Problem Set 4

Hari Dharshini Koundinya Swaminathen

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: HDKS

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 requests
from bs4 import BeautifulSoup
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')
alt.renderers.enable("png")
```

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

Step 1: Develop initial scraper and crawler

```
url = "https://oig.hhs.gov/fraud/enforcement/"
myheader = {"User-Agent": "DAP30538CourseBot/1.0 (hari2021@uchicago.edu)"}
response = requests.get(url, headers=myheader)
soup = BeautifulSoup(response.text, "lxml")
print(soup.text[:200])

title_links = soup.select("h2 a") #used inspect to find that the title text
 ↪  lives inside the <a>. the link lives in the href and both are wrapped in
 ↪  <h2> and asked ChatGPT how to go about it.
print("count:", len(title_links))

a0 = title_links[0]
container = a0.find_parent("div")
print(container.get_text("\n", strip=True))
#ChatGPT explained to me the importance of having containers instead of
 ↪  running my original code: titles = soup.select("h2 a"), dates =
 ↪  soup.select("span.text-base-dark.padding-right-105"), cats =
 ↪  soup.select("ul.display-inline.add-list-reset li"), having containers
 ↪  make sure we know which links and dates belong to which title.

date =
 ↪  container.select_one("span.text-base-dark.padding-right-105").get_text(strip=True)
 ↪  #used inspect to find these

cats = [li.get_text(strip=True) for li in
 ↪  container.select("ul.display-inline.add-list-reset li")] #used inspect to
 ↪  find these
```

```
print(date)
print(cats)

rows = []

for a in title_links:
    title = a.get_text(strip=True)
    link = a.get("href")

    container = a.find_parent("div")

    date =
↪   container.select_one("span.text-base-dark.padding-right-105").get_text(strip=True)
    cats = [li.get_text(strip=True) for li in
↪   container.select("ul.display-inline.add-list-reset li")]
    category = "; ".join(cats)

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

df = pd.DataFrame(rows)
print(df.head())
```

Enforcement Actions | Office of Inspector General | Government Oversight | U.S. Department of Health and Human Services

Skip to main content

An official websi
count: 20
Houston Transplant Doctor Indicted For Making False Statements In Patients' Medical Records
February 5, 2026
Criminal and Civil Actions
February 5, 2026
['Criminal and Civil Actions']

	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  /fraud/enforcement/houston-transplant-doctor-i...
1  /fraud/enforcement/multicare-health-system-to-...
2  /fraud/enforcement/brooklyn-banker-pleads-guil...
3  /fraud/enforcement/delafield-man-sentenced-to-...
4  /fraud/enforcement/former-nfl-player-convicted...
```

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code

1. Check year validity

- If the input year is earlier than 2013:
  - Print a message informing the user that only enforcement actions from 2013 onward are available
  - Exit the function

2. Set output filename

- Construct a filename based on the input year and month

3. Check run indicator

- If run_flag is False:
  - Attempt to load the previously saved CSV file
  - If the file exists, return the loaded dataframe
  - If the file does not exist, print an error message and exit

4. Define start date

- Create a datetime object representing the first day of the input month and year

5. Initialize scraping variables

- Define the base website URL
- Set the initial enforcement actions page URL (page 1)
- Define request headers
- Initialize an empty list to store scraped records

6. Begin page-by-page crawling loop

- While more pages are available:
  - Request the current enforcement actions page
  - Parse the page HTML using BeautifulSoup
  - Extract all enforcement action title links on the page
  - If no titles are found, stop crawling

7. Extract enforcement actions from the page

- For each title link on the page:
- Extract the title text and link
- Locate the parent container holding metadata
- Extract the enforcement action date
- Convert the date string to a datetime object
- Extract enforcement action category labels
- If the enforcement action date is on or after the start date:
- Store the title, date, category, and link in the results list
- Track all dates on the page for stopping logic

8. Check stopping condition

- If the newest date on the page is earlier than the start date:
  - Stop crawling additional pages

9. Move to the next page

- Locate the “Next” pagination link

- If no next page exists, stop crawling
- Construct the next page URL
- Pause for one second to avoid server overload
- Continue crawling from the next page

