PS-5

Kishika Mahajan & Nidhi Srivastava

2024-10-11

Submission Steps (10 pts)

2. *Partner 1.*
    - Kishika Mahajan (kishika):
    - Nidhi Srivastava (nsrivastava1):
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: KM, NS
5. "I have uploaded the names of anyone else other than my partner and I worked with on the problem set **here**" (Yes)
6. Late coins used this pset: 2 (Kishika) Late coins left after submission: 1 (Kishika) , 1 (Nidhi)

```
import pandas as pd
import altair as alt
import time

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

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

Step 1: Develop initial scraper and crawler

1. Scraping (PARTNER 1)

```
from bs4 import BeautifulSoup
import requests
url = "https://oig.hhs.gov/fraud/enforcement/"

response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")

tag = soup.find_all("li")
len(tag)
# we can see that this has 136 entries, so , we can try to narrow them down
li_us_card = soup.find_all("li", class_="usa-card card--list
 ↪  pep-card--minimal mobile:grid-col-12")
len(li_us_card)
```

```
20
```

Now, we have the 20 observations.

```
# Extracting the titles
titles = []

for li in li_us_card:
    title = li.find("h2",
 ↪  class_="usa-card__heading").find("a").get_text(strip=True)
    # Append the title to the list
    titles.append(title)
```

```
# Extracting the dates
dates = []

for li in li_us_card:
  date = li.find(class_= "text-base-dark
 ↪  padding-right-105").get_text(strip=True)
  dates.append(date)
```

```
# Extracting the category
categories = []

for li in li_us_card:
    category = li.find("ul", class_="display-inline
 ↪  add-list-reset").find("li").get_text(strip=True)
    # Append the category to the list
    categories.append(category)
```

```
# Extracting the links
a_elements = []

# Loop over each 'li' element and find 'a' elements inside
for li in li_us_card:
    a_tags = li.find_all("a")
    a_elements.extend(a_tags)

a_links = [a['href'] for a in a_elements if 'href' in a.attrs]
```

```
# Making the data into a dataframe
data = {
    "Title of Enforcement Action": titles,
    "Date": dates,
    "Category": categories,
    "Link": a_links
}

enforcement_actions_df = pd.DataFrame(data)
enforcement_actions_df.head()
```

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

2. Crawling (PARTNER 1)

As we can see, the links are rleative paths and not absolute paths, so, before getting agency name, we need to get the complete urls

```
from urllib.parse import urljoin
base_url = "https://oig.hhs.gov/fraud/enforcement/"

# Converting the relative links to absolute links
absolute_links = [urljoin(base_url, link) for link in a_links]
```

Now, we have complete urls. As the next step, we need to enter each of these urls and get the agency name

```
# List to store the text from the second 'li' elements
agencies = []

# Loop through each link
for link in absolute_links:
    response_link = requests.get(link)
    soup = BeautifulSoup(response_link.text, "html.parser")

    # Find all 'ul' elements with the specified class
    li_class = soup.find_all("ul", class_="usa-list usa-list--unstyled
 ↪  margin-y-2")

    for ul in li_class:
        li_elements = ul.find_all("li")

        # Ensuring there are at least two 'li' elements
        if len(li_elements) > 1:
            # Get the text of the second 'li' element and append it to the
             ↪  list
```

```
            second_li_text =
 ↪  li_elements[1].get_text(strip=True).replace("Agency:", "").strip()
            agencies.append(second_li_text)
```

```
# making this into a dictionary to add to our df
agencies_dict = {"Agency": agencies}

# converting it into the df
agencies_df = pd.DataFrame(agencies_dict)

# updating our old df
updated_enforcement_actions_df = pd.concat([enforcement_actions_df,
 ↪  agencies_df], axis=1)
updated_enforcement_actions_df.head()
```

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

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

Attrition: We created a function for the scraper to run but it took around 16.5 minutes to scrape data from Jan 2023 to present. We weren't satisfied with this and decided to look for alternatives to increase the efficiency of our function. This is when we came across the library to write concurrent code. Asyncio allows multiple operations to run concurrently within a single thread and in doing so it increases efficiencies in tasks like web scraping. In this particular code, instead of progressing each task sequentially, it does the extraction for agencies and everything else at once. in particular, asyncio.gather(*tasks) makes sure these tasks run concurrently. So, the time taken is way lesser than it would be in the case of step by step extraction.

We will attach the old function at the end of the document as well.

- a. Pseudo-Code (PARTNER 2) **New pseudo-code**

Define async function fetch_page(session, url): Asynchronously fetch HTML content from url using session Return HTML content

Define async function scrape_action_page(session, link): Fetch HTML content of the action page Parse HTML to extract agency information Return agency name or "N/A" if not found

Define async function scrape_enforcement_actions(year, month): If year $<$ 2013: Print error message and return None

```
Initialize variables (base_url, current_page, lists for data)
Set target_date to first day of given year and month

Create aiohttp session with SSL verification disabled

While True:
    Construct current_url based on current_page
    Fetch and parse HTML of current_url

    Find all enforcement action entries on the page

    If no entries found:
        Break the loop

    Initialize list for scraping tasks

    For each entry:
        Extract date, title, category, and link
        If entry date >= target_date:
            Append data to respective lists
            Add task to scrape individual action page
        Else:
            Process remaining tasks
            Return DataFrame with collected data

    Process all tasks for current page
    Increment current_page
    Wait for 1 second (to be polite to the server)

Return DataFrame with all collected data
```

Define async function main(): Set year and month Call scrape_enforcement_actions with year and month Print resulting DataFrame Save DataFrame to CSV file Return DataFrame

If script is run as main program: Run main() function using asyncio.run()

- b. Create Dynamic Scraper (PARTNER 2)

