

30538 Problem Set 4: Web Scraping

Manav Mutneja

2026-02-05

Due 02/07 at 5:00PM Central.

“This submission is my work alone and complies with the 30538 integrity policy.” Add your initials to indicate your agreement: **MM**

Github Classroom Assignment Setup and Submission Instructions

1. Accepting and Setting up the PS4 Assignment Repository

- Each student must individually accept the repository for the problem set from Github Classroom (“ps4”) – <https://classroom.github.com/a/hWhcHqH>
 - You will be prompted to select your cnetid from the list in order to link your Github account to your cnetid.
 - If you can’t find your cnetid in the link above, click “continue to next step” and accept the assignment, then add your name, cnetid, and Github account to this Google Sheet and we will manually link it: <https://rb.gy/9u7fb6>
- If you authenticated and linked your Github account to your device, you should be able to clone your PS4 assignment repository locally.
- Contents of PS4 assignment repository:
 - `ps4_template.qmd`: this is the Quarto file with the template for the problem set. You will write your answers to the problem set here.

2. Submission Process:

- Knit your completed solution `ps4.qmd` as a pdf `ps4.pdf`.
 - Your submission does not need runnable code. Instead, you will tell us either what code you ran or what output you got.
- To submit, push `ps4.qmd` and `ps4.pdf` to your PS4 assignment repository. Confirm on Github.com that your work was successfully pushed.

Grading

- You will be graded on what was last pushed to your PS4 assignment repository before the assignment deadline
- Problem sets will be graded for completion as: {missing (0%); - (incomplete, 50%); + (excellent, 100%)}
 - The percent values assigned to each problem denote how long we estimate the problem will take as a share of total time spent on the problem set, not the points they are associated with.
- In order for your submission to be considered complete, you need to push both your `ps4.qmd` and `ps4.pdf` to your repository. Submissions that do not include both files will automatically receive 50% credit.

```

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

```

import requests
from bs4 import BeautifulSoup
import pandas as pd

# Accessing the webpage with User-Agent headers (to prevent blocking)
url = "https://oig.hhs.gov/fraud/enforcement/"
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
    ↪ AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124
    ↪ Safari/537.36"
}
response = requests.get(url, headers=headers)
soup = BeautifulSoup(response.text, 'lxml')

# Find all entries
entries = soup.find_all('div', class_='usa-card__container')
actions = []

for entry in entries:
    row = {}

    # Title & Link
    try:
        heading = entry.find('h2', class_='usa-card__heading')
        title_tag = heading.find('a')
        row['Title'] = title_tag.text.strip()
        row['Link'] = "https://oig.hhs.gov" + title_tag['href']
    except:
        row['Title'] = "N/A"

```

```

        row['Link'] = "N/A"

# Date
try:
    date_tag = entry.find('span', class_='text-base-dark')
    row['Date'] = date_tag.text.strip()
except:
    row['Date'] = "N/A"

# Categories
try:
    category_list = entry.find('ul', class_='display-inline
↪ add-list-reset')

    if category_list:
        categories = [li.text.strip() for li in
↪ category_list.find_all('li')]
        row['Category'] = ", ".join(categories)
    else:
        row['Category'] = "N/A"
except:
    row['Category'] = "N/A"

actions.append(row)

