Problem Set 4

Abraham Sadat

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:**A.S**

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.

```python
import pandas as pd
import altair as alt
import time
import requests as requests
import pandas as pd
from bs4 import BeautifulSoup
from urllib.parse import urljoin
from datetime import date

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

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

Step 1: Develop initial scraper and crawler

```python
import requests as requests

from bs4 import BeautifulSoup

with open("/Users/abrahamsadat/My Drive/ps3-abrahamsadat96/ps4-abrahamsadat96/Enforcement
↪   Actions _ Office of Inspector General _ Government Oversight _ U.S. Department of
↪   Health and Human Services.html",
        "r", encoding="utf-8") as f:
    soup = BeautifulSoup(f, "lxml")
```

```python
import pandas as pd
from urllib.parse import urljoin

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

rows = []

with open(
    "/Users/abrahamsadat/My Drive/ps3-abrahamsadat96/ps4-abrahamsadat96/Enforcement
    ↪   Actions _ Office of Inspector General _ Government Oversight _ U.S. Department of
    ↪   Health and Human Services.html",
    "r",
    encoding="utf-8"
) as f:
    soup = BeautifulSoup(f, "lxml")

CARD_SELECTOR = (
"li.usa-card.card--list.pep-card--minimal."
"mobile\\:grid-col-12"
)

cards = soup.select(CARD_SELECTOR)
```

```
for card in cards:
    # Title + link
    a = card.select_one("h2 a")
    title = a.get_text(strip=True) if a else None
    link = urljoin(BASE_URL, a["href"]) if a and a.has_attr("href") else None

    # Date
    date_span = card.select_one("span.text-base-dark.padding-right-105")
    date = date_span.get_text(strip=True) if date_span else None

    # Category (can be multiple tags)
    TAG_SELECTOR = (
    "li.display-inline-block.usa-tag.text-no-lowercase."
    "text-base-darkest.bg-base-lightest.margin-right-1"
)
    cat_tags = card.select(TAG_SELECTOR)

    category = "; ".join(t.get_text(strip=True) for t in cat_tags) if cat_tags else None

    rows.append({
        "Title of the enforcement action": title,
        "Date": date,
        "Category": category,
        "Link associated with the enforcement action": link
    })

df = pd.DataFrame(rows)

