Problem Set 4

Ayusha Aryal

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: **AA**

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

url = "https://oig.hhs.gov/fraud/enforcement/"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'lxml')

#Upon inspection, I found the section we are interested in:
section = soup.find("ul", class_="usa-card-group padding-y-0")
section

rows =[]
for entry in section.find_all("li", recursive = False):
  link_tag = entry.find("a", href=True)
  title = link_tag.get_text(strip=True)
  link = "https://oig.hhs.gov" + link_tag["href"]

  for span in entry.find_all("span"):
    date = span.get_text(strip=True)

  category_list = entry.find("ul")
  categories = []
  for item in category_list.find_all("li"):
    text = item.get_text(strip=True)
    categories.append(text)
    category = ", ".join(categories)

  rows.append({"Enforcement Title": title, "Date": date, "Category":
↪ category, "Link": link})
```

```
df = pd.DataFrame(rows)
df["Enforcement Title"].unique()
df["Date"].unique()
df["Category"].unique()

df.head()
```

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

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code

FUNCTION dynamic_scraper(month, year, run_indicator) IF run_indicator is FALSE RETURN nothing END IF

IF year < 2013 PRINT "Restrict the search till 2013" RETURN nothing END IF

start_date <- year, month, 1 rows <- emptylist page <- 1 stop <- FALSE

WHILE TRUE (Here, we are using a WHILE loop because we don't know how many pages we need to crawl and scrape.) url = "https://oig.hhs.gov/fraud/enforcement/?page=" + page Download url Parse HTML

```
section <- find enforcement actions section on page

IF section does not exist
    BREAK loop
END IF

FOR each enforcement entry in section

    title <- extract title text
    link <- extract link and add website base URL
    spans <- find all span elements in entry
    date_text <- text of last span
    date <- convert date_text to date format

    categories <- emptylist
    FOR each category in entry
        category_text <- extract text
        ADD category_text to categories list
    END FOR

    category <- join categories into single string

    IF date >= start_date
        ADD (title, date, category, link) to rows
    ELSE
        stop <- TRUE
        BREAK for loop
    END IF

END FOR

IF stop is TRUE
    BREAK while loop
END IF

page <- page + 1
WAIT 1 second
```

END WHILE

Convert rows into dataframe Save dataframe as CSV file "enforcement_actions-year_month.csv"

RETURN dataframe

END FUNCTION

- b. Create Dynamic Scraper

```python
def dynamic_scraper(month, year, run_indicator):
  if run_indicator == False:
    return None

  if year < 2013:
    print("Restrict the search till 2013")
    return None

  start_date = pd.Timestamp(year, month, 1)
  rows = []
  page = 1
  stop = False

  while True:
    url = f"https://oig.hhs.gov/fraud/enforcement/?page={page}"
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'lxml')
    section = soup.find("ul", class_="usa-card-group padding-y-0")

    if section is None:
      break

    for entry in section.find_all("li", recursive = False):
      link_tag = entry.find("a", href=True)
      title = link_tag.get_text(strip=True)
      link = "https://oig.hhs.gov" + link_tag["href"]

      for span in entry.find_all("span"):
        dt = span.get_text(strip=True)
        date = pd.to_datetime(dt)

      category_list = entry.find("ul")
      categories = []
      for item in category_list.find_all("li"):
        text = item.get_text(strip=True)
        categories.append(text)
        category = ", ".join(categories)

      if date >= start_date:
        rows.append({"Enforcement Title": title, "Date": date, "Category":
↪   category, "Link": link})
      else:
        stop = True
        break
    if stop:
      break

    page += 1
    time.sleep(1)
  df_dynamic = pd.DataFrame(rows)
  file = f"enforcement_actions-{year}_{month}.csv"
  df_dynamic.to_csv(file, index = False)
  return df_dynamic

df_jan_2024 = dynamic_scraper(month=1, year= 2024, run_indicator=False)

df1 = pd.read_csv("enforcement_actions-2024_1.csv")
df1["Date"].unique()
df1["Category"].unique()

observations = len(df1)
print(f"For the final dataframe of enforcement actions listed from January
 ↪  2024 till today, the total number of observations is {observations}.")

