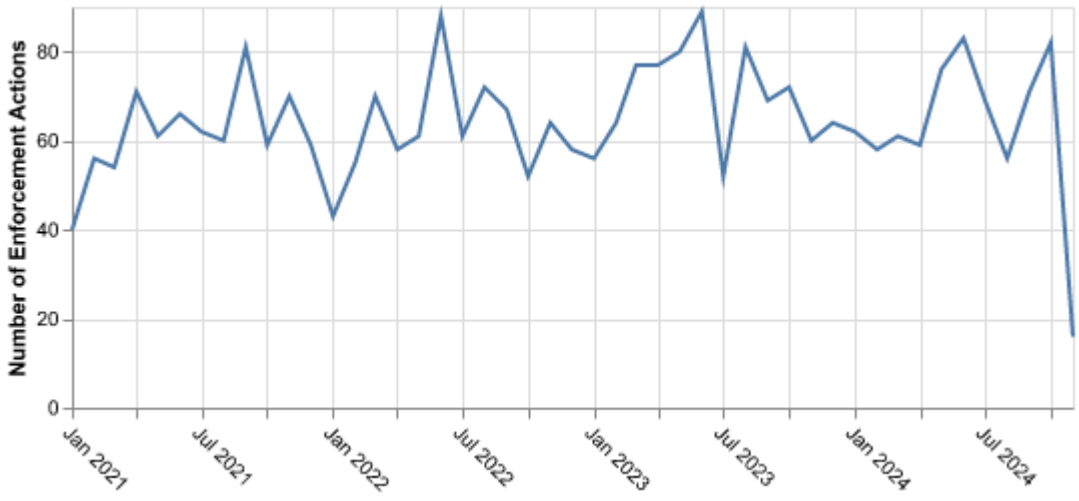
There are 2,998 enforcement actions in the final dataframe. The earliest enforcement action scraped was from January 4, 2021, and the title, classification, and url that was scraped for this enforcement action is listed above.

Step 3: Plot data based on scraped data

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

```
#create year and month variables
enforcement_actions_012021["year"] =
 ↪  enforcement_actions_012021["Date_obj"].dt.year
enforcement_actions_012021["month"] =
 ↪  enforcement_actions_012021["Date_obj"].dt.month

#group by yearmonth and calculate number of enforcement actions
enforcement_actions_grouped = enforcement_actions_012021.groupby(
  ["year", "month"]).size().reset_index(name="n_enforcements")

#create date variable from month and year
enforcement_actions_grouped["date"] = pd.to_datetime(
  enforcement_actions_grouped["year"].astype(str) +
  "-" + enforcement_actions_grouped["month"].astype(str) + "-01")
```

```
#plot number of enforcements over time
alt.Chart(enforcement_actions_grouped).mark_line().encode(
    alt.X("yearmonth(date):T",
        axis=alt.Axis(
          title="",
          labelAngle=45,)),
    alt.Y("n_enforcements:Q",
        axis=alt.Axis(title="Number of Enforcement Actions"))
).properties(
    height=200,
    width=500)
```

