Problem Set 5

Lara Tamer and Mario Venegas

2024-11-06

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): Lara Tamer, ltamer
    - Partner 2 (name and cnet ID): Mario Venegas, Mvenegas
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: LT MV **___** **___**
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 by Mario this pset: 1 **___** Late coins left after submission: 1 **___**
7. Late coins used by Lara this pset: 1 **___** Late coins left after submission: 1 **___**
8. 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.
9. (Partner 1): push `ps5.qmd` and `ps5.pdf` to your github repo.
10. (Partner 1): submit `ps5.pdf` via Gradescope. Add your partner on Gradescope.
11. (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)

#conducted research to optimally navigate the nests #https://discuss.codecademy.com/t/this-exercise-uses-selectors-that-target-elements-which-are-direct-children-of-parent-elements-can-we-similarly-target-more-deeply-nested-elements/340956/43

#https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_nesting/Using_CSS_nesting

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

url = 'https://oig.hhs.gov/fraud/enforcement/'
response = requests.get(url)
soup = BeautifulSoup(response.content, 'html.parser')

# Extract titles and links from <h2> tags with nested <a> tags
titles = [element.text.strip() for element in
 ↪  soup.select("h2.usa-card__heading a")]
links = ["https://oig.hhs.gov" + element.get("href") for element in
 ↪  soup.select("h2.usa-card__heading a")]

# Extract dates from <span> tags with the specific class
dates = [element.text.strip() for element in soup.find_all("span", class_ =
 ↪  "text-base-dark padding-right-105")]

# Extract categories from <ul> tags with the specified class
categories = [element.text.strip() for element in
 ↪  soup.select("ul.display-inline.add-list-reset li")]
```

```
# Create a tidy DataFrame
data = {
    "Title": titles,
    "Date": dates,
    "Category": categories,
    "Link": links
}
df = pd.DataFrame(data)

# Display the first 5 rows
print(df.head(5))
```

```
                                               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)

```
import re

agencies = []
```

```
for link in df['Link']:
    detail_response = requests.get(link)
    detail_soup = BeautifulSoup(detail_response.content, 'html.parser')

    # Default agency name if no information is found
    agency_name = "Agency Not Found"

    # Find the <span> with text "Agency:"
    agency_span = detail_soup.find("span", class_ = "padding-right-2
↪  text-base", string = "Agency:")

    # If the <span> is found, move to the parent <li> and extract text after
    ↪  "Agency:"
    if agency_span:
        agency_li = agency_span.parent

        # Extract the full text of <li> and remove the "Agency:" label to get
        ↪  the agency name
        full_text = agency_li.get_text(separator = " ", strip = True)
        agency_name = full_text.replace("Agency:", "").strip()

    agencies.append(agency_name)

# Add the collected agency names as a new column in the existing DataFrame
df['Agency'] = agencies

#I asked chatgpt how can i remove the date from the agency output (i.e., in
↪  the case of Boise Nurse Practitioner Sentenced To 48 Months For
↪  Conspiracy To Distribute Controlled Substances) and it recommended I
↪  import re and do the below

def clean_agency_name(agency_name):
    # Use regex to remove any "Month Day, Year" format
    return
    ↪  re.sub(r'\b(?:January|February|March|April|May|June|July|August|September|October|Nov
    ↪  \d{1,2}, \d{4};?\s*', '', agency_name).strip()

df['Agency'] = df['Agency'].apply(clean_agency_name)

print(df.head(5))
```

```
                                                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          U.S. Attorney's Office, District of Idaho
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)

Function: scrape_enforcement_actions(month, year)

- The function should start by checking if the input 'year' is >= 2013. If the year < 2013, print a message saying: "Restrict to year >=2013, since only enforcement actions after 2013 are listed" and stop the function.
- initialize an empty list "all_data" to store the enforcement actions data
- use a while loop to navigate through each page and fetch the data until there are no more pages. Within the loop:

i. Construct the URL for the current page
ii. Seek for the requested information and extract it
iii. Parse the page with BeautifulSoup to locate titles, dates, categories, and links

- For each link, we need to go to the detailed page and scrape the agency name and append the data to "all_data"
- As suggested, we should use "time.sleep(x)" " to wait for x seconds before moving to the next page to avoid server blocks
- End the loop when there are no more pages
- Finally, the function should save the data in "all_data" as a CSV file named "enforcement_actions_year_month.csv"
- b. Create Dynamic Scraper (PARTNER 2)

Our code was taking some time to run, so I gathered the feedback from the TAs and my classmates from Ed and requested chatgpt to adjust my code accordingly. Next, I asked chatgpt to provide me with ideas to accelerate the process as it was still taking time and it recommended ThreadPoolExecutor which hugely helped. I also did extra research on ThreadPoolExecutor to better understand the tool and make sure that we're properly using it #https://stackoverflow.com/questions/51239251/how-does-concurrent-futures-as-completed-work

#https://superfastpython.com/threadpoolexecutor-as-completed/

```
import time
from datetime import datetime
from concurrent.futures import ThreadPoolExecutor, as_completed

def scrape_enforcement_actions(month, year):
    # Validate starting year
    if year < 2013:
        print("Provide a year >= 2013.")
        return None

    base_date = datetime(year, month, 1)
    all_data = []
    page_number = 1
    should_exit = False

    # Start the scraping loop
    while not should_exit:
        url = f"https://oig.hhs.gov/fraud/enforcement/?page={page_number}"
```

```
        print(f"Scraping page: {page_number}", flush = True) #Ed Discussion
        response = requests.get(url)
        soup = BeautifulSoup(response.content, 'html.parser')

        # Find all enforcement actions on the page
        actions = soup.find_all('li', class_ = "usa-card card--list
↪  pep-card--minimal mobile:grid-col-12")
        if not actions:
            print("No more actions found, ending scrape.") #Ed Discussion
            break

        for action in actions:
            # Extract Title and link
            title_tag = action.find('h2', class_ =
↪  'usa-card__heading').find('a')
            title = title_tag.get_text(strip = True)
            link = f"https://oig.hhs.gov{title_tag['href']}"