df1_sorted= df1.sort_values("Date").head()
n1_obs = df1.sort_values("Date").iloc[0]
n1_obs

print("In this dataset, the earliest enforcement action was in 2024-01-03 and
 ↪  is titled 'Former Nurse Aide Indicted In Death Of Clarksville Patient
 ↪  Arrested In Georgia' which falls under the category 'State Enforcement
 ↪  Agencies'.")
```

```
For the final dataframe of enforcement actions listed from January 2024 till
today, the total number of observations is 1787.
In this dataset, the earliest enforcement action was in 2024-01-03 and is
titled 'Former Nurse Aide Indicted In Death Of Clarksville Patient Arrested
In Georgia' which falls under the category 'State Enforcement Agencies'.
```

- c. Test Your Code

```
df_jan_2022 = dynamic_scraper(month=1, year= 2022, run_indicator=False)
df2 = pd.read_csv("enforcement_actions-2022_1.csv")
df2["Date"].unique()

observations = len(df2)
print(f"For the final dataframe of enforcement actions listed from January
 ↪  2022 till today, the total number of observations is {observations}.")

df2_sorted= df2.sort_values("Date").head()
```

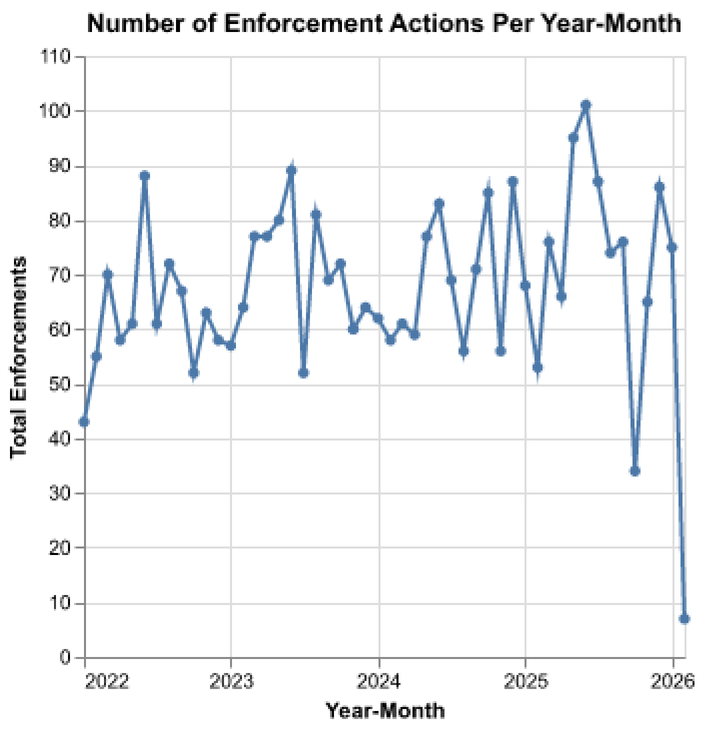
```python
n1_obs = df2.sort_values("Date").iloc[0]
n1_obs

print("In this dataset, the earliest enforcement action was in 2022-01-04 and
↪ is titled 'Integrated Pain Management Medical Group Agreed to Pay $10,000
↪ for Allegedly Violating the Civil Monetary Penalties Law by Employing
↪ Excluded Individuals' which falls under the category 'Fraud
↪ Self-Disclosures'.")
```

