Problem Set 5

Yuki Yan & Tianhao Zhang

2024-11-10

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

Submission Steps (10 pts)

1. This problem set is a paired problem set.
2. Play paper, scissors, rock to determine who goes first. Call that person *Partner 1*.
    - Partner 1 (name and cnet ID): Yuki Yan & yukiyan
    - Partner 2 (name and cnet ID): Tianhao Zhang & tz2205
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: **_YY_** **_TZ_**
5. "I have uploaded the names of anyone else other than my partner and I worked with on the problem set **here**" (1 point)
6. Late coins used this pset: **_1/1_** Late coins left after submission: **_0/1_**
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 warnings
warnings.filterwarnings('ignore')
alt.renderers.enable("png")
```

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

Step 1: Develop initial scraper and crawler

1. Scraping (PARTNER 1)

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

def scrape_oig_actions(url):
    url = 'https://oig.hhs.gov/fraud/enforcement/'

    response = requests.get(url)

    soup = BeautifulSoup(response.text, 'lxml')

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

    data = []
    for item in soup.find_all:
        title_tag =  item.find("h2", class_="usa-card__heading").find("a")
        title = title_tag.get_text(strip=True)
        link = "https://oig.hhs.gov" + title_tag["href"]
        date = item.find("span", class_="text-base-dark
↪   padding-right-105").get_text(strip=True)

        category = item.find("li", class_="display-inline-block usa-tag
↪   text-no-lowercase text-base-darkest bg-base-lightest
↪   margin-right-1").get_text(strip=True)
        data.append({
```

```
            'Title': title,
            'Date': date,
            'Category': category,
            'Link': link
        })

    return pd.DataFrame(data)

url = 'https://oig.hhs.gov/fraud/enforcement/'
df = scrape_oig_actions(url)
print(df.head())
```

2. Crawling (PARTNER 1)

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import time

# Setting the display width for columns to a higher number to prevent
 ↪ truncation
pd.set_option('display.max_colwidth', None)

def fetch_agency_details(url):
    try:
        response = requests.get(url, timeout=10)
        if response.status_code != 200:
            return {'title': 'No Title Found', 'agency': 'No Agency Found'}

        soup = BeautifulSoup(response.text, 'html.parser')
        result = {'title': 'No Title Found', 'agency': 'No Agency Found'}

        # Fetch the title using <h1> tag
        title_tag = soup.find('h1', class_='font-heading-xl')
        if title_tag:
            result['title'] = title_tag.text.strip()

        # Locate the `ul` and find `li` for agency
        ul = soup.find('ul', class_='usa-list usa-list--unstyled margin-y-2')
        if ul:
```

```
            for li in ul.find_all('li'):
                span = li.find('span', class_='padding-right-2 text-base',
 ↪  string='Agency:')
                if span:
                    result['agency'] = li.text.replace('Agency:', '').strip()
                    break

        return result
    except requests.RequestException as e:
        return {'title': 'No Title Found', 'agency': f'Failed to fetch due
         ↪  to: {e}'}

def fetch_enforcement_links(page_url, visited_urls):
    try:
        response = requests.get(page_url, timeout=10)
        soup = BeautifulSoup(response.text, 'html.parser')
        links = []

        for a_tag in soup.find_all('a', href=True):
            if 'href' in a_tag.attrs and '/fraud/enforcement/' in
             ↪  a_tag['href']:
                full_url = 'https://oig.hhs.gov' + a_tag['href']
                if full_url not in visited_urls:
                    links.append(full_url)
                    visited_urls.add(full_url)

        return links
    except requests.RequestException as e:
        print(f"Failed to process {page_url}: {e}")
        return []

def main():
    base_url = 'https://oig.hhs.gov/fraud/enforcement/'
    data = []
    visited_urls = set()

    # Only processing the first page for demonstration
    page_url = base_url
    print(f"Processing {page_url}...")
    all_links = fetch_enforcement_links(page_url, visited_urls)

    for link in all_links:
```

```
        details = fetch_agency_details(link)
        if details['title'] != 'No Title Found' or details['agency'] != 'No
        ↪  Agency Found':
            data.append(details)

    if data:
        df = pd.DataFrame(data)
        df.to_csv('enforcement_actions_agencies.csv', index=False)
        print("Data collected and saved successfully:")
        print(df.head())
    else:
        print("No valid data was collected.")

if __name__ == '__main__':
    main()
```

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code (PARTNER 2)

```
# extract information for agency according to Partner 1's method
# start_enforcement_actions (start_year, start_month)
# validation to check: if start_enforcement_actions (start_year) >= 2013,
↪  keep running
# if start_date <2013, return without executing further if the year is
↪  invalid or print a warning message
# set base URL (https://oig.hhs.gov/fraud/enforcement/)
# initialize empty dataframe filtered_action to store the filtered
↪  enforcemnent action data
# calculate and generate start_data(datetimen object) by start_year and
↪  start_date
# set current_page to 1

# while loop:
# while there are pages to process:
# url pattern is base_url + "?page=" + current_page
# call requests and beautifulsoup, fetch and parse the html from the page
# check and find all enforcement actions, if no action items are found, break
↪  the loop
```