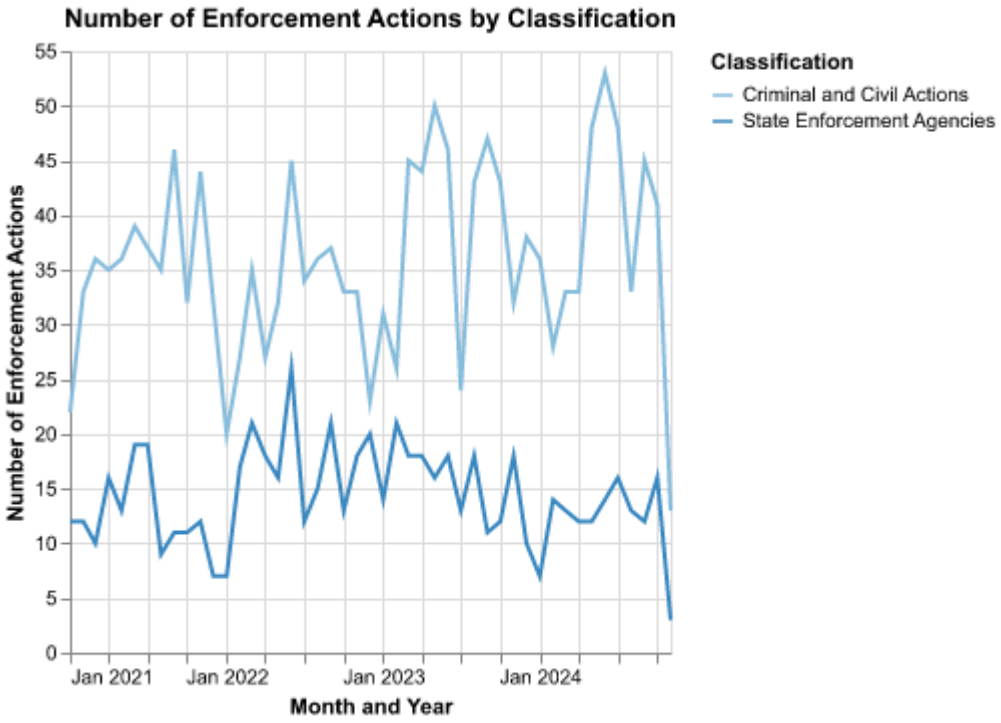


2. Plot the number of enforcement actions categorized: (PARTNER 1)

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

```
#Making a month and year variable
enforcement_actions_012021["date"] =
 ↪  pd.to_datetime(enforcement_actions_012021["year"].astype(str\
  ) + "-" + enforcement_actions_012021["month"].astype(str) + "-01")
```

```
#Creating plot while filetering for criminal and civil actions and state
 ↪  enforcement agencies
plot1 = alt.Chart(enforcement_actions_012021, title="Number of Enforcement
 ↪  Actions by
 ↪  Classification").mark_line().transform_filter((alt.datum.Classification
 ↪  == 'Criminal and Civil Actions') | (alt.datum.Classification == 'State
 ↪  Enforcement Agencies')).encode(
    x=alt.X('yearmonth(date):T', axis=alt.Axis(title= 'Month and Year')),
    y=alt.Y('count():Q', axis=alt.Axis(title= 'Number of Enforcement
 ↪  Actions')),
    color=alt.Color('Classification:O')
)

plot1
```