            # Extract and parse the Date
            date_text = action.find('span', class_ = "text-base-dark
↪  padding-right-105").get_text(strip = True)
            date_obj = datetime.strptime(date_text, "%B %d, %Y")

            # If the date is earlier than the base date, set the flag to exit
             ↪  the loop
            if date_obj < base_date:
                print(f"Exiting early: {date_obj} is before {base_date}",
                 ↪  flush = True) #Ed Discussion
                should_exit = True
                break

            # Extract Category
            category = action.find('ul', class_ = 'display-inline
↪  add-list-reset').get_text(strip = True)
            all_data.append({"Title": title, "Date": date_text, "Category":
↪  category, "Link": link})

        # Print intermediate results for debugging
        print(f"Collected {len(all_data)} actions so far.", flush = True)

        page_number += 1
        time.sleep(1)  # Delay to avoid overloading the server
```

```
    # Create DataFrame from the collected data
    df = pd.DataFrame(all_data)
    print("Scraping complete. Total records:", len(df))

    # Function to fetch agency information for each action
    def fetch_agency(link):
        try:
            detail_response = requests.get(link)
            detail_soup = BeautifulSoup(detail_response.content,
↪   'html.parser')
            agency_tag = detail_soup.find("span", string = "Agency:")
            if agency_tag:
                agency_li = agency_tag.parent
                full_text = agency_li.get_text(separator = " ", strip = True)
                return
                 ↪  re.sub(r'\b(?:January|February|March|April|May|June|July|August|Septembe
                 ↪  \d{1,2}, \d{4};?\s*', '', full_text.replace("Agency:",
                 ↪  "").strip())
        except requests.exceptions.RequestException as e:
            print(f"Error fetching agency for {link}: {e}")
        return 'N/A'

    # Fetch agency data in parallel (I asked chatgpt to optimize my below
     ↪  code)
    with ThreadPoolExecutor(max_workers = 10) as executor:
        futures = {executor.submit(fetch_agency, row['Link']): index for
↪   index, row in df.iterrows()}
        agencies = []
        for future in as_completed(futures):
            agencies.append(future.result())
        df['Agency'] = agencies

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

    # Check if any dates failed to parse
    if df['Date'].isna().any():
        print("Warning: Some dates could not be parsed and are NaT.")

    # Sort by 'Date' column after dropping NaT values
    df_sorted = df.dropna(subset = ['Date']).sort_values(by = 'Date',
↪   ascending = True)
```

```
    # Save DataFrame to CSV
    file_name = f"enforcement_actions_{year}_{month:02}.csv"
    df_sorted.to_csv(file_name, index = False)
    print(f"Data saved to {file_name}")

    # Display the earliest enforcement action with a valid date
    earliest_action = df_sorted.iloc[0]
    print("Earliest action:", df['Date'].min())
    print(f"Title: {earliest_action['Title']}")
    print(f"Category: {earliest_action['Category']}")
    print(f"Agency: {earliest_action['Agency']}")
    print(f"Link: {earliest_action['Link']}")

    return df_sorted

# Run the function for January 2023
df_2023 = scrape_enforcement_actions(1, 2023)
```

```
Scraping page: 1
Collected 20 actions so far.
Scraping page: 2
Collected 40 actions so far.
Scraping page: 3
Collected 60 actions so far.
Scraping page: 4
Collected 80 actions so far.
Scraping page: 5
Collected 100 actions so far.
Scraping page: 6
Collected 120 actions so far.
Scraping page: 7
Collected 140 actions so far.
Scraping page: 8
Collected 160 actions so far.
Scraping page: 9
Collected 180 actions so far.
Scraping page: 10
Collected 200 actions so far.
Scraping page: 11
Collected 220 actions so far.
Scraping page: 12
```

```
Collected 240 actions so far.
Scraping page: 13
Collected 260 actions so far.
Scraping page: 14
Collected 280 actions so far.
Scraping page: 15
Collected 300 actions so far.
Scraping page: 16
Collected 320 actions so far.
Scraping page: 17
Collected 340 actions so far.
Scraping page: 18
Collected 360 actions so far.
Scraping page: 19
Collected 380 actions so far.
Scraping page: 20
Collected 400 actions so far.
Scraping page: 21
Collected 420 actions so far.
Scraping page: 22
Collected 440 actions so far.
Scraping page: 23
Collected 460 actions so far.
Scraping page: 24
Collected 480 actions so far.
Scraping page: 25
Collected 500 actions so far.
Scraping page: 26
Collected 520 actions so far.
Scraping page: 27
Collected 540 actions so far.
Scraping page: 28
Collected 560 actions so far.
Scraping page: 29
Collected 580 actions so far.
Scraping page: 30
Collected 600 actions so far.
Scraping page: 31
Collected 620 actions so far.
Scraping page: 32
Collected 640 actions so far.
Scraping page: 33
Collected 660 actions so far.
```

```
Scraping page: 34
Collected 680 actions so far.
Scraping page: 35
Collected 700 actions so far.
Scraping page: 36
Collected 720 actions so far.
Scraping page: 37
Collected 740 actions so far.
Scraping page: 38
Collected 760 actions so far.
Scraping page: 39
Collected 780 actions so far.
Scraping page: 40
Collected 800 actions so far.
Scraping page: 41
Collected 820 actions so far.
Scraping page: 42
Collected 840 actions so far.
Scraping page: 43
Collected 860 actions so far.
Scraping page: 44
Collected 880 actions so far.
Scraping page: 45
Collected 900 actions so far.
Scraping page: 46
Collected 920 actions so far.
Scraping page: 47
Collected 940 actions so far.
Scraping page: 48
Collected 960 actions so far.
Scraping page: 49
Collected 980 actions so far.
Scraping page: 50
Collected 1000 actions so far.
Scraping page: 51
Collected 1020 actions so far.
Scraping page: 52
Collected 1040 actions so far.
Scraping page: 53
Collected 1060 actions so far.
Scraping page: 54
Collected 1080 actions so far.
Scraping page: 55
```

