PS4

jiayu zhao

Invalid Date

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: JIAYUZHAO

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.

```python
# Test imports - check if all libraries are available
print("Testing library imports...")

try:
    import pandas as pd

    print("pandas imported successfully")
except ImportError as e:
    print(f"pandas import failed: {e}")

try:
    import altair as alt

    print("altair imported successfully")
except ImportError as e:
    print(f"altair import failed: {e}")

try:
    import time

    print("time imported successfully")
except ImportError as e:
    print(f"time import failed: {e}")

try:
    import warnings

    warnings.filterwarnings("ignore")
    alt.renderers.enable("png")
    print("Configuration successful")
except Exception as e:
    print(f"Configuration failed: {e}")

print("\nAll imports complete!")
```

```
Testing library imports...
pandas imported successfully
altair imported successfully
time imported successfully
Configuration successful

All imports complete!
```

Step 1: Develop initial scraper and crawler

```
# Import necessary libraries
import requests
from bs4 import BeautifulSoup
import pandas as pd
import time
import re
from datetime import datetime

# Set target URL
url = "https://oig.hhs.gov/fraud/enforcement/"

# Send HTTP request
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
    ↪  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124
    ↪  Safari/537.36'
}

response = requests.get(url, headers=headers)
print(f"Status code: {response.status_code}")

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

# Find enforcement action entries using h2 tags (which contain titles)
enforcement_actions = []

# Look for h2 headings and use their parent containers
h2_tags = soup.find_all('h2')
articles = [h2.parent for h2 in h2_tags if h2.parent]

print(f"Found {len(articles)} enforcement action entries\n")

# Extract information for each enforcement action
for article in articles:
    # Extract title and link
    heading = article.find(['h2', 'h3'])
    if heading:
        link_element = heading.find('a')
        if link_element:
```

```
            title = link_element.get_text(strip=True)
            link = link_element.get('href')
            if link and not link.startswith('http'):
                link = 'https://oig.hhs.gov' + link
        else:
            title = heading.get_text(strip=True)
            link = None
    else:
        continue

    # Extract date
    date_element = article.find('time')
    if date_element:
        date = date_element.get_text(strip=True)
    else:
        # Try pattern matching
        text = article.get_text()
        date_pattern =
↪   r'(January|February|March|April|May|June|July|August|September|October|November|December)
        match = re.search(date_pattern, text)
        date = match.group(0) if match else None

    # Extract categories
    categories = []
    category_elements = article.find_all('li')
    for cat in category_elements:
        cat_text = cat.get_text(strip=True)
        if cat_text:
            categories.append(cat_text)
    category = ', '.join(categories) if categories else None

    # Add to list
    enforcement_actions.append({
        'title': title,
        'date': date,
        'category': category,
        'link': link
    })

# Create DataFrame
df = pd.DataFrame(enforcement_actions)

print(f"Successfully extracted {len(df)} records")
```

```
print("\nFirst few rows:")
print(df.head())
```

```
Status code: 200
Found 24 enforcement action entries

Successfully extracted 24 records

First few rows:
                                               title              date  \
0                                  In This Section              None
1                                         Archives              None
2  Houston Transplant Doctor Indicted For Making ...  February 5, 2026
3  MultiCare Health System to Pay Millions to Set...  February 4, 2026
4  Brooklyn Banker Pleads Guilty to Laundering Pr...  February 3, 2026

                                            category  \
0  About Enforcement Actions, Civil Monetary Pena...
1                                               None
2                         Criminal and Civil Actions
3                         Criminal and Civil Actions
4                                           COVID-19

                                                link
0                                               None
1                                               None
2  https://oig.hhs.gov/fraud/enforcement/houston-...
3  https://oig.hhs.gov/fraud/enforcement/multicar...
4  https://oig.hhs.gov/fraud/enforcement/brooklyn...
```

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

a. Pseudo-Code

Function: `scrape_enforcement_actions(month, year, run=False)`

Steps:

1. Input validation:
   - Check if year >= 2013 (data only available from 2013 onwards)
   - If year < 2013, print warning and return None

