ps5

Cedar Liu

2024-11-09

Due 11/9 at 5:00PM Central. Worth 100 points + 10 points extra credit.

Submission Steps (10 pts)

1. This problem set is a paired problem set.
2. Play paper, scissors, rock to determine who goes first. Call that person *Partner 1*.
    - Partner 1 (name and cnet ID): Cedar Liu, 12424464
    - Partner 2 (name and cnet ID): None
3. Partner 1 will accept the `ps5` and then share the link it creates with their partner. You can only share it with one partner so you will not be able to change it after your partner has accepted.
4. "This submission is our work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: **_CL_** **___**
5. "I have uploaded the names of anyone else other than my partner and I worked with on the problem set **here**" (1 point)
6. Late coins used this pset: **_0_** Late coins left after submission: **_1_**
7. Knit your `ps5.qmd` to an PDF file to make `ps5.pdf`,
    - The PDF should not be more than 25 pages. Use `head()` and re-size figures when appropriate.
8. (Partner 1): push `ps5.qmd` and `ps5.pdf` to your github repo.
9. (Partner 1): submit `ps5.pdf` via Gradescope. Add your partner on Gradescope.
10. (Partner 1): tag your submission in Gradescope

```
import pandas as pd
import altair as alt
import time
from datetime import datetime
import re
import requests
from bs4 import BeautifulSoup
import geopandas as gpd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
alt.renderers.enable("png")
```

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

Step 1: Develop initial scraper and crawler

1. Scraping (PARTNER 1)

```
# Send a GET request to the page
oig_hhs_url = 'https://oig.hhs.gov/fraud/enforcement/'
oig_hhs_res = requests.get(oig_hhs_url)
# Parse the page content with bs
oig_hhs_soup = BeautifulSoup(oig_hhs_res.text, 'lxml')
# Initialize lists to store the data
titles = []
dates = []
categories = []
links = []
# Loop through each enforcement action entry
for action in oig_hhs_soup.find_all('li', class_=
'usa-card card--list pep-card--minimal mobile:grid-col-12'):
    title_tag = action.find('h2', class_='usa-card__heading').find('a')
    title = title_tag.get_text(strip=True)
    link = title_tag['href']
    full_link = f'https://oig.hhs.gov{link}'
    titles.append(title)
    links.append(full_link)

    date_tag = action.find('span', class_='text-base-dark padding-right-105')
```

```
    date = date_tag.get_text(strip=True)
    dates.append(date)

    category_tag = action.find('ul', class_='display-inline
↪   add-list-reset').find('li')
    category = category_tag.get_text(strip=True)
    categories.append(category)
# Create a DataFrame from the collected data
oig_hhs_df = pd.DataFrame({
    'Title': titles,
    'Date': dates,
    'Category': categories,
    'Link': links
})
print(oig_hhs_df.head())
```

```
                                               Title              Date  \
0  Pharmacist and Brother Convicted of $15M Medic...  November 8, 2024
1  Boise Nurse Practitioner Sentenced To 48 Month...  November 7, 2024
2  Former Traveling Nurse Pleads Guilty To Tamper...  November 7, 2024
3  Former Arlington Resident Sentenced To Prison ...  November 7, 2024
4  Paroled Felon Sentenced To Six Years For Fraud...  November 7, 2024

                    Category  \
0  Criminal and Civil Actions
1  Criminal and Civil Actions
2  Criminal and Civil Actions
3  Criminal and Civil Actions
4  Criminal and Civil Actions

                                                Link
0  https://oig.hhs.gov/fraud/enforcement/pharmaci...
1  https://oig.hhs.gov/fraud/enforcement/boise-nu...
2  https://oig.hhs.gov/fraud/enforcement/former-t...
3  https://oig.hhs.gov/fraud/enforcement/former-a...
4  https://oig.hhs.gov/fraud/enforcement/paroled-...
```

2. Crawling (PARTNER 1)

```
agencies = []
# Loop through each link in the DataFrame to extract the agency
for link in oig_hhs_df['Link']:
    try:
        response = requests.get(link)
        soup = BeautifulSoup(response.text, 'lxml')
        agency_label = soup.find('span', text="Agency:")
        agency = agency_label.find_next_sibling(text=True).strip()
        agencies.append(agency)
        time.sleep(1)
    except Exception as e:
        agencies.append("Error retrieving agency")
        print(f"Error with link {link}: {e}")
oig_hhs_df['Agency'] = agencies
print(oig_hhs_df.head())
```

```
Error with link
https://oig.hhs.gov/fraud/enforcement/michael-depalma-md-and-virginia-i-spine-physicians-agre
'NoneType' object has no attribute 'find_next_sibling'
Error with link
https://oig.hhs.gov/fraud/enforcement/mercy-health-youngstown-agreed-to-pay-69000-for-allege
'NoneType' object has no attribute 'find_next_sibling'
                                                Title              Date  \
0  Pharmacist and Brother Convicted of $15M Medic...  November 8, 2024
1  Boise Nurse Practitioner Sentenced To 48 Month...  November 7, 2024
2  Former Traveling Nurse Pleads Guilty To Tamper...  November 7, 2024
3  Former Arlington Resident Sentenced To Prison ...  November 7, 2024
4  Paroled Felon Sentenced To Six Years For Fraud...  November 7, 2024

                    Category  \
0  Criminal and Civil Actions
1  Criminal and Civil Actions
2  Criminal and Civil Actions
3  Criminal and Civil Actions
4  Criminal and Civil Actions

                                                Link  \
0  https://oig.hhs.gov/fraud/enforcement/pharmaci...
1  https://oig.hhs.gov/fraud/enforcement/boise-nu...
2  https://oig.hhs.gov/fraud/enforcement/former-t...
3  https://oig.hhs.gov/fraud/enforcement/former-a...
```

```
4  https://oig.hhs.gov/fraud/enforcement/paroled-...

                                                Agency
0                          U.S. Department of Justice
1  November 7, 2024; U.S. Attorney's Office, Dist...
2  U.S. Attorney's Office, District of Massachusetts
3  U.S. Attorney's Office, Eastern District of Vi...
4  U.S. Attorney's Office, Middle District of Flo...
```

