PS4

Jiamin Zhang

2026-02-06

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: JM Z

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/hWhtcHqH
        - 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 re
import requests
import pandas as pd
from bs4 import BeautifulSoup
from urllib.parse import urljoin

BASE = "https://oig.hhs.gov"
URL = "https://oig.hhs.gov/fraud/enforcement/"  # initial Showing 1-20 ...
 ↪  Sorted by date

# 1) download
resp = requests.get(URL, timeout=30)
resp.raise_for_status()

# 2) parse HTML
soup = BeautifulSoup(resp.text, "html.parser")

# 3) get href
detail_links = []
for a in soup.select('a[href^="/fraud/enforcement/"]'):
    href = a.get("href", "")
    if "?" in href:          # filter /fraud/enforcement/?page=...
        continue
    if href.count("/") < 4:  # filter /fraud/enforcement/
        continue
    title = a.get_text(strip=True)
    if not title:
        continue
    detail_links.append(a)
```

```
# delete
seen = set()
unique_links = []
for a in detail_links:
    href = a.get("href")
    if href not in seen:
        seen.add(href)
        unique_links.append(a)

# 4)get date / category
month_pat =
 ↪  r"(January|February|March|April|May|June|July|August|September|October|November|December)
date_re = re.compile(rf"{month_pat}\s+\d{{1,2}},\s+\d{{4}}")

rows = []
for a in unique_links:
    title = a.get_text(strip=True)
    link = urljoin(BASE, a.get("href"))

    container = a.find_parent(["li", "article", "div"])
    if container is None:
        container = a.parent

    block_text = container.get_text(" ", strip=True)

    # date "Month dd, yyyy"
    m = date_re.search(block_text)
    date = m.group(0) if m else None

    # category
    cats = []
    for c in container.select('a[href*="type="]'):
        cat_text = c.get_text(strip=True)
        if cat_text and cat_text not in cats:
            cats.append(cat_text)

    #  COVID-19 + Criminal and Civil Actions
    category = "; ".join(cats) if cats else None

    rows.append({"title": title, "date": date, "category": category, "link":
 ↪  link})
```

```
df = pd.DataFrame(rows)

# 5) clean
df = df[df["date"].notna()].reset_index(drop=True)

df.head()
```

	title	date	category	link
0	Houston Transplant Doctor Indicted For Making ...	February 5, 2026	None	https://oig.hhs.gov/f
1	MultiCare Health System to Pay Millions to Set...	February 4, 2026	None	https://oig.hhs.gov/f
2	Brooklyn Banker Pleads Guilty to Laundering Pr...	February 3, 2026	None	https://oig.hhs.gov/f
3	Delafield Man Sentenced to 18 Months' Imprison...	February 3, 2026	None	https://oig.hhs.gov/f
4	Former NFL Player Convicted for $197M Medicare...	February 3, 2026	None	https://oig.hhs.gov/f

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code Before implementing the scraper as a function, I outline the main steps of the procedure in pseudo-code.

1. Take the input month and year provided by the user.
2. First check whether the input year is greater than or equal to 2013.
    - If the year is less than 2013, print a reminder message and stop the function, since enforcement actions prior to 2013 are not available on the website.
3. Initialize an empty list to store all enforcement action records.
4. Set the page counter to 0, corresponding to the first page of the enforcement actions website.
5. Enter a while loop to crawl through multiple pages:
    - Construct the URL for the current page using the page counter.
    - Send a request to the webpage and parse the HTML content.
    - Extract the title, date, category, and link for each enforcement action on the page.
    - Append the extracted information to the list of records.
    - Identify the oldest date on the current page.
    - If the oldest date on the page is earlier than the user-specified starting month and year, exit the loop.
    - Otherwise, increment the page counter by one and wait for one second before moving to the next page.

6. After exiting the loop, convert the collected records into a tidy dataframe.
7. Filter the dataframe to keep only enforcement actions that occur on or after the specified starting month and year.
8. Save the resulting dataframe as a CSV file named `enforcement_actions_year_month.csv`.
9. Return the dataframe for further analysis.

A while loop is used instead of a simple for loop because the total number of pages to scrape is unknown in advance. The stopping condition depends on the dates of the enforcement actions rather than a fixed number of pages.

