PS4

Jianghan YUAN

2026-02-06

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: YJH

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
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

```python
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin

BASE = "https://oig.hhs.gov"
URL = "https://oig.hhs.gov/fraud/enforcement/"


response = requests.get(URL, headers={"User-Agent": "Mozilla/5.0"})
response.raise_for_status()

soup = BeautifulSoup(response.text, "html.parser")


rows = soup.select("li.usa-card")

records = []

for row in rows:
  a = row.select_one("h2 a")
  if not a:
    continue

  title = a.get_text(strip = True)
  link = urljoin(BASE, a.get("href", ""))

  date_el = row.select_one("span.text-base-dark")
  if date_el is None:
    meta = row.select_one("div.font-body-sm")
    date_el = meta.select_one("span") if meta else None

  date = date_el.get_text(strip = True) if date_el else None

  cat_els = row.select("li.usa-tag")
  categories = [c.get_text(strip = True) for c in cat_els]
  category = "; ".join(categories) if categories else None

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

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

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

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

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

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code Before implementing the dynamic scraper, I first planned the overall process the function would execute. The function takes the year and month as input parameters, aiming to scrape all enforcement actions from the specified start time up to the current date. At the beginning of the function, it first checks if the user-input year is greater than or equal to 2013; if the year is earlier than 2013, it will prompt the user that the website only provides enforcement action data from 2013 onwards and terminate the function.

After passing the year check, the function initializes several key variables, including the base URL, a page number counter, and an empty list to store the scraped results. Because it's impossible to determine in advance how many pages need to be traversed to cover the target time range, the scraper does not use a fixed-number for loop, but instead employs a while loop with a clear stopping condition. In each loop iteration, the function constructs the corresponding page URL based on the current page number, sends a request to the website, and parses the returned HTML content.

Next, the function extracts all enforcement action records from the current page and sequentially scrapes the title, date, category, and link of each record, then appends this information to the results list. Since the pages are usually arranged in reverse chronological order, the function checks whether the date of the earliest enforcement action on the current page is still later than or equal to the user-specified start year and month. If the condition is still met, it means that further pages need to be traversed; if it is earlier than the start time, it means that enough data has been scraped, and the function will stop pagination after retaining the records that meet the time criteria.

During the pagination process, to avoid overloading the server and prevent being blocked, the function pauses for one second before requesting each new page. After the loop ends, the function organizes the accumulated results into a clean dataframe and saves it as a CSV file with a specified name for subsequent analysis.

- b. Create Dynamic Scraper

```python
BASE = "https://oig.hhs.gov"
BASE_URL = "https://oig.hhs.gov/fraud/enforcement/"
HEADERS = {"User-Agent": "Mozilla/5.0"}

def scrape_from_2024(year=2024, month=1, run_scraper=True):
    if year < 2013:
        print("Year must be >= 2013 because the site only lists enforcement
         ↪  actions after 2013.")
        return pd.DataFrame(columns=["title", "date", "category", "link"])

    start_date = pd.Timestamp(year=year, month=month, day=1)

    out_csv = f"enforcement_actions_{year}_{month:02d}.csv"

    if not run_scraper:
        print(f"run_scraper=False: Skipping scraping and reading {out_csv}
         ↪  instead.")
        return pd.read_csv(out_csv)

    records = []
    page = 0

    while True:
        url = BASE_URL if page == 0 else f"{BASE_URL}?page={page}"

        resp = requests.get(url, headers=HEADERS, timeout=30)
        resp.raise_for_status()
        soup = BeautifulSoup(resp.text, "html.parser")

        rows = soup.select("li.usa-card")
        if not rows:
            break

        page_dates = []

        for row in rows:
            a = row.select_one("h2 a")
            if not a:
                continue
            title = a.get_text(strip=True)
            link = urljoin(BASE, a.get("href", ""))

            date_el = row.select_one("span.text-base-dark")
            if date_el is None:
                meta = row.select_one("div.font-body-sm")
                date_el = meta.select_one("span") if meta else None

            date_str = date_el.get_text(strip=True) if date_el else None
            date_dt = pd.to_datetime(date_str, errors="coerce")
            page_dates.append(date_dt)
            cats = [x.get_text(strip=True) for x in row.select("li.usa-tag")]
            category = "; ".join(cats) if cats else None

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

        page_dates = [d for d in page_dates if pd.notna(d)]
        if page_dates and min(page_dates) < start_date:
            break

        time.sleep(1)
        page += 1

    df = pd.DataFrame(records)
    df = df.dropna(subset=["date"]).copy()
    df = df[df["date"] >= start_date].copy()
    df = df.sort_values("date", ascending=False).reset_index(drop=True)

    df.to_csv(out_csv, index=False)
    print(f"Saved: {out_csv} (Do NOT commit this file to GitHub.)")

