# title

author

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: **___**

## 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
import pandas as pd

url = "https://oig.hhs.gov/fraud/enforcement/"
response = requests.get(url)

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

actions = []
cards = soup.find_all("li", class_="usa-card")
for card in cards:

    # ----- Title & Link -----
    title_tag = card.find("h2", class_="usa-card__heading")
    link_tag = title_tag.find("a") if title_tag else None

    title = link_tag.text.strip() if link_tag else None
    link = link_tag.get("href") if link_tag else None

    # ----- Date -----
    date_tag = card.find("span", class_="text-base-dark")
    date = date_tag.text.strip() if date_tag else None

    # ----- Category -----
    meta_div = card.find("div", class_="font-body-sm")
    category_tag = None
    category = None
    if meta_div:
        category_tag = meta_div.find("li", class_="usa-tag")
        if category_tag:
          category = category_tag.text.strip()

    actions.append({
        "title": title,
        "date": date,
        "category": category,
        "link": link
    })
df_step1 = pd.DataFrame(actions)
df_step1.head()
```

| | title | date | category | lin |
|---|---|---|---|---|
| 0 | Houston Transplant Doctor Indicted For Making ... | February 5, 2026 | Criminal and Civil Actions | /fr |
| 1 | MultiCare Health System to Pay Millions to Set... | February 4, 2026 | Criminal and Civil Actions | /fr |
| 2 | Brooklyn Banker Pleads Guilty to Laundering Pr... | February 3, 2026 | COVID-19 | /fr |
| 3 | Delafield Man Sentenced to 18 Months' Imprison... | February 3, 2026 | Criminal and Civil Actions | /fr |
| 4 | Former NFL Player Convicted for $197M Medicare... | February 3, 2026 | Criminal and Civil Actions | /fr |

## Step 2: Making the scraper dynamic

### 1. Turning the scraper into a function

- a. Pseudo-Code Define function scrape_enforcement_actions(start_year, start_month, run_scraper):

  If start_year < 2013: Print a message reminding the user to restrict input to year >= 2013 Exit the function

  If run_scraper is False: Load enforcement_actions_year_month.csv Return the loaded dataframe

  Initialize an empty list called all_actions Set page_number = 0 Set keep_scraping = True

  While keep_scraping is True:

```
  Construct the page URL using page_number
  Send a request to the webpage
  Parse the HTML using BeautifulSoup
```

```
    Identify all enforcement action cards on the page

    For each card on the page:
        Extract title, date, category, and link
        Convert the date string to a datetime object

        If the action date is earlier than (start_year, start_month):
            Set keep_scraping = False
            Break out of the loop over cards

        Append the extracted information to all_actions

    Increment page_number by 1
    Pause for 1 second using time.sleep(1) to avoid rate limiting
```

Convert all_actions into a tidy dataframe Save the dataframe as enforcement_actions_year_month.csv
Return the dataframe

- b. Create Dynamic Scraper

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

def scrape_enforcement_actions(start_year, start_month, run_scraper=True):

    if start_year < 2013:
        print("Please restrict the input to year >= 2013. Enforcement actions
         ↪  are only available after 2013.")
        return None

    if not run_scraper:
        return pd.read_csv("enforcement_actions_year_month.csv")

    all_actions = []
    page = 0
    keep_scraping = True

    start_date = datetime(start_year, start_month, 1)

    while keep_scraping:
        url = f"https://oig.hhs.gov/fraud/enforcement/?page={page}"
        response = requests.get(url)
        soup = BeautifulSoup(response.text, "html.parser")

        cards = soup.find_all("li", class_="usa-card")

        if len(cards) == 0:
            break

        for card in cards:

            title_tag = card.find("h2", class_="usa-card__heading")
            link_tag = title_tag.find("a") if title_tag else None

            title = link_tag.text.strip() if link_tag else None
            link = link_tag.get("href") if link_tag else None

            date_tag = card.find("span", class_="text-base-dark")
            date_str = date_tag.text.strip() if date_tag else None
            action_date = datetime.strptime(date_str, "%B %d, %Y")

            if action_date < start_date:
                keep_scraping = False
                break

            meta_div = card.find("div", class_="font-body-sm")
            category_tag = meta_div.find("li", class_="usa-tag") if meta_div
↪   else None
            category = category_tag.text.strip() if category_tag else None

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

        page += 1
        time.sleep(1)

    df = pd.DataFrame(all_actions)
    df.to_csv("enforcement_actions_year_month.csv", index=False)
    return df
```

