PS4

Zhen Zeng

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: **ZZ**

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 requests
from bs4 import BeautifulSoup
import pandas as pd

# Fetch the webpage
url = "https://oig.hhs.gov/fraud/enforcement/"
response = requests.get(url)
response.raise_for_status()  # Check if request was successful

# Parse HTML
soup = BeautifulSoup(response.text, 'html.parser')

# Find all enforcement action cards
cards = soup.select('li.usa-card')

# Extract data from each card
data = []
for card in cards:
    # Get title and link from the heading anchor
    title_element = card.select_one('.usa-card__heading a')

    # Get date from the header span
    date_element = card.select_one('header span.text-base-dark')

    # Get all category tags (there may be multiple)
    category_elements = card.select('.usa-tag')

    if title_element:
        # Extract title text
```

```
        title = title_element.get_text(strip=True)

        # Extract and format the link (ensure it's a full URL)
        link = title_element.get('href', '')
        if link.startswith('/'):
            link = "https://oig.hhs.gov" + link

        # Extract date
        date = date_element.get_text(strip=True) if date_element else None

        # Extract categories (join multiple categories with "; ")
        categories = "; ".join([cat.get_text(strip=True) for cat in
↪  category_elements]) if category_elements else None

        data.append({
            'Title': title,
            'Date': date,
            'Category': categories,
            'Link': link
        })

# Create a tidy DataFrame
df = pd.DataFrame(data)

# Print the head of the dataframe
print(df.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...
```

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code

```
PSEUDO-CODE for Dynamic Scraper Function:

# Helper function to parse date strings
FUNCTION parse_date(date_str):
    TRY:
        RETURN parse "Month DD, YYYY" format to datetime object
    EXCEPT:
        RETURN None

FUNCTION scrape_enforcement_actions(start_month, start_year,
run_scraper=False):

    # Step 1: Check the run_scraper indicator
    IF run_scraper is False:
        PRINT "Scraper is turned off. Set run_scraper=True to run."
        RETURN None

    # Step 2: Validate year input (must be >= 2013)
    IF start_year < 2013:
        PRINT "Error: Please use year >= 2013. Only enforcement actions after
        2013 are listed."
        RETURN None

    # Step 3: Initialize variables
    SET base_url = "https://oig.hhs.gov/fraud/enforcement/"
    SET all_data = empty list
    SET current_page = 1
    SET cutoff_date = datetime(start_year, start_month, 1)  # Client-side
    filter
    SET reached_cutoff = False

    # Step 4: Use a WHILE loop to iterate through pages
```

```
# NOTE: URL date parameters do NOT work server-side when using requests
# So we use client-side date filtering instead

WHILE reached_cutoff is False:
    # Step 4a: Construct URL with only page parameter
    # (Date filtering happens client-side by parsing each record's date)
    SET url = base_url + "?page=" + current_page

    # Step 4b: Make HTTP request
    TRY:
        response = GET request to url with timeout
        IF response status is not OK:
            BREAK loop
    EXCEPT:
        PRINT error message
        BREAK loop

    # Step 4c: Parse HTML and extract cards
    soup = parse HTML from response
    cards = select all 'li.usa-card' elements

    # Step 4d: Check if we have any cards (stopping condition 1)
    IF no cards found:
        BREAK loop  # No more data, exit the while loop

    # Step 4e: Extract data from each card WITH date checking
    FOR each card in cards:
        title = extract from '.usa-card__heading a' text
        link = extract from '.usa-card__heading a' href attribute
        date_str = extract from 'header span.text-base-dark' text
        category = extract from '.usa-tag' elements, join with "; "

        # CLIENT-SIDE DATE FILTERING (stopping condition 2)
        parsed_date = parse_date(date_str)
        IF parsed_date < cutoff_date:
            SET reached_cutoff = True
            BREAK inner for loop  # Stop scraping, we've gone past our
            date range

        APPEND {title, date_str, category, link} to all_data

    # Step 4f: Increment page counter
    current_page = current_page + 1
```

```
        # Step 4g: Wait 1 second before next request (to prevent server
        block)
        SLEEP 1 second

