title

Grace Yao

2026-02-03

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: Grace Yao

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
import re
import altair as alt
import time

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


def scrape_enforcement_actions():
    """
    Scrape the first page of HHS OIG Enforcement Actions
    Returns a pandas DataFrame with Title, Date, Category, and Link
    """

    # URL for the first page of enforcement actions
    url = "https://oig.hhs.gov/fraud/enforcement/"

    # Send GET request with headers to mimic a browser
    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'
    }

    # Add a small delay to be respectful to the server
```

```
    time.sleep(1)

    response = requests.get(url, headers=headers)
    response.raise_for_status()

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

    # Initialize lists to store data
    titles = []
    dates = []
    categories = []
    links = []

    # Find all h2 tags that contain enforcement action titles
    # Based on the actual HTML structure, each enforcement action has:
    # - An h2 tag with a link (title and URL)
    # - A date after the h2
    # - A ul with categories as li items

    h2_tags = soup.find_all('h2')

    for h2 in h2_tags:
        # Get the parent element (should be an li or similar container)
        parent = h2.find_parent('li')
        if not parent:
            continue

        # Extract title and link from h2 > a
        a_tag = h2.find('a')
        if not a_tag:
            continue

        title = a_tag.get_text(strip=True)
        link = a_tag.get('href', '')

        # Make sure it's a full URL
        if link and not link.startswith('http'):
            link = 'https://oig.hhs.gov' + link

        # Extract date using regex
        # Date format: "Month Day, Year" (e.g., "December 10, 2025")
        text_content = parent.get_text()
```

```
        date_pattern =
↪   r'(January|February|March|April|May|June|July|August|September|October|November|December)
        date_match = re.search(date_pattern, text_content)

        date_text = date_match.group(0) if date_match else "N/A"

        # Extract categories
        # Categories are in a nested ul within the parent li
        category_ul = parent.find('ul')
        if category_ul:
            category_items = category_ul.find_all('li')
            categories_list = [cat.get_text(strip=True) for cat in
↪   category_items]
            category = ", ".join(categories_list) if categories_list else
↪   "N/A"
        else:
            category = "N/A"

        # Append to lists
        titles.append(title)
        dates.append(date_text)
        categories.append(category)
        links.append(link)

    # Create DataFrame
    df = pd.DataFrame({
        'Title': titles,
        'Date': dates,
        'Category': categories,
        'Link': links
    })

    return df


def main():
    """
    Main function to run the scraper and display results
    """
    print("="*80)
    print("Step 1: Scraping HHS OIG Enforcement Actions")
    print("="*80)
    print(f"URL: https://oig.hhs.gov/fraud/enforcement/")
```

```
print()

try:
    # Run the scraper
    df = scrape_enforcement_actions()

    # Display head of dataframe
    print("DataFrame Head:")
    print("-"*80)
    print(df.head())
    print()

    # Display basic statistics
    print("="*80)
    print(f"Total enforcement actions scraped: {len(df)}")
    print(f"DataFrame shape: {df.shape}")
    print()

    # Check data quality
    print("Data Quality Check:")
    print("-"*80)
    print("Missing values:")
    print(df.isnull().sum())
    print()

    # Display data types
    print("Data types:")
    print(df.dtypes)
    print()

    # Save to CSV
    output_file = 'enforcement_actions.csv'
    df.to_csv(output_file, index=False)
    print(f"  Data saved to '{output_file}'")
    print("="*80)

    return df

except Exception as e:
    print(f"  Error occurred: {e}")
    import traceback
    traceback.print_exc()
    return None
```

```python
if __name__ == "__main__":
    df = main()
```

```
================================================================================
Step 1: Scraping HHS OIG Enforcement Actions
================================================================================
URL: https://oig.hhs.gov/fraud/enforcement/

DataFrame Head:
--------------------------------------------------------------------------------
                                               Title              Date  \
0  Delafield Man Sentenced to 18 Months' Imprison...  February 3, 2026
1  AG's Office Secures Indictments Against Peabod...  February 2, 2026
2  Florida Man Pleads Guilty to Conspiracy to Vio...  January 30, 2026
3  Yadkinville Woman Sentenced in Connection with...  January 29, 2026
4  Slidell Chiropractor Sentenced for Health Care...  January 28, 2026

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

                                                Link
0  https://oig.hhs.gov/fraud/enforcement/delafiel...
1  https://oig.hhs.gov/fraud/enforcement/ags-offi...
2  https://oig.hhs.gov/fraud/enforcement/florida-...
3  https://oig.hhs.gov/fraud/enforcement/yadkinvi...
4  https://oig.hhs.gov/fraud/enforcement/slidell-...

================================================================================
Total enforcement actions scraped: 20
DataFrame shape: (20, 4)

Data Quality Check:
--------------------------------------------------------------------------------
Missing values:
Title       0
Date        0
Category    0
```

```
Link        0
dtype: int64

Data types:
Title       str
Date        str
Category    str
Link        str
dtype: object

 Data saved to 'enforcement_actions.csv'
================================================================================
```

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code “ “ ” Step 2a: Pseudo-code for Dynamic Scraper Function

Function: scrape_enforcement_actions_from_date(year, month, run=False)

INPUTS: - year: integer (e.g., 2022, 2024) - month: integer (1-12) - run: boolean (default False) - indicator to control whether function actually runs

OUTPUTS: - CSV file: “enforcement_actions_{year}_{month}.csv” - Returns: pandas DataFrame

PSEUDO-CODE:

1. CHECK if run indicator is False:
    - If False: print message and return None (don’t run the scraper)
    - If True: proceed with scraping
2. VALIDATE input year:
    - If year $<$ 2013:
        - Print error message
        - Return None
    - Else: proceed
3. CREATE target start date from year and month
4. INITIALIZE:

- Empty list to store all enforcement actions data
- page_number = 1
- base_url = “https://oig.hhs.gov/fraud/enforcement/”
- found_target_date = False

5. LOOP STRUCTURE - Use WHILE loop (not simple FOR loop): Why WHILE loop?

   - We don’t know in advance how many pages we need to scrape
   - We need to keep going until we reach the target date
   - A WHILE loop allows us to continue until a condition is met

   WHILE NOT found_target_date:

   5.1. CONSTRUCT page URL: - If page_number == 1: url = base_url - Else: url = base_url + “?page=” + page_number

   5.2. FETCH page content: - Send GET request with headers - Parse HTML with BeautifulSoup

   5.3. EXTRACT all enforcement actions on current page: - Find all h2 tags (containing titles and links) - For each h2: a. Extract title and link b. Extract date c. Extract categories d. Append to temporary list

   5.4. CHECK dates on current page: - For each action on this page: - Parse the date - If date >= target_date: - Add to results list - If date < target_date: - Set found_target_date = True - BREAK out of inner loop

   5.5. CHECK if we should continue: - If found_target_date is True: - BREAK out of WHILE loop - If current page has no more actions: - BREAK out of WHILE loop

   5.6. INCREMENT page counter: - page_number = page_number + 1

   5.7. WAIT 1 second (respect server): - time.sleep(1)

6. CREATE DataFrame from collected data

7. SAVE to CSV:

   - Filename: f”enforcement_actions_{year}_{month}.csv”

8. PRINT summary statistics:

   - Total number of actions scraped
   - Date range
   - Earliest action details

9. RETURN DataFrame

LOOP TYPE DISCUSSION:

Why WHILE loop instead of FOR loop?

FOR loop limitations: - Requires knowing the range in advance (e.g., for i in range(10)) - Not suitable when we don't know how many iterations we need - Would require estimating or setting an arbitrary large number

WHILE loop advantages: - Continues until condition is met (found target date) - Can break early if target is reached - More flexible for unknown number of pages - Natural fit for web scraping with pagination

Alternative approaches considered: 1. FOR loop with break: for i in range(1000) - but 1000 is arbitrary 2. WHILE True with break: works but less explicit about condition 3. WHILE NOT condition: BEST - clearly expresses the stopping criterion " "

- b. Create Dynamic Scraper