- c. Test Your Code

```
df_2024 = scrape_enforcement_actions(2024, 1, run_scraper=True)
len(df_2024)
df_2024.tail(1)

df_2022 = scrape_enforcement_actions(2022, 1, run_scraper=True)
len(df_2022)
df_2022.tail(1)
```

| | title | date | category | link |
|---|---|---|---|---|
| 3396 | Integrated Pain Management Medical Group Agree... | 2022-01-04 | Fraud Self-Disclosures | /fraud/en |

## Step 3: Plot data based on scraped data

### 1. Plot the number of enforcement actions over time

### 2. Plot the number of enforcement actions categorized:

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

```
import pandas as pd
import altair as alt
alt.renderers.enable("default")

df = pd.read_csv("enforcement_actions_year_month.csv")
df["date"] = pd.to_datetime(df["date"])
df_2022 = df[df["date"] >= "2022-01-01"]
df_monthly = (
    df_2022
    .assign(month=df_2022["date"].dt.to_period("M").dt.to_timestamp())
    .groupby("month")
    .size()
    .reset_index(name="num_actions")
)
chart_overall = (
    alt.Chart(df_monthly)
    .mark_line()
    .encode(
        x=alt.X("month:T", title="Month"),
        y=alt.Y("num_actions:Q", title="Number of Enforcement Actions"),
        tooltip=[
            alt.Tooltip("month:T", title="Month"),
            alt.Tooltip("num_actions:Q", title="Actions")
        ]
    )
    .properties(
        title="Monthly HHS OIG Enforcement Actions (Since January 2022)"
    )
)

chart_overall
```

```
alt.Chart(...)
```

- based on five topics

```
df_main_categories = df_2022[
    df_2022["category"].isin([
        "Criminal and Civil Actions",
        "State Enforcement Agencies"
    ])
]
df_cat_monthly = (
    df_main_categories

↪   .assign(month=df_main_categories["date"].dt.to_period("M").dt.to_timestamp())
    .groupby(["month", "category"])
    .size()
    .reset_index(name="num_actions")
)
chart_category = (
    alt.Chart(df_cat_monthly)
    .mark_line()
    .encode(
        x=alt.X("month:T", title="Month"),
        y=alt.Y("num_actions:Q", title="Number of Enforcement Actions"),
        color=alt.Color("category:N", title="Category"),
        tooltip=[
            alt.Tooltip("month:T", title="Month"),
            alt.Tooltip("category:N", title="Category"),
            alt.Tooltip("num_actions:Q", title="Actions")
        ]
    )
    .properties(
        title="Enforcement Actions by Category (Since January 2022)"
    )
)

chart_category

df_criminal = df_2022[
    df_2022["category"] == "Criminal and Civil Actions"
].copy()
def classify_topic(title):
    title = title.lower()

    if any(word in title for word in ["medicare", "medicaid", "health",
     ↪  "hospital", "clinic"]):
        return "Health Care Fraud"
    elif any(word in title for word in ["bank", "financial", "loan", "money",
     ↪  "wire", "securities"]):
        return "Financial Fraud"
    elif any(word in title for word in ["drug", "opioid", "pharmacy",
     ↪  "controlled substance"]):
        return "Drug Enforcement"
    elif any(word in title for word in ["bribe", "bribery", "corruption",
     ↪  "kickback"]):
        return "Bribery/Corruption"
    else:
        return "Other"
df_criminal["topic"] = df_criminal["title"].apply(classify_topic)
df_topic_monthly = (
    df_criminal
    .assign(month=df_criminal["date"].dt.to_period("M").dt.to_timestamp())
    .groupby(["month", "topic"])
    .size()
    .reset_index(name="num_actions")
)
chart_topics = (
    alt.Chart(df_topic_monthly)
    .mark_line()
    .encode(
        x=alt.X("month:T", title="Month"),
        y=alt.Y("num_actions:Q", title="Number of Enforcement Actions"),
        color=alt.Color("topic:N", title="Topic"),
        tooltip=[
            alt.Tooltip("month:T", title="Month"),
            alt.Tooltip("topic:N", title="Topic"),
            alt.Tooltip("num_actions:Q", title="Actions")
        ]
    )
    .properties(
        title="Criminal and Civil Actions by Topic (Since January 2022)"
    )
)

chart_topics
```

```
alt.Chart(...)
```