```
Collected 1100 actions so far.
Scraping page: 56
Collected 1120 actions so far.
Scraping page: 57
Collected 1140 actions so far.
Scraping page: 58
Collected 1160 actions so far.
Scraping page: 59
Collected 1180 actions so far.
Scraping page: 60
Collected 1200 actions so far.
Scraping page: 61
Collected 1220 actions so far.
Scraping page: 62
Collected 1240 actions so far.
Scraping page: 63
Collected 1260 actions so far.
Scraping page: 64
Collected 1280 actions so far.
Scraping page: 65
Collected 1300 actions so far.
Scraping page: 66
Collected 1320 actions so far.
Scraping page: 67
Collected 1340 actions so far.
Scraping page: 68
Collected 1360 actions so far.
Scraping page: 69
Collected 1380 actions so far.
Scraping page: 70
Collected 1400 actions so far.
Scraping page: 71
Collected 1420 actions so far.
Scraping page: 72
Collected 1440 actions so far.
Scraping page: 73
Collected 1460 actions so far.
Scraping page: 74
Collected 1480 actions so far.
Scraping page: 75
Collected 1500 actions so far.
Scraping page: 76
Collected 1520 actions so far.
```

```
Scraping page: 77
Exiting early: 2022-12-22 00:00:00 is before 2023-01-01 00:00:00
Collected 1534 actions so far.
Scraping complete. Total records: 1534
Data saved to enforcement_actions_2023_01.csv
Earliest action: 2023-01-03 00:00:00
Title: Podiatrist Pays $90,000 To Settle False Billing Allegations
Category: Criminal and Civil Actions
Agency: U.S. Attorney's Office, Southern District of Texas
Link:
https://oig.hhs.gov/fraud/enforcement/podiatrist-pays-90000-to-settle-false-billing-allegati
```

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

```
df_2021 = scrape_enforcement_actions(1, 2021)
```

```
Scraping page: 1
Collected 20 actions so far.
Scraping page: 2
Collected 40 actions so far.
Scraping page: 3
Collected 60 actions so far.
Scraping page: 4
Collected 80 actions so far.
Scraping page: 5
Collected 100 actions so far.
Scraping page: 6
Collected 120 actions so far.
Scraping page: 7
Collected 140 actions so far.
Scraping page: 8
Collected 160 actions so far.
Scraping page: 9
Collected 180 actions so far.
Scraping page: 10
Collected 200 actions so far.
Scraping page: 11
Collected 220 actions so far.
Scraping page: 12
Collected 240 actions so far.
Scraping page: 13
Collected 260 actions so far.
Scraping page: 14
```

```
Collected 280 actions so far.
Scraping page: 15
Collected 300 actions so far.
Scraping page: 16
Collected 320 actions so far.
Scraping page: 17
Collected 340 actions so far.
Scraping page: 18
Collected 360 actions so far.
Scraping page: 19
Collected 380 actions so far.
Scraping page: 20
Collected 400 actions so far.
Scraping page: 21
Collected 420 actions so far.
Scraping page: 22
Collected 440 actions so far.
Scraping page: 23
Collected 460 actions so far.
Scraping page: 24
Collected 480 actions so far.
Scraping page: 25
Collected 500 actions so far.
Scraping page: 26
Collected 520 actions so far.
Scraping page: 27
Collected 540 actions so far.
Scraping page: 28
Collected 560 actions so far.
Scraping page: 29
Collected 580 actions so far.
Scraping page: 30
Collected 600 actions so far.
Scraping page: 31
Collected 620 actions so far.
Scraping page: 32
Collected 640 actions so far.
Scraping page: 33
Collected 660 actions so far.
Scraping page: 34
Collected 680 actions so far.
Scraping page: 35
Collected 700 actions so far.
```

```
Scraping page: 36
Collected 720 actions so far.
Scraping page: 37
Collected 740 actions so far.
Scraping page: 38
Collected 760 actions so far.
Scraping page: 39
Collected 780 actions so far.
Scraping page: 40
Collected 800 actions so far.
Scraping page: 41
Collected 820 actions so far.
Scraping page: 42
Collected 840 actions so far.
Scraping page: 43
Collected 860 actions so far.
Scraping page: 44
Collected 880 actions so far.
Scraping page: 45
Collected 900 actions so far.
Scraping page: 46
Collected 920 actions so far.
Scraping page: 47
Collected 940 actions so far.
Scraping page: 48
Collected 960 actions so far.
Scraping page: 49
Collected 980 actions so far.
Scraping page: 50
Collected 1000 actions so far.
Scraping page: 51
Collected 1020 actions so far.
Scraping page: 52
Collected 1040 actions so far.
Scraping page: 53
Collected 1060 actions so far.
Scraping page: 54
Collected 1080 actions so far.
Scraping page: 55
Collected 1100 actions so far.
Scraping page: 56
Collected 1120 actions so far.
Scraping page: 57
```

```
Collected 1140 actions so far.
Scraping page: 58
Collected 1160 actions so far.
Scraping page: 59
Collected 1180 actions so far.
Scraping page: 60
Collected 1200 actions so far.
Scraping page: 61
Collected 1220 actions so far.
Scraping page: 62
Collected 1240 actions so far.
Scraping page: 63
Collected 1260 actions so far.
Scraping page: 64
Collected 1280 actions so far.
Scraping page: 65
Collected 1300 actions so far.
Scraping page: 66
Collected 1320 actions so far.
Scraping page: 67
Collected 1340 actions so far.
Scraping page: 68
Collected 1360 actions so far.
Scraping page: 69
Collected 1380 actions so far.
Scraping page: 70
Collected 1400 actions so far.
Scraping page: 71
Collected 1420 actions so far.
Scraping page: 72
Collected 1440 actions so far.
Scraping page: 73
Collected 1460 actions so far.
Scraping page: 74
Collected 1480 actions so far.
Scraping page: 75
Collected 1500 actions so far.
Scraping page: 76
Collected 1520 actions so far.
Scraping page: 77
Collected 1540 actions so far.
Scraping page: 78
Collected 1560 actions so far.
```

