ps4

Iraj Butt

Invalid Date

Due 02/07/2026 at 5:00PM Central

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

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.

(40%) Step 1: Develop initial scraper and crawler

Scraping: Go to the first page of the HHS OIG's "Enforcement Actions" page and scrape and collect the following into a dataset: * Title of the enforcement action * Date * Category (e.g, "Criminal and Civil Actions") * Link associated with the enforcement action

Collect your output into a tidy dataframe and print its `head`.

Hint: if you go to James A. Robinson's profile page at the Nobel Prize website here, right-click anywhere along the line "Affiliation at the time of the award: University of Chicago, Chicago, IL, USA", and select Inspect, you'll see that this affiliation information is located at the third `<p>` tag out of five `<p>` tags under the `<div class="content">`. Think about how you can select the third element of `<p>` out of five `<p>` elements so you're sure you scrape the affiliation information, not other. This way, you can scrape the name of agency to answer this question.

```
import pandas as pd
import altair as alt

alt.renderers.enable("default")

import requests
from bs4 import BeautifulSoup
from datetime import datetime
import time

import warnings
warnings.filterwarnings('ignore')
alt.renderers.enable("png")
```

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

1. We extract enforcement action data from the HHS OIG's enforcement actions database at https://oig.hhs.gov/fraud/enforcement/. Each enforcement action appears as a list item with the usa-card class. The title and its corresponding link are found in an tag labeled usa-card___heading that wraps an element. The date when the action occurred is stored in a element with class text-base-dark. The category of the enforcement action, such as whether it's a criminal case or state agency action, appears in an tag with the display-inline-block class.

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

```
cards = soup.find_all('li', class_='usa-card')

titles = []
dates = []
categories = []
links = []

for card in cards:
    heading = card.find('h2', class_='usa-card__heading')
    if heading:
        a_tag = heading.find('a')
        title = a_tag.get_text(strip=True)
        link = "https://oig.hhs.gov" + a_tag['href']

        date_span = card.find('span', class_='text-base-dark')
        date_str = date_span.get_text(strip=True) if date_span else None

        tag_li = card.find('li', class_='display-inline-block')
        category = tag_li.get_text(strip=True) if tag_li else None

        titles.append(title)
        dates.append(date_str)
        categories.append(category)
        links.append(link)

df_page1 = pd.DataFrame({
    'title': titles,
    'date': dates,
    'category': categories,
    'link': links
})

print(df_page1.head())
```

```
                                               title             date  \
0  Houston Transplant Doctor Indicted For Making ...  February 5, 2026
1  MultiCare Health System to Pay Millions to Set...  February 4, 2026
2  Brooklyn Banker Pleads Guilty to Laundering Pr...  February 3, 2026
3  Delafield Man Sentenced to 18 Months' Imprison...  February 3, 2026
4  Former NFL Player Convicted for $197M Medicare...  February 3, 2026

                    category  \
```

```
0  Criminal and Civil Actions
1  Criminal and Civil Actions
2                    COVID-19
3  Criminal and Civil Actions
4  Criminal and Civil Actions

                                                  link
0  https://oig.hhs.gov/fraud/enforcement/houston-...
1  https://oig.hhs.gov/fraud/enforcement/multicar...
2  https://oig.hhs.gov/fraud/enforcement/brooklyn...
3  https://oig.hhs.gov/fraud/enforcement/delafiel...
4  https://oig.hhs.gov/fraud/enforcement/former-n...
```

(40%) Step 2: Making the scraper dynamic

1. **Turning the scraper into a function**: You will write a function that takes as input a month and a year, and then pulls and formats the enforcement actions like in Step 1 starting from that month+year to today.

    - It is *very important* to make sure that you include an indicator whether or not to actually run the function. If you do not include an indicator, then each time you try to knit the qmd file, the scraper will run, and it will take a very long time to compile your pdf. Instead, run the function once to create a file called enforcement_actions_year_month.csv, which you can use in subsequent parts of the pset. Then turn the indicator off so that knitting your qmd file goes smoothly.
    - This function should first check that the year inputted >= 2013 before starting to scrape. If the year inputted < 2013, it should print a statement reminding the user to restrict to year >= 2013, since only enforcement actions after 2013 are listed.
    - It should save the dataframe output into a .csv file named as "enforcement_actions_*year_month*.csv" (do not commit this file to git)
    - If you're crawling multiple pages, always add 1 second wait before going to the next page to prevent potential server-side block. To implement this in Python, you may look up `.sleep()` function from `time` library.