```
"""
Step 2b: Dynamic Scraper for HHS OIG Enforcement Actions
Test: Collect enforcement actions since January 2024
Author: Fe
Date: February 2026
"""

import requests
from bs4 import BeautifulSoup
import pandas as pd
import re
import time
from datetime import datetime
import altair as alt

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


def parse_date_string(date_str):
    """
    Parse date string from "Month Day, Year" format to datetime object

    Args:
        date_str: String in format "December 10, 2025"
```

```
    Returns:
        datetime object or None if parsing fails
    """
    try:
        return datetime.strptime(date_str, "%B %d, %Y")
    except:
        return None


def scrape_single_page(url, headers):
    """
    Scrape a single page of enforcement actions

    Args:
        url: URL to scrape
        headers: HTTP headers for request

    Returns:
        List of dictionaries containing enforcement action data
    """
    response = requests.get(url, headers=headers)
    response.raise_for_status()

    soup = BeautifulSoup(response.content, 'html.parser')

    actions = []
    h2_tags = soup.find_all('h2')

    for h2 in h2_tags:
        parent = h2.find_parent('li')
        if not parent:
            continue

        # Extract title and link
        a_tag = h2.find('a')
        if not a_tag:
            continue

        title = a_tag.get_text(strip=True)
        link = a_tag.get('href', '')

        if link and not link.startswith('http'):
```

```
            link = 'https://oig.hhs.gov' + link

        # Extract date
        text_content = parent.get_text()
        date_pattern =
↪   r'(January|February|March|April|May|June|July|August|September|October|November|December)
        date_match = re.search(date_pattern, text_content)
        date_text = date_match.group(0) if date_match else "N/A"

        # Extract categories
        category_ul = parent.find('ul')
        if category_ul:
            category_items = category_ul.find_all('li')
            categories_list = [cat.get_text(strip=True) for cat in
↪   category_items]
            category = ", ".join(categories_list) if categories_list else
↪   "N/A"
        else:
            category = "N/A"

        actions.append({
            'Title': title,
            'Date': date_text,
            'Category': category,
            'Link': link
        })

    return actions


def scrape_enforcement_actions_from_date(year, month, run=False):
    """
    Scrape enforcement actions from a specific month/year to present day.

    Args:
        year: Integer year (must be >= 2013)
        month: Integer month (1-12)
        run: Boolean indicator - must be True to actually run the scraper

    Returns:
        pandas DataFrame with enforcement actions, or None if not run
    """
```

```
    # Check if we should actually run
    if not run:
        print("  Scraper not executed. Set run=True to execute.")
        print("   This prevents the scraper from running every time you knit
         ↪  the document.")
        return None

    # Validate year
    if year < 2013:
        print(f"  Error: Year must be >= 2013.")
        print(f"   Only enforcement actions from 2013 onwards are
         ↪  available.")
        print(f"   You entered: {year}")
        return None

    # Validate month
    if month < 1 or month > 12:
        print(f"  Error: Month must be between 1 and 12.")
        print(f"   You entered: {month}")
        return None

    # Create target start date
    target_date = datetime(year, month, 1)

    print("="*80)
    print(f"Starting dynamic scraper...")
    print(f"Target start date: {target_date.strftime('%B %Y')}")
    print(f"Scraping from: {target_date.strftime('%B %d, %Y')} to present")
    print("="*80)
    print()

    # Initialize
    all_actions = []
    page_number = 1
    base_url = "https://oig.hhs.gov/fraud/enforcement/"
    found_target_date = False

    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'
    }
```

```
# Use WHILE loop because we don't know how many pages we need
while not found_target_date:
    # Construct URL for current page
    if page_number == 1:
        url = base_url
    else:
        url = f"{base_url}?page={page_number}"

    print(f"  Scraping page {page_number}... ", end='', flush=True)

    try:
        # Scrape current page
        page_actions = scrape_single_page(url, headers)

        if not page_actions:
            print("(no more actions found)")
            break

        print(f"found {len(page_actions)} actions")

        # Check each action's date
        for action in page_actions:
            action_date = parse_date_string(action['Date'])

            if action_date is None:
                # If we can't parse the date, include it to be safe
                all_actions.append(action)
            elif action_date >= target_date:
                # Action is within our target range
                all_actions.append(action)
            else:
                # Action is before our target date - stop scraping
                found_target_date = True
                print(f"    Reached target date. Stopping at:
                 ↪  {action['Date']}")
                break

        # Check if we should continue to next page
        if found_target_date:
            break

        # Increment page counter
```

```
            page_number += 1

            # Wait 1 second to be respectful to the server
            time.sleep(1)

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

# Create DataFrame
df = pd.DataFrame(all_actions)

print()
print("="*80)
print("Scraping completed!")
print("="*80)
print(f"Total enforcement actions scraped: {len(df)}")
print(f"Pages scraped: {page_number}")

if len(df) > 0:
    # Add parsed date column for analysis
    df['Date_Parsed'] = df['Date'].apply(parse_date_string)
    df_sorted = df.sort_values('Date_Parsed')

    earliest = df_sorted.iloc[0]
    latest = df_sorted.iloc[-1]

    print(f"Date range: {earliest['Date']} to {latest['Date']}")
    print()
    print("Earliest enforcement action scraped:")
    print("-"*80)
    print(f"Date: {earliest['Date']}")
    print(f"Title: {earliest['Title']}")
    print(f"Category: {earliest['Category']}")
    print(f"Link: {earliest['Link']}")
    print()

    # Remove the helper column before saving
    df = df.drop('Date_Parsed', axis=1)

# Save to CSV
filename = f"enforcement_actions_{year}_{month}.csv"
df.to_csv(filename, index=False)
```

```
    print(f"  Data saved to '{filename}'")
    print("="*80)

    return df


if __name__ == "__main__":
    print("="*80)
    print("STEP 2b: Collecting enforcement actions since January 2024")
    print("="*80)
    print()

    # Run scraper for January 2024
    df_2024 = scrape_enforcement_actions_from_date(year=2024, month=1,
↪   run=True)

    if df_2024 is not None:
        print()
        print("="*80)
        print("RESULTS FOR STEP 2b")
        print("="*80)
        print(f"How many enforcement actions? {len(df_2024)}")
        print()
        print("DataFrame head:")
        print(df_2024.head())
        print()
        print("  Step 2b completed!")
```

```
================================================================================
STEP 2b: Collecting enforcement actions since January 2024
================================================================================

================================================================================
Starting dynamic scraper...
Target start date: January 2024
Scraping from: January 01, 2024 to present
================================================================================

  Scraping page 1... found 20 actions
  Scraping page 2... found 20 actions
  Scraping page 3... found 20 actions
  Scraping page 4... found 20 actions
  Scraping page 5... found 20 actions
```

```
Scraping page 6... found 20 actions
Scraping page 7... found 20 actions
Scraping page 8... found 20 actions
Scraping page 9... found 20 actions
Scraping page 10... found 20 actions
Scraping page 11... found 20 actions
Scraping page 12... found 20 actions
Scraping page 13... found 20 actions
Scraping page 14... found 20 actions
Scraping page 15... found 20 actions
Scraping page 16... found 20 actions
Scraping page 17... found 20 actions
Scraping page 18... found 20 actions
Scraping page 19... found 20 actions
Scraping page 20... found 20 actions
Scraping page 21... found 20 actions
Scraping page 22... found 20 actions
Scraping page 23... found 20 actions
Scraping page 24... found 20 actions
Scraping page 25... found 20 actions
Scraping page 26... found 20 actions
Scraping page 27... found 20 actions
Scraping page 28... found 20 actions
Scraping page 29... found 20 actions
Scraping page 30... found 20 actions
Scraping page 31... found 20 actions
Scraping page 32... found 20 actions
Scraping page 33... found 20 actions
Scraping page 34... found 20 actions
Scraping page 35... found 20 actions
Scraping page 36... found 20 actions
Scraping page 37... found 20 actions
Scraping page 38... found 20 actions
Scraping page 39... found 20 actions
Scraping page 40... found 20 actions
Scraping page 41... found 20 actions
Scraping page 42... found 20 actions
Scraping page 43... found 20 actions
Scraping page 44... found 20 actions
Scraping page 45... found 20 actions
Scraping page 46... found 20 actions
Scraping page 47... found 20 actions
Scraping page 48... found 20 actions
```