    print(f"Total enforcement actions since {start_date.date()}: {len(df)}")

    earliest = df.sort_values("date", ascending=True).head(1)
    print("Earliest enforcement action in this dataframe:")
    print(earliest[["date", "title", "category",
     ↪  "link"]].to_string(index=False))

    return df


df_2024 = scrape_from_2024(run_scraper=False)
```

```
run_scraper=False: Skipping scraping and reading
enforcement_actions_2024_01.csv instead.
```

- c. Test Your Code

```
df_2022 = scrape_from_2024(year=2022, month=1, run_scraper=False)

print("\n--- Part (c) results (since 2022-01) ---")
print(f"Total enforcement actions since {pd.Timestamp(2022,1,1).date()}:
 ↪  {len(df_2022)}")

earliest_2022 = df_2022.sort_values("date", ascending=True).head(1)
print("Earliest enforcement action in this dataframe:")
print(earliest_2022[["date", "title", "category",
 ↪  "link"]].to_string(index=False))
```

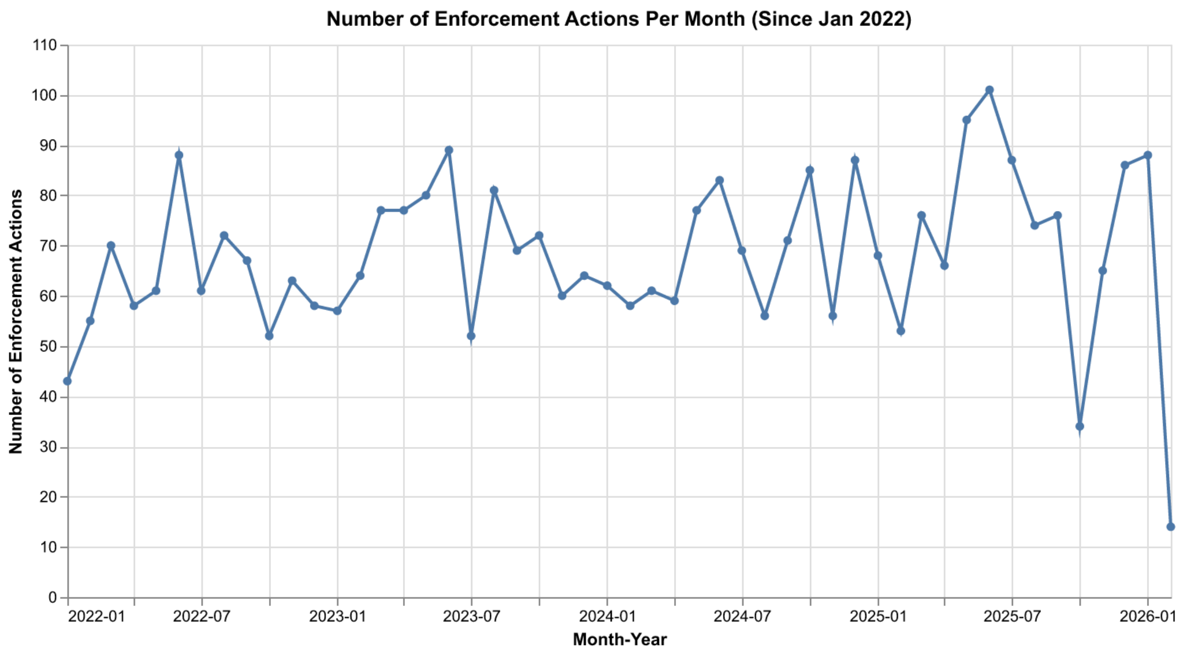
```
run_scraper=False: Skipping scraping and reading
enforcement_actions_2022_01.csv instead.

--- Part (c) results (since 2022-01) ---
Total enforcement actions since 2022-01-01: 3397
Earliest enforcement action in this dataframe:
      date
      title               category
      link
2022-01-04 Integrated Pain Management Medical Group Agreed to Pay $10,000 for
Allegedly Violating the Civil Monetary Penalties Law by Employing Excluded
Individuals Fraud Self-Disclosures
https://oig.hhs.gov/fraud/enforcement/integrated-pain-management-medical-group-agreed-to-pay-
```

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

