30538 Problem Set 5: Web Scraping

Surya Hardiansyah and Astari Raihanah

2024-11-08

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): Surya Hardiansyah | sur
    - Partner 2 (name and cnet ID): Astari Raihanah | astari
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: **SH AR**
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: **00** Late coins left after submission: **04**
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

# Prepare for Scraping
url = "https://oig.hhs.gov/fraud/enforcement/"
response = requests.get(url)

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

# Find all relevant elements within the specified container
containers = soup.find_all("div", class_="usa-card__container")

# Extract data from within the specified containers
titles = [container.find("h2",
 ↪  class_="usa-card__heading").get_text(strip=True) for container in
 ↪  containers]
dates = [container.find("span", class_="text-base-dark
 ↪  padding-right-105").get_text(strip=True) for container in containers]
categories = [container.find("li", class_="display-inline-block usa-tag
 ↪  text-no-lowercase text-base-darkest bg-base-lightest
 ↪  margin-right-1").get_text(strip=True) for container in containers]

# Extract links within the specified containers
link_content = [
    "https://oig.hhs.gov" + a["href"]
    for container in containers
```

```
    for a in container.find_all("a", href=True)
    if "/fraud/enforcement/" in a["href"]]

# Combine extracted data into a DataFrame
data = {
    "Title": titles,
    "Date": dates,
    "Category": categories,
    "Link": link_content[:len(titles)]  # Ensure the number of links matches
    ↪ the titles
}
df = pd.DataFrame(data)

# Print the head of df
print(df.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

                    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

                                                Link
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 (PARTNER 1)

```python
# Function to extract agency information from a given link
def extract_agency_name(link):
    response = requests.get(link)
    soup = BeautifulSoup(response.text, 'lxml')

    # Find the div containing the action details
    details_div = soup.find('div', class_="margin-top-5 padding-y-3
 ↪  border-y-1px border-base-lighter")
    if details_div:
        # Find all <li> elements within the div
        li_elements = details_div.find_all('li')
        for li in li_elements:
            # Check if the <li> contains 'Agency:' and extract the text after
             ↪  it
            if 'Agency:' in li.get_text():
                return li.get_text().replace('Agency:', '').strip()

    return 'N/A'

# Append agency info into df
df["Agency"] = df["Link"].apply(extract_agency_name)

# Print the head of df
print(df.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

                    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

                                                Link  \
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)

```
DEFINE FUNCTION scrape_enforcement_actions(start_month, start_year)

1. Check if start year is valid:
IF start_year is before 2013:
Print a message: "Please restrict the start year to >= 2013"
RETURN (exit the function early)

2. Initialize variables:
SET results to an empty list (this will hold our scraped data)
SET start_date to the first day of start_month in start_year
SET current_date to today's date

3. Begin scraping from page 1:
SET page to 1 (we'll start at the first page)

4. Start looping through pages:
WHILE True (this will keep going until we break out):
- SET url to the enforcement page for the current page
- Print: "Scraping {url}..."
- GET the page content and parse it with BeautifulSoup

5. Find all enforcement action entries on the page:
- SET containers to all div elements with a specific class (these hold each
enforcement action)
```

```
6. If no containers are found, stop:
- IF containers is empty:
    - BREAK the loop (we're done scraping)

7. Process each container found on the page:
- FOR EACH container in containers:
    - Extract:
        - title from the heading
        - date from a specific span tag
        - category from a list tag
        - link from the anchor tag's URL
    - IF date is missing, skip to the next container

8. Convert and check date:
- Convert date to a datetime object (date_obj)
- IF date_obj is earlier than start_date:
        - Print: "Encountered data older than the specified start date.
        Stopping scrape."
        - BREAK the loop (no need to look further)

9. If date is within range, save data:
- IF start_date <= date_obj <= current_date:
        - Add title, date, category, and link to results

10. Go to the next page:
- Increment page by 1
- WAIT for a second to avoid overloading the site

11. Retrieve agency names:
- FOR EACH link in results:
    - Call a separate function to retrieve the agency name from the link
    - Append the agency name to the corresponding entry in results

12. Save results as CSV:
- Convert results to a DataFrame
- Save as a CSV file named with the given start_year and start_month
- Print: "Data saved to [filename]"
```

In this way, we use a `while` loop to iterate through pages until we reach a stopping condition (either no more relevant data or data older than the start date). This structure allows us to handle the pages dynamically, stopping once we've collected all the needed data without requiring a predefined number of iterations.

- b. Create Dynamic Scraper (PARTNER 2)