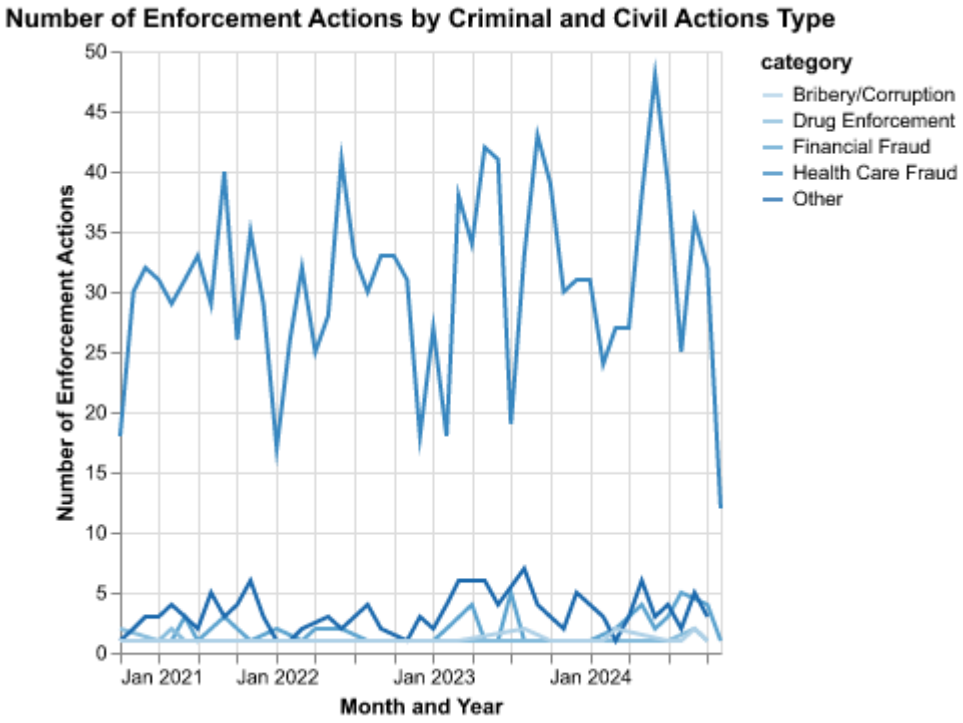


- based on five topics

```
#Creating the conditions to filter the five categories on
conditions = [

 ↪  enforcement_actions_012021['Title'].str.contains('health|doctor|pharm|prescription|\

 ↪  hospital|surgeon|nurs|physician|lab|medi|provider|patient|dent|dermatology|\

 ↪  podiatrist|COVID-19|care|urolog|pediatric|cardiologist|chiropractor|organ|\
        hospice|radiology|eye|heart|respiratory|ambula|clinic|psych',
 ↪  na=False,\
           case=False),
    enforcement_actions_012021['Title'].str.contains('finance|bank|social
 ↪  security|\
      business|embezzl|monetary|defraud|tax|rebate|financial|wire
 ↪  fraud|broker|\
         launder', na=False, case=False),

 ↪  enforcement_actions_012021['Title'].str.contains('drug|oxy|opioid|cocaine|heroin|\
      substance', na=False, case=False),
```

```
    enforcement_actions_012021['Title'].str.contains('bribery|kickback',
 ↪  na=False, \
      case=False)
]

#Developing the categories and applying the conditions to the dataset
choices = ["Health Care Fraud", "Financial Fraud", "Drug Enforcement",
 ↪  "Bribery/Corruption"]
enforcement_actions_012021['category'] = np.select(conditions, choices,
 ↪  default='Other')
```

```
#Creating the plot by five categories
plot_categories = alt.Chart(enforcement_actions_012021, title="Number of
 ↪  Enforcement\
   Actions by Criminal and Civil Actions
 ↪  Type").mark_line().transform_filter(\
    alt.datum.Classification == 'Criminal and Civil Actions').encode(
    x=alt.X('yearmonth(date):T', axis=alt.Axis(title= 'Month and Year')),
    y=alt.Y('count():Q', axis=alt.Axis(title= 'Number of Enforcement
 ↪  Actions')),
    color=alt.Color('category:O')
)

plot_categories
```



