Web Scarping

Han Zhang

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: ZH

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

```
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin
from datetime import date

url = 'https://oig.hhs.gov/fraud/enforcement/'
HHS_html = requests.get(url)
HHS_html.raise_for_status()
HHS_soup = BeautifulSoup(HHS_html.text, 'lxml')

base = "https://oig.hhs.gov"

cards = HHS_soup.select("li.usa-card.card--list")

rows = []

for card in cards:
    # 2) title + link
    a = card.select_one("h2.usa-card__heading a")
    if a is None:
        continue

    title = a.get_text(strip=True)
    link = urljoin(base, a.get("href", ""))

    # 3) date
    date_span = card.select_one("span.text-base-dark")
    date = date_span.get_text(strip=True) if date_span else None
```

```
    # 4) category
    cats = [x.get_text(strip=True) for x in card.select("li.usa-tag")]

    # 5) tidy one row action × one category
    if cats:
        for cat in cats:
            rows.append(
                {"title": title, "date": date, "category": cat, "link": link}
            )
    else:
        rows.append(
            {"title": title, "date": date, "category": None, "link": link}
        )

df = pd.DataFrame(rows)

# convert date to datetime
df["date"] = pd.to_datetime(df["date"], errors="coerce")

print(df.head())
```

```
                                               title       date  \
0  Houston Transplant Doctor Indicted For Making ... 2026-02-05
1  MultiCare Health System to Pay Millions to Set... 2026-02-04
2  Brooklyn Banker Pleads Guilty to Laundering Pr... 2026-02-03
3  Delafield Man Sentenced to 18 Months' Imprison... 2026-02-03
4  Former NFL Player Convicted for $197M Medicare... 2026-02-03

                    category  \
0  Criminal and Civil Actions
1  Criminal and Civil Actions
2                   COVID-19
3  Criminal and Civil Actions
4  Criminal and Civil Actions

                                                link
0  https://oig.hhs.gov/fraud/enforcement/houston-...
1  https://oig.hhs.gov/fraud/enforcement/multicar...
2  https://oig.hhs.gov/fraud/enforcement/brooklyn...
3  https://oig.hhs.gov/fraud/enforcement/delafiel...
4  https://oig.hhs.gov/fraud/enforcement/former-n...
```

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

```
def scrape_enforcement_actions(year, month, run_scraper=True):
    """
    Scrape HHS OIG enforcement actions from a given year and month to today.
    """

    # ---- indicator to prevent re-running during knit ----
    if not run_scraper:
        print("Scraper not run (run_scraper=False).")
        return None

    # ---- input check ----
    if year < 2013:
        print("Please input a year >= 2013. Enforcement actions before 2013
        ↪  are not listed.")
        return None

    # ---- setup ----
    start_date = pd.Timestamp(year=year, month=month, day=1)
    today = pd.Timestamp.today()

    base_url = "https://oig.hhs.gov"
    list_url = "https://oig.hhs.gov/fraud/enforcement/"

    rows = []
    page = 1

    # ---- crawl pages until stopping condition is met ----
    while True:
        url = list_url if page == 1 else f"{list_url}?page={page}"
        response = requests.get(url)
        response.raise_for_status()

        soup = BeautifulSoup(response.text, "lxml")
        cards = soup.select("li.usa-card.card--list")

        if not cards:
            break
```

```
        oldest_date_on_page = None

        for card in cards:
            a = card.select_one("h2.usa-card__heading a")
            if a is None:
                continue

            title = a.get_text(strip=True)
            link = urljoin(base_url, a.get("href", ""))

            date_span = card.select_one("span.text-base-dark")
            action_date = pd.to_datetime(
                date_span.get_text(strip=True),
                errors="coerce"
            )

            if pd.isna(action_date):
                continue

            if oldest_date_on_page is None or action_date <
             ↪  oldest_date_on_page:
                oldest_date_on_page = action_date

            categories = [c.get_text(strip=True) for c in
↪  card.select("li.usa-tag")]
            if not categories:
                categories = [None]

            if start_date <= action_date <= today:
                for cat in categories:
                    rows.append({
                        "title": title,
                        "date": action_date,
                        "category": cat,
                        "link": link
                    })

        # ---- stopping rule based on date ----
        if oldest_date_on_page is not None and oldest_date_on_page <
         ↪  start_date:
            break

        page += 1
```

```
        time.sleep(1)

    df = pd.DataFrame(rows)

    filename = f"enforcement_actions_{year}_{month:02d}.csv"
    df.to_csv(filename, index=False)

    return df
```