```
import time
from datetime import datetime

# Function to scrape enforcement actions starting from a specific month/year
def scrape_enforcement_actions(start_month, start_year):
    # Ensure start date is within range
    if start_year < 2013:
        print("Please restrict the start year to >= 2013")
        return

    # Initialize variables
    results = []
    start_date = datetime(start_year, start_month, 1)
    current_date = datetime.now()

    # Start from page 1 and continue ascendingly
    page = 1
    while True:
        url = f"https://oig.hhs.gov/fraud/enforcement/?page={page}"
        print(f"Scraping {url}...")
        response = requests.get(url)
        soup = BeautifulSoup(response.text, 'lxml')

        # Find all enforcement actions containers
        containers = soup.find_all('div', class_="usa-card__container")

        # Stop if there are no containers
        if not containers:
            break

        # Extract data from each container on the page
        for container in containers:
            title_elem = container.find('h2', class_="usa-card__heading")
            date_elem = container.find('span', class_="text-base-dark
 ↪  padding-right-105")
            category_elem = container.find('li', class_="display-inline-block
 ↪  usa-tag text-no-lowercase text-base-darkest bg-base-lightest
 ↪  margin-right-1")
            link_elem = container.find('a', href=True)
```

```
            # Extract and validate each element
            title = title_elem.get_text(strip=True) if title_elem else "N/A"
            date = date_elem.get_text(strip=True) if date_elem else None
            category = category_elem.get_text(strip=True) if category_elem
↪   else "N/A"
            link = "https://oig.hhs.gov" + link_elem['href'] if link_elem
↪   else "N/A"

            # Only proceed if the date is present
            if date:
                # Convert date string to datetime object
                date_obj = datetime.strptime(date, "%B %d, %Y")

                # Stop scraping if the date is older than the start date
                if date_obj < start_date:
                    print("Encountered data older than the specified start
                     ↪  date. Stopping scrape.")
                    break

                # Append data if it's within the range
                if start_date <= date_obj <= current_date:
                    results.append({
                        'Title': title,
                        'Date': date,
                        'Category': category,
                        'Link': link,
                        'Agency': None
                    })

        # Check if we stopped due to an older date and should break the loop
        if containers and date and date_obj < start_date:
            break

        # Move to the next page and wait before the next request
        page += 1
        time.sleep(1)

    # Second phase: retrieve agency names using Partner 1's scraping function
    for entry in results:
        entry['Agency'] = extract_agency_name(entry['Link'])

    # Convert results to DataFrame
```

```
    df = pd.DataFrame(results)

    # Save the DataFrame as CSV
    output_filename = f"enforcement_actions_{start_year}_{start_month}.csv"
    df.to_csv(output_filename, index=False)
    print(f"Data saved to {output_filename}")

    return df
```

```
# Run the function to scrape from January 2023
autoscrape_df = scrape_enforcement_actions(1, 2023)
```

We scraped our data on November 8th

```
# Substitute with scrape_enforcement_actions(1, 2023) result for faster
 ↪ knitting
file202301 = pd.read_csv("enforcement_actions_2023_1.csv")

# Convert 'Date' column to datetime format
file202301["Date"] = pd.to_datetime(file202301["Date"])

# Print results and check the earliest enforcement action
print(f"Total enforcement actions scraped: {len(file202301)}")
print("Earliest enforcement action scraped:")
print(file202301.sort_values(by='Date').head(1))
```

```
Total enforcement actions scraped: 1533
Earliest enforcement action scraped:
                                                  Title       Date  \
1532  Podiatrist Pays $90,000 To Settle False Billin... 2023-01-03

                        Category  \
1532  Criminal and Civil Actions

                                                  Link  \
1532  https://oig.hhs.gov/fraud/enforcement/podiatri...

                                                Agency
1532  U.S. Attorney's Office, Southern District of T...
```

In this function, we first gather all titles, dates, categories, and links for each enforcement action without immediately fetching agency names. This approach is more efficient because

it minimizes the number of sequential HTTP requests. After creating the initial database, we loop through each article's link to retrieve the agency name separately.

We scraped our data on November 8th. My partner's autoscrape code works in scraping and crawling the enforcement action details from January 2023 onward. It has 1,533 rows based on the title, date, category, link, and agency name for each enforcement action. "Podiatrist Pays $90,000 To Settle False Billin…" is the earliest data from January 3rd, 2023.

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

```
# Run the function to scrape from January 2021
autoscrape_df = scrape_enforcement_actions(1, 2021)
```

We scraped our data on November 8th

```
# Substitute with scrape_enforcement_actions(1, 2021) result for faster
 ↪  knitting
file202101 = pd.read_csv("enforcement_actions_2021_1.csv")

# Convert 'Date' column to datetime format
file202101["Date"] = pd.to_datetime(file202101["Date"])

# Print results and check the earliest enforcement action
print(f"Total enforcement actions scraped: {len(file202101)}")
print("Earliest enforcement action scraped:")
print(file202101.sort_values(by='Date').head(1))
```

```
Total enforcement actions scraped: 3021
Earliest enforcement action scraped:
                                                  Title       Date  \
3020  The United States And Tennessee Resolve Claims... 2021-01-04

                        Category  \
3020  Criminal and Civil Actions

                                                  Link  \
3020  https://oig.hhs.gov/fraud/enforcement/the-unit...