```
Scraping page: 79
Collected 1580 actions so far.
Scraping page: 80
Collected 1600 actions so far.
Scraping page: 81
Collected 1620 actions so far.
Scraping page: 82
Collected 1640 actions so far.
Scraping page: 83
Collected 1660 actions so far.
Scraping page: 84
Collected 1680 actions so far.
Scraping page: 85
Collected 1700 actions so far.
Scraping page: 86
Collected 1720 actions so far.
Scraping page: 87
Collected 1740 actions so far.
Scraping page: 88
Collected 1760 actions so far.
Scraping page: 89
Collected 1780 actions so far.
Scraping page: 90
Collected 1800 actions so far.
Scraping page: 91
Collected 1820 actions so far.
Scraping page: 92
Collected 1840 actions so far.
Scraping page: 93
Collected 1860 actions so far.
Scraping page: 94
Collected 1880 actions so far.
Scraping page: 95
Collected 1900 actions so far.
Scraping page: 96
Collected 1920 actions so far.
Scraping page: 97
Collected 1940 actions so far.
Scraping page: 98
Collected 1960 actions so far.
Scraping page: 99
Collected 1980 actions so far.
Scraping page: 100
```

```
Collected 2000 actions so far.
Scraping page: 101
Collected 2020 actions so far.
Scraping page: 102
Collected 2040 actions so far.
Scraping page: 103
Collected 2060 actions so far.
Scraping page: 104
Collected 2080 actions so far.
Scraping page: 105
Collected 2100 actions so far.
Scraping page: 106
Collected 2120 actions so far.
Scraping page: 107
Collected 2140 actions so far.
Scraping page: 108
Collected 2160 actions so far.
Scraping page: 109
Collected 2180 actions so far.
Scraping page: 110
Collected 2200 actions so far.
Scraping page: 111
Collected 2220 actions so far.
Scraping page: 112
Collected 2240 actions so far.
Scraping page: 113
Collected 2260 actions so far.
Scraping page: 114
Collected 2280 actions so far.
Scraping page: 115
Collected 2300 actions so far.
Scraping page: 116
Collected 2320 actions so far.
Scraping page: 117
Collected 2340 actions so far.
Scraping page: 118
Collected 2360 actions so far.
Scraping page: 119
Collected 2380 actions so far.
Scraping page: 120
Collected 2400 actions so far.
Scraping page: 121
Collected 2420 actions so far.
```

```
Scraping page: 122
Collected 2440 actions so far.
Scraping page: 123
Collected 2460 actions so far.
Scraping page: 124
Collected 2480 actions so far.
Scraping page: 125
Collected 2500 actions so far.
Scraping page: 126
Collected 2520 actions so far.
Scraping page: 127
Collected 2540 actions so far.
Scraping page: 128
Collected 2560 actions so far.
Scraping page: 129
Collected 2580 actions so far.
Scraping page: 130
Collected 2600 actions so far.
Scraping page: 131
Collected 2620 actions so far.
Scraping page: 132
Collected 2640 actions so far.
Scraping page: 133
Collected 2660 actions so far.
Scraping page: 134
Collected 2680 actions so far.
Scraping page: 135
Collected 2700 actions so far.
Scraping page: 136
Collected 2720 actions so far.
Scraping page: 137
Collected 2740 actions so far.
Scraping page: 138
Collected 2760 actions so far.
Scraping page: 139
Collected 2780 actions so far.
Scraping page: 140
Collected 2800 actions so far.
Scraping page: 141
Collected 2820 actions so far.
Scraping page: 142
Collected 2840 actions so far.
Scraping page: 143
```

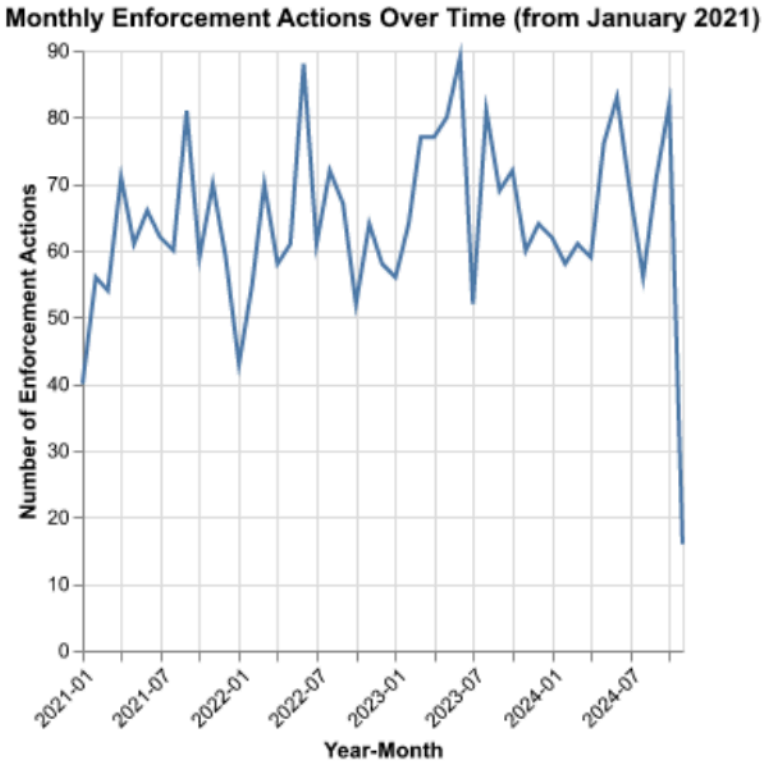
```
Collected 2860 actions so far.
Scraping page: 144
Collected 2880 actions so far.
Scraping page: 145
Collected 2900 actions so far.
Scraping page: 146
Collected 2920 actions so far.
Scraping page: 147
Collected 2940 actions so far.
Scraping page: 148
Collected 2960 actions so far.
Scraping page: 149
Collected 2980 actions so far.
Scraping page: 150
Collected 3000 actions so far.
Scraping page: 151
Collected 3020 actions so far.
Scraping page: 152
Exiting early: 2020-12-30 00:00:00 is before 2021-01-01 00:00:00
Collected 3022 actions so far.
Scraping complete. Total records: 3022
Data saved to enforcement_actions_2021_01.csv
Earliest action: 2021-01-04 00:00:00
Title: 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
Category: Criminal and Civil Actions
Agency: U.S. Attorney's Office, Middle District of Tennessee
Link:
https://oig.hhs.gov/fraud/enforcement/the-united-states-and-tennessee-resolve-claims-with-th
```

Step 3: Plot data based on scraped data

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