    # Step 5: Create DataFrame from collected data
    df = create DataFrame from all_data

    # Step 6: Save to CSV file
    filename = "enforcement_actions_" + start_year + "_" + start_month +
    ".csv"
    SAVE df to filename

    # Step 7: Return the DataFrame
    RETURN df

END FUNCTION

---
KEY DESIGN DECISIONS:

1. WHILE loop vs FOR loop:
   - A simple FOR loop with range() is NOT ideal because we don't know
     the total number of pages in advance.
   - A WHILE loop with a flag (reached_cutoff) is more appropriate because:
     * We continue scraping until we find a record older than our start date
     * This handles dynamic/unknown total page counts
     * The stopping condition is data-driven (date comparison)

2. CLIENT-SIDE date filtering (Important!):
   - URL date parameters do NOT work server-side when using Python requests
   - The website's date filtering is JavaScript-based (client-side only)
   - Solution: Parse each record's date and compare to cutoff_date
   - When we encounter a record with date < cutoff_date, stop scraping

3. Two stopping conditions:
   - No more cards on page (empty page) -> exhausted all data
   - Record date < cutoff_date -> reached our target date range

4. Run indicator:
   - A boolean flag controls whether scraping actually runs
   - Prevents accidental long-running operations when knitting the document
```

- b. Create Dynamic Scraper

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

def parse_date(date_str):
    """Parse date string like 'January 15, 2024' to datetime object"""
    try:
        return datetime.strptime(date_str, '%B %d, %Y')
    except:
        return None

def scrape_enforcement_actions(start_month, start_year, run_scraper=False):
    """
    Scrape HHS OIG enforcement actions from a given start date to today.

    Parameters:
    -----------
    start_month : int
        Starting month (1-12)
    start_year : int
        Starting year (must be >= 2013)
    run_scraper : bool
        If False, the scraper won't run (to prevent long compilation times)

    Returns:
    --------
    pd.DataFrame or None
        DataFrame containing scraped enforcement actions, or None if not run
    """

    # Check if scraper should run
    if not run_scraper:
        print("Scraper is turned off. Set run_scraper=True to run.")
        return None

    # Validate year input
    if start_year < 2013:
        print("Error: Please use year >= 2013. Only enforcement actions after
        ↪  2013 are listed.")
        return None
```

```
    # Initialize variables
    base_url = "https://oig.hhs.gov/fraud/enforcement/"
    all_data = []
    current_page = 1

    # Create a cutoff date for filtering (client-side filtering since URL
     ↪  params don't work)
    cutoff_date = datetime(start_year, start_month, 1)
    end_date = datetime.now()

    print(f"Starting scrape from {start_month:02d}/01/{start_year} to
     ↪  {end_date.strftime('%m/%d/%Y')}...")
    print(f"Will stop when reaching records before {cutoff_date.strftime('%B
     ↪  %d, %Y')}")

    reached_cutoff = False

    # Use a while loop since we don't know the total number of pages
    while not reached_cutoff:
        url = f"{base_url}?page={current_page}"

        try:
            # Make HTTP request
            response = requests.get(url, timeout=30)
            response.raise_for_status()
        except requests.RequestException as e:
            print(f"Error fetching page {current_page}: {e}")
            break

        # Parse HTML
        soup = BeautifulSoup(response.text, 'html.parser')
        cards = soup.select('li.usa-card')

        # Check if we have any cards (stopping condition)
        if not cards:
            print(f"No more results found. Stopped at page {current_page}.")
            break

        # Extract data from each card
        for card in cards:
            title_element = card.select_one('.usa-card__heading a')
            date_element = card.select_one('header span.text-base-dark')
            category_elements = card.select('.usa-tag')
```

```
            if title_element:
                title = title_element.get_text(strip=True)
                link = title_element.get('href', '')
                if link.startswith('/'):
                    link = "https://oig.hhs.gov" + link

                date_str = date_element.get_text(strip=True) if date_element
↪   else None
                categories = "; ".join([cat.get_text(strip=True) for cat in
↪   category_elements]) if category_elements else None

                # Parse the date and check if it's before cutoff
                parsed_date = parse_date(date_str)
                if parsed_date and parsed_date < cutoff_date:
                    reached_cutoff = True
                    print(f"\nReached cutoff date at page {current_page}.")
                    break

                all_data.append({
                    'Title': title,
                    'Date': date_str,
                    'Category': categories,
                    'Link': link
                })

        if current_page % 25 == 0:
            print(f"Page {current_page}: Total so far: {len(all_data)}")

        # Increment page counter
        current_page += 1

        # Wait 1 second before next request to prevent server-side block
        time.sleep(1)

    # Create DataFrame
    df = pd.DataFrame(all_data)

    # Save to CSV file
    filename = f"enforcement_actions_{start_year}_{start_month:02d}.csv"
    df.to_csv(filename, index=False)
    print(f"Data saved to {filename}")
    print(f"Total enforcement actions scraped: {len(df)}")
```

```
    return df
```

