# ps4

Byeol Choi

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: BC **___**

### 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

import warnings
warnings.filterwarnings('ignore')
alt.renderers.enable("png")
```

```
RendererRegistry.enable('png')
```

### Step 1: Develop initial scraper and crawler

```
from bs4 import BeautifulSoup
import requests

base_url = "https://oig.hhs.gov/fraud/enforcement/"
resp = requests.get(base_url)
resp.raise_for_status()

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

rows = []

for li in soup.find_all("li"):
    h2 = li.find("h2")
    if not h2:
        continue

    a = h2.find("a", href=True)
    if not a:
        continue

    title = a.get_text(strip=True)
    link = "https://oig.hhs.gov" + a["href"]

    # assume date is the first text line
    text_lines = li.get_text("\n", strip=True).split("\n")
    date = text_lines[0] if len(text_lines) > 0 else None

    if "Criminal and Civil Actions" in li.get_text():
        category = "Criminal and Civil Actions"
```

```
    elif "State Enforcement Agencies" in li.get_text():
        category = "State Enforcement Agencies"
    else:
        category = "Other"

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

df_step1 = pd.DataFrame(rows)
df_step1.head()
```

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

## Step 2: Making the scraper dynamic

### 1. Turning the scraper into a function

- a. Pseudo-Code

FUNCTION scrape_enforcement_actions(start_year, start_month, run_scraper):

0) Guardrails and "run" indicator

IF start_year < 2013: PRINT "Please restrict to year >= 2013 (only post-2013 actions are listed)." RETURN None

IF run_scraper IS FALSE: PRINT "run_scraper=False: skipping scraping; use the saved CSV instead." RETURN None

1) Setup

SET start_date = first day of (start_year, start_month) SET base_url = "https://oig.hhs.gov/fraud/enforceme
SET page_num = 1 SET results = empty list SET stop_flag = FALSE

2) Crawl pages until stop condition is met

WHILE stop_flag IS FALSE:

```
IF page_num == 1:
    SET page_url = base_url
ELSE:
    SET page_url = base_url + "?page=" + page_num

REQUEST html at page_url
PARSE html into a document object

EXTRACT all enforcement action entries on this page
IF number of extracted entries == 0:
    BREAK  # stop condition: no entries means nothing left to crawl

3) Parse and store entries from the current page

FOR EACH entry in extracted entries:

    EXTRACT title
    EXTRACT link (convert to absolute URL if needed)
    EXTRACT date text and convert to a date object (or keep as string
    initially)
    EXTRACT category

    IF entry_date < start_date:
        SET stop_flag = TRUE
        BREAK out of FOR loop  # stop condition: reached older-than-start
        window

    APPEND (title, date, category, link) to results


4) Be polite to the server, then go to next page

WAIT 1 second
INCREMENT page_num by 1
```

5) Output

CONVERT results into a tidy dataframe (one row per enforcement action) SAVE dataframe to “enforcement_actions_{start_year}_{start_month}.csv” RETURN dataframe

END FUNCTION

- b. Create Dynamic Scraper

```
from datetime import datetime, date as dt_date

def scrape_enforcement_actions(start_year: int,
                               start_month: int,
                               run_scraper: bool = True) -> pd.DataFrame |
 ↪  None:

    if start_year < 2013:
        print("Please restrict to year >= 2013 (only post-2013 enforcement
         ↪  actions are listed).")
        return None

    if not run_scraper:
        print("run_scraper=False: skipping scraping (use the saved CSV
         ↪  instead).")
        return None

    if start_month < 1 or start_month > 12:
        raise ValueError("start_month must be between 1 and 12.")

    start_date = date(start_year, start_month, 1)
    base_url = "https://oig.hhs.gov/fraud/enforcement/"
    domain_prefix = "https://oig.hhs.gov"

    rows = []
    page_num = 1
    stop_flag = False

    def _parse_date_from_lines(lines):
        for ln in lines:
            ln = ln.strip()
            try:
                return datetime.strptime(ln, "%B %d, %Y").date()
            except ValueError:
                continue
        return None

    while not stop_flag:
        page_url = base_url if page_num == 1 else
 ↪  f"{base_url}?page={page_num}"
```

