30538 Problem Set 4: Web Scraping

Ziye Yang

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: Ziye Yang

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 re
from urllib.parse import urljoin

import pandas as pd
import requests
from bs4 import BeautifulSoup, Tag

URL = "https://oig.hhs.gov/fraud/enforcement/"

resp = requests.get(
    URL,
    headers={
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
        ↪  AppleWebKit/537.36 (KHTML, like Gecko) Chrome Safari"
    },
    timeout=30,
)
resp.raise_for_status()
soup = BeautifulSoup(resp.text, "html.parser")

# ---- helpers ----
MONTHS =
 ↪  "January|February|March|April|May|June|July|August|September|October|November|December"
DATE_RE = re.compile(rf"^({MONTHS})\s+\d{{1,2}},\s+\d{{4}}$")

def find_date_text(container: Tag) -> str | None:

    for t in container.stripped_strings:
        if DATE_RE.match(t):
```

```
            return t
    return None

def find_categories(container: Tag) -> list[str]:
    cats = []
    for li in container.select("ul li"):
        txt = li.get_text(" ", strip=True)
        if not txt:
            continue
        if len(txt) > 60:
            continue
        if DATE_RE.match(txt):
            continue

        cats.append(txt)

    seen = set()
    out = []
    for c in cats:
        if c not in seen:
            out.append(c)
            seen.add(c)
    return out

rows = []

view_rows = soup.select(".views-row")

if view_rows:
    for row in view_rows:
        a = row.select_one("h2 a")
        if not a:
            continue

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

        date = find_date_text(row)

        cats = find_categories(row)
        category = " | ".join(cats)

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

```
else:
    for h2 in soup.find_all("h2"):
        a = h2.find("a")
        if not a or not a.get("href"):
            continue

        title = a.get_text(strip=True)
        link = urljoin(URL, a["href"])

        date = None
        cats = []

        sib = h2.next_sibling
        while sib is not None:
            if isinstance(sib, Tag) and sib.name == "h2":
                break

            if isinstance(sib, Tag):
                if date is None:
                    maybe = find_date_text(sib)
                    if maybe:
                        date = maybe

                cats.extend(find_categories(sib))

            sib = sib.next_sibling

        seen = set()
        cats_clean = []
        for c in cats:
            if c not in seen:
                cats_clean.append(c)
                seen.add(c)

        rows.append({"title": title, "date": date, "category": " |
 ↪  ".join(cats_clean), "link": link})

df = pd.DataFrame(rows)

# basic sanity clean
df = df[df["title"].notna() & (df["title"].str.len() >
 ↪  0)].reset_index(drop=True)
```

```
print(df.head())
```

```
                                               title              date  \
0  Houston Transplant Doctor Indicted For Making ...  February 5, 2026
1  MultiCare Health System to Pay Millions to Set...  February 4, 2026
2  Brooklyn Banker Pleads Guilty to Laundering Pr...  February 3, 2026
3  Delafield Man Sentenced to 18 Months' Imprison...  February 3, 2026
4  Former NFL Player Convicted for $197M Medicare...  February 3, 2026

                     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

- a. Pseudo-Code

Input month, year, run_scraper -> if year < 2013 print reminder and set year = 2013 -> define start_date = (year-month-01) and out_csv = enforcement_actions_{year}_{month}.csv -> if run_scraper == False then read out_csv and return (so knitting won't re-scrape) -> set page = 1, all_rows = [] -> while True: build URL (base if page=1 else base?page=page) -> request HTML -> parse this page into rows (title/date/category/link) -> keep only rows with date >= start_date and append to all_rows -> compute earliest_date_on_page = min(date on this page) -> if earliest_date_on_page < start_date break (next pages are older) else page += 1 and sleep(1) -> after loop: combine all rows into one dataframe -> drop duplicates by link -> sort by date -> save to out_csv -> return dataframe (and later print total count + earliest action details).

- b. Create Dynamic Scraper

```
import time

BASE_URL = "https://oig.hhs.gov/fraud/enforcement/"