- c. Test Your Code

Test b: Scrape from January 2024

```
# b. Test the function - Scrape from January 2024
# SET run_scraper=True to actually run the scraper
# IMPORTANT: After running once, set run_scraper=False to avoid long knit
 ↪  times

# Uncomment to run:
# df_2024 = scrape_enforcement_actions(start_month=1, start_year=2024,
 ↪  run_scraper=True)

# For knitting, load from saved CSV:
df_2024 = pd.read_csv("enforcement_actions_2024_01.csv")
print(f"Total enforcement actions from Jan 2024: {len(df_2024)}")
print(f"\nDataFrame head:")
print(df_2024.head())
print(f"\nEarliest enforcement action:")
print(df_2024.tail(1))
```

```
Total enforcement actions from Jan 2024: 1771

DataFrame head:
                                               Title              Date  \
0  MultiCare Health System to Pay Millions to Set...  February 4, 2026
1  Brooklyn Banker Pleads Guilty to Laundering Pr...  February 3, 2026
2  Delafield Man Sentenced to 18 Months' Imprison...  February 3, 2026
3  Former NFL Player Convicted for $197M Medicare...  February 3, 2026
4  AG's Office Secures Indictments Against Peabod...  February 2, 2026

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

                                                Link
```

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

Earliest enforcement action:
                                                  Title             Date  \
1770  Former Nurse Aide Indicted In Death Of Clarksv...  January 3, 2024

                        Category  \
1770  State Enforcement Agencies

                                                   Link
1770  https://oig.hhs.gov/fraud/enforcement/former-n...
```

Results for b (January 2024 to today): - Total enforcement actions: **1,771** - Earliest enforcement action: - Title: "Former Nurse Aide Indicted In Death Of Clarksville Patient Arrested In Georgia" - Date: January 3, 2024 - Category: State Enforcement Agencies

---

Test c: Scrape from January 2022

```python
# c. Test the function - Scrape from January 2022 (this will be used for
 ↪  subsequent analysis)
# SET run_scraper=True to actually run the scraper (note: takes ~3-4 minutes)

# Uncomment to run:
# df_2022 = scrape_enforcement_actions(start_month=1, start_year=2022,
 ↪  run_scraper=True)

# For knitting, load from saved CSV:
df = pd.read_csv("enforcement_actions_2022_01.csv")
print(f"Total enforcement actions from Jan 2022: {len(df)}")
print(f"\nDataFrame head:")
print(df.head())
print(f"\nEarliest enforcement action:")
print(df.tail(1))
```