                                                Agency
3020  U.S. Attorney's Office, Middle District of Ten...
```

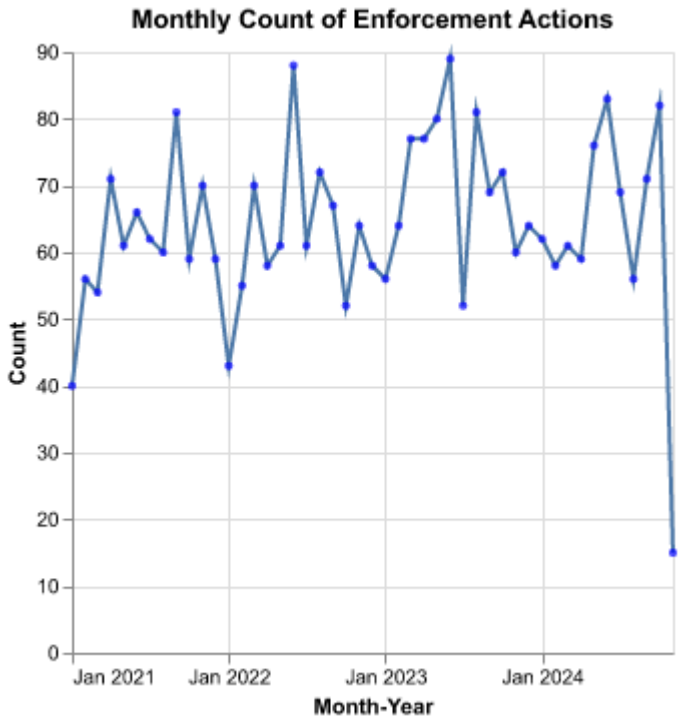
We scraped our data on November 8th. My partner's autoscrape code works in scraping and crawling the enforcement action details from January 2021 onward. It has 3,021 rows based on the title, date, category, link, and agency name for each enforcement action. "The United States And Tennessee Resolve Claims..." is the earliest data from January 4th, 2021.

Step 3: Plot data based on scraped data

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

```
# Convert Date column to datetime format
file202101["Date"] = pd.to_datetime(file202101["Date"], format="%B %d, %Y")

# Extract year and month for grouping
file202101["YearMonth"] = file202101["Date"].dt.to_period("M")

# Aggregate by YearMonth
monthly_count =
 ↪  file202101.groupby("YearMonth").size().reset_index(name="Count")

# Convert YearMonth back to datetime for plotting
monthly_count["YearMonth"] = monthly_count["YearMonth"].dt.to_timestamp()

# Create the line chart with dots
line_chart = alt.Chart(monthly_count).mark_line().encode(
    alt.X(
        'YearMonth:T',
        title="Month-Year",
        axis=alt.Axis(
            format='%b %Y',
            tickCount='year',
            grid=True
            )
    ),
    alt.Y('Count', title='Count')
).properties(
    title="Monthly Count of Enforcement Actions"
)

# Add points to indicate each data point on the line
points = alt.Chart(monthly_count).mark_point(size=5, color='blue').encode(
    x='YearMonth:T',
```

```
    y='Count:Q'
)

