PS4

Yice Zhang

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: YZ

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
from urllib.parse import urljoin

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
import pandas as pd
```

```
url = 'https://oig.hhs.gov/fraud/enforcement/'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'lxml')

actions = []

cards = soup.find_all('li', class_='usa-card')

for card in cards:
    h2 = card.find('h2', class_='usa-card__heading')
    a_tag = h2.find('a') if h2 else None
    title = a_tag.text.strip() if a_tag else None
    link = a_tag.get('href') if a_tag else None

    date_span = card.find('span', class_='text-base-dark')
    date = date_span.text.strip() if date_span else None

    category_tag = card.find('li', class_='usa-tag')
    category = category_tag.text.strip() if category_tag else None

    actions.append({
        'title': title,
        'date': date,
```

```
        'category': category,
        'link': link
    })

df = pd.DataFrame(actions)
df.head()
```

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

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code

FUNCTION scrape_enforcement_actions(start_month, start_year, run=False):

```
1. Input validation:
   - Check if year >= 2013
   - If not, print error message and return

2. Initialize:
   - Create empty list to store all actions
   - Set start_date from month/year parameters
   - Set page_number = 1
   - Set continue_scraping = True

3. Main crawling loop:
   WHILE continue_scraping:
       a. Construct URL with current page_number
       b. Make request and parse HTML
       c. Find all enforcement action cards

       d. FOR each card:
          - Extract title, date, category, link
          - Convert date string to datetime object
```

```
            - IF date >= start_date:
                - Add to actions list
            - ELSE:
                - Set continue_scraping = False
                - Break out of FOR loop

        e. IF no more cards found on page:
           - Set continue_scraping = False

        f. Increment page_number
        g. Sleep for 1 second

4. Convert to DataFrame
5. Save to CSV with filename "enforcement_actions_year_month.csv"
6. Return DataFrame
```

- b. Create Dynamic Scraper

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

def scrape_enforcement_actions_since(year: int, month: int, run_scraper: bool
 ↪  = False) -> pd.DataFrame:

    if not run_scraper:
        return pd.DataFrame()

    if year < 2013:
        return pd.DataFrame()

    start_date = pd.Timestamp(year=year, month=month, day=1)

    actions = []
    page = 1

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

        resp = requests.get(url, timeout=30)
        resp.raise_for_status()
        soup = BeautifulSoup(resp.text, "lxml")
```

```
        cards = soup.find_all("li", class_="usa-card")
        if not cards:
            break  # no more pages / content

        page_dates = []

        for card in cards:
            # title + link
            h2 = card.find("h2", class_="usa-card__heading")
            a_tag = h2.find("a") if h2 else None
            title = a_tag.get_text(strip=True) if a_tag else None
            link_raw = a_tag.get("href") if a_tag else None
            link = urljoin(BASE_URL, link_raw) if link_raw else None

            # date
            date_span = card.find("span", class_="text-base-dark")
            date_str = date_span.get_text(strip=True) if date_span else None
            date_dt = pd.to_datetime(date_str, errors="coerce")

            # category (could be multiple tags)
            cat_tags = card.find_all("li", class_="usa-tag")
            category = "; ".join([c.get_text(strip=True) for c in cat_tags])
↪   if cat_tags else None

            if pd.notna(date_dt):
                page_dates.append(date_dt)

            # keep only from start_date to today
            if pd.notna(date_dt) and date_dt >= start_date:
                actions.append({
                    "title": title,
                    "date": date_str,
                    "date_dt": date_dt,
                    "category": category,
                    "link": link
                })

        # stop crawling once this page already goes earlier than start_date
        if page_dates and min(page_dates) < start_date:
            break

        time.sleep(1)
```

```
        page += 1

    df = pd.DataFrame(actions)

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

    out_name = f"enforcement_actions_{year}_{month:02d}.csv"
    df.drop(columns=["date_dt"], errors="ignore").to_csv(out_name,
↪   index=False)
    print(f"Saved {out_name}. Rows = {len(df)}")

    return df
```

```
RUN_SCRAPER = False
df_2024 = scrape_enforcement_actions_since(2024, 1, run_scraper=RUN_SCRAPER)

df_2024.head()
```

```
n_actions_2024 = len(df_2024)
n_actions_2024
```

```
0
```

```
df_2024.columns
```

```
RangeIndex(start=0, stop=0, step=1)
```

```
# df_2024["date_dt"] = pd.to_datetime(df_2024["date"], errors="coerce")
# df_2024 = df_2024.dropna(subset=["date_dt"])

# earliest_2024 = df_2024.sort_values("date_dt", ascending=True).iloc[0]
# earliest_2024
```