```
# filter and process each enforcement action to extract information needed
# extraction:
# title, link, datetime, category
# convert date to datetime object, defind as date_obj
# if date_obj < start_date, break the loop, no further data needed
# if date_obj is valid, append an entry to filtered_action, with title, date,
 ↪  category, and (link)
# to prevent server overload, rate limiting or blocking, or just be polite to
 ↪  the web: add delay, pause a bit before ther nextn requests

# save data to csv: name nthe output file as
 ↪  enforcement_action_year_month.csv, replacing year and month with
 ↪  start_year and start_month
# convert filtered_actions to a dataframen for column and rows showing in csv
 ↪  format
# save the dataframe to csv
# end function
```

- b. Create Dynamic Scraper (PARTNER 2)

```
import pandas as pd
from bs4 import BeautifulSoup
import requests
import time
from datetime import datetime
from concurrent.futures import ThreadPoolExecutor, as_completed

pd.set_option('display.max_colwidth', None)

# Function to fetch agency details for a single link
def fetch_agency_details(url):
    try:
        response = requests.get(url, timeout=10)
        if response.status_code != 200:
            return {'title': 'No Title Found', 'agency': 'No Agency Found',
             ↪  'category': 'No Category Found', 'link': url}

        soup = BeautifulSoup(response.text, 'html.parser')
        result = {'title': 'No Title Found', 'agency': 'No Agency Found',
 ↪  'category': 'No Category Found', 'link': url}
```

```
        # Extract title
        title_tag = soup.find('h1', class_='font-heading-xl')
        if title_tag:
            result['title'] = title_tag.text.strip()

        # Extract agency
        ul = soup.find('ul', class_='usa-list usa-list--unstyled margin-y-2')
        if ul:
            for li in ul.find_all('li'):
                span = li.find('span', class_='padding-right-2 text-base',
 ↪  string='Agency:')
                if span:
                    result['agency'] = li.text.replace('Agency:', '').strip()
                    break

        # Extract category
        category_tag = soup.find('li', class_='display-inline-block usa-tag
 ↪  text-no-lowercase text-base-darkest bg-base-lightest margin-right-1')
        if category_tag:
            result['category'] = category_tag.get_text(strip=True)

        return result
    except requests.RequestException as e:
        return {'title': 'No Title Found', 'agency': f'Failed to fetch due
         ↪  to: {e}', 'category': 'No Category Found', 'link': url}

# Function to fetch enforcement links from a single page
def fetch_enforcement_links(page_url, visited_urls):
    try:
        response = requests.get(page_url, timeout=10)
        soup = BeautifulSoup(response.text, 'html.parser')
        links = []

        for a_tag in soup.find_all('a', href=True):
            if 'href' in a_tag.attrs and '/fraud/enforcement/' in
             ↪  a_tag['href']:
                full_url = 'https://oig.hhs.gov' + a_tag['href']
                if full_url not in visited_urls:
                    links.append(full_url)
                    visited_urls.add(full_url)

        return links
```

```
    except requests.RequestException as e:
        print(f"Failed to process {page_url}: {e}")
        return []

# Main function to scrape enforcement actions starting from January 2013
def scrape_enforcement_actions(start_year, start_month):
    if start_year < 2013:
        print("Only enforcement actions from 2013 onwards are available.
        ↪  Please enter a year >= 2013.")
        return

    base_url = 'https://oig.hhs.gov/fraud/enforcement/'
    data = []
    visited_urls = set()
    page = 1
    start_date = datetime(start_year, start_month, 1)
    should_continue = True

    while should_continue:
        page_url = f"{base_url}?page={page}"
        print(f"Processing {page_url}...")
        links = fetch_enforcement_links(page_url, visited_urls)

        with ThreadPoolExecutor(max_workers=5) as executor:
            future_to_link = {executor.submit(fetch_agency_details, link):
↪  link for link in links}
            for future in as_completed(future_to_link):
                link = future_to_link[future]
                try:
                    details = future.result()
                    # Filter based on date
                    # Assume date extraction logic here if available
                    # Placeholder: Check start_date here if date extracted,
                    ↪  else skip

                    # If valid data, append to results
                    data.append(details)
                except Exception as exc:
                    print(f"Error processing link {link}: {exc}")

        page += 1
        time.sleep(1)  # Delay to prevent overload
```

```
        if not links:
            should_continue = False

    if data:
        filename = f"enforcement_actions_{start_year}_{start_month}.csv"
        df = pd.DataFrame(data)
        df.to_csv(filename, index=False)
        print(f"Data saved to {filename}")
        print(df.head())
    else:
        print("No valid data was collected.")