```
import altair as alt

df_3_1 = df_2021.copy()

df_3_1['Date'] = pd.to_datetime(df_3_1['Date'], errors = 'coerce')
df_3_1['YearMonth'] = df_3_1['Date'].dt.to_period('M')
```

```
monthly_actions_3_1 = df_3_1.groupby('YearMonth').size().reset_index(name =
 ↪  'ActionCount')
monthly_actions_3_1['YearMonth'] =
 ↪  monthly_actions_3_1['YearMonth'].dt.to_timestamp()

chart_3_1 = alt.Chart(monthly_actions_3_1).mark_line().encode(
    x = alt.X('YearMonth:T', title = 'Year-Month', axis = alt.Axis(format =
 ↪  '%Y-%m', labelAngle = -45)),
    y = alt.Y('ActionCount:Q', title = 'Number of Enforcement Actions')
).properties(
    title = 'Monthly Enforcement Actions Over Time (from January 2021)'
)
chart_3_1.display()
```

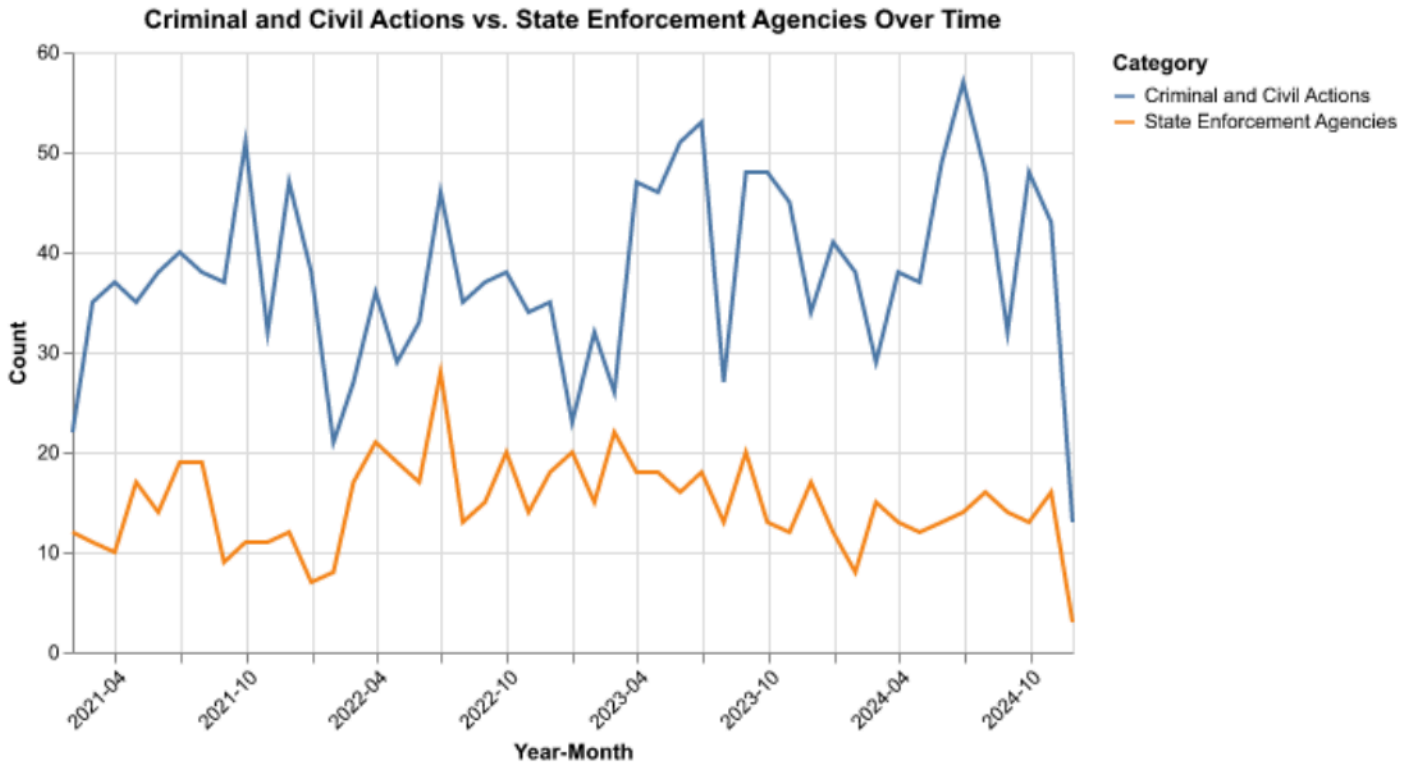


2. Plot the number of enforcement actions categorized: (PARTNER 1)

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

```
import altair as alt

df_3_2_1 = df_2021.copy()

df_3_2_1['Date'] = pd.to_datetime(df_3_2_1['Date'])

# Standardize the Category values using partial matching
def standardize_category(category):
    if pd.isna(category):
        return 'Other'
    category = str(category)
    if 'Criminal and Civil Actions' in category:
        return 'Criminal and Civil Actions'
    elif 'State Enforcement Agencies' in category:
        return 'State Enforcement Agencies'
    else:
        return 'Other'

