# Problem Set 4

Wyatt Hartwig

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: SWH

## 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 numpy as np
import pandas as pd
import altair as alt
import time
import vl_convert as vlc
from IPython.display import Image, display
import requests
from datetime import datetime
import calendar
from urllib.parse import urljoin
import csv
from bs4 import BeautifulSoup as bs
import os
os.chdir('C:\\Users\\wyatt\\student30538-w26\\ps4-whartwig')

import warnings
warnings.filterwarnings('ignore')
#alt.renderers.enable("png")
alt.data_transformers.disable_max_rows()

#This function will work with (alt.Chat + alt.Chart) as well
def display_to_pdf(altchart, scale=2):
    png_bytes = vlc.vegalite_to_png(altchart.to_dict(), scale=scale)
    display(Image(png_bytes))
```

### Step 1: Develop initial scraper and crawler

```
with open('webpage.html', 'w') as page:
  text = requests.get('https://oig.hhs.gov/fraud/enforcement/').text

soup = bs(text, 'lxml')
```

(PSeudo code made after the fact with AI assistance as I missed the requierment to write pseudo code for part a. PSeudo code in question 2 is my own written and structured pseudo code.)

```
START
Initialize: Set the starting URL, an empty list for events, and a page
 ↪  counter to 0.
```

```
Repeat while a URL exists AND the page counter is less than 10:
  Fetch: Download the webpage content.
  Find: Identify all "event cards" on the current page.
  For each card found:
    Extract: Pull out the Title, Hyperlink, and Date.
  Filter Categories:
    Collect all category tags for the card.
    Remove "COVID-19" from that list.
    If a category remains, pick the first one; otherwise, label it "Other."
  Save: Add the gathered info (Name, Category, Date, Link) to the event list.
  Find Pagination: Look for a "Next" button.
  Update:
    If "Next" exists: Update the URL to the next page and add 1 to the page
 ↪  counter.
    If "Next" is missing: Clear the URL to stop the loop.
END
```

```python
#First instance of the list of events
#soup.find_all(lambda tag: tag.name == "li" and tag.find('a'))[81]

#We will need this root to find html links
root = 'https://oig.hhs.gov/fraud/enforcement/'
current_url = 'https://oig.hhs.gov/fraud/enforcement/'
event_log = []
count = 0
while current_url:
    if count == 10:
      break
    response = requests.get(current_url)
    soup = bs(response.text, 'lxml')

    #Scrape current page
    cards = soup.find_all('li', class_='usa-card')

    for card in cards:
      a_tag = card.find('h2', class_='usa-card__heading').find('a')

      title = a_tag.get_text(strip=True)
      link = urljoin(root, a_tag['href'])

      date = card.find('span', class_='text-base-dark').get_text(strip=True)
    #We have to filter out the COVID-19 category doubles
    cat_tags = card.find_all('li', class_='usa-tag')
    all_cat = [c.get_text(strip=True) for c in cat_tags]
    cat = [c for c in all_cat if c != 'COVID-19']
    cat = cat[0] if cat else 'Other'

    event_log.append({'Name': title, 'Category': cat, 'Date': date, 'Link':
↪   link})

# 2. Find the "Next" button link
next_button = soup.find('a', class_='pagination-next')

if next_button and 'href' in next_button.attrs:
    # Update current_url to the next page's link
    current_url = urljoin(root, next_button['href'])
    count += 1
else:
    current_url = None
```

```
event_log = pd.DataFrame(event_log)
event_log.to_csv('event_log.csv', index=False)
```