```
import asyncio
import aiohttp
import nest_asyncio
from datetime import datetime

async def fetch_page(session, url):
    async with session.get(url) as response:
        return await response.text()

async def scrape_action_page(session, link):
    html = await fetch_page(session, link)
    action_soup = BeautifulSoup(html, "html.parser")
    agency_ul = action_soup.find("ul", class_="usa-list usa-list--unstyled
 ↪  margin-y-2")
    if agency_ul and len(agency_ul.find_all("li")) > 1:
        agency =
 ↪  agency_ul.find_all("li")[1].get_text(strip=True).replace("Agency:",
 ↪  "").strip()
    else:
        agency = "N/A"
    return agency

async def scrape_enforcement_actions(year, month):
    if year < 2013:
        print("Please input a year >= 2013. Only enforcement actions after
         ↪  2013 are listed.")
        return None

    base_url = "https://oig.hhs.gov/fraud/enforcement/"
    current_page = 1

    titles, dates, categories, links, agencies = [], [], [], [], []
    target_date = datetime(year, month, 1).date()

    # Create a connector with SSL verification disabled
    connector = aiohttp.TCPConnector(ssl=False)

    async with aiohttp.ClientSession(connector=connector) as session:
        while True:
            current_url = f"{base_url}?page={current_page}" if current_page >
 ↪  1 else base_url
```

```
            html = await fetch_page(session, current_url)
            soup = BeautifulSoup(html, "html.parser")

            li_us_card = soup.find_all("li", class_="usa-card card--list
↪  pep-card--minimal mobile:grid-col-12")

            if not li_us_card:
                print(f"No more entries found on page {current_page}.
                 ↪  Stopping.")
                break

            tasks = []
            for li in li_us_card:
                date_str = li.find(class_="text-base-dark
↪  padding-right-105").get_text(strip=True)
                entry_date = datetime.strptime(date_str, "%B %d, %Y").date()

                if entry_date >= target_date:
                    title = li.find("h2",
↪  class_="usa-card__heading").find("a").get_text(strip=True)
                    category = li.find("ul", class_="display-inline
↪  add-list-reset").find("li").get_text(strip=True)
                    link = urljoin(base_url, li.find("a")["href"])

                    titles.append(title)
                    dates.append(date_str)
                    categories.append(category)
                    links.append(link)

                    tasks.append(scrape_action_page(session, link))
                else:
                    agencies.extend(await asyncio.gather(*tasks))
                    return pd.DataFrame({
                        "Title of Enforcement Action": titles,
                        "Date": dates,
                        "Category": categories,
                        "Link": links,
                        "Agency": agencies
                    })

            agencies.extend(await asyncio.gather(*tasks))
```

```
            current_page += 1
            await asyncio.sleep(1)

    return pd.DataFrame({
        "Title of Enforcement Action": titles,
        "Date": dates,
        "Category": categories,
        "Link": links,
        "Agency": agencies
    })

async def main():
    year, month = 2023, 1
    enforcement_df_Jan_2023 = await scrape_enforcement_actions(year, month)
    print(enforcement_df_Jan_2023)

    # Save the DataFrame to a CSV file
    filename = f"enforcement_actions_{year}_{month:02d}.csv"
    enforcement_df_Jan_2023.to_csv(filename, index=False)
    print(f"Data saved to {filename}")

    return enforcement_df_Jan_2023
```

```
# getting the dataframe
nest_asyncio.apply()
if __name__ == "__main__":
    enforcement_df_Jan_2023 = asyncio.run(main())
```

```
enforcement_df_Jan_2023 =
 ↪  pd.read_csv("/Users/kishikamahajan/Desktop/enforcement_actions_2023_01
 ↪  2.csv")
enforcement_df_Jan_2023.head()
```

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

```
print(f"The number of enforcement actions in this dataframe are
 ↪  {enforcement_df_Jan_2023.shape[0]}.")

# getting the details of the last row to get the first entry
last_row_dict = enforcement_df_Jan_2023.iloc[-1].to_dict()
print(last_row_dict)
```

```
The number of enforcement actions in this dataframe are 1534.
{'Title of Enforcement Action': 'Podiatrist Pays $90,000 To Settle False
Billing Allegations', 'Date': 'January 3, 2023', 'Category': 'Criminal and
Civil Actions', 'Link':
'https://oig.hhs.gov/fraud/enforcement/podiatrist-pays-90000-to-settle-false-billing-allegat
'Agency': 'U.S. Attorney's Office, Southern District of Texas'}
```

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

Testing for Jan, 2021

```
async def fetch_page(session, url):
    async with session.get(url) as response:
        return await response.text()

async def scrape_action_page(session, link):
    html = await fetch_page(session, link)
    action_soup = BeautifulSoup(html, "html.parser")
    agency_ul = action_soup.find("ul", class_="usa-list usa-list--unstyled
 ↪  margin-y-2")
    if agency_ul and len(agency_ul.find_all("li")) > 1:
        agency =
 ↪  agency_ul.find_all("li")[1].get_text(strip=True).replace("Agency:",
 ↪  "").strip()
    else:
        agency = "N/A"
    return agency

async def scrape_enforcement_actions(year, month):
    if year < 2013:
        print("Please input a year >= 2013. Only enforcement actions after
         ↪  2013 are listed.")
        return None