# Create DataFrame and Test Check
df_step1 = pd.DataFrame(actions)
print(df_step1.head())

```

	Title \	Link	Date \
0	Brooklyn Banker Pleads Guilty to Laundering Pr...	https://oig.hhs.gov/fraud/enforcement/brooklyn...	February 3, 2026
1	Delafield Man Sentenced to 18 Months' Imprison...	https://oig.hhs.gov/fraud/enforcement/delafiel...	February 3, 2026
2	Former NFL Player Convicted for \$197M Medicare...	https://oig.hhs.gov/fraud/enforcement/former-n...	February 3, 2026
3	AG's Office Secures Indictments Against Peabod...	https://oig.hhs.gov/fraud/enforcement/ags-offi...	February 2, 2026
4	Florida Man Pleads Guilty to Conspiracy to Vio...	https://oig.hhs.gov/fraud/enforcement/florida-...	January 30, 2026

	Category
0	COVID-19
1	Criminal and Civil Actions
2	Criminal and Civil Actions
3	State Enforcement Agencies
4	Criminal and Civil Actions

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

1.

Choice of Loop: We will use a while loop.

Reasoning: A standard for loop iterates over a known sequence (e.g., “Pages 1 to 50”). In this web crawling task, we do not know in advance how many pages of data exist back to our target date. If we guessed 50 pages, we might miss data; if we guessed 1000, we might waste resources.

Logic: A while loop allows us to continue scraping indefinitely until a specific “Stop Condition” is met (in this case, finding a date older than our target year/month). This is the standard approach for crawlers where the depth is unknown.

- a. Pseudo-Code

- (1) Define Function `scrape_enforcement_actions(year, month)`: -> Input Validation: IF `year < 2013`: Print warning “Please restrict to year `>= 2013`” and RETURN. -> Setup: -> Define `stop_date` using the input year and month. -> Initialize `all_actions` empty list. -> Initialize `page = 1`. -> Set boolean flag `keep_scraping = True`.
- (2) Start Crawling Loop (while `keep_scraping` is True):
 - a. Request Handling (Slide 10 & 16): -> Construct URL with current page number. -> Crucial: Send GET request with User-Agent headers to identify as a browser (this prevents blocking). -> Parse: convert HTML to soup using `lxml`.
 - b. Page Validation: -> If no enforcement actions are found on the page -> BREAK loop (End of content).
 - c. Extraction Loop (Iterate through items): For each enforcement action on the page: -> Extract Title, Link, Category, and Date. -> Parse Date: Convert text date to a Python Date Object. -> Safety Check (The Exit Condition): -> IF `current/action_date < stop_date`: -> Set `keep_scraping = False`. -> BREAK inner loop immediately. -> ELSE: -> Append data to `all_actions`.

- d. Pagination & Etiquette (Slide 16): -> Increment page by 1. -> Wait: Execute `time.sleep(1)` to respect server load and avoid rate-limiting.
- (3) Output: -> Convert all `_actions` to `DataFrame`. -> Save to “`enforcement_actions_{year}_{month}.csv`”.
- > Return `DataFrame`.
- b. Create Dynamic Scraper

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

def scrape_enforcement_actions(year, month):
    # Constraint 2: Input Validation
    if year < 2013:
        print("Please restrict to year >= 2013, since only enforcement
        ↪ actions after 2013 are listed.")
        return None

    # Set the target stop date
    stop_date = datetime(year, month, 1)

    base_url = "https://oig.hhs.gov/fraud/enforcement/"
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
        ↪ AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124
        ↪ Safari/537.36"
    }

    all_rows = []
    page = 1
    keep_scraping = True

    while keep_scraping:
        url = f"{base_url}?page={page}"
        print(f"Scraping: {url}")

        try:
            # Fetch with headers
            response = requests.get(url, headers=headers)
            soup = BeautifulSoup(response.text, 'lxml')
```

```

entries = soup.find_all('div', class_='usa-card__container')

# If page is empty, we are done
if not entries:
    break

for entry in entries:
    row = {}

    # Title & Link
    try:
        title_tag = entry.find('h2',
↪ class_='usa-card__heading').find('a')
        row['Title'] = title_tag.text.strip()
        row['Link'] = "https://oig.hhs.gov" + title_tag['href']
    except:
        continue

    # Categories
    try:
        category_list = entry.find('ul', class_='display-inline
↪ add-list-reset')
        if category_list:
            categories = [li.text.strip() for li in
↪ category_list.find_all('li')]
            row['Category'] = ", ".join(categories)
        else:
            row['Category'] = "N/A"
    except:
        row['Category'] = "N/A"

    # Date & Stop Condition
    try:
        date_text = entry.find('span',
↪ class_='text-base-dark').text.strip()
        row['Date'] = date_text

        current_date = datetime.strptime(date_text, "%B %d, %Y")

        # STOP CONDITION
        if current_date < stop_date:

```

```

        keep_scraping = False
        break
    except:
        row['Date'] = "N/A"

    all_rows.append(row)

    # Constraint 4: Wait 1 second
    page += 1
    time.sleep(1)

except Exception as e:
    print(f"Error on page {page}: {e}")
    break

# Constraint 3: Save to CSV
df = pd.DataFrame(all_rows)
filename = f"enforcement_actions_{year}_{month}.csv"
df.to_csv(filename, index=False)
print(f"Scraping complete. Saved {len(df)} rows to {filename}")

return df