# Example usage
scrape_enforcement_actions(2013, 1)
```

possible adjustment

```
import pandas as pd
from datetime import datetime

# Load the merged enforcement actions file
df = pd.read_csv("merged_enforcement_actions.csv")

# Convert the 'Date' column to datetime format, ignoring errors
df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

# Set the start date to January 1, 2023
start_date = datetime(2023, 1, 1)

# Filter the DataFrame to only include rows with dates on or after the start
 ↪  date
df_since_2023 = df[df['Date'] >= start_date]

# Save the filtered data to a new CSV file for later use
filename = "enforcement_actions_since_2023.csv"
df_since_2023.to_csv(filename, index=False)
print(f"Data saved to {filename}")
```

```
# Display the total number of enforcement actions collected
print(f"Total enforcement actions collected: {len(df_since_2023)}")

# Find and display the earliest enforcement action
earliest_action = df_since_2023.sort_values(by='Date').iloc[0]
print("Earliest enforcement action:")
print(earliest_action)
# Data saved to enforcement_actions_since_2023.csv
# Total enforcement actions collected: 1277
# Earliest enforcement action:
# Title       Podiatrist Pays $90,000 To Settle False Billin...
# Date                                      2023-01-03 00:00:00
# Category                            Criminal and Civil Actions
# Link        https://oig.hhs.gov/fraud/enforcement/podiatri...
# Agency      U.S. Attorney's Office, Southern District of T...
# Name: 1420, dtype: object
```

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

```
import pandas as pd
from datetime import datetime
import pandas as pd

df = pd.read_csv("merged_enforcement_actions.csv")

df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

start_date = datetime(2021, 1, 1)

df_since_2021 = df[df['Date'] >= start_date]

#here is a new csv we could use in later for data since 2021
filename = "enforcement_actions_since_2021.csv"
df_since_2021.to_csv(filename, index=False)
print(f"Data saved to {filename}")

print(f"Total enforcement actions collected: {len(df_since_2021)}")

earliest_action = df_since_2021.sort_values(by='Date').iloc[0]
print("Earliest enforcement action:")
print(earliest_action)
# Data saved to enforcement_actions_since_2021.csv
```

```
# Total enforcement actions collected: 2533
# Earliest enforcement action:
# Title       The United States And Tennessee Resolve Claims...
# Date                                      2021-01-04 00:00:00
# Category                            Criminal and Civil Actions
# Link        https://oig.hhs.gov/fraud/enforcement/the-unit...
# Agency      U.S. Attorney's Office, Middle District of Ten...
# Name: 2656, dtype: object
```

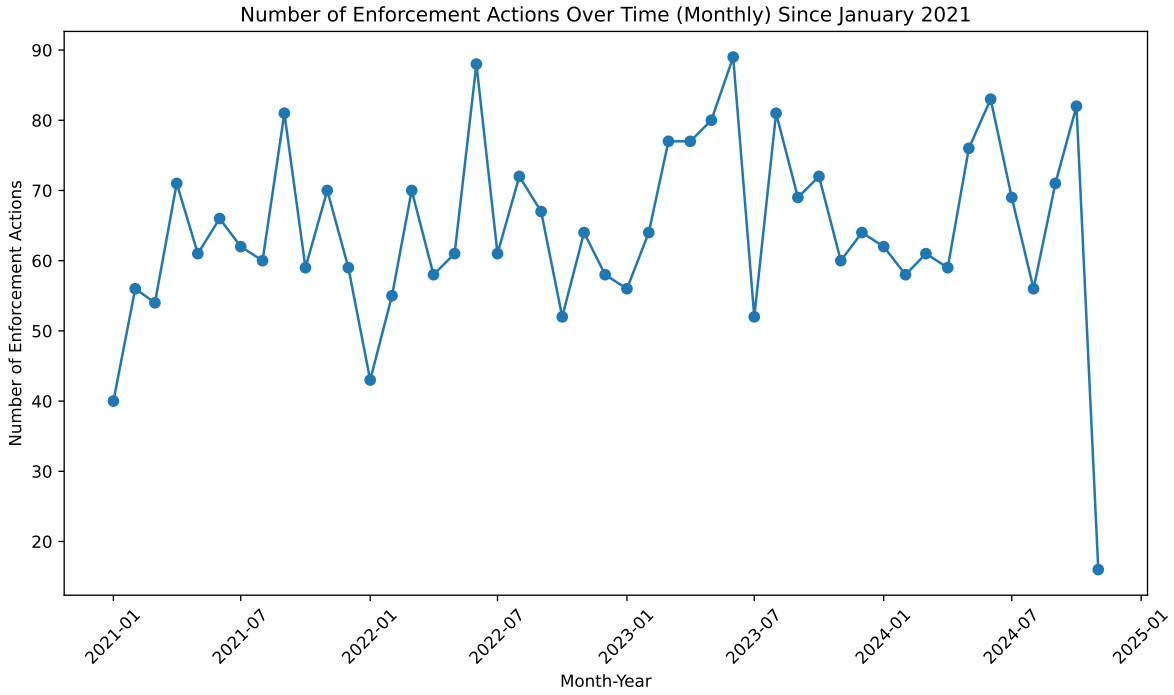
Step 3: Plot data based on scraped data

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

```
import os
print(os.getcwd())


import pandas as pd
import matplotlib.pyplot as plt

# Load the DataFrame with enforcement actions
df = pd.read_csv("enforcement_actions.csv")

# Convert the 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

# Filter data for actions since January 2021
df = df[df['Date'] >= '2021-01-01']

# Extract year and month for grouping
df['Year_Month'] = df['Date'].dt.to_period('M')  # 'M' for monthly period

# Count the number of actions for each month
monthly_counts =
 ↪  df.groupby('Year_Month').size().reset_index(name='Action_Count')

