Problem Set 5 - DAP II

Ralph Valery Valiere

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. But Ralph submitted alone
2. Play paper, scissors, rock to determine who goes first. Call that person *Partner 1*. (Not necessary)
    - Partner 1 (name and cnet ID): Ralph Valery Valiere (ralphvaleryv)
    - 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: RVV **___** **___**
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: 1 **___** Late coins left after submission: 2 **___**
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 os
import datetime
import time
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import altair as alt
import geopandas as gpd
import shapely
import bs4
import requests
import warnings
import sys # We will use this to exit our if statement
import numpy as np
from vega_datasets import data
from bs4 import BeautifulSoup
from shapely import Polygon, Point, MultiPolygon
from numpy import mean, nan # PS: Some version of numpy only consider NaN. So
 ↪  graders should consider this when this chunk of code is ran.
alt.renderers.enable("png")
warnings.filterwarnings('ignore')
```

Step 1: Develop initial scraper and crawler

1. Scraping (PARTNER 1)

```
hhsoig_enforce_path = r'https://oig.hhs.gov/fraud/enforcement/'

hhsoig_enforce_retrived = requests.get(hhsoig_enforce_path)

hhsoig_enforce_content = BeautifulSoup(hhsoig_enforce_retrived.content,
 ↪  'lxml')
```

We have located the lists of element we want to extract, which are nested inside a “ul” tag. We have identified the specific attribute for this tag and the elements inside, which will make the scrapping easier.

```
# Retrieving the unordered list ('ul') to find the list of enforcement in the
 ↪  first page on the website, avoinding retrieving all the elements ('li')
 ↪  from the website.
list_enforce_firstpage = hhsoig_enforce_content.find_all('ul', class_ =
 ↪  "usa-card-group padding-y-0")
```

After exploring the webpage, we have realized: 1- The 'links' are located in 'a' tags (Fortunately, we will be looking only in the unordered list we scrapped) 2- The "titles" are located inside those same 'a' tags 3- The 'dates' are located inside 'span' tags 4- The 'categories' are located inside 'li' tags with attribute: 'class_ = "display-inline-block usa-tag text-no-lowercase text-base-darkest bg-base-lightest margin-right-1" '. We will retrive each of them separately and then append them into a tidy dataframe

Let's retrieve the list of links first.

```
list_links_ongoing = list_enforce_firstpage[0].find_all('a') # Extracting the
 ↪  'a' tags
list_links_final = [] # Initializing the list of links we will append
for a_tags in list_links_ongoing:
  a_link_partial = a_tags.get('href') # Only partial links
  a_link_complete = 'https://oig.hhs.gov' + a_link_partial
  list_links_final.append(a_link_complete)
```

As mentioned before, we can also extract the title of each of the list elements by using the same 'a' tags.

```
list_title_final = [] # Initializing the list of titles we will append
for a_tags in list_links_ongoing:
  title_text = a_tags.text
  list_title_final.append(title_text)
```

The dates are located in 'span' tags. We will extract the text first and then convert those texts to date type data.

```
# Extracting the 'span' tags from our 'ul' scrapped data
list_dates_ongoing = list_enforce_firstpage[0].find_all('span')
list_dates_final = [] # Initializing the list of dates we will append
for span_tags in list_dates_ongoing:
  date_text = span_tags.text
  list_dates_final.append(date_text)
```

```
# Now let's clean the dates text to later transform it into a date data type
# We will replace all the space by "/" and remove all the "," to have a
 ↪  uniform format.
for index in range(len(list_dates_final)):
  list_dates_final[index] = list_dates_final[index].replace(' ', '/')
  list_dates_final[index] = list_dates_final[index].replace(',', '')

# We are also converting the month name into month rank number
for index in range(len(list_dates_final)):
  if list_dates_final[index].startswith('November'):
    list_dates_final[index] = list_dates_final[index].replace('November',
 ↪  '11')
  elif list_dates_final[index].startswith('October'):
    list_dates_final[index] = list_dates_final[index].replace('October',
 ↪  '10')
  else:
    pass

# Finally, we can convert those dates into date type
for index in range(len(list_dates_final)):
  list_dates_final[index] =
 ↪  datetime.datetime.strptime(list_dates_final[index], '%m/%d/%Y')
```

The next step is to extract the categories, which are in "li" tags with attribute class_ = "display-inline-block usa-tag text-no-lowercase text-base-darkest bg-base-lightest margin-right-1".