- a. Pseudo-Code

1. **Define a function that takes year and month as inputs.**

2. **Check whether the input year is at least 2013. If not, print a message and exit the function.**

3. **Initialize an empty list to store scraped enforcement actions.**

4. **Set the starting page to the first enforcement actions page.**

5.

While enforcement actions on the current page are more recent than the input month and year:

Request the current page.

Parse the HTML and extract each enforcement action card.

For each card:

Extract the title, date, category, and link. # Convert the date to a datetime object. # If the action date is within the desired time range, store it. # Move to the next page. # Pause for one second to avoid overloading the server.

6.

Once actions earlier than the specified month and year are encountered, stop crawling.

7.

Convert the collected records into a tidy dataframe and save it as a CSV file.

8.

A while loop is used instead of a simple for loop because the number of pages that need to be scraped is not known in advance.

The crawler continues until enforcement actions earlier than the specified start date are reached.

- b. Create Dynamic Scraper
- Unique enforcement actions since Jan 2024 1787

- Earliest date 2024-01-03
- Details:
- ‘Laredo Resident Admits To Impersonating Licensed Nurse’
- ‘Criminal and Civil Actions’
- ‘https://oig.hhs.gov/fraud/enforcement/laredo-resident-admits-to-impersonating-licensed-nurse/’
- ‘Former Nurse Aide Indicted In Death Of Clarksville Patient Arrested In Georgia’
- ‘State Enforcement Agencies’
- ‘https://oig.hhs.gov/fraud/enforcement/former-nurse-aide-indicted-in-death-of-clarksville-patient-arrested-in-georgia/’

```
df_2024 = scrape_enforcement_actions(2024, 1, run_scraper=True)
n_actions = df_2024["link"].nunique()
n_actions
```

```
1787
```

```
earliest_date = df_2024["date"].min()
earliest_date
```

```
Timestamp('2024-01-03 00:00:00')
```

```
earliest_details = (
    df_2024.loc[df_2024["date"] == earliest_date, ["title", "category",
↪   "link"]]
    .drop_duplicates()
)
earliest_details["title"].tolist()
```

```
['Laredo Resident Admits To Impersonating Licensed Nurse',
 'Former Nurse Aide Indicted In Death Of Clarksville Patient Arrested In
 Georgia']
```

```
earliest_details["category"].tolist()
```

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

```
earliest_details["link"].tolist()
```

```
['https://oig.hhs.gov/fraud/enforcement/laredo-resident-admits-to-impersonating-licensed-nurs

 'https://oig.hhs.gov/fraud/enforcement/former-nurse-aide-indicted-in-death-of-clarksville-pa
```

- c. Test Your Code

```
df_2022 = scrape_enforcement_actions(2022, 1, run_scraper=True)
n_actions_2022 = df_2022["link"].nunique()
n_actions_2022
```

```
3377
```

```
earliest_date_2022 = df_2022["date"].min()
earliest_date_2022
```

```
Timestamp('2022-01-04 00:00:00')
```

```
earliest_details_2022 = (
    df_2022.loc[df_2022["date"] == earliest_date_2022, ["title", "category",
 ↪  "link"]]
    .drop_duplicates()
)
earliest_details_2022
```

	title	category	link
3542	Central Medical Systems, LLC, Alan Trent Harle...	Criminal and Civil Actions	https://oig.hhs.gov
3543	Ohio home healthcare provider agrees to pay $5...	Criminal and Civil Actions	https://oig.hhs.gov
3544	Integrated Pain Management Medical Group Agree...	Fraud Self-Disclosures	https://oig.hhs.gov

```
df_2022 = scrape_enforcement_actions(2022, 1, run_scraper=False)
df_2022 = pd.read_csv("enforcement_actions_2022_01.csv",
 ↪  parse_dates=["date"])
```

```
Scraper not run (run_scraper=False).
```