```
Total enforcement actions from Jan 2022: 3361
```

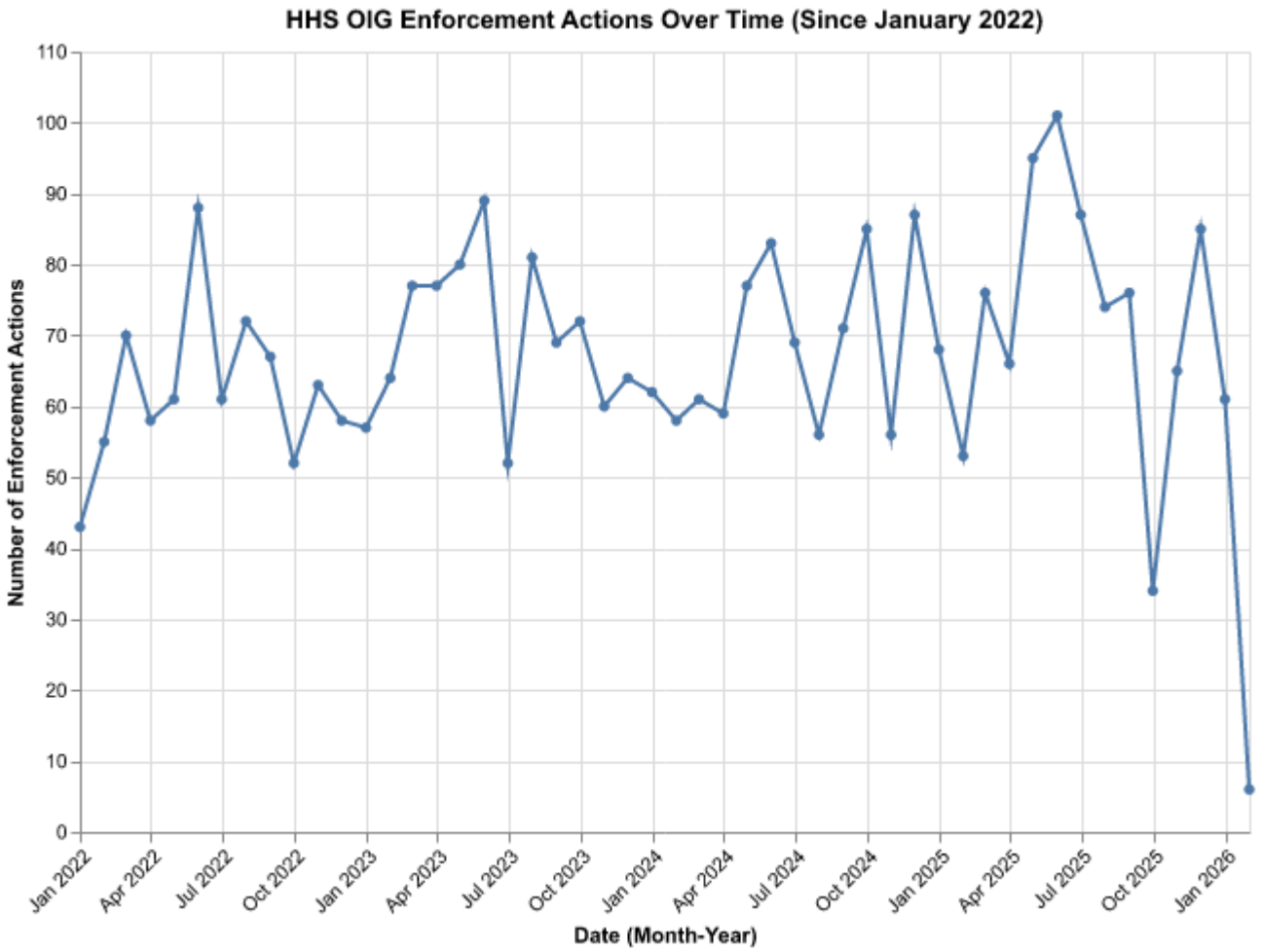
```
DataFrame head:
                                                Title              Date  \
0  Houston Transplant Doctor Indicted For Making ...  February 5, 2026
1  MultiCare Health System to Pay Millions to Set...  February 4, 2026
2  Brooklyn Banker Pleads Guilty to Laundering Pr...  February 3, 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...