# Combine line and points
chart_with_dots = line_chart + points
chart_with_dots
```



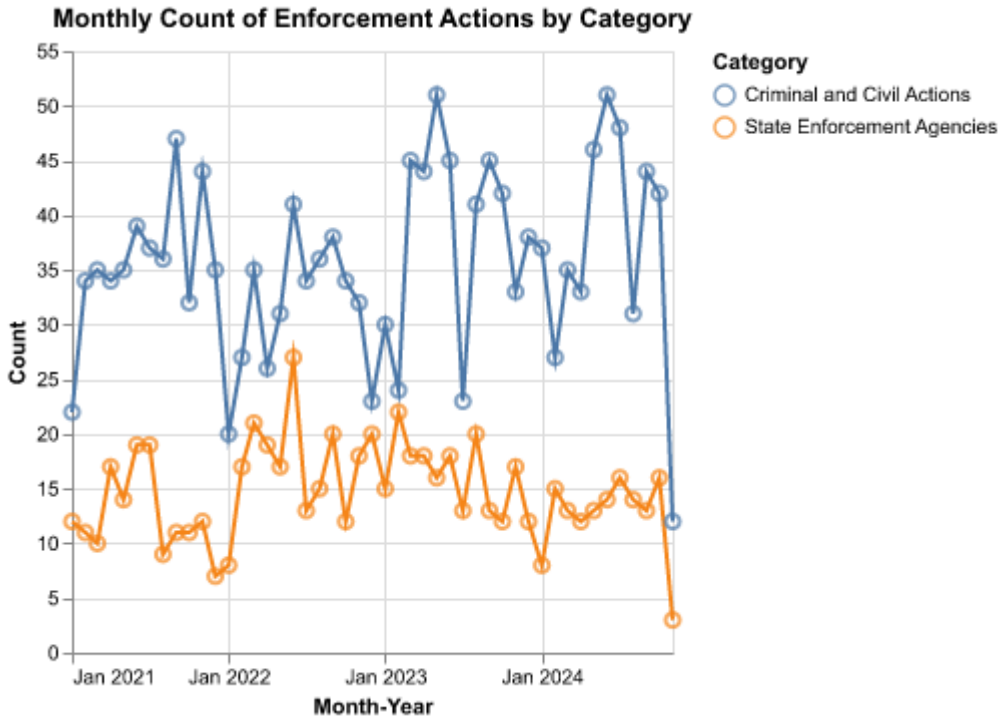
The line chart reveals the monthly count of Enforcement Actions, with periodic peaks and troughs. There is a consistent pattern of variation where some months showing spikes, while other months dip lower. The sharp decline toward the end of the timeline might suggests incomplete data for the most recent months.

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

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

```
# Filter for the relevant categories
filtered_df = file202101[file202101["Category"].isin(["Criminal and Civil
 ↪  Actions", "State Enforcement Agencies"])]
```

```
# Aggregate by YearMonth and Category
monthly_count = filtered_df.groupby(["YearMonth",
 ↪  "Category"]).size().reset_index(name="Count")

# Convert YearMonth to datetime for plotting
monthly_count["YearMonth"] = monthly_count["YearMonth"].dt.to_timestamp()

# Create the line chart with separate lines for each category
line_chart = alt.Chart(monthly_count).mark_line().encode(
    x=alt.X(
        'YearMonth:T',
        title="Month-Year",
        axis=alt.Axis(
            format='%b %Y',
            tickCount='year',
            grid=True
        )
    ),
    y=alt.Y('Count:Q', title='Count'),
    color=alt.Color('Category:N', title="Category")
).properties(
    title="Monthly Count of Enforcement Actions by Category"
)

# Add points to indicate each data point on the line
points = alt.Chart(monthly_count).mark_point(size=50).encode(
    x='YearMonth:T',
    y='Count:Q',
    color='Category:N'
)

# Combine line and points
chart_with_dots = line_chart + points
chart_with_dots
```

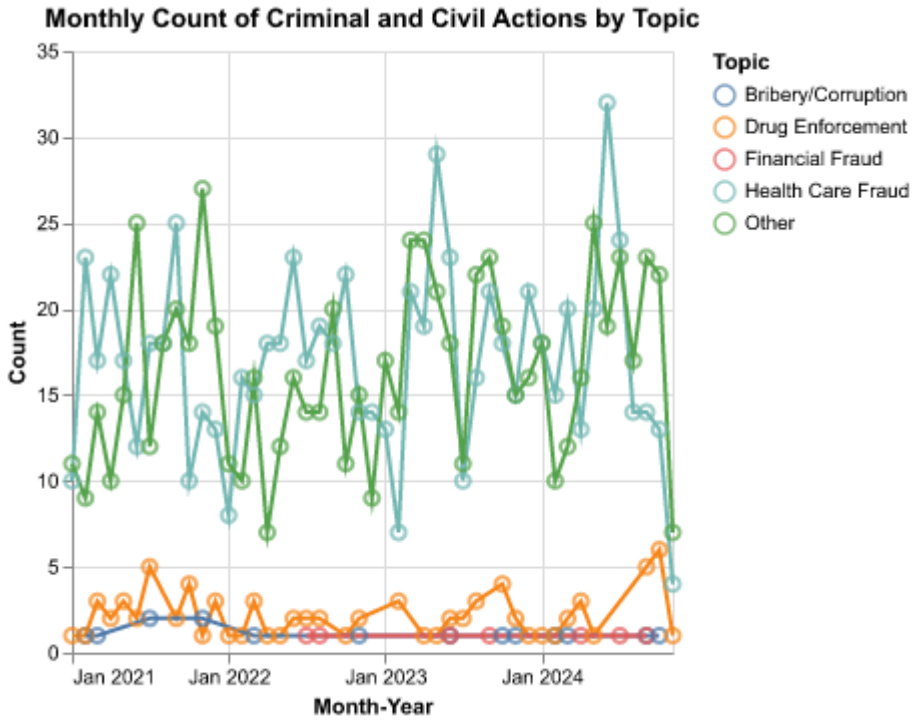


The chart shows that the Criminal and Civil Actions consistently having higher monthly counts (around 20-50 actions per month) than State Enforcement Agencies (around 5-20 actions per month). This difference highlights that federal criminal and civil cases are generally more frequent than state-enforced action over observation period. Both categories shows a sharp drop in the most recent months, which could indicate incomplete data or a real decrease in enforcement actions.

- based on five topics