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

```
    current_page = 1

    titles, dates, categories, links, agencies = [], [], [], [], []
    target_date = datetime(year, month, 1).date()

    # Create a connector with SSL verification disabled
    connector = aiohttp.TCPConnector(ssl=False)

    async with aiohttp.ClientSession(connector=connector) as session:
        while True:
            current_url = f"{base_url}?page={current_page}" if current_page >
↪   1 else base_url

            html = await fetch_page(session, current_url)
            soup = BeautifulSoup(html, "html.parser")

            li_us_card = soup.find_all("li", class_="usa-card card--list
↪   pep-card--minimal mobile:grid-col-12")

            if not li_us_card:
                print(f"No more entries found on page {current_page}.
                ↪  Stopping.")
                break

            tasks = []
            for li in li_us_card:
                date_str = li.find(class_="text-base-dark
↪   padding-right-105").get_text(strip=True)
                entry_date = datetime.strptime(date_str, "%B %d, %Y").date()

                if entry_date >= target_date:
                    title = li.find("h2",
↪   class_="usa-card__heading").find("a").get_text(strip=True)
                    category = li.find("ul", class_="display-inline
↪   add-list-reset").find("li").get_text(strip=True)
                    link = urljoin(base_url, li.find("a")["href"])

                    titles.append(title)
                    dates.append(date_str)
                    categories.append(category)
                    links.append(link)
```

```
                    tasks.append(scrape_action_page(session, link))
                else:
                    agencies.extend(await asyncio.gather(*tasks))
                    return pd.DataFrame({
                        "Title of Enforcement Action": titles,
                        "Date": dates,
                        "Category": categories,
                        "Link": links,
                        "Agency": agencies
                    })

            agencies.extend(await asyncio.gather(*tasks))
            current_page += 1
            await asyncio.sleep(1)

    return pd.DataFrame({
        "Title of Enforcement Action": titles,
        "Date": dates,
        "Category": categories,
        "Link": links,
        "Agency": agencies
    })

async def main():
    year, month = 2021, 1
    enforcement_df_Jan_2021 = await scrape_enforcement_actions(year, month)
    print(enforcement_df_Jan_2021)

    # Save the DataFrame to a CSV file
    filename = f"enforcement_actions_{year}_{month:02d}.csv"
    enforcement_df_Jan_2021.to_csv(filename, index=False)
    print(f"Data saved to {filename}")

    return enforcement_df_Jan_2021
```

```
# getting the dataframe
nest_asyncio.apply()
if __name__ == "__main__":
    enforcement_df_Jan_2021 = asyncio.run(main())
```

```
enforcement_df_Jan_2021 =
 ↪  pd.read_csv("/Users/kishikamahajan/Desktop/enforcement_actions_2021_01
 ↪  2.csv")
enforcement_df_Jan_2021.head()
```

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

```
print(f"The number of enforcement actions in this dataframe are
 ↪  {enforcement_df_Jan_2021.shape[0]}.")

# getting the details of the last row to get the first entry
last_row_dict = enforcement_df_Jan_2021.iloc[-1].to_dict()
print(last_row_dict)
```

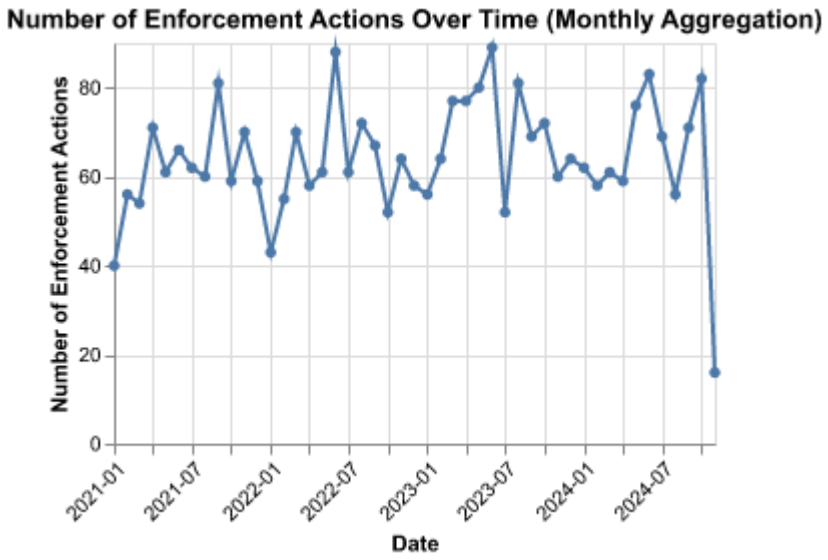
```
The number of enforcement actions in this dataframe are 3022.
{'Title of Enforcement Action': 'The United States And Tennessee Resolve
Claims With Three Providers For False Claims Act Liability Relating To
‘P-Stim’ Devices For A Total Of $1.72 Million', 'Date': 'January 4, 2021',
'Category': 'Criminal and Civil Actions', 'Link':
'https://oig.hhs.gov/fraud/enforcement/the-united-states-and-tennessee-resolve-claims-with-t
'Agency': "U.S. Attorney's Office, Middle District of Tennessee"}
```

Step 3: Plot data based on scraped data

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

```
import altair as alt
#Convert Date to Datetime
enforcement_df_Jan_2021['Date'] =
 ↪  pd.to_datetime(enforcement_df_Jan_2021['Date'], errors='coerce')


# Group by year and month, and count the number of actions
```