```
event_log = pd.read_csv('event_log.csv')
```

```
event_log
```

| | Name | Category | Date |
|---|---|---|---|
| 0 | Brooklyn Banker Pleads Guilty to Laundering Pr... | Other | February 3, 2026 |
| 1 | Delafield Man Sentenced to 18 Months' Imprison... | Criminal and Civil Actions | February 3, 2026 |
| 2 | Former NFL Player Convicted for $197M Medicare... | Criminal and Civil Actions | February 3, 2026 |
| 3 | AG's Office Secures Indictments Against Peabod... | State Enforcement Agencies | February 2, 2026 |
| 4 | Florida Man Pleads Guilty to Conspiracy to Vio... | Criminal and Civil Actions | January 30, 2026 |
| ... | ... | ... | ... |
| 195 | Baltimore County Woman Sentenced For Impersona... | Criminal and Civil Actions | November 13, 202 |
| 196 | Alabama Doctor Pleads Guilty To $6 Million Tel... | Criminal and Civil Actions | November 13, 202 |
| 197 | Repeat Offender Pleads Guilty To Health Care F... | Criminal and Civil Actions | November 13, 202 |
| 198 | ICYMI: Charges, Pleas, Sentencings And Settlem... | Criminal and Civil Actions | November 13, 202 |
| 199 | Bradenton Woman Indicted For Passport Fraud An... | Criminal and Civil Actions | November 13, 202 |

## Step 2: Making the scraper dynamic

### 1. Turning the scraper into a function

- a. Pseudo-Code

```
BEGIN FUNCTION
crawler_function(month, year):
  BEGIN INDICATOR
  if year < 2013:
    end function and print a message to the user to choose another date
  END INDICATOR

  root = underlying url link for the first web page
  current_url = link to first page of content (same as above, but will be
↪ used to find next page)
  event_log = empty list for storing resutls

  target_date = reformat user input with datetime to use in the loop

  BEGIN WHILE LOOP
  while current_url:
    response = use requests package to access the webpage and get html
    soup = create beautifulsoup object that houses the html as plain text
↪ accessible

    cards = scrape all the cases (case cards) from the indexible soup object

    BEGIN MAIN FOR LOOP
    for card in cards:
      BEGIN DATE CHECK LOOP
        scrape_date = get the plain text of the date
        card_date = convert the scrape_date to machine readable datetime

        BEGIN COMP IF STMT
        if card_date < target_date:
          print a message stating the input date was reached
          reset the scrapping indicator to True
          break the loop outer loop
        END COMP IF STMT
      END DATE CHECK LOOP

      a_tag = use the .find on the card to find the main tag and associated
↪ class
```

```
      title = recover the title in plain text of the event from the card

      link = root + pull the url from the a_tag card with 'href'

      date = scpare the date based on the relevant tag and class

      BEGIN FILTER (here we must fix the COVID-19 doubles)
      pull all tags for the categories in the card
      use a list comprehension to subset the aobve to the plain text
      set the category to a list (using a comprehension) that does not
 ↪  include 'COVID-19'
      cat = the first item in the resulting list (should be a string object)
      END FILTER

      event_log = append all the above items using a dictionary to the
 ↪  event_log

      BEGIN FIND NEXT PAGE
      next_button = use .find to pull the tag and class combonation for the
 ↪  page that signals the next button
      END FIND NEXT PAGE

      BEGIN NEXT BUTTON IF LOGIC
        next_button and the link in 'href' class are in next_button
 ↪  attribute:
          reset current_url to equal the root url plus the url found in the
 ↪  next_button 'href'
        else (if there is not next_button):
          set current_url equal to None so the scraper stops
      END NEXT BUTTON IF LOGIC
    END MAIN FOR LOOP
  END OUTER WHILE LOOP

  use csv module to save the resulting event_log as a csv within the current
 ↪  directory
END FUNCTION
```

- b. Create Dynamic Scraper

```python
def get_enforc_actions(month, year):
  if int(year) < 2013:
    return print(f'Your selection of {year} is too far back, please select a
     ↪  year from 2013 or later.')

  root = 'https://oig.hhs.gov/fraud/enforcement/'
  current_url = 'https://oig.hhs.gov/fraud/enforcement/'
  event_log = []

  target_date = datetime.strptime(f'{month} 1, {year}', '%B %d, %Y')

  stop_scraping = False

  while current_url and not stop_scraping:
    #pull content from the webpage
    response = requests.get(current_url)
    soup = bs(response.text, 'lxml')

    #Scrape current page
    cards = soup.find_all('li', class_='usa-card')