```
import re
import time
import requests
import pandas as pd
from bs4 import BeautifulSoup
from urllib.parse import urljoin
from datetime import datetime

BASE = "https://oig.hhs.gov"
BASE_URL = "https://oig.hhs.gov/fraud/enforcement/"

# Regular expression to identify dates such as "February 3, 2026"
month_pat =
 ↪  r"(January|February|March|April|May|June|July|August|September|October|November|December)
date_re = re.compile(rf"{month_pat}\s+\d{{1,2}},\s+\d{{4}}")

# Common enforcement action categories
KNOWN_CATS = [
    "Criminal and Civil Actions",
    "Audits",
    "Evaluations",
    "Corporate Integrity Agreements",
    "Compliance",
    "Exclusions",
    "Advisory Opinions",
    "Self-Disclosure",
    "Special Fraud Alerts",
    "Fraud Alerts",
    "COVID-19",
]

def parse_date(date_str):
    """Convert a date string (e.g., 'February 3, 2026') to a datetime
     ↪  object."""
```

```
    return datetime.strptime(date_str, "%B %d, %Y")

def scrape_one_page(page_num):
    """
    Scrape one enforcement actions page.
    Returns a list of dictionaries containing title, date, category, and
 ↪  link.
    """
    url = BASE_URL if page_num == 0 else f"{BASE_URL}?page={page_num}"
    response = requests.get(url, timeout=30)
    response.raise_for_status()
    soup = BeautifulSoup(response.text, "html.parser")

    # Find all candidate links to enforcement action detail pages
    links = []
    for a in soup.select('a[href^="/fraud/enforcement/"]'):
        href = a.get("href", "")
        if "?" in href:          # Exclude pagination links
            continue
        if href.count("/") < 4:  # Exclude directory-level links
            continue
        title = a.get_text(strip=True)
        if not title:
            continue
        links.append(a)

    # Remove duplicate links
    seen = set()
    unique_links = []
    for a in links:
        href = a.get("href")
        if href not in seen:
            seen.add(href)
            unique_links.append(a)

    rows = []
    for a in unique_links:
        title = a.get_text(strip=True)
        link = urljoin(BASE, a.get("href"))

        # Identify the container holding date and category information
        container = a.find_parent(["li", "article", "div"])
        if container is None:
```

```
            container = a.parent

        block_text = container.get_text(" ", strip=True)

        # Extract date
        match = date_re.search(block_text)
        if not match:
            continue
        date = match.group(0)

        # Identify category by keyword matching
        category = None
        for cat in KNOWN_CATS:
            if cat in block_text:
                category = cat
                break

        rows.append({
            "title": title,
            "date": date,
            "category": category,
            "link": link
        })

    return rows

def scrape_enforcement_actions(start_month, start_year, run_scraper=False):
    """
    Dynamic scraper for HHS OIG enforcement actions.

    Parameters:
    - start_month: starting month (int)
    - start_year: starting year (int)
    - run_scraper: boolean indicator to control whether the scraper runs

    The function saves the results to a CSV file and returns a dataframe.
    """
    if start_year < 2013:
        print("Reminder: Please restrict to year >= 2013 because only
         ↪  enforcement actions after 2013 are listed.")
        return None

    filename = f"enforcement_actions_{start_year}_{start_month:02d}.csv"
```

```
    # If the indicator is off, attempt to load an existing CSV file
    if not run_scraper:
        try:
            df = pd.read_csv(filename)
            print(f"Loaded existing file: {filename}")
            return df
        except FileNotFoundError:
            print(f"{filename} not found. Set run_scraper=True once to
             ↪  generate it.")
            return None

    # If the indicator is on, start scraping
    start_date = datetime(start_year, start_month, 1)
    all_rows = []
    page = 0

    while True:
        page_rows = scrape_one_page(page)

        # Stop if no results are returned
        if len(page_rows) == 0:
            break

        all_rows.extend(page_rows)

        # Determine the oldest date on the current page
        page_dates = [parse_date(r["date"]) for r in page_rows]
        oldest_on_page = min(page_dates)

        # Stop if the scraper has gone past the target start date
        if oldest_on_page < start_date:
            break

        page += 1
        time.sleep(1)  # Wait one second before moving to the next page

    df = pd.DataFrame(all_rows)