```

- c. Test Your Code

```

# Constraint 1: Indicator to prevent running during knit
run_scraper = False

if run_scraper:
    # --- PART 2b: January 2024 ---
    print("--- Running for Jan 2024 ---")
    df_2024 = scrape_enforcement_actions(2024, 1)

    # Answers for 2b:
    print(f"Total actions since Jan 2024: {len(df_2024)}")
    if not df_2024.empty:
        print(f"Earliest action details (2024): \n{df_2024.iloc[-1]}")

    # --- PART 2c: January 2022 ---
    print("\n--- Running for Jan 2022 ---")
    df_2022 = scrape_enforcement_actions(2022, 1)

```

```

# Answers for 2c:
print(f"Total actions since Jan 2022: {len(df_2022)}")
if not df_2022.empty:
    print(f"Earliest action details (2022): \n{df_2022.iloc[-1]}")

else:
    try:
        df_2024 = pd.read_csv("enforcement_actions_2024_1.csv")
        df_2022 = pd.read_csv("enforcement_actions_2022_1.csv")
        print("Loaded data from CSVs.")
    except:
        print("CSVs not found. Set run_scraper = True to generate them.")

```

Loaded data from CSVs.

— **Running for Jan 2024** — Scraping complete. Saved 1769 rows to enforcement_actions_2024_1.csv

Total actions since Jan 2024: 1769

Earliest action details (2024): *Title:* Former Nurse Aide Indicted In Death Of Clarksville Resident *Link:* <https://oig.hhs.gov/fraud/enforcement/former-n...> *Category:* State Enforcement Agencies *Date:* January 3, 2024

Name: 1768, dtype: object

— **Running for Jan 2022** — scraping complete. Saved 3359 rows to enforcement_actions_2022_1.csv

Total actions since Jan 2022: 3359

Earliest action details (2022): *Title:* Integrated Pain Management Medical Group Agree... *Link:* <https://oig.hhs.gov/fraud/enforcement/integrat...> *Category:* Fraud Self-Disclosures *Date:* January 4, 2022