# --- helpers for parsing ---
MONTHS =
 ↪  "January|February|March|April|May|June|July|August|September|October|November|December"
DATE_RE = re.compile(rf"^({MONTHS})\s+\d{{1,2}},\s+\d{{4}}$")

def _find_date_text(container: Tag) -> str | None:
    for t in container.stripped_strings:
        if DATE_RE.match(t):
            return t
    return None

def _find_categories(container: Tag) -> list[str]:
    """
    Categories are short tags (often in a UL/LI list) within each record
 ↪  container.
    We use a heuristic to avoid accidentally capturing long titles.
    """
    cats = []
    for li in container.select("ul li"):
        txt = li.get_text(" ", strip=True)
        if not txt:
            continue
        if DATE_RE.match(txt):
            continue
        if len(txt) > 60:  # avoid long sentences / titles
            continue
        cats.append(txt)

    # dedupe preserving order
    seen = set()
    out = []
    for c in cats:
        if c not in seen:
            out.append(c)
            seen.add(c)
    return out

def _parse_one_page(html: str) -> pd.DataFrame:
    """
```

```
    Parse ONE enforcement listing page into tidy df with:
    title, date (as Timestamp), category, link
    """
    soup = BeautifulSoup(html, "html.parser")
    rows = []

    view_rows = soup.select(".views-row")

    if view_rows:
        # preferred: each record in its own container
        for row in view_rows:
            a = row.select_one("h2 a")
            if not a or not a.get("href"):
                continue

            title = a.get_text(strip=True)
            link = urljoin(BASE_URL, a["href"])

            date_str = _find_date_text(row)
            date = pd.to_datetime(date_str) if date_str else pd.NaT

            cats = _find_categories(row)
            category = " | ".join(cats) if cats else ""

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

    else:
        # fallback: treat each h2 as start of a record, only scan siblings
         ↪  until next h2
        for h2 in soup.find_all("h2"):
            a = h2.find("a")
            if not a or not a.get("href"):
                continue

            title = a.get_text(strip=True)
            link = urljoin(BASE_URL, a["href"])

            date = pd.NaT
            cats = []

            sib = h2.next_sibling
            while sib is not None:
```

```
                if isinstance(sib, Tag) and sib.name == "h2":
                    break
                if isinstance(sib, Tag):
                    if pd.isna(date):
                        date_str = _find_date_text(sib)
                        if date_str:
                            date = pd.to_datetime(date_str)
                    cats.extend(_find_categories(sib))
                sib = sib.next_sibling

            # dedupe cats
            seen = set()
            cats_clean = []
            for c in cats:
                if c not in seen:
                    cats_clean.append(c)
                    seen.add(c)

            rows.append({"title": title, "date": date, "category": " |
↪   ".join(cats_clean), "link": link})

    df = pd.DataFrame(rows)
    # drop weird empty rows if any
    df = df[df["title"].notna() & (df["title"].str.len() > 0)].copy()
    return df.reset_index(drop=True)

def scrape_enforcement_actions(month: int, year: int, run_scraper: bool =
↪   False) -> pd.DataFrame:
    """
    Crawl HHS OIG Enforcement Actions from (year, month) to today.
    Saves to enforcement_actions_{year}_{month}.csv and returns dataframe.

    run_scraper=False:
      - do NOT crawl (for knitting). Instead, load existing CSV if present.
    """
    if year < 2013:
        print("Note: Only enforcement actions after 2013 are listed on this
        ↪   site. Restricting to year >= 2013.")
        year = 2013

    start_date = pd.Timestamp(year=year, month=month, day=1)
    out_csv = f"enforcement_actions_{year}_{month:02d}.csv"
```

```
    if not run_scraper:
        # knitting-safe path: load previous output
        return pd.read_csv(out_csv, parse_dates=["date"])

    session = requests.Session()
    session.headers.update({
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
        ↪  AppleWebKit/537.36 (KHTML, like Gecko) Chrome Safari"
    })

    all_pages = []
    page = 1

    while True:
        url = BASE_URL if page == 1 else f"{BASE_URL}?page={page}"
        resp = session.get(url, timeout=30)
        resp.raise_for_status()

        df_page = _parse_one_page(resp.text)