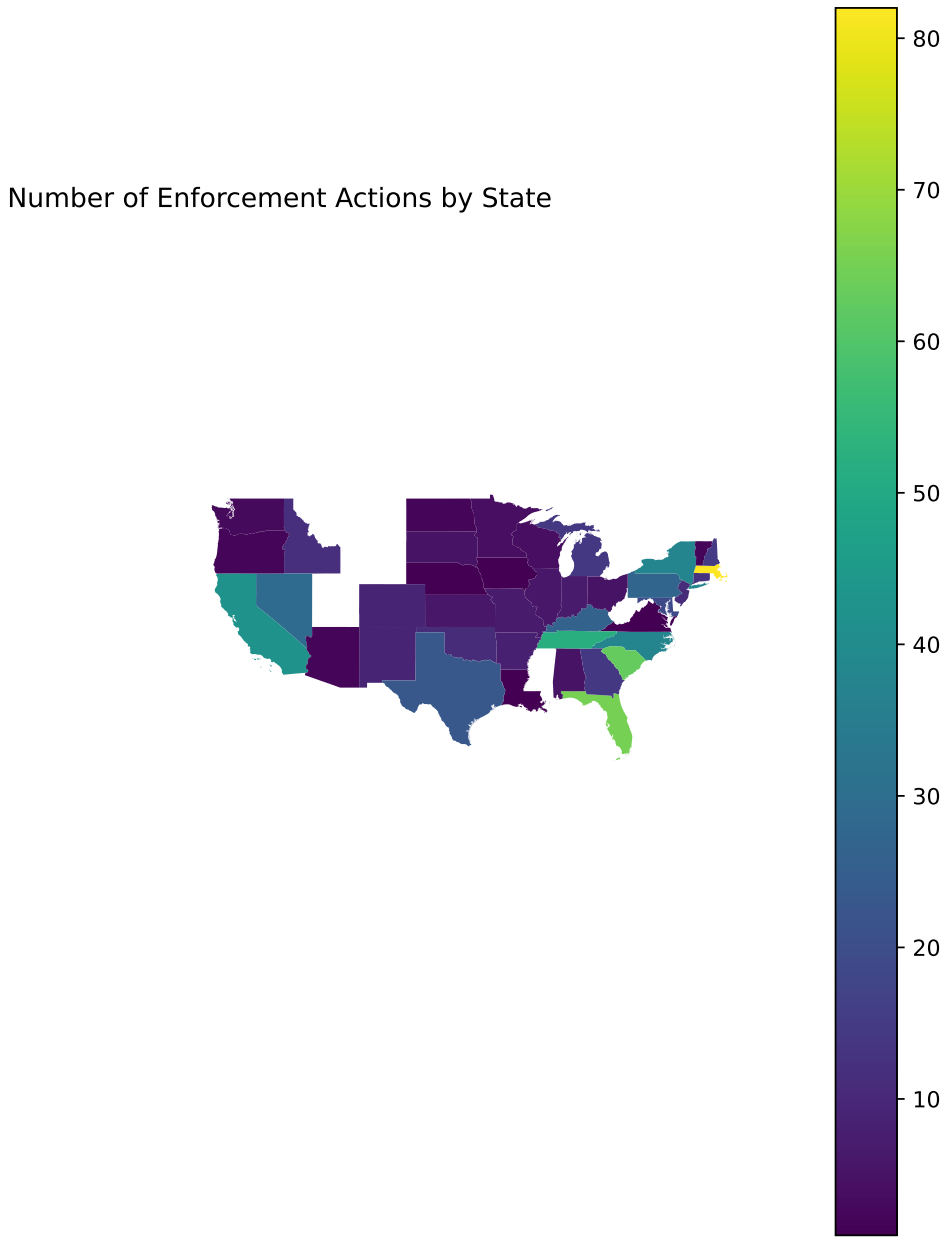
Step 4: Create maps of enforcement activity

1. Map by State (PARTNER 1)

```
shapefile = "Data/cb_2018_us_state_500k.shp"
census_state = gpd.read_file(shapefile)
states =  [
    "Alabama", "Alaska", "Arizona", "Arkansas", "California",
    "Colorado", "Connecticut", "Delaware", "Florida", "Georgia",
    "Hawaii", "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"
]
enforcement_actions_012021['Agency'] =
 ↪  enforcement_actions_012021['Agency'].apply(str)
```

```
#Asked chatGPT for help indexing so that the state would be in the correct
 ↪  row.
enforcement_actions_012021["State"]= None
enforcement_actions_012021_state =
 ↪  enforcement_actions_012021[~enforcement_actions_012021['Agency'].str.contains('district'
 ↪  case=False, na=False)]
for index, row in enforcement_actions_012021_state.iterrows():
    agency = row['Agency']
    for state in states:
        if state.lower() in agency.lower():
            enforcement_actions_012021_state.at[index, 'State'] = state
            break

enforcement_actions_state =
 ↪  enforcement_actions_012021_state.groupby('State')['URL'].count(\
  ).reset_index()

enforcement_actions_state_map = census_state.set_index('NAME').join(\
  enforcement_actions_state.set_index('State'))
```

```
fig,ax =plt.subplots(figsize=(10,10))
enforcement_actions_state_map.plot(column='URL', legend=True, ax=ax)
ax.set_title("Number of Enforcement Actions by State")
ax.set_axis_off()
ax.set_xlim(-170, -65)
ax.set_ylim(0, 75)
plt.show()
```