Earliest enforcement action:
                                                   Title              Date  \
3360  Integrated Pain Management Medical Group Agree...  January 4, 2022

                    Category  \
3360  Fraud Self-Disclosures

                                                   Link
3360  https://oig.hhs.gov/fraud/enforcement/integrat...
```

Results for c (January 2022 to today): - Total enforcement actions: **3,361** - Earliest enforcement action: - Title: "Integrated Pain Management Medical Group Agreed to Pay $10,000 for Allegedly Violating the Civil Monetary Penalties Law by Employing Excluded Individuals" - Date: January 4, 2022 - Category: Fraud Self-Disclosures

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

```
import pandas as pd
import altair as alt
from datetime import datetime

# Load the data from CSV
df = pd.read_csv("enforcement_actions_2022_01.csv")

# Parse the date column (format: "January 15, 2024")
df['ParsedDate'] = pd.to_datetime(df['Date'], format='%B %d, %Y')

# Create a YearMonth column for aggregation
df['YearMonth'] = df['ParsedDate'].dt.to_period('M').astype(str)

# Aggregate: count enforcement actions per month
monthly_counts = df.groupby('YearMonth').size().reset_index(name='Count')

# Convert YearMonth back to datetime for proper ordering in chart
monthly_counts['YearMonth'] = pd.to_datetime(monthly_counts['YearMonth'])

# Create the line chart
chart = alt.Chart(monthly_counts).mark_line(
    point=True,  # Add points at each data point
    strokeWidth=2
).encode(
    x=alt.X('YearMonth:T',
            title='Date (Month-Year)',
            axis=alt.Axis(format='%b %Y', labelAngle=-45)),
    y=alt.Y('Count:Q',
            title='Number of Enforcement Actions'),
    tooltip=[
        alt.Tooltip('YearMonth:T', title='Month', format='%B %Y'),
        alt.Tooltip('Count:Q', title='Actions')
    ]
).properties(
    title='HHS OIG Enforcement Actions Over Time (Since January 2022)',
    width=600,
    height=400
)

chart
```



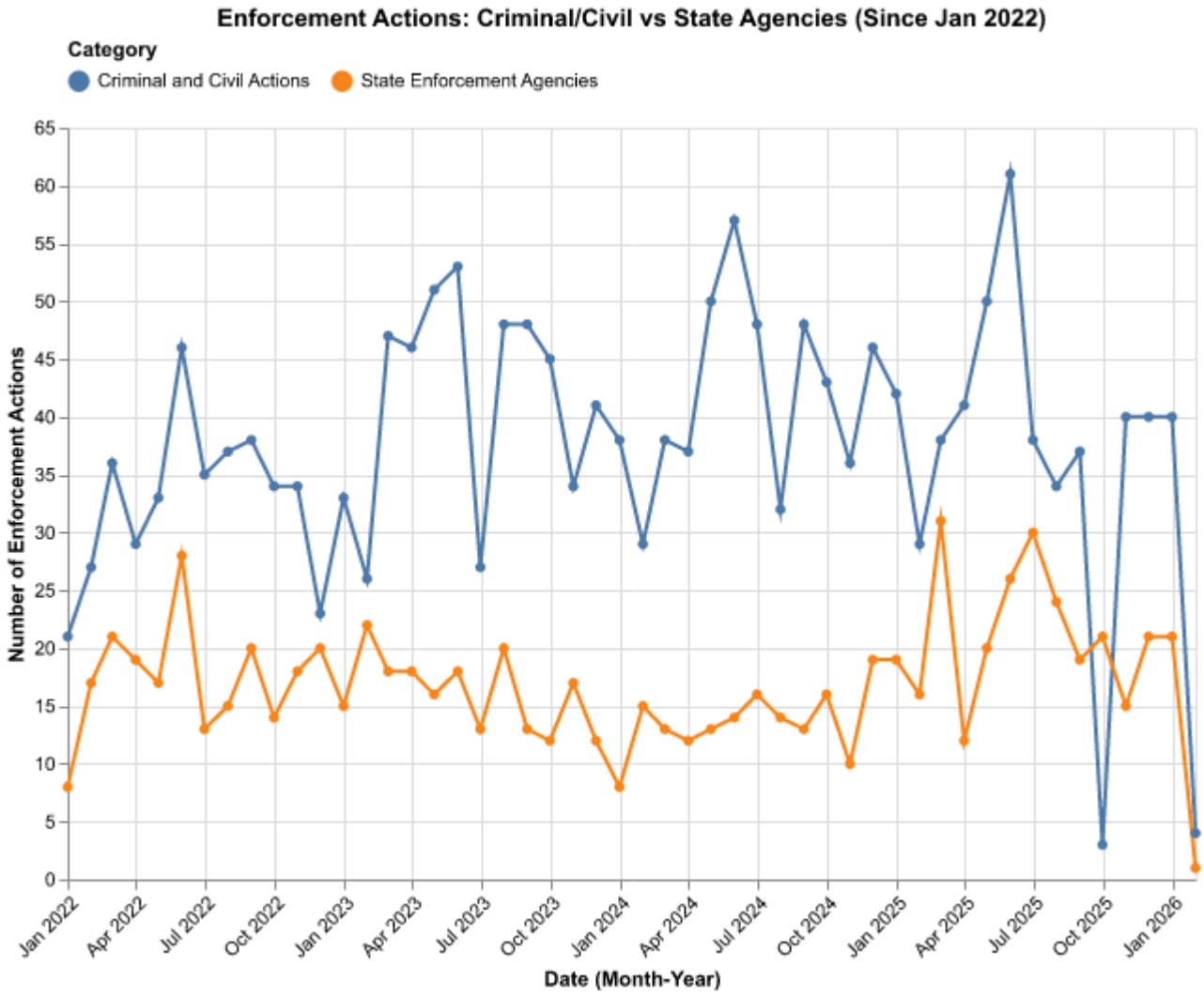
2. Plot the number of enforcement actions categorized:

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