a. Before writing out your function, write down pseudo-code of the steps that your function will go through. If you use a loop, discuss what kind of loop you will use and how you will define it. *Hint: Note that a simple* `for` *loop may not be sufficient for what this crawler requires. Use online resources to look into different types of loops or different ways of using* `for` *loops to see if there is something that is more appropriate for this task.*
b. Now code up your dynamic scraper and run it to start collecting the enforcement actions since January 2024. How many enforcement actions do you get in your final dataframe? What is the date and details of the earliest enforcement action it scraped?

c. Now, let's go a little further back. Test your code by collecting the actions since January 2022. *Note that this can take a while.* How many enforcement actions do you get in your final dataframe? What is the date and details of the earliest enforcement action it scraped? Use the dataframe from this process for every question after this.

Hint:

```
If you go to the next page in this HHS OIG's "Enforcement Actions" page,
you'll notice a pattern:

  * Second page URL: https://oig.hhs.gov/fraud/enforcement/?page=2
  * Third page URL: https://oig.hhs.gov/fraud/enforcement/?page=3
  * and so on ...
```

a. Pseudo-code:

```
FUNCTION scrape_enforcement_actions(month, year):
    IF year < 2013:
        PRINT "Error: year must be >= 2013"
        RETURN empty_dataframe

    actions = []
    page = 1

    WHILE TRUE:
        url = "https://oig.hhs.gov/fraud/enforcement/?page=" + page
        response = FETCH(url)
        soup = PARSE(response)

        cards = FIND_ALL(soup, "li", class="usa-card")

        IF cards is EMPTY:
            BREAK

        FOR each card IN cards:
            title = EXTRACT(card, "h2.usa-card__heading > a")
            link = GET_ATTRIBUTE(card, "href")
            date = EXTRACT(card, "span.text-base-dark")
            category = EXTRACT(card, "li.display-inline-block")

            date_obj = PARSE_DATE(date)

            IF date_obj < start_date:
                BREAK
```

```
            APPEND {title, date, category, link} TO actions

        page = page + 1
        SLEEP(1)

    df = CREATE_DATAFRAME(actions)
    SAVE_CSV(df, "enforcement_actions_" + year + "_" + month + ".csv")
    RETURN df
```

b. Dynamic scraper

```
ENABLE_SCRAPING = False

def collect_enforcement_data(starting_month, starting_year):
    """
    Collects HHS OIG enforcement action records beginning from a specified
    month and year through the present day. Iterates through paginated
    results and stops when encountering actions that predate the start
↪   period.
    """

    if starting_year < 2013:
        print("Note: HHS OIG enforcement action records begin in 2013. "
              "Please select a starting year of 2013 or later.")
        return None

    cutoff_date = datetime(starting_year, starting_month, 1)

    titles = []
    dates = []
    categories = []
    urls = []

    # Initialize pagination variables
    current_page = 1
    has_more_pages = True

    while has_more_pages:
        page_url =
↪   f"https://oig.hhs.gov/fraud/enforcement/?page={current_page}"
        page_response = requests.get(page_url)
        parsed_html = BeautifulSoup(page_response.text, 'html.parser')
```

```
        action_items = parsed_html.find_all('li', class_='usa-card')

        if not action_items:
            break

        for item in action_items:
            heading_section = item.find('h2', class_='usa-card__heading')

            if heading_section:
                link_element = heading_section.find('a')
                action_title = link_element.get_text(strip=True)
                action_url = "https://oig.hhs.gov" + link_element['href']

                date_container = item.find('span', class_='text-base-dark')
                action_date_text = (date_container.get_text(strip=True)
                                    if date_container else None)

                category_item = item.find('li',
↪   class_='display-inline-block')
                action_category = (category_item.get_text(strip=True)
                                   if category_item else None)

                if action_date_text:
                    parsed_action_date = datetime.strptime(action_date_text,
↪   "%B %d, %Y")

                    if parsed_action_date < cutoff_date:
                        has_more_pages = False
                        break

                titles.append(action_title)
                dates.append(action_date_text)
                categories.append(action_category)
                urls.append(action_url)

        current_page += 1
        time.sleep(1)

    results_table = pd.DataFrame({
        'title': titles,
        'date': dates,
        'category': categories,
```

```
        'link': urls
    })

    output_filename =
↪   f"enforcement_actions_{starting_year}_{starting_month}.csv"
    results_table.to_csv(output_filename, index=False)
    print(f"Exported {len(results_table)} records to {output_filename}")

    return results_table

# Execute the data collection process for January 2024
if ENABLE_SCRAPING:
    data_2024 = collect_enforcement_data(1, 2024)
    print(f"\nTotal records retrieved since Jan 2024: {len(data_2024)}")
    print(f"\nOldest record in dataset:")
    print(data_2024.tail(1))
```

c.

```
if ENABLE_SCRAPING:
       df_2022 = collect_enforcement_data(1, 2022)
       print(f"Total: {len(df_2022)}")
       print(df_2022.tail(1))
```

(20%) Step 3: Plot data based on scraped data

Note: To complete this part of the pset, reference the csv file you created in step 2, enforcement_actions_year_month.csv.