Step 3: Plot data based on scraped data

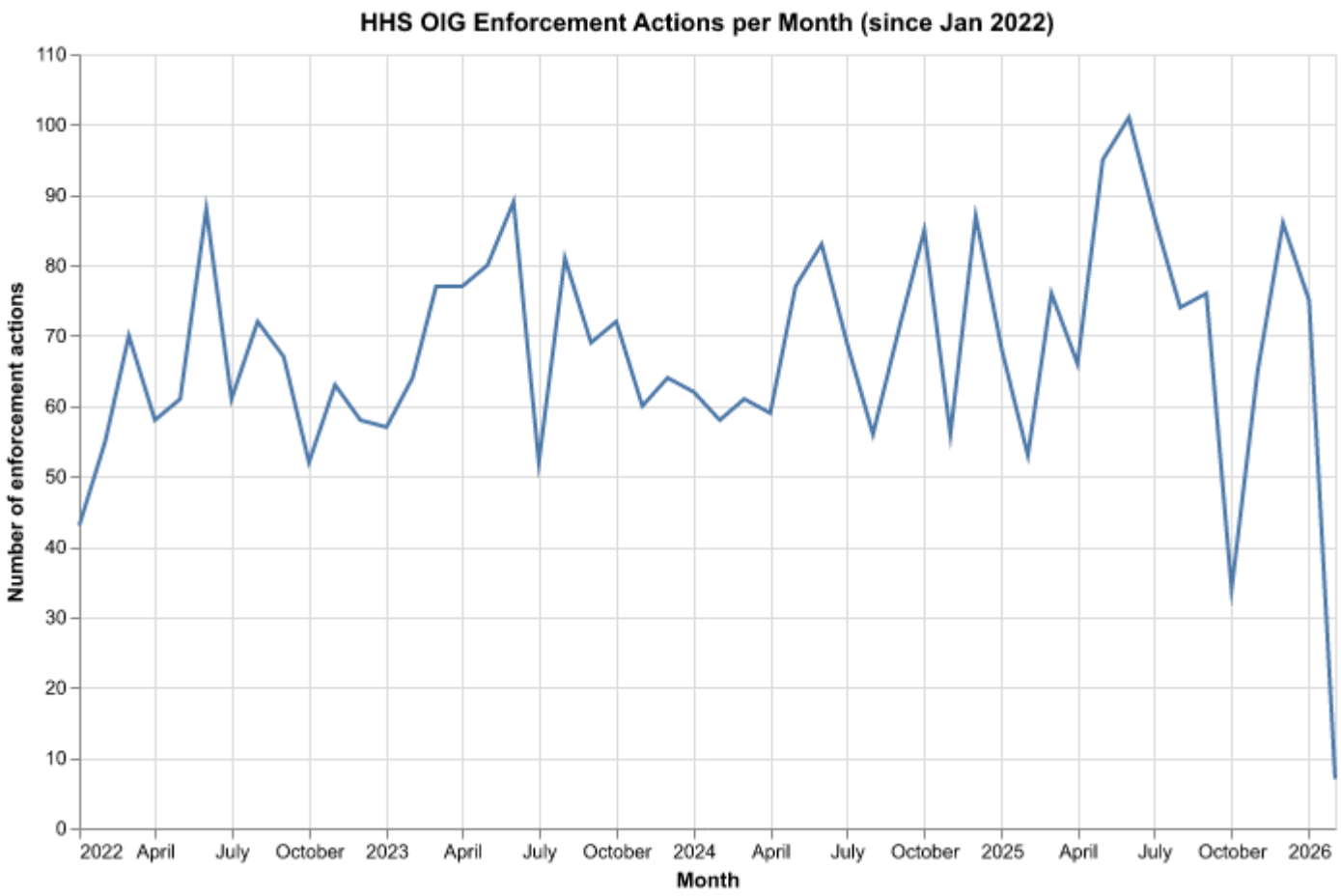
1. Plot the number of enforcement actions over time

```
import altair as alt
monthly_counts = (
    df_2022
    .assign(month=df_2022["date"].dt.to_period("M").dt.to_timestamp())
    .groupby("month", as_index=False)
    .agg(n_actions=("link", "nunique"))
    .sort_values("month")
)
monthly_counts.head()
```

