Problem Set 4

Tiffany Tu

2026-02-07

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: TT

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
from datetime import datetime
import requests
from bs4 import BeautifulSoup

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/'
response = requests.get(url)

soup = BeautifulSoup(response.text, 'lxml')
soup.text[0:50]
```

```
'\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\nEnforcement Actions | Office '
```

```
# pulls the 20 search result entries
class_tag = soup.find_all('div', class_='usa-card__container')
print(len(class_tag))

display(class_tag[0:4])
```

```
20
```

```
[<div class="usa-card__container">
 <header class="usa-card__header">
 <h2 class="usa-card__heading">
 <a
 href="/fraud/enforcement/houston-transplant-doctor-indicted-for-making-false-statements-in-p
 Transplant Doctor Indicted For Making False Statements In Patients' Medical
 Records</a>
 </h2>
 <div class="font-body-sm margin-top-1">
```

```
<span class="text-base-dark padding-right-105">February 5, 2026</span><ul
class="display-inline add-list-reset"><li class="display-inline-block
usa-tag text-no-lowercase text-base-darkest bg-base-lightest
margin-right-1">Criminal and Civil Actions</li></ul>
</div>
</header>
</div>,
<div class="usa-card__container">
<header class="usa-card__header">
<h2 class="usa-card__heading">
<a
href="/fraud/enforcement/multicare-health-system-to-pay-millions-to-settle-fraud-case/">Mult
Health System to Pay Millions to Settle Fraud Case</a>
</h2>
<div class="font-body-sm margin-top-1">
<span class="text-base-dark padding-right-105">February 4, 2026</span><ul
class="display-inline add-list-reset"><li class="display-inline-block
usa-tag text-no-lowercase text-base-darkest bg-base-lightest
margin-right-1">Criminal and Civil Actions</li></ul>
</div>
</header>
</div>,
<div class="usa-card__container">
<header class="usa-card__header">
<h2 class="usa-card__heading">
<a
href="/fraud/enforcement/brooklyn-banker-pleads-guilty-to-laundering-proceeds-of-medicare-fr
Banker Pleads Guilty to Laundering Proceeds of Medicare Fraud for
Transnational Criminal Organization</a>
</h2>
<div class="font-body-sm margin-top-1">
<span class="text-base-dark padding-right-105">February 3, 2026</span><ul
class="display-inline add-list-reset"><li class="display-inline-block
usa-tag text-no-lowercase text-base-darkest bg-base-lightest
margin-right-1">COVID-19</li></ul>
</div>
</header>
</div>,
<div class="usa-card__container">
<header class="usa-card__header">
<h2 class="usa-card__heading">
```

```
 <a
 href="/fraud/enforcement/delafield-man-sentenced-to-18-months-imprisonment-for-conspiracy-to
 Man Sentenced to 18 Months' Imprisonment for Conspiracy to Pay Health Care
 Kickbacks</a>
 </h2>
 <div class="font-body-sm margin-top-1">
 <span class="text-base-dark padding-right-105">February 3, 2026</span><ul
 class="display-inline add-list-reset"><li class="display-inline-block
 usa-tag text-no-lowercase text-base-darkest bg-base-lightest
 margin-right-1">Criminal and Civil Actions</li></ul>
 </div>
 </header>
 </div>]
```

```
data_list = []

for result in class_tag:
  header = result.find('header', class_='usa-card__header')

  link_tag = header.find('a')
  href = link_tag.get('href')
  title = link_tag.get_text(strip=True)

  date = header.find('span', class_='text-base-dark
 ↪  padding-right-105').get_text(strip=True)

  category = header.find('li', class_='usa-tag').get_text(strip=True)

  data_list.append({
      'Enforcement Action': title,
      'Date': date,
      'Category': category,
      'Link': href
  })


display(data_list[0:2])
```

```
[{'Enforcement Action': 'Houston Transplant Doctor Indicted For Making False
Statements In Patients' Medical Records',
  'Date': 'February 5, 2026',
  'Category': 'Criminal and Civil Actions',
```