```
        resp = requests.get(page_url, timeout=30)
        resp.raise_for_status()

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

        candidate_items = []
        for li in soup.find_all("li"):
            h2 = li.find("h2")
            if h2 is None:
                continue
            a = h2.find("a", href=True)
            if a is None:
                continue
            candidate_items.append(li)

        if len(candidate_items) == 0:
            break

        for li in candidate_items:
            h2 = li.find("h2")
            a = h2.find("a", href=True)

            title = a.get_text(strip=True)
            href = a["href"].strip()
            link = href if href.startswith("http") else domain_prefix + href

            text_lines = li.get_text("\n", strip=True).split("\n")
            action_date = _parse_date_from_lines(text_lines)

            if action_date is None:
                continue

            if action_date < start_date:
                stop_flag = True
                break

            blob = " ".join(text_lines)
            if "Criminal and Civil Actions" in blob:
                category = "Criminal and Civil Actions"
            elif "State Enforcement Agencies" in blob:
                category = "State Enforcement Agencies"
            else:
                category = "Other"
```

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

        time.sleep(1)
        page_num += 1

    df = pd.DataFrame(rows)

    if not df.empty:
        df = df.sort_values("date", ascending=False).reset_index(drop=True)

    out_name = f"enforcement_actions_{start_year}_{start_month:02d}.csv"
    df.to_csv(out_name, index=False)

    print(f"Saved: {out_name}")
    print(f"Rows collected: {len(df)}")
    if not df.empty:
        print("Earliest scraped action date:", df["date"].min())

    return df
```

- c. Test Your Code

```
df_2022 = scrape_enforcement_actions(
    start_year=2022,
    start_month=1,
    run_scraper=True
)
```

```
CSV_NAME = "enforcement_actions_2022_01.csv"

df_2022 = pd.read_csv(CSV_NAME)
df_2022["date"] = pd.to_datetime(df_2022["date"], errors="coerce")
df_2022 = df_2022.dropna(subset=["date"]).copy()

n_actions = len(df_2022)
```

```
earliest_row = df_2022.sort_values("date", ascending=True).iloc[0]

print("Number of enforcement actions (since Jan 2022):", n_actions)
print("Earliest enforcement action scraped:")
print("  date:", earliest_row["date"].date())
print("  title:", earliest_row["title"])
print("  category:", earliest_row["category"])
print("  link:", earliest_row["link"])
```

```
Number of enforcement actions (since Jan 2022): 3377
Earliest enforcement action scraped:
  date: 2022-01-04
  title: Integrated Pain Management Medical Group Agreed to Pay $10,000 for
  Allegedly Violating the Civil Monetary Penalties Law by Employing Excluded
  Individuals
  category: Other
  link:
  https://oig.hhs.gov/fraud/enforcement/integrated-pain-management-medical-group-agreed-to-pa
```