```
list_category_ongoing = list_enforce_firstpage[0].find_all('li', class_ =
 ↪  "display-inline-block usa-tag text-no-lowercase text-base-darkest
 ↪  bg-base-lightest margin-right-1")
list_category_final = [] # Initializing the list of categories we will append
for li_tags in list_category_ongoing:
  category_text = li_tags.text
  list_category_final.append(category_text)
```

Finally, now let's store those data in a tidy dataframe.

```
enforcement_action_data = pd.DataFrame({
  'title_enforcement' : list_title_final,
  'date_enforcement' : list_dates_final,
```

```
  'category_enforcement' : list_category_final,
  'link_enforcement' : list_links_final
})
```

We are printing the dataframe in two segments to have the table size fit the page width when knitting the quarto file.

```
# Segment 1
enforcement_action_data[['title_enforcement', 'date_enforcement']].head(5)
```

	title_enforcement	date_enforcement
0	Pharmacist and Brother Convicted of $15M Medic...	2024-11-08
1	Boise Nurse Practitioner Sentenced To 48 Month...	2024-11-07
2	Former Traveling Nurse Pleads Guilty To Tamper...	2024-11-07
3	Former Arlington Resident Sentenced To Prison ...	2024-11-07
4	Paroled Felon Sentenced To Six Years For Fraud...	2024-11-07

```
# Segment 2
enforcement_action_data[['category_enforcement', 'link_enforcement']].head(5)
```

	category_enforcement	link_enforcement
0	Criminal and Civil Actions	https://oig.hhs.gov/fraud/enforcement/pharmaci...
1	Criminal and Civil Actions	https://oig.hhs.gov/fraud/enforcement/boise-nu...
2	Criminal and Civil Actions	https://oig.hhs.gov/fraud/enforcement/former-t...
3	Criminal and Civil Actions	https://oig.hhs.gov/fraud/enforcement/former-a...
4	Criminal and Civil Actions	https://oig.hhs.gov/fraud/enforcement/paroled-...

2. Crawling (PARTNER 1)

For this step, we will run a “for loop” to collect the name of the agency involved, from each of the links in the dataframe. We made sure to check each of the link to verify if this information is available (for just one page, it’s ok but we won’t be doing thi when scrapping multiple pages)

Moreover, we also found that the information about the agency that is involved is located in ‘li’ tag nested in a ‘ul’ tag with attribute class=“usa-list usa-list–unstyled margin-y-2” and it is always the second ‘li’ tag inside the ‘ul’ tag. There is a ‘span’ tag inside those ‘li’ tags but

we will ignore them as they content identifier for the text ('Agency' in our case). This will make the scrapping easier.

```
list_agency_final = [] # Initializing the list of agency we will append

# Building the "for loop" to go over each link in "enforcement_action_data"
 ↪  and retrieve the agency name, and store it in "list_agency_final".
for link in enforcement_action_data['link_enforcement']:
  enforcement_retrived = requests.get(link)
  enforcement_content = BeautifulSoup(enforcement_retrived.content, 'lxml')
  box_agency = enforcement_content.find_all('ul', class_ = "usa-list
 ↪  usa-list--unstyled margin-y-2")
  agency_info_ongoing = box_agency[0].find_all('li')[1].text
  agency_info_final = agency_info_ongoing.replace('Agency:', '')
  list_agency_final.append(agency_info_final)
```

Let's append the list of agencies to the dataframe now.