```
Scraping page 49... found 20 actions
Scraping page 50... found 20 actions
Scraping page 51... found 20 actions
Scraping page 52... found 20 actions
Scraping page 53... found 20 actions
Scraping page 54... found 20 actions
Scraping page 55... found 20 actions
Scraping page 56... found 20 actions
Scraping page 57... found 20 actions
Scraping page 58... found 20 actions
Scraping page 59... found 20 actions
Scraping page 60... found 20 actions
Scraping page 61... found 20 actions
Scraping page 62... found 20 actions
Scraping page 63... found 20 actions
Scraping page 64... found 20 actions
Scraping page 65... found 20 actions
Scraping page 66... found 20 actions
Scraping page 67... found 20 actions
Scraping page 68... found 20 actions
Scraping page 69... found 20 actions
Scraping page 70... found 20 actions
Scraping page 71... found 20 actions
Scraping page 72... found 20 actions
Scraping page 73... found 20 actions
Scraping page 74... found 20 actions
Scraping page 75... found 20 actions
Scraping page 76... found 20 actions
Scraping page 77... found 20 actions
Scraping page 78... found 20 actions
Scraping page 79... found 20 actions
Scraping page 80... found 20 actions
Scraping page 81... found 20 actions
Scraping page 82... found 20 actions
Scraping page 83... found 20 actions
Scraping page 84... found 20 actions
Scraping page 85... found 20 actions
Scraping page 86... found 20 actions
Scraping page 87... found 20 actions
Scraping page 88... found 20 actions
Scraping page 89... found 20 actions
   Reached target date. Stopping at: December 22, 2023
```

```
================================================================================
Scraping completed!
================================================================================
Total enforcement actions scraped: 1767
Pages scraped: 89
Date range: January 3, 2024 to February 3, 2026

Earliest enforcement action scraped:
--------------------------------------------------------------------------------
Date: January 3, 2024
Title: Former Nurse Aide Indicted In Death Of Clarksville Patient Arrested In
Georgia
Category: State Enforcement Agencies
Link:
https://oig.hhs.gov/fraud/enforcement/former-nurse-aide-indicted-in-death-of-clarksville-pati

 Data saved to 'enforcement_actions_2024_1.csv'
================================================================================

================================================================================
RESULTS FOR STEP 2b
================================================================================
How many enforcement actions? 1767

DataFrame head:
                                               Title              Date  \
0  Delafield Man Sentenced to 18 Months' Imprison...  February 3, 2026
1  AG's Office Secures Indictments Against Peabod...  February 2, 2026
2  Florida Man Pleads Guilty to Conspiracy to Vio...  January 30, 2026
3  Yadkinville Woman Sentenced in Connection with...  January 29, 2026
4  Slidell Chiropractor Sentenced for Health Care...  January 28, 2026

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

                                                Link
0  https://oig.hhs.gov/fraud/enforcement/delafiel...
1  https://oig.hhs.gov/fraud/enforcement/ags-offi...
2  https://oig.hhs.gov/fraud/enforcement/florida-...
```

```
3  https://oig.hhs.gov/fraud/enforcement/yadkinvi...
4  https://oig.hhs.gov/fraud/enforcement/slidell-...

 Step 2b completed!
```

- c. Test Your Code

```
"""
Step 2c: Dynamic Scraper for HHS OIG Enforcement Actions
Test: Collect enforcement actions since January 2022
Author: Fe
Date: February 2026

NOTE: This will take longer to run as it needs to scrape more pages (back to
 ↪  2022).
Use this dataframe for all subsequent questions.
"""

import requests
from bs4 import BeautifulSoup
import pandas as pd
import re
import time
from datetime import datetime
import altair as alt

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


def parse_date_string(date_str):
    """
    Parse date string from "Month Day, Year" format to datetime object

    Args:
        date_str: String in format "December 10, 2025"

    Returns:
        datetime object or None if parsing fails
    """
    try:
        return datetime.strptime(date_str, "%B %d, %Y")
```

```
    except:
        return None


def scrape_single_page(url, headers):
    """
    Scrape a single page of enforcement actions

    Args:
        url: URL to scrape
        headers: HTTP headers for request

    Returns:
        List of dictionaries containing enforcement action data
    """
    response = requests.get(url, headers=headers)
    response.raise_for_status()

    soup = BeautifulSoup(response.content, 'html.parser')

    actions = []
    h2_tags = soup.find_all('h2')

    for h2 in h2_tags:
        parent = h2.find_parent('li')
        if not parent:
            continue

        # Extract title and link
        a_tag = h2.find('a')
        if not a_tag:
            continue

        title = a_tag.get_text(strip=True)
        link = a_tag.get('href', '')

        if link and not link.startswith('http'):
            link = 'https://oig.hhs.gov' + link

        # Extract date
        text_content = parent.get_text()
        date_pattern =
↪   r'(January|February|March|April|May|June|July|August|September|October|November|December)
```

```
        date_match = re.search(date_pattern, text_content)
        date_text = date_match.group(0) if date_match else "N/A"

        # Extract categories
        category_ul = parent.find('ul')
        if category_ul:
            category_items = category_ul.find_all('li')
            categories_list = [cat.get_text(strip=True) for cat in
↪   category_items]
            category = ", ".join(categories_list) if categories_list else
↪   "N/A"
        else:
            category = "N/A"

        actions.append({
            'Title': title,
            'Date': date_text,
            'Category': category,
            'Link': link
        })

    return actions


def scrape_enforcement_actions_from_date(year, month, run=False):
    """
    Scrape enforcement actions from a specific month/year to present day.

    Args:
        year: Integer year (must be >= 2013)
        month: Integer month (1-12)
        run: Boolean indicator - must be True to actually run the scraper

    Returns:
        pandas DataFrame with enforcement actions, or None if not run
    """

    # Check if we should actually run
    if not run:
        print("  Scraper not executed. Set run=True to execute.")
        print("  This prevents the scraper from running every time you knit
↪   the document.")
        return None
```

```
    # Validate year
    if year < 2013:
        print(f"  Error: Year must be >= 2013.")
        print(f"   Only enforcement actions from 2013 onwards are
         ↪  available.")
        print(f"   You entered: {year}")
        return None

    # Validate month
    if month < 1 or month > 12:
        print(f"  Error: Month must be between 1 and 12.")
        print(f"   You entered: {month}")
        return None

    # Create target start date
    target_date = datetime(year, month, 1)

    print("="*80)
    print(f"Starting dynamic scraper...")
    print(f"Target start date: {target_date.strftime('%B %Y')}")
    print(f"Scraping from: {target_date.strftime('%B %d, %Y')} to present")
    print("="*80)
    print()

    # Initialize
    all_actions = []
    page_number = 1
    base_url = "https://oig.hhs.gov/fraud/enforcement/"
    found_target_date = False

    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'
    }

    # Use WHILE loop because we don't know how many pages we need
    while not found_target_date:
        # Construct URL for current page
        if page_number == 1:
            url = base_url
        else:
```

```
        url = f"{base_url}?page={page_number}"

    print(f"  Scraping page {page_number}... ", end='', flush=True)

    try:
        # Scrape current page
        page_actions = scrape_single_page(url, headers)

        if not page_actions:
            print("(no more actions found)")
            break

        print(f"found {len(page_actions)} actions")

        # Check each action's date
        for action in page_actions:
            action_date = parse_date_string(action['Date'])

            if action_date is None:
                # If we can't parse the date, include it to be safe
                all_actions.append(action)
            elif action_date >= target_date:
                # Action is within our target range
                all_actions.append(action)
            else:
                # Action is before our target date - stop scraping
                found_target_date = True
                print(f"    Reached target date. Stopping at:
                 ↪  {action['Date']}")
                break