```
  'Link':
  '/fraud/enforcement/houston-transplant-doctor-indicted-for-making-false-statements-in-patie
 {'Enforcement Action': 'MultiCare Health System to Pay Millions to Settle
 Fraud Case',
  'Date': 'February 4, 2026',
  'Category': 'Criminal and Civil Actions',
  'Link':
  '/fraud/enforcement/multicare-health-system-to-pay-millions-to-settle-fraud-case/'}]
```

```
data_df = pd.DataFrame(data_list)
#display(data_df.head())

data_df['Date'] = pd.to_datetime(data_df['Date'])
```

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code

DEFINE function scrape_enforcement_actions(start_month, start_year):

IF start_year < 2013: PRINT "Please enter a date in or after 2013" RETURN None

SET cutoff_date = first day of start_month, start_year

INITIALIZE empty list to store scraped data

SET page = 0 SET keep_scraping = True

WHILE keep_scraping is True: CREATE url for page REQUEST page PARSE html with BeautifulSoup FIND all enforcement action containers on page

FOR each container: EXTRACT title, link, date, category CONVERT date to datetime IF date >= cutoff_date: ADD record to list of scraped results ELSE: SET keep_scraping = False BREAK out of loop

INCREMENT page by 1 SLEEP for 1 second

CONVERT results list to DataFrame SAVE DataFrame to CSV RETURN DataFrame

A simple for loop may not work for this function because that number of pages to scrape may vary, so instead I am using a while loop which will continue to run or scrape as long as the conditions hold true.

- b. Create Dynamic Scraper

```
def scrape_enforcement_actions(start_month, start_year):
  if start_year < 2013:
    print("Please enter a date in or after 2013")
    return None

  cutoff_date = datetime(start_year, start_month, 1)

  data_list = []
  base_url = 'https://oig.hhs.gov'
  page = 0
  keep_scraping = True

  while keep_scraping:
    url = f'{base_url}/fraud/enforcement/?page={page}'
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'lxml')
    class_tag = soup.find_all('div', class_='usa-card__container')

    for result in class_tag:
      header = result.find('header', class_='usa-card__header')

      link_tag = header.find('a')
      href = link_tag.get('href')
      title = link_tag.get_text(strip=True)

      date_str = header.find('span', class_='text-base-dark
↪  padding-right-105').get_text(strip=True)
      date = pd.to_datetime(date_str)

      category = header.find('li', class_='usa-tag').get_text(strip=True)

      if date >= cutoff_date:
        data_list.append({
            'Enforcement Action': title,
            'Date': date,
            'Category': category,
            'Link': href
        })
      else:
        keep_scraping = False
        break

    page += 1
```

```
    time.sleep(1)

  data_df = pd.DataFrame(data_list)
  data_df.to_csv(f'enforcement_actions_{start_year}_{start_month}.csv',
↪  index=False)

  return data_df
```

```
RUN_FUNC = False
if RUN_FUNC:
  df_2024_01 = scrape_enforcement_actions(1, 2024)

df_2024_01 = pd.read_csv('enforcement_actions_2024_1.csv')
```

```
print(len(df_2024_01))
display(df_2024_01[df_2024_01['Date'] == df_2024_01['Date'].min()])
```

1807

	Enforcement Action	Date	Category	Link
1805	Laredo Resident Admits To Impersonating Licens...	2024-01-03	Criminal and Civil Actions	/frauc
1806	Former Nurse Aide Indicted In Death Of Clarksv...	2024-01-03	State Enforcement Agencies	/frauc

- c. Test Your Code

```
RUN_FUNC = False
if RUN_FUNC:
  df_2022_01 = scrape_enforcement_actions(1, 2022)