df[["Title of the enforcement action", "Date", "Category"]].head()
```

	Title of the enforcement action	Date	Category
0	Houston Transplant Doctor Indicted For Making ...	February 5, 2026	Criminal and Civil Actions
1	MultiCare Health System to Pay Millions to Set...	February 4, 2026	Criminal and Civil Actions
2	Brooklyn Banker Pleads Guilty to Laundering Pr...	February 3, 2026	COVID-19
3	Delafield Man Sentenced to 18 Months' Imprison...	February 3, 2026	Criminal and Civil Actions
4	Former NFL Player Convicted for $197M Medicare...	February 3, 2026	Criminal and Civil Actions

Link to AI prompt used to develop the BeautifulSoup code for this question: https://chatgpt.com/share/6987426c-a2bc-800a-8cf3-7ee5f17ef031

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code

def scraper(month, year, run=True): for i in enforcement_actions: if month_year == month, year: if year >= 2013: create month_year.csv else: return "function cannot run because year is not within range"

- b. Create Dynamic Scraper

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

def scrape_enforcement_actions_from(month, year, out_dir=".", run=True):
    # Skip running (for knitting)
    if not run:
        print("Scraper skipped (run=False).")
        return pd.DataFrame(columns=["Title", "Date", "Category", "Link"])

    # Guardrail: only 2013+
    if year < 2013:
        print("Please restrict your input to year >= 2013. Only enforcement actions after
        ↪  2013 are listed.")
        return pd.DataFrame(columns=["Title", "Date", "Category", "Link"])

    # Validate month
    if not (1 <= month <= 12):
        raise ValueError("month must be an integer from 1 to 12.")

    start_dt = date(year, month, 1)

    session = requests.Session()
    headers = {"User-Agent": "Mozilla/5.0"}

    # Define selectors ONCE (also prevents PDF cutoff)
    CARD_SELECTOR = (
        "li.usa-card.card--list.pep-card--minimal."
        "mobile\\:grid-col-12"
    )
    TAG_SELECTOR = (
        "li.display-inline-block.usa-tag.text-no-lowercase."
        "text-base-darkest.bg-base-lightest.margin-right-1"
    )

    rows = []
    page = 1

    while True:
        page_url = BASE_URL if page == 1 else f"{BASE_URL}?page={page}"

        resp = session.get(page_url, headers=headers, timeout=30)
        resp.raise_for_status()

        soup = BeautifulSoup(resp.text, "lxml")   # <-- THIS was missing
        cards = soup.select(CARD_SELECTOR)

        # Stop when no enforcement cards remain
        if not cards:
            break
```

```python
        page_dates = []

        for card in cards:
            # Title + link
            h2 = card.find("h2")
            a = h2.find("a", href=True) if h2 else card.find("a", href=True)
            title = a.get_text(strip=True) if a else None
            link = urljoin(BASE_URL, a["href"]) if a and a.has_attr("href") else None

            # Date
            date_span = card.select_one("span.text-base-dark.padding-right-105")
            date_str = date_span.get_text(strip=True) if date_span else None

            parsed_dt = None
            if date_str:
                parsed = pd.to_datetime(date_str, errors="coerce")
                if pd.notna(parsed):
                    parsed_dt = parsed.date()
                    page_dates.append(parsed_dt)

            # Category
            cat_tag = card.select_one(TAG_SELECTOR)
            category = cat_tag.get_text(strip=True) if cat_tag else None

            # Keep only rows on/after start
            if parsed_dt and parsed_dt >= start_dt:
                rows.append(
                    {"Title": title, "Date": date_str, "Category": category, "Link":
↪  link}
                )

        # Early stop if we've crossed the start date
        if page_dates and min(page_dates) < start_dt:
            break

        page += 1
        time.sleep(1)  # be polite

    df = pd.DataFrame(rows).dropna(subset=["Title"]).reset_index(drop=True)

    out_path = f"{out_dir.rstrip('/')}/enforcement_actions_{year}_{month:02d}.csv"
    df.to_csv(out_path, index=False)
    print(f"Saved CSV to: {out_path}")

    return df
```

- c. Test Your Code

2024 Data:

```python
enforcement_csv = scrape_enforcement_actions_from(month=1, year=2024, run=False)
```

```
enforcement_csv.head()

print(f'There are {len(enforcement_csv)} enforcement actions after 2013 listed')

enforcement_csv.tail()
```

```
Scraper skipped (run=False).
There are 0 enforcement actions after 2013 listed
```

Title	Date	Category	Link

2022 Data:

```
enforcement_csv = scrape_enforcement_actions_from(month=1, year=2022, run=False)

print(f'There are {len(enforcement_csv)} enforcement actions after 2013 listed')
```

```
Scraper skipped (run=False).
There are 0 enforcement actions after 2013 listed
```