- c. Test Your Code

```
RUN_SCRAPER = False
df_2022 = scrape_enforcement_actions_since(2022, 1, run_scraper=RUN_SCRAPER)

df_2022.head()
```

```
n_actions_2022 = len(df_2022)
n_actions_2022
```

```
0
```

```
# earliest_2022 = df_2022.sort_values("date_dt", ascending=True).iloc[0]
# earliest_2022[["date", "title", "category", "link"]]
```

```
## Step 3: Plot data based on scraped data

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

::: {.cell execution_count=12}
``` {.python .cell-code}
FILENAME = "enforcement_actions_2024_01.csv"

df = pd.read_csv(FILENAME)
df["date_dt"] = pd.to_datetime(df["date"], errors="coerce")

df = df.dropna(subset=["date_dt"]).copy()

df["month"] = df["date_dt"].dt.to_period("M").dt.to_timestamp()

df.head()
```

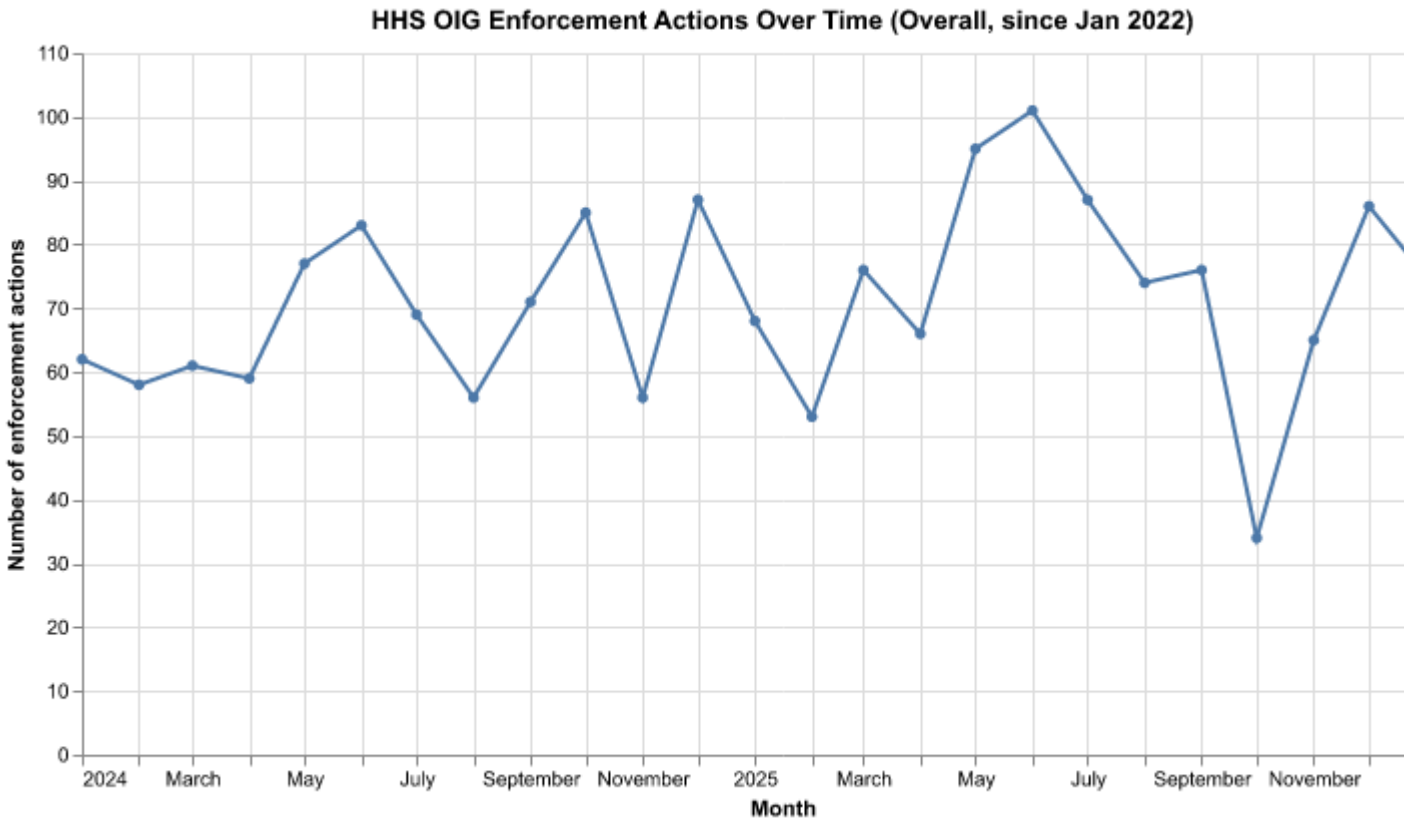
	title	date	category	li
0	Houston Transplant Doctor Indicted For Making ...	February 5, 2026	Criminal and Civil Actions	ht
1	MultiCare Health System to Pay Millions to Set...	February 4, 2026	Criminal and Civil Actions	ht
2	Brooklyn Banker Pleads Guilty to Laundering Pr...	February 3, 2026	COVID-19	ht
3	Attorney General Hanaway Obtains Medicaid Frau...	February 3, 2026	State Enforcement Agencies	ht
4	Delafield Man Sentenced to 18 Months' Imprison...	February 3, 2026	Criminal and Civil Actions	ht
```