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code (PARTNER 2)

def scrape_enforcement_actions(year, month):

```
# Step 1: Validate input year
if year < 2013:
    print("Please provide a year >= 2013, as only enforcement actions from
    2013 onwards are available.")
    return
# Step 2: Initialize variables
base_url = "https://oig.hhs.gov/fraud/enforcement/"
data = []
start_date = datetime(year, month, 1)
today = datetime.today()
# Step 3: Start page loop
page = 1
stop_scraping = False
while not stop_scraping:
    url = f"{base_url}?page={page}"
    for attempt in range(3):
        try:
            response = request.get(url)
            break
        except requests.exceptions.RequestException as e:
            print(f'Error retrieving page {page}: {e}. Retrying...')
            time.sleep(5)
    else:
        print('Failed to retrieve page after multiple attempts. Stopping.')
        break
```

```
    soup = BeautifulSoup(response.text, 'html.parser')
    actions = soup.find_all('li', class_='usa-card card--list
    pep-card--minimal mobile:grid-col-12')
    if not actions:
        break
    # Step 4: Process each action on the page
    for action in actions:
        # Extract details for each enforcement action
        title_tag = action.find('h2', class_='usa-card__heading').find('a')
        title = title_tag.get_text(strip=True)
        link = title_tag['href']
        full_link = f'https://oig.hhs.gov{link}'

        # Extract date and convert it to datetime
        date_tag = action.find('span', class_='text-base-dark
        padding-right-105')
        date_text = date_tag.get_text(strip=True)
        try:
            action_date = datetime.strptime(date_text, '%B %d, %Y')
        except ValueError:
            continue
        # Only collect actions within the desired date range
        if action_date < start_date:
            print('Reached start date. Stopping scraping.')
            stop_scraping = True
            break
        if action_date <= today:
            category_tag = action.find('ul', class_='display-inline
            add-list-reset').find('li')
            category = category_tag.get_text(strip=True)

            data.append({
                'Title': title,
                'Date': date_text,
                'Category': category,
                'Link': full_link
            })
    # Step 5: Move to the next page and wait
    page += 1
    time.sleep(1)
# Step 6: Convert to DataFrame and save to CSV
df = pd.DataFrame(data)
output_filename = f"enforcement_actions_{year}_{month}.csv"
```

```
df.to_csv(output_filename, index=False)
print(f"Data saved to {output_filename}")
```

Discussion of the Loop: Use a while loop to iterate through each pag. Initialize a variable page set to 1 (to start from the first page). Construct the URL by appending ?page=page to the base URL. Fetch the page content and parse it. For each enforcement action on the page: Extract the Title, Date, Category, and Link. Check if the Date of the action falls between the start date and today. If it does, add the enforcement action details to the list. If it's older than the start date, break the loop as there's no need to proceed further. Increment the page number. Add a time.sleep(1) to prevent server overload.

- b. Create Dynamic Scraper (PARTNER 2)

```
def scrape_enforcement_actions(year, month):
    if year < 2013:
        print("Please provide a year >= 2013, as only enforcement actions
        ↪  from 2013 onwards are available.")
        return
    base_url = "https://oig.hhs.gov/fraud/enforcement/"
    data = []
    start_date = datetime(year, month, 1)
    today = datetime.today()
    page = 1
    stop_scraping = False
    while not stop_scraping:
        print(f'Fetching page {page}...')
        url = f"{base_url}?page={page}"
        for attempt in range(3):
            try:
                response = requests.get(url)
                #print(f'Page {page} fetched successfully.')
                break
            except requests.exceptions.RequestException as e:
                #print(f"Error retrieving page {page}: {e}. Retrying...")
                time.sleep(5)
        else:
            #print("Failed to retrieve page after multiple attempts.
            ↪  Stopping.")
            break

        soup = BeautifulSoup(response.text, 'lxml')
        actions = soup.find_all('li', class_='usa-card card--list
↪  pep-card--minimal mobile:grid-col-12')
```

```
        if not actions:
            #print('No more actions found, ending scraping.')
            break
        #print(f'Processing actions on pages {page}...')
        for action in actions:
            title_tag = action.find('h2',
↪  class_='usa-card__heading').find('a')
            title = title_tag.get_text(strip=True)
            link = title_tag['href']
            full_link = f'https://oig.hhs.gov{link}'

            date_tag = action.find('span', class_='text-base-dark
↪  padding-right-105')
            date_text = date_tag.get_text(strip=True)
            try:
                action_date = datetime.strptime(date_text, '%B %d, %Y')
            except ValueError:
                #print(f'Skipping action due to date format issue:
                 ↪  {date_text}')
                continue
            if action_date < start_date:
                #print('Reached start date. Stopping scraping.')
                stop_scraping = True
                break
            if action_date <= today:
                category_tag = action.find('ul', class_='display-inline
↪  add-list-reset').find('li')
                category = category_tag.get_text(strip=True)
                data.append({
                    'Title': title,
                    'Date': date_text,
                    'Category': category,
                    'Link': full_link
                })
        if stop_scraping:
          break
        page += 1
        time.sleep(1)
    #print('Coverting data to DataFrame...')
    df = pd.DataFrame(data)
    output_filename =
↪  "D:/Github/vsc/hwk5/enforcement_actions_{}_{}.csv".format(year, month)
```

```
    try:
        df.to_csv(output_filename, index=False)
        #print(f"Data saved to {output_filename}")
    except Exception as e:
        print(f"Error saving file: {e}")
scrape_enforcement_actions(2023, 1)
```

```
# Load the CSV file into a DataFrame
enf_df = pd.read_csv("enforcement_actions_2023_1.csv")
enf_df['Date'] = pd.to_datetime(enf_df['Date'],
 ↪  errors='coerce').dt.strftime('%Y-%m-%d')
# Number of enforcement actions
num_actions = enf_df.shape[0]
print(f"Total number of enforcement actions: {num_actions}")
# Details of the earliest enforcement action
earliest_action = enf_df.sort_values(by="Date").iloc[0]
print("Earliest enforcement action:")
print(earliest_action)
```

```
Total number of enforcement actions: 1534
Earliest enforcement action:
Title       Podiatrist Pays $90,000 To Settle False Billin...
Date                                                2023-01-03
Category                           Criminal and Civil Actions
Link        https://oig.hhs.gov/fraud/enforcement/podiatri...
Name: 1533, dtype: object
```