```
enforcement_action_data['agency_enforcement'] = list_agency_final
```

Step 2: Making the scraper dynamic

To make the scraper dynamic, we will write a function that takes as input a month and a year, and then pulls and formats the enforcement actions starting from that month+year to today.

In the link for each page, there is a specific string that mention the page number. For example, the link for the first page is "https://oig.hhs.gov/fraud/enforcement/?page=1", while the link for the last page is "https://oig.hhs.gov/fraud/enforcement/?page=480". At the moment we are writing this code, there are 482 pages on the website. However, since the website is dynamic, the number of pages might increased. We will make sure we find this number first before iterating over the number of pages. By changing the last digit on the link, we can then directly access each ot those pages and avoid to retrieve those links manually. This is going to be fun!!!!!!

1. Turning the scraper into a function

- a. Pseudo-Code

First, let's write the pseudo code for this function:

1. Define the two arguments that will be provided (month and year). Both arguments will only be "int" type (less risk of making typo than if we were requesting strings for the month)
2. Use "if" statement to check if month is "int" type, if it is in the correct format or in the correct range (Error message will be printed if not and code will break. Anyone calling the function will need to start over! Oh my God! This is so fuuuunn!)
3. If month-argument type test passed, use another "if" statement to check if year is "int" type, if it is in the correct format and if year >= 2013. Print Error message if not and code will break. Anyone calling the function will need to start over!
4. Initialize the five lists that will contain data about 'title_enforcement', 'date_enforcement', 'category_enforcement', 'link_enforcement', and 'agency_enforcement'.
5. We will retrieve the number of pages first to now exactly on how many pages we will scrape for our crawling (although we might not use all of them). It's good thing that we found that the last page is always displayed on any page of the website and the content is stored in a "a" tag, with one of its attributes being 'class="pagination___link" '. This will prevent our code from missing information whenever its ran in a different day as the website updates daily.
6. Start the "for loop" by requesting and extracting the data from each of the pages, using request and Beautiful Soup. We will extract the dates first, to allow the filtering to have the most recent dates. Filtering by date will allow us to break from the loop and not continue over all the pages if we find a date that is older than the year and month provided. Moreover, we already know from the website the elements are sorted by date. This means that we won't miss any elements. We created a special trigger to alert the function when to break out of the loop. The trigger is set to False at the beginning of the loop. 6.1 Inside the loop, Extract the dates for each enforcement. For each page, we check if all the dates are at least as recent as the year and month provided. 6.1.1 If at least one of the dates is not as recent as the year and month provided (using an if statement), extract only those that meet this criteria, find the index of those that meet the criteria in the list and extract the titles, links, categories, agencies only for those same index. Then, the special trigger is set to "True" to allow the loop to break at the end of this iteration. 6.1.2 If all dates are as recent as the year and month provided, extract those dates. 6.1.2.1 Inside the 'for loop', Extract the 'titles' for each enforcement and append them to a list. 6.1.2.2 Inside the loop, Extract 'links' associated with each enforcement and append them to a list. 6.1.2.3 Inside the loop, Extract the 'categories' for each enforcement and append them to a list. For categories, we will do differently than Section 1 as for some enforcement there are more than one category displayed. We will extract those categories, using the links for the titles, combined the cases of more than one categories to form a joint category (we can deal later with the joint categories) 6.1.2.4 Inside the loop, Extract the 'agencies' for each enforcement, using the links for the titles, and append them to a list. 6.2 Check the value of the trigger. If it's still False, continue the iteration. If it's set to True, break out of the loop.
7. Append all the generated lists to a tidy dataframe, which would be already filtered.
8. Return the filtered dataframe as the final result of the function.

- b. Then, since we have our pseudo-code, we can build the function from this, to create the dynamic Scraper, and start collecting the enforcement actions since January 2023.

```
# Base path for the enforcement action webpages
hhsoig_enforcement_page = r'https://oig.hhs.gov/fraud/enforcement/?page='

# This function is going to be long. So sorry for the graders!

def crawl_enforcement_data(year, month):
  # Verifying validity for month
  if type(month) != int:
    print("TypeError: Argument 'month' only accepts int type.\nThis is
     ↪  Ralph's predefined error message. Another message will also be
     ↪  displayed after exiting the system. Please ignore message
     ↪  'SystemExit'.")
    sys.exit() # This will exit the code and not run any other lines. I found
 ↪  this method on Stackoverflow.
  elif month < 1 or month > 12:
    print("RangeError: Argument 'month' only accepts values from 0 to
     ↪  12.\nThis is Ralph predefined error message. Another message will
     ↪  also be displayed after exiting the system. Please ignore message
     ↪  'SystemExit'.")
    sys.exit() # This will exit the code and not run any other lines.
  else:
    pass