:::

```
monthly_overall = (
    df.groupby("month")
      .size()
      .reset_index(name="count")
      .sort_values("month")
)

chart_overall = (
    alt.Chart(monthly_overall)
    .mark_line(point=True)
    .encode(
        x=alt.X("month:T", title="Month"),
        y=alt.Y("count:Q", title="Number of enforcement actions"),
        tooltip=[
            alt.Tooltip("month:T", title="Month"),
            alt.Tooltip("count:Q", title="Count")
        ]
    )
    .properties(
        title="HHS OIG Enforcement Actions Over Time (Overall, since Jan
 ↪  2022)",
        width=700,
        height=350
    )
)

chart_overall
```



2. Plot the number of enforcement actions categorized:

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

```
def to_broad_category(cat):
    if pd.isna(cat):
        return "Other"
    cat = str(cat)
    if "Criminal and Civil Actions" in cat:
        return "Criminal and Civil Actions"
    if "State Enforcement Agencies" in cat:
        return "State Enforcement Agencies"
    return "Other"

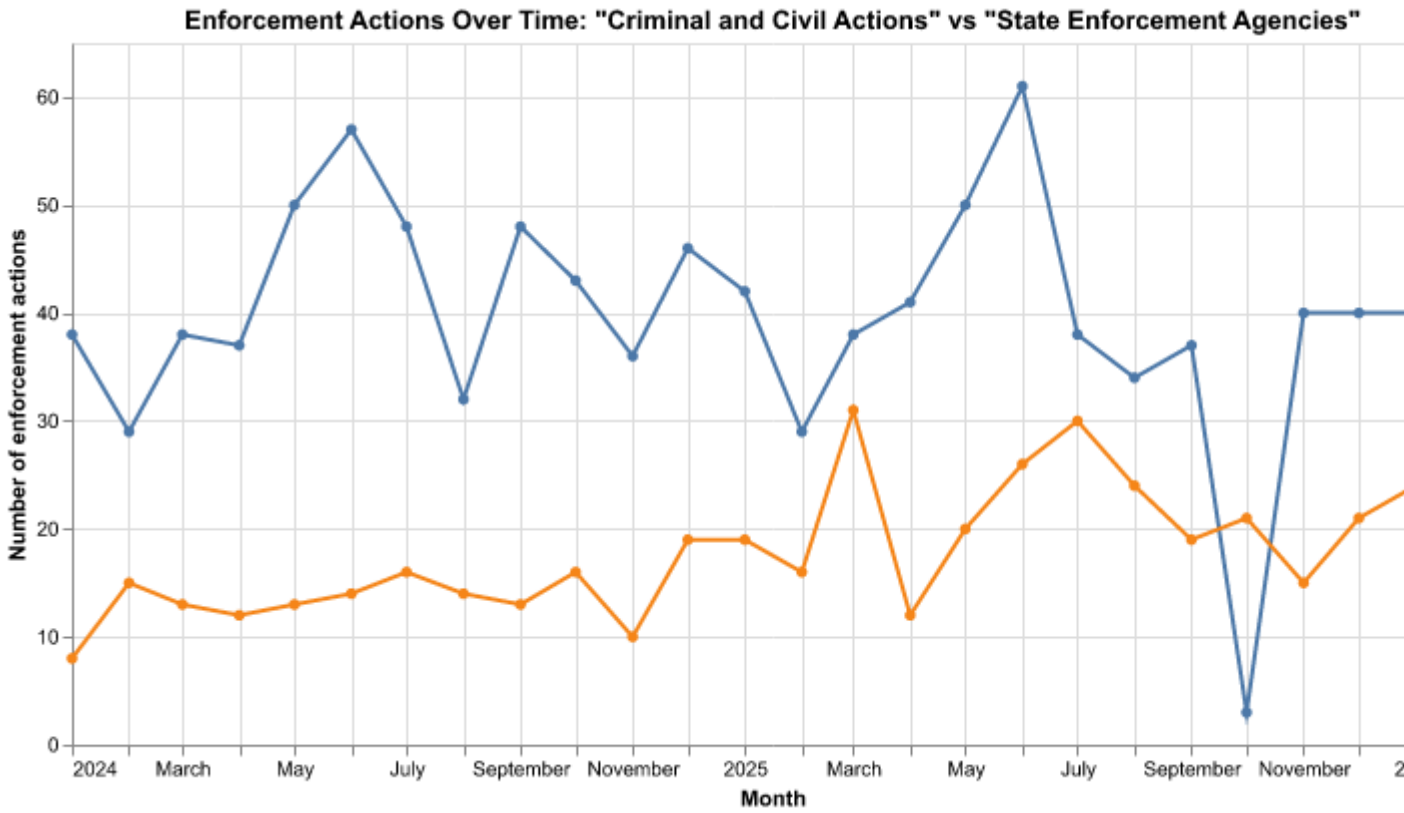
df["broad_category"] = df["category"].apply(to_broad_category)

df["broad_category"].value_counts()
```

```
monthly_by_broad = (
    df[df["broad_category"].isin(["Criminal and Civil Actions", "State
 ↪  Enforcement Agencies"])]
      .groupby(["month", "broad_category"])
      .size()
      .reset_index(name="count")
      .sort_values("month")
)

chart_broad = (
    alt.Chart(monthly_by_broad)
    .mark_line(point=True)
    .encode(
        x=alt.X("month:T", title="Month"),
        y=alt.Y("count:Q", title="Number of enforcement actions"),
        color=alt.Color("broad_category:N", title="Category"),
        tooltip=[
            alt.Tooltip("month:T", title="Month"),
            alt.Tooltip("broad_category:N", title="Category"),
            alt.Tooltip("count:Q", title="Count")
        ]
    )
    .properties(
        title='Enforcement Actions Over Time: "Criminal and Civil Actions" vs
 ↪  "State Enforcement Agencies"',
        width=700,
        height=350
    )
)

chart_broad
```