- c. Test Partner's Code (PARTNER 1)

```
def scrape_enforcement_actions(year, month):
    if year < 2013:
        print("Please provide a year >= 2013, as only enforcement actions
         ↪  from 2013 onwards are available.")
        return
    base_url = "https://oig.hhs.gov/fraud/enforcement/"
    data = []
    start_date = datetime(year, month, 1)
    today = datetime.today()
    page = 1
    stop_scraping = False
    while not stop_scraping:
```

```
        #print(f'Fetching page {page}...')
        url = f"{base_url}?page={page}"
        for attempt in range(3):
            try:
                response = requests.get(url)
                #print(f'Page {page} fetched successfully.')
                break
            except requests.exceptions.RequestException as e:
                #print(f"Error retrieving page {page}: {e}. Retrying...")
                time.sleep(5)
        else:
            #print("Failed to retrieve page after multiple attempts.
             ↪  Stopping.")
            break
        soup = BeautifulSoup(response.text, 'lxml')
        actions = soup.find_all('li', class_='usa-card card--list
↪  pep-card--minimal mobile:grid-col-12')
        if not actions:
            #print('No more actions found, ending scraping.')
            break
        #print(f'Processing actions on pages {page}...')
        for action in actions:
            title_tag = action.find('h2',
↪  class_='usa-card__heading').find('a')
            title = title_tag.get_text(strip=True)
            link = title_tag['href']
            full_link = f'https://oig.hhs.gov{link}'

            date_tag = action.find('span', class_='text-base-dark
↪  padding-right-105')
            date_text = date_tag.get_text(strip=True)
            try:
                action_date = datetime.strptime(date_text, '%B %d, %Y')
            except ValueError:
                #print(f'Skipping action due to date format issue:
                 ↪  {date_text}')
                continue
            if action_date < start_date:
                #print('Reached start date. Stopping scraping.')
                stop_scraping = True
                break
            if action_date <= today:
```

```
                category_tag = action.find('ul', class_='display-inline
↪  add-list-reset').find('li')
                category = category_tag.get_text(strip=True)
                data.append({
                    'Title': title,
                    'Date': date_text,
                    'Category': category,
                    'Link': full_link
                })
        if stop_scraping:
          break
        page += 1
        time.sleep(1)
    #print('Coverting data to DataFrame...')
    df = pd.DataFrame(data)
    output_filename =
↪  "D:/Github/vsc/hwk5/enforcement_actions_{}_{}.csv".format(year, month)
    try:
        df.to_csv(output_filename, index=False)
        #print(f"Data saved to {output_filename}")
    except Exception as e:
        print(f"Error saving file: {e}")
scrape_enforcement_actions(2021, 1)
```

```
# Load the CSV file into a DataFrame
enf_df = pd.read_csv("enforcement_actions_2021_1.csv")
enf_df['Date'] = pd.to_datetime(enf_df['Date'],
↪  errors='coerce').dt.strftime('%Y-%m-%d')
# Number of enforcement actions
num_actions = enf_df.shape[0]
print(f"Total number of enforcement actions: {num_actions}")
# Details of the earliest enforcement action
earliest_action = enf_df.sort_values(by="Date").iloc[0]
print("Earliest enforcement action:")
print(earliest_action)
```

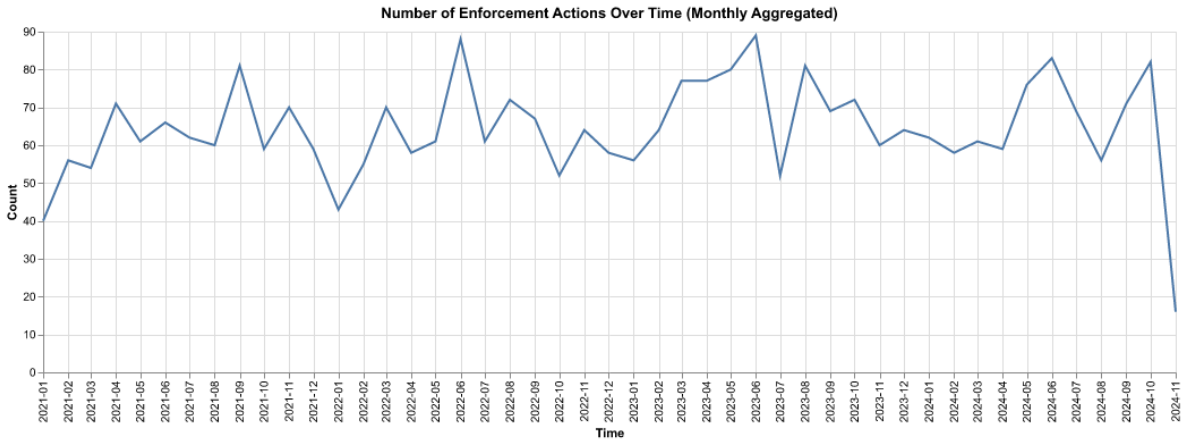
```
Total number of enforcement actions: 3022
Earliest enforcement action:
Title       The United States And Tennessee Resolve Claims...
Date                                                2021-01-04
Category                           Criminal and Civil Actions
```

```
Link        https://oig.hhs.gov/fraud/enforcement/the-unit...
Name: 3021, dtype: object
```

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time (PARTNER 2)

```
# Load the file and dataframe
enf202101_df = pd.read_csv("enforcement_actions_2021_1.csv")
enf202101_df['Date'] = pd.to_datetime(enf202101_df['Date'], errors='coerce')
# Create a new column for the Year-Month
enf202101_df['YearMonth'] =
 ↪  enf202101_df['Date'].dt.to_period('M').astype(str)
monthly_counts =
 ↪  enf202101_df.groupby('YearMonth').size().reset_index(name='Count')
monthly_counts['YearMonth'] = pd.to_datetime(monthly_counts['YearMonth'],
 ↪  format='%Y-%m')
# Plot using Altair
enf202101_chart = alt.Chart(monthly_counts).mark_line().encode(
    x=alt.X('YearMonth:T', title='Time', axis=alt.Axis(
      format='%Y-%m',
      tickCount='month',
      labelAngle=-90,
      labelOverlap='parity')),
    y=alt.Y('Count:Q', title='Count')
).properties(
    title='Number of Enforcement Actions Over Time (Monthly Aggregated)',
    width=1000
)
enf202101_chart
```