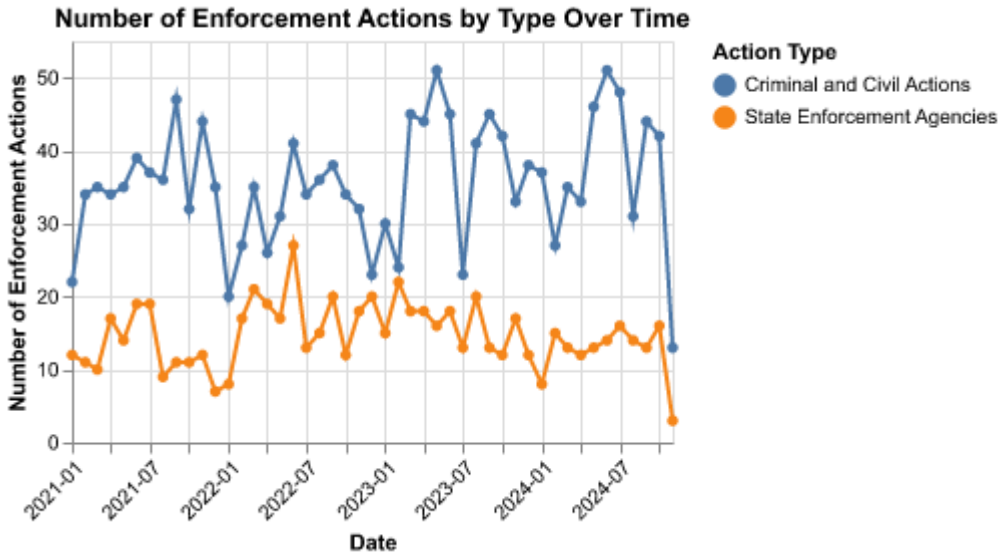
```
monthly_actions = (
    enforcement_df_Jan_2021
    .groupby(enforcement_df_Jan_2021['Date'].dt.to_period("M"))
    .size()
    .reset_index(name='Number of Enforcement Actions')
)

# Convert the 'Date' to a standard datetime format (first day of each month)
monthly_actions['Date'] = monthly_actions['Date'].dt.to_timestamp()

# Create the line chart with Altair
line_chart = alt.Chart(monthly_actions).mark_line(point=True).encode(
    x=alt.X('Date:T', title='Date', axis=alt.Axis(format='%Y-%m',
 ↪  labelAngle=-45)),
    y=alt.Y('Number of Enforcement Actions:Q', title='Number of Enforcement
 ↪  Actions'),
    tooltip=['Date:T', 'Number of Enforcement Actions:Q']
).properties(
    title='Number of Enforcement Actions Over Time (Monthly Aggregation)',
    width=300,
    height=200
)

line_chart.display()
```



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

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

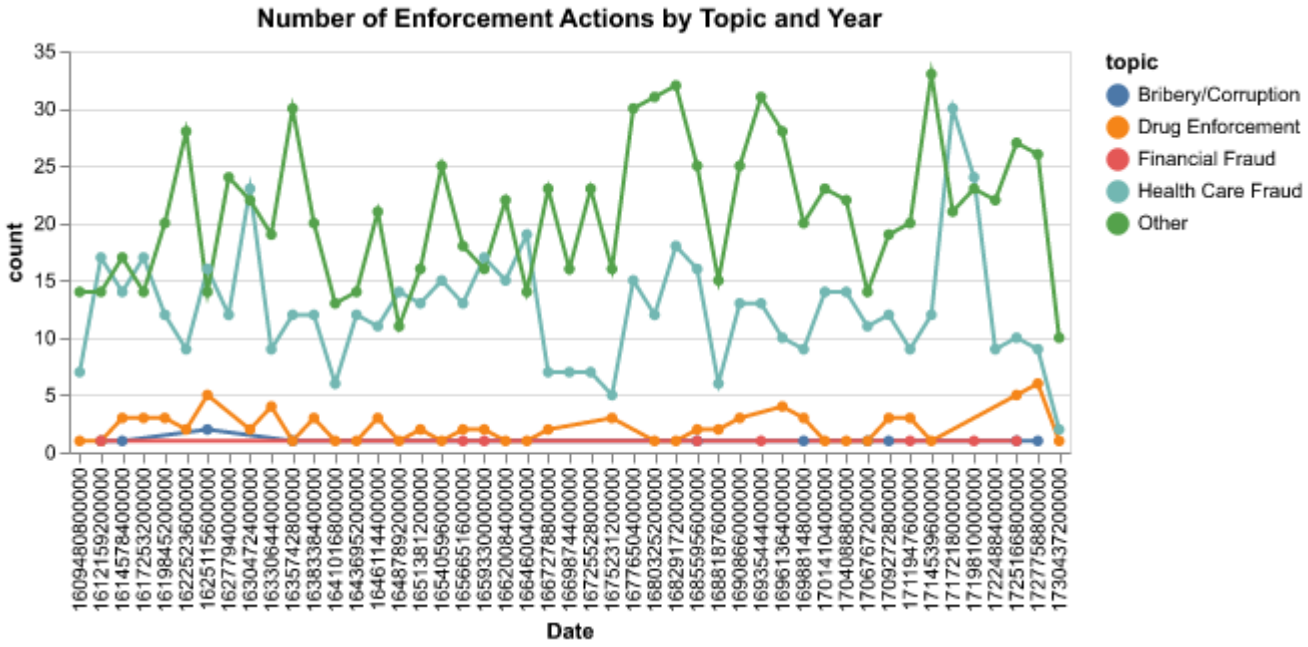
```
#Getting Year Variable
# Convert date to datetime and extract year

enforcement_df_Jan_2021['Year'] = enforcement_df_Jan_2021['Date'].dt.year

#Filtering data based on Category
enforcement_df_Jan_2021_subset = enforcement_df_Jan_2021[
    (enforcement_df_Jan_2021['Category'] == "Criminal and Civil Actions") |
    (enforcement_df_Jan_2021['Category'] == "State Enforcement Agencies")
]

#Grouping data to get count values for number of enforcements per category
enforcement_df_Jan_2021_count =
 ↪  enforcement_df_Jan_2021_subset.groupby([enforcement_df_Jan_2021_subset['Date'].dt.to_peri
 ↪  'Category']).size().reset_index(name='count')
enforcement_df_Jan_2021_count

# Convert the 'Date' to a standard datetime format (first day of each month)
enforcement_df_Jan_2021_count['Date'] =
 ↪  enforcement_df_Jan_2021_count['Date'].dt.to_timestamp()

# Plotting the line chart after subsetting the data
line_chart =
 ↪  alt.Chart(enforcement_df_Jan_2021_count).mark_line(point=True).encode(
    x=alt.X('Date:T', title='Date', axis=alt.Axis(format='%Y-%m',
 ↪  labelAngle=-45)),
    y=alt.Y('count:Q', title='Number of Enforcement Actions'),
    color=alt.Color('Category:N', title='Action Type'),
    tooltip=['Date:T', 'Category:N', 'Number of Enforcement Actions:Q']
).properties(
    title='Number of Enforcement Actions by Type Over Time',
    width=300,
    height=200
)

line_chart.display()
```