    #Check the date andf set the date
    for card in cards:
      scraped_date = card.find('span',
↪   class_='text-base-dark').get_text(strip=True)
      card_date = datetime.strptime(scraped_date, "%B %d, %Y")

      if card_date < target_date:
        print(f"{target_date} reached and {len(event_log)} events pulled")
        stop_scraping = True
        break

      #Pull the title and link from the card
      a_tag = card.find('h2', class_='usa-card__heading').find('a')
      title = a_tag.get_text(strip=True)
      link = root + a_tag['href']

      #Find all the category labels and then filter against COVID-19 doubles
      cat_tags = card.find_all('li', class_='usa-tag')
      all_cat = [c.get_text(strip=True) for c in cat_tags]
      cat = [c for c in all_cat if c != 'COVID-19']
      cat = cat[0] if cat else 'Unknown'

      #Append pieces to event_log
      event_log.append({'Name': title, 'Category': cat,
                        'Date': scraped_date,
                        'Link': link})

    #Find the "Next" button link and see if we need to continue
    if not stop_scraping:
      next_button = soup.find('a', class_='pagination-next')
      if next_button and 'href' in next_button.attrs:
        #Update current_url to the next page's link to get to next page
        current_url = urljoin(root, next_button['href'])
        time.sleep(0.5)
      else:
        current_url = None

  #Convert and export data
  event_log = pd.DataFrame(event_log)
  event_log.to_csv(f'enforcement_actions_{year}_{month}.csv', index=False)
  return f'Pulled {len(event_log)} items from the webpage'
```

```
get_enforc_actions('January', 2024)
pd.read_csv('enforcement_actions_2024_January.csv').iloc[-1,:][0]
```

The data frame returns 1,769 events (as of 04FEB26). The earliest event was a State Enforcement Agencies event on 03 January 2024 where a former nurse aide indicted in the death of a Clarksville patient was arrested in Georgia.

- c. Test Your Code

```
get_enforc_actions('January', 2022)
```

```
jan2022 = pd.read_csv('enforcement_actions_2022_January.csv')
jan2022.iloc[-1,:]
```