```
import vl_convert as vlc
from IPython.display import Image, display

df = pd.read_csv("enforcement_actions_2022_01.csv")

df["date"] = pd.to_datetime(df["date"], errors="coerce")

df["month_year"] = df["date"].dt.to_period("M").dt.to_timestamp()

monthly_total = (
    df.groupby("month_year", as_index=False)
```

```
      .size()
      .rename(columns={"size": "n_actions"})
)

chart1 = (
    alt.Chart(monthly_total)
    .mark_line(point=True)
    .encode(
        x=alt.X(
            "month_year:T",
            title="Month-Year",
            axis=alt.Axis(
                format="%Y-%m",
                tickCount=12
            )
        ),
        y=alt.Y("n_actions:Q", title="Number of Enforcement Actions"),
        tooltip=[
            alt.Tooltip("month_year:T", title="Month", format="%Y-%m"),
            alt.Tooltip("n_actions:Q", title="Count")
        ]
    )
    .properties(
        title="Number of Enforcement Actions Per Month (Since Jan 2022)",
        width=700,
        height=350
    )
)

chart1

png_bytes = vlc.vegalite_to_png(chart1.to_dict(), scale = 2)

out_path = "chart1.png"
with open(out_path, "wb") as f:
    f.write(png_bytes)

display(Image(filename = out_path))
```

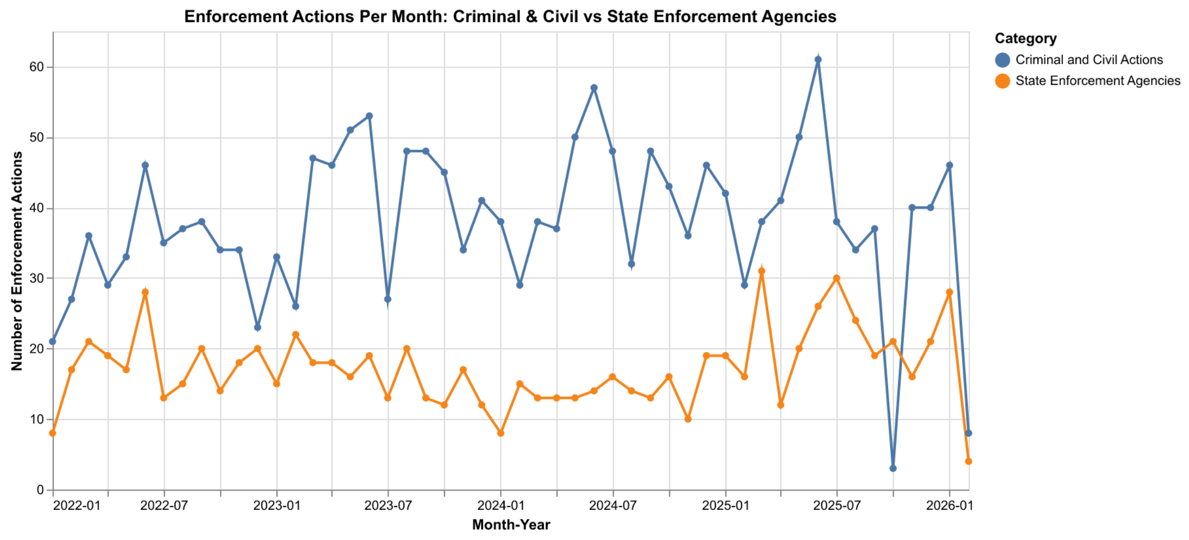


2. Plot the number of enforcement actions categorized:

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

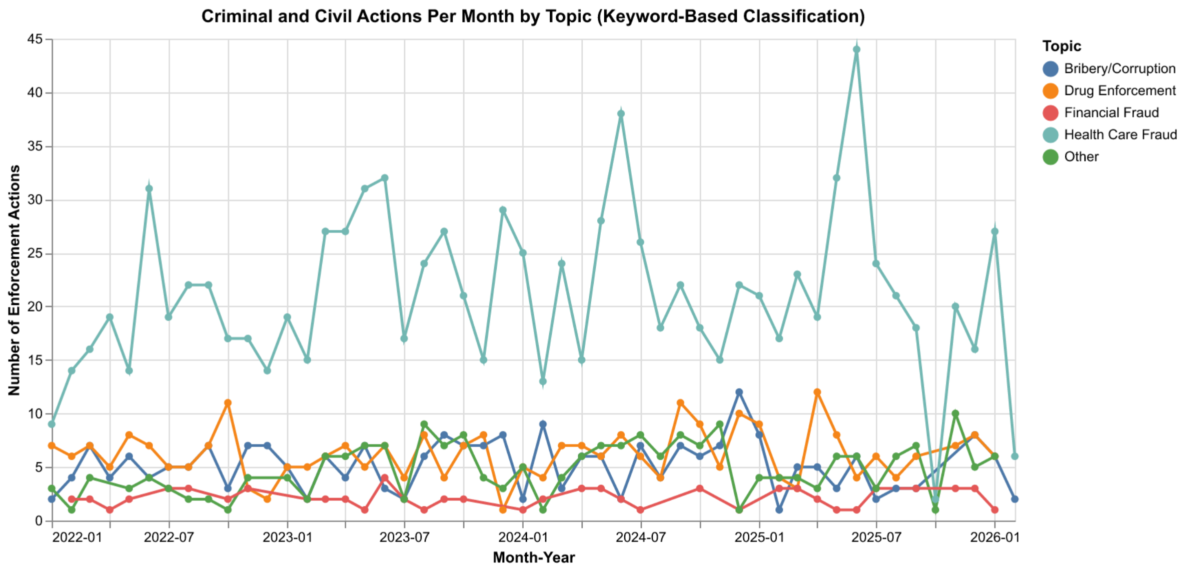
```
df_cat = df.copy()
df_cat["category_clean"] =
↪  df_cat["category"].fillna("").str.split(r"\s*;\s*")
df_cat = df_cat.explode("category_clean")
df_cat["category_clean"] = df_cat["category_clean"].str.strip()
df_cat = df_cat[df_cat["category_clean"] != ""].copy()