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

```
enforcement_csv =
↪  pd.read_csv("/Users/abrahamsadat/Downloads/enforcement_actions_2022_01.csv")
```

```
enforcement_csv['Date'] = pd.to_datetime(enforcement_csv['Date'], errors='coerce')

# create year-month column
enforcement_csv["year_month"] = enforcement_csv["Date"].dt.to_period("M").astype(str)

enforcement_csv_grouped =
↪  enforcement_csv.groupby('year_month').size().reset_index(name='size_count')
```
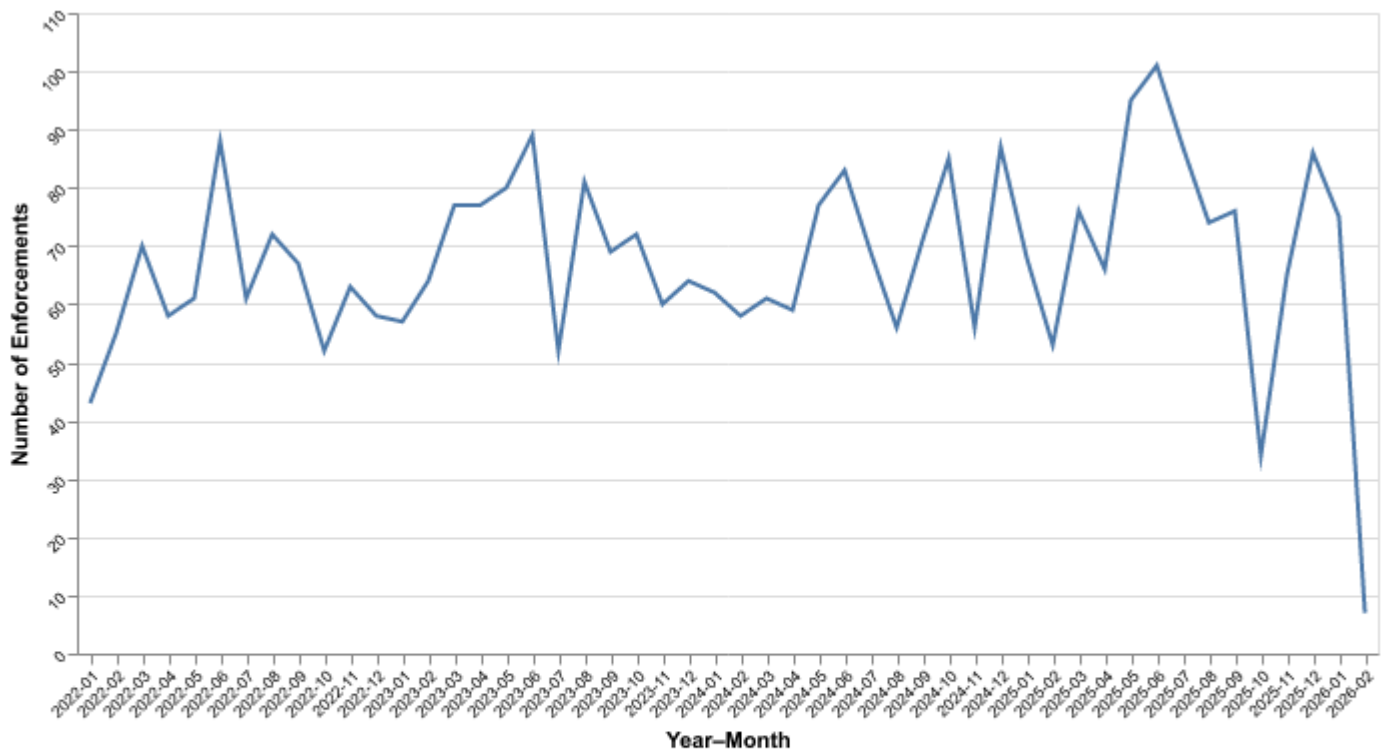
```
chart = (
    alt.Chart(enforcement_csv_grouped)
    .mark_line()
    .encode(
        x=alt.X("year_month:O", title="Year-Month"),
        y=alt.Y("size_count:Q", title="Number of Enforcements"),
    )
    .properties(
```

```
        width=650,
        height=320,
        padding={"left": 10, "right": 10, "top": 10, "bottom": 120},
    )
    .configure_axis(
        labelAngle=-45,
        labelFontSize=8,
        titleFontSize=11,
        labelLimit=200,
    )
)

chart
```