df_2022_01 = pd.read_csv('enforcement_actions_2022_1.csv')
```

```
print(len(df_2022_01))
display(df_2022_01[df_2022_01['Date'] == df_2022_01['Date'].min()])
```

3397

	Enforcement Action	Date	Category	Link
3394	Central Medical Systems, LLC, Alan Trent Harle...	2022-01-04	Criminal and Civil Actions	/frau

	Enforcement Action	Date	Category	Link
3395	Ohio home healthcare provider agrees to pay $5...	2022-01-04	Criminal and Civil Actions	/frau
3396	Integrated Pain Management Medical Group Agree...	2022-01-04	Fraud Self-Disclosures	/frau

Step 3: Plot data based on scraped data

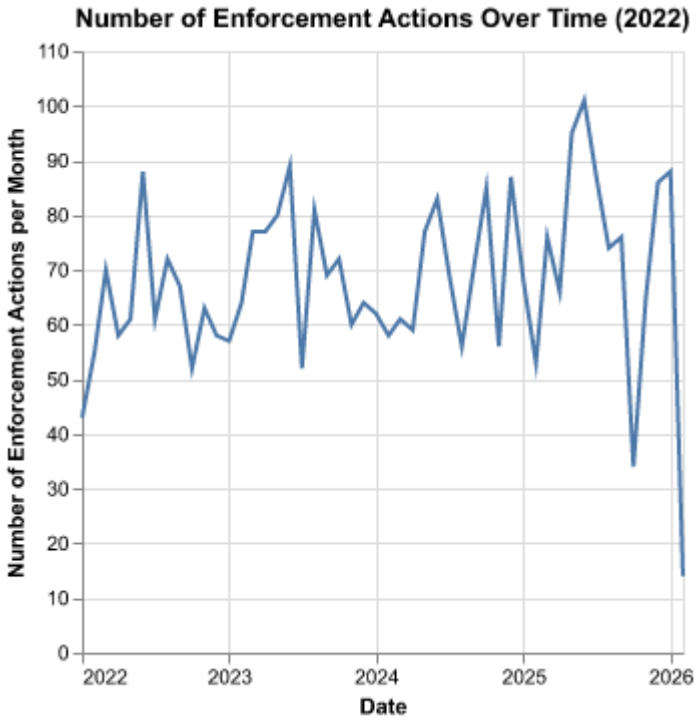
1. Plot the number of enforcement actions over time

```
df_2022_01['Date'] = pd.to_datetime(df_2022_01['Date'])
df_2022_01['yearmonth'] =
 ↪  df_2022_01['Date'].dt.to_period('M').dt.to_timestamp()

df_monthly = df_2022_01.groupby('yearmonth').size().reset_index(name='count')
```

```
line_chart = alt.Chart(df_monthly).mark_line().encode(
  x=alt.X('yearmonth:T', title='Date', axis=alt.Axis(format='%Y')),
  y=alt.Y('count:Q', title='Number of Enforcement Actions per Month')
).properties(
  title='Number of Enforcement Actions Over Time (2022)'
)

line_chart
```



2. Plot the number of enforcement actions categorized:

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

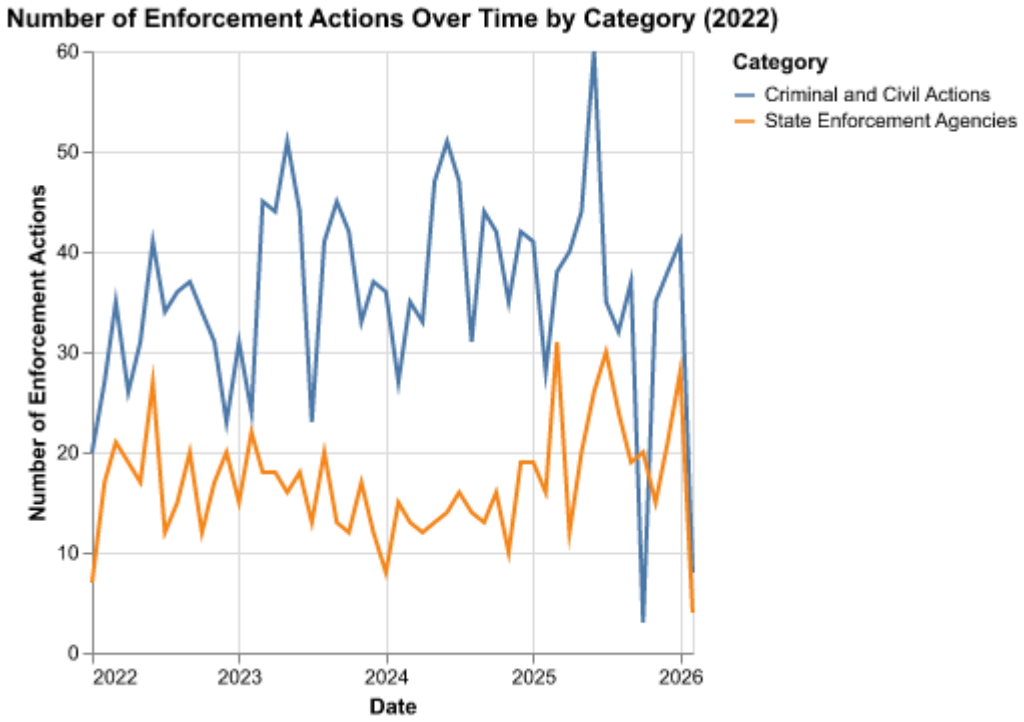
```
filtered_categories = ['Criminal and Civil Actions', 'State Enforcement
 ↪  Agencies']