# Convert 'Year_Month' back to datetime for plotting
monthly_counts['Year_Month'] = monthly_counts['Year_Month'].dt.to_timestamp()

# Plotting the line chart
```

```
plt.figure(figsize=(10, 6))
plt.plot(monthly_counts['Year_Month'], monthly_counts['Action_Count'],
 ↪  marker='o', linestyle='-')
plt.title("Number of Enforcement Actions Over Time (Monthly) Since January
 ↪  2021")
plt.xlabel("Month-Year")
plt.ylabel("Number of Enforcement Actions")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
/Users/jeasonzhang/Desktop/ps5
```

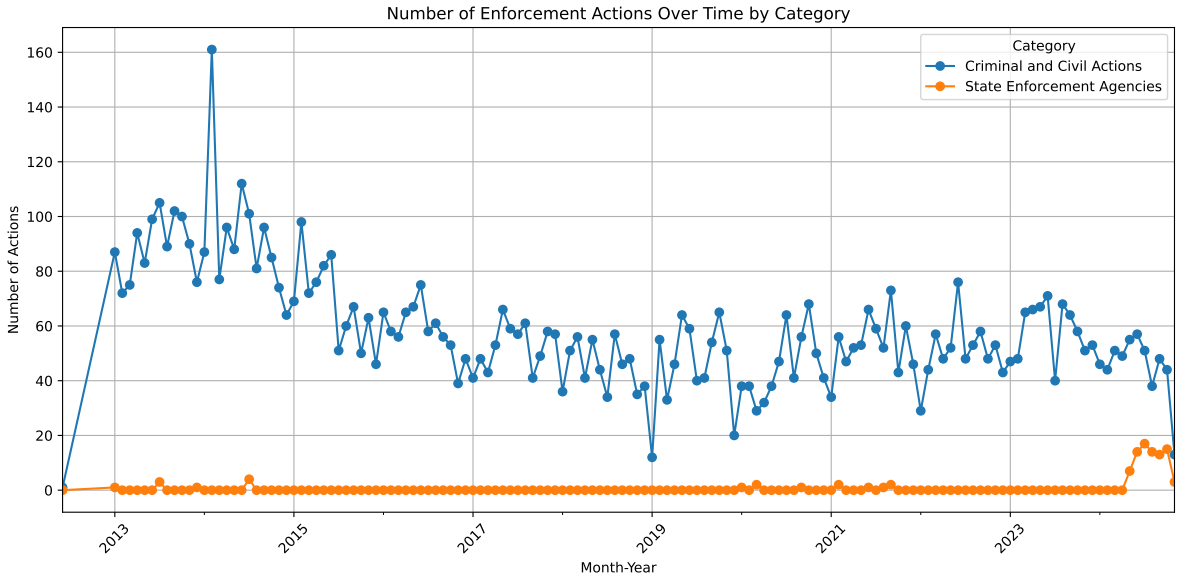


2. Plot the number of enforcement actions categorized: (PARTNER 1)

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

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('merged_enforcement_actions.csv')

df['Date'] = pd.to_datetime(df['Date'])
df['MonthYear'] = df['Date'].dt.to_period('M')

df['Category'] = df['Agency'].apply(lambda x: "State Enforcement Agencies" if
 ↪  "state of" in str(x).lower() else "Criminal and Civil Actions")

grouped = df.groupby(['MonthYear', 'Category']).size().unstack(fill_value=0)

# Plot
plt.figure(figsize=(12, 6))
grouped.plot(kind='line', marker='o', linestyle='-', ax=plt.gca())
plt.title('Number of Enforcement Actions Over Time by Category')
plt.xlabel('Month-Year')
plt.ylabel('Number of Actions')
plt.grid(True)
plt.legend(title='Category')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



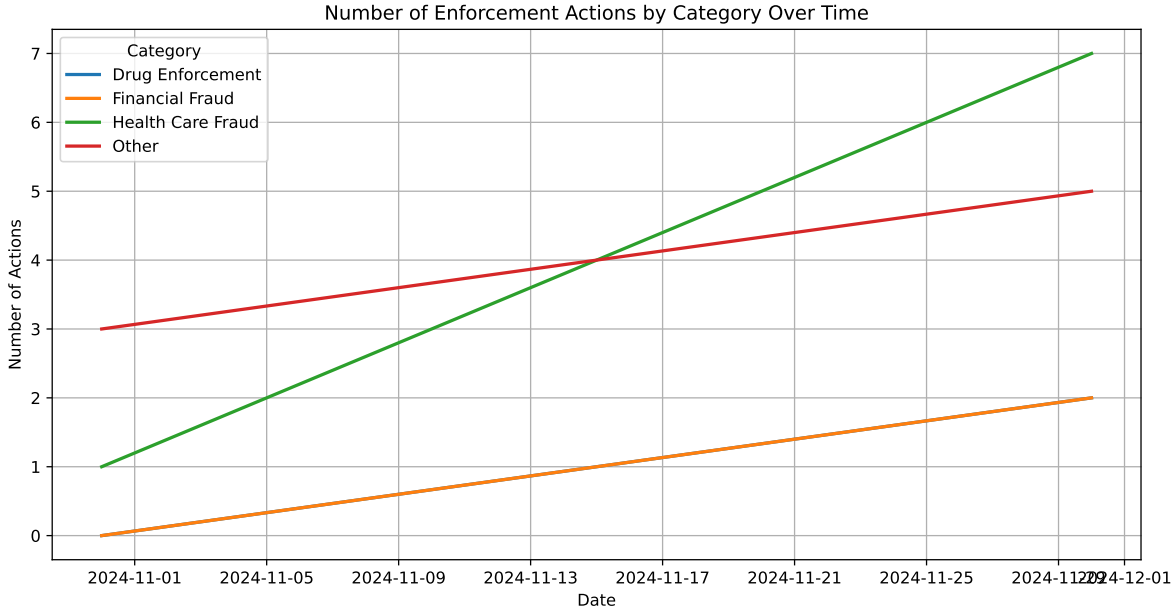
- based on five topics

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import matplotlib.pyplot as plt

#scrap data into five topics
def categorize_title(title):
    health_keywords = ["healthcare", "medicare", "medicaid"]
    financial_keywords = ["bank", "financial", "money", "fraud"]
    drug_keywords = ["drug", "narcotic", "substance"]
    bribery_keywords = ["bribery", "corruption", "kickback"]

    title_lower = title.lower()
    if any(word in title_lower for word in health_keywords):
        return "Health Care Fraud"
    elif any(word in title_lower for word in financial_keywords):
        return "Financial Fraud"
    elif any(word in title_lower for word in drug_keywords):
        return "Drug Enforcement"
    elif any(word in title_lower for word in bribery_keywords):
        return "Bribery/Corruption"
    else:
        return "Other"

def fetch_titles_and_categories(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    data = []

    for item in soup.find_all('li', class_='usa-card card--list
     ↪  pep-card--minimal mobile:grid-col-12'):
        title_tag = item.find("h2", class_="usa-card__heading").find("a")
        title = title_tag.get_text(strip=True)
        date_span = item.find("span", class_="text-base-dark
 ↪  padding-right-105")
        date = pd.to_datetime(date_span.get_text(strip=True),
 ↪  errors='coerce') if date_span else None
        category = categorize_title(title)
        data.append({
            'Title': title,
            'Date': date,
            'Category': category
```

```
        })

    return pd.DataFrame(data)


url = 'https://oig.hhs.gov/fraud/enforcement/'
df = fetch_titles_and_categories(url)

print(df.head())

#plot the data
def main():
    url = 'https://oig.hhs.gov/fraud/enforcement/'
    df = fetch_titles_and_categories(url)

    # Ensure 'Date' is a datetime object and set as DataFrame index
    df['Date'] = pd.to_datetime(df['Date'])
    df.set_index('Date', inplace=True)

    # Group by Category and resample by month, counting entries
    grouped = df.groupby('Category').resample('M').size()

    # Pivot the data to have dates on the index and categories as columns
    pivot = grouped.unstack(level=0, fill_value=0)

    # Plotting
    plt.figure(figsize=(12, 6))
    for column in pivot.columns:
        plt.plot(pivot.index, pivot[column], marker='', linewidth=2,
 ↪  label=column)
    plt.title('Number of Enforcement Actions by Category Over Time')
    plt.xlabel('Date')
    plt.ylabel('Number of Actions')
    plt.legend(title='Category')
    plt.grid(True)
    plt.show()

if __name__ == '__main__':
    main()
```

```
                                               Title       Date  \
0  Pharmacist and Brother Convicted of $15M Medic... 2024-11-08
```

```
1  Boise Nurse Practitioner Sentenced To 48 Month... 2024-11-07
2  Former Traveling Nurse Pleads Guilty To Tamper... 2024-11-07
3  Former Arlington Resident Sentenced To Prison ... 2024-11-07
4  Paroled Felon Sentenced To Six Years For Fraud... 2024-11-07

            Category
0  Health Care Fraud
1   Drug Enforcement
2              Other
3  Health Care Fraud
4    Financial Fraud
```