  # Verifying validity for year
  if type(year) != int:
    print("TypeError: Argument 'year' only accepts int type.\nThis is Ralph
     ↪  predefined error message. Another message will also be displayed
     ↪  after exiting the system. Please ignore message 'SystemExit'")
    sys.exit() # This will exit the code and not run any other lines.
  elif len(str(year)) != 4:
    print("FormatError: Please enter the 'year' in the correct format (e.g.
     ↪  1804)\nThis is Ralph predefined error message. Another message will
     ↪  also be displayed after exiting the system. Please ignore message
     ↪  'SystemExit'.")
    sys.exit() # This will exit the code and not run any other lines.
  elif year < 2013 or  year > int(datetime.datetime.now().year):
    print(f'RangeError: Argument "year" only accepts values from 2013 to
     ↪  {int(datetime.datetime.now().year)}.\nThis is Ralph predefined error
     ↪  message. Another message will also be displayed after exiting the
     ↪  system. Please ignore message "SystemExit".')
```

```
  sys.exit()
else:
  pass

# Iniatizaling the lists that will be appended to the tidy dataframe
titles_final = []
dates_final = []
categories_final= []
links_final = []
agencies_final = []
list_months = ['January', 'February', 'March', 'April', 'May', 'June',
↪  'July', 'August', 'September', 'October', 'November', 'December'] # This
↪  will be handy for conversion into month rank
rank_month = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11',
↪  '12']

# Finding the number of pages we will crawl through
time.sleep(1) # Adding 2 seconds wait to prevent potential server-side
↪  block.
page_numbers = hhsoig_enforce_content.find_all('a', class_ =
↪  "pagination__link")
last_page_contents = page_numbers[-1].text # Last page text always last
↪  index
last_page = '' # Initializing the object that will store the last page
↪  string.
for string in last_page_contents: #  Cleaning to retrieve only the digits
  if string.isdigit():
    last_page = last_page + string
last_page = int(last_page) # Converting the last page into an int type

# Creating the foor loop to extract the data using the crawl
for page_number in range(1, last_page + 1):
  page_link = hhsoig_enforcement_page + str(page_number)
  time.sleep(1) # Adding 2 seconds wait to prevent potential server-side
↪  block.
  page_path = requests.get(page_link)
  page_contents = BeautifulSoup(page_path.content, 'lxml')
  unordered_box = page_contents.find_all('ul', class_ = "usa-card-group
↪  padding-y-0") # All the info we need is contained in this unordered list
  ############

  # Extracting the dates
```

```
    found_date_older = False # To alert us when to break the loop if a date
↪   is older than the year and month provided
    all_dates = unordered_box[0].find_all('span')
    list_dates_temp = [] # Temporary container for the dates. Not our final
↪   list container which is already created outside of the loop
    for span_tags in all_dates: # Extracting all dates for this page
      date_text = span_tags.text
      list_dates_temp.append(date_text)
    for index in range(len(list_dates_temp)): # Cleaning the dates
      list_dates_temp[index] = list_dates_temp[index].replace(' ', '/')
      list_dates_temp[index] = list_dates_temp[index].replace(',', '')
    for index in range(len(list_dates_temp)): # Convert month name into month
     ↪  rank
      month_name = list_dates_temp[index].split('/')[0] # Find month name
      index_month = list_months.index(month_name) # The list of months was
↪   helpful
      list_dates_temp[index] = list_dates_temp[index].replace(month_name,
↪   rank_month[index_month])
    for index in range(len(list_dates_temp)): # Converting into date type
      list_dates_temp[index] =
↪   datetime.datetime.strptime(list_dates_temp[index], '%m/%d/%Y')
      ## Generating a date object as reference to better filter
    first_day_of_month = str(month) + '/1/' + str(year)
    first_day_of_month = datetime.datetime.strptime(first_day_of_month,
↪   '%m/%d/%Y')
      ## Checking for older date and retrieving dates
    if any(date < first_day_of_month for date in list_dates_temp) and (not
     ↪  all(date < first_day_of_month for date in list_dates_temp)):
      found_date_older = True # Setting "ON" the alert for older dates
      list_index_recent = [] # To store the index for dates NOT older
      for index in range(len(list_dates_temp)):
        if list_dates_temp[index] >= first_day_of_month:
          list_index_recent.append(index)
        else:
          pass
      # Now we can append to the final list of dates
      for index in list_index_recent:
        dates_final.append(list_dates_temp[index])
      # Can also find the other variables, which will depend on the list of
       ↪  index for recent dates. The values for each index will match the
       ↪  list of the other variables.
      # Extracting the titles for each enforcement that are recent
```