```
# Filter for the "Criminal and Civil Actions" category only
filtered_df = file202101[file202101["Category"] == "Criminal and Civil
 ↪  Actions"]

# Function to classify topics based on keywords in the title
def classify_topic(title):
    title = title.lower()
    if "health" in title or "care" in title:
        return "Health Care Fraud"
    elif "financial" in title or "bank" in title:
        return "Financial Fraud"
    elif "drug" in title or "narcotic" in title:
        return "Drug Enforcement"
```

```
    elif "bribe" in title or "corruption" in title:
        return "Bribery/Corruption"
    else:
        return "Other"

# Apply the classification function to create a new Topic column
filtered_df["Topic"] = filtered_df["Title"].apply(classify_topic)

# Aggregate by YearMonth and Topic
monthly_count = filtered_df.groupby(["YearMonth",
 ↪  "Topic"]).size().reset_index(name="Count")

# Convert YearMonth to datetime for plotting
monthly_count["YearMonth"] = monthly_count["YearMonth"].dt.to_timestamp()

# Create the line chart with separate lines for each topic
line_chart = alt.Chart(monthly_count).mark_line().encode(
    x=alt.X(
        'YearMonth:T',
        title="Month-Year",
        axis=alt.Axis(
            format='%b %Y',
            tickCount='year',
            grid=True
        )
    ),
    y=alt.Y('Count:Q', title='Count'),
    color=alt.Color('Topic:N', title="Topic")
).properties(
    title="Monthly Count of Criminal and Civil Actions by Topic"
)

# Add points to indicate each data point on the line
points = alt.Chart(monthly_count).mark_point(size=50).encode(
    x='YearMonth:T',
    y='Count:Q',
    color='Topic:N'
)

# Combine line and points
chart_with_dots = line_chart + points
chart_with_dots
```

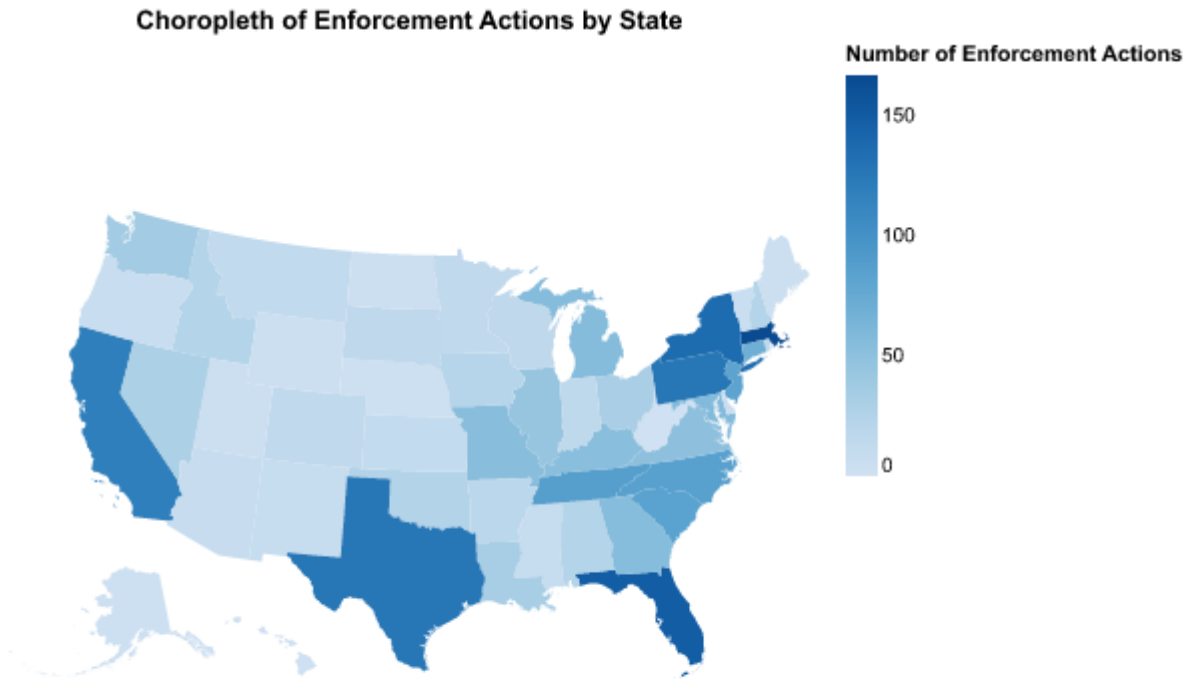


Step 4: Create maps of enforcement activity

1. Map by State (PARTNER 1)