2. Plot the number of enforcement actions categorized:

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

```python
criminal_and_civil_actions = enforcement_csv[enforcement_csv['Category'] == 'Criminal and
↪  Civil Actions']

criminal_and_civil_actions_grouped = criminal_and_civil_actions.groupby('year_month'
↪  ).size().reset_index(name='size_count')

state_enforcement_agency = enforcement_csv[enforcement_csv['Category'] == 'State
↪  Enforcement Agencies']

state_enforcement_agency_grouped =
↪  state_enforcement_agency.groupby('year_month').size().reset_index(name='size_count')

criminal_and_civil_actions_grouped["type"] = "Criminal & Civil Actions"
state_enforcement_agency_grouped["type"] = "State Enforcement Agencies"

combined = pd.concat(
    [criminal_and_civil_actions_grouped, state_enforcement_agency_grouped],
    ignore_index=True
)
```
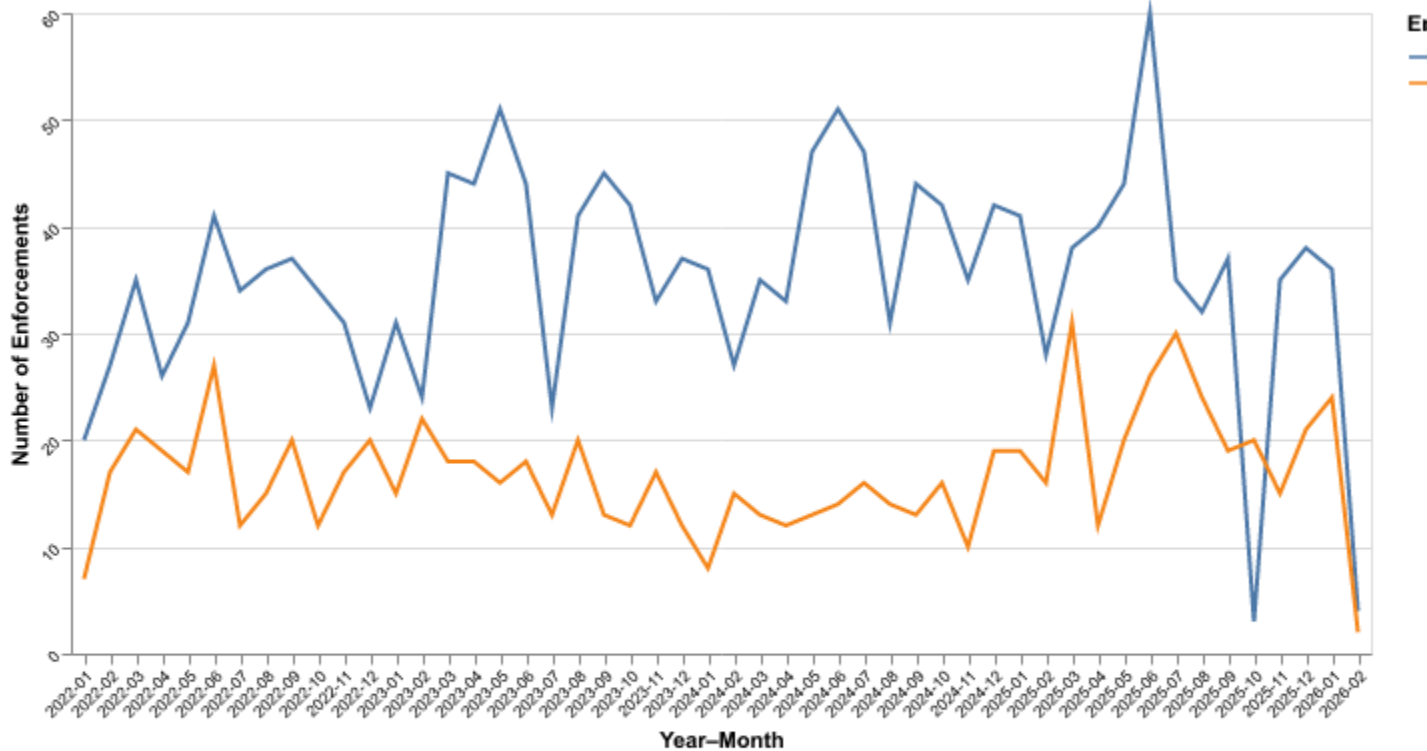
```python
chart2 = (
    alt.Chart(combined)
    .mark_line()
    .encode(
        x=alt.X("year_month:O", title="Year-Month"),
        y=alt.Y("size_count:Q", title="Number of Enforcements"),
        color=alt.Color("type:N", title="Enforcement Type"),
    )
    .properties(
        width=650,
        height=320,
        padding={"left": 10, "right": 10, "top": 10, "bottom": 120},
    )
    .configure_axis(
        labelAngle=-45,
        labelFontSize=8,
        titleFontSize=11,
        labelLimit=200,
    )
)

chart2
```