# Apply the function to create consistent Category values
df_3_2_1['Category'] = df_3_2_1['Category'].apply(standardize_category)

# Set Date as the index for resampling
df_3_2_1.set_index('Date', inplace = True)

# Resample data monthly and count occurrences for each category
monthly_counts =
 ↪  df_3_2_1.groupby('Category').resample('M').size().unstack(level = 0,
 ↪  fill_value = 0).reset_index()

# Convert to long format for Altair
monthly_counts_long = monthly_counts.melt(id_vars = 'Date', value_vars =
 ↪  ['Criminal and Civil Actions', 'State Enforcement Agencies'],
                                          var_name = 'Category', value_name =
 ↪  'Count')

# Create Altair line chart with Year-Month on the x-axis and rotated labels
chart = alt.Chart(monthly_counts_long).mark_line().encode(
    x = alt.X('Date:T', title = 'Year-Month', axis = alt.Axis(format =
 ↪  '%Y-%m', labelAngle = -45)),
    y = 'Count:Q',
    color = 'Category:N'
).properties(
```

```
    title = 'Criminal and Civil Actions vs. State Enforcement Agencies Over
↪   Time',
    width = 500,
    height = 300
)

chart
```

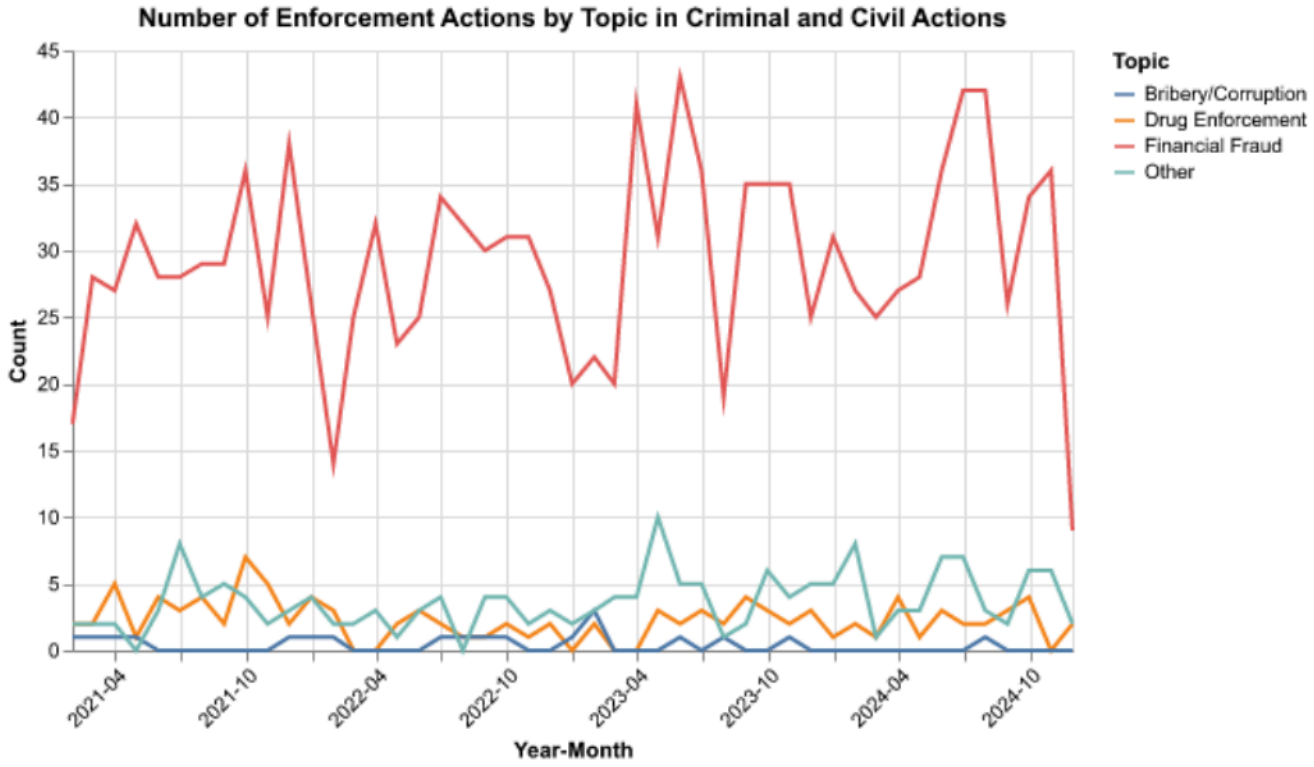


- based on five topics

```
df_3_2_2 = df_2021.copy()

# Ensure the Date column is in datetime format
df_3_2_2['Date'] = pd.to_datetime(df_3_2_2['Date'])

# Define keyword lists for each topic
keywords = {
    'Financial Fraud': ["claims", "pay", "fraud", "settlement", "scheme",
    ↪  "embezzlement", "billing", "kickbacks"],
```

```
    'Drug Enforcement': ["trafficking", "prescribing", "medication",
    ↪  "opioids", "drug", "substance", "distribution", "narcotics"],
    'Bribery/Corruption': ["bribery", "corruption", "kickbacks", "scheme",
    ↪  "conspiracy", "fraud"],
    'Other': ["compliance", "non-fraudulent", "policy", "regulatory"]
}

# Function to categorize each enforcement action by topic based on keywords
↪  in Title
def assign_topic(title):
    title = str(title).lower()
    for topic, words in keywords.items():
        if any(word in title for word in words):
            return topic
    return "Other"

# Apply the function to create a new column 'Topic' for each enforcement
↪  action
df_3_2_2['Topic'] = df_3_2_2['Title'].apply(assign_topic)

# Filter only "Criminal and Civil Actions" category
df_3_2_2 = df_3_2_2[df_3_2_2['Category'] == 'Criminal and Civil Actions']

# Resample data monthly and count occurrences for each topic
df_3_2_2.set_index('Date', inplace = True)
monthly_topic_counts =
↪  df_3_2_2.groupby('Topic').resample('M').size().unstack(level = 0,
↪  fill_value = 0).reset_index()