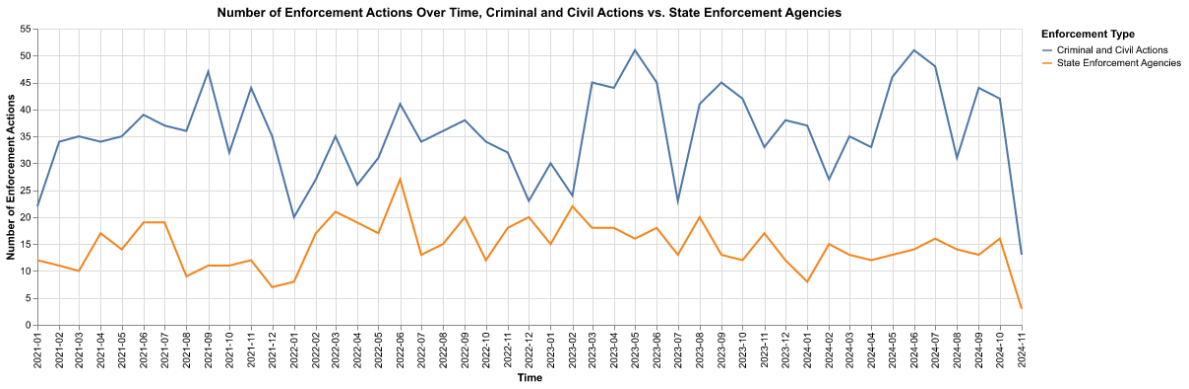


2. Plot the number of enforcement actions categorized: (PARTNER 1)

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

```
# Load the file and dataframe
enf202101_df =
 ↪  pd.read_csv("D:/Github/vsc/hwk5/enforcement_actions_2021_1.csv")
enf202101_df['Date'] = pd.to_datetime(enf202101_df['Date'], errors='coerce')
cca_vs_sea_df = enf202101_df[enf202101_df['Category'].isin(["Criminal and
 ↪  Civil Actions", "State Enforcement Agencies"])]
# Create a new column for the Year-Month
cca_vs_sea_df['YearMonth'] =
 ↪  cca_vs_sea_df['Date'].dt.to_period('M').astype(str)
monthly_counts = cca_vs_sea_df.groupby(['YearMonth',
 ↪  'Category']).size().reset_index(name='Count')
monthly_counts['YearMonth'] = pd.to_datetime(monthly_counts['YearMonth'],
 ↪  format='%Y-%m')
# Plot using Altair
cca_vs_sea_chart = alt.Chart(monthly_counts).mark_line().encode(
    x=alt.X('YearMonth:T', title='Time', axis=alt.Axis(
        format='%Y-%m',
        tickCount='month',
        labelAngle=-90
    )),
    y=alt.Y('Count:Q', title='Number of Enforcement Actions'),
    color=alt.Color('Category:N', title='Enforcement Type')
).properties(
    title='Number of Enforcement Actions Over Time, Criminal and Civil
 ↪  Actions vs. State Enforcement Agencies',
```

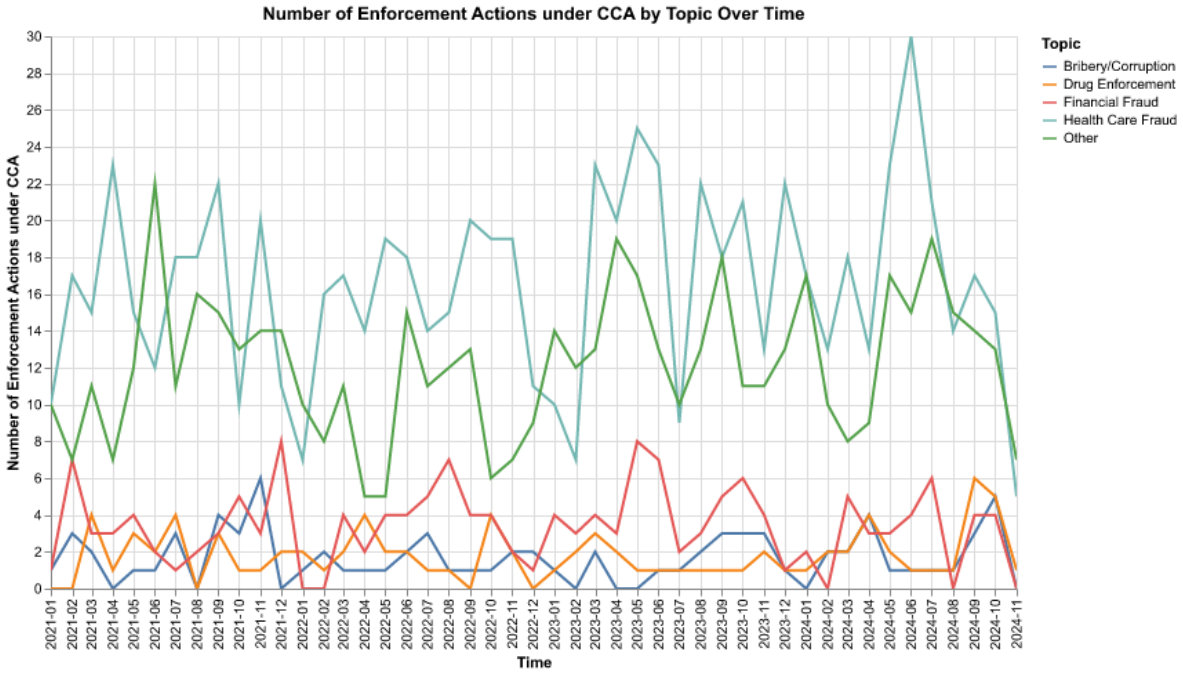
```
    width=1000
)
cca_vs_sea_chart
```