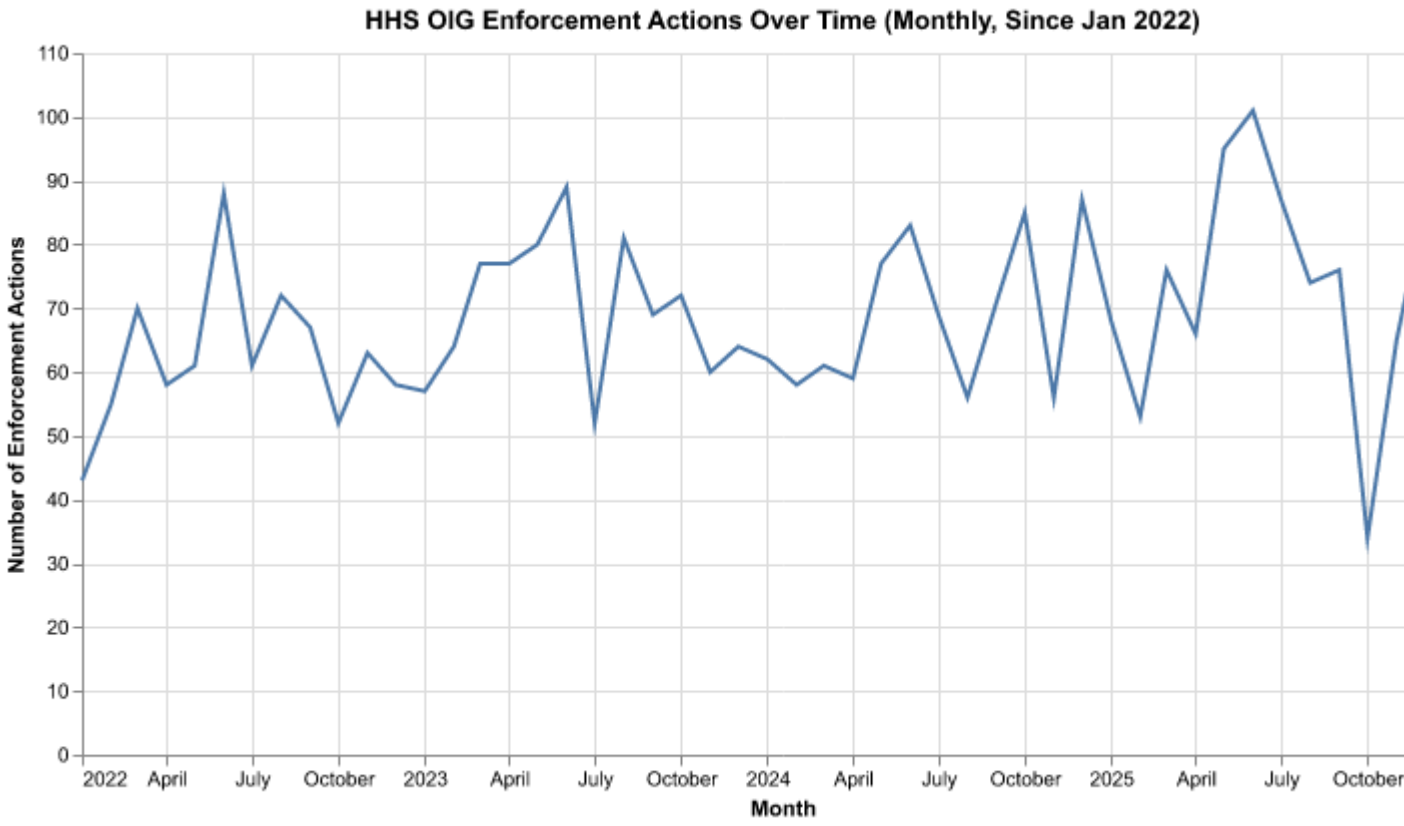
Collecting HHS OIG enforcement actions since January 2022 returned 3,377 enforcement actions in the final dataframe. The earliest enforcement action scraped is dated 2022-01-04, titled "Integrated Pain Management Medical Group Agreed to Pay $10,000 for Allegedly Violating the Civil Monetary Penalties Law by Employing Excluded Individuals." The scraped category label for this entry is Other, and the detail-page link is: https://oig.hhs.gov/fraud/enforcement/integrated-pain-management-medical-group-agreed-to-pay-10000-for-allegedly-violating-the-civil-monetary-penalties-law-by-employing-excluded-individuals/ . I use this January 2022 dataframe (saved as enforcement_actions_2022_01.csv) for all subsequent questions.

## Step 3: Plot data based on scraped data

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

```
df_2022["date"] = pd.to_datetime(df_2022["date"], errors="coerce")

df_2022 = df_2022.dropna(subset=["date"]).copy()

df_2022["year_month"] = df_2022["date"].dt.to_period("M").dt.to_timestamp()

monthly_counts = (
    df_2022.groupby("year_month")
```

```
      .size()
      .reset_index(name="n_actions")
      .sort_values("year_month")
)

chart = (
    alt.Chart(monthly_counts)
      .mark_line()
      .encode(
          alt.X("year_month:T", title="Month"),
          alt.Y("n_actions:Q", title="Number of Enforcement Actions"),
          tooltip=[
              alt.Tooltip("year_month:T", title="Month"),
              alt.Tooltip("n_actions:Q", title="Actions")
          ],
      )
      .properties(
          title="HHS OIG Enforcement Actions Over Time (Monthly, Since Jan
 ↪  2022)",
          width=700,
          height=350
      )
)

chart
```