```
import geopandas as gpd

# Read shapefile
state_shp = gpd.read_file("cb_2018_us_state_500k.shp")

# Prepare list of unique state names from state_shp
state_list = state_shp["NAME"].unique()

# Create a function to match each agency name with a state name
def assign_state(agency):
    for state in state_list:
        if state in agency:
            return state
    return ""
```

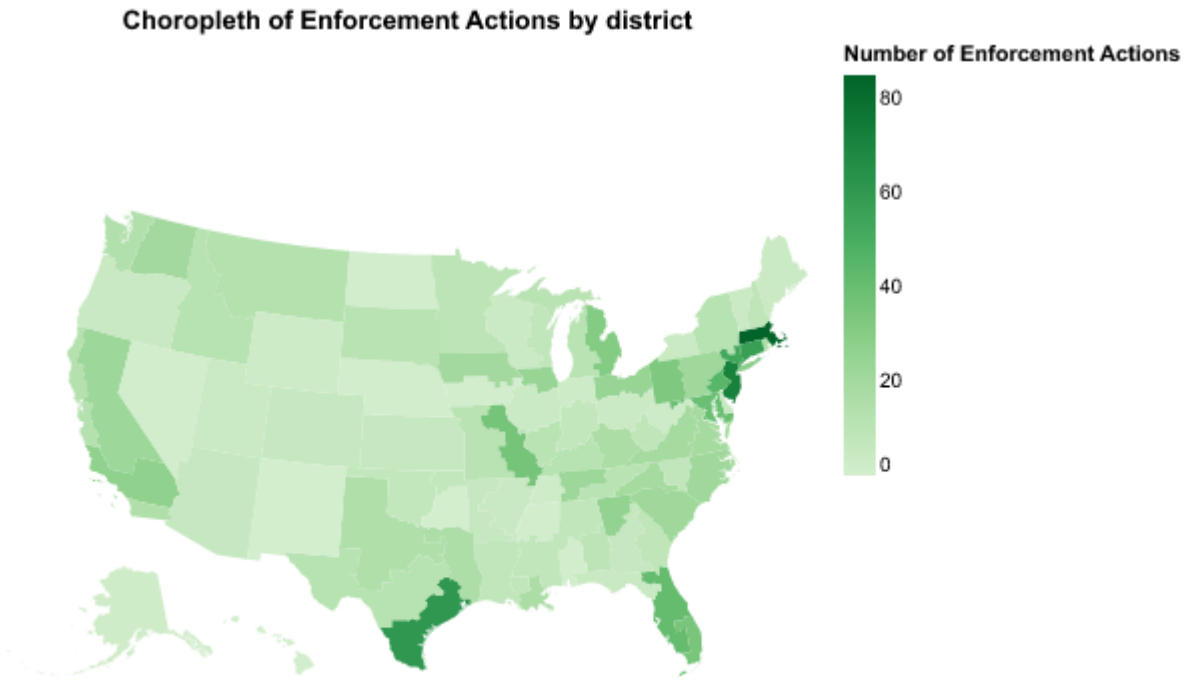
```
# Ensure 'Agency' column has no NaN values by filling them with an empty
 ↪ string
file202101["Agency"] = file202101["Agency"].fillna("")

# Apply the function to the "Agency" column in file202101
file202101["State"] = file202101["Agency"].apply(assign_state)

# Calculate the number of enforcement actions per state
state_counts = file202101["State"].value_counts().reset_index()
state_counts.columns = ["State", "Count"]

# Merge with state shapefile data
state_shp_merged = state_shp.merge(state_counts, left_on="NAME",
 ↪ right_on="State", how="left").fillna(0)

# Convert geometries to a simplified format and prepare for Altair
state_shp_merged = state_shp_merged.to_crs("EPSG:4326")  # Convert to WGS84
 ↪ for worldwide compatibility