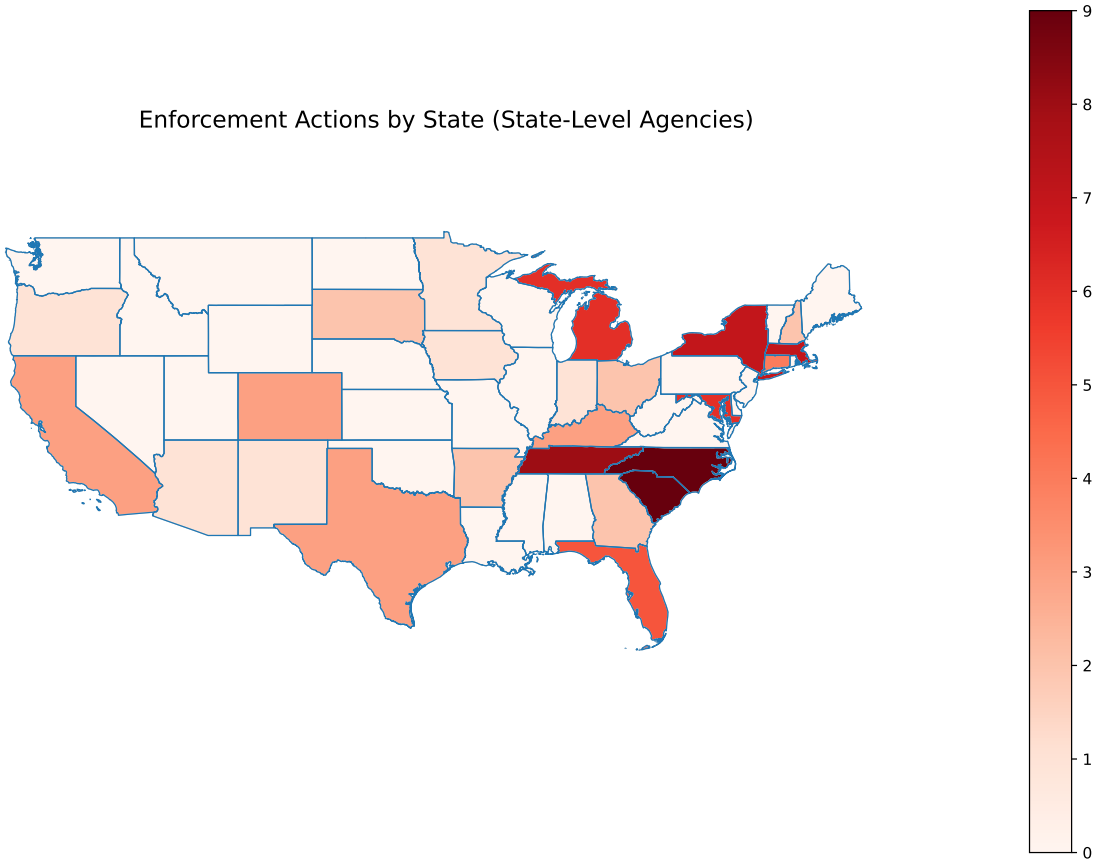
Step 4: Create maps of enforcement activity