# Convert to long format for Altair
monthly_topic_counts_long = monthly_topic_counts.melt(id_vars = 'Date',
                                                      value_vars =
↪  ['Financial Fraud', 'Drug Enforcement', 'Bribery/Corruption', 'Other'],
                                                      var_name = 'Topic',
↪  value_name = 'Count')

# Create Altair line chart with Year-Month on the x-axis and rotated labels
chart = alt.Chart(monthly_topic_counts_long).mark_line().encode(
    x = alt.X('Date:T', title = 'Year-Month', axis = alt.Axis(format =
↪  '%Y-%m', labelAngle = -45)),
    y = 'Count:Q',
    color = 'Topic:N'
```

```
).properties(
    title = 'Number of Enforcement Actions by Topic in Criminal and Civil
 ↪  Actions',
    width = 500,
    height = 300
)

chart
```



Step 4: Create maps of enforcement activity

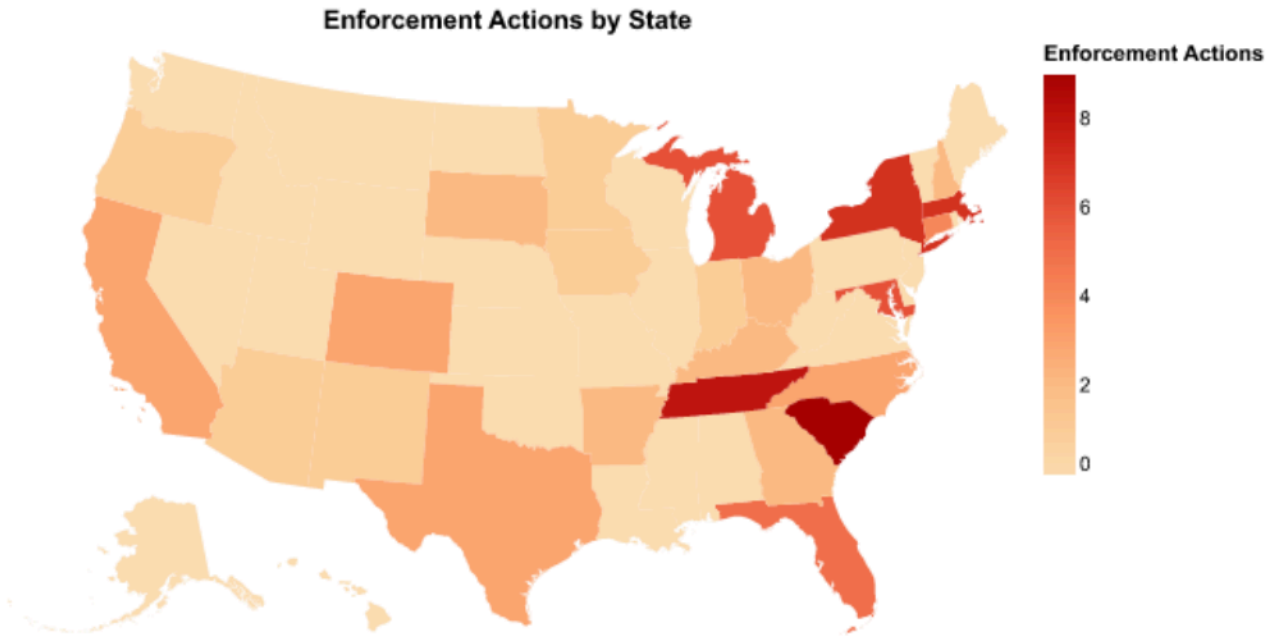
1. Map by State (PARTNER 1)

Research done for the below: - Converting geodataframe to geoJson (also referred to chatgpt for this one) #https://stackoverflow.com/questions/69484892/problem-plotting-geodataframe-with-altair

- Best map projection for the US #https://www.mapbox.com/election-tiles/albers-usa-projection-style#:~:text=Albers%20USA%20is%20typically%20the,the%20rest%20of%20the%20states.

```
import geopandas as gpd
import json

df_4_1 = df_2021.copy()

#Filter for state-level agencies and clean state names
state_actions_df = df_4_1[df_4_1['Agency'].str.contains("State of", na =
 ↪  False)].copy()
state_actions_df['State'] = state_actions_df['Agency'].str.replace("State of
 ↪  ", "").str.strip()

# Count the number of enforcement actions by state
state_action_counts = state_actions_df['State'].value_counts().reset_index()
state_action_counts.columns = ['State', 'ActionCount']

#Load Census state shapefile
state_filepath =
 ↪  '/Users/laratamer/Downloads/cb_2018_us_state_500k/cb_2018_us_state_500k.shp'
states_gdf = gpd.read_file(state_filepath)

# Merge the state count data with the shapefile GeoDataFrame
states_gdf = states_gdf.rename(columns = {'NAME': 'State'})
merged_gdf = states_gdf.merge(state_action_counts, on = 'State', how =
 ↪  'left')
merged_gdf['ActionCount'] = merged_gdf['ActionCount'].fillna(0)

# Convert the merged GeoDataFrame to GeoJSON for Altair
merged_gdf = merged_gdf.to_crs("EPSG:4326")  # Convert to WGS84 for web maps
 ↪  (research)
geojson_data = json.loads(merged_gdf.to_json())

#Create Altair choropleth map
chart2 =
 ↪  alt.Chart(alt.Data(values=geojson_data['features'])).mark_geoshape().encode(
    color = alt.Color('properties.ActionCount:Q', scale = alt.Scale(scheme =
 ↪  'orangered'), title = 'Enforcement Actions'),
    tooltip = ['properties.State:N', 'properties.ActionCount:Q']
).properties(
    width = 500,
    height = 300,
    title = 'Enforcement Actions by State'
).project(
```