#
 ↪ https://medium.com/dataexplorations/creating-choropleth-maps-in-altair-eeb7085779a1
# https://altair-viz.github.io/gallery/choropleth.html
# Use Altair to create a choropleth map
choropleth = alt.Chart(state_shp_merged).mark_geoshape().encode(
    color=alt.Color("Count:Q", title="Number of Enforcement Actions",
 ↪ scale=alt.Scale(scheme="blues")),
    tooltip=["NAME:N", "Count:Q"]
).project(
    type="albersUsa"
).properties(
    title="Choropleth of Enforcement Actions by State",
    width=400,
    height=400
)

choropleth
```



2. Map by District (PARTNER 2)

```
# Read shapefiles
district_shp =
 ↪  gpd.read_file("geo_export_c56ea527-1283-4852-8400-507244d1e4f5.shp")

# Prepare list of unique district names from district_shp
district_list = district_shp["judicial_d"].unique()

# Create a function to match each agency name with a district name
def assign_district(agency):
    for district in district_list:
        if district in agency:
            return district
    return ""
```

```
# Apply the function to the "Agency" column in file202101
file202101["District"] = file202101["Agency"].apply(assign_district)

# Calculate the number of enforcement actions per district
district_counts = file202101["District"].value_counts().reset_index()
district_counts.columns = ["District", "Count"]

# Merge with district shapefile data
district_shp_merged = district_shp.merge(district_counts,
 ↪  left_on="judicial_d", right_on="District", how="left").fillna(0)

# Convert geometries to a simplified format and prepare for Altair
district_shp_merged = district_shp_merged.to_crs("EPSG:4326")  # Convert to
 ↪  WGS84 for worldwide compatibility

# Use Altair to create a choropleth map
choropleth = alt.Chart(district_shp_merged).mark_geoshape().encode(
    color=alt.Color("Count:Q", title="Number of Enforcement Actions",
 ↪  scale=alt.Scale(scheme="greens")),
    tooltip=["NAME:N", "Count:Q"]
).project(
    type="albersUsa"
).properties(
    title="Choropleth of Enforcement Actions by district",
    width=400,
    height=400
)

choropleth
```



Extra Credit

1. Merge zip code shapefile with population

```
# Read shapefile and csv
zip_shp = gpd.read_file("gz_2010_us_860_00_500k.shp")
zip_pop = pd.read_csv("DECENNIALDHC2020.P1-Data.csv")

# Clean up the `NAME` column in zip_pop to only contain ZIP codes
zip_pop["ZIP"] = zip_pop["NAME"].str.replace("ZCTA5 ", "", regex=False)

# Merge the data on ZIP codes
zip_shp_merged = zip_shp.merge(zip_pop, left_on="ZCTA5", right_on="ZIP",
 ↪  how="inner")
```

```
# Inspect the merged data
print(zip_shp_merged.head())
```

```
         GEO_ID_x  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        GEO_ID_y  \
0  POLYGON ((-72.62734 42.16203, -72.62764 42.162...  860Z200US01040
1  POLYGON ((-72.95393 42.34379, -72.95385 42.343...  860Z200US01050
2  POLYGON ((-72.68286 42.37002, -72.68287 42.369...  860Z200US01053
3  POLYGON ((-72.39529 42.18476, -72.39653 42.183...  860Z200US01056
4  MULTIPOLYGON (((-72.39191 42.08066, -72.39077 ...  860Z200US01057

        NAME_y P1_001N  Unnamed: 3    ZIP
0  ZCTA5 01040   38238         NaN  01040
1  ZCTA5 01050    2467         NaN  01050
2  ZCTA5 01053    2031         NaN  01053
3  ZCTA5 01056   21002         NaN  01056
4  ZCTA5 01057    8152         NaN  01057
```

2. Conduct spatial join

```
# Convert district_shp's CRS into zip_shp_merged's CRS
district_shp = district_shp.to_crs(zip_shp_merged.crs)

# Perform a spatial join between zip_shp_merged and district_shp using sjoin
zip_district_joined = gpd.sjoin(
    zip_shp_merged,
    district_shp,
    how="inner",
    predicate="intersects"
)