1. Plot a line chart with altair that shows: **the number of enforcement actions** over time (aggregated to each month+year) overall since January 2022,
2. Plot a line chart with altair that shows: **the number of enforcement actions** split out by:
    - "Criminal and Civil Actions" vs. "State Enforcement Agencies"
    - Five topics in the "Criminal and Civil Actions" category: "Health Care Fraud", "Financial Fraud", "Drug Enforcement", "Bribery/Corruption", and "Other". *Hint: You will need to divide the five topics manually by looking at the title and assigning the relevant topic. For example, if you find the word "bank" or "financial" in the title of an action, then that action should probably belong to "Financial Fraud" topic.*

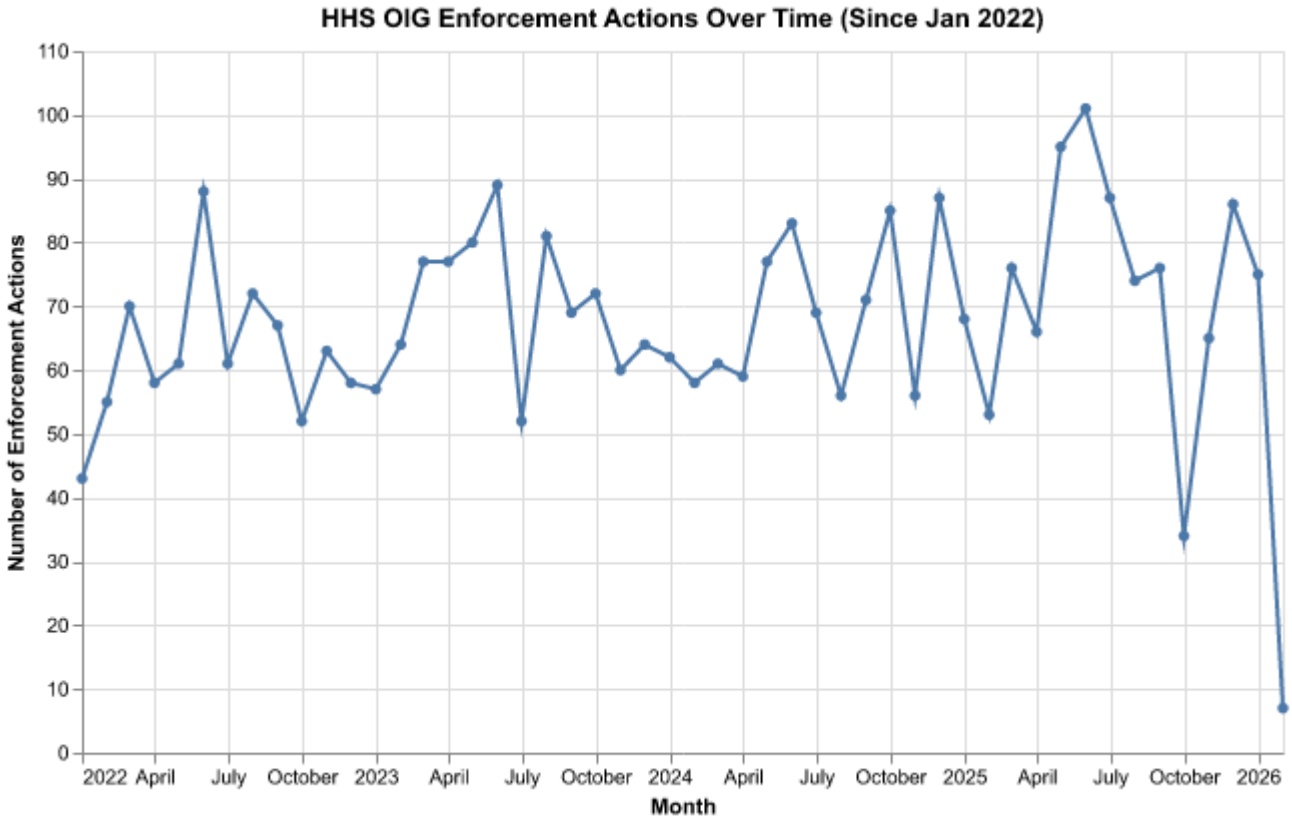
We suggest using AI to identify patterns in your scraped data and suggest ways of classifying based on the titles.

```
df = pd.read_csv("enforcement_actions_2022_1.csv")
df['date'] = pd.to_datetime(df['date'], format='%B %d, %Y')
df['year_month'] = df['date'].dt.to_period('M').dt.to_timestamp()
```

```
monthly_counts = (df.groupby('year_month')
                  .size()
                  .reset_index(name='count'))

chart1 = (alt.Chart(monthly_counts)
    .mark_line(point=True)
    .encode(
        x=alt.X('year_month:T', title='Month'),
        y=alt.Y('count:Q', title='Number of Enforcement Actions'),
        tooltip=['year_month:T', 'count:Q']
    )
    .properties(
        title='HHS OIG Enforcement Actions Over Time (Since Jan 2022)',
        width=600,
        height=350
    )
)

chart1
```



