title

author

Invalid Date

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: HMF

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

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

Step 1: Develop initial scraper and crawler

```python
import requests
from bs4 import BeautifulSoup
import pandas as pd

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

headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
}

response = requests.get(url, headers=headers)
soup = BeautifulSoup(response.text, "html.parser")

rows = soup.select(".views-row")
print("Rows found:", len(rows))  # sanity check

data = []

for row in rows:
    title_tag = row.select_one("h3 a")
    date_tag = row.select_one(".date-display-single")
    category_tag = row.select_one(".views-field-field-category a")

    data.append({
        "Title": title_tag.text.strip(),
        "Date": date_tag.text.strip(),
        "Category": category_tag.text.strip(),
        "Link": "https://oig.hhs.gov" + title_tag["href"]
    })
```

```
df = pd.DataFrame(data)
print(df.head())
```

```
Rows found: 0
Empty DataFrame
Columns: []
Index: []
```

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code

To be able to specifiy cut-offs at specific dates, I used a while loop for my scraper function.

FUNCTION scrape_enforcement_actions(start_month, start_year, run_scraper):

```
IF start_year < 2013:
    PRINT warning message
    RETURN None

IF run_scraper is False:
    PRINT message and exit function

DEFINE cutoff_date = first day of (start_year, start_month)
DEFINE base_url
DEFINE page = 0
DEFINE keep_scraping = True
INITIALIZE empty list to store results

WHILE keep_scraping:
    REQUEST page (base_url?page=page)
    PARSE HTML

    SELECT all enforcement action rows

    FOR each row:
        EXTRACT title, link, category, date
        CONVERT date string to datetime

        IF date < cutoff_date:
            SET keep_scraping = False
            BREAK out of loop

        ADD row data to results list

    WAIT 1 second
    INCREMENT page by 1

CONVERT results list to dataframe
SAVE dataframe as CSV
RETURN dataframe
```

- b. Create Dynamic Scraper

```python
from datetime import datetime
import time

def scrape_enforcement_actions(start_month, start_year, run_scraper=True):

    if start_year < 2013:
        print("Please restrict to year >= 2013. Enforcement actions prior to
         ↪  2013 are not listed.")
        return None

    if not run_scraper:
        print("Scraper indicator is off. Function will not run.")
        return None

    cutoff_date = datetime(start_year, start_month, 1)

    base_url = "https://oig.hhs.gov/fraud/enforcement/"
    headers = {
        "User-Agent": (
            "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) "
            "AppleWebKit/537.36 (KHTML, like Gecko) "
            "Chrome/120.0.0.0 Safari/537.36"
        )
    }

    page = 0
    keep_scraping = True
    data = []
    while keep_scraping:
        url = f"{base_url}?page={page}"
        response = requests.get(url, headers=headers)
        soup = BeautifulSoup(response.text, "html.parser")

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


        if not rows:
            break

        for row in rows:
            title_tag = row.select_one("h3 a")
            date_tag = row.select_one(".date-display-single")
            category_tag = row.select_one(".views-field-field-category a")

            date = datetime.strptime(date_tag.text.strip(), "%B %d, %Y")

            if date < cutoff_date:
                keep_scraping = False
                break

            data.append({
                "Title": title_tag.text.strip(),
                "Date": date.strftime("%Y-%m-%d"),
                "Category": category_tag.text.strip(),
                "Link": "https://oig.hhs.gov" + title_tag["href"]
            })

        page += 1
        time.sleep(1)

    df = pd.DataFrame(data)

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

    return df

df_2024 = scrape_enforcement_actions(
    start_month=1,
    start_year=2024,
    run_scraper=True
)

scrape_enforcement_actions(
    start_month=1,
    start_year=2024,
    run_scraper=False
)

len(df_2024)
df_2024.tail(1)
```

```
Scraper indicator is off. Function will not run.
```

- c. Test Your Code

```
df_2022 = scrape_enforcement_actions(1, 2022, True)
len(df_2022)
df_2022.tail(1)
```

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

I was not ultimately successful in scraping the data into a csv file, but if I had been able to assemble this dataset, I would use the following code to create the subsequent three altair plots:

df = pd.read_csv("enforcement_actions_2022_01.csv")

df['Date'] = pd.to_datetime(df['Date']) df.head()

df['YearMonth'] = df['Date'].dt.to_period('M') monthly_counts = df.groupby('YearMonth').size().reset_index(name='Count') monthly_counts['YearMonth'] = monthly_counts['YearMonth'].astype(str) monthly_counts.head()

import altair as alt

chart_overall = alt.Chart(monthly_counts).mark_line(point=True).encode(x='YearMonth:T', y='Count:Q').properties(title='Number of Enforcement Actions per Month').interactive()

chart_overall

2. Plot the number of enforcement actions categorized:

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

df['CategorySimple'] = df['Category'].apply(lambda x: 'Criminal and Civil Actions' if 'Criminal' in x else 'State Enforcement Agencies')

df['YearMonth'] = df['Date'].dt.to_period('M') monthly_category = df.groupby(['YearMonth', 'CategorySimple']).size().reset_index(name='Count') monthly_category['YearMonth'] = monthly_category['YearMonth'].astype(str)

chart_category = alt.Chart(monthly_category).mark_line(point=True).encode(x='YearMonth:T', y='Count:Q', color='CategorySimple:N').properties(title='Enforcement Actions by Category Over Time').interactive()

chart_category

- based on five topics

import re

def assign_topic(title): title = title.lower() if re.search(r'health|medicare|medicaid|hospital|patient', title): return 'Health Care Fraud' elif re.search(r'financial|bank|credit|loan', title): return 'Financial Fraud' elif re.search(r'drug|controlled substance|opioid|pharmacy', title): return 'Drug Enforcement' elif re.search(r'bribery|corruption|kickback', title): return 'Bribery/Corruption' else: return 'Other'

df['Topic'] = df.apply(lambda row: assign_topic(row['Title']) if row['CategorySimple']=='Criminal and Civil Actions' else 'State Enforcement Agencies', axis=1)

monthly_topic = df.groupby(['YearMonth','Topic']).size().reset_index(name='Count') monthly_topic['YearMonth'] = monthly_topic['YearMonth'].astype(str)

chart_topic = alt.Chart(monthly_topic).mark_line(point=True).encode(x='YearMonth:T', y='Count:Q', color='Topic:N').properties(title='Enforcement Actions by Topic Over Time').interactive()

chart_topic