Name: 3358, dtype: object

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

```

import altair as alt
import pandas as pd

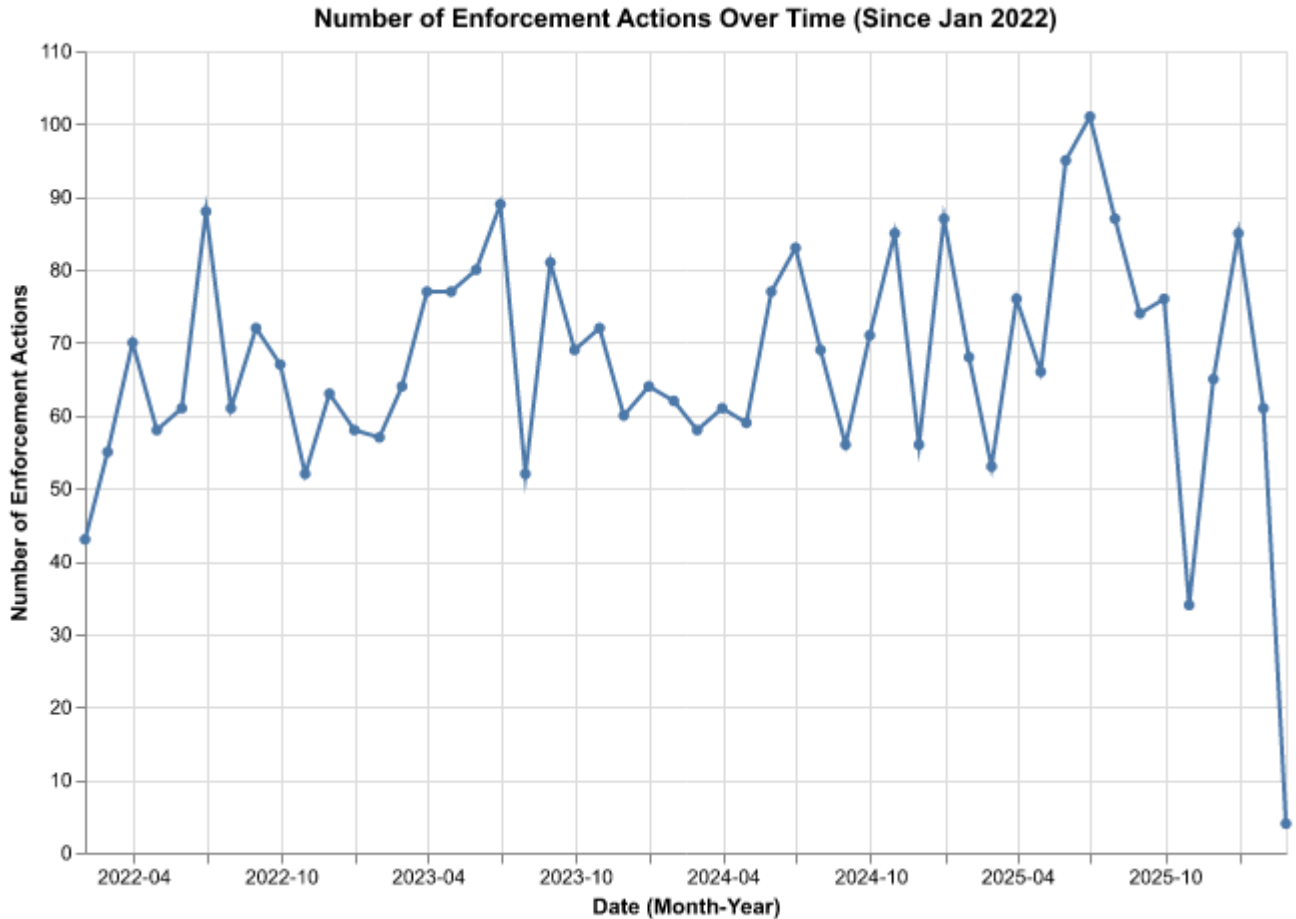
# Load the dataset (using the 2021 file gives us the best view)
df = pd.read_csv("enforcement_actions_2022_1.csv")

# Convert the 'Date' column to real datetime objects
df['Date'] = pd.to_datetime(df['Date'])

# Aggregate: Count actions per Month
monthly_counts = df.groupby(pd.Grouper(key='Date',
↪   freq='ME')).size().reset_index(name='Count')

# Create the Line Chart
chart_3a = alt.Chart(monthly_counts).mark_line(point=True).encode(
    x=alt.X('Date', title='Date (Month-Year)',
    ↪   axis=alt.Axis(format='%Y-%m')),
    y=alt.Y('Count', title='Number of Enforcement Actions'),
    tooltip=['Date', 'Count']
).properties(
    title='Number of Enforcement Actions Over Time (Since Jan 2022)',
    width=600,
    height=400
)
# Display the chart
chart_3a.display()

```



Interpretation of Enforcement Actions Over Time (Since Jan 2022):

High Volatility: The most immediate observation is that enforcement activity is not consistent month-to-month. The chart shows a “sawtooth” pattern with sharp fluctuations, indicating that HHS OIG enforcement actions tend to be reported in batches or waves rather than a steady daily stream.

General Range: Despite the volatility, the volume of actions appears relatively stable over the long term, generally hovering between 50 and 90 actions per month. There is no massive upward or downward structural break in the dataset over these four years.

Significant Peaks: There are distinct spikes where activity exceeded 90 or even 100 actions (notably around mid-2023 and late 2025). This could correspond to the conclusion of major investigation cycles or fiscal year reporting deadlines.

The “Current Month” Drop-off: The sharp plunge at the very end of the chart (far right) is an artifact of incomplete data. Since the data was scraped in early February 2026, the count

for that final month is close to zero simply because the month has just begun, not because enforcement has stopped.