2. Plot the number of enforcement actions categorized:

"Criminal and Civil Actions" vs. "State Enforcement Agencies"

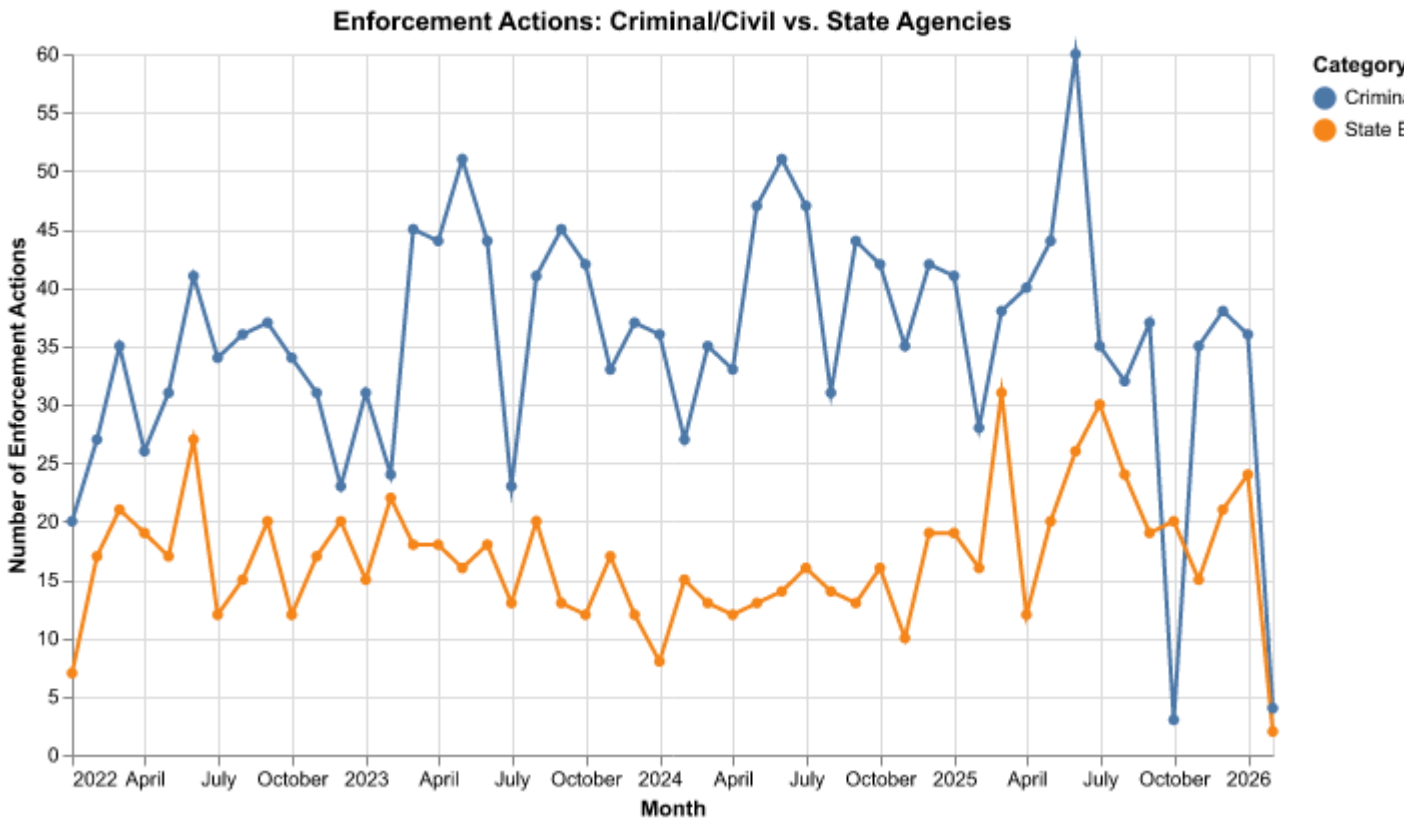
```
# Filter to just the two main categories
df_two_cat = df[df['category'].isin(
    ['Criminal and Civil Actions', 'State Enforcement Agencies']
)]

monthly_by_cat = (df_two_cat.groupby(['year_month', 'category'])
                  .size()
                  .reset_index(name='count'))

chart2 = (alt.Chart(monthly_by_cat)
    .mark_line(point=True)
    .encode(
        x=alt.X('year_month:T', title='Month'),
        y=alt.Y('count:Q', title='Number of Enforcement Actions'),
        color=alt.Color('category:N', title='Category'),
```

```
        tooltip=['year_month:T', 'category:N', 'count:Q']
    )
    .properties(
        title='Enforcement Actions: Criminal/Civil vs. State Agencies',
        width=600,
        height=350
    )
)

chart2
```