```
import pandas as pd
import altair as alt

# Load data
df = pd.read_csv("enforcement_actions_2022_01.csv")
df['ParsedDate'] = pd.to_datetime(df['Date'], format='%B %d, %Y')
df['YearMonth'] = df['ParsedDate'].dt.to_period('M').astype(str)

# Create a simplified category: Criminal and Civil Actions vs State
 ↪  Enforcement Agencies vs Other
```

```
def categorize_agency(category):
    if pd.isna(category):
        return 'Other'
    elif 'Criminal and Civil Actions' in category:
        return 'Criminal and Civil Actions'
    elif 'State Enforcement Agencies' in category:
        return 'State Enforcement Agencies'
    else:
        return 'Other'

df['AgencyType'] = df['Category'].apply(categorize_agency)

# Filter to only the two main categories
df_filtered = df[df['AgencyType'].isin(['Criminal and Civil Actions', 'State
 ↪  Enforcement Agencies'])]

# Aggregate by month and agency type
monthly_by_agency = df_filtered.groupby(['YearMonth',
 ↪  'AgencyType']).size().reset_index(name='Count')
monthly_by_agency['YearMonth'] =
 ↪  pd.to_datetime(monthly_by_agency['YearMonth'])

# Create line chart
chart_agency = alt.Chart(monthly_by_agency).mark_line(
    point=True,
    strokeWidth=2
).encode(
    x=alt.X('YearMonth:T',
            title='Date (Month-Year)',
            axis=alt.Axis(format='%b %Y', labelAngle=-45)),
    y=alt.Y('Count:Q',
            title='Number of Enforcement Actions'),
    color=alt.Color('AgencyType:N',
                    title='Category',
                    legend=alt.Legend(orient='top')),
    tooltip=[
        alt.Tooltip('YearMonth:T', title='Month', format='%B %Y'),
        alt.Tooltip('AgencyType:N', title='Category'),
        alt.Tooltip('Count:Q', title='Actions')
    ]
).properties(
    title='Enforcement Actions: Criminal/Civil vs State Agencies (Since Jan
 ↪  2022)',
```