- If run=False, print message and return None without scraping

2. Initialize variables:
    - Create empty list for enforcement actions
    - Set base URL and headers
    - Calculate target start date from month/year
3. Loop through pages (while loop):
    - Why while loop? We don't know how many pages exist beforehand. We need to keep scraping until we reach the target date or run out of pages. A while loop allows us to check conditions on each iteration and stop dynamically.
    - Start with page number 0
    - For each page:
        - Construct URL (add ?page=N for pages after first)
        - Make HTTP request
        - Add 1 second delay (time.sleep(1)) to avoid server blocking
        - Parse HTML and find all article containers (using h2 parent elements)
        - For each article:
            - Extract title, date, category, and link
            - Parse date and compare with target date
            - If date >= target date, add to list
            - If date < target date, set flag to stop
        - Check if more pages exist
        - If no more pages or reached target date, exit loop
        - Otherwise, increment page number
4. Process and save results:
    - Convert list to DataFrame
    - Save to CSV: "enforcement_actions_{year}_{month}.csv"
    - Print summary statistics
    - Return DataFrame

b. Create Dynamic Scraper

```
def scrape_enforcement_actions(month, year, run=False):
    """
    Scrape HHS OIG enforcement actions from a given month/year to today.

    Parameters:
    month : int (1-12)
    year : int (must be >= 2013)
    run : bool (default: False)
```

```
    Returns:
    pandas.DataFrame or None
    """

    # Check if function should run
    if not run:
        print("Function not executed. Set run=True to scrape data.")
        return None

    # Validate year
    if year < 2013:
        print(f"Error: Year must be >= 2013. You entered: {year}")
        return None

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

    print(f"Starting to scrape from {month}/{year} to today...")

    # Initialize
    target_date = datetime(year, month, 1)
    all_actions = []
    base_url = "https://oig.hhs.gov/fraud/enforcement/"
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
        ↪  AppleWebKit/537.36"
    }

    page_num = 0
    should_continue = True

    # While loop for pagination
    while should_continue:
        # Construct URL
        url = base_url if page_num == 0 else f"{base_url}?page={page_num}"
        print(f"Scraping page {page_num + 1}...")

        # Make request
        try:
            response = requests.get(url, headers=headers)
```

```
            response.raise_for_status()
        except:
            print(f"Error fetching page {page_num + 1}")
            break

        # Parse HTML
        soup = BeautifulSoup(response.content, "html.parser")
        h2_tags = soup.find_all("h2")
        items = [h2.parent for h2 in h2_tags if h2.parent]

        if not items:
            print("No more items found.")
            break

        print(f"  Found {len(items)} items")

        # Extract data
        for item in items:
            # Title and link
            heading = item.find(["h2", "h3"])
            if not heading:
                continue

            link_element = heading.find("a")
            if link_element:
                title = link_element.get_text(strip=True)
                link = link_element.get("href")
                if link and not link.startswith("http"):
                    link = "https://oig.hhs.gov" + link
            else:
                continue

            # Date
            date_element = item.find("time")
            if date_element:
                date_str = date_element.get_text(strip=True)
            else:
                text = item.get_text()
                match = re.search(

                    ↪  r"(January|February|March|April|May|June|July|August|September|Octobe
                    text,
                )
```

```
            date_str = match.group(0) if match else None

        # Parse date for comparison
        if date_str:
            try:
                article_date = datetime.strptime(date_str, "%B %d, %Y")
            except:
                article_date = datetime.now()
        else:
            article_date = datetime.now()

        # Check date range
        if article_date >= target_date:
            # Categories
            categories = []
            for cat in item.find_all("li"):
                cat_text = cat.get_text(strip=True)
                if cat_text:
                    categories.append(cat_text)
            category = ", ".join(categories) if categories else None

            all_actions.append(
                {
                    "title": title,
                    "date": date_str,
                    "category": category,
                    "link": link,
                }
            )
        else:
            print(f"  Reached target date. Stopping.")
            should_continue = False
            break

    # Check for next page (simplified: stop after 100 pages max)
    page_num += 1
    if page_num >= 100:
        should_continue = False

    # Delay between requests
    if should_continue:
        time.sleep(1)
```

```
    # Create DataFrame and save
    if all_actions:
        df = pd.DataFrame(all_actions)
        filename = f"enforcement_actions_{year}_{month}.csv"
        df.to_csv(filename, index=False)

        print(f"\nTotal actions collected: {len(df)}")
        print(f"Saved to: {filename}")
        return df
    else:
        print("No data collected.")
        return None
```