        # if page is empty, stop
        if df_page.empty:
            break

        # keep only from start_date onward
        df_keep = df_page[df_page["date"].notna() & (df_page["date"] >=
↪   start_date)].copy()
        all_pages.append(df_keep)

        # stopping rule: once the earliest date on this page is older than
        ↪  start_date,
        # the next pages will be even older, so we can stop.
        earliest_on_page = df_page["date"].min()
        if pd.notna(earliest_on_page) and earliest_on_page < start_date:
            break

        page += 1
        time.sleep(1)

    df_all = pd.concat(all_pages, ignore_index=True) if all_pages else
↪   pd.DataFrame(columns=["title","date","category","link"])
```

```
    # final cleanup: de-dupe, sort (optional but nice)
    df_all = (
        df_all.drop_duplicates(subset=["link"])
              .sort_values("date")
              .reset_index(drop=True)
    )

    df_all.to_csv(out_csv, index=False)
    return df_all


# --------- RUN ONCE for assignment part (b): since Jan 2024 ---------
df_2024 = scrape_enforcement_actions(month=1, year=2024, run_scraper=True)

print(f"Total enforcement actions since Jan 2024: {len(df_2024):,}")

if len(df_2024) > 0:
    earliest = df_2024.iloc[0]
    print("Earliest enforcement action in this scrape:")
    print(f"  Date : {earliest['date'].date()}")
    print(f"  Title: {earliest['title']}")
    print(f"  Link : {earliest['link']}")

df_2024.head()
```

```
Total enforcement actions since Jan 2024: 1,787
Earliest enforcement action in this scrape:
  Date : 2024-01-03
  Title: Former Nurse Aide Indicted In Death Of Clarksville Patient Arrested
  In Georgia
  Link :
  https://oig.hhs.gov/fraud/enforcement/former-nurse-aide-indicted-in-death-of-clarksville-pa
```

	title	date	category	lin
0	Former Nurse Aide Indicted In Death Of Clarksv...	2024-01-03	State Enforcement Agencies	htt
1	Laredo Resident Admits To Impersonating Licens...	2024-01-03	Criminal and Civil Actions	htt
2	Recover-Care Plaza West Care Center Agreed to ...	2024-01-04	Fraud Self-Disclosures	htt
3	Patricia Perez Agreed to Be Excluded for 7 Yea...	2024-01-04	CMP and Affirmative Exclusions	htt
4	Memphis-Based Methodist Le Bonheur Healthcare ...	2024-01-04	Criminal and Civil Actions	htt

- c. Test Your Code

```python
df_2022 = scrape_enforcement_actions(month=1, year=2022, run_scraper=True)

print(f"Total enforcement actions since Jan 2022: {len(df_2022):,}")

if len(df_2022) > 0:
    earliest = df_2022.iloc[0]
    print("Earliest enforcement action in this scrape:")
    print(f"  Date : {earliest['date'].date()}")
    print(f"  Title: {earliest['title']}")
    print(f"  Link : {earliest['link']}")
    print(f"  Category: {earliest['category']}")