10. Create final dataset

- Convert the collected records into a dataframe
- Save the dataframe as a CSV file
- Print confirmation of saved file and number of rows

11. Return output

- Return the dataframe containing enforcement actions

- b. Create Dynamic Scraper I use a while loop rather than a for loop because the number of pages to scrape is not known in advance. The scraper must continue requesting subsequent pages until a stopping condition is met, either the enforcement actions become older than the specified month and year, or there is no next page available. A while loop naturally models this condition-based termination, whereas a for loop requires a predetermined number of iterations.This was suggested by ChatGPT and I striclty restricted it from giving me the code.

```
def scrape_enforcement_actions(month, year, run_flag=False):
    # Step 1: year check
    if year < 2013:
        print("Please restrict to year >= 2013, since only enforcement
        ↪  actions after 2013 are listed.")
        return None

    filename = f"/tmp/enforcement_actions_{year}_{month:02d}.csv" #had to do
↪  this becuause I kept getting the error: "PermissionError: [Errno 13]
↪  Permission denied: 'enforcement_actions_2024_02.csv'". I think this is
↪  because I was trying to save the file in a directory where I don't have
↪  write permissions. By changing the path to /tmp, which is a common
↪  directory for temporary files that typically has write permissions for
↪  all users, I should be able to avoid this error and successfully save the
↪  CSV file.

    if run_flag is False:
        print(f"run_flag=False, not scraping. Trying to load: {filename}")
        try:
```

```
            return pd.read_csv(filename)
        except FileNotFoundError:
            print(f"File not found: {filename}. Set run_flag=True once to
             ↪  generate it.")
            return None

    # Step 2: start date
    start_date = datetime(year, month, 1)

    # Step 4: setup
    base_url = "https://oig.hhs.gov"
    current_page_url = "https://oig.hhs.gov/fraud/enforcement/"
    headers = {"User-Agent": "DAP30538CourseBot/1.0 (hari2021@uchicago.edu)"}

    rows = []

    # Step 5-6: crawl pages
    while True:
        response = requests.get(current_page_url, headers=headers)
        soup = BeautifulSoup(response.text, "lxml")

        title_links = soup.select("h2 a")
        if not title_links:
            break

        page_dates = []

        for a in title_links:
            title = a.get_text(strip=True)
            link = a.get("href")

            card = a.find_parent("div")
            if card is None:
                continue

            date_tag =
↪   card.select_one("span.text-base-dark.padding-right-105")
            if date_tag is None:
                 continue
            date_str = date_tag.get_text(strip=True)

            try:
                date_dt = datetime.strptime(date_str, "%B %d, %Y")
```

```
            except ValueError:
                continue

            page_dates.append(date_dt)

            cats = [
                li.get_text(strip=True)
                for li in card.select("ul.display-inline.add-list-reset li")
            ]
            category = "; ".join(cats) if cats else None

            if date_dt >= start_date:
                rows.append({
                    "title": title,
                    "date": date_str,
                    "category": category,
                    "link": link
                })

        # Stop condition based on date
        if page_dates and max(page_dates) < start_date:
            break

        # Find next page (relative link)
        next_a = soup.select_one("a.pagination-next") #used inspect to find
↪  this
        if next_a is None or not next_a.get("href"):
            break
        next_href = next_a.get("href")
        current_page_url = "https://oig.hhs.gov/fraud/enforcement/" +
↪  next_href
        time.sleep(1)
    df = pd.DataFrame(rows)
    df.to_csv(filename, index=False)
    print(f"Saved {filename} with {len(df)} rows.")
    return df

df = scrape_enforcement_actions(
    month=2,
    year=2024,
    run_flag=False)

print(df.head())
```

```
print("rows:", len(df))

df["date_dt"] = pd.to_datetime(df["date"], format="%B %d, %Y")
earliest = df.loc[df["date_dt"].idxmin()]
earliest
```

```
run_flag=False, not scraping. Trying to load:
/tmp/enforcement_actions_2024_02.csv
                                                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  /fraud/enforcement/houston-transplant-doctor-i...
1  /fraud/enforcement/multicare-health-system-to-...
2  /fraud/enforcement/brooklyn-banker-pleads-guil...
3  /fraud/enforcement/delafield-man-sentenced-to-...
4  /fraud/enforcement/former-nfl-player-convicted...
rows: 1725

title       Baltimore County Serial Fraudster Sentenced to...
date                                          February 1, 2024
category                            Criminal and Civil Actions
link        /fraud/enforcement/baltimore-county-serial-fra...
date_dt                                   2024-02-01 00:00:00
Name: 1724, dtype: object
```