        # Check if we should continue to next page
        if found_target_date:
            break

        # Increment page counter
        page_number += 1

        # Wait 1 second to be respectful to the server
        time.sleep(1)

    except Exception as e:
        print(f"\n  Error on page {page_number}: {e}")
```

```
            break

    # Create DataFrame
    df = pd.DataFrame(all_actions)

    print()
    print("="*80)
    print("Scraping completed!")
    print("="*80)
    print(f"Total enforcement actions scraped: {len(df)}")
    print(f"Pages scraped: {page_number}")

    if len(df) > 0:
        # Add parsed date column for analysis
        df['Date_Parsed'] = df['Date'].apply(parse_date_string)
        df_sorted = df.sort_values('Date_Parsed')

        earliest = df_sorted.iloc[0]
        latest = df_sorted.iloc[-1]

        print(f"Date range: {earliest['Date']} to {latest['Date']}")
        print()
        print("Earliest enforcement action scraped:")
        print("-"*80)
        print(f"Date: {earliest['Date']}")
        print(f"Title: {earliest['Title']}")
        print(f"Category: {earliest['Category']}")
        print(f"Link: {earliest['Link']}")
        print()

        # Remove the helper column before saving
        df = df.drop('Date_Parsed', axis=1)

    # Save to CSV
    filename = f"enforcement_actions_{year}_{month}.csv"
    df.to_csv(filename, index=False)
    print(f"  Data saved to '{filename}'")
    print("="*80)

    return df


if __name__ == "__main__":
```

```
    print("="*80)
    print("STEP 2c: Collecting enforcement actions since January 2022")
    print("="*80)
    print()
    print("   NOTE: This may take several minutes as it needs to scrape many
     ↪  pages.")
    print("    Please be patient...")
    print()

    # Run scraper for January 2022
    df_2022 = scrape_enforcement_actions_from_date(year=2022, month=1,
↪   run=True)

    if df_2022 is not None:
        print()
        print("="*80)
        print("RESULTS FOR STEP 2c")
        print("="*80)
        print(f"How many enforcement actions? {len(df_2022)}")
        print()
        print("DataFrame head:")
        print(df_2022.head())
        print()
        print("DataFrame tail:")
        print(df_2022.tail())
        print()
        print("  Step 2c completed!")
        print()
        print("  IMPORTANT: Use this dataframe
         ↪  (enforcement_actions_2022_1.csv)")
        print("   for all subsequent questions in the assignment.")
```

```
================================================================================
STEP 2c: Collecting enforcement actions since January 2022
================================================================================

   NOTE: This may take several minutes as it needs to scrape many pages.
     Please be patient...

================================================================================
Starting dynamic scraper...
Target start date: January 2022
Scraping from: January 01, 2022 to present
```

```
================================================================================

  Scraping page 1... found 20 actions
  Scraping page 2... found 20 actions
  Scraping page 3... found 20 actions
  Scraping page 4... found 20 actions
  Scraping page 5... found 20 actions
  Scraping page 6... found 20 actions
  Scraping page 7... found 20 actions
  Scraping page 8... found 20 actions
  Scraping page 9... found 20 actions
  Scraping page 10... found 20 actions
  Scraping page 11... found 20 actions
  Scraping page 12... found 20 actions
  Scraping page 13... found 20 actions
  Scraping page 14... found 20 actions
  Scraping page 15... found 20 actions
  Scraping page 16... found 20 actions
  Scraping page 17... found 20 actions
  Scraping page 18... found 20 actions
  Scraping page 19... found 20 actions
  Scraping page 20... found 20 actions
  Scraping page 21... found 20 actions
  Scraping page 22... found 20 actions
  Scraping page 23... found 20 actions
  Scraping page 24... found 20 actions
  Scraping page 25... found 20 actions
  Scraping page 26... found 20 actions
  Scraping page 27... found 20 actions
  Scraping page 28... found 20 actions
  Scraping page 29... found 20 actions
  Scraping page 30... found 20 actions
  Scraping page 31... found 20 actions
  Scraping page 32... found 20 actions
  Scraping page 33... found 20 actions
  Scraping page 34... found 20 actions
  Scraping page 35... found 20 actions
  Scraping page 36... found 20 actions
  Scraping page 37... found 20 actions
  Scraping page 38... found 20 actions
  Scraping page 39... found 20 actions
  Scraping page 40... found 20 actions
  Scraping page 41... found 20 actions
```

```
Scraping page 42... found 20 actions
Scraping page 43... found 20 actions
Scraping page 44... found 20 actions
Scraping page 45... found 20 actions
Scraping page 46... found 20 actions
Scraping page 47... found 20 actions
Scraping page 48... found 20 actions
Scraping page 49... found 20 actions
Scraping page 50... found 20 actions
Scraping page 51... found 20 actions
Scraping page 52... found 20 actions
Scraping page 53... found 20 actions
Scraping page 54... found 20 actions
Scraping page 55... found 20 actions
Scraping page 56... found 20 actions
Scraping page 57... found 20 actions
Scraping page 58... found 20 actions
Scraping page 59... found 20 actions
Scraping page 60... found 20 actions
Scraping page 61... found 20 actions
Scraping page 62... found 20 actions
Scraping page 63... found 20 actions
Scraping page 64... found 20 actions
Scraping page 65... found 20 actions
Scraping page 66... found 20 actions
Scraping page 67... found 20 actions
Scraping page 68... found 20 actions
Scraping page 69... found 20 actions
Scraping page 70... found 20 actions
Scraping page 71... found 20 actions
Scraping page 72... found 20 actions
Scraping page 73... found 20 actions
Scraping page 74... found 20 actions
Scraping page 75... found 20 actions
Scraping page 76... found 20 actions
Scraping page 77... found 20 actions
Scraping page 78... found 20 actions
Scraping page 79... found 20 actions
Scraping page 80... found 20 actions
Scraping page 81... found 20 actions
Scraping page 82... found 20 actions
Scraping page 83... found 20 actions
Scraping page 84... found 20 actions
```

```
Scraping page 85... found 20 actions
Scraping page 86... found 20 actions
Scraping page 87... found 20 actions
Scraping page 88... found 20 actions
Scraping page 89... found 20 actions
Scraping page 90... found 20 actions
Scraping page 91... found 20 actions
Scraping page 92... found 20 actions
Scraping page 93... found 20 actions
Scraping page 94... found 20 actions
Scraping page 95... found 20 actions
Scraping page 96... found 20 actions
Scraping page 97... found 20 actions
Scraping page 98... found 20 actions
Scraping page 99... found 20 actions
Scraping page 100... found 20 actions
Scraping page 101... found 20 actions
Scraping page 102... found 20 actions
Scraping page 103... found 20 actions
Scraping page 104... found 20 actions
Scraping page 105... found 20 actions
Scraping page 106... found 20 actions
Scraping page 107... found 20 actions
Scraping page 108... found 20 actions
Scraping page 109... found 20 actions
Scraping page 110... found 20 actions
Scraping page 111... found 20 actions
Scraping page 112... found 20 actions
Scraping page 113... found 20 actions
Scraping page 114... found 20 actions
Scraping page 115... found 20 actions
Scraping page 116... found 20 actions
Scraping page 117... found 20 actions
Scraping page 118... found 20 actions
Scraping page 119... found 20 actions
Scraping page 120... found 20 actions
Scraping page 121... found 20 actions
Scraping page 122... found 20 actions
Scraping page 123... found 20 actions
Scraping page 124... found 20 actions
Scraping page 125... found 20 actions
Scraping page 126... found 20 actions
Scraping page 127... found 20 actions
```