2. Plot the number of enforcement actions categorized:

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

```
import pandas as pd
import altair as alt

# Load the dataset
df = pd.read_csv("enforcement_actions_2022_1.csv")

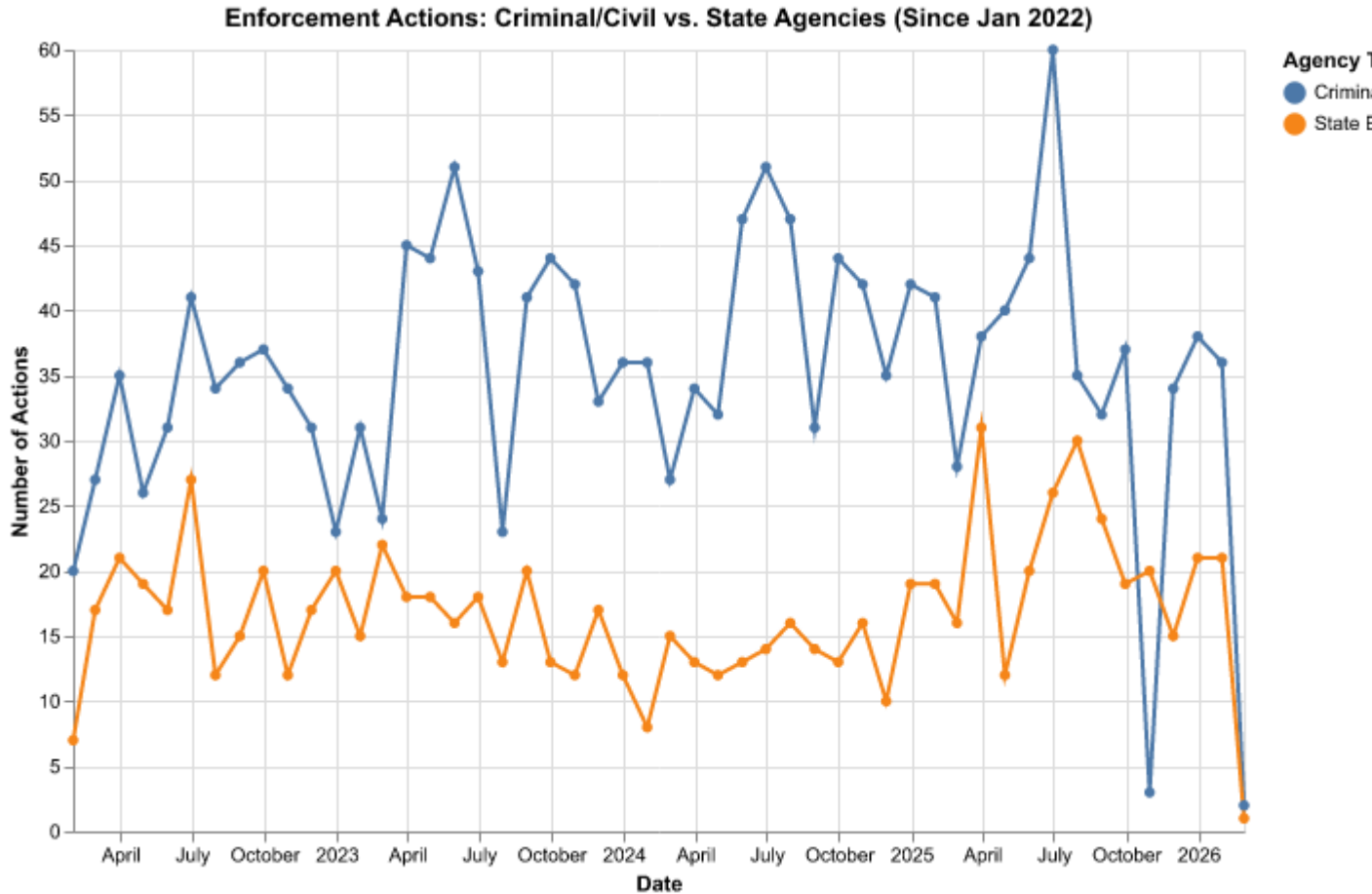
# Convert Date to datetime objects
df['Date'] = pd.to_datetime(df['Date'])

# Filter: Select only the two categories of interest
target_categories = ["Criminal and Civil Actions", "State Enforcement
↪ Agencies"]
df_filtered = df[df['Category'].isin(target_categories)]

# Aggregate: Group by Month AND Category
monthly_cat_counts = df_filtered.groupby([pd.Grouper(key='Date', freq='ME'),
↪ 'Category']).size().reset_index(name='Count')

# Plot: Line chart split by color
chart_3b_i = alt.Chart(monthly_cat_counts).mark_line(point=True).encode(
    x=alt.X('Date', title='Date'),
    y=alt.Y('Count', title='Number of Actions'),
    color=alt.Color('Category', title='Agency Type'),
    tooltip=['Date', 'Category', 'Count']
).properties(
    title='Enforcement Actions: Criminal/Civil vs. State Agencies (Since Jan
↪ 2022)',
    width=600,
    height=400
)

chart_3b_i.display()
```