df_2022.head()
```

```
Total enforcement actions since Jan 2022: 3,377
Earliest enforcement action in this scrape:
  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
  Link :
  https://oig.hhs.gov/fraud/enforcement/integrated-pain-management-medical-group-agreed-to-pa
  Category: Fraud Self-Disclosures
```

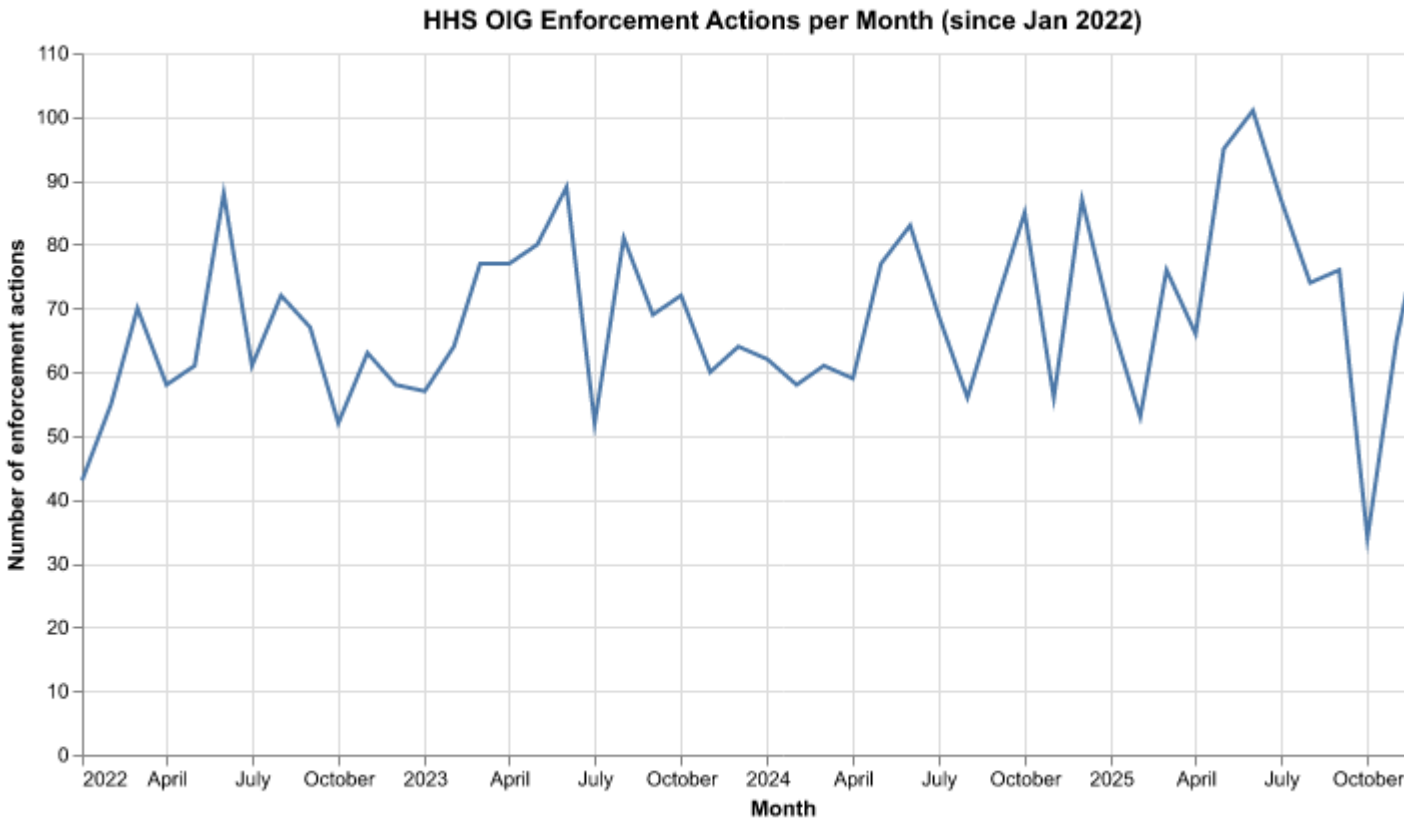
	title	date	category	lin
0	Integrated Pain Management Medical Group Agree...	2022-01-04	Fraud Self-Disclosures	htt
1	Ohio home healthcare provider agrees to pay $5...	2022-01-04	Criminal and Civil Actions	htt
2	Central Medical Systems, LLC, Alan Trent Harle...	2022-01-04	Criminal and Civil Actions	htt
3	Emergency Ambulance Service Agreed to Pay $430...	2022-01-05	CMP and Affirmative Exclusions	htt
4	Attorney General Alan Wilson announces arrest ...	2022-01-05	State Enforcement Agencies	htt

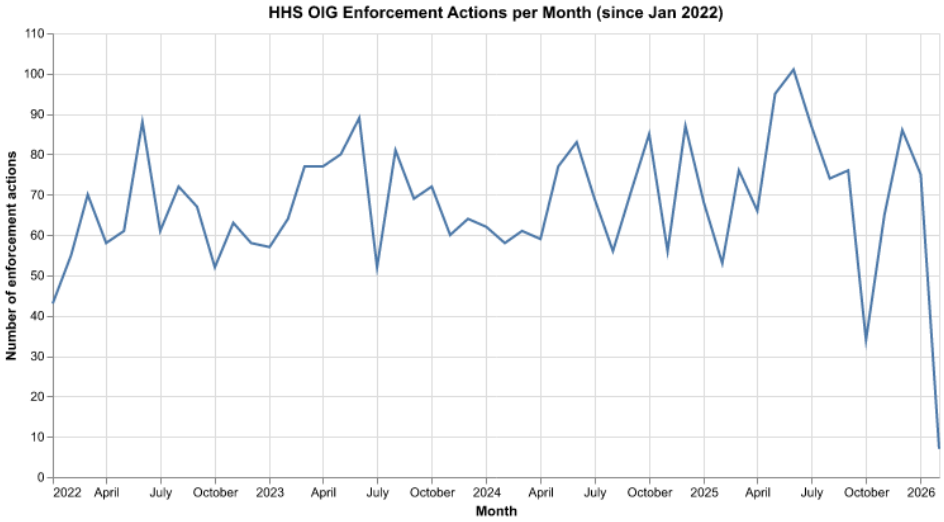
Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