```
For the final dataframe of enforcement actions listed from January 2022 till
today, the total number of observations is 3377.
In this dataset, the earliest enforcement action was in 2022-01-04 and is
titled 'Integrated Pain Management Medical Group Agreed to Pay $10,000 for
Allegedly Violating the Civil Monetary Penalties Law by Employing Excluded
Individuals' which falls under the category 'Fraud Self-Disclosures'.
```

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

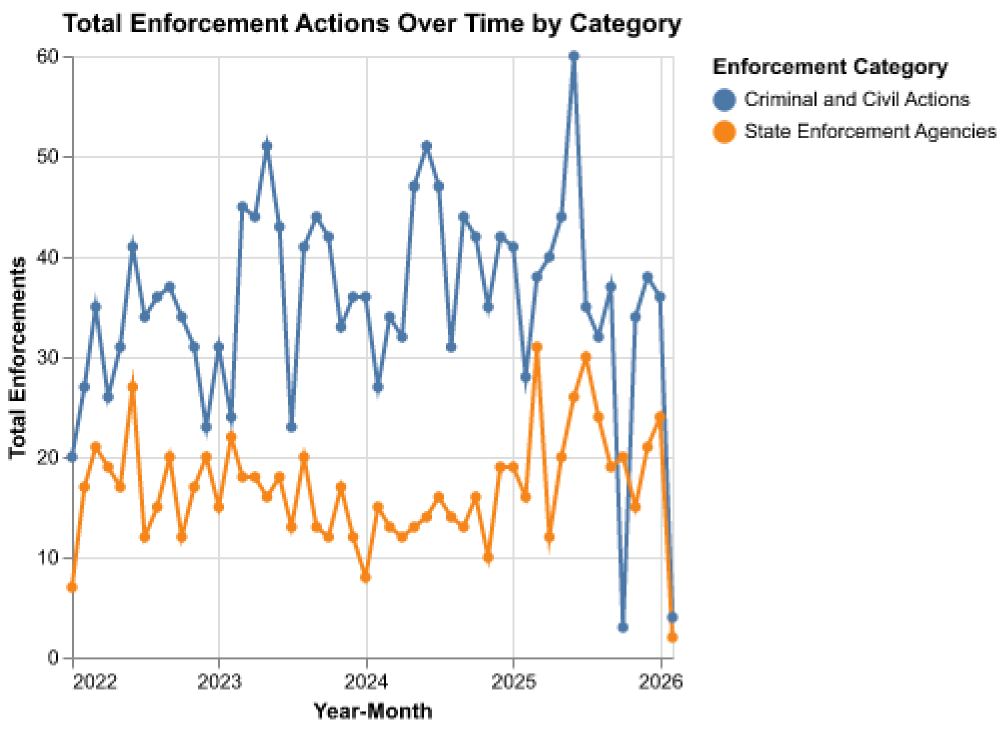
```python
df2["Date"] = pd.to_datetime(df2["Date"])
df2["Year_Month"] = df2["Date"].dt.to_period("M").dt.to_timestamp()
df2

df2_my = df2.groupby("Year_Month").size().reset_index(name="count")

df2_my_plot = alt.Chart(df2_my, title = "Number of Enforcement Actions Per
↪ Year-Month").mark_line(point=True).encode(
  alt.X("Year_Month:T", title = "Year-Month"),
  alt.Y("count:Q", title="Total Enforcements")
)
df2_my_plot
```



2. Plot the number of enforcement actions categorized:

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

```
df_categories = df2[df2["Category"].isin(["Criminal and Civil Actions",
↪  "State Enforcement Agencies"])]

df_categories_my = df_categories.groupby(["Year_Month",
↪  "Category"]).size().reset_index(name="count")