```
    links_and_titles = unordered_box[0].find_all('a')
    list_titles_temp = []
    for a_tags in links_and_titles:
      title_text = a_tags.text
      list_titles_temp.append(title_text)
    for index in list_index_recent:
      titles_final.append(list_titles_temp[index])
    # Extracting the links for recent enforcement
    list_links_temp = []
    list_links_filtered = [] # We will use this to find the agencies and
↪ the categories which can be retrieved from each link
    for a_tags in links_and_titles:
      a_link_partial = a_tags.get('href') # Only the partial links
      a_link_complete = 'https://oig.hhs.gov' + a_link_partial
      list_links_temp.append(a_link_complete)
    for index in list_index_recent:
      links_final.append(list_links_temp[index])
      list_links_filtered.append(list_links_temp[index])
    # Extracting the categories of recent enforcement
    # Using a different method than Section 1
    for link in list_links_filtered:
      time.sleep(1) # Adding 2 seconds wait to prevent potential
↪ server-side block.
      enforcement_retrieved = requests.get(link)
      enforcement_content = BeautifulSoup(enforcement_retrieved.content,
↪ 'lxml')
      category_box = enforcement_content.find_all('li', class_ =
↪ "display-inline") # This class is unique to the 'li' tag for enforcement
↪ type
      category_box_items = [li.text for li in category_box]
      category_text = ' -&- '.join(category_box_items) # Joining categories
      category_text = category_text.replace('\n', '') # Cleaning
      category_text = ' '.join(category_text.split()) # To remove multiple
↪ blank space
      category_text = category_text.replace(', -&- ', ' -&- ') # Cleaning
      categories_final.append(category_text)
    # Extracting the agencies for recent enforcement
    for link in list_links_filtered:
      time.sleep(2) # Adding 2 seconds wait to prevent potential
↪ server-side block.
      enforcement_retrieved = requests.get(link)
      enforcement_content = BeautifulSoup(enforcement_retrieved.content,
↪ 'lxml')
```

```
        box_agency = enforcement_content.find_all('ul', class_ = "usa-list
↪  usa-list--unstyled margin-y-2")
        agency_info_prelim = box_agency[0].find_all('li')[1].text
        agency_info_final = agency_info_prelim.replace('Agency:', '')
        agencies_final.append(agency_info_final)
    elif any(date < first_day_of_month for date in list_dates_temp) and
     ↪  all(date < first_day_of_month for date in list_dates_temp):
      found_date_older = True # Setting on the alert for older dates
      pass
    if all(date >= first_day_of_month for date in list_dates_temp):
      found_date_older = False # Setting "OFF" the alert for older dates,
↪  just in case it was set "ON"
      # Extracting all the dates
      for date in list_dates_temp:
        dates_final.append(date)
      # Extracting the titles for each enforcement
      links_and_titles = unordered_box[0].find_all('a')
      for a_tags in links_and_titles:
        title_text = a_tags.text
        titles_final.append(title_text)
      # Extracting the links
      list_links_temp = [] # This will become handy when retrieving the
↪  agencies and the categories
      for a_tags in links_and_titles:
        a_link_partial = a_tags.get('href') # Only the partial links
        a_link_complete = 'https://oig.hhs.gov' + a_link_partial
        list_links_temp.append(a_link_complete)
        links_final.append(a_link_complete)
      # Extracting the categories
      for link in list_links_temp:
        time.sleep(1) # Adding 2 seconds wait to prevent potential
↪  server-side block.
        enforcement_retrieved = requests.get(link)
        enforcement_content = BeautifulSoup(enforcement_retrieved.content,
↪  'lxml')
        category_box = enforcement_content.find_all('li', class_ =
↪  "display-inline") # This class is unique inside the 'li' tag for
↪  enforcement type
        category_box_items = [li.text for li in category_box]
        category_text = ' -&- '.join(category_box_items) # Joining categories
        category_text = category_text.replace('\n', '') # Cleaning
        category_text = ' '.join(category_text.split()) # To remove multiple
↪  blank space
```

```
        category_text = category_text.replace(', -&- ', ' -&- ') # Cleaning
        categories_final.append(category_text)
      # Extracting the agencies
      for link in list_links_temp:
        time.sleep(2) # Adding 2 seconds wait to prevent potential
↪  server-side block.
        enforcement_retrieved = requests.get(link)
        enforcement_content = BeautifulSoup(enforcement_retrieved.content,
↪  'lxml')
        box_agency = enforcement_content.find_all('ul', class_ = "usa-list
↪  usa-list--unstyled margin-y-2")
        agency_info_prelim = box_agency[0].find_all('li')[1].text
        agency_info_final = agency_info_prelim.replace('Agency:', '')
        agencies_final.append(agency_info_final)