- based on five topics

```
criminal_df = df[df["broad_category"] == "Criminal and Civil Actions"].copy()
import re

def assign_topic(title):
    if pd.isna(title):
        return "Other"
    t = str(title).lower()

    drug_kw = [
        "fentanyl", "opioid", "controlled substance", "drug", "narcotic",
        ↪  "meth", "heroin", "pill", "overdose"
    ]
    if any(k in t for k in drug_kw):
        return "Drug Enforcement"

    bribery_kw = [
        "brib", "corrupt", "extortion", "racketeer", "money for", "payoff"
```

```
    ]
    if any(k in t for k in bribery_kw):
        return "Bribery/Corruption"

    financial_kw = [
        "bank", "wire", "financial", "loan", "mortgage", "investment",
        ↪ "securities", "tax",
        "money laundering", "launder", "credit", "debit", "check fraud"
    ]
    if any(k in t for k in financial_kw):
        return "Financial Fraud"

    healthcare_kw = [
        "medicare", "medicaid", "health care", "healthcare", "hospital",
        ↪ "clinic", "physician", "doctor",
        "nurse", "hospice", "pharmacy", "lab", "laboratory", "billing",
        ↪ "claims", "medical"
    ]
    if any(k in t for k in healthcare_kw):
        return "Health Care Fraud"

    return "Other"