df_categories_my_plot = alt.Chart(df_categories_my, title = "Total
↪  Enforcement Actions Over Time by Category").mark_line(point=True).encode(
  alt.X("Year_Month:T", title = "Year-Month"),
  alt.Y("count:Q", title= "Total Enforcements"),
  alt.Color("Category:N", title= "Enforcement Category")
  )
df_categories_my_plot
```

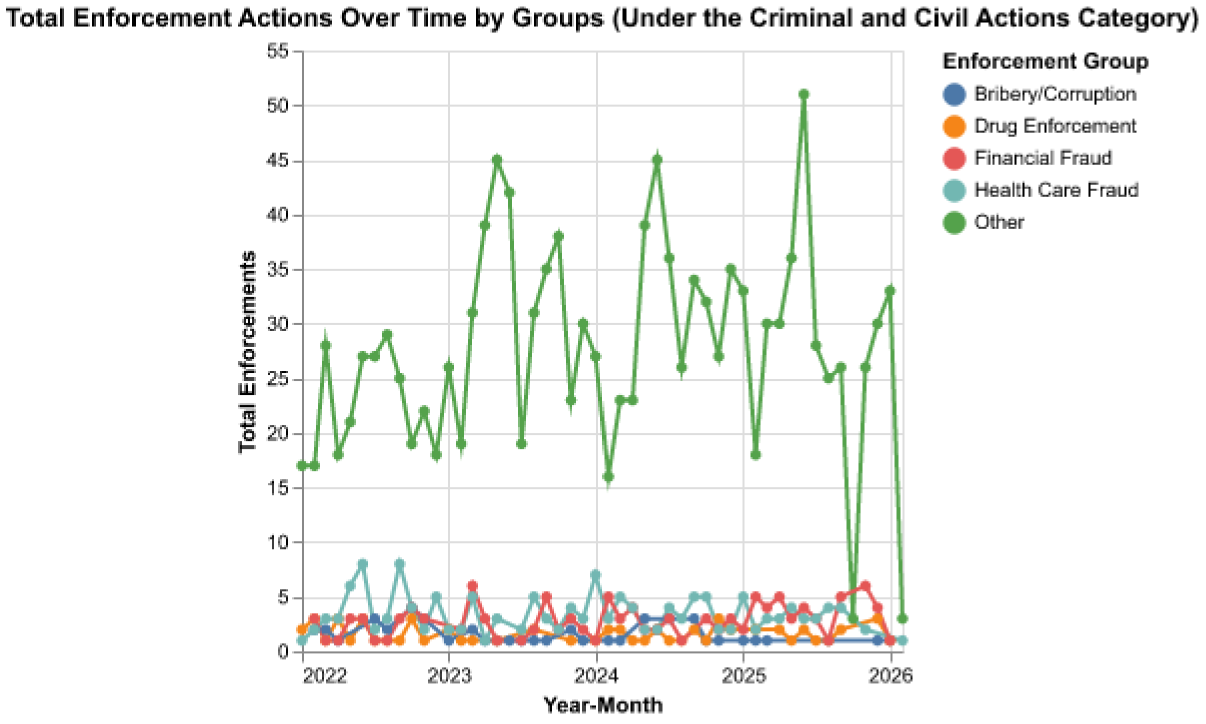


- based on five topics

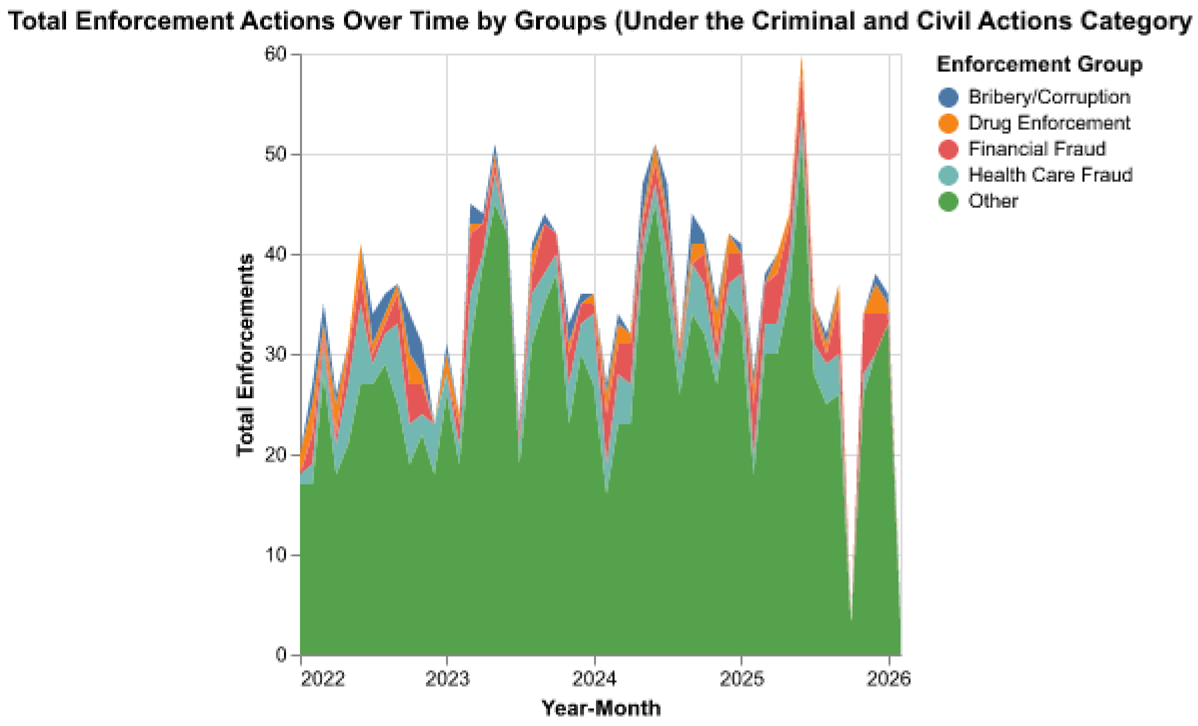
```
df2_criminal = df2[df2["Category"] == "Criminal and Civil Actions"]
df2_criminal

#In my summer internship, I used NLP for similar tasks, so I am employing NLP
 ↪  for this question.