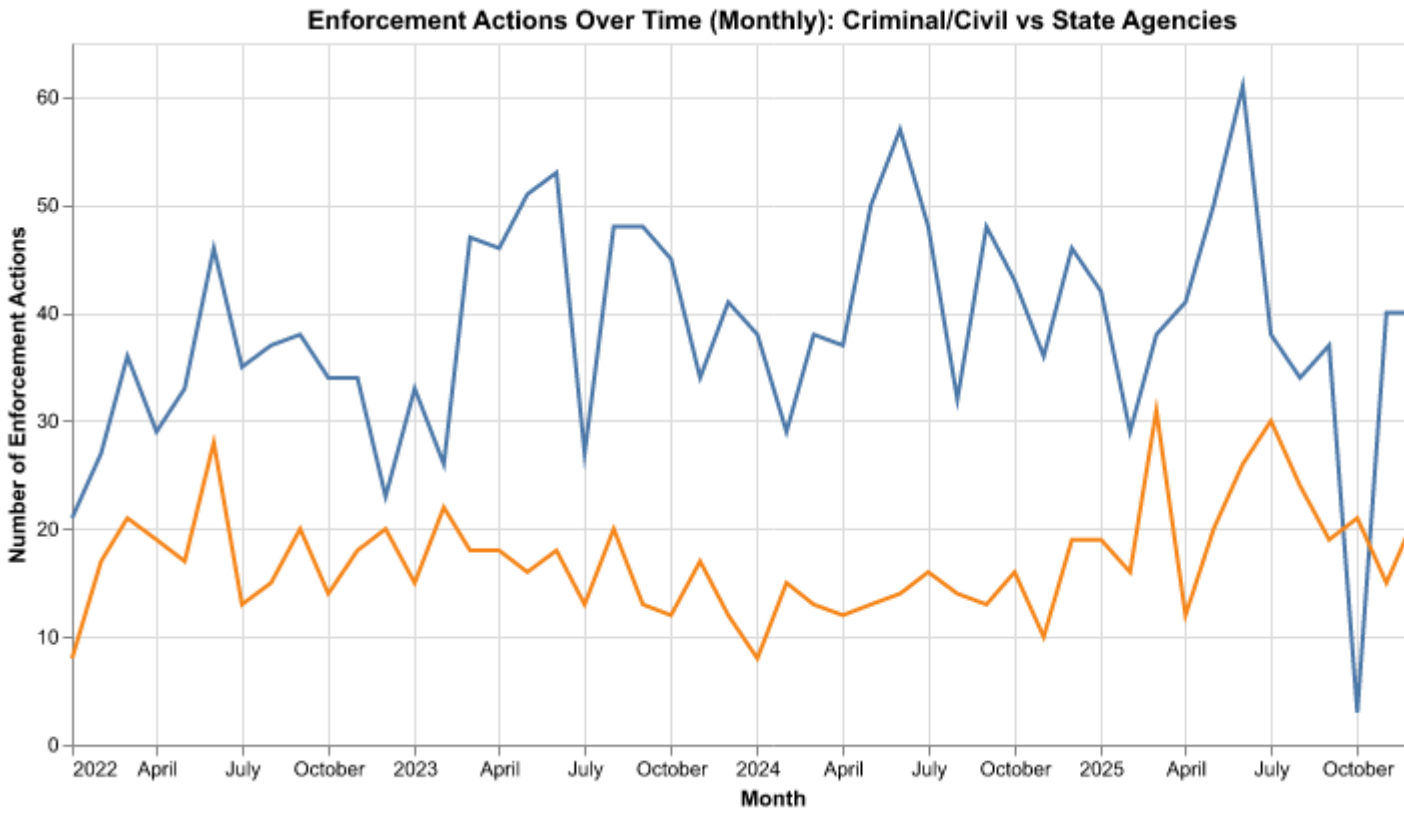

**2. Plot the number of enforcement actions categorized:**

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

```
df_2022["date"] = pd.to_datetime(df_2022["date"], errors="coerce")
df_2022 = df_2022.dropna(subset=["date"]).copy()

keep_categories = ["Criminal and Civil Actions", "State Enforcement
 ↪  Agencies"]

df_two = df_2022[df_2022["category"].isin(keep_categories)].copy()

df_two["year_month"] = df_two["date"].dt.to_period("M").dt.to_timestamp()

monthly_two = (
    df_two.groupby(["year_month", "category"])
          .size()
          .reset_index(name="n_actions")
```

```
          .sort_values(["year_month", "category"])
)

print("Rows in aggregated table:", len(monthly_two))
print(monthly_two.head())

chart_two = (
    alt.Chart(monthly_two)
      .mark_line()
      .encode(
          alt.X("year_month:T", title="Month"),
          alt.Y("n_actions:Q", title="Number of Enforcement Actions"),
          color=alt.Color("category:N", title="Category"),
          tooltip=[
              alt.Tooltip("year_month:T", title="Month"),
              alt.Tooltip("category:N", title="Category"),
              alt.Tooltip("n_actions:Q", title="Actions")
          ],
      )
      .properties(
          title="Enforcement Actions Over Time (Monthly): Criminal/Civil vs
 ↪  State Agencies",
          width=700,
          height=350
      )
)

chart_two
```

```
Rows in aggregated table: 100
  year_month                    category  n_actions
0 2022-01-01  Criminal and Civil Actions         21
1 2022-01-01  State Enforcement Agencies          8
2 2022-02-01  Criminal and Civil Actions         27
3 2022-02-01  State Enforcement Agencies         17
4 2022-03-01  Criminal and Civil Actions         36
```