two = df_cat[df_cat["category_clean"].isin([
    "Criminal and Civil Actions",
    "State Enforcement Agencies"
])].copy()


monthly_two = (
    two.groupby(["month_year", "category_clean"], as_index=False)
       .size()
       .rename(columns={"size": "n_actions"})
)
```

```
chart2 = (
    alt.Chart(monthly_two)
    .mark_line(point=True)
    .encode(
        x=alt.X(
            "month_year:T",
            title="Month-Year",
            axis=alt.Axis(format="%Y-%m", tickCount=12)
        ),
        y=alt.Y("n_actions:Q", title="Number of Enforcement Actions"),
        color=alt.Color("category_clean:N", title="Category"),
        tooltip=[
            alt.Tooltip("month_year:T", title="Month", format="%Y-%m"),
            alt.Tooltip("category_clean:N", title="Category"),
            alt.Tooltip("n_actions:Q", title="Count")
        ]
    )
    .properties(
        title="Enforcement Actions Per Month: Criminal & Civil vs State
↪   Enforcement Agencies",
        width=700,
        height=350
    )
)

chart2

png_bytes = vlc.vegalite_to_png(chart2.to_dict(), scale = 2)

out_path = "chart2.png"
with open(out_path, "wb") as f:
    f.write(png_bytes)

display(Image(filename = out_path))
```



- based on five topics

```python
criminal = df_cat[df_cat["category_clean"] == "Criminal and Civil
 ↪  Actions"].copy()

criminal["title_lower"] = criminal["title"].fillna("").str.lower()

def assign_topic(t: str) -> str:
    # Bribery/Corruption
    if any(k in t for k in ["brib", "kickback", "corrupt", "extortion",
     ↪  "payoff"]):
        return "Bribery/Corruption"

    # Drug Enforcement
    if any(k in t for k in ["drug", "opioid", "fentanyl", "pill", "controlled
     ↪  substance", "pharma"]):
        return "Drug Enforcement"

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

    # Health Care Fraud
    if any(k in t for k in ["medicare", "medicaid", "health", "hospital",
     ↪  "clinic", "physician", "doctor", "medical", "patient", "billing",
     ↪  "claims", "nursing", "home health", "pharmacy"]):
        return "Health Care Fraud"

    return "Other"

criminal["topic"] = criminal["title_lower"].apply(assign_topic)

monthly_topic = (
    criminal.groupby(["month_year", "topic"], as_index=False)
            .size()
            .rename(columns={"size": "n_actions"})
)

chart3 = (
    alt.Chart(monthly_topic)
    .mark_line(point=True)
    .encode(
        x=alt.X(
            "month_year:T",
            title="Month-Year",
            axis=alt.Axis(format="%Y-%m", tickCount=12)
        ),
        y=alt.Y("n_actions:Q", title="Number of Enforcement Actions"),
        color=alt.Color("topic:N", title="Topic"),
        tooltip=[
            alt.Tooltip("month_year:T", title="Month", format="%Y-%m"),
            alt.Tooltip("topic:N", title="Topic"),
            alt.Tooltip("n_actions:Q", title="Count")
        ]
    )
    .properties(
        title="Criminal and Civil Actions Per Month by Topic (Keyword-Based
 ↪  Classification)",
        width=700,
        height=350
    )
)

chart3

png_bytes = vlc.vegalite_to_png(chart3.to_dict(), scale = 2)

out_path = "chart3.png"
with open(out_path, "wb") as f:
    f.write(png_bytes)

display(Image(filename = out_path))
```


Criminal and Civil Actions Per Month by Topic (Keyword-Based Classification)
Number of Enforcement Actions
Month-Year
Topic
Bribery/Corruption
Drug Enforcement
Financial Fraud
Health Care Fraud
Other