    # Now triggering the alert for older dates if the alert has been set "ON"
    if found_date_older == True:
      print(f'At least one date is older than the year and month provided.
       ↪  Scraping stopped at page {page_number}.')
      break
    else:
      pass

  # We have all the list filled, we can create the tidy dataframe now
  enforcement_data = pd.DataFrame({
    'title_enforcement' : titles_final,
    'date_enforcement' : dates_final,
    'agency_enforcement' : agencies_final,
    'category_enforcement' : categories_final,
    'link_enforcement' : links_final
  })

  # Returning final filtered dataframe
  return enforcement_data
```

With the function that we built, let's start collecting the enforcement actions since January 2023

```
enforcement_since_2023 = crawl_enforcement_data(2023, 1)

print(f'There are {len(enforcement_since_2023)} enforcement actions in our
↪  final dataframe')
```

```
# Saving the dataframe into a .csv file
enforcement_since_2023.to_csv('N:/3 MES DOSSIERS SECONDAIRES/MASTER
 ↪  PREPARATION PROGRAM/University of Chicago/DAP
 ↪  II/problem-set-5-RalphValiere/enforcement_actions_2023_january.csv',
 ↪  index = False)
```

Since we have already scrapped and save the data, we will not evaluate the previous code when knitting. We will use the saved .csv file instead. This is only for knitting purpose.

```
base_path = "N:/3 MES DOSSIERS SECONDAIRES/MASTER PREPARATION
 ↪  PROGRAM/University of Chicago/DAP II/problem-set-5-RalphValiere"
file_path = os.path.join(base_path, 'enforcement_actions_2023_january.csv')

enforcement_since_2023 = pd.read_csv(file_path)
```

Based on those results, we can find the details for the earliest enforcement in our dataframe.

```
print(f'The date for the earliest enforcement action scraped is:
 ↪  {min(enforcement_since_2023['date_enforcement'])}')

# Finding the detail for this enforcement
details_earliest_2023after =
 ↪  enforcement_since_2023.sort_values('date_enforcement').head(1)
print('The details for the earliest enforcement is:\n',
'title: ', details_earliest_2023after['title_enforcement'], '\n',
'date: ', details_earliest_2023after['date_enforcement'], '\n',
'agency: ', details_earliest_2023after['agency_enforcement'], '\n',
'catergory: ', details_earliest_2023after['category_enforcement'], '\n',
'link: ', details_earliest_2023after['link_enforcement'], '\n'
)
print('Please also note that in some case, there might be several enforcement
 ↪  at the earliest dates.\nIn those cases, this program will only pick just
 ↪  one of the earliest cases.')
```

```
The date for the earliest enforcement action scraped is: 2023-01-03
The details for the earliest enforcement is:
 title:  1533    Podiatrist Pays $90,000 To Settle False Billin...
Name: title_enforcement, dtype: object
 date:  1533    2023-01-03
Name: date_enforcement, dtype: object
```

```
 agency:  1533    U.S. Attorney's Office, Southern District of T...
Name: agency_enforcement, dtype: object
 catergory:  1533    Criminal and Civil Actions
Name: category_enforcement, dtype: object
 link:  1533    https://oig.hhs.gov/fraud/enforcement/podiatri...
Name: link_enforcement, dtype: object

Please also note that in some case, there might be several enforcement at the
earliest dates.
In those cases, this program will only pick just one of the earliest cases.
```

- c. Let's test our dynamic scraper function by collecting the actions since January 2021. (This will indeed take time again).

```
enforcement_since_2021 = crawl_enforcement_data(2021, 1)

print(f'There are {len(enforcement_since_2021)} enforcement actions in our
 ↪  final dataframe')
```

We have noticed that the "agency_enforcement" needs a bit of cleaning. However, we will clean those values in the next sections in specific cases, as it might to tedious to clean the entire dataset for now. We can save this data set as .csv file.