Analysis: The plot compares the volume of “Criminal and Civil Actions” versus “State Enforcement Agencies.”

Dominant Category: “Criminal and Civil Actions” (blue line) consistently tracks significantly higher than “State Enforcement Agencies” (orange line) throughout the entire period. This indicates that federal-level criminal and civil enforcement is the primary driver of HHS OIG’s public reporting volume.

Volatility: The “Criminal and Civil Actions” line shows high volatility with sharp peaks (e.g., reaching 60 actions in mid-2025), whereas “State Enforcement Agencies” remains relatively flatter, typically fluctuating between 10 and 30 actions per month.

Correlation: There appears to be a loose correlation in reporting cycles; often when one category spikes or dips (such as the dip in early 2024), the other follows a similar, albeit smaller, trend.

- based on five topics

```

# Filter: Start with ONLY "Criminal and Civil Actions"
df_criminal = df[df['Category'] == "Criminal and Civil Actions"].copy()

# Define the classification function
def classify_topic(title):
    title = title.lower() # Convert to lowercase so "Bank" and "bank" both
    ↪ match

    # Keywords for "Health Care Fraud"
    if any(word in title for word in ['medicare', 'medicaid', 'doctor',
    ↪ 'hospital', 'pharmacy', 'nurse', 'medical', 'health', 'provider',
    ↪ 'clinic', 'care']):
        return "Health Care Fraud"

    # Keywords for "Financial Fraud" (Prompt Hint: "bank", "financial")
    elif any(word in title for word in ['bank', 'financial', 'laundering',
    ↪ 'embezzle', 'theft', 'tax', 'fraud', 'wire', 'identity']):
        return "Financial Fraud"

    # Keywords for "Drug Enforcement"
    elif any(word in title for word in ['drug', 'opioid', 'pill', 'prescri',
    ↪ 'narcotic', 'controlled substance', 'fentanyl', 'diversion']):
        return "Drug Enforcement"

    # Keywords for "Bribery/Corruption"