- c. Test Your Code

```
df = scrape_enforcement_actions(month=1, year=2022, run_flag=False)

print("Total enforcement actions scraped:", len(df))
```

```
df["date_dt"] = pd.to_datetime(df["date"], format="%B %d, %Y")

earliest_idx = df["date_dt"].idxmin()
earliest_row = df.loc[earliest_idx]

print("\nEarliest enforcement action scraped:")
print("Date:", earliest_row["date"])
print("Title:", earliest_row["title"])
print("Category:", earliest_row["category"])
print("Link:", earliest_row["link"])
```

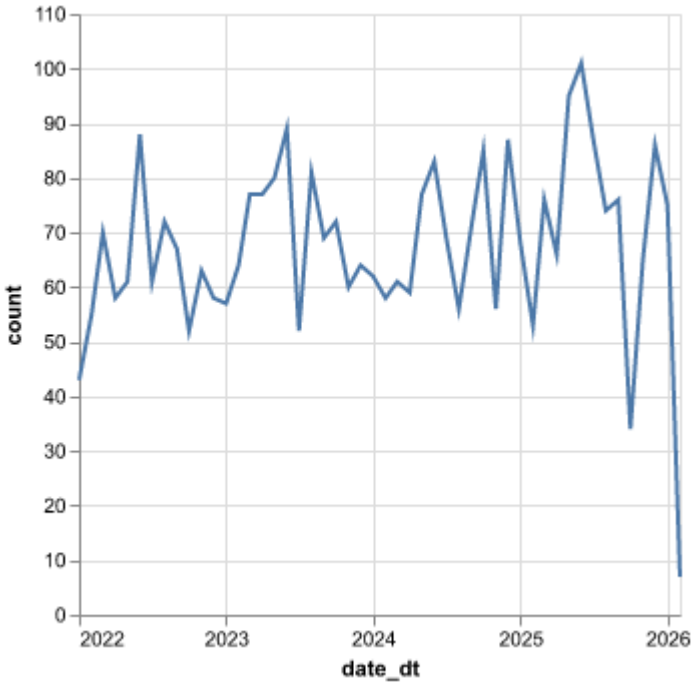
```
run_flag=False, not scraping. Trying to load:
/tmp/enforcement_actions_2022_01.csv
Total enforcement actions scraped: 3377

Earliest enforcement action scraped:
Date: January 4, 2022
Title: Central Medical Systems, LLC, Alan Trent Harley And Joan Harley Agree
To Pay $600K To Settle False Claims Act Liability
Category: Criminal and Civil Actions
Link:
/fraud/enforcement/central-medical-systems-llc-alan-trent-harley-and-joan-harley-agree-to-pay
```

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

```
df = pd.read_csv("/tmp/enforcement_actions_2022_01.csv")

df["date_dt"] = pd.to_datetime(df["date"], format="%B %d, %Y")
monthly_counts = (
    df
    .groupby(pd.Grouper(key="date_dt", freq="MS"))
    .size()
    .reset_index(name="count")
)


chart = alt.Chart(monthly_counts).mark_line().encode(
```

```
    x="date_dt:T",
    y="count:Q"
)

chart
```