df_filtered = df_2022_01[df_2022_01['Category'].isin(filtered_categories)]

df_comparison = df_filtered.groupby(['yearmonth',
 ↪  'Category']).size().reset_index(name='count')
```

```
comparison_chart = alt.Chart(df_comparison).mark_line().encode(
 x=alt.X('yearmonth:T', title='Date', axis=alt.Axis(format='%Y%')),
 y=alt.Y('count:Q', title='Number of Enforcement Actions'),
 color=alt.Color('Category:N', title='Category')
).properties(
  title='Number of Enforcement Actions Over Time by Category (2022)'
)
```

```
comparison_chart
```



- based on five topics

```
df_cc = df_2022_01[df_2022_01['Category'] == 'Criminal and Civil
 ↪  Actions'].copy()
```

```
import re
def clean_title(title):
    title = title.lower()
    title = re.sub(r'[^a-z\s]', '', title)
    return title.split()

df_cc['Keywords'] = df_cc['Enforcement Action'].apply(clean_title)
```

```
from collections import Counter

all_keywords = [
  word
```

```
  for keywords in df_cc['Keywords']
  for word in keywords
]

stopwords = {
    'the', 'and', 'of', 'for', 'in', 'to', 'with',
    'on', 'by', 'at', 'from', 'after', 'case'
}

filtered_counts = Counter(
    word for word in all_keywords if word not in stopwords
)

filtered_counts.most_common(20)
```

```
[('fraud', 754),
 ('health', 581),
 ('care', 518),
 ('million', 491),
 ('sentenced', 430),
 ('pay', 411),
 ('scheme', 402),
 ('claims', 386),
 ('false', 384),
 ('allegations', 322),
 ('guilty', 262),
 ('act', 262),
 ('medicare', 242),
 ('resolve', 231),
 ('prison', 217),
 ('pleads', 204),
 ('doctor', 192),
 ('medical', 192),
 ('man', 186),
 ('agrees', 178)]
```

```
def assign_topic(title):
    title = title.lower()

    if any(word in title for word in [
        'health', 'care', 'medicare', 'medicaid', 'medical',
        'doctor', 'physician', 'hospital', 'clinic', 'claims'
```

```
    ]):
        return 'Health Care Fraud'

    elif any(word in title for word in [
        'drug', 'opioid', 'fentanyl', 'controlled substance',
        'pharmaceutical', 'distribution'
    ]):
        return 'Drug Enforcement'

    elif any(word in title for word in [
        'bank', 'financial', 'loan', 'wire', 'money',
        'tax', 'credit', 'scheme'
    ]):
        return 'Financial Fraud'

    elif any(word in title for word in [
        'bribe', 'bribery', 'corruption', 'kickback',
        'embezzle', 'embezzlement'
    ]):
        return 'Bribery/Corruption'

    else:
        return 'Other'
```

```
df_cc['Topic'] = df_cc['Enforcement Action'].apply(assign_topic)
```

```
df_topic_monthly = (
    df_cc
    .groupby(['yearmonth', 'Topic'])
    .size()
    .reset_index(name='count')
)
```

```
topic_line_chart = alt.Chart(df_topic_monthly).mark_line().encode(
    x=alt.X('yearmonth:T', title='Date'),
    y=alt.Y('count:Q', title='Number of Enforcement Actions'),
    color=alt.Color('Topic:N', title='Topic'),
    tooltip=['yearmonth', 'Topic', 'count']
).properties(
    title='Enforcement Actions Over Time by Topic (Criminal and Civil
 ↪  Actions)',
```

```
    width=600,
    height=400
)

topic_line_chart
```