```
    width=600,
    height=400
)

chart_agency
```



- based on five topics

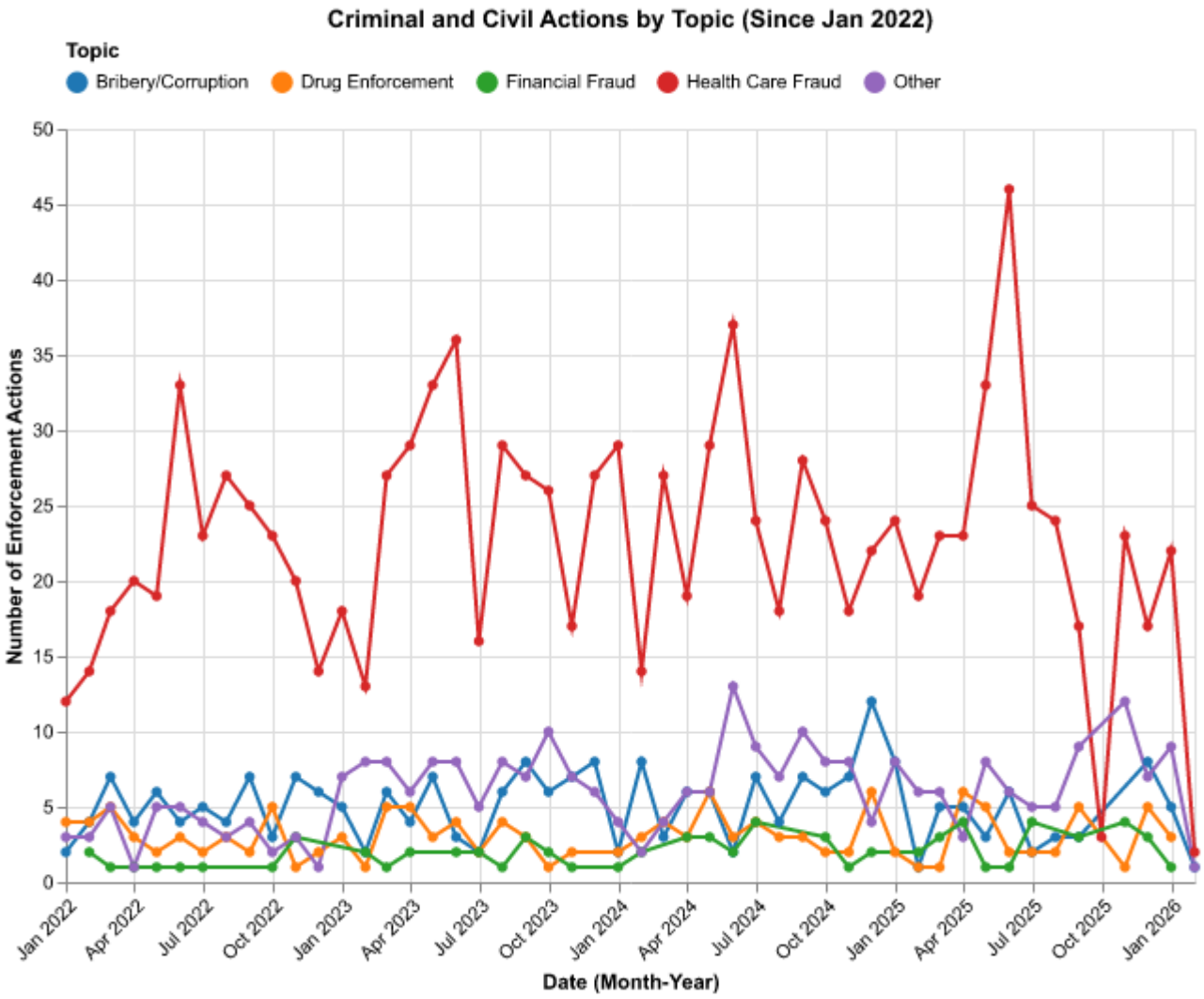
```
import pandas as pd
import altair as alt
import re
```

```
# Load data
df = pd.read_csv("enforcement_actions_2022_01.csv")
df['ParsedDate'] = pd.to_datetime(df['Date'], format='%B %d, %Y')
df['YearMonth'] = df['ParsedDate'].dt.to_period('M').astype(str)

# Filter to Criminal and Civil Actions only
df_criminal = df[df['Category'].str.contains('Criminal and Civil Actions',
 ↪  na=False)].copy()

# Classify into 5 topics based on title keywords
def classify_topic(title):
    """
    Classify enforcement action into one of 5 topics based on title keywords.
    Priority order matters - more specific topics are checked first.
    """
    title_lower = title.lower() if pd.notna(title) else ''