five topics

```
# Classify "Criminal and Civil Actions" into five topics based on title
df_cca = df[df['category'] == 'Criminal and Civil Actions'].copy()

def classify_topic(title):
    title_lower = title.lower()
```

```
    # Drug Enforcement
    drug_keywords = ['drug', 'opioid', 'fentanyl', 'narcotic', 'controlled
↪   substance',
                     'prescription', 'pill mill', 'distributing medication',
                     'illegal distribution', 'dea', 'methamphetamine',
                      ↪  'cocaine',
                     'heroin', 'morphine', 'adderall', 'oxycodone',
                      ↪  'hydrocodone',
                     'pharmaceutical diversion', 'drug diversion',
                      ↪  'substance']

    # Bribery/Corruption
    bribery_keywords = ['brib', 'corrupt', 'embezzl', 'kickback', 'extort',
                        'gratuity', 'influence', 'public official', 'gift',
                        'anti-kickback']

    # Financial Fraud
    financial_keywords = ['bank', 'financial', 'money launder', 'tax
↪   evasion',
                          'tax fraud', 'wire fraud', 'wire transfer',
                          'embezzlement', 'identity theft', 'identity fraud',
                          'passport fraud', 'stolen identity', 'forgery',
                          'counterfeit', 'theft of government', 'theft
                           ↪  related',
                          'government funds']

    # Health Care Fraud (broad catch for medical/health care fraud)
    health_keywords = ['health care fraud', 'medicare', 'medicaid',
↪   'tricare',
                       'false claims', 'false billing', 'billing scheme',
                       'upcoding', 'phantom billing', 'unnecessary services',
                       'medically unnecessary', 'durable medical equipment',
                       'home health', 'hospice', 'nursing', 'hospital',
                       'physician', 'doctor', 'clinic', 'telemedicine',
                       'telehealth', 'laboratory', 'ambulance', 'pharmacy',
                       'patient', 'medical', 'health', 'hhs', 'cms',
                       'insurance fraud', 'claims', 'restitution',
                       'settle', 'resolve', 'plea', 'sentenced', 'convicted',
                       'indicted', 'charged', 'guilty']

    # Check in order of specificity
```