```
Scraping page 128... found 20 actions
Scraping page 129... found 20 actions
Scraping page 130... found 20 actions
Scraping page 131... found 20 actions
Scraping page 132... found 20 actions
Scraping page 133... found 20 actions
Scraping page 134... found 20 actions
Scraping page 135... found 20 actions
Scraping page 136... found 20 actions
Scraping page 137... found 20 actions
Scraping page 138... found 20 actions
Scraping page 139... found 20 actions
Scraping page 140... found 20 actions
Scraping page 141... found 20 actions
Scraping page 142... found 20 actions
Scraping page 143... found 20 actions
Scraping page 144... found 20 actions
Scraping page 145... found 20 actions
Scraping page 146... found 20 actions
Scraping page 147... found 20 actions
Scraping page 148... found 20 actions
Scraping page 149... found 20 actions
Scraping page 150... found 20 actions
Scraping page 151... found 20 actions
Scraping page 152... found 20 actions
Scraping page 153... found 20 actions
Scraping page 154... found 20 actions
Scraping page 155... found 20 actions
Scraping page 156... found 20 actions
Scraping page 157... found 20 actions
Scraping page 158... found 20 actions
Scraping page 159... found 20 actions
Scraping page 160... found 20 actions
Scraping page 161... found 20 actions
Scraping page 162... found 20 actions
Scraping page 163... found 20 actions
Scraping page 164... found 20 actions
Scraping page 165... found 20 actions
Scraping page 166... found 20 actions
Scraping page 167... found 20 actions
Scraping page 168... found 20 actions
   Reached target date. Stopping at: December 30, 2021
```

```
================================================================================
Scraping completed!
================================================================================
Total enforcement actions scraped: 3357
Pages scraped: 168
Date range: January 4, 2022 to February 3, 2026

Earliest enforcement action scraped:
--------------------------------------------------------------------------------
Date: January 4, 2022
Title: Integrated Pain Management Medical Group Agreed to Pay $10,000 for
Allegedly Violating the Civil Monetary Penalties Law by Employing Excluded
Individuals
Category: Fraud Self-Disclosures
Link:
https://oig.hhs.gov/fraud/enforcement/integrated-pain-management-medical-group-agreed-to-pay

  Data saved to 'enforcement_actions_2022_1.csv'
================================================================================

================================================================================
RESULTS FOR STEP 2c
================================================================================
How many enforcement actions? 3357

DataFrame head:
                                               Title              Date  \
0  Delafield Man Sentenced to 18 Months' Imprison...  February 3, 2026
1  AG's Office Secures Indictments Against Peabod...  February 2, 2026
2  Florida Man Pleads Guilty to Conspiracy to Vio...  January 30, 2026
3  Yadkinville Woman Sentenced in Connection with...  January 29, 2026
4  Slidell Chiropractor Sentenced for Health Care...  January 28, 2026

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

                                                Link
0  https://oig.hhs.gov/fraud/enforcement/delafiel...
1  https://oig.hhs.gov/fraud/enforcement/ags-offi...
```

```
2  https://oig.hhs.gov/fraud/enforcement/florida-...
3  https://oig.hhs.gov/fraud/enforcement/yadkinvi...
4  https://oig.hhs.gov/fraud/enforcement/slidell-...

DataFrame tail:
                                                  Title             Date  \
3352  North Carolina Physician Indicted for Adultera...  January 5, 2022
3353  Attorney General Alan Wilson announces arrest ...  January 5, 2022
3354  Central Medical Systems, LLC, Alan Trent Harle...  January 4, 2022
3355  Ohio home healthcare provider agrees to pay $5...  January 4, 2022
3356  Integrated Pain Management Medical Group Agree...  January 4, 2022

                       Category  \
3352  Criminal and Civil Actions
3353  State Enforcement Agencies
3354  Criminal and Civil Actions
3355  Criminal and Civil Actions
3356      Fraud Self-Disclosures

                                                  Link
3352  https://oig.hhs.gov/fraud/enforcement/north-ca...
3353  https://oig.hhs.gov/fraud/enforcement/attorney...
3354  https://oig.hhs.gov/fraud/enforcement/central-...
3355  https://oig.hhs.gov/fraud/enforcement/ohio-hom...
3356  https://oig.hhs.gov/fraud/enforcement/integrat...

 Step 2c completed!

 IMPORTANT: Use this dataframe (enforcement_actions_2022_1.csv)
  for all subsequent questions in the assignment.
```

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

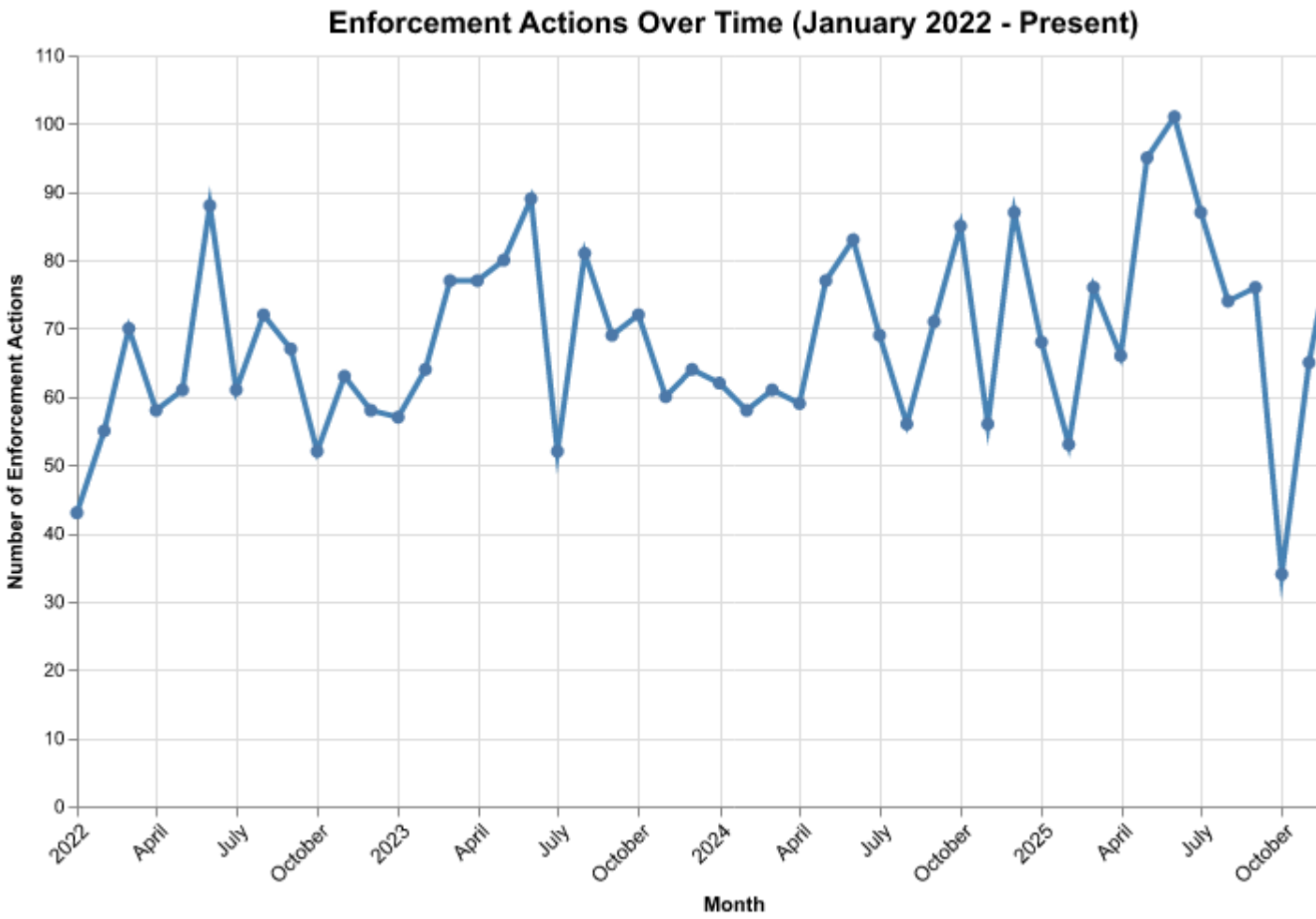
```
"""
Step 3.1: Overall Trend - Line chart of enforcement actions over time
Author: Fe
Date: February 2026

Plot: Number of enforcement actions aggregated by month+year since January
 ↪  2022
```

```
"""