    # 1. Drug Enforcement - check first as it's more specific
    drug_keywords = ['drug trafficking', 'narcotics', 'cocaine', 'heroin',
 ↪  'fentanyl',
                     'methamphetamine', 'opioid', 'controlled substance',
                      ↪  'drug distribution',
                     'illegal drugs', 'drug dealer']
    if any(kw in title_lower for kw in drug_keywords):
        return 'Drug Enforcement'

    # 2. Bribery/Corruption
    bribery_keywords = ['bribe', 'bribery', 'corruption', 'kickback',
 ↪  'illegal gratuity']
    if any(kw in title_lower for kw in bribery_keywords):
        return 'Bribery/Corruption'

    # 3. Financial Fraud - non-healthcare financial crimes
    financial_keywords = ['money laundering', 'bank fraud', 'wire fraud',
 ↪  'tax fraud',
                          'tax evasion', 'embezzle', 'securities fraud',
                           ↪  'investment fraud',
                          'laundering proceeds', 'financial fraud']
    if any(kw in title_lower for kw in financial_keywords):
        return 'Financial Fraud'
```

```
    # 4. Health Care Fraud - the most common category
    healthcare_keywords = ['medicare', 'medicaid', 'health care fraud',
 ↪  'healthcare fraud',
                           'hospital', 'medical', 'patient', 'doctor',
                            ↪  'nurse', 'pharmacy',
                           'prescription', 'clinic', 'physician', 'therapist',
                            ↪  'false claims',
                           'durable medical', 'home health', 'hospice',
                            ↪  'laboratory',
                           'chiropractor', 'dentist', 'billing fraud']
    if any(kw in title_lower for kw in healthcare_keywords):
        return 'Health Care Fraud'

    # 5. Other - anything that doesn't fit above categories
    return 'Other'

df_criminal['Topic'] = df_criminal['Title'].apply(classify_topic)

# Display classification summary
print("Topic Classification Summary:")
print(df_criminal['Topic'].value_counts())

# Aggregate by month and topic
monthly_by_topic = df_criminal.groupby(['YearMonth',
 ↪  'Topic']).size().reset_index(name='Count')
monthly_by_topic['YearMonth'] = pd.to_datetime(monthly_by_topic['YearMonth'])

# Define custom color scheme
topic_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd']

# Create line chart
chart_topics = alt.Chart(monthly_by_topic).mark_line(
    point=True,
    strokeWidth=2
).encode(
    x=alt.X('YearMonth:T',
            title='Date (Month-Year)',
            axis=alt.Axis(format='%b %Y', labelAngle=-45)),
    y=alt.Y('Count:Q',
            title='Number of Enforcement Actions'),
    color=alt.Color('Topic:N',
                    title='Topic',
```

```
                    scale=alt.Scale(range=topic_colors),
                    legend=alt.Legend(orient='top')),
    tooltip=[
        alt.Tooltip('YearMonth:T', title='Month', format='%B %Y'),
        alt.Tooltip('Topic:N', title='Topic'),
        alt.Tooltip('Count:Q', title='Actions')
    ]
).properties(
    title='Criminal and Civil Actions by Topic (Since Jan 2022)',
    width=600,
    height=400
)

chart_topics
```

```
Topic Classification Summary:
Topic
Health Care Fraud     1136
Other                  293
Bribery/Corruption     243
Drug Enforcement       145
Financial Fraud         74
Name: count, dtype: int64
```


Criminal and Civil Actions by Topic (Since Jan 2022)
Topic
Bribery/Corruption
Drug Enforcement
Financial Fraud
Health Care Fraud
Other
Number of Enforcement Actions
0
5
10
15
20
25
30
35
40
45
50
Jan 2022
Apr 2022
Jul 2022
Oct 2022
Jan 2023
Apr 2023
Jul 2023
Oct 2023
Jan 2024
Apr 2024
Jul 2024
Oct 2024
Jan 2025
Apr 2025
Jul 2025
Oct 2025
Jan 2026
Date (Month-Year)