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

criminal_df["topic"].value_counts()

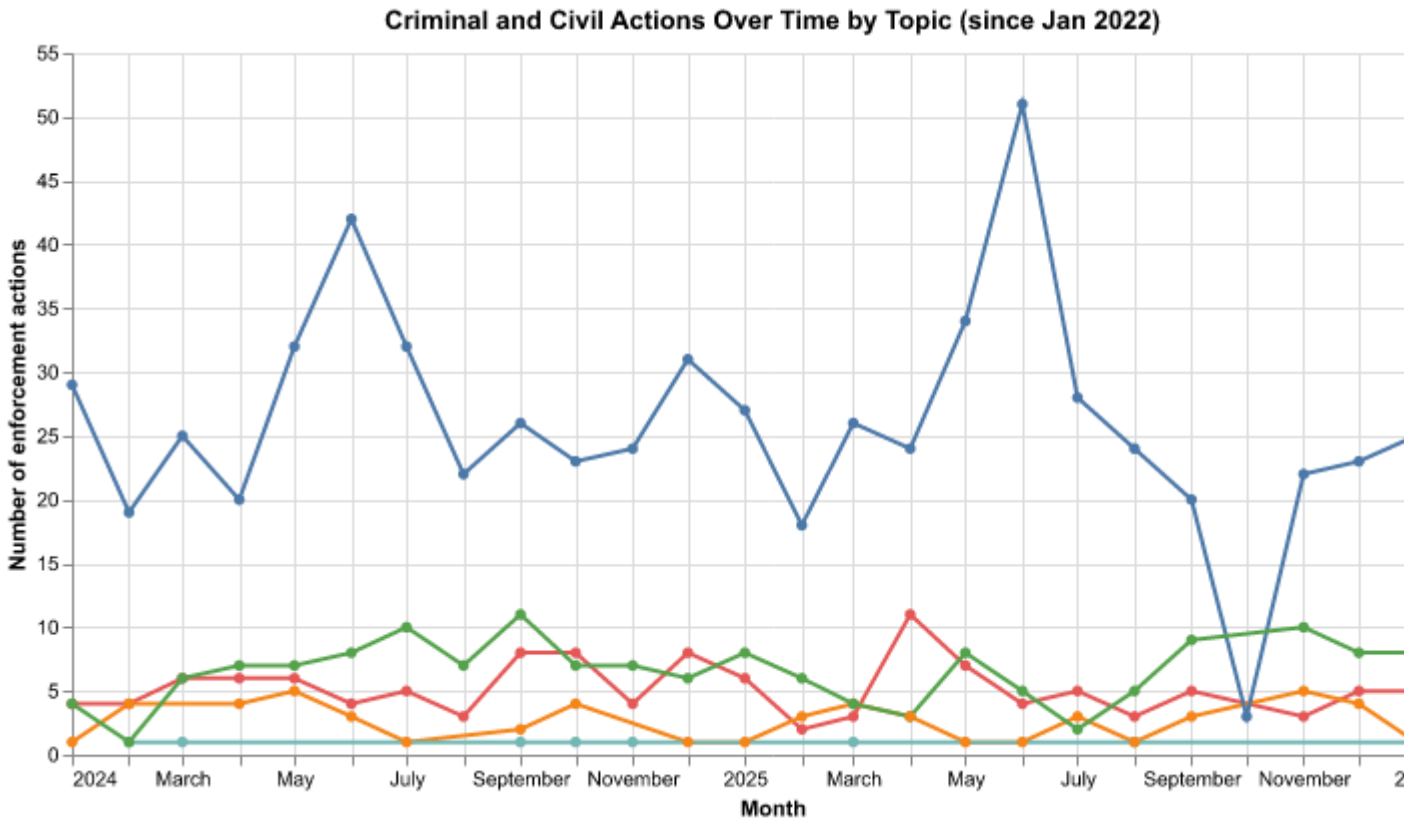
monthly_by_topic = (
    criminal_df.groupby(["month", "topic"])
               .size()
               .reset_index(name="count")
               .sort_values("month")
)

topic_order = [
    "Health Care Fraud", "Financial Fraud", "Drug Enforcement",
    ↪ "Bribery/Corruption", "Other"
]

chart_topic = (
    alt.Chart(monthly_by_topic)
    .mark_line(point=True)
```

```
    .encode(
        x=alt.X("month:T", title="Month"),
        y=alt.Y("count:Q", title="Number of enforcement actions"),
        color=alt.Color("topic:N", sort=topic_order, title="Topic"),
        tooltip=[
            alt.Tooltip("month:T", title="Month"),
            alt.Tooltip("topic:N", title="Topic"),
            alt.Tooltip("count:Q", title="Count")
        ]
    )
    .properties(
        title='Criminal and Civil Actions Over Time by Topic (since Jan
↪   2022)',
        width=700,
        height=350
    )
)

chart_topic
```


Criminal and Civil Actions Over Time by Topic (since Jan 2022)
Number of enforcement actions
55
50
45
40
35
30
25
20
15
10
5
0
2024
March
May
July
September
November
2025
March
May
July
September
November
Month