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

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


def prepare_data(csv_file):
    """
    Load and prepare data for plotting
    """
    df = pd.read_csv(csv_file)

    # Parse dates
    df['Date_Parsed'] = pd.to_datetime(df['Date'], format='%B %d, %Y',
    errors='coerce')

    # Create Year-Month column for aggregation
    df['Year_Month'] = df['Date_Parsed'].dt.to_period('M')
    df['Year_Month_Str'] = df['Year_Month'].astype(str)

    return df


def plot_overall_trend(df):
    """
    Create line chart showing number of enforcement actions over time
    (monthly)
    """
    # Aggregate by month
    monthly_counts =
    df.groupby('Year_Month_Str').size().reset_index(name='Count')

    # Create line chart
    chart = alt.Chart(monthly_counts).mark_line(
        point=alt.OverlayMarkDef(filled=True, size=50),
        strokeWidth=3,
        color='steelblue'
    ).encode(
```

```
        x=alt.X('Year_Month_Str:T',
                title='Month',
                axis=alt.Axis(labelAngle=-45, labelFontSize=11)),
        y=alt.Y('Count:Q',
                title='Number of Enforcement Actions',
                scale=alt.Scale(zero=False)),
        tooltip=[
            alt.Tooltip('Year_Month_Str:T', title='Month'),
            alt.Tooltip('Count:Q', title='Count')
        ]
    ).properties(
        title={
            "text": 'Enforcement Actions Over Time (January 2022 - Present)',
            "fontSize": 16,
            "fontWeight": 'bold'
        },
        width=700,
        height=400
    )

    return chart


def main():
    """
    Main function
    """
    print("="*80)
    print("STEP 3.1: Overall Trend Visualization")
    print("="*80)
    print()

    csv_file = 'enforcement_actions_2022_1.csv'

    try:
        # Load data
        print("Loading data...")
        df = prepare_data(csv_file)
        print(f"  Loaded {len(df)} enforcement actions")
        print()

        # Create chart
        print("Creating visualization...")
```

```
        chart = plot_overall_trend(df)

        # Save as PNG (for PDF export)
        chart.save('plot1_overall_trend.png', scale_factor=2.0)
        print("  Saved as 'plot1_overall_trend.png'")

        # Display in VSCode (interactive)
        chart.show()
        print("  Displayed in VSCode")

        print()
        print("="*80)
        print("Visualization completed!")
        print("="*80)

        return chart

    except FileNotFoundError:
        print(f"  Error: Could not find '{csv_file}'")
        print("   Make sure you have run Step 2c to create this file.")
        return None
    except Exception as e:
        print(f"  Error: {e}")
        import traceback
        traceback.print_exc()
        return None


if __name__ == "__main__":
    chart = main()
```

```
================================================================================
STEP 3.1: Overall Trend Visualization
================================================================================

Loading data...
  Loaded 3357 enforcement actions

Creating visualization...
  Saved as 'plot1_overall_trend.png'
```




```
  Displayed in VSCode

================================================================================
Visualization completed!
================================================================================
```

2. Plot the number of enforcement actions categorized:

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

```
"""
Step 3.2: By Category - Line chart split by main categories
Author: Fe
Date: February 2026
```

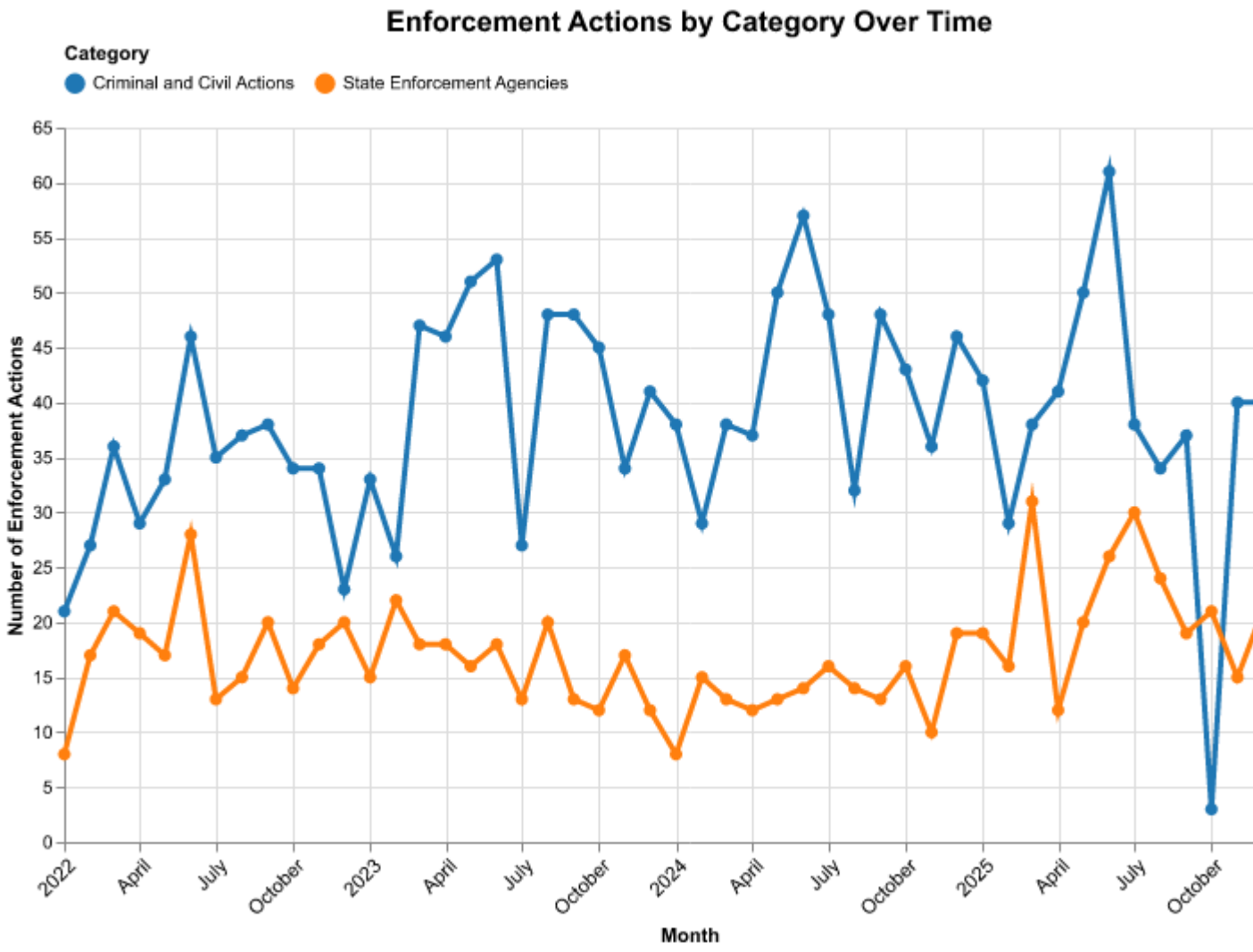
```
Plot: Number of enforcement actions split by:
- "Criminal and Civil Actions"
- "State Enforcement Agencies"
"""

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

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


def prepare_data(csv_file):
    """
    Load and prepare data for plotting
    """
    df = pd.read_csv(csv_file)

    # Parse dates
    df['Date_Parsed'] = pd.to_datetime(df['Date'], format='%B %d, %Y',
 ↪  errors='coerce')

    # Create Year-Month column for aggregation
    df['Year_Month'] = df['Date_Parsed'].dt.to_period('M')
    df['Year_Month_Str'] = df['Year_Month'].astype(str)

    # Create simplified category column
    # Check if "Criminal and Civil Actions" or "State Enforcement Agencies"
     ↪  is in Category
    df['Category_Simple'] = df['Category'].apply(
        lambda x: 'Criminal and Civil Actions' if 'Criminal and Civil
         ↪  Actions' in str(x)
        else ('State Enforcement Agencies' if 'State Enforcement Agencies' in
         ↪  str(x)
        else 'Other')
    )

    return df
```

```
def plot_by_category(df):
    """
    Create line chart showing enforcement actions by main category
    """
    # Aggregate by month and category
    category_monthly = df.groupby(['Year_Month_Str',
↪   'Category_Simple']).size().reset_index(name='Count')