1. Map by State (PARTNER 1)

```python
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# Load the state shapefile
state_gdf = gpd.read_file("cb_2018_us_state_500k.shp")
```

```
# Load the merged enforcement actions data
df_enforcement = pd.read_csv("merged_enforcement_actions.csv")

# Filter for state-level agencies in the "Agency" column
df_state_enforcement =
 ↪  df_enforcement[df_enforcement["Agency"].str.contains("State of",
 ↪  na=False)]

# Clean and standardize state names in the "Agency" column to match state
 ↪  abbreviations or names
df_state_enforcement["Cleaned_State"] =
 ↪  df_state_enforcement["Agency"].str.replace("State of ", "").str.strip()

# Group by "Cleaned_State" and count the occurrences
state_counts =
 ↪  df_state_enforcement["Cleaned_State"].value_counts().reset_index()
state_counts.columns = ["STATE_NAME", "Enforcement_Actions"]

# Merge the state counts with the state shapefile data
state_gdf = state_gdf.to_crs("EPSG:4326")  # Ensure CRS is set to match for
 ↪  plotting
merged_state_gdf = state_gdf.merge(state_counts, how="left", left_on="NAME",
 ↪  right_on="STATE_NAME")

# Set the NaN values in the "Enforcement_Actions" column to 0 for states with
 ↪  no enforcement actions
merged_state_gdf["Enforcement_Actions"].fillna(0, inplace=True)

# Plot the choropleth map for state-level enforcement actions
fig, ax = plt.subplots(1, 1, figsize=(15, 10))

# Plot the data with adjusted color scale
merged_state_gdf.boundary.plot(ax=ax, linewidth=0.8)
merged_state_gdf.plot(column="Enforcement_Actions", cmap="Reds",
 ↪  linewidth=0.8, ax=ax, edgecolor="0.8", legend=True)

# Adjust limits to focus on the contiguous United States
ax.set_xlim(-130, -60)
ax.set_ylim(20, 55)

# Title and labels
```