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

title = df2_criminal["Enforcement Title"]
vector = TfidfVectorizer(stop_words="english")
title_vectors = vector.fit_transform(title)

title
title.to_excel("Titles.xlsx")

#Here, I gave ChatGPT the Title excel file, and asked it to build me a
 ↪  dictionary of relevant keywords for the four categories based on the
 ↪  pattern in the data. I haven't created an "Others" category dictionary
 ↪  here. My plan is to include those cases whose similarity score < 10 %
 ↪  under the "Others" category
```

```
keywords = {
    "Health Care Fraud": """
        medicare medicaid healthcare health care hospital doctor physician
 ↪  nurse clinic medical pharmacy
        prescription prescriptions billing telemedicine home health durable
 ↪  medical equipment dme tricare
        transplant lab testing covid testing covid test kickback
    """,
    "Financial Fraud": """
        wire fraud mail fraud bank fraud loan fraud ppp paycheck protection
 ↪  program covid relief fraud
        securities investment ponzi tax fraud irs identity theft credit card
 ↪  money laundering embezzlement
        bitcoin crypto forgery mortgage fraud insurance fraud
    """,
    "Drug Enforcement": """
        drug trafficking narcotics fentanyl heroin cocaine meth
 ↪  methamphetamine opioid distribution
        controlled substance intent to distribute cartel
    """,
    "Bribery/Corruption": """
        bribery bribe corruption extortion bid rigging public official honest
 ↪  services illegal payments
        kickback kickbacks contract steering
    """
}

topic_names = list(keywords.keys())
vectorized_topics = vector.transform(list(keywords.values()))

similarity = cosine_similarity(title_vectors, vectorized_topics)
best_idx = np.argmax(similarity, axis=1)
best_score = similarity[np.arange(similarity.shape[0]), best_idx]

df2_criminal["Group"] = [topic_names[i] for i in best_idx]
df2_criminal["Topic_Score"] = best_score

THRESHOLD = 0.10
df2_criminal.loc[df2_criminal["Topic_Score"] < THRESHOLD, "Group"] = "Other"
df2_criminal
df2_criminal["Group"].value_counts()
```

```
df2_criminal[df2_criminal["Group"] == "Other"]

df_groups_my = df2_criminal.groupby(["Year_Month",
 ↪  "Group"]).size().reset_index(name="count")

df_groups_my_plot = alt.Chart(df_groups_my, title = "Total Enforcement
 ↪  Actions Over Time by Groups (Under the Criminal and Civil Actions
 ↪  Category)").mark_line(point=True).encode(
  alt.X("Year_Month:T", title = "Year-Month"),
  alt.Y("count:Q", title= "Total Enforcements"),
  alt.Color("Group:N", title= "Enforcement Group")
  )
df_groups_my_plot

#Here, my others category might still have cases that actually belong to some
 ↪  other category. Because of time constraint, I wasn't able to further my
 ↪  refinement. But if I had more time, I would conduct n-gram analysis to
 ↪  create a list of recurring phrases, and with that I would build another
 ↪  dictionary for all the categories.
```



```
# Also, the question specifically asked for a line chart so I plotted that,
↪  but I believe there could be other strong ways to visualize the data.


df_groups_my_plot2 = alt.Chart(df_groups_my, title="Total Enforcement Actions
↪  Over Time by Groups (Under the Criminal and Civil Actions
↪  Category").mark_area().encode(
    alt.X("Year_Month:T", title = "Year-Month"),
    alt.Y("count:Q", title = "Total Enforcements"),
    alt.Color("Group:N", title= "Enforcement Group")
    )

df_groups_my_plot2
```


Total Enforcement Actions Over Time by Groups (Under the Criminal and Civil Actions Category
Total Enforcements
Year-Month
Enforcement Group
Bribery/Corruption
Drug Enforcement
Financial Fraud
Health Care Fraud
Other