	month	n_actions
0	2022-01-01	43
1	2022-02-01	55
2	2022-03-01	70
3	2022-04-01	58
4	2022-05-01	61

```
# Overall since 2012
line = (
    alt.Chart(monthly_counts)
    .mark_line()
    .encode(
        x=alt.X("month:T", title="Month"),
        y=alt.Y("n_actions:Q", title="Number of enforcement actions"),
        tooltip=[
            alt.Tooltip("month:T", title="Month"),
            alt.Tooltip("n_actions:Q", title="Enforcement actions"),
        ],
    )
    .properties(
        width=650,
        height=400,
        title="HHS OIG Enforcement Actions per Month (since Jan 2022)",
    )
)
```

```
line
```



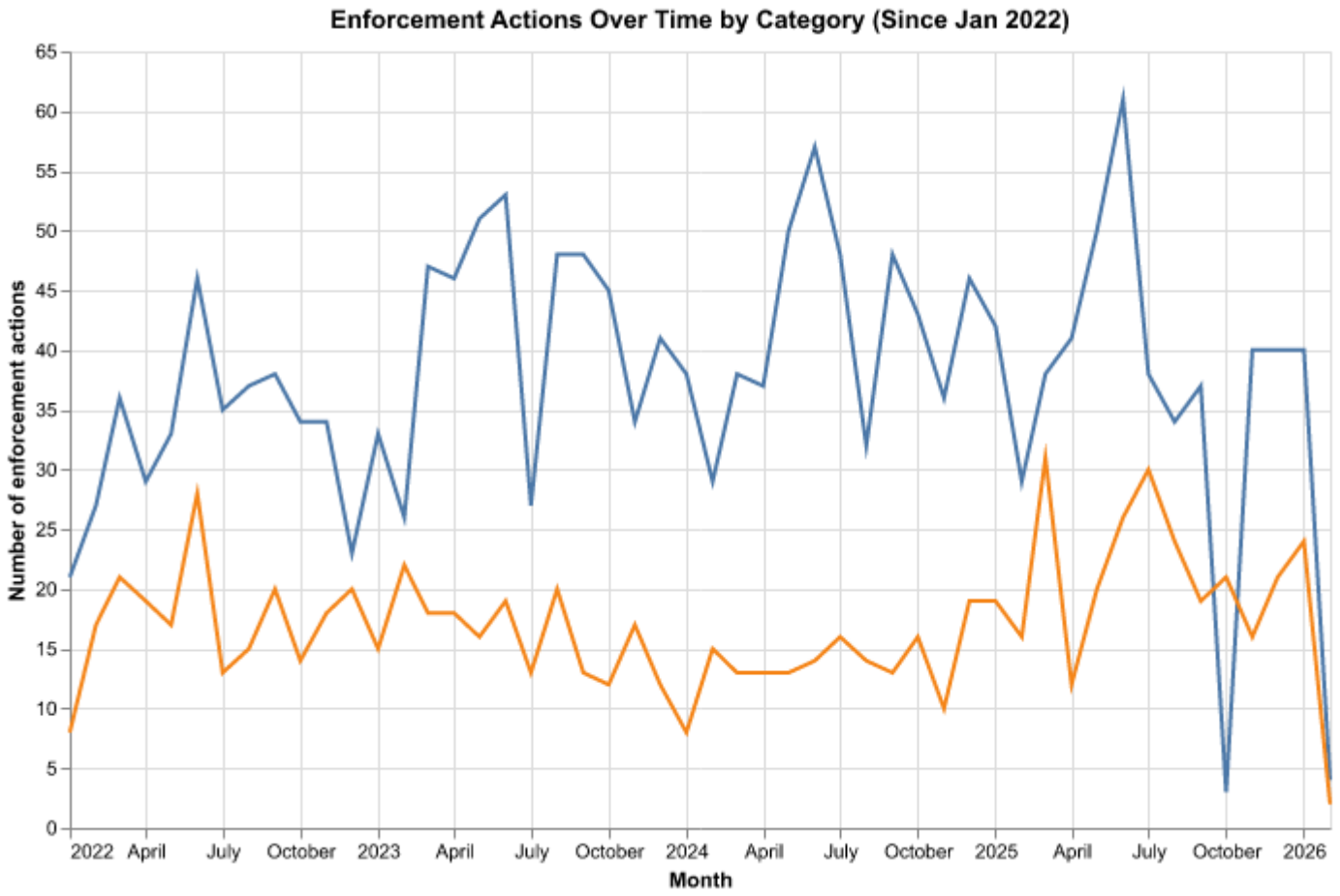
2. Plot the number of enforcement actions categorized:

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

```
df = pd.read_csv("enforcement_actions_2022_01.csv", parse_dates=["date"])
df_main = df[df["category"].isin([
    "Criminal and Civil Actions",
    "State Enforcement Agencies"
])].copy()
```

```
monthly_main = (
    df_main
    .assign(month=df_main["date"].dt.to_period("M").dt.to_timestamp())
    .groupby(["month", "category"], as_index=False)
    .agg(n_actions=("link", "nunique"))
    .sort_values("month")
)
```

```
chart_main = (
    alt.Chart(monthly_main)
    .mark_line()
    .encode(
        x=alt.X("month:T", title="Month"),
        y=alt.Y("n_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("n_actions:Q", title="Actions"),
        ],
    )
    .properties(
        width=650,
        height=400,
        title="Enforcement Actions Over Time by Category (Since Jan 2022)"
    )
)

chart_main
```

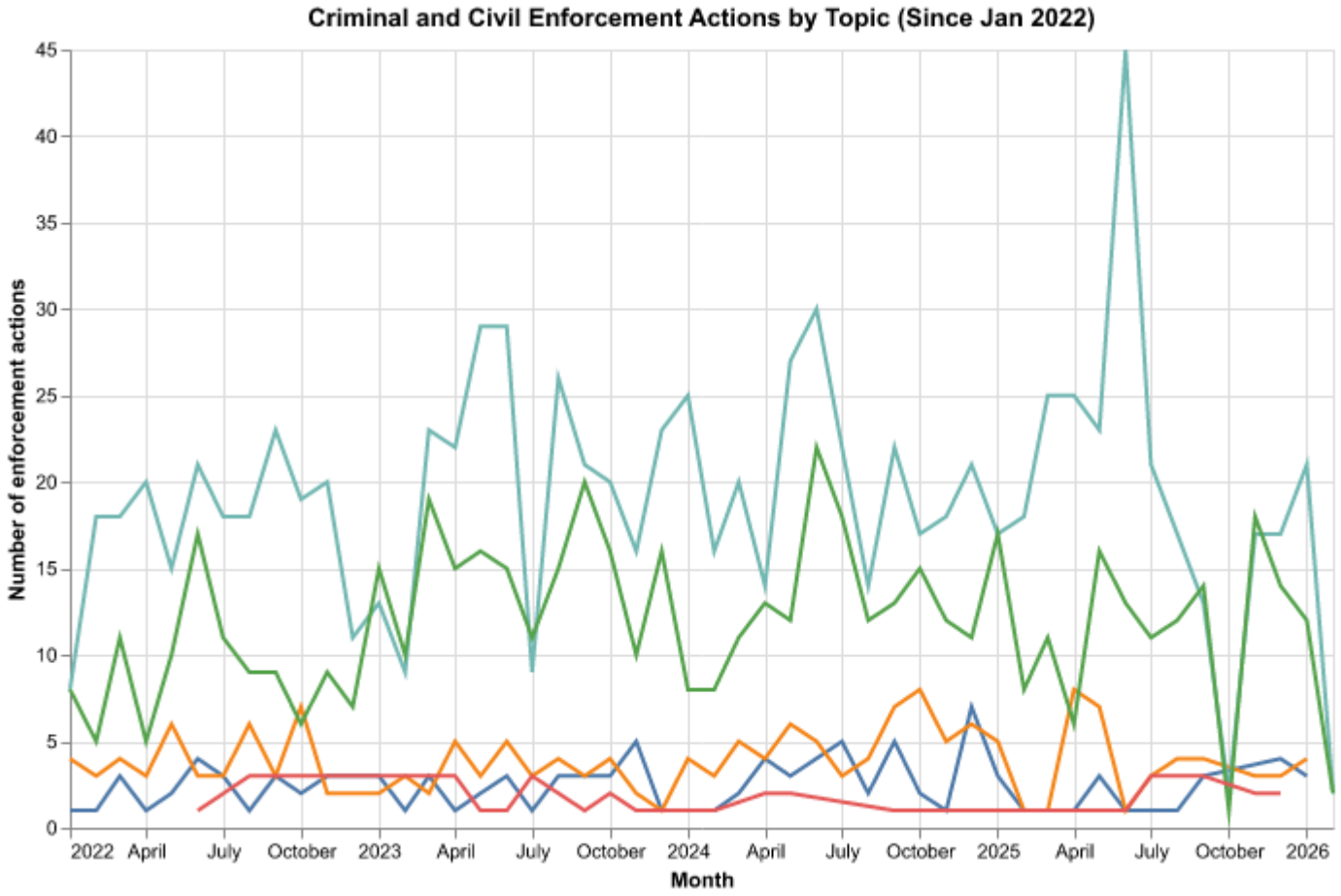


- based on five topics

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

def classify_topic(title):
    t = title.lower()

    if any(k in t for k in ["medicare", "medicaid", "health care",
     ↪  "healthcare", "hospital", "clinic"]):
        return "Health Care Fraud"

    elif any(k in t for k in ["bank", "financial", "wire fraud", "money
     ↪  laundering", "tax"]):
        return "Financial Fraud"

    elif any(k in t for k in ["drug", "opioid", "fentanyl", "pharmacy",
     ↪  "controlled substance"]):
```

```
        return "Drug Enforcement"

    elif any(k in t for k in ["bribe", "bribery", "kickback", "corruption"]):
        return "Bribery/Corruption"

    else:
        return "Other"

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

monthly_topics = (
    df_cc
    .assign(month=df_cc["date"].dt.to_period("M").dt.to_timestamp())
    .groupby(["month", "topic"], as_index=False)
    .agg(n_actions=("link", "nunique"))
    .sort_values("month")
)
```

```
chart_topics = (
    alt.Chart(monthly_topics)
    .mark_line()
    .encode(
        x=alt.X("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("month:T", title="Month"),
            alt.Tooltip("topic:N", title="Topic"),
            alt.Tooltip("n_actions:Q", title="Actions"),
        ],
    )
    .properties(
        width=650,
        height=400,
        title="Criminal and Civil Enforcement Actions by Topic (Since Jan
 ↪  2022)"
    )
)

chart_topics
```


Criminal and Civil Enforcement Actions by Topic (Since Jan 2022)
Number of enforcement actions
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
October
2026
Month