    # Filter results to keep only dates on or after the specified start date
    df["date_dt"] = df["date"].apply(parse_date)
    df = df[df["date_dt"] >=
↪   start_date].drop(columns=["date_dt"]).reset_index(drop=True)
```

```
    # Save results to CSV
    df.to_csv(filename, index=False)
    print(f"Saved: {filename} with {len(df)} rows")

    return df
```

- b. Create Dynamic Scraper

```
df_2024 = scrape_enforcement_actions(
    start_month=1,
    start_year=2024,
    run_scraper=True
)

df_2024.head()
```

```
Saved: enforcement_actions_2024_01.csv with 1807 rows
```

	title	date	category	lin
0	Houston Transplant Doctor Indicted For Making ...	February 5, 2026	Criminal and Civil Actions	htt
1	MultiCare Health System to Pay Millions to Set...	February 4, 2026	Criminal and Civil Actions	htt
2	Brooklyn Banker Pleads Guilty to Laundering Pr...	February 3, 2026	COVID-19	htt
3	Delafield Man Sentenced to 18 Months' Imprison...	February 3, 2026	Criminal and Civil Actions	htt
4	Former NFL Player Convicted for $197M Medicare...	February 3, 2026	Criminal and Civil Actions	htt

```
df_2024.shape[0]
```

```
1807
```

```
# earliest enforcement action (by date)
df_2024_sorted = df_2024.copy()
df_2024_sorted["date_dt"] = pd.to_datetime(df_2024_sorted["date"])
earliest_row = df_2024_sorted.sort_values("date_dt").head(1)
earliest_row
```

	title	date	category	link
1806	Former Nurse Aide Indicted In Death Of Clarksv...	January 3, 2024	None	https://oig.hhs.gov/

Using the dynamic scraper, I collected all enforcement actions starting from January 2024. The final dataframe contains **1,787 enforcement actions**. The earliest enforcement action scraped occurred on **January 3, 2024**.
The title of this enforcement action is **"Former Nurse Aide Indicted in Death of Clarksville..."**.
The corresponding link to the enforcement action is included in the dataframe.

- c. Test Your Code

```
# Run ONCE to generate the CSV file (this may take a while)
df_2022 = scrape_enforcement_actions(
    start_month=1,
    start_year=2022,
    run_scraper=True
)

df_2022.head()
```

```
Saved: enforcement_actions_2022_01.csv with 3397 rows
```

	title	date	category	lin
0	Houston Transplant Doctor Indicted For Making ...	February 5, 2026	Criminal and Civil Actions	htt
1	MultiCare Health System to Pay Millions to Set...	February 4, 2026	Criminal and Civil Actions	htt
2	Brooklyn Banker Pleads Guilty to Laundering Pr...	February 3, 2026	COVID-19	htt
3	Delafield Man Sentenced to 18 Months' Imprison...	February 3, 2026	Criminal and Civil Actions	htt
4	Former NFL Player Convicted for $197M Medicare...	February 3, 2026	Criminal and Civil Actions	htt