c. Test Your Code

```
# Set run=False to skip scraping during rendering
# Only set run=True when you actually want to scrape data

# Test: Scrape from January 2022 to today
print("Starting scraping test...")
df_2022 = scrape_enforcement_actions(month=1, year=2022, run=False)

# If scraping was skipped, try to load existing CSV
if df_2022 is None:
    print("\nScraping was skipped (run=False)")
    print("Attempting to load existing CSV file...")
    try:
        df_2022 = pd.read_csv("enforcement_actions_2022_1.csv")
        print(f"Successfully loaded existing CSV with {len(df_2022)}
        ↪  records")
    except FileNotFoundError:
        print("CSV file not found.")
        print("\nTo create the data file:")
        print("1. Change run=True in the code above")
        print("2. Run this cell (it will take several minutes)")
        print("3. After data is collected, change run=False")
        print("4. The CSV file will be used for subsequent analysis")

if df_2022 is not None and len(df_2022) > 0:
    print("RESULTS:")
    print(f"Total enforcement actions: {len(df_2022)}")
```

```
    # Earliest action
    print(f"\nEarliest Enforcement Action:")
    earliest = df_2022.iloc[-1]
    print(f"Date: {earliest['date']}")
    print(f"Title: {earliest['title']}")
    print(f"Category: {earliest['category']}")
    print(f"Link: {earliest['link']}")

    # Most recent action
    print(f"\nMost Recent Enforcement Action:")
    most_recent = df_2022.iloc[0]
    print(f"Date: {most_recent['date']}")
    print(f"Title: {most_recent['title']}")
else:
    print("\nNo data available. Cannot proceed with analysis.")
```

```
Starting scraping test...
Function not executed. Set run=True to scrape data.

Scraping was skipped (run=False)
Attempting to load existing CSV file...
Successfully loaded existing CSV with 2000 records
RESULTS:
Total enforcement actions: 2000

Earliest Enforcement Action:
Date: October 2, 2023
Title: South Carolina Physician & Nephrology Practice Agree to Pay Over
$585,000 to Settle Laboratory Kickback Allegations
Category: Criminal and Civil Actions
Link:
https://oig.hhs.gov/fraud/enforcement/south-carolina-physician-nephrology-practice-agree-to-p

Most Recent Enforcement Action:
Date: February 5, 2026
Title: Houston Transplant Doctor Indicted For Making False Statements In
Patients' Medical Records
```

Alternative: Create sample data for testing

If scraping fails repeatedly, you can create a sample CSV file manually for testing:

```
# Only run this if scraping completely fails
# This creates a small sample dataset for testing visualization code

import pandas as pd
from datetime import datetime, timedelta

# Create sample data
sample_data = []
base_date = datetime(2022, 1, 15)

for i in range(100):
    date = base_date + timedelta(days=i*10)
    sample_data.append({
        'title': f'Sample Enforcement Action {i+1}',
        'date': date.strftime('%B %d, %Y'),
        'category': 'Criminal and Civil Actions' if i % 3 else 'State
        ↪  Enforcement Agencies',
        'link': f'https://oig.hhs.gov/fraud/enforcement/action{i+1}'
    })

# Save sample data
sample_df = pd.DataFrame(sample_data)
sample_df.to_csv('enforcement_actions_2022_1.csv', index=False)
print(f"Created sample CSV with {len(sample_df)} records for testing")
```

Step 3: Plot data based on scraped data

```
# Load data and visualization library
print("Step 3: Preparing data for visualization\n")

# Use matplotlib for PDF output (better compatibility)
import matplotlib
# Set backend before importing pyplot
matplotlib.rcParams['figure.dpi'] = 150
matplotlib.rcParams['savefig.dpi'] = 150
import matplotlib.pyplot as plt
HAS_ALTAIR = False
print("Using matplotlib for PDF-compatible visualizations")
print(f"Matplotlib backend: {matplotlib.get_backend()}")
```

```
# Read from CSV file created in Step 2
try:
    df = pd.read_csv('enforcement_actions_2022_1.csv')
    print(f"Loaded CSV with {len(df)} records\n")

    # Parse dates and create month-year column
    df['date_parsed'] = pd.to_datetime(df['date'], format='%B %d, %Y',
 ↪  errors='coerce')
    df['month_year'] = df['date_parsed'].dt.to_period('M')
    df = df.sort_values('date_parsed')

    print(f"Date range: {df['date_parsed'].min()} to
     ↪  {df['date_parsed'].max()}")
    print(f"Ready for visualization!\n")

except FileNotFoundError:
    print("Error: enforcement_actions_2022_1.csv not found.")
    print("\nThis file should have been created in Step 2c.")
    print("Please go back to Step 2c and:")
    print("1. Set run=True")
    print("2. Run the scraping function")
    print("3. Wait for it to complete (may take several minutes)")
    print("4. Then return here")
    df = None

# Stop here if no data
if df is None:
    raise FileNotFoundError("Cannot proceed without data file. See
     ↪  instructions above.")
```

```
Step 3: Preparing data for visualization

Using matplotlib for PDF-compatible visualizations
Matplotlib backend: Agg
Loaded CSV with 2000 records

Date range: 2023-10-02 00:00:00 to 2026-02-05 00:00:00
Ready for visualization!
```