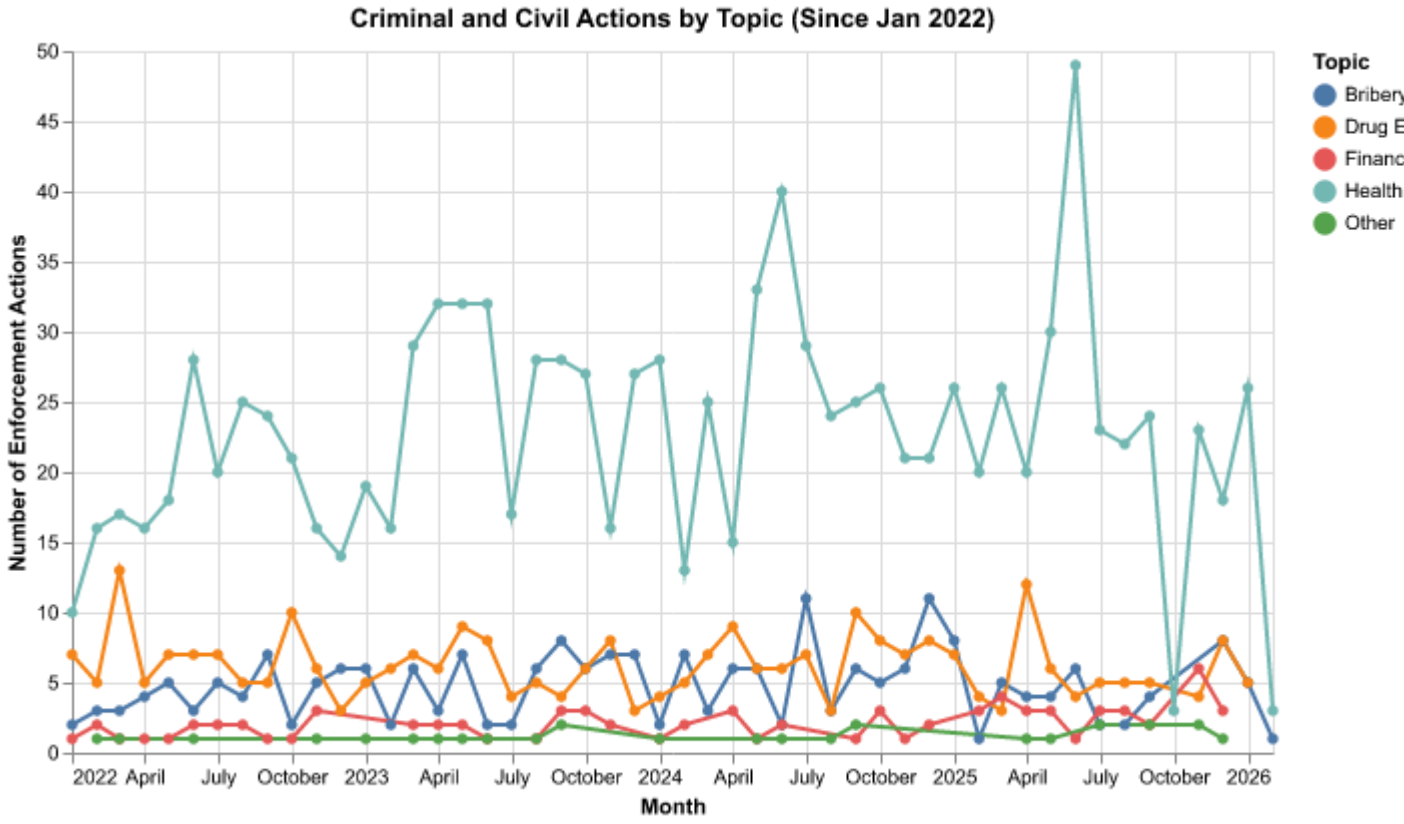
```
    if any(kw in title_lower for kw in drug_keywords):
        return 'Drug Enforcement'
    elif any(kw in title_lower for kw in bribery_keywords):
        return 'Bribery/Corruption'
    elif any(kw in title_lower for kw in financial_keywords):
        return 'Financial Fraud'
    elif any(kw in title_lower for kw in health_keywords):
        return 'Health Care Fraud'
    else:
        return 'Other'

df_cca['topic'] = df_cca['title'].apply(classify_topic)

monthly_by_topic = (df_cca.groupby(['year_month', 'topic'])
                    .size()
                    .reset_index(name='count'))

chart3 = (alt.Chart(monthly_by_topic)
    .mark_line(point=True)
    .encode(
        x=alt.X('year_month:T', title='Month'),
        y=alt.Y('count:Q', title='Number of Enforcement Actions'),
        color=alt.Color('topic:N', title='Topic'),
        tooltip=['year_month:T', 'topic:N', 'count:Q']
    )
    .properties(
        title='Criminal and Civil Actions by Topic (Since Jan 2022)',
        width=600,
        height=350
    )
)

chart3
```


Criminal and Civil Actions by Topic (Since Jan 2022)
Number of Enforcement Actions
50
45
40
35
30
25
20
15
10
5
0
2022
April
July
October
2023
April
July
October
2024
April
July
October
2025
April
July
October
2026
Month
Topic
Bribery
Drug E
Financ
Health
Other