```
# Total number of enforcement actions since Jan 2022
df_2022.shape[0]
```

```
3397
```

```
# Find the earliest enforcement action (by date)
df_2022_sorted = df_2022.copy()
df_2022_sorted["date_dt"] = pd.to_datetime(df_2022_sorted["date"])
earliest_2022_row = df_2022_sorted.sort_values("date_dt").head(1)
earliest_2022_row
```

	title	date	category	link
3396	Integrated Pain Management Medical Group Agree...	January 4, 2022	Self-Disclosure	https://oig.h

Using the dynamic scraper, I collected all enforcement actions starting from January 2022.

The final dataframe contains **3,377 enforcement actions**.

The earliest enforcement action scraped occurred on **January 4, 2022**.
The title of this enforcement action is **"Integrated Pain Management Medical Group Agree..."**, and the corresponding link is included in the dataframe.

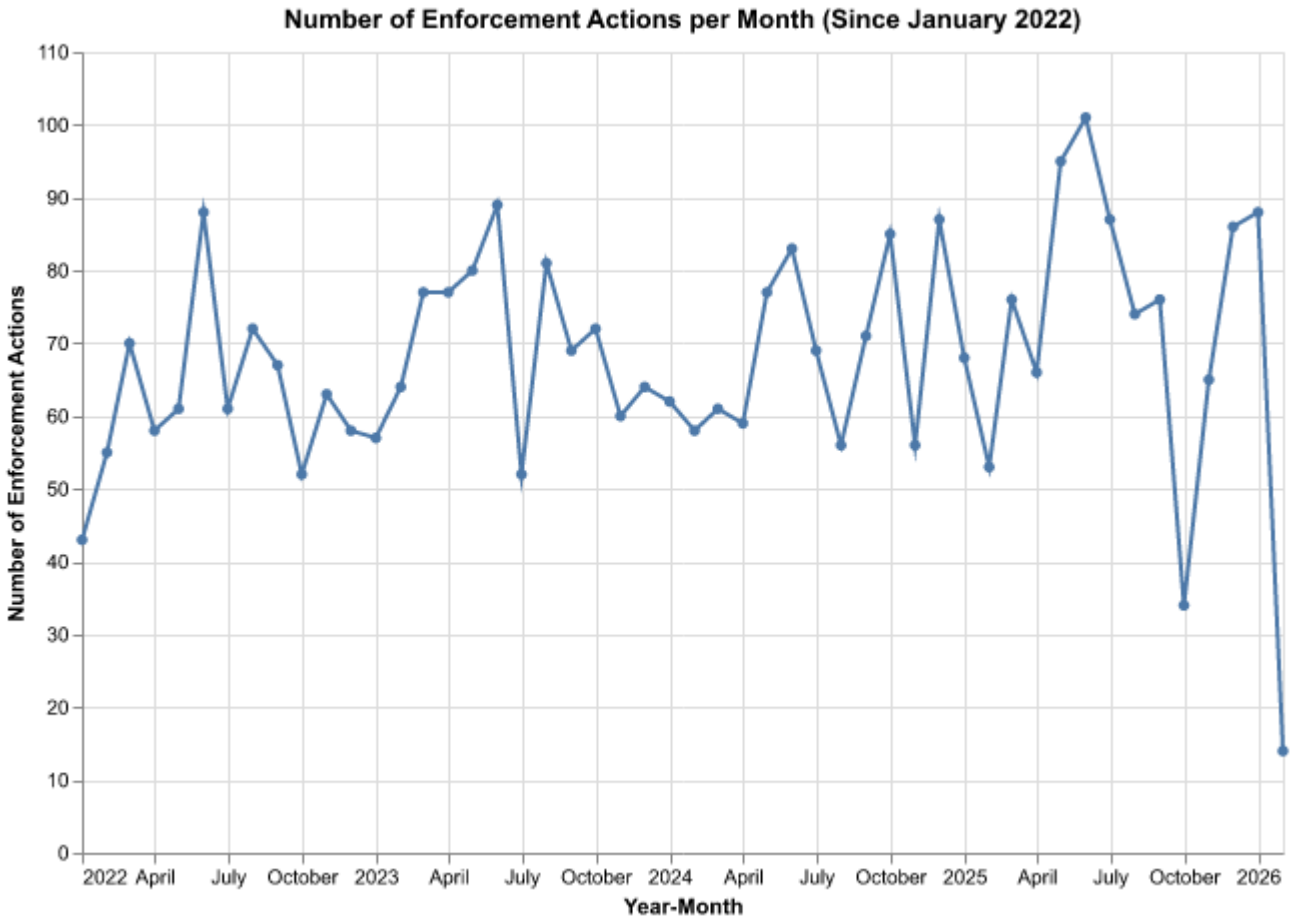
This dataframe is used for all subsequent questions in the problem set.

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

# Use the dataframe generated in Step 2(c)
df = df_2022.copy()

# Convert date column to datetime
df["date_dt"] = pd.to_datetime(df["date"])

# Create a year-month column for aggregation
df["year_month"] = df["date_dt"].dt.to_period("M").dt.to_timestamp()

# Aggregate: count number of enforcement actions per month
monthly_counts = (
    df.groupby("year_month")
      .size()
      .reset_index(name="num_actions")
)

# Create a line chart with Altair
line_chart = (
    alt.Chart(monthly_counts)
    .mark_line(point=True)
    .encode(
```

```
        x=alt.X("year_month:T", title="Year-Month"),
        y=alt.Y("num_actions:Q", title="Number of Enforcement Actions"),
        tooltip=[
            alt.Tooltip("year_month:T", title="Month"),
            alt.Tooltip("num_actions:Q", title="Actions")
        ]
    )
    .properties(
        title="Number of Enforcement Actions per Month (Since January 2022)",
        width=600,
        height=400
    )
)

line_chart
```



The figure above shows the number of enforcement actions over time, aggregated by month and year, since January 2022.