```
# Saving the dataframe into a .csv file
enforcement_since_2021.to_csv('N:/3 MES DOSSIERS SECONDAIRES/MASTER
 ↪  PREPARATION PROGRAM/University of Chicago/DAP
 ↪  II/problem-set-5-RalphValiere/enforcement_actions_2021_january.csv',
 ↪  index = False)
```

Since we have already scrapped and save the data, we will not evaluate the previous code when knitting. We will use the saved .csv file instead. This is only for knitting purpose.

```
base_path = "N:/3 MES DOSSIERS SECONDAIRES/MASTER PREPARATION
 ↪  PROGRAM/University of Chicago/DAP II/problem-set-5-RalphValiere"
file_path = os.path.join(base_path, 'enforcement_actions_2021_january.csv')

enforcement_since_2021 = pd.read_csv(file_path)
```

We can find the details for the earliest enforcement in this new dataframe.

```
print(f'The date for the earliest enforcement action scraped is:
 ↪  {min(enforcement_since_2021['date_enforcement'])}')

# Finding the detail for this enforcement
details_earliest_2021after =
 ↪  enforcement_since_2021.sort_values('date_enforcement').head(1)
print('The details for the earliest enforcement is:\n',
'title: ', details_earliest_2021after['title_enforcement'], '\n',
'date: ', details_earliest_2021after['date_enforcement'], '\n',
'agency: ', details_earliest_2021after['agency_enforcement'], '\n',
'catergory: ', details_earliest_2021after['category_enforcement'], '\n',
'link: ', details_earliest_2021after['link_enforcement'], '\n'
)
print('Please also note that in some case, there might be several enforcement
 ↪  at the earliest dates.\nIn those cases, this program will only pick just
 ↪  one of the earliest cases.')
```

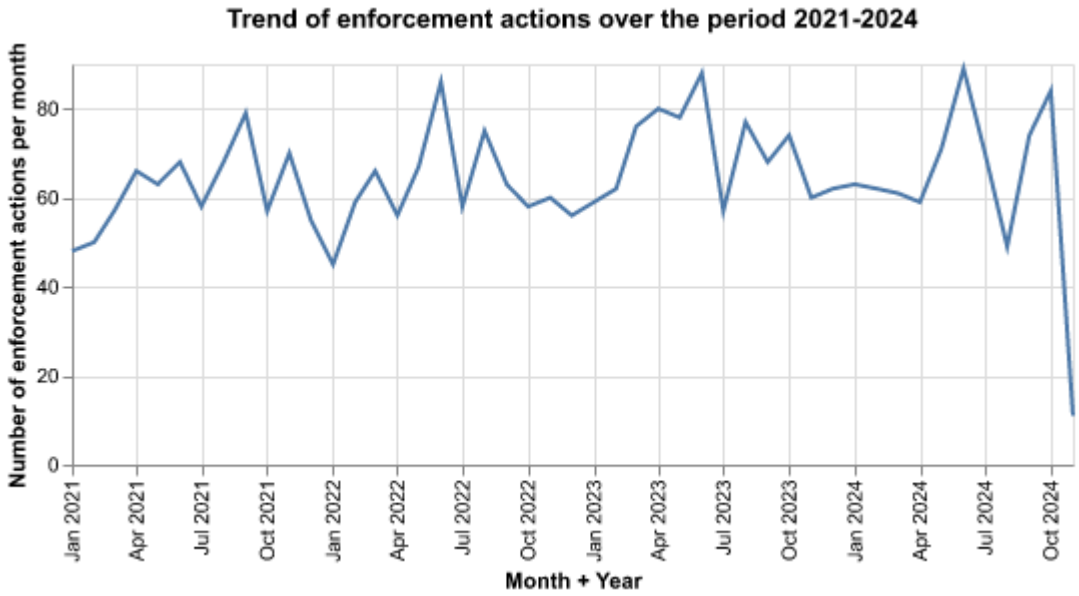
```
The date for the earliest enforcement action scraped is: 2021-01-04
The details for the earliest enforcement is:
 title:  3021    The United States And Tennessee Resolve Claims...
Name: title_enforcement, dtype: object
 date:  3021    2021-01-04
Name: date_enforcement, dtype: object
 agency:  3021    U.S. Attorney's Office, Middle District of Ten...
Name: agency_enforcement, dtype: object
 catergory:  3021    Criminal and Civil Actions
Name: category_enforcement, dtype: object
 link:  3021    https://oig.hhs.gov/fraud/enforcement/the-unit...
Name: link_enforcement, dtype: object

Please also note that in some case, there might be several enforcement at the
earliest dates.
In those cases, this program will only pick just one of the earliest cases.
```

Step 3: Plot data based on scraped data

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

Let's plot a line chart that shows the number of enforcement actions over time (aggregated to each month+year) overall since January 2021.