```
    type = 'albersUsa'  # Suitable projection for U.S. maps (research)
)

chart2.display()
```



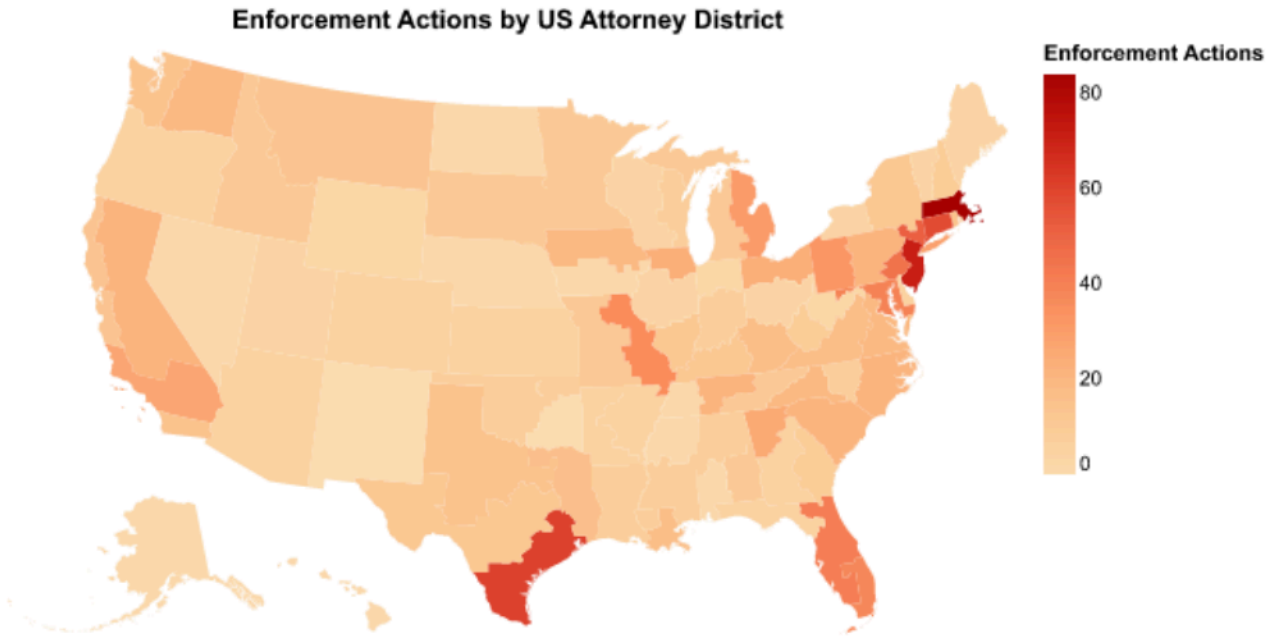
Please note that you can simply see the states along with the detailed count by moving the cursor on the maps in the qmd file

2. Map by District (PARTNER 2)

```
df_4_2 = df_2021.copy()

district_geojson_path = '/Users/laratamer/Downloads/US Attorney Districts
 ↪  Shapefile simplified_20241110.geojson'
districts_gdf = gpd.read_file(district_geojson_path)

# Step 1: Filter for district-level agencies and extract the district name
 ↪  with descriptors
# Capture optional descriptors like "Western, Central, Middle, Norhtern,
 ↪  Eastern" "District" with everything that follows (I asked chatgpt how i
 ↪  can capture anything from District onwards and any word before in case it
 ↪  contains Western, Central, Middle, Norhtern, Eastern)
district_df = df_4_2[df_4_2['Agency'].str.contains("District", na =
 ↪  False)].copy()
```

```
district_df['District'] =
 ↪  district_df['Agency'].str.extract(r'((?:Middle|Central|Southern|Northern|Eastern|Western

# Duplicate entries for each district mentioned in the "District" column
# Split the 'District' column into separate entries where multiple districts
 ↪  are listed (like for instance U.S. Attorney's Office, Western District of
 ↪  Kentucky and U.S. Attorney's Office, Southern District of Florida)
expanded_districts = district_df['District'].str.split(r'\s+and\s+|,\s*U\.S\.
 ↪  Attorney\'s Office', expand = True).stack().reset_index(level = 1, drop =
 ↪  True)

# Join the expanded districts back to the main DataFrame, creating a separate
 ↪  row for each district
district_df = district_df.drop(columns =
 ↪  ['District']).join(expanded_districts.rename('District')).reset_index(drop
 ↪  = True)

# I still had U.S. Attorney in some cells so I applied Step 1 again to remove
 ↪  any remaining "U.S. Attorney's Office" text from the "District" column if
 ↪  it still exists
district_df['District'] =
 ↪  district_df['District'].str.extract(r'((?:Middle|Central|Southern|Northern|Eastern|Weste

#Count the number of enforcement actions by district
district_action_counts = district_df['District'].value_counts().reset_index()
district_action_counts.columns = ['District', 'ActionCount']

# sstandardize the district column in the GeoDataFrame and merge with action
 ↪  counts
districts_gdf = districts_gdf.rename(columns = {'judicial_district':
 ↪  'District'})
merged_gdf = districts_gdf.merge(district_action_counts, on = 'District', how
 ↪  = 'left')
merged_gdf['ActionCount'] = merged_gdf['ActionCount'].fillna(0)  # Set NaN to
 ↪  0 for districts with no actions

# Convert the merged GeoDataFrame to GeoJSON format for Altair visualization
geojson_data = json.loads(merged_gdf.to_json())

#create an Altair choropleth map
chart4 = alt.Chart(alt.Data(values =
 ↪  geojson_data['features'])).mark_geoshape().encode(
```

```
    color = alt.Color('properties.ActionCount:Q', scale = alt.Scale(scheme =
↪   'orangered'), title = 'Enforcement Actions'),
    tooltip = ['properties.District:N', 'properties.ActionCount:Q']
).properties(
    width = 500,
    height = 300,
    title = 'Enforcement Actions by US Attorney District'
).project(
    type = 'albersUsa'
)

chart4.display()
```



Please note that you can simply see the districts along with the detailed count by moving the cursor on the maps in the qmd file