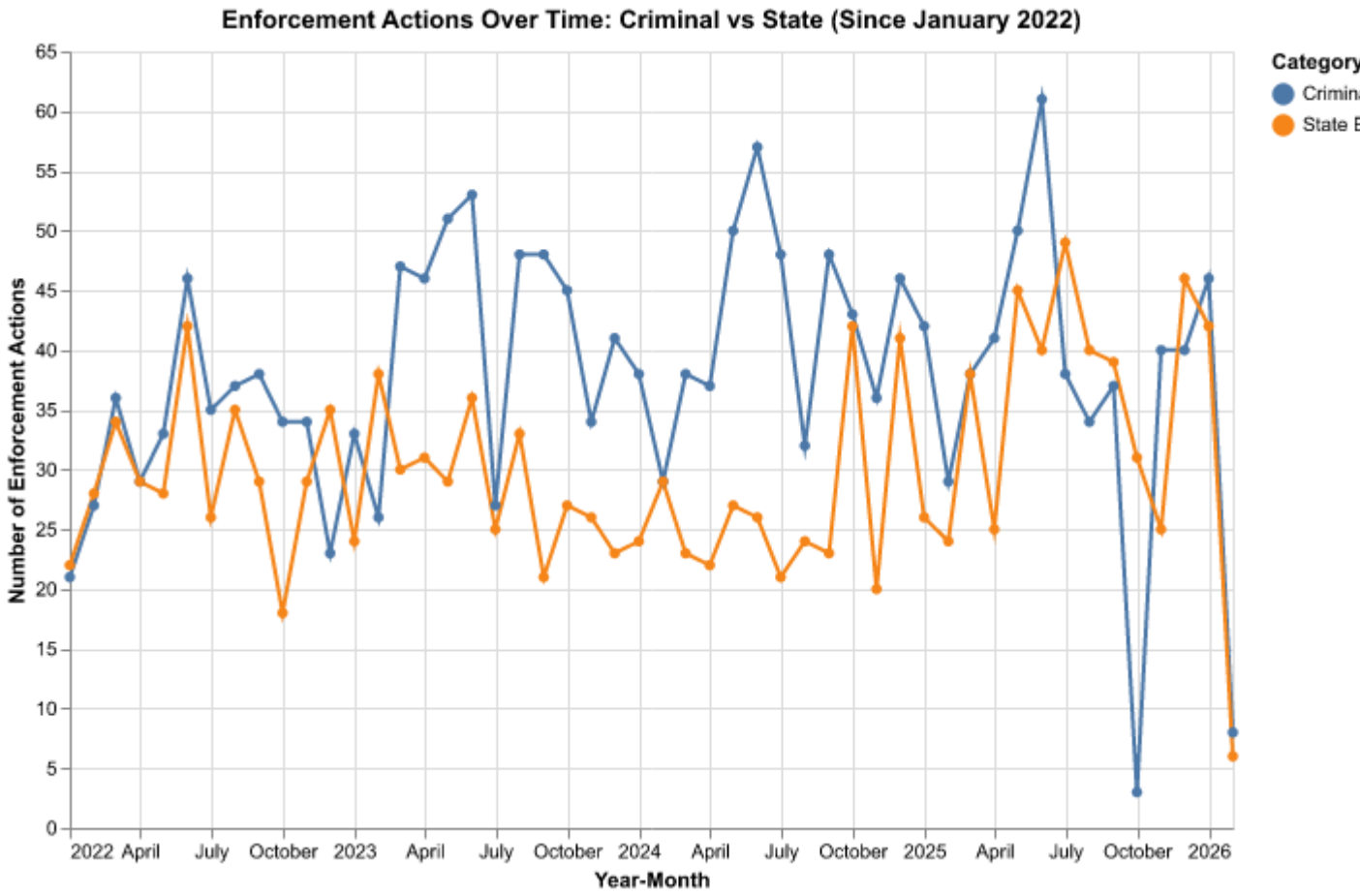
Each point represents the total number of enforcement actions in a given month. Overall, enforcement activity fluctuates over time, with noticeable month-to-month variation.