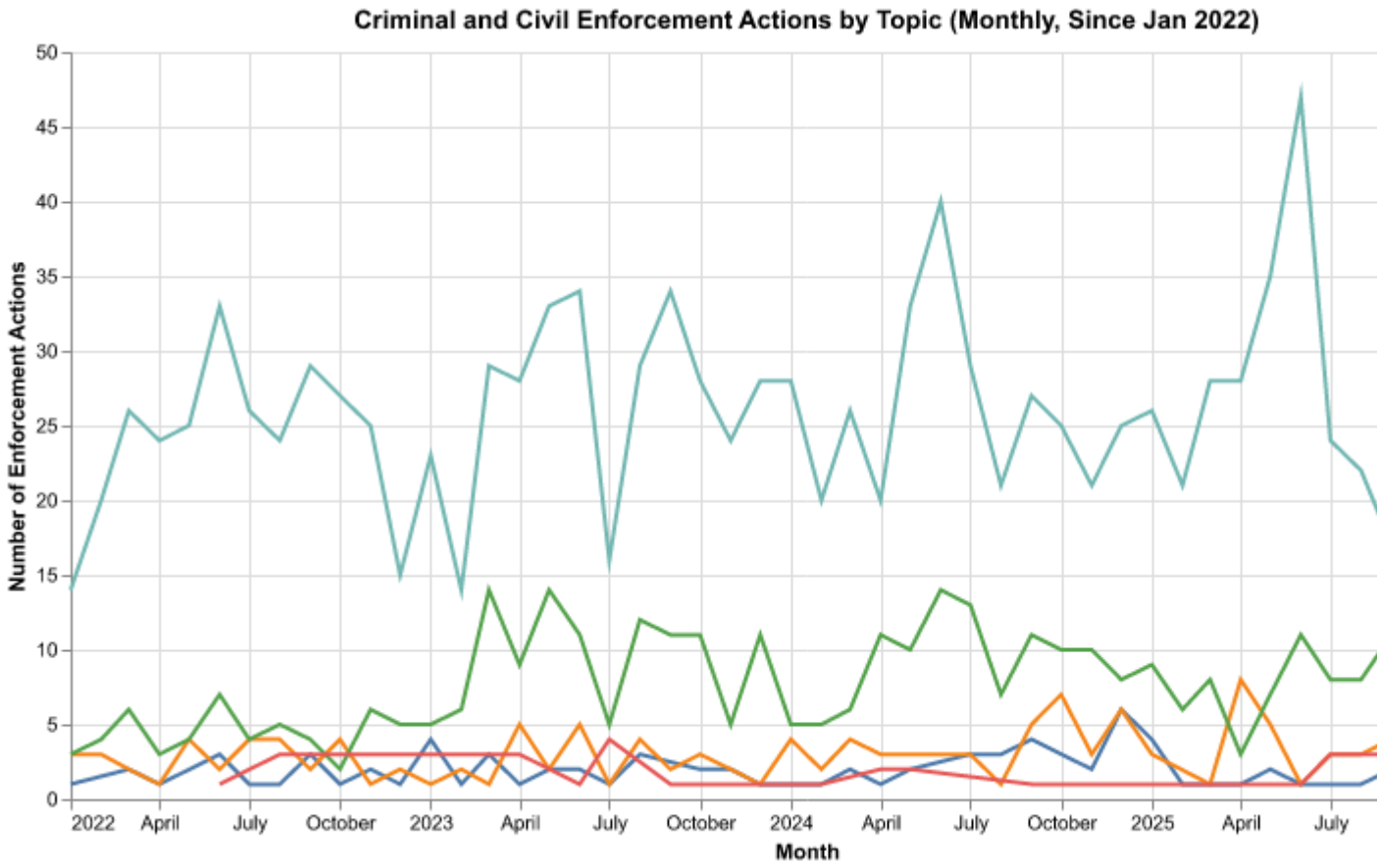

- based on five topics

```
df = pd.read_csv("enforcement_actions_2022_01.csv")
df["date"] = pd.to_datetime(df["date"], errors="coerce")
df = df.dropna(subset=["date"]).copy()

df_cc = df[df["category"] == "Criminal and Civil Actions"].copy()

def assign_topic(title: str) -> str:
    t = title.lower()

    health_kw = [
        "medicare", "medicaid", "health", "healthcare", "hospital",
        "clinic", "physician", "medical", "billing"
    ]
    financial_kw = [
        "bank", "financial", "wire", "loan", "mortgage",
        "credit", "money laundering", "tax", "securities"
    ]
```

```
    drug_kw = [
        "drug", "opioid", "pharmacy", "pharmaceutical",
        "controlled substance", "fentanyl"
    ]
    bribery_kw = [
        "bribe", "bribery", "kickback", "corruption", "conspiracy"
    ]

    if any(k in t for k in health_kw):
        return "Health Care Fraud"
    elif any(k in t for k in financial_kw):
        return "Financial Fraud"
    elif any(k in t for k in drug_kw):
        return "Drug Enforcement"
    elif any(k in t for k in bribery_kw):
        return "Bribery/Corruption"
    else:
        return "Other"

df_cc["topic"] = df_cc["title"].apply(assign_topic)

df_cc["year_month"] = df_cc["date"].dt.to_period("M").dt.to_timestamp()

monthly_topics = (