1. Plot the number of enforcement actions over time

```
# Aggregate by month-year
monthly_counts = df.groupby('month_year').size().reset_index(name='count')
monthly_counts['month_year_str'] = monthly_counts['month_year'].astype(str)

print(f"Aggregated data by month: {len(monthly_counts)} months")
print(f"Date range: {monthly_counts['month_year_str'].min()} to
 ↪  {monthly_counts['month_year_str'].max()}")

# Create line chart with matplotlib
plt.figure(figsize=(10, 5))
plt.plot(monthly_counts['month_year_str'], monthly_counts['count'],
 ↪  marker='o', linewidth=2, color='#1f77b4')
plt.xlabel('Month-Year', fontsize=12)
plt.ylabel('Number of Enforcement Actions', fontsize=12)
plt.title('Enforcement Actions Over Time (Oct 2023 - Present)', fontsize=14,
 ↪  fontweight='bold')
plt.xticks(rotation=45, ha='right')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.gcf()  # Get current figure to display it
```

```
Aggregated data by month: 29 months
Date range: 2023-10 to 2026-02

<Figure size 1500x750 with 1 Axes>
```

Figure 1: Number of enforcement actions over time

2. Plot the number of enforcement actions categorized:

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

```
# Classify by main category
df['main_category'] = df['category'].apply(
    lambda x: 'State Enforcement Agencies' if pd.notna(x) and 'State
     ↪  Enforcement' in str(x)
    else 'Criminal and Civil Actions'
)
```

```
print("Main category distribution:")
print(df['main_category'].value_counts())

# Aggregate by month and category
monthly_category = df.groupby(['month_year',
 ↪  'main_category']).size().reset_index(name='count')
monthly_category['month_year_str'] =
 ↪  monthly_category['month_year'].astype(str)

# Create line chart with matplotlib
plt.figure(figsize=(10, 5))
for category in monthly_category['main_category'].unique():
    data = monthly_category[monthly_category['main_category'] == category]
    plt.plot(data['month_year_str'], data['count'], marker='o',
 ↪  label=category, linewidth=2)

plt.xlabel('Month-Year', fontsize=12)
plt.ylabel('Number of Actions', fontsize=12)
plt.title('Criminal and Civil vs. State Enforcement Agencies', fontsize=14,
 ↪  fontweight='bold')
plt.xticks(rotation=45, ha='right')
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.gcf()  # Get current figure to display it
```

```
Main category distribution:
main_category
Criminal and Civil Actions    1509
State Enforcement Agencies     491
Name: count, dtype: int64

<Figure size 1500x750 with 1 Axes>
```

Figure 2: Criminal and Civil Actions vs. State Enforcement Agencies

- based on five topics

```
# Classify into 5 topics based on title keywords
def classify_topic(title):
```

```
    if pd.isna(title):
        return 'Other'

    title_lower = title.lower()

    # Keywords for each topic
    if any(word in title_lower for word in ['bribery', 'bribe', 'kickback',
     ↪  'corruption']):
        return 'Bribery/Corruption'
    elif any(word in title_lower for word in ['drug', 'opioid', 'fentanyl',
     ↪  'controlled substance']):
        return 'Drug Enforcement'
    elif any(word in title_lower for word in ['bank', 'financial', 'money
     ↪  laundering', 'wire fraud']):
        return 'Financial Fraud'
    elif any(word in title_lower for word in ['health care', 'healthcare',
     ↪  'medicare', 'medicaid', 'hospital', 'medical']):
        return 'Health Care Fraud'
    else:
        return 'Other'

# Apply classification to Criminal and Civil Actions only
df['topic'] = df.apply(
    lambda row: classify_topic(row['title'])
    if row['main_category'] == 'Criminal and Civil Actions'
    else row['main_category'],
    axis=1
)

# Summary statistics
print("Topic Distribution (Criminal and Civil Actions):")
criminal_civil = df[df['main_category'] == 'Criminal and Civil
 ↪  Actions'].copy()
print(criminal_civil['topic'].value_counts())

# Filter for Criminal and Civil Actions and aggregate
monthly_topic = criminal_civil.groupby(['month_year',
 ↪  'topic']).size().reset_index(name='count')
monthly_topic['month_year_str'] = monthly_topic['month_year'].astype(str)

# Create line chart with matplotlib
plt.figure(figsize=(11, 6))
```

```
for topic in monthly_topic['topic'].unique():
    data = monthly_topic[monthly_topic['topic'] == topic]
    plt.plot(data['month_year_str'], data['count'], marker='o', label=topic,
 ↪  linewidth=2)

plt.xlabel('Month-Year', fontsize=12)
plt.ylabel('Number of Actions', fontsize=12)
plt.title('Criminal and Civil Actions by Topic', fontsize=14,
 ↪  fontweight='bold')
plt.xticks(rotation=45, ha='right')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.gcf()  # Get current figure to display it
```

```
Topic Distribution (Criminal and Civil Actions):
topic
Health Care Fraud     631
Other                 563
Bribery/Corruption    145
Drug Enforcement      131
Financial Fraud        39
Name: count, dtype: int64

<Figure size 1650x900 with 1 Axes>
```

Figure 3: Criminal and Civil Actions by Topic