- based on five topics

```
# Load the enforcement actions data
enf202101_path = "enforcement_actions_2021_1.csv"
enf202101_df = pd.read_csv(enf202101_path)
# Define function to categorize topics based on keywords in titles
def categorize_topic(title):
    title = title.lower()
    if re.search(r'\bhealth\b|\bmedicare\b|\bmedicaid\b|\bpatient\b', title):
        return "Health Care Fraud"
    elif re.search(r'\bfinancial\b|\bbank\b|\bfraud\b|\binsurance\b', title):
        return "Financial Fraud"
    elif re.search(r'\bdrug\b|\bopioid\b|\bsubstance\b|\bnarcotic\b', title):
        return "Drug Enforcement"
    elif re.search(r'\bbribery\b|\bcorruption\b|\bkickback\b', title):
        return "Bribery/Corruption"
    else:
        return "Other"
# Apply categorization to create a 'Topic' column
enf202101_df['Topic'] = enf202101_df['Title'].apply(categorize_topic)
# Convert 'Date' column to datetime format and extract year-month for
 ↪  grouping
enf202101_df['Date'] = pd.to_datetime(enf202101_df['Date'], errors='coerce')
enf202101_df['YearMonth'] =
 ↪  enf202101_df['Date'].dt.to_period('M').astype(str)
enf202101_df['YearMonth'] = pd.to_datetime(enf202101_df['YearMonth'],
 ↪  format='%Y-%m')
```

```
# Filter to only "Criminal and Civil Actions" category
cca_df = enf202101_df[enf202101_df['Category'] == "Criminal and Civil
 ↪  Actions"]
# Group by YearMonth and Topic, count the number of enforcement actions per
 ↪  topic per month
df_counts = cca_df.groupby(['YearMonth',
 ↪  'Topic']).size().unstack(fill_value=0)
# Plot
df_counts = df_counts.reset_index()
# Melt the DataFrame to long format for Altair
df_long = df_counts.melt(id_vars='YearMonth', var_name='Topic',
 ↪  value_name='Count')
# Create the line chart
topic_chart = alt.Chart(df_long).mark_line().encode(
    x=alt.X('YearMonth:T', title='Time', axis=alt.Axis(format='%Y-%m',
        tickCount='month',
        labelAngle=-90)),
    y=alt.Y('Count:Q', title='Number of Enforcement Actions under CCA'),
    color='Topic:N'
).properties(
    title='Number of Enforcement Actions under CCA by Topic Over Time',
    width=700,
    height=400
).interactive()
topic_chart
```



Step 4: Create maps of enforcement activity

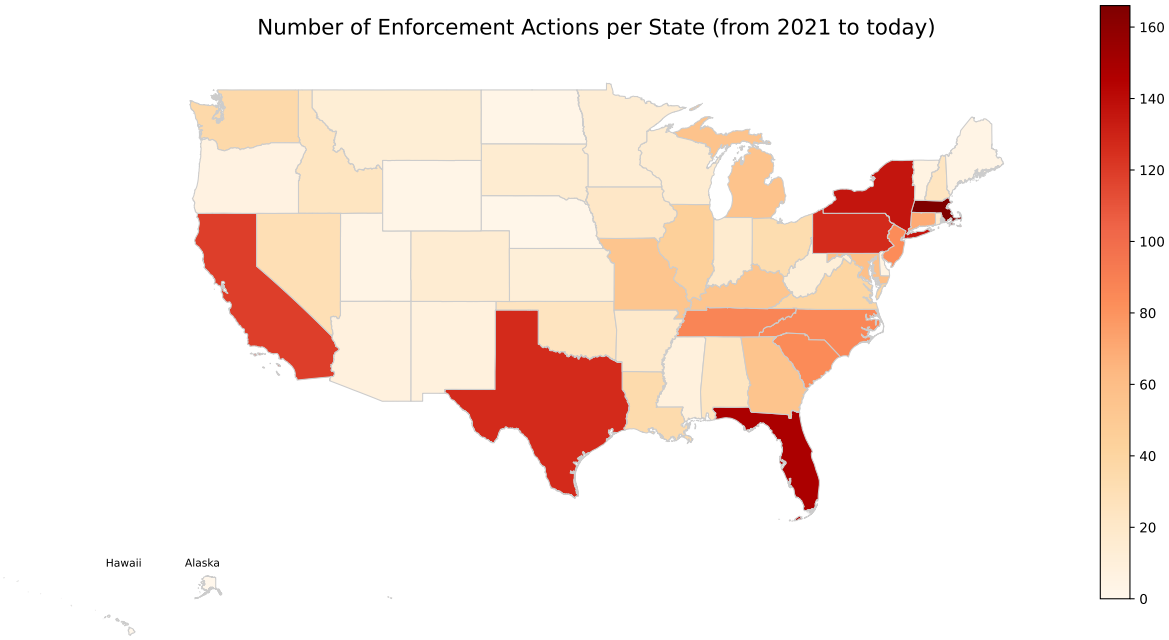
1. Map by State (PARTNER 1)

```
enf202101_df = pd.read_csv("enforcement_actions_2021_1.csv")
agencies = []
# Loop through each link in the DataFrame to extract the agency
for link in enf202101_df['Link']:
    try:
        response = requests.get(link)
        soup = BeautifulSoup(response.text, 'lxml')
        agency_label = soup.find('span', text="Agency:")
        if agency_label:
            agency = agency_label.find_next_sibling(text=True).strip()
        else:
            agency=""
        agencies.append(agency)
        time.sleep(1)
    except Exception as e:
        agencies.append("Error retrieving agency")
        print(f"Error with link {link}: {e}")
```

```
enf202101_df['Agency'] = agencies
enf202101_df.to_csv("enforcement_actions_2021_1.csv", index=False)
```

```
enf202101_df = pd.read_csv("enforcement_actions_2021_1.csv")
# Dictionary to map full state names to their abbreviations
state_abbreviations = {
    'Alabama': 'AL', 'Alaska': 'AK', 'Arizona': 'AZ', 'Arkansas': 'AR',
    'California': 'CA', 'Colorado': 'CO', 'Connecticut': 'CT', 'Delaware':