    df_cc.groupby(["year_month", "topic"])
         .size()
         .reset_index(name="n_actions")
         .sort_values(["year_month", "topic"])
)

print(monthly_topics.head())

chart_topics = (
    alt.Chart(monthly_topics)
      .mark_line()
      .encode(
          x=alt.X("year_month:T", title="Month"),
          y=alt.Y("n_actions:Q", title="Number of Enforcement Actions"),
          color=alt.Color("topic:N", title="Topic"),
          tooltip=[
              alt.Tooltip("year_month:T", title="Month"),
              alt.Tooltip("topic:N", title="Topic"),
              alt.Tooltip("n_actions:Q", title="Actions")
```

```
            ],
        )
        .properties(
            title="Criminal and Civil Enforcement Actions by Topic (Monthly,
 ↪  Since Jan 2022)",
            width=750,
            height=380
        )
)

chart_topics
```

```
  year_month               topic  n_actions
0 2022-01-01  Bribery/Corruption          1
1 2022-01-01    Drug Enforcement          3
2 2022-01-01   Health Care Fraud         14
3 2022-01-01               Other          3
4 2022-02-01    Drug Enforcement          3
```


Criminal and Civil Enforcement Actions by Topic (Monthly, Since Jan 2022)
Number of Enforcement Actions
50
45
40
35
30
25
20
15
10
5
0
2022
April
July
October
2023
April
July
October
2024
April
July
October
2025
April
July
Month