- based on five topics

```
# Assign topics based on keywords
def assign_topic(title):
    title = title.lower()
    if "health" in title:
        return "Health Care Fraud"
    elif "financial" in title or "bank" in title:
        return "Financial Fraud"
    elif "drug" in title:
        return "Drug Enforcement"
    elif "bribery" in title or "corruption" in title:
        return "Bribery/Corruption"
    else:
        return "Other"

# Apply the function to assign topics
enforcement_df_Jan_2021['topic'] = enforcement_df_Jan_2021.apply(lambda row:
 ↪  assign_topic(row['Title of Enforcement Action']) if row['Category'] ==
 ↪  "Criminal and Civil Actions" else None, axis=1)

#Subsetting the data
enforcement_df_Jan_2021_subset_topic =
 ↪  enforcement_df_Jan_2021[enforcement_df_Jan_2021['Category'] == 'Criminal
 ↪  and Civil Actions']
enforcement_df_Jan_2021_subset
```

```
#Grouping data by topic
enforcement_df_Jan_2021_count_topic =
 ↪  enforcement_df_Jan_2021_subset_topic.groupby([enforcement_df_Jan_2021_subset['Date'].dt.t
 ↪  'topic']).size().reset_index(name='count')


# Convert the 'Date' to a standard datetime format (first day of each month)
enforcement_df_Jan_2021_count_topic['Date'] =
 ↪  enforcement_df_Jan_2021_count_topic['Date'].dt.to_timestamp()

# Create the Altair line chart
chart =
 ↪  alt.Chart(enforcement_df_Jan_2021_count_topic).mark_line(point=True).encode(
    x='Date:O',
    y='count:Q',
    color='topic:N',
    tooltip=['Date', 'topic', 'count']
).properties(
    title="Number of Enforcement Actions by Topic and Year",
    width=500,
    height=200
).interactive()

chart
```



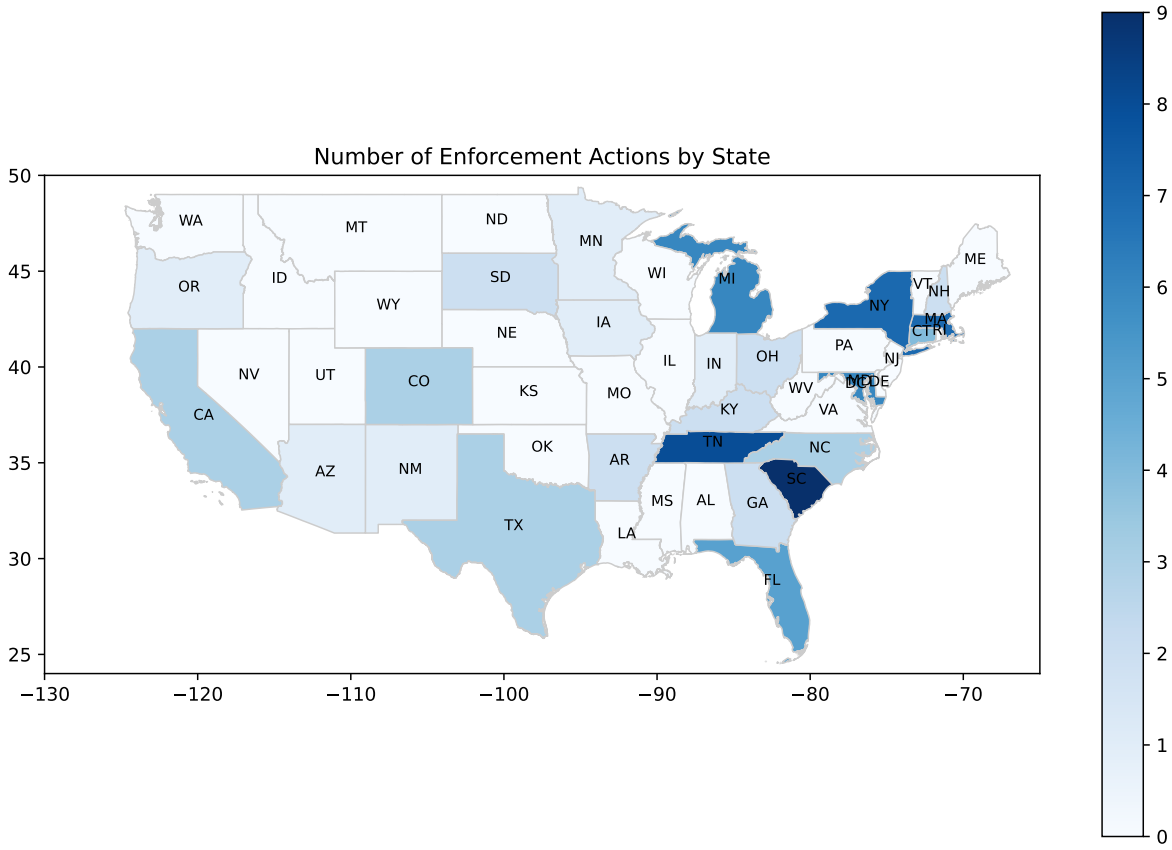
Step 4: Create maps of enforcement activity