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

    # Create line chart
    chart = alt.Chart(category_monthly).mark_line(
        point=alt.OverlayMarkDef(filled=True, size=50),
        strokeWidth=3
    ).encode(
        x=alt.X('Year_Month_Str:T',
                title='Month',
                axis=alt.Axis(labelAngle=-45, labelFontSize=11)),
        y=alt.Y('Count:Q',
                title='Number of Enforcement Actions',
                scale=alt.Scale(zero=False)),
        color=alt.Color('Category_Simple:N',
                        title='Category',
                        scale=alt.Scale(
                            domain=['Criminal and Civil Actions', 'State
↪   Enforcement Agencies'],
                            range=['#1f77b4', '#ff7f0e']
                        ),
                        legend=alt.Legend(orient='top')),
        tooltip=[
            alt.Tooltip('Year_Month_Str:T', title='Month'),
            alt.Tooltip('Category_Simple:N', title='Category'),
            alt.Tooltip('Count:Q', title='Count')
        ]
    ).properties(
        title={
            "text": 'Enforcement Actions by Category Over Time',
            "fontSize": 16,
```

```
            "fontWeight": 'bold'
        },
        width=700,
        height=400
    )

    return chart


def main():
    """
    Main function
    """
    print("="*80)
    print("STEP 3.2: By Category Visualization")
    print("="*80)
    print()

    csv_file = 'enforcement_actions_2022_1.csv'

    try:
        # Load data
        print("Loading data...")
        df = prepare_data(csv_file)
        print(f"  Loaded {len(df)} enforcement actions")
        print()

        # Show category distribution
        print("Category distribution:")
        print(df['Category_Simple'].value_counts())
        print()

        # Create chart
        print("Creating visualization...")
        chart = plot_by_category(df)

        # Save as PNG (for PDF export)
        chart.save('plot2_by_category.png', scale_factor=2.0)
        print("  Saved as 'plot2_by_category.png'")

        # Display in VSCode (interactive)
        chart.show()
        print("  Displayed in VSCode")
```

```
        print()
        print("="*80)
        print("Visualization completed!")
        print("="*80)

        return chart

    except FileNotFoundError:
        print(f"  Error: Could not find '{csv_file}'")
        print("   Make sure you have run Step 2c to create this file.")
        return None
    except Exception as e:
        print(f"  Error: {e}")
        import traceback
        traceback.print_exc()
        return None


if __name__ == "__main__":
    chart = main()
```

```
================================================================================
STEP 3.2: By Category Visualization
================================================================================

Loading data...
  Loaded 3357 enforcement actions

Category distribution:
Category_Simple
Criminal and Civil Actions    1888
State Enforcement Agencies     843
Other                          626
Name: count, dtype: int64

Creating visualization...
  Saved as 'plot2_by_category.png'
```




```
Displayed in VSCode

================================================================================
Visualization completed!
================================================================================
```

- based on five topics

```
"""
Step 3.3: By Topic - Line chart split by five topics
Author: Fe
Date: February 2026
```

```
Plot: Number of enforcement actions in "Criminal and Civil Actions" category
 ↪  split by:
- Health Care Fraud
- Financial Fraud
- Drug Enforcement
- Bribery/Corruption
- Other
"""

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

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


def classify_action_topic(title, category):
    """
    Classify an enforcement action into one of five topics based on its
 ↪  title.

    Topics:
    - Health Care Fraud
    - Financial Fraud
    - Drug Enforcement
    - Bribery/Corruption
    - Other

    Args:
        title: String title of the enforcement action
        category: Category field from the data

    Returns:
        String topic name
    """
    title_lower = title.lower()

    # Health Care Fraud keywords
    health_keywords = [
```

```
        'health care', 'healthcare', 'medicare', 'medicaid', 'medical',
        'hospital', 'physician', 'doctor', 'nurse', 'clinic', 'patient',
        'prescription', 'billing', 'diagnosis', 'treatment', 'therapy',
        'pharmacy', 'diagnostic', 'laboratory', 'home health', 'durable
        ↪  medical',
        'nursing home', 'assisted living', 'rehab'
    ]

    # Financial Fraud keywords
    financial_keywords = [
        'bank', 'financial', 'money laundering', 'wire fraud', 'securities',
        'investment', 'loan', 'mortgage', 'tax', 'irs', 'treasury',
        'embezzle', 'ponzi', 'scheme', 'credit card', 'identity theft',
        'fraud scheme', 'false claims', 'mail fraud'
    ]

    # Drug Enforcement keywords
    drug_keywords = [
        'drug', 'opioid', 'fentanyl', 'cocaine', 'heroin', 'methamphetamine',
        'controlled substance', 'narcotic', 'prescription fraud', 'pill',
        'oxycodone', 'hydrocodone', 'pharmaceutical', 'distribute drugs',
        'drug distribution', 'unlawful distribution'
    ]

    # Bribery/Corruption keywords
    bribery_keywords = [
        'bribery', 'bribe', 'corruption', 'corrupt', 'kickback', 'pay to
        ↪  play',
        'illegal gratuity', 'conflict of interest', 'self-dealing',
        ↪  'extortion'
    ]

    # Check each category (order matters - more specific first)
    # Drug Enforcement
    if any(keyword in title_lower for keyword in drug_keywords):
        return "Drug Enforcement"

    # Bribery/Corruption
    if any(keyword in title_lower for keyword in bribery_keywords):
        return "Bribery/Corruption"

    # Financial Fraud
    if any(keyword in title_lower for keyword in financial_keywords):
```

```
        return "Financial Fraud"

    # Health Care Fraud (most common, check last among specific categories)
    if any(keyword in title_lower for keyword in health_keywords):
        return "Health Care Fraud"

    # Default to Other
    return "Other"


def prepare_data(csv_file):
    """
    Load and prepare data for plotting
    """
    df = pd.read_csv(csv_file)

    # Parse dates
    df['Date_Parsed'] = pd.to_datetime(df['Date'], format='%B %d, %Y',
↪   errors='coerce')

    # Create Year-Month column for aggregation
    df['Year_Month'] = df['Date_Parsed'].dt.to_period('M')
    df['Year_Month_Str'] = df['Year_Month'].astype(str)

    # Create simplified category column
    df['Category_Simple'] = df['Category'].apply(
        lambda x: 'Criminal and Civil Actions' if 'Criminal and Civil
        ↪  Actions' in str(x)
        else ('State Enforcement Agencies' if 'State Enforcement Agencies' in
        ↪  str(x)
        else 'Other')
    )

    # Classify topics for Criminal and Civil Actions only
    df['Topic'] = df.apply(
        lambda row: classify_action_topic(row['Title'], row['Category'])
        if row['Category_Simple'] == 'Criminal and Civil Actions'
        else 'N/A',
        axis=1
    )