    ↪  'DE',
    'Florida': 'FL', 'Georgia': 'GA', 'Hawaii': 'HI', 'Idaho': 'ID',
    'Illinois': 'IL', 'Indiana': 'IN', 'Iowa': 'IA', 'Kansas': 'KS',
    'Kentucky': 'KY', 'Louisiana': 'LA', 'Maine': 'ME', 'Maryland': 'MD',
    'Massachusetts': 'MA', 'Michigan': 'MI', 'Minnesota': 'MN',
    ↪  'Mississippi': 'MS',
    'Missouri': 'MO', 'Montana': 'MT', 'Nebraska': 'NE', 'Nevada': 'NV',
    'New Hampshire': 'NH', 'New Jersey': 'NJ', 'New Mexico': 'NM', 'New
    ↪  York': 'NY',
    'North Carolina': 'NC', 'North Dakota': 'ND', 'Ohio': 'OH', 'Oklahoma':
    ↪  'OK',
    'Oregon': 'OR', 'Pennsylvania': 'PA', 'Rhode Island': 'RI', 'South
    ↪  Carolina': 'SC',
    'South Dakota': 'SD', 'Tennessee': 'TN', 'Texas': 'TX', 'Utah': 'UT',
    'Vermont': 'VT', 'Virginia': 'VA', 'Washington': 'WA', 'West Virginia':
    ↪  'WV',
    'Wisconsin': 'WI', 'Wyoming': 'WY'
}
# Compile a regex pattern to match full state names and abbreviations
state_pattern = re.compile(r'\b(' + '|'.join(state_abbreviations.keys()) +
↪  r'|' + '|'.join(state_abbreviations.values()) + r')\b', re.IGNORECASE)
# Function to extract only the state abbreviation
def extract_state_abbreviation(text):
    if pd.isna(text):
        return ""
    match = state_pattern.search(text)
    if match:
        state = match.group(0)
        return state_abbreviations.get(state.title(), state.upper())
    return ""
enf202101_df['Agency'] =
↪  enf202101_df['Agency'].apply(extract_state_abbreviation)
enf202101_df.to_csv("enforcement_actions_2021_1_state.csv", index=False)
```

```
# Load the shapefile for U.S. states
states_shp_path = "cb_2018_us_state_500k/cb_2018_us_state_500k.shp"
gdf_states = gpd.read_file(states_shp_path)
enf202101_df = pd.read_csv('enforcement_actions_2021_1_state.csv')
# Count the number of enforcement actions for each state
state_counts = enf202101_df['Agency'].value_counts().reset_index()
state_counts.columns = ['State', 'Count']
# Merge the state counts with the geodataframe on the state abbreviation
gdf_states = gdf_states.merge(state_counts, how="left", left_on="STUSPS",
 ↪  right_on="State")
# Replace NaN values in the Count column with 0 (for states with no
 ↪  enforcement actions)
gdf_states['Count'] = gdf_states['Count'].fillna(0)
gdf_contiguous = gdf_states[~gdf_states['STUSPS'].isin(['AK', 'HI', 'PR',
 ↪  'VI', 'GU', 'MP', 'AS'])]
gdf_hi = gdf_states[gdf_states['STUSPS'] == 'HI']
gdf_ak = gdf_states[gdf_states['STUSPS'] == 'AK']
# Plot the choropleth map
fig, ax = plt.subplots(1, 1, figsize=(15, 8))
gdf_contiguous.plot(column='Count', cmap='OrRd', linewidth=0.8, ax=ax,
 ↪  edgecolor='0.8', legend=True)
# Add an inset for Hawaii
inset_ax_hi = fig.add_axes([0.02, 0.05, 0.1, 0.1])
gdf_hi.plot(column='Count', cmap='OrRd', linewidth=0.8, ax=inset_ax_hi,
 ↪  edgecolor='0.8')
inset_ax_hi.set_title("Hawaii", fontsize=8, loc='right')
inset_ax_hi.axis('off')
# Add an inset for Alaska
inset_ax_ak = fig.add_axes([0.15, 0.05, 0.15, 0.15])
gdf_ak.plot(column='Count', cmap='OrRd', linewidth=0.8, ax=inset_ax_ak,
 ↪  edgecolor='0.8')
inset_ax_ak.set_title("Alaska", fontsize=8, loc='left')
inset_ax_ak.axis('off')
# Add title and remove axes for clarity
ax.set_title("Number of Enforcement Actions per State (from 2021 to today)",
 ↪  fontsize=16, pad=20)
ax.axis('off')
plt.show()
```