```
Name        Integrated Pain Management Medical Group Agree...
Category                                Fraud Self-Disclosures
Date                                           January 4, 2022
Link        https://oig.hhs.gov/fraud/enforcement//fraud/e...
Name: 3358, dtype: object
```

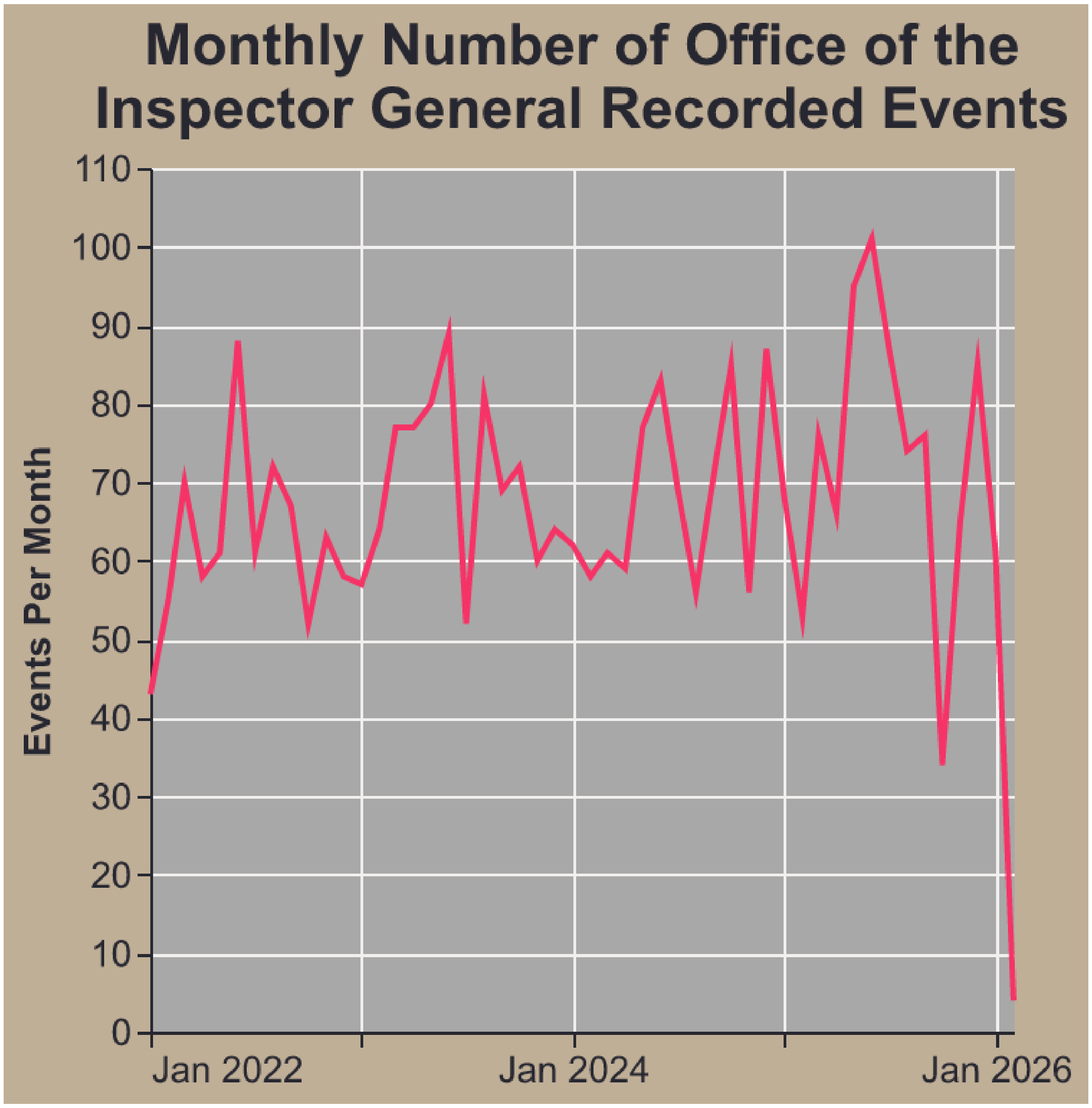
The function returns a data frame with 3,359 events (as of 04FEB26). The oldest event is a Fraud Self-Disclosures occuring on 04 January 2022 in which the (an?) Integrated Pain Management Medical Group agreed to pay $10,000 for Allegedly Violating the Civil Monetary Penalties Law by Employing Excluded Individuals.

## Step 3: Plot data based on scraped data

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