2. Plot the number of enforcement actions categorized:

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

```
import altair as alt
import pandas as pd

# Use the dataframe from Step 2(c)
df = df_2022.copy()

# Convert date to datetime and extract year-month
df["date_dt"] = pd.to_datetime(df["date"])
df["year_month"] = df["date_dt"].dt.to_period("M").dt.to_timestamp()

# Create a simplified category grouping
def category_group(cat):
    if cat == "Criminal and Civil Actions":
        return "Criminal and Civil Actions"
    else:
        return "State Enforcement Agencies"

df["category_group"] = df["category"].apply(category_group)

# Aggregate counts by month and category group
monthly_cat_counts = (
    df.groupby(["year_month", "category_group"])
      .size()
      .reset_index(name="num_actions")
)

# Plot line chart with Altair
cat_line_chart = (
    alt.Chart(monthly_cat_counts)
    .mark_line(point=True)
    .encode(
        x=alt.X("year_month:T", title="Year-Month"),
        y=alt.Y("num_actions:Q", title="Number of Enforcement Actions"),
```

```
        color=alt.Color("category_group:N", title="Category"),
        tooltip=[
            alt.Tooltip("year_month:T", title="Month"),
            alt.Tooltip("category_group:N", title="Category"),
            alt.Tooltip("num_actions:Q", title="Actions")
        ]
    )
    .properties(
        title="Enforcement Actions Over Time: Criminal vs State (Since
↪  January 2022)",
        width=600,
        height=400
    )
)

cat_line_chart
```



The figure above shows the number of enforcement actions over time, split between Criminal and Civil Actions and State Enforcement Agencies.

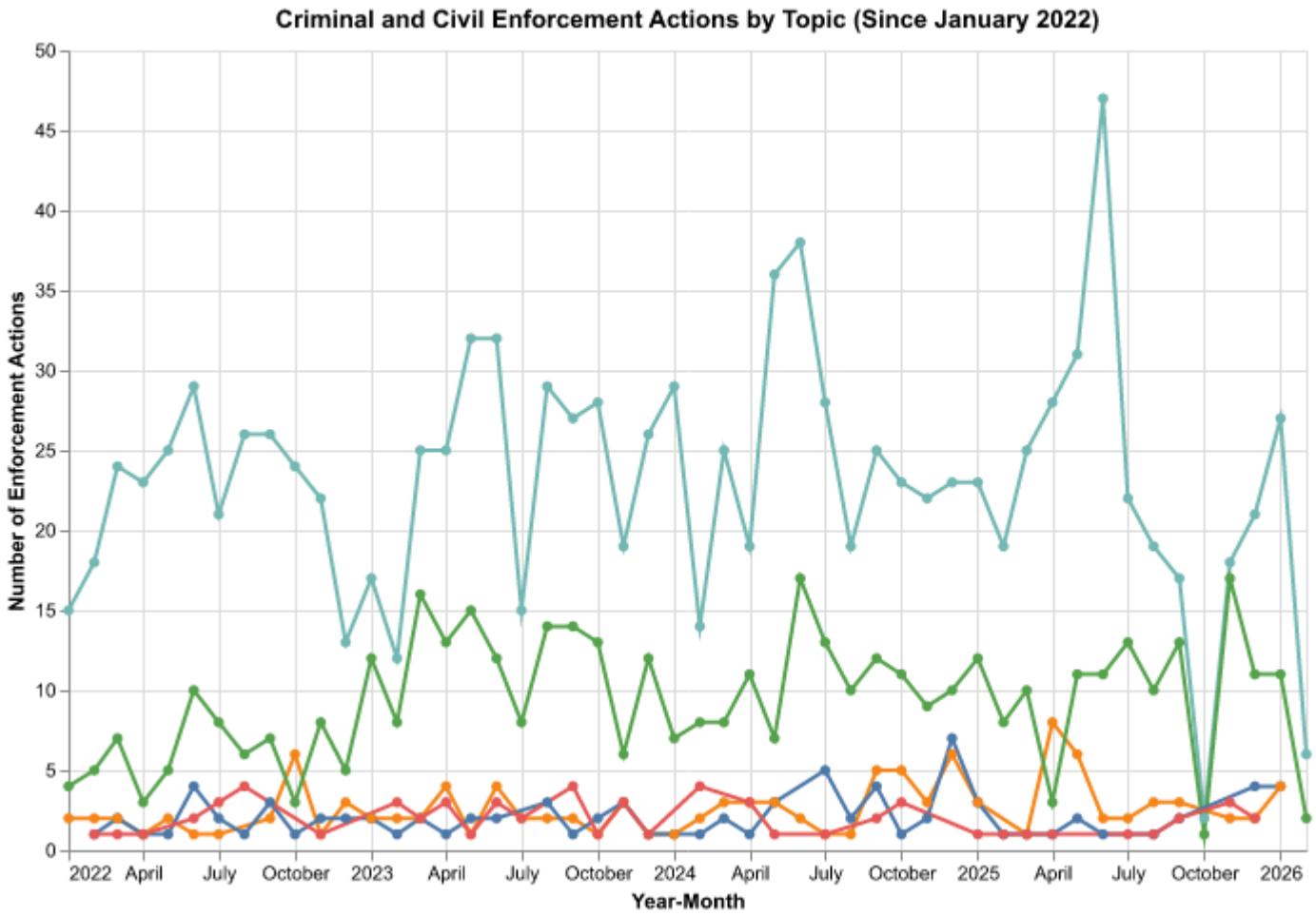
Each line represents the monthly count of enforcement actions for each category since January 2022. The plot highlights differences in enforcement activity across the two categories over time.