2. Map by District (PARTNER 2)

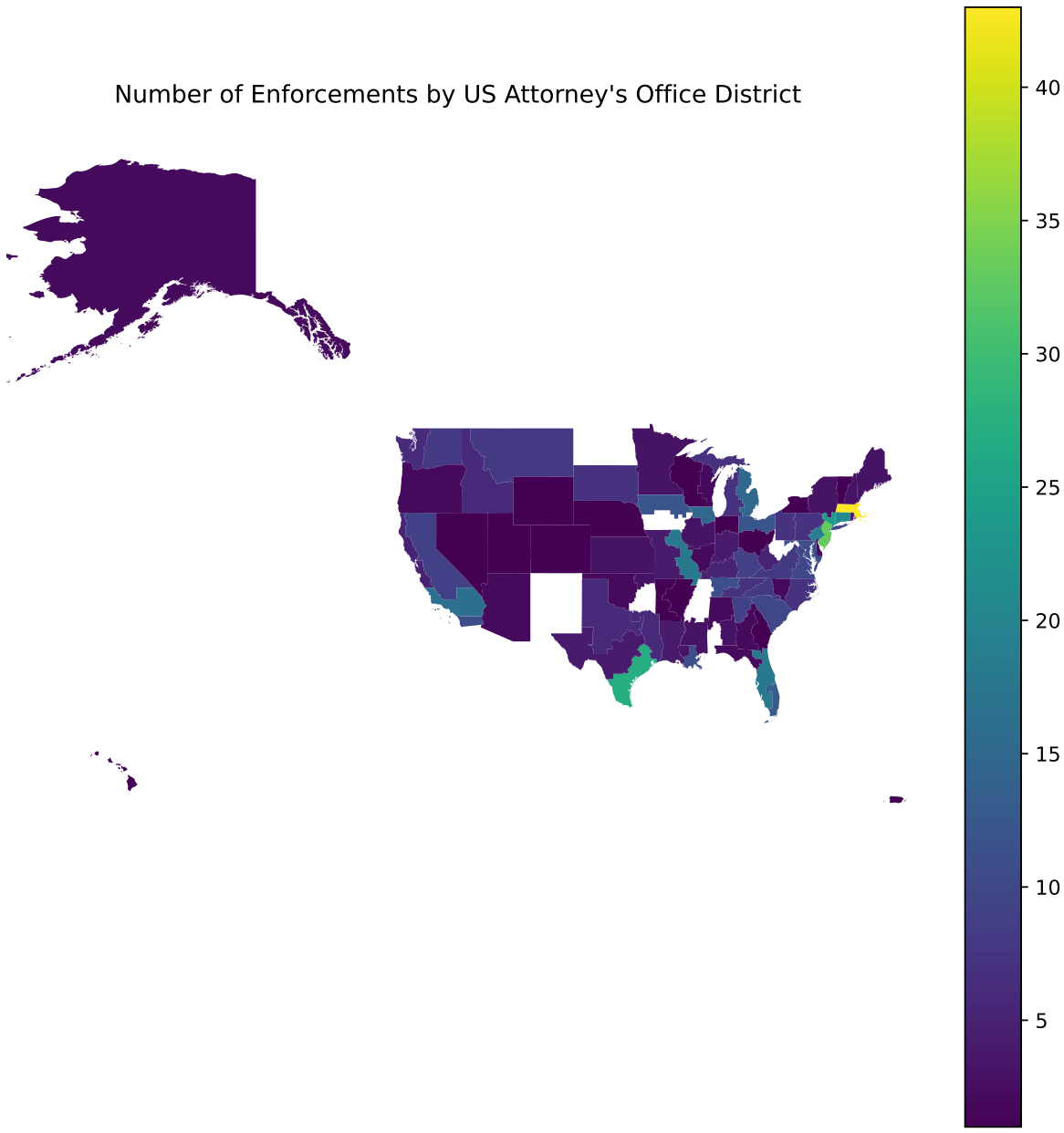
```
#import district shapefile
district_shapefile =
 ↪  "Data/geo_export_31250f05-1141-499d-9d2a-48d87472e3a4.shp"
```

```
districts_sp = gpd.read_file(district_shapefile)
```

```
#extract us attorney's office enforcements
district_enforcements = enforcement_actions_012021.loc[
  enforcement_actions_012021["Agency"].str.startswith("U.S. Attorney's
 ↪  Office")]

#create district name variable
district_enforcements["judicial_d"] =
 ↪  district_enforcements["Agency"].str.slice(start=24)

#calculate number of enforcements in each district
district_enforcements = district_enforcements.groupby(["judicial_d"]).size(
  ).reset_index(name="n_enforcements")

#merge with district shapefile
district_enforcements = pd.merge(districts_sp, district_enforcements,
 ↪  on="judicial_d", how="inner" )
```

```
C:\Users\aliso\AppData\Local\Temp\ipykernel_4164\1516422560.py:6:
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/indexing.html#returning-a-view-versus
  district_enforcements["judicial_d"] =
  district_enforcements["Agency"].str.slice(start=24)
```

```
#map number of enforcements per district
fig, ax = plt.subplots(figsize=(10, 10))
district_enforcements.plot(column="n_enforcements",
                           legend=True,
                           ax=ax,
                           missing_kwds=dict(color="lightgray"))
plt.title("Number of Enforcements by US Attorney's Office District")
plt.axis("off")
ax.set_xlim(-170, -65)
ax.set_ylim(0, 75)
plt.show()
```