# Ensure the data type of population
zip_district_joined["P1_001N"]=zip_district_joined["P1_001N"].astype(int)
```

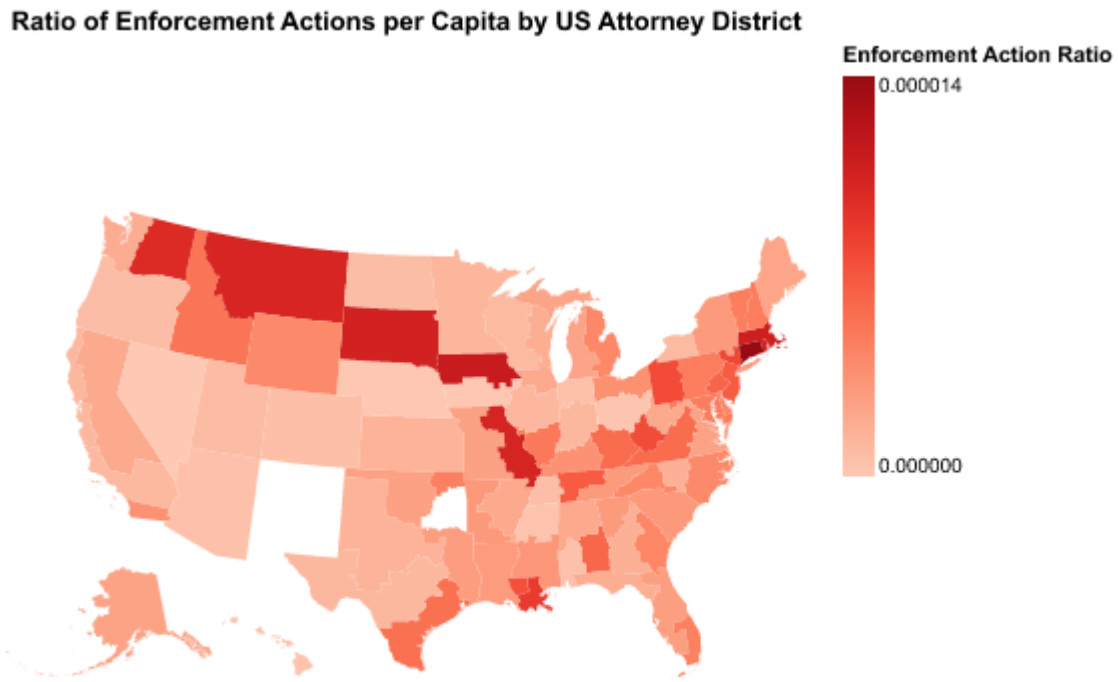
```
# Aggregate population by district
district_population =
 ↪  zip_district_joined.groupby("judicial_d")["P1_001N"].sum().reset_index()

# Rename columns for clarity
district_population.columns = ["District", "Total Population"]

# Display the result
print(district_population.head())
```

```
                        District  Total Population
0  Central District of California          19621862
1    Central District of Illinois           2684528
2              District of Alaska            707199
3             District of Arizona           7334666
4            District of Colorado           5935657
```

3. Map the action ratio in each district

```
# Merge district_counts with district_population
district_enforcement = district_counts.merge(district_population,
 ↪  on="District", how="inner")

# Latest file202101 contains data from January 2021
# Data in district_counts is the aggregated number of enforcements in
 ↪  file202101 by district
# Aggregating the number of enforcement actions since January 2021 per
 ↪  district (has been done in the district_counts)

# Get the enforcement ratio (number of enforcement per dpopulation) of each
 ↪  district
district_enforcement["Ratio"]=district_enforcement["Count"]/district_enforcement["Total
 ↪  Population"]

# Merge distric_shp and district_enforcement
distric_shp_enforcement = district_shp.merge(district_enforcement,
 ↪  left_on="judicial_d", right_on="District", how="inner")

# Create a choropleth map of enforcement action ratios by district
choropleth = alt.Chart(distric_shp_enforcement).mark_geoshape().encode(
```

```
    color=alt.Color("Ratio:Q", title="Enforcement Action Ratio",
↪   scale=alt.Scale(scheme="reds")),
    tooltip=["judicial_d:N", "Ratio:Q", "Count:Q", "Total Population:Q"]
).project(
    type="albersUsa"
).properties(
    title="Ratio of Enforcement Actions per Capita by US Attorney District",
    width=400,
    height=400
)

choropleth
```



The choropleth map illustrates the ratio of enforcement actions per capita for each U.S. Attorney District by dividing the total number of actions since January 2021 by districts population. Some districts in smaller population like Montana, South Dakota, and Iowa show higher ratios despite having fewer absolute actions. The New Mexico and Eastern Oklahoma are shaded in

white, this absence could suggest that data for these districts might be incomplete or missing. This discrepancy occured due to lack of "Agency" information in the "Action Details" in New Mexico enforcement actions webpage, resulting in this district appearing as if they had zero enforcement actions per capita. However, for the Eastern District of Oklahoma, the polygon appeared purely white because there is no enforcement actions data in the OIG website.