```
ThemeRegistry.enable('my_theme')
```

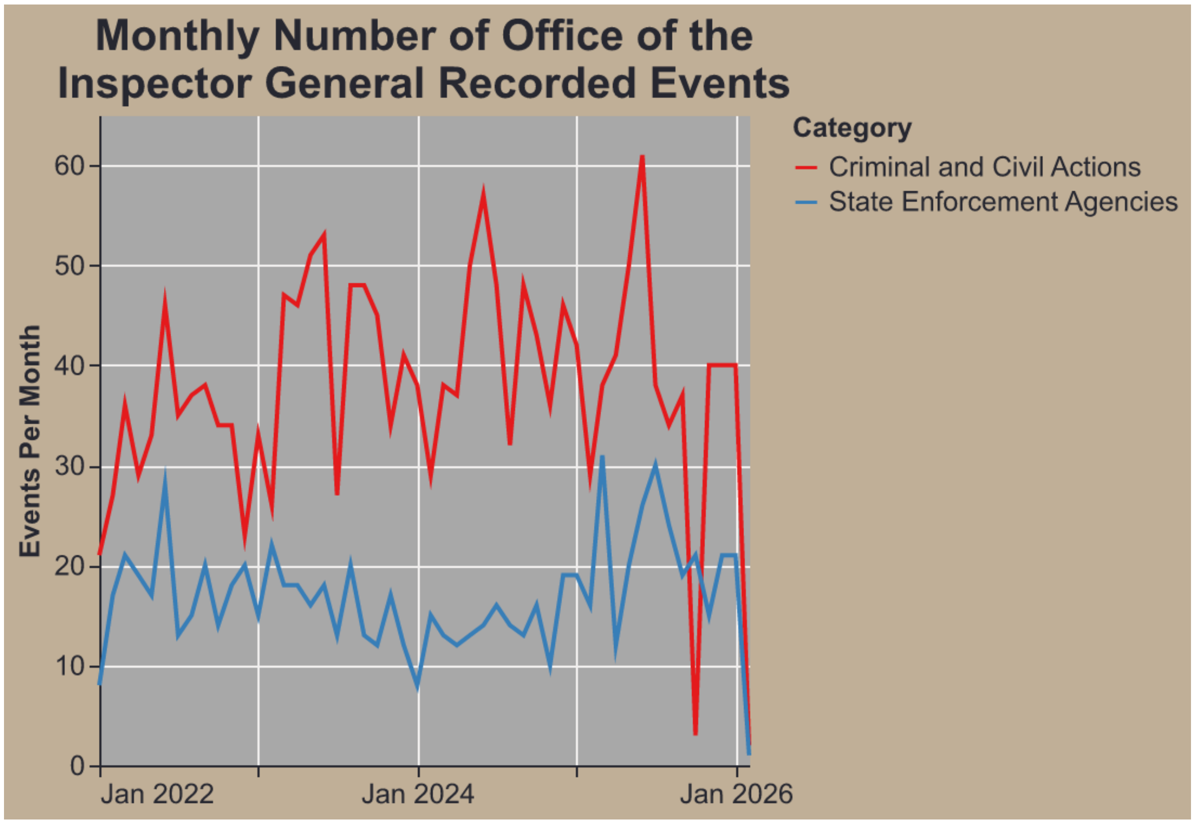
```
line_plot = alt.Chart(jan2022, title='Monthly Number of Office of the\n
↪  Inspector     General Recorded Events').mark_line().encode(
  alt.X('yearmonth(Date):T', title=''),
  alt.Y('count()', title='Events Per Month')
)
display_to_pdf(line_plot)
```



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

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

```
crim_state = alt.Chart(jan2022, title='Monthly Number of Office of the\n
↪  Inspector     General Recorded Events').mark_line().encode(
  alt.X('yearmonth(Date):T', title=''),
  alt.Y('count()', title='Events Per Month'),
  alt.Color('Category').legend(labelLimit=400)
).transform_filter("datum.Category == 'Criminal and Civil Actions' ||
↪  datum.Category == 'State Enforcement Agencies' ")
display_to_pdf(crim_state)
```

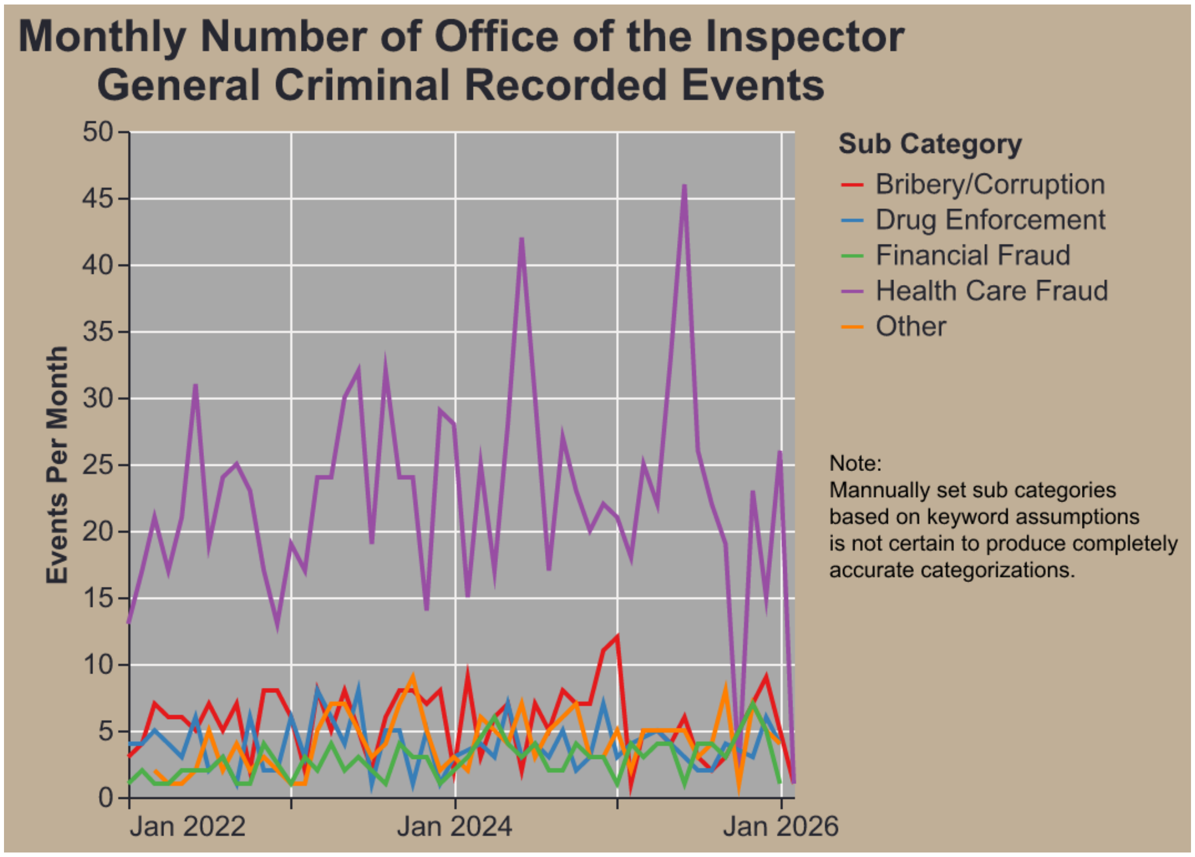


- based on five topics