```python
import altair as alt

df = pd.read_csv("enforcement_actions_2022_01.csv", parse_dates=["date"])
df["month"] = df["date"].dt.to_period("M").dt.to_timestamp()
```

```
monthly_counts = (
    df.groupby("month")
      .size()
      .reset_index(name="num_actions")
      .sort_values("month")
)

chart = (
    alt.Chart(monthly_counts)
      .mark_line()
      .encode(
          x=alt.X("month:T", title="Month"),
          y=alt.Y("num_actions:Q", title="Number of enforcement actions"),
          tooltip=[
              alt.Tooltip("month:T", title="Month"),
              alt.Tooltip("num_actions:Q", title="Actions")
          ]
      )
      .properties(
          title="HHS OIG Enforcement Actions per Month (since Jan 2022)",
          width=700,
          height=350
      )
)

chart
```

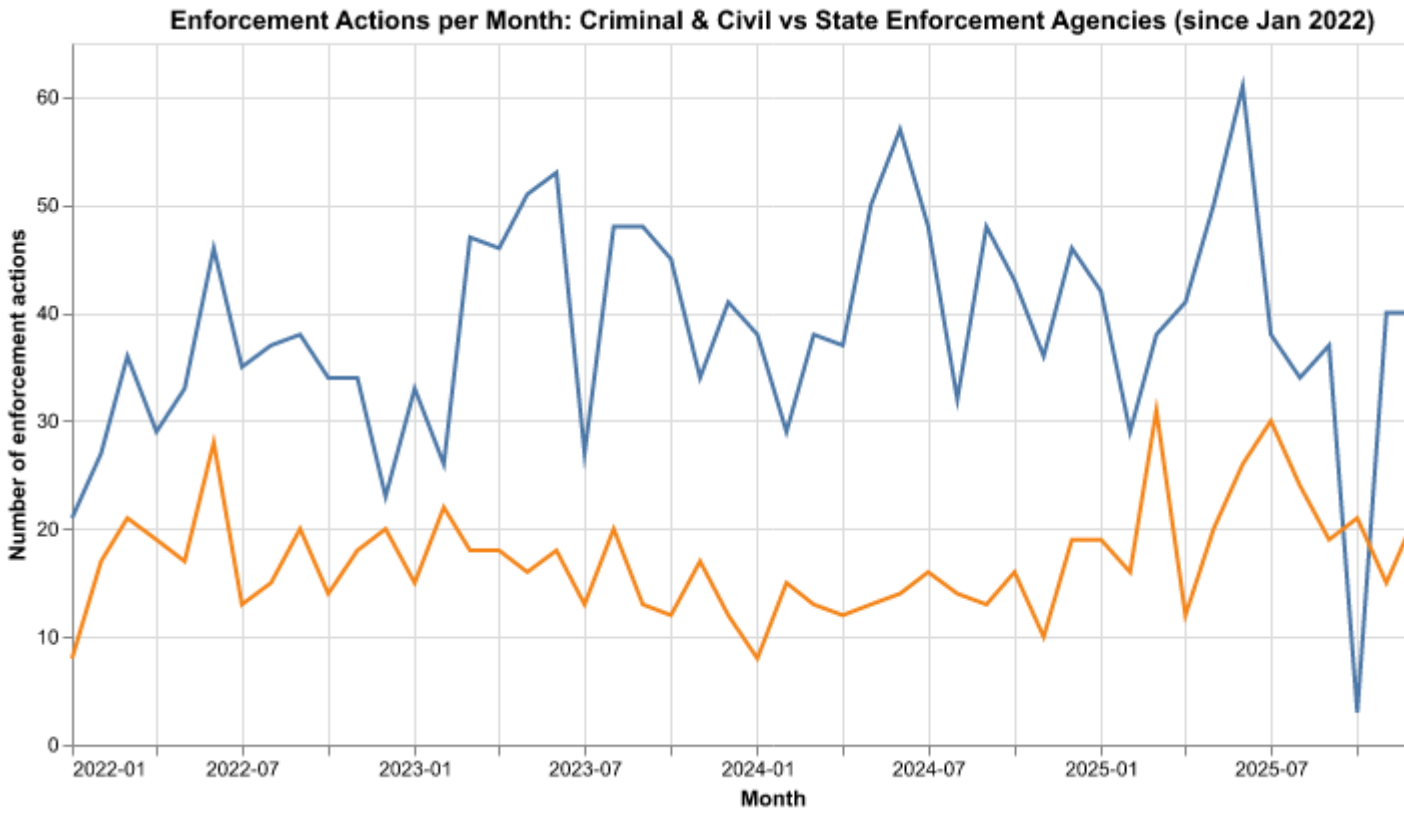


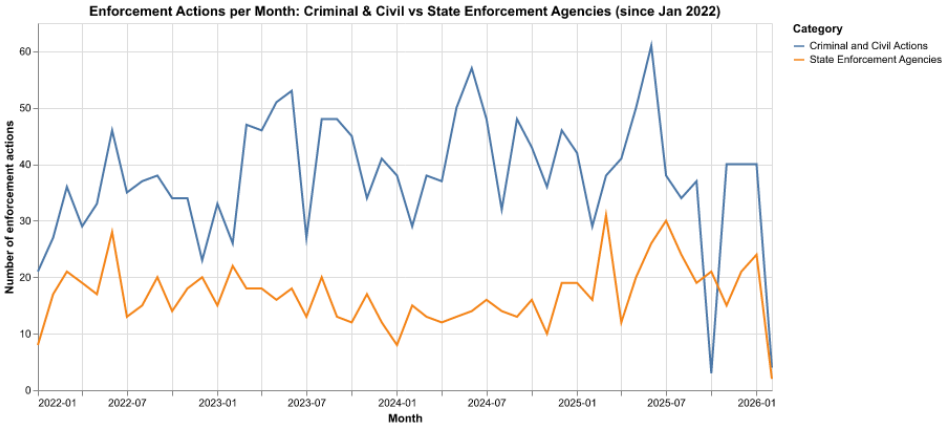


2. Plot the number of enforcement actions categorized:

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