- based on five topics

```
#Restrict to Criminal and Civil Actions only
cca = enforcement_csv[
    enforcement_csv["Category"] == "Criminal and Civil Actions"
].copy()

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

    if any(word in title for word in ["health", "medicare", "medicaid", "hospital",
    ↪   "clinic", "physician", "pharmacy"]):
        return "Health Care Fraud"

    if any(word in title for word in ["bank", "financial", "wire", "loan", "credit",
    ↪   "tax", "securities"]):
        return "Financial Fraud"

    if any(word in title for word in ["drug", "opioid", "narcotic", "controlled
    ↪   substance", "fentanyl"]):
        return "Drug Enforcement"
```

```python
    if any(word in title for word in ["bribe", "bribery", "kickback", "corruption"]):
        return "Bribery/Corruption"

    return "Other"

cca["topic"] = cca["Title"].apply(classify_topic)
```

```python
def make_topic_chart(topic_name):
    df_topic = cca[cca["topic"] == topic_name]

    df_grouped = (
        df_topic
        .groupby("year_month")
        .size()
        .reset_index(name="size_count")
    )

    chart = (
        alt.Chart(df_grouped)
        .mark_line()
        .encode(
            x=alt.X("year_month:O", title="Year-Month"),
            y=alt.Y("size_count:Q", title="Number of Enforcements"),
        )
        .properties(
            width=650,
            height=280,
            padding={"left": 10, "right": 10, "top": 10, "bottom": 160},
        )
        .configure_axis(
            labelAngle=-45,
            labelFontSize=8,
            titleFontSize=11,
            labelLimit=200,
        )
    )

    return chart
```
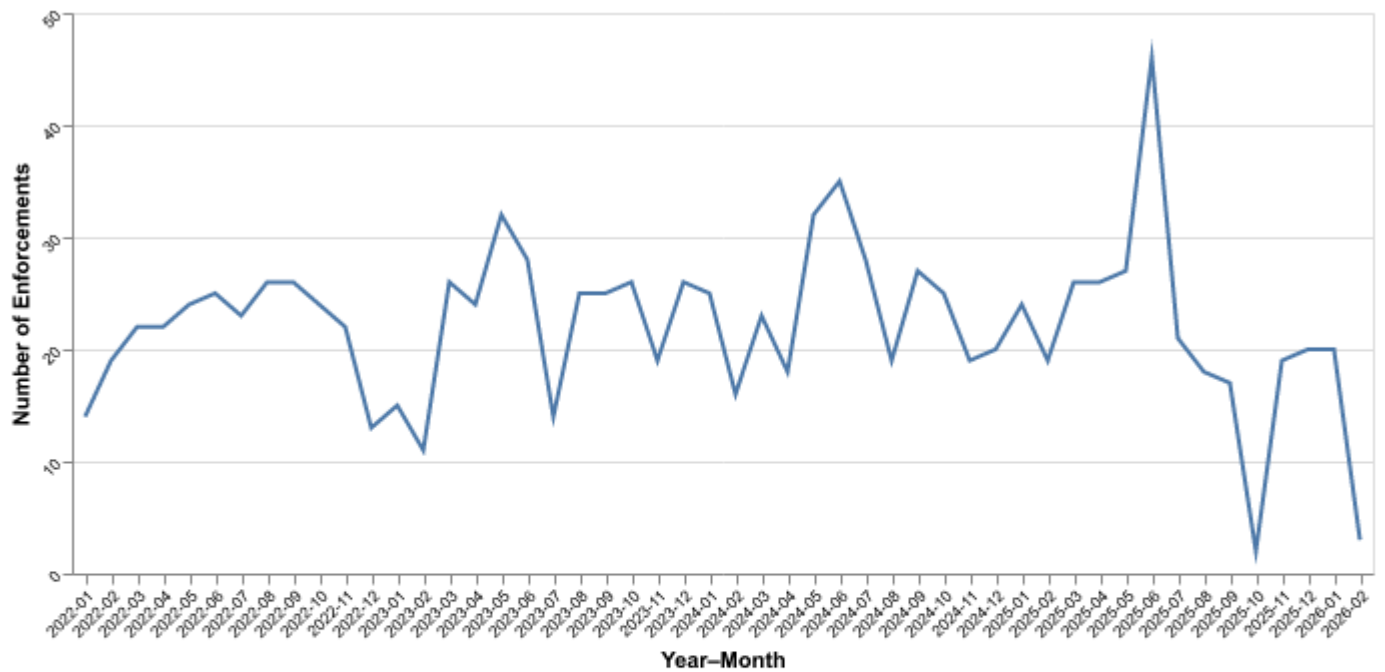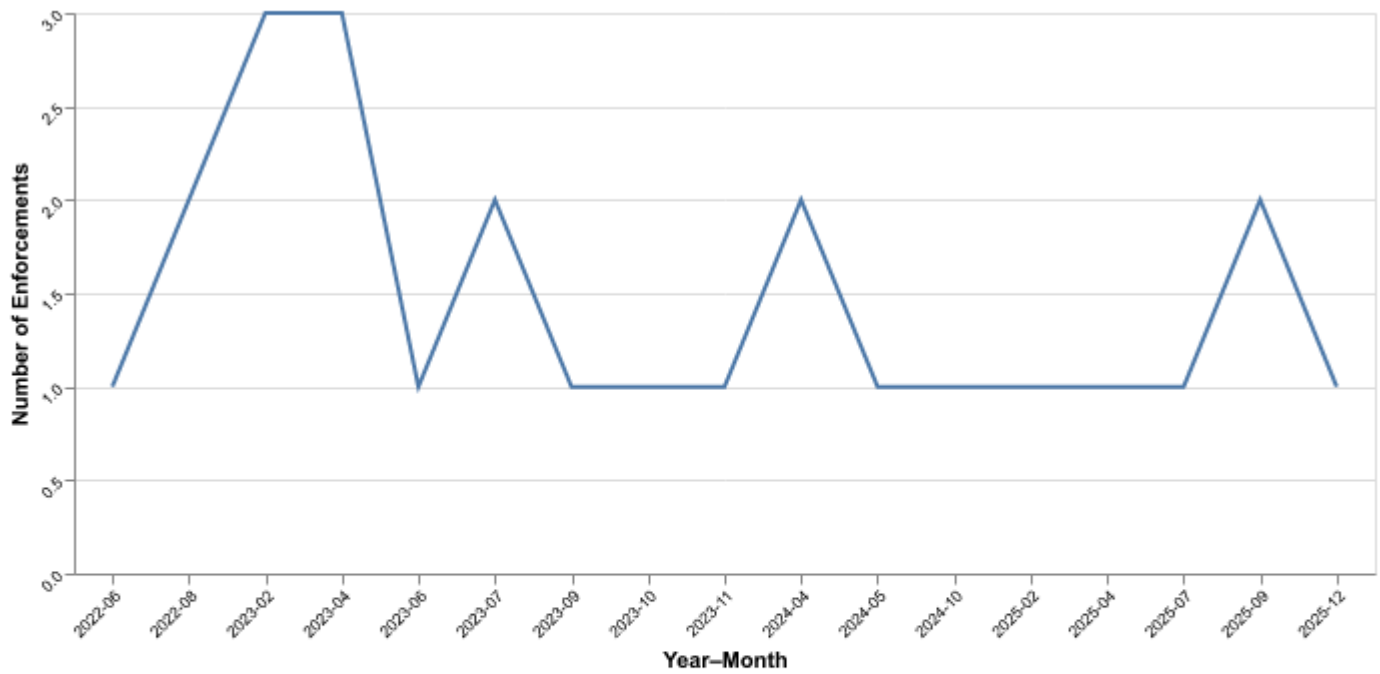
Health Care Fraud:

```python