Number of Enforcements by US Attorney's Office District
40
35
30
25
20
15
10
5

Extra Credit

1. Merge zip code shapefile with population

```
#import zipcode shapefile
zipcode_shapefile = "Data/gz_2010_us_860_00_500k.shp"
zipcodes_sp = gpd.read_file(zipcode_shapefile)

#import zipcode population file
zip_pop = pd.read_csv("Data/DECENNIALDHC2020.P1-Data.csv")
```

```
#create zipcode variable
zip_pop = zip_pop.drop(index=0).reset_index(drop=True)
zip_pop["ZCTA5"] = zip_pop["GEO_ID"].str[-5:].astype(int).astype(str)

#change zipcode to integer
zipcodes_sp["ZCTA5"] = zipcodes_sp["ZCTA5"].astype(int).astype(str)

#merge population counts to zipcode shapefile
zipcodes_sp = pd.merge(zipcodes_sp, zip_pop, on=["ZCTA5"], how="left")
```

2. Conduct spatial join

```
#spatial join districts to zipcodes
zipcodes_to_districts = gpd.sjoin(zipcodes_sp,
  districts_sp, how="inner", predicate="intersects")

#recast population as integer
zipcodes_to_districts["P1_001N"] =
 ↪  zipcodes_to_districts["P1_001N"].fillna(0).astype(int)

#aggregate zipcode population to districts
district_pops =
 ↪  zipcodes_to_districts.groupby(["abbr"])["P1_001N"].sum().reset_index()
```