```
import numpy as np

is_criminal = df["category"].fillna("").str.contains("Criminal and Civil
 ↪  Actions", regex=False)
is_state = df["category"].fillna("").str.contains("State Enforcement
 ↪  Agencies", regex=False)

df_2a = df[is_criminal | is_state].copy()
df_2a["group"] = np.where(is_criminal.loc[df_2a.index], "Criminal and Civil
 ↪  Actions", "State Enforcement Agencies")

monthly_2a = (
    df_2a.groupby(["month", "group"])
         .size()
         .reset_index(name="num_actions")
         .sort_values("month")
)

chart_2a = (
    alt.Chart(monthly_2a)
      .mark_line()
      .encode(
          x=alt.X("month:T", title="Month", axis=alt.Axis(format="%Y-%m")),
          y=alt.Y("num_actions:Q", title="Number of enforcement actions"),
          color=alt.Color("group:N", title="Category"),
          tooltip=[
              alt.Tooltip("month:T", title="Month"),
              alt.Tooltip("group:N", title="Category"),
              alt.Tooltip("num_actions:Q", title="Actions")
          ]
      )
      .properties(
          title="Enforcement Actions per Month: Criminal & Civil vs State
 ↪  Enforcement Agencies (since Jan 2022)",
          width=700,
          height=350
      )
)

chart_2a
```



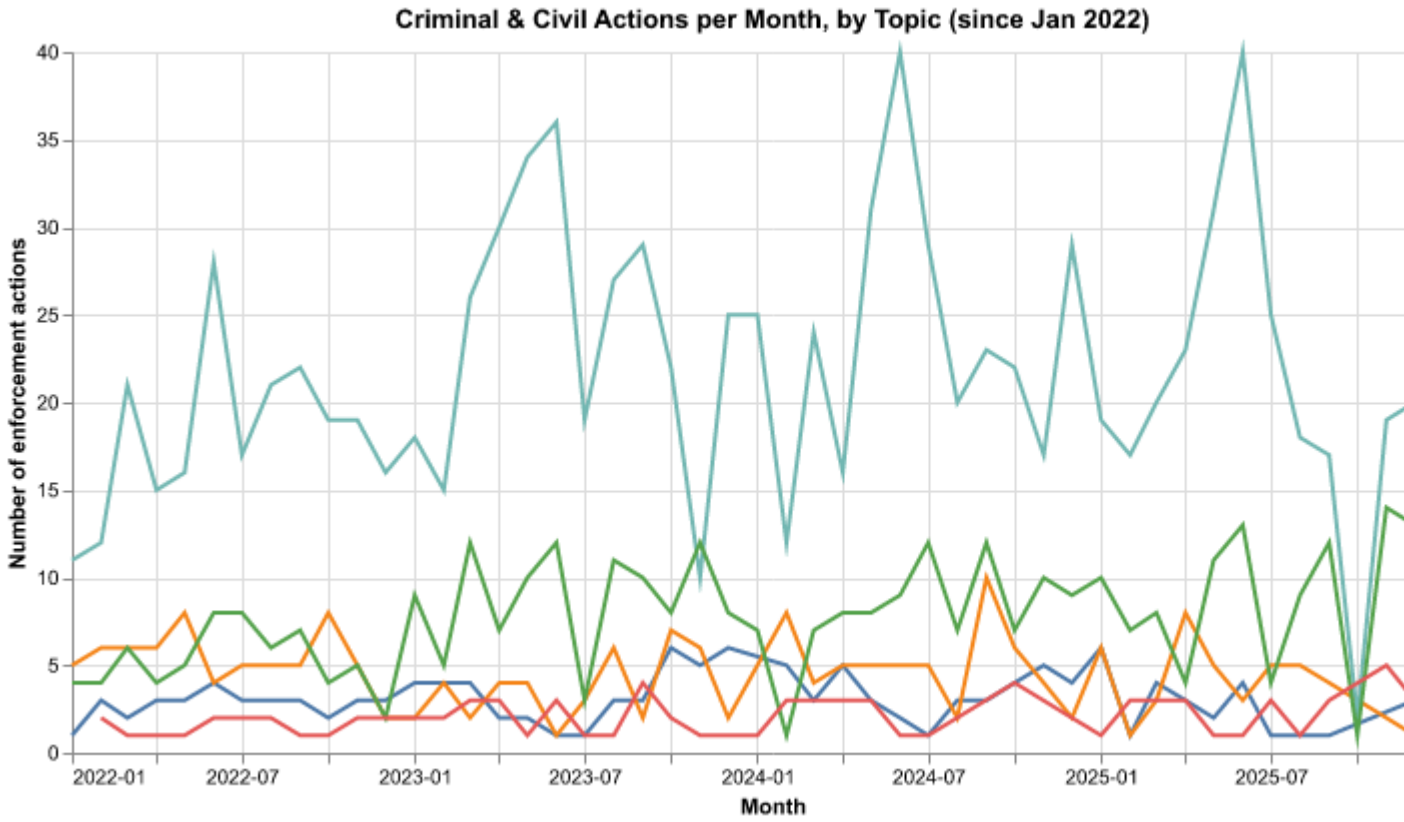


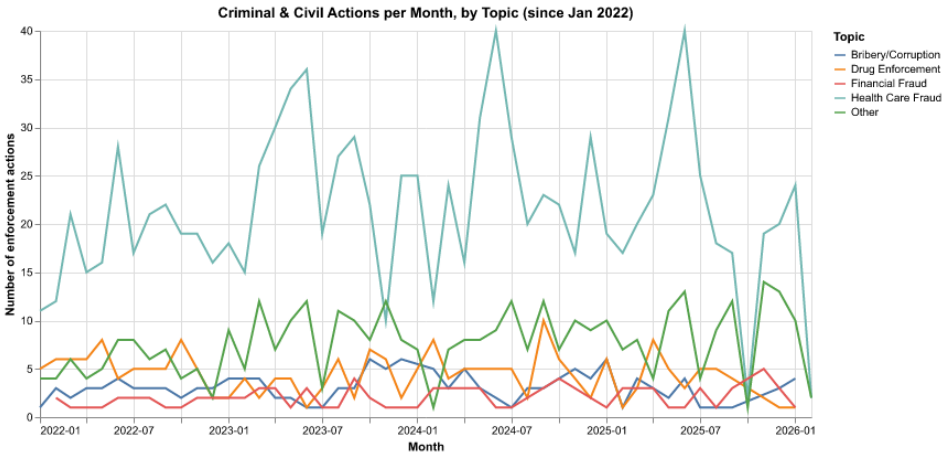
- based on five topics