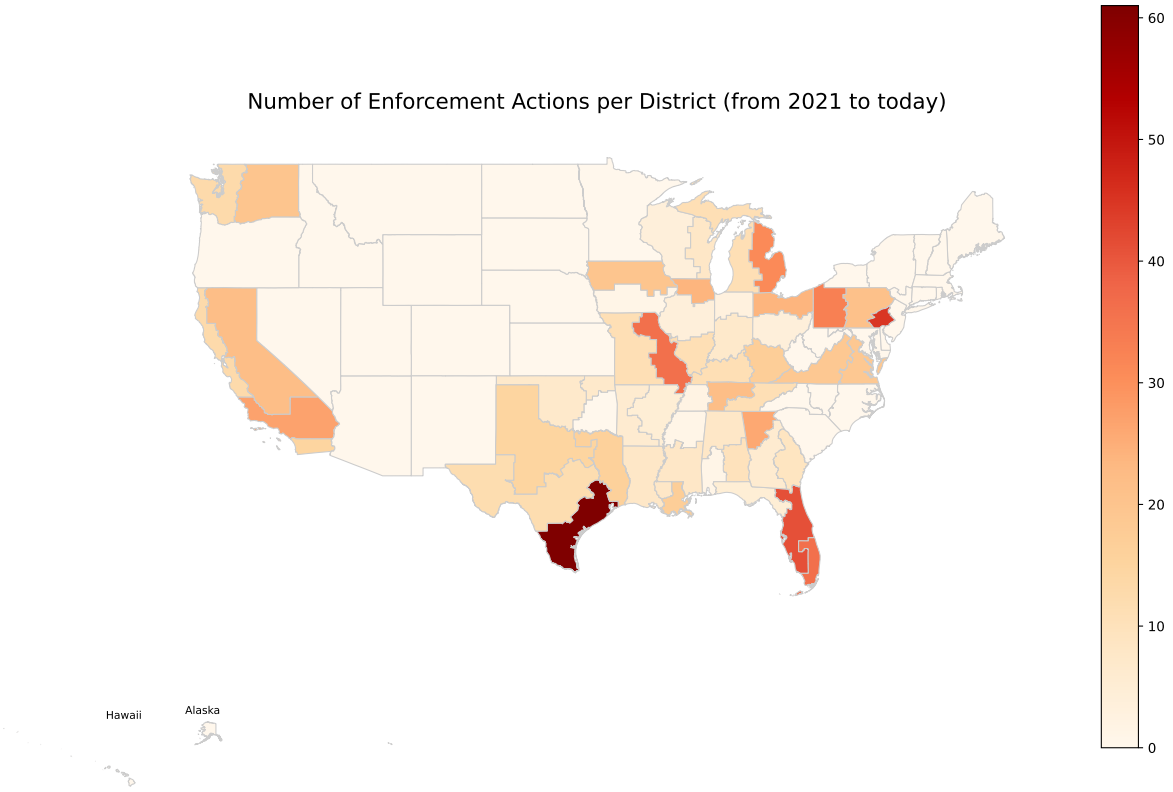


2. Map by District (PARTNER 2)

```
# Load the CSV file
enf202101_df = pd.read_csv('enforcement_actions_2021_1.csv')
# Define a function to extract district names from the Agency column
def extract_district(text):
    if pd.isna(text):
        return ""
    match = re.search(r'\b\w+ District of \w+\b', text)
    return match.group(0) if match else ""
enf202101_df['Agency'] = enf202101_df['Agency'].apply(extract_district)
enf202101_df.to_csv("enforcement_actions_2021_1_districts.csv", index=False)
```

```
# Load the shapefile for U.S. districts
ds_shp_path = "US Attorney Districts Shapefile
 ↪  simplified_20241108/geo_export_babcb301-3e34-43ac-98c5-eee029e991a6.shp"
gdf_districts = gpd.read_file(ds_shp_path)
enf202101_df = pd.read_csv("enforcement_actions_2021_1_districts.csv")
# Count the number of enforcement actions for each district
district_counts = enf202101_df['Agency'].value_counts().reset_index()
district_counts.columns = ['District', 'Count']
```

```
# Merge the district counts with the geodataframe on the district name
gdf_districts = gdf_districts.merge(district_counts, how="left",
 ↪  left_on="judicial_d", right_on="District")
# Replace NaN values in the Count column with 0 (for districts with no
 ↪  enforcement actions)
gdf_districts['Count'] = gdf_districts['Count'].fillna(0)
gdf_districts = gdf_districts[~gdf_districts['state'].isin(['Puerto Rico',
 ↪  'US Virgin Islands', 'Guam', 'Northern Marianas Islands', 'American
 ↪  Samoa'])]
gdf_hawaii = gdf_districts[gdf_districts['state'] == 'Hawaii']
gdf_alaska = gdf_districts[gdf_districts['state'] == 'Alaska']
gdf_districts = gdf_districts[~gdf_districts['state'].isin(['Hawaii',
 ↪  'Alaska'])]
# Plot the choropleth map
fig, ax = plt.subplots(1, 1, figsize=(15, 10))
gdf_districts.plot(column='Count', cmap='OrRd', linewidth=0.8, ax=ax,
 ↪  edgecolor='0.8', legend=True)
# Add an inset for Hawaii in the bottom-left corner with a frame
inset_ax_hawaii = fig.add_axes([0.02, 0.05, 0.1, 0.1])
gdf_hawaii.plot(column='Count', cmap='OrRd', linewidth=0.8,
 ↪  ax=inset_ax_hawaii, edgecolor='0.8')
inset_ax_hawaii.set_title("Hawaii", fontsize=8, loc='right', pad=5)
inset_ax_hawaii.axis('off')
# Add an inset for Alaska in the bottom-left corner with a frame
inset_ax_alaska = fig.add_axes([0.15, 0.05, 0.15, 0.15])
gdf_alaska.plot(column='Count', cmap='OrRd', linewidth=0.8,
 ↪  ax=inset_ax_alaska, edgecolor='0.8')
inset_ax_alaska.set_title("Alaska", fontsize=8, loc='left', pad=5)
inset_ax_alaska.axis('off')
# Customize the appearance
ax.set_title("Number of Enforcement Actions per District (from 2021 to
 ↪  today)", fontsize=16, pad=20)
ax.axis('off')
plt.show()
```



Extra Credit

1. Merge zip code shapefile with population

```
# Load the dataset
pop2020_df =
 ↪  pd.read_csv("DECENNIALDHC2020.P1_2024-11-08T181030/DECENNIALDHC2020.P1-Data.csv")
# Drop the second row
pop2020_df = pop2020_df.drop(index=0)
pop2020_df = pop2020_df.drop(columns=['Unnamed: 3'])
# Clean the 'NAME' column to retain only the 5-digit ZIP code
pop2020_df['NAME'] = pop2020_df['NAME'].apply(lambda x:
 ↪  re.search(r'\b\d{5}\b', str(x)).group(0) if re.search(r'\b\d{5}\b',
 ↪  str(x)) else "")
# Ensure the 'NAME' column is of object data type
pop2020_df['NAME'] = pop2020_df['NAME'].astype(str)
```

```
shapefile_path = "cb_2020_us_zcta520_500k/cb_2020_us_zcta520_500k.shp"
gdf_zipcodes = gpd.read_file(shapefile_path)
# Load the population data
pop2020_path =
 ↪  "DECENNIALDHC2020.P1_2024-11-08T181030/DECENNIALDHC2020.P1-Data.csv"
pop2020_df = pd.read_csv(pop2020_path)
# Drop the second row
pop2020_df = pop2020_df.drop(index=0)
pop2020_df = pop2020_df.drop(columns=['Unnamed: 3'])
# Clean the 'NAME' column to retain only the 5-digit ZIP code
pop2020_df['NAME'] = pop2020_df['NAME'].apply(lambda x:
 ↪  re.search(r'\b\d{5}\b', str(x)).group(0) if re.search(r'\b\d{5}\b',
 ↪  str(x)) else "")
# Ensure the 'NAME' column is of object data type
pop2020_df['NAME'] = pop2020_df['NAME'].astype(str)
pop2020_zp_gdf = gdf_zipcodes.merge(pop2020_df, how="left",
 ↪  left_on="ZCTA5CE20", right_on="NAME")
# Optionally, save the merged data to a new file for further use
pop2020_zp_gdf.to_file("merged_zipcode_pop_data_2020.shp")
```