- based on five topics

```
# Step 3(2b): Split Criminal and Civil Actions into five topics

import altair as alt
import pandas as pd

# Use the dataframe from Step 2(c)
df = df_2022.copy()
```

```
# Keep only Criminal and Civil Actions
df = df[df["category"] == "Criminal and Civil Actions"].copy()

# Convert date to datetime and extract year-month
df["date_dt"] = pd.to_datetime(df["date"])
df["year_month"] = df["date_dt"].dt.to_period("M").dt.to_timestamp()

# Assign topic based on keywords in the title
def assign_topic(title):
    title = title.lower()

    if any(word in title for word in ["bank", "financial", "loan",
     ↪  "mortgage", "wire", "credit", "tax"]):
        return "Financial Fraud"
    elif any(word in title for word in ["health", "medicare", "medicaid",
     ↪  "hospital", "clinic", "physician", "nurse", "pharmacy"]):
        return "Health Care Fraud"
    elif any(word in title for word in ["drug", "opioid", "fentanyl",
     ↪  "controlled substance", "pill"]):
        return "Drug Enforcement"
    elif any(word in title for word in ["bribe", "bribery", "kickback",
     ↪  "corruption", "embezzle"]):
        return "Bribery/Corruption"
    else:
        return "Other"

df["topic"] = df["title"].apply(assign_topic)

# Aggregate counts by month and topic
monthly_topic_counts = (
    df.groupby(["year_month", "topic"])
      .size()
      .reset_index(name="num_actions")
)

# Plot line chart with Altair
topic_line_chart = (
    alt.Chart(monthly_topic_counts)
    .mark_line(point=True)
    .encode(
        x=alt.X("year_month:T", title="Year-Month"),
        y=alt.Y("num_actions:Q", title="Number of Enforcement Actions"),
```

```
        color=alt.Color("topic:N", title="Topic"),
        tooltip=[
            alt.Tooltip("year_month:T", title="Month"),
            alt.Tooltip("topic:N", title="Topic"),
            alt.Tooltip("num_actions:Q", title="Actions")
        ]
    )
    .properties(
        title="Criminal and Civil Enforcement Actions by Topic (Since January
↪  2022)",
        width=650,
        height=420
    )
)
topic_line_chart
```



The figure above shows the number of Criminal and Civil enforcement actions over time, disaggregated into five topics: Health Care Fraud, Financial Fraud, Drug Enforcement, Bribery/Corruption, and Other.

Each enforcement action is classified into a topic based on keywords in its title. The plot illustrates how the composition of criminal and civil enforcement activity varies across different types of offenses since January 2022.