    elif any(word in title for word in ['bribe', 'kickback', 'corrupt',
    ↪ 'payoff', 'extortion']):
        return "Bribery/Corruption"

    # Everything else goes to "Other"
    else:
        return "Other"

# Apply the function to create a new 'Topic' column
df_criminal['Topic'] = df_criminal['Title'].apply(classify_topic)

# Aggregate by Month and Topic
topic_counts = df_criminal.groupby([pd.Grouper(key='Date', freq='ME'),
    ↪ 'Topic']).size().reset_index(name='Count')

# Plot the final line chart
chart_3b_ii = alt.Chart(topic_counts).mark_line(point=True).encode(

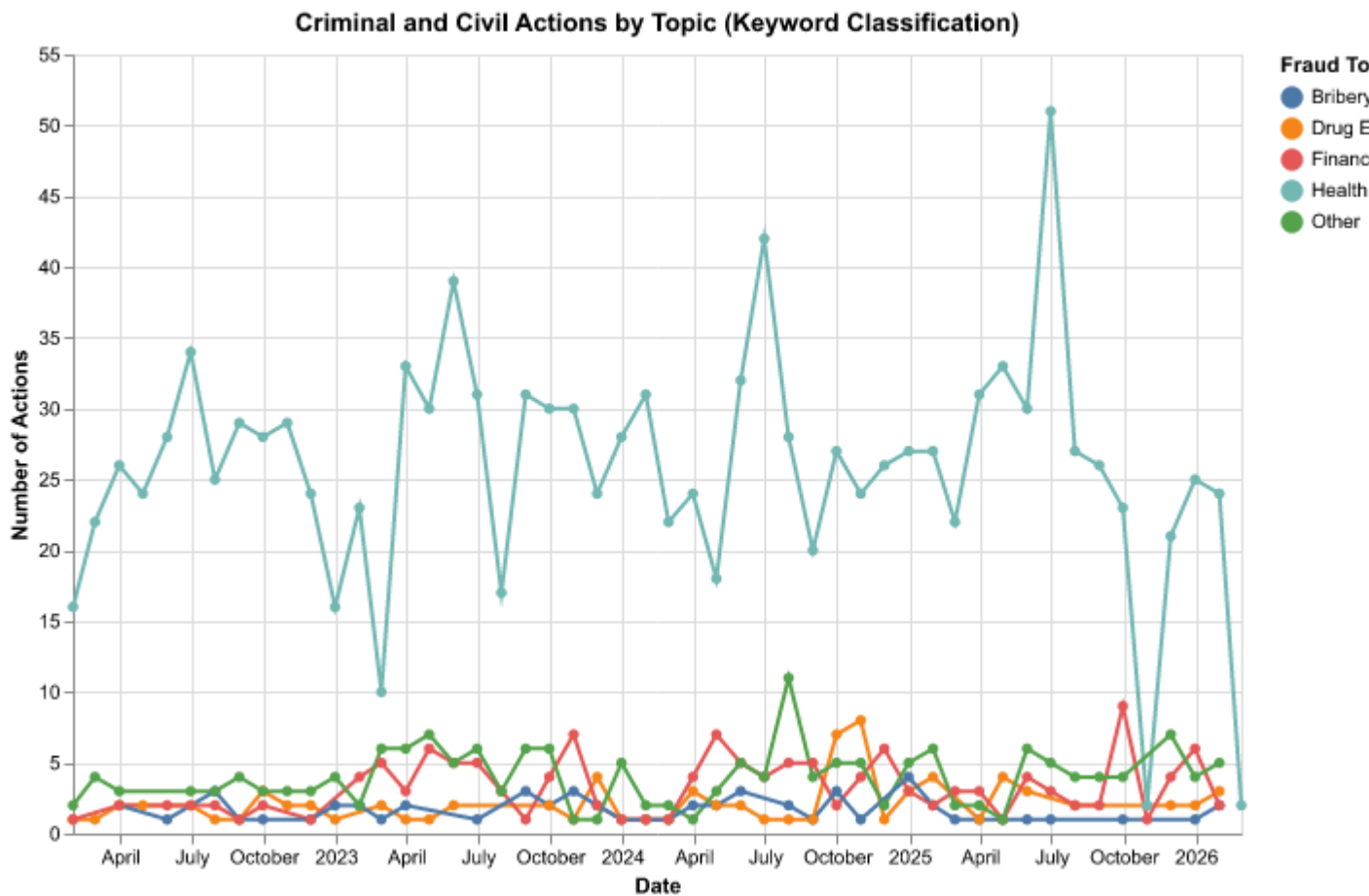
```

```

x=alt.X('Date', title='Date'),
y=alt.Y('Count', title='Number of Actions'),
color=alt.Color('Topic', title='Fraud Topic'),
tooltip=['Date', 'Topic', 'Count']
).properties(
    title='Criminal and Civil Actions by Topic (Keyword Classification)',
    width=600,
    height=400
)

chart_3b_ii.display()

```



Analysis: This plot breaks down the “Criminal and Civil Actions” category into five specific topics based on keyword analysis of the titles.

Health Care Dominance: The “Health Care Fraud” topic (teal line) is overwhelmingly the

dominant category, consistently accounting for the vast majority of enforcement actions (often 20–50 per month). This aligns with the HHS OIG’s primary mission to oversee Medicare and Medicaid programs.

Secondary Topics: “Financial Fraud,” “Drug Enforcement,” and “Bribery/Corruption” represent a very small fraction of the total actions, rarely exceeding 5–8 actions per month. This suggests that while these issues exist, they are niche compared to general billing and provider fraud.

The “*Other*” Category: The “Other” line (green) remains low, which is a positive sign for our classification function. It means that our “Health Care” keywords successfully captured the bulk of the relevant cases, leaving very few unclassified actions.