1. Map by State (PARTNER 1)

```
#Importing libraries
import geopandas as gpd
import matplotlib.pyplot as plt
import json

#Filtering State agency Enforcment Actions only
state_enforcement_data =
 ↪  enforcement_df_Jan_2021[enforcement_df_Jan_2021['Agency'].str.contains("State
 ↪  of", na=False)]

#Clean the State names
state_enforcement_data['State'] =
 ↪  state_enforcement_data['Agency'].str.replace("State of ", "").str.strip()

# Group by state and count
state_counts =
 ↪  state_enforcement_data.groupby('State').size().reset_index(name='count')
```

```
#Reading the census state shapefile
states_shp =
 ↪  gpd.read_file('/Users/kishikamahajan/Desktop/cb_2018_us_state_500k/cb_2018_us_state_500k

#Merge state enforcement data with state census file
states_shp = states_shp.merge(state_counts, left_on='NAME', right_on='State',
 ↪  how='left').fillna(0)

statecode_drop = ['AS', 'HI', 'PR', 'VI', 'MP', 'GU', 'AK']

#Manually dropping certain state codes to get zoomed in choropleth
states_shp = states_shp[~states_shp['STUSPS'].isin(statecode_drop)]
```

```
#Plot
# Plot the choropleth with a zoomed-in view
fig, ax = plt.subplots(1, 1, figsize=(12, 8))
states_shp.plot(column='count', cmap='Blues', linewidth=0.8, ax=ax,
 ↪  edgecolor='0.8', legend=True)

# Set the extent to continental U.S. (approximate lat/long bounds)
ax.set_xlim([-130, -65])  # Longitude bounds for the continental U.S.
ax.set_ylim([24, 50])

# Add state names or codes
for idx, row in states_shp.iterrows():
    # Use the centroid for placement of the label
    plt.text(row.geometry.centroid.x, row.geometry.centroid.y,
             row['STUSPS'],  # Replace with 'STATE_NAME' for full names
             fontsize=8, ha='center', color='black')

# Add title and display
ax.set_title('Number of Enforcement Actions by State')
plt.show()
```



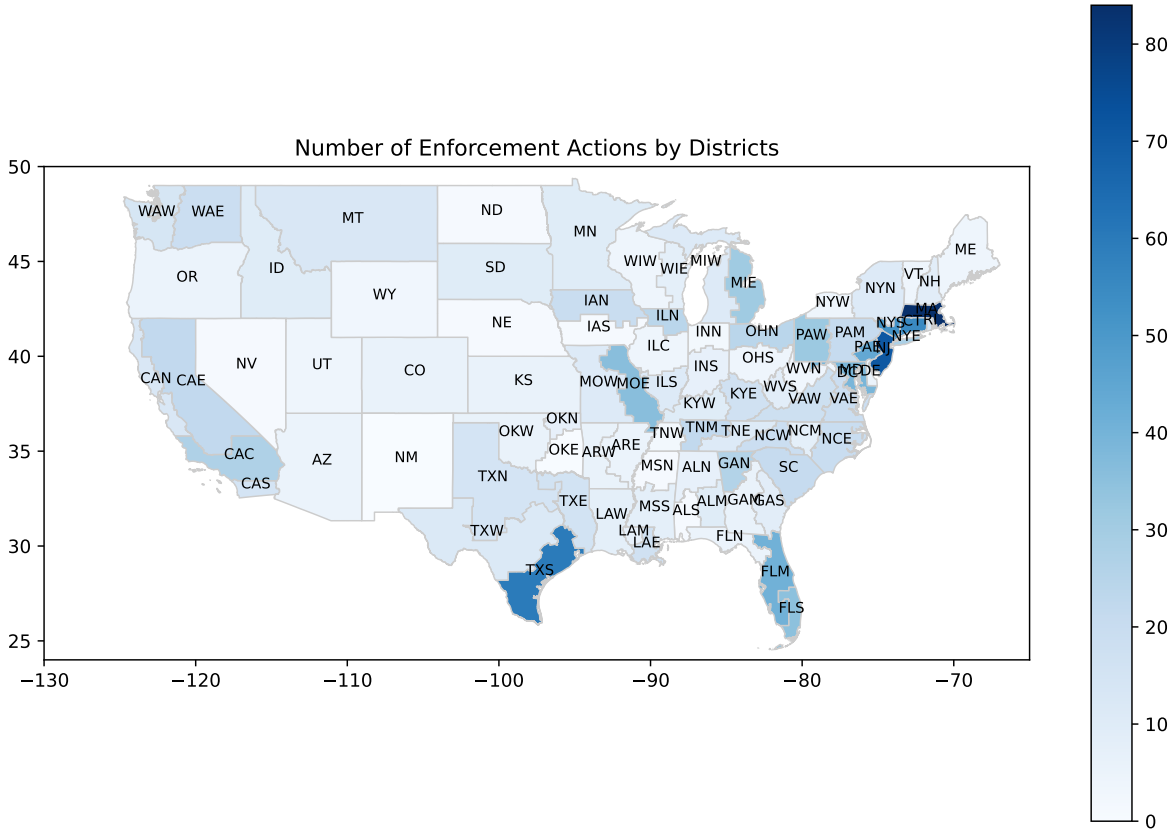
2. Map by District (PARTNER 2)

```
#Filtering for district level data
district_enforcement_data =
 ↪  enforcement_df_Jan_2021[enforcement_df_Jan_2021['Agency'].str.contains("District",
 ↪  na=False)]

#Getting district names
district_enforcement_data['District'] =
 ↪  district_enforcement_data['Agency'].str.split(',').str[1].str.strip()

# Aggregate enforcement data by district
district_counts =
 ↪  district_enforcement_data.groupby('District').size().reset_index(name='count')
```

```
#Reading District shapefile
district_shapefile = gpd.read_file('/Users/kishikamahajan/Desktop/US Attorney
 ↪  Districts Shapefile
 ↪  simplified_20241108/geo_export_6122d5c7-042c-4e6e-9edd-c15058235201.shp')

#Merging with district on Judicial District name
district_shapefile.columns = district_shapefile.columns.str.strip()
district_merge = pd.merge(district_shapefile, district_counts,
 ↪  left_on='judicial_d', right_on='District', how='left').fillna(0)
```

```
code_drop = ['AS', 'HI', 'PR', 'VI', 'MP', 'GU', 'AK']

#Manually dropping certain state codes to get zoomed in choropleth
district_merge = district_merge[~district_merge['abbr'].isin(code_drop)]
```

```
#Plot
fig, ax = plt.subplots(1, 1, figsize=(12, 8))
district_merge.plot(column='count', cmap='Blues', linewidth=0.8, ax=ax,
 ↪  edgecolor='0.8', legend=True)

# Set the extent to continental U.S. (approximate lat/long bounds)
ax.set_xlim([-130, -65])  # Longitude bounds for the continental U.S.
ax.set_ylim([24, 50])

# Add district names or codes
for idx, row in district_merge.iterrows():
    # Use the centroid for placement of the label
    plt.text(row.geometry.centroid.x, row.geometry.centroid.y,
             row['abbr'],  # Replace with 'DISTRICT_NAME' for full names
             fontsize=8, ha='center', color='black')

# Add title and display
ax.set_title('Number of Enforcement Actions by Districts')
plt.show()
```