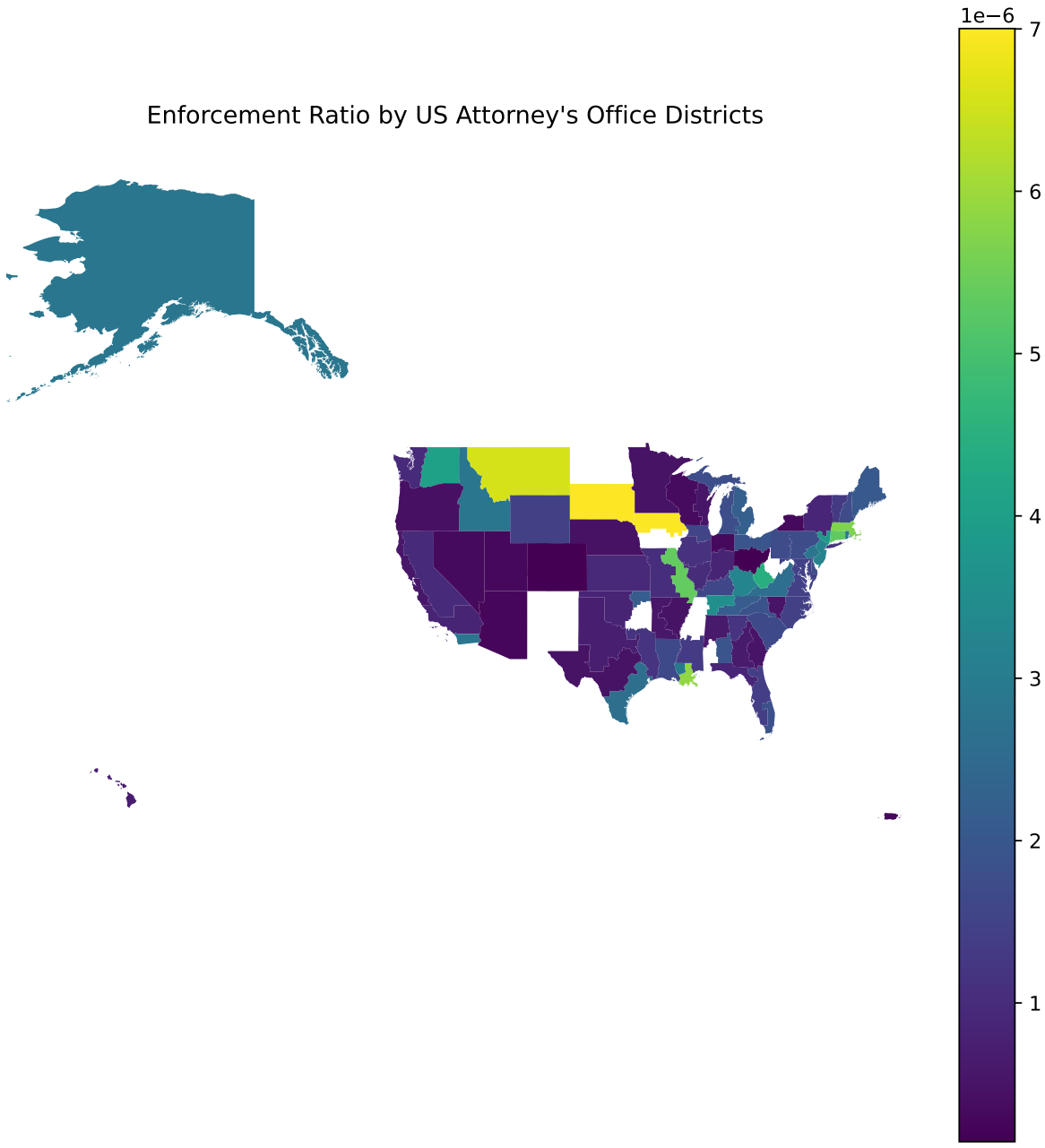
```
C:\Users\aliso\AppData\Local\Temp\ipykernel_4164\37112745.py:2: UserWarning:
CRS mismatch between the CRS of left geometries and the CRS of right
geometries.
```

```
Use `to_crs()` to reproject one of the input geometries to match the CRS of
the other.

Left CRS: EPSG:4269
Right CRS: EPSG:4326

  zipcodes_to_districts = gpd.sjoin(zipcodes_sp,
```

3. Map the action ratio in each district

```
#merge district populations to district shapefile and enforcements
district_enforcements = pd.merge(district_enforcements, district_pops,
  on="abbr", how="inner")

#calculate enforcement ratio
district_enforcements["enforcement_ratio"] =
 ↪  district_enforcements["n_enforcements"
  ] / district_enforcements["P1_001N"]
```

```
#map enforcement ratio
fig, ax = plt.subplots(figsize=(10, 10))
district_enforcements.plot(column="enforcement_ratio",
                           legend=True,
                           ax=ax,
                           missing_kwds=dict(color="lightgray"))
plt.title("Enforcement Ratio by US Attorney's Office Districts")
plt.axis("off")
ax.set_xlim(-170, -65)
ax.set_ylim(0, 75)
plt.show()
```


Enforcement Ratio by US Attorney's Office Districts
1e−6
7
6
5
4
3
2
1