    return df
```

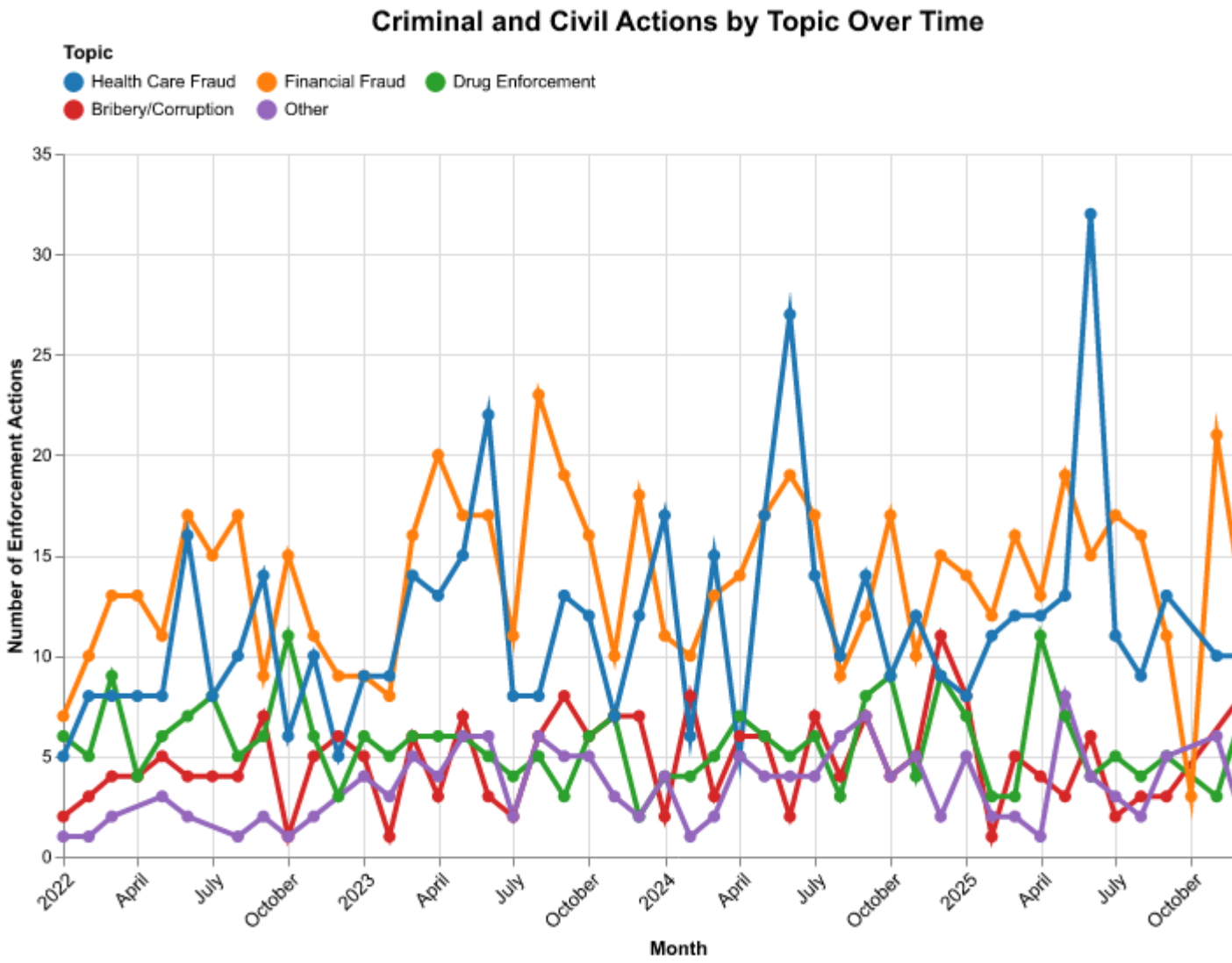
```
def plot_by_topic(df):
    """
    Create line chart showing Criminal and Civil Actions by topic
    """
    # Filter to only Criminal and Civil Actions
    criminal_df = df[df['Category_Simple'] == 'Criminal and Civil Actions']

    # Aggregate by month and topic
    topic_monthly = criminal_df.groupby(['Year_Month_Str',
↪   'Topic']).size().reset_index(name='Count')

    # Create line chart
    chart = alt.Chart(topic_monthly).mark_line(
        point=alt.OverlayMarkDef(filled=True, size=50),
        strokeWidth=3
    ).encode(
        x=alt.X('Year_Month_Str:T',
                title='Month',
                axis=alt.Axis(labelAngle=-45, labelFontSize=11)),
        y=alt.Y('Count:Q',
                title='Number of Enforcement Actions',
                scale=alt.Scale(zero=False)),
        color=alt.Color('Topic:N',
                        title='Topic',
                        scale=alt.Scale(
                            domain=['Health Care Fraud', 'Financial Fraud',
↪   'Drug Enforcement',
                                    'Bribery/Corruption', 'Other'],
                            range=['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728',
                                ↪   '#9467bd']
                        ),
                        legend=alt.Legend(orient='top', columns=3)),
        tooltip=[
            alt.Tooltip('Year_Month_Str:T', title='Month'),
            alt.Tooltip('Topic:N', title='Topic'),
            alt.Tooltip('Count:Q', title='Count')
        ]
    ).properties(
        title={
            "text": 'Criminal and Civil Actions by Topic Over Time',
            "fontSize": 16,
            "fontWeight": 'bold'
```

```
        },
        width=700,
        height=400
    )

    return chart


def main():
    """
    Main function
    """
    print("="*80)
    print("STEP 3.3: By Topic Visualization")
    print("="*80)
    print()

    csv_file = 'enforcement_actions_2022_1.csv'

    try:
        # Load data
        print("Loading data...")
        df = prepare_data(csv_file)
        print(f"  Loaded {len(df)} enforcement actions")
        print()

        # Show topic distribution
        criminal_df = df[df['Category_Simple'] == 'Criminal and Civil
↪   Actions']
        print("Topic distribution (Criminal and Civil Actions only):")
        print(criminal_df['Topic'].value_counts())
        print()

        # Show some examples
        print("Sample classifications:")
        print("-"*80)
        for topic in ['Health Care Fraud', 'Financial Fraud', 'Drug
         ↪  Enforcement',
                      'Bribery/Corruption', 'Other']:
            examples = criminal_df[criminal_df['Topic'] ==
↪   topic]['Title'].head(2)
            if len(examples) > 0:
                print(f"\n{topic}:")
```

```
                for title in examples:
                    print(f"  - {title}")
        print()

        # Create chart
        print("Creating visualization...")
        chart = plot_by_topic(df)

        # Save as PNG (for PDF export)
        chart.save('plot3_by_topic.png', scale_factor=2.0)
        print("  Saved as 'plot3_by_topic.png'")

        # Display in VSCode (interactive)
        chart.show()
        print("  Displayed in VSCode")

        print()
        print("="*80)
        print("Visualization completed!")
        print("="*80)

        return chart

    except FileNotFoundError:
        print(f"  Error: Could not find '{csv_file}'")
        print("   Make sure you have run Step 2c to create this file.")
        return None
    except Exception as e:
        print(f"  Error: {e}")
        import traceback
        traceback.print_exc()
        return None


if __name__ == "__main__":
    chart = main()
```

```
================================================================================
STEP 3.3: By Topic Visualization
================================================================================

Loading data...
  Loaded 3357 enforcement actions
```

```
Topic distribution (Criminal and Civil Actions only):
Topic
Financial Fraud       680
Health Care Fraud     554
Drug Enforcement      271
Bribery/Corruption    224
Other                 159
Name: count, dtype: int64

Sample classifications:
--------------------------------------------------------------------------------

Health Care Fraud:
  - Slidell Chiropractor Sentenced for Health Care Fraud
  - Repeat Health Care Fraud Offender Sentenced for Defrauding New Hampshire
  Medicaid

Financial Fraud:
  - Yadkinville Woman Sentenced in Connection with Multi-Million Dollar
  Medicaid Fraud Scheme
  - Scranton Heart Institute Agrees To Pay $48,709.20 To Settle False Claims
  Act Allegations

Drug Enforcement:
  - Rheumatologist Agrees To Resolve False Claims Act Allegations Related To
  Unapproved Drugs
  - Cheshire Nurse Admits Illegally Distributing Controlled Substances

Bribery/Corruption:
  - Delafield Man Sentenced to 18 Months' Imprisonment for Conspiracy to Pay
  Health Care Kickbacks
  - Florida Man Pleads Guilty to Conspiracy to Violate the Anti-Kickback
  Statute

Other:
  - Memphis Woman Sentenced to 41 Months in Federal Prison for Defrauding
  COVID-19 Relief Programs of Over $560,000
  - Honduran Man and Kansas Woman Indicted for Fraudulently Obtaining Custody
  of an Unaccompanied Alien Child

Creating visualization...
  Saved as 'plot3_by_topic.png'
```



Displayed in VSCode

```
================================================================================
Visualization completed!
================================================================================
```