Extra Credit

1. Merge zip code shapefile with population

```
# Loading both the files
zip_code_shapefile =
 ↪  gpd.read_file("/Users/kishikamahajan/Desktop/gz_2010_us_860_00_500k/gz_2010_us_860_00_500

population =
 ↪  pd.read_csv("/Users/kishikamahajan/Desktop/DECENNIALDHC2020.P1-Data.csv")
```

```
# Merging the two datasets together
population = population.drop(index=0).reset_index(drop=True)

# making the column ready for the merge
```

```
population['NAME'] = population['NAME'].str.replace('ZCTA5 ', '',
 ↪  regex=False)

# change the column name
population = population.rename(columns={'NAME': 'ZCTA5'})

# merging the files
merged_file = pd.merge(zip_code_shapefile , population, on = "ZCTA5" , how =
 ↪  "left")

merged_file = merged_file.rename(columns={'P1_001N': 'POPULATION'})
```

2. Conduct spatial join

```
joined_gdf = gpd.sjoin(merged_file, district_shapefile, how = "inner",
 ↪  predicate = "within")
```

```
# Aggregating population by district
population_by_district =
 ↪  joined_gdf.groupby("judicial_d")["POPULATION"].sum().reset_index()

district_population_gdf = district_shapefile.merge(population_by_district,
 ↪  on="judicial_d")

district_population_gdf =
 ↪  district_population_gdf.rename(columns={'judicial_d': 'District'})
```