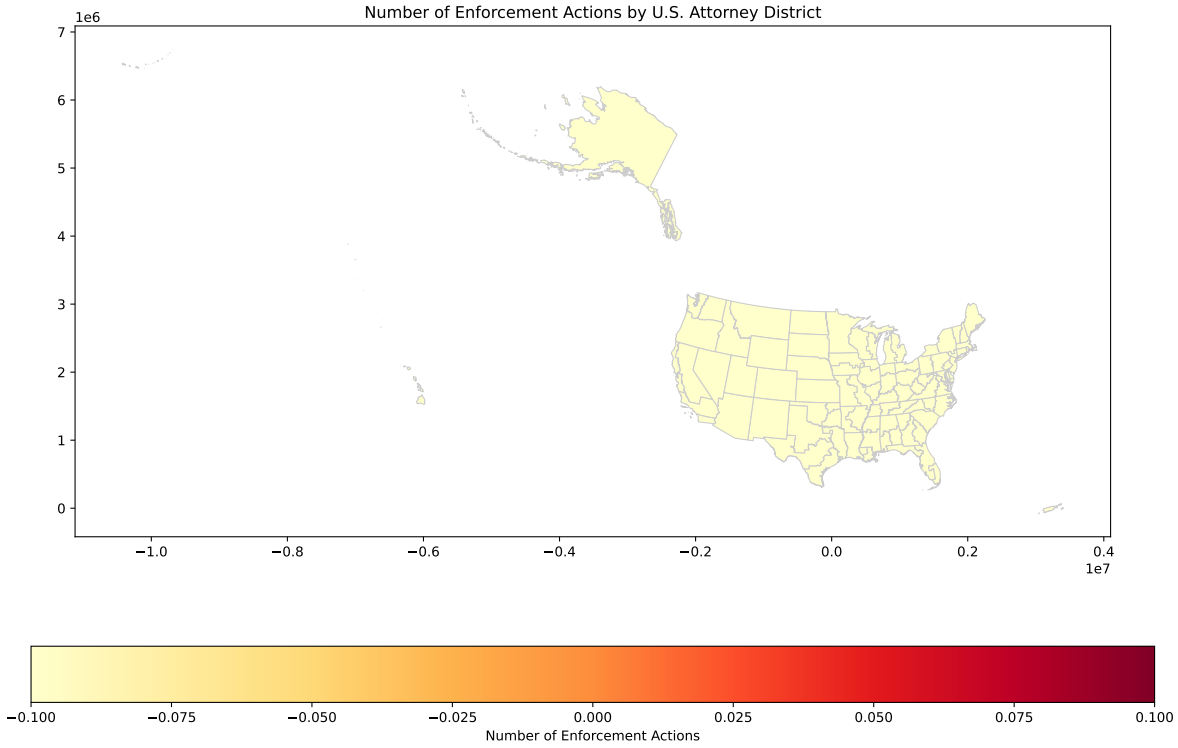
```
ax.set_title("Enforcement Actions by State (State-Level Agencies)",
 ↪  fontsize=15)
ax.axis("off")

# Show the plot
plt.show()
```



2. Map by District (PARTNER 2)

```
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt

# Load the state shapefile and district CSV data
```

```
state_gdf = gpd.read_file("cb_2018_us_state_500k.shp")
district_gdf = gpd.read_file("geo_export.shp")
enforcement_data = pd.read_csv("merged_enforcement_actions.csv")

district_enforcement_data =
 ↪  enforcement_data[enforcement_data["Agency"].str.contains("District",
 ↪  case=False, na=False)]

district_enforcement_data["Cleaned_District"] = (
    district_enforcement_data["Agency"]
    .str.replace("U.S. Attorney's Office, ", "", regex=False)
    .str.replace(" District", "", regex=False)
    .str.strip()
)

district_counts =
 ↪  district_enforcement_data.groupby("Cleaned_District").size().reset_index(name="Count")

# Rename the district column in the shapefile for clarity, if needed
district_gdf = district_gdf.rename(columns={"district_n": "District"})

# Merge the district counts with the district shapefile data using 'District'
district_gdf = district_gdf.merge(district_counts, how="left",
 ↪  left_on="District", right_on="Cleaned_District")

# Fill NaN values in 'Count' with 0 for visualization
district_gdf["Count"] = district_gdf["Count"].fillna(0)

# Project to Albers Equal Area for better visualization of the U.S.
 ↪  (optional)
district_gdf = district_gdf.to_crs("EPSG:5070")

# Plot the choropleth map for district-level enforcement actions
fig, ax = plt.subplots(1, 1, figsize=(15, 10))
district_gdf.plot(column="Count", cmap="YlOrRd", linewidth=0.8, ax=ax,
 ↪  edgecolor="0.8", legend=True,
                  legend_kwds={'label': "Number of Enforcement Actions",
 ↪  'orientation': "horizontal"})

plt.title("Number of Enforcement Actions by U.S. Attorney District")
plt.show()
```



Extra Credit

1. Merge zip code shapefile with population

```
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt


shapefile_path = '/Users/jeasonzhang/Desktop/ps5/gz_2010_us_860_00_500k.dbf'
gdf = gpd.read_file(shapefile_path)

csv_path = '/Users/jeasonzhang/Desktop/ps5/zip code- level population
 ↪  data.csv'
pop_data = pd.read_csv(csv_path)

pop_data.rename(columns={'P1_001N': 'Population'}, inplace=True)
```

```
gdf['GEO_ID'] = gdf['GEO_ID'].astype(str)
pop_data['GEO_ID'] = pop_data['GEO_ID'].astype(str)

# Merge
merged_data = gdf.merge(pop_data, on='GEO_ID', how='left')

print(merged_data.head())
```

```
           GEO_ID  ZCTA5 NAME_x   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 NAME_y Population  \
0  POLYGON ((-72.62734 42.16203, -72.62764 42.162...    NaN        NaN
1  POLYGON ((-72.95393 42.34379, -72.95385 42.343...    NaN        NaN
2  POLYGON ((-72.68286 42.37002, -72.68287 42.369...    NaN        NaN
3  POLYGON ((-72.39529 42.18476, -72.39653 42.183...    NaN        NaN
4  MULTIPOLYGON (((-72.39191 42.08066, -72.39077 ...    NaN        NaN

   Unnamed: 3
0         NaN
1         NaN
2         NaN
3         NaN
4         NaN
```