```
df_cc = df[df["category"].fillna("").str.contains("Criminal and Civil
 ↪  Actions", regex=False)].copy()

def classify_topic(title: str) -> str:
```

```
    t = (title or "").lower()

    if re.search(r"\b(opioid|fentanyl|controlled
     ↪  substance|drug|dea|pharmacy|prescription|pill|traffick)\b", t):
        return "Drug Enforcement"

    if
     ↪  re.search(r"\b(kickback|brib(e|ery)|corrupt|embezzl|extortion|anti-kickback|aks)\b",
     ↪  t):
        return "Bribery/Corruption"

    if re.search(r"\b(bank|banker|wire fraud|money
     ↪  laundering|launder|tax|securities|forex|bitcoin|crypto|financial)\b",
     ↪  t):
        return "Financial Fraud"

    if re.search(r"\b(medicare|medicaid|health care
     ↪  fraud|billing|claims|provider|clinic|hospital|physician|doctor|nurse|home
     ↪  health)\b", t):
        return "Health Care Fraud"

    return "Other"

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

monthly_2b = (
    df_cc.groupby(["month", "topic"])
         .size()
         .reset_index(name="num_actions")
         .sort_values("month")
)

chart_2b = (
    alt.Chart(monthly_2b)
      .mark_line()
      .encode(
          x=alt.X("month:T", title="Month", axis=alt.Axis(format="%Y-%m")),
          y=alt.Y("num_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("num_actions:Q", title="Actions")
        ]
    )
    .properties(
        title="Criminal & Civil Actions per Month, by Topic (since Jan
↪  2022)",
        width=700,
        height=350
    )
)

chart_2b
```




Criminal & Civil Actions per Month, by Topic (since Jan 2022)
Number of enforcement actions
Month
Topic
Bribery/Corruption
Drug Enforcement
Financial Fraud
Health Care Fraud
Other
0
5
10
15
20
25
30
35
40
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
2025-01
2025-07
2026-01