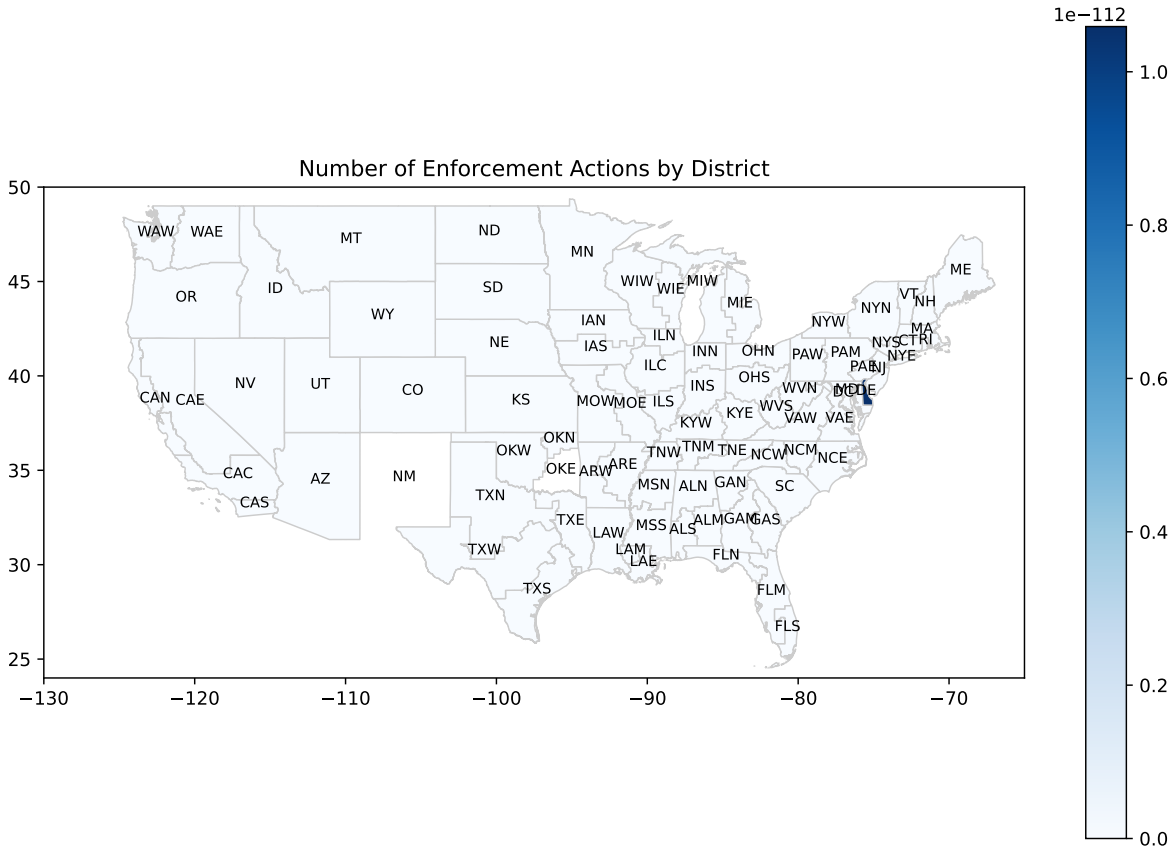
3. Map the action ratio in each district

```
# getting the enforcement actions per district from Jan 2021

district_enforcement_data_grouped =
 ↪  district_enforcement_data.groupby("District").size().reset_index(name="Count")

# merging the two files
ratio_df = pd.merge(district_population_gdf ,
 ↪  district_enforcement_data_grouped, on = "District" , how = "left")
```

```
ratio_df["Count"] = ratio_df["Count"].astype(float)
ratio_df["POPULATION"] = ratio_df["POPULATION"].astype(float)

ratio_df["Ratio"] = ratio_df["Count"]/ratio_df["POPULATION"]
```

```
#Dropping the codes
code_drop = ['AS', 'HI', 'PR', 'VI', 'MP', 'GU', 'AK']

ratio_df = ratio_df[~ratio_df['abbr'].isin(code_drop)]
```

```
# Plot the choropleth with a zoomed-in view
fig, ax = plt.subplots(1, 1, figsize=(12, 8))
ratio_df.plot(column='Ratio', cmap='Blues', linewidth=0.8, ax=ax,
 ↪  edgecolor='0.8', legend=True)

# Set the extent to continental U.S. (approximate lat/long bounds)
ax.set_xlim([-130, -65])  # Longitude bounds for the continental U.S.
ax.set_ylim([24, 50])

# Add state names or codes
for idx, row in ratio_df.iterrows():
    # Use the centroid for placement of the label
    plt.text(row.geometry.centroid.x, row.geometry.centroid.y,
             row['abbr'],  # Replace with 'STATE_NAME' for full names
             fontsize=8, ha='center', color='black')

# Add title and display
ax.set_title('Number of Enforcement Actions by District')
plt.show()
```



Old function for scraping

```
#from datetime import datetime

#def scrape_enforcement_actions(year, month):
    #if year < 2013:
        #print("Please input a year >= 2013. Only enforcement actions after
        ↪  2013 are listed.")
        #return None

    #base_url = "https://oig.hhs.gov/fraud/enforcement/"
    #current_page = 1  # Start from page 1

    #titles, dates, categories, links, agencies = [], [], [], [], []
    #target_date = datetime(year, month, 1).date()

    #while True:
        # Construct the URL dynamically based on the current page
```

```
        #if current_page == 1:
            #current_url = base_url  # First page doesn't have ?page=1
        #else:
            #current_url = f"{base_url}?page={current_page}"

        #print(f"Scraping: {current_url}")
        #response = requests.get(current_url)
        #soup = BeautifulSoup(response.text, "html.parser")

        # Find all enforcement action cards on the current page
        #li_us_card = soup.find_all("li", class_="usa-card card--list
         ↪  pep-card--minimal mobile:grid-col-12")

        # If no cards are found on the page, stop scraping (end of pages)
        #if not li_us_card:
            #print(f"No more entries found on page {current_page}.
             ↪  Stopping.")
            #break

        # Scrape data from each card
        #for li in li_us_card:
            #date_str = li.find(class_="text-base-dark
             ↪  padding-right-105").get_text(strip=True)
            #entry_date = datetime.strptime(date_str, "%B %d, %Y").date()

            #if entry_date >= target_date:
                #title = li.find("h2",
                 ↪  class_="usa-card__heading").find("a").get_text(strip=True)
                #category = li.find("ul", class_="display-inline
                 ↪  add-list-reset").find("li").get_text(strip=True)
                #link = urljoin(base_url, li.find("a")["href"])

                #titles.append(title)
                #dates.append(date_str)
                #categories.append(category)
                #links.append(link)

                # Crawl individual action page for agency
                #action_response = requests.get(link)
                #action_soup = BeautifulSoup(action_response.text,
                 ↪  "html.parser")
                #agency_ul = action_soup.find("ul", class_="usa-list
                 ↪  usa-list--unstyled margin-y-2")
```

```
                #if agency_ul and len(agency_ul.find_all("li")) > 1:
                    #agency =
                     ↪  agency_ul.find_all("li")[1].get_text(strip=True).replace("Agency:",
                     ↪  "").strip()
                #else:
                    #agency = "N/A"
                #agencies.append(agency)
            #else:
                # If we've reached an entry older than the target date, stop
                 ↪  scraping
                #return pd.DataFrame({
                    #"Title of Enforcement Action": titles,
                    #"Date": dates,
                    #"Category": categories,
                    #"Link": links,
                    #"Agency": agencies
                #})

        # Increment the page counter to move to the next page
        #current_page += 1

        # Add a delay between requests to avoid overloading the server
        #time.sleep(0.2)

    # If we've gone through all pages without finding an older entry
    #return pd.DataFrame({
        #"Title of Enforcement Action": titles,
        #"Date": dates,
        #"Category": categories,
        #"Link": links,
        #"Agency": agencies
    #})
```

```
#result_df = scrape_enforcement_actions(2023, 1)
```