```
topics = jan2022.copy()
topics = topics[topics['Category']=='Criminal and Civil Actions']
case_types = ["Health Care Fraud", "Financial Fraud", "Drug Enforcement",
              "Bribery/Corruption",  "Other"]
```

```
conditions = [
  #Drug Keywords

↪  topics['Name'].str.contains('Dealer|Distribut|trafficking|controlled|substance|opiod|narc
↪  prescrib', case=False, na=False),
  #Bribery/corruption keywords

↪  topics['Name'].str.contains('kickback|bribe|corruption|unnecessary|solicit',
↪  case=False, na=False),
  #Finance keywords
  topics['Name'].str.contains('bank|wire|tax|financial|money|theft|embez',
↪  case=False, na=False),
  #Health keywords

↪  topics['Name'].str.contains('health|medicine|medicare|medical|healthcare|lab|practice|CO
↪  case=False, na=False),
]

choices = ['Drug Enforcement',  'Bribery/Corruption', 'Financial Fraud',
           'Health Care Fraud']

topics['Sub Category'] = np.select(conditions, choices, default='Other')
```

```
five_line = alt.Chart(topics, title='Monthly Number of Office of the
↪  Inspector\n General Criminal Recorded Events').mark_line().encode(
  alt.X('yearmonth(Date):T', title=''),
  alt.Y('count()', title='Events Per Month'),
  alt.Color('Sub Category').legend(labelLimit=400)
)
footer = alt.Chart().mark_text(
    align='left', baseline='middle', fontSize=10, color='black',
    dx=166
).encode(
    text=alt.value("Note:\nMannually set sub categories\n based on keyword
↪  assumptions\n is not certain to produce completely\n accurate
↪  categorizations.")
)
display_to_pdf(footer + five_line)
```


Monthly Number of Office of the Inspector General Criminal Recorded Events
Events Per Month
50
45
40
35
30
25
20
15
10
5
0
Jan 2022
Jan 2024
Jan 2026
Sub Category
Bribery/Corruption
Drug Enforcement
Financial Fraud
Health Care Fraud
Other
Note:
Mannually set sub categories
based on keyword assumptions
is not certain to produce completely
accurate categorizations.