health_care_fraud_chart = make_topic_chart("Health Care Fraud")
health_care_fraud_chart
```
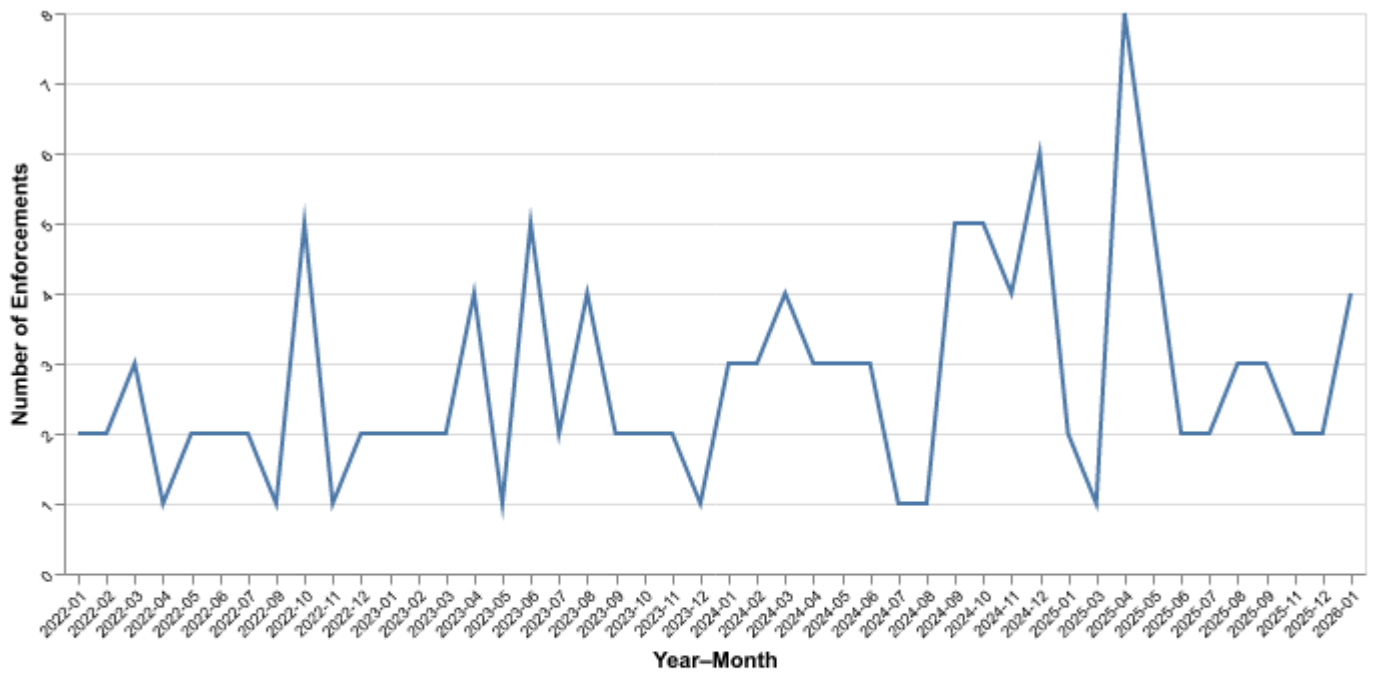
Financial Fraud:

```
financial_fraud_chart = make_topic_chart("Financial Fraud")
financial_fraud_chart
```
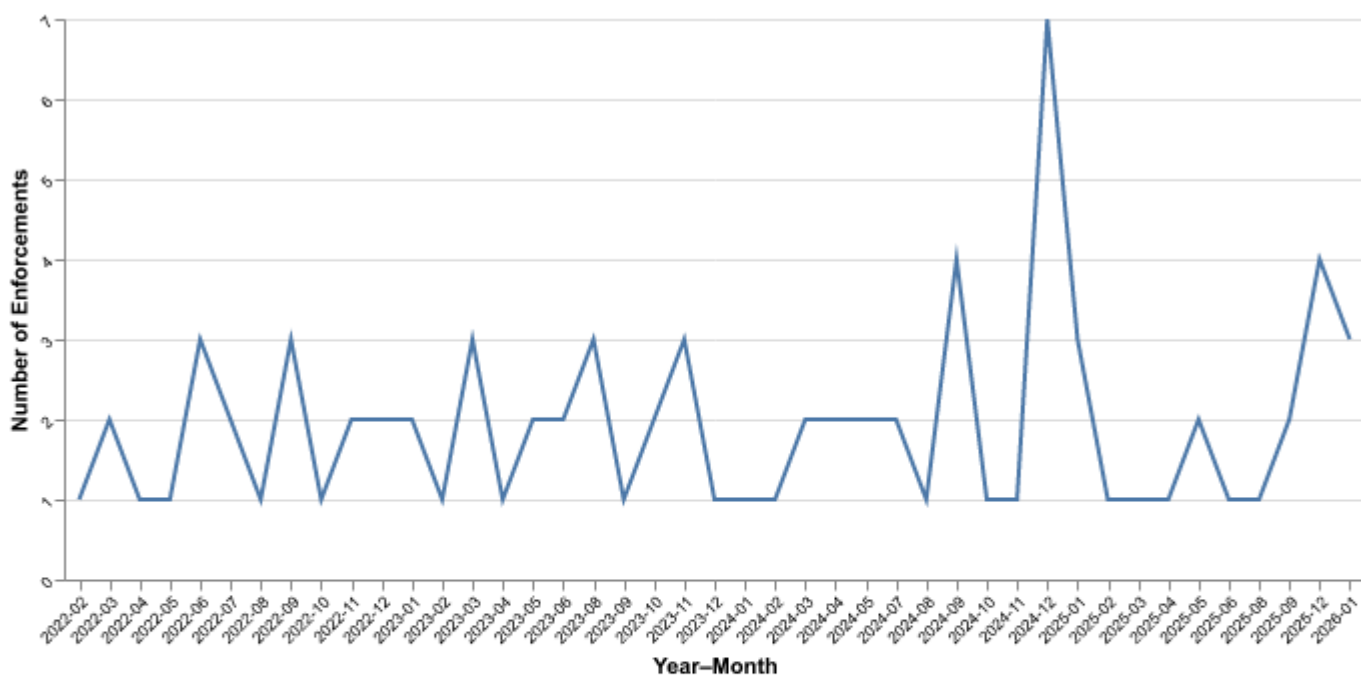
Drug Enforcement:

```
drug_enforcement_chart = make_topic_chart("Drug Enforcement")
drug_enforcement_chart
```

Bribery/Corruption:

```
bribery_corruption_chart = make_topic_chart("Bribery/Corruption")
bribery_corruption_chart
```

Other

```
other_chart = make_topic_chart("Other")
other_chart
```

Year–Month