2. Conduct spatial join

```
import geopandas as gpd
import pandas as pd
import os

# Check current directory contents (optional, for debugging)
print(os.listdir())

# Load zip code shapefile and district shapefile
zip_codes = gpd.read_file('gz_2010_us_860_00_500k.shp')
```

```
districts = gpd.read_file('geo_export.shp')
print("Zip Codes Columns:", zip_codes.columns)
print("Districts Columns:", districts.columns)

# Load and prepare population data
population_data = pd.read_csv('zip code- level population data.csv')
population_data['GEO_ID'] = population_data['GEO_ID'].str[-5:]  # Format
 ↪  GEO_ID to match ZCTA5 format
print("Population Data Columns:", population_data.columns)

# Merge zip codes with population data on ZCTA5
zip_codes = zip_codes.merge(population_data, left_on='ZCTA5',
 ↪  right_on='GEO_ID', how='left')

# Ensure both GeoDataFrames are using the same CRS for spatial join
districts = districts.to_crs(zip_codes.crs)

# Conduct spatial join between zip codes and districts
joined_data = gpd.sjoin(zip_codes, districts, how='inner',
 ↪  predicate='intersects')

# Aggregate population data by district
district_population =
 ↪  joined_data.groupby('judicial_d')['P1_001N'].sum().reset_index()
district_population.columns = ['district_id', 'total_population']

# Create the 'ps5' directory if it does not exist
if not os.path.exists('ps5'):
    os.makedirs('ps5')

# Export results to the 'ps5' directory
district_population.to_csv('ps5/district_population.csv', index=False)
print("District Population Data:")
print(district_population)
```

```
['geo_export.prj', 'debug.ipynb', '.Rhistory', 'cb_2018_us_state_500k.dbf',
'.DS_Store', 'ps5_template.qmd', 'csvmerge.ipynb', '3.1.png', 'ps5.pdf',
'cb_2018_us_state_500k.shp.iso.xml', 'gz_2010_us_860_00_500k.prj',
'enforcement_actions_since_2013.csv',
'geo_export_5882ccf2-1a9a-4c56-b638-180b2647064c.shp', 'zip code- level
population data.csv', 'geo_export_5882ccf2-1a9a-4c56-b638-180b2647064c.shx',
'ps5', 'ps5_template.pdf', 'cb_2018_us_state_500k.shx',
'district_population.csv',
'US_Attorney_Districts_Shapefile_simplified_20241110.csv',
'enforcement_actions.csv', 'cb_2018_us_state_500k.shp',
'cb_2018_us_state_500k.cpg', 'webscraping_2.pdf', 'geo_export.shp',
'geo_export.cpg', 'webscraping_1.pdf', 'geo_export.shx',
'cb_2018_us_state_500k.shp.ea.iso.xml',
'enforcement_actions_with_agency.csv', 'gz_2010_us_860_00_500k.shx',
'gz_2010_us_860_00_500k', 'enforcement_actions_2013_1.csv',
'gz_2010_us_860_00_500k.shp', 'enforcement_actions_2023_01.csv',
'enforcement_actions_agencies.csv', 'enforcement_actions_since_2023.csv',
'gz_2010_us_860_00_500k.dbf', 'merged_enforcement_actions.csv',
'enforcement_actions_since_2021.csv', 'enforcement_actions_2013_01.csv',
'ps5_template.quarto_ipynb', 'ps5_template1.qmd',
'gz_2010_us_860_00_500k.xml', 'cb_2018_us_state_500k.prj', 'geo_export.dbf']
Zip Codes Columns: Index(['GEO_ID', 'ZCTA5', 'NAME', 'LSAD', 'CENSUSAREA',
'geometry'], dtype='object')
Districts Columns: Index(['statefp', 'judicial_d', 'aland', 'awater',
'state', 'chief_judg',
       'nominating', 'term_as_ch', 'shape_leng', 'shape_area', 'abbr',
       'district_n', 'shape__are', 'shape__len', 'geometry'],
      dtype='object')
Population Data Columns: Index(['GEO_ID', 'NAME', 'P1_001N', 'Unnamed: 3'],
dtype='object')
District Population Data:
                       district_id  \
0   Central District of California
1     Central District of Illinois
2               District of Alaska
3              District of Arizona
4             District of Colorado
..                             ...
86   Western District of Tennessee
87       Western District of Texas
88    Western District of Virginia
89  Western District of Washington
90   Western District of Wisconsin
```

```
                                      total_population
0   71701201750885617642593966359176445678860455546...
1   32191866861763053013949967258369231436172366681...
2   24183423756297343942178698611716517015892116698...
3   29651142856402322211223732655629204481637112771...
4   19172861042462331205867112713056552026736258336...
..                                                  ...
86  44641156191101390148322370865443721155717184998...
87  23080399354037826989902773637951110389423818336...
88  45546536151716266819214748204821236297201629889...
89  61626965149436086415624399360733913640871265188...
90  18510249170851702297226151923999547748659668411...

[91 rows x 2 columns]
```

3. Map the action ratio in each district