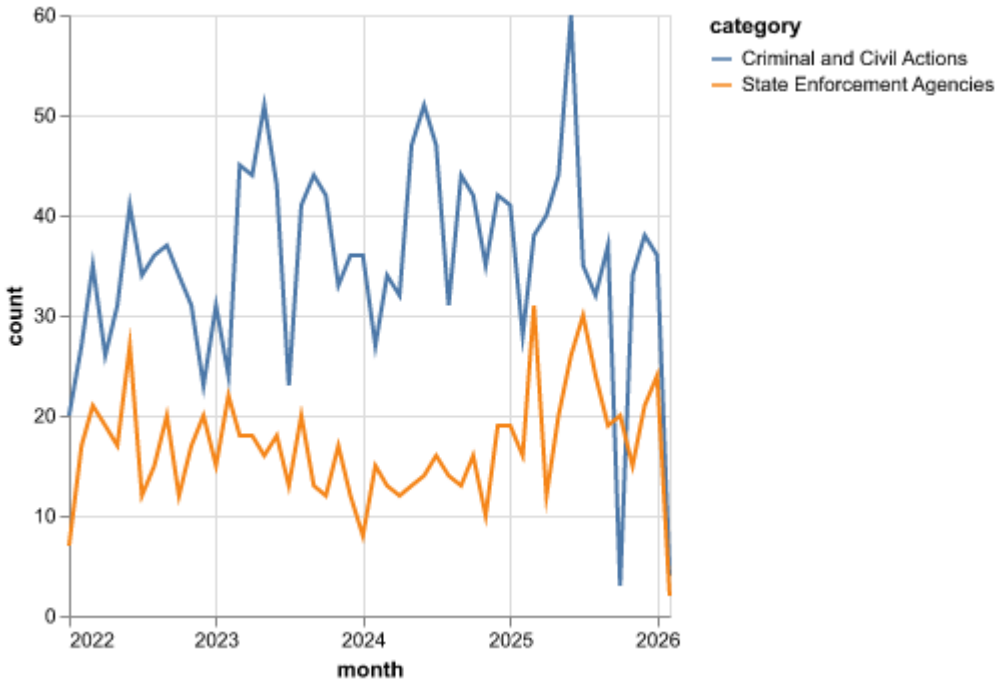


2. Plot the number of enforcement actions categorized:

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

```
df["month"] = df["date_dt"].dt.to_period("M").dt.to_timestamp()

df[["date", "date_dt", "month", "category"]].head()

# keeping only the two categories we care about
df_cat = df[df["category"].isin([
    "Criminal and Civil Actions",
    "State Enforcement Agencies"
])]
```

```
# aggregating the number of actions per month per category
monthly_cat_counts = (
    df_cat
    .groupby(["month", "category"])
    .size()
    .reset_index(name="count")
)

monthly_cat_counts.head()

chart_cat = alt.Chart(monthly_cat_counts).mark_line().encode(
    x="month:T",
    y="count:Q",
    color="category:N"
)

chart_cat
```



- based on five topics I wrote some words based on my observations for each category and ChatGPT helped me finetune it.

```
cca = df[df["category"] == "Criminal and Civil Actions"].copy()

# lowercase titles for easy keyword matching
cca["title_lower"] = cca["title"].str.lower()

cca[["title", "title_lower"]].head()

def classify_topic(title_lower):
    # Health Care Fraud
    if any(k in title_lower for k in [
        "medicare", "medicaid", "health", "hospital", "clinic",
        "physician", "doctor", "nurse", "pharmacy", "patient",
        "home health", "hospice"
    ]):
        return "Health Care Fraud"

    # Financial Fraud
    if any(k in title_lower for k in [
        "bank", "banker", "financial", "loan", "wire fraud",
        "money laundering", "launder", "check", "credit",
        "mortgage", "securities", "investment", "embezzl", "tax"
    ]):
        return "Financial Fraud"

    # Drug Enforcement
    if any(k in title_lower for k in [
        "drug", "opioid", "fentanyl", "meth", "heroin",
        "cocaine", "prescription", "controlled substance"
    ]):
        return "Drug Enforcement"

    # Bribery / Corruption
    if any(k in title_lower for k in [
        "brib", "kickback", "corrupt", "extortion",
        "racketeer", "conspiracy"
    ]):
        return "Bribery/Corruption"

    # Everything else
    return "Other"


# apply classification
```

```
cca["topic"] = cca["title_lower"].apply(classify_topic)

# quick check
cca[["title", "topic"]].head(10)

topic_monthly_counts = (
    cca
    .groupby(["month", "topic"])
    .size()
    .reset_index(name="count")
)

topic_monthly_counts.head()

chart_topics = alt.Chart(topic_monthly_counts).mark_line().encode(
    x="month:T",
    y="count:Q",
    color="topic:N"
)

chart_topics
```