2. Conduct spatial join

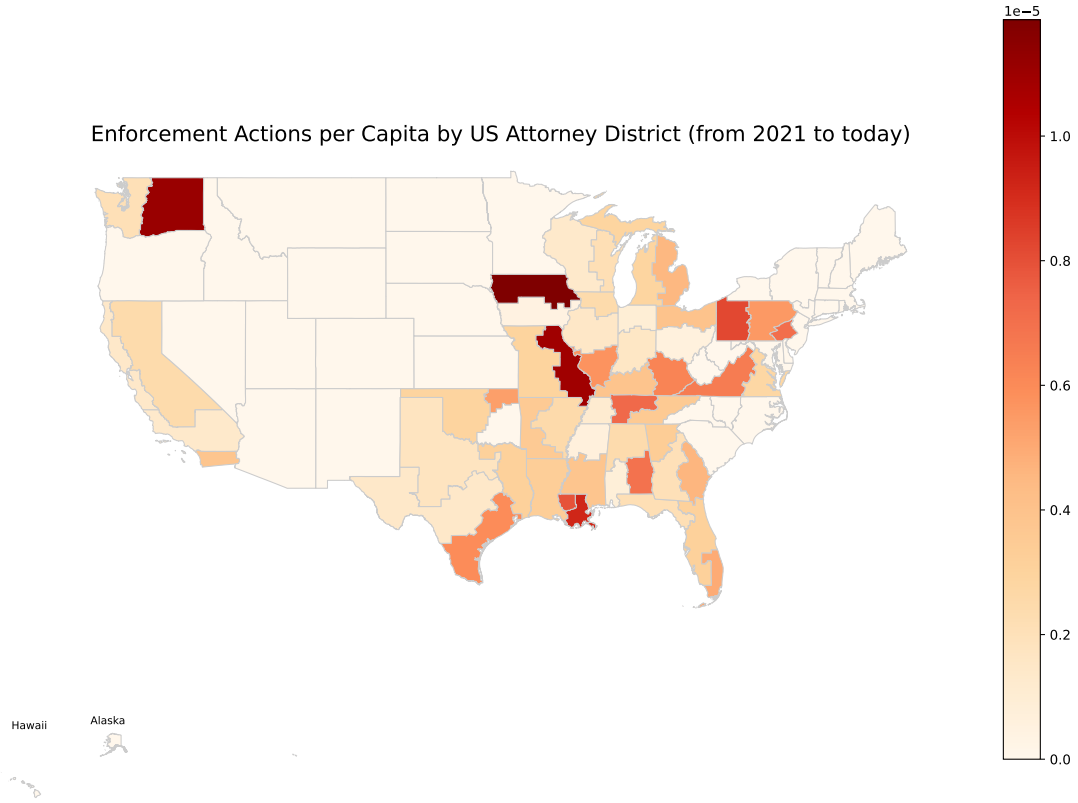
```
# Load the ZIP code shapefile with population data (already merged)
zc_pop_path = "merged_zipcode_pop_data_2020/merged_zipcode_pop_data_2020.shp"
zc_pop_gdf = gpd.read_file(zc_pop_path)
# Load the district shapefile
ds_path = "US Attorney Districts Shapefile
 ↪  simplified_20241108/geo_export_babcb301-3e34-43ac-98c5-eee029e991a6.shp"
ds_gdf = gpd.read_file(ds_path)
zc_pop_gdf['P1_001N'] = pd.to_numeric(zc_pop_gdf['P1_001N'], errors='coerce')
# Conduct a spatial join
zc_pop_ds = gpd.sjoin(zc_pop_gdf, ds_gdf, how="inner",
 ↪  predicate="intersects")
# Aggregate the population by district
pop_by_ds = zc_pop_ds.groupby("judicial_d")["P1_001N"].sum().reset_index()
pop_by_ds.to_csv("population_by_district_2020.csv", index=False)
```

3. Map the action ratio in each district

```
# Load the enforcement actions data by district
enforcement_data_path = "enforcement_actions_2021_1_districts.csv"
df_enforcement = pd.read_csv(enforcement_data_path)
# Load the population data by district
population_data_path = "population_by_district_2020.csv"
df_population = pd.read_csv(population_data_path)
# Aggregate the number of enforcement actions per district
enforcement_counts = df_enforcement['Agency'].value_counts().reset_index()
enforcement_counts.columns = ['District', 'Enforcement_Actions']
# Merge the enforcement counts with the population data
enf_pop_ds = pd.merge(enforcement_counts, df_population, how='left',
 ↪  left_on='District', right_on='judicial_d')
# Calculate the enforcement actions per capita (ratio)
enf_pop_ds['Enforcement_Per_Capita'] = enf_pop_ds['Enforcement_Actions'] /
 ↪  enf_pop_ds['P1_001N']
enf_pop_ds.to_csv("enforcement_ratio_per_district.csv", index=False)
```

```
# Load the district shapefile
district_shapefile_path = "US Attorney Districts Shapefile
 ↪  simplified_20241108/geo_export_babcb301-3e34-43ac-98c5-eee029e991a6.shp"
gdf_districts = gpd.read_file(district_shapefile_path)
# Load the enforcement actions per capita data by district
enforcement_ratio_path = "enforcement_ratio_per_district.csv"
df_ratio = pd.read_csv(enforcement_ratio_path)
# Merge the ratio data with the district shapefile on the district name
gdf_districts = gdf_districts.merge(df_ratio, how="left",
 ↪  left_on="judicial_d", right_on="District")
# Fill NaN values for districts with no enforcement actions with zero
gdf_districts['Enforcement_Actions'] =
 ↪  gdf_districts['Enforcement_Actions'].fillna(0)
gdf_districts['Enforcement_Per_Capita'] =
 ↪  gdf_districts['Enforcement_Per_Capita'].fillna(0)
gdf_districts = gdf_districts[~gdf_districts['state'].isin(['Puerto Rico',
 ↪  'US Virgin Islands', 'Guam', 'Northern Marianas Islands', 'American
 ↪  Samoa'])]
gdf_hawaii = gdf_districts[gdf_districts['state'] == 'Hawaii']
gdf_alaska = gdf_districts[gdf_districts['state'] == 'Alaska']
gdf_districts = gdf_districts[~gdf_districts['state'].isin(['Hawaii',
 ↪  'Alaska'])]
```

```
# Plot the choropleth map based on the enforcement actions per capita
fig, ax = plt.subplots(1, 1, figsize=(15, 10))
gdf_districts.plot(column='Enforcement_Per_Capita', cmap='OrRd',
 ↪  linewidth=0.8, ax=ax, edgecolor='0.8', legend=True)
# Add an inset for Hawaii in the bottom-left corner with a frame
inset_ax_hawaii = fig.add_axes([0.02, 0.05, 0.1, 0.1])
gdf_hawaii.plot(column='Enforcement_Per_Capita', cmap='OrRd', linewidth=0.8,
 ↪  ax=inset_ax_hawaii, edgecolor='0.8')
inset_ax_hawaii.set_title("Hawaii", fontsize=8, loc='right')
inset_ax_hawaii.axis('off')
# Add an inset for Alaska in the bottom-left corner with a frame
inset_ax_alaska = fig.add_axes([0.15, 0.05, 0.15, 0.15])
gdf_alaska.plot(column='Enforcement_Per_Capita', cmap='OrRd', linewidth=0.8,
 ↪  ax=inset_ax_alaska, edgecolor='0.8')
inset_ax_alaska.set_title("Alaska", fontsize=8, loc='left')
inset_ax_alaska.axis('off')
# Customize the appearance
ax.set_title("Enforcement Actions per Capita by US Attorney District (from
 ↪  2021 to today)", fontsize=16)
ax.axis('off')
plt.show()
```


Enforcement Actions per Capita by US Attorney District (from 2021 to today)
1e−5
1.0
0.8
0.6
0.4
0.2
0.0
Hawaii
Alaska