```
graph_enforcement_year =
 ↪  alt.Chart(enforcement_since_2021).mark_line().encode(
  alt.X(
    'yearmonth(date_enforcement)',
    title = 'Month + Year',
    axis = alt.Axis(labelAngle = 270)
  ),
  alt.Y('count()', title = 'Number of enforcement actions per month')
).properties(
  title = 'Trend of enforcement actions over the period 2021-2024',
  width = 500,
  height = 200
)
graph_enforcement_year
```



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

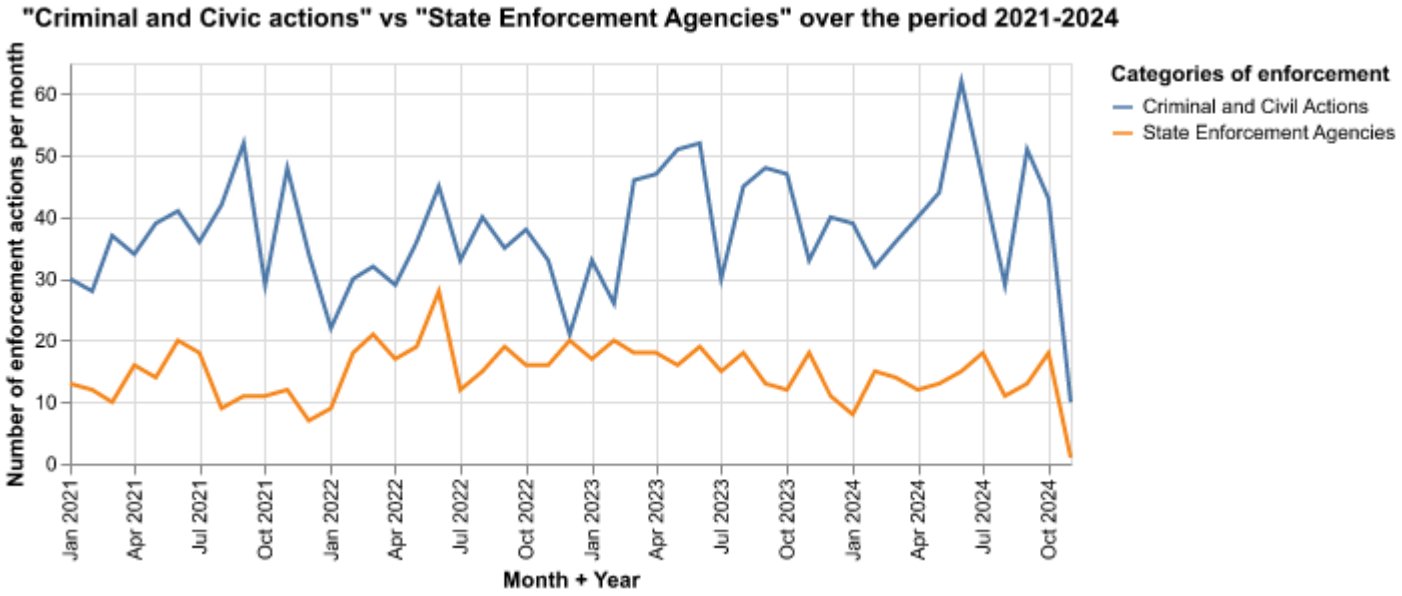
For this part, we will plot a line chart that shows the number of enforcement actions split out by: "Criminal and Civil Actions" vs. "State Enforcement Agencies".

First, we will deal with the joint categories we had previously created, as some enforcement fits into two categories. We will split those rows into two, with the only difference being the category. Considering the number of observations we have, and considering that there are not many of those cases, double counting won't affect our results significantly.

Second, we will filter to only have two categories we are plotting : “Criminal and Civil Actions” and “State Enforcement Agencies”.

Then we will make the plot.

```
# Splitting the joint categories
enforcement_since_2021['category_enforcement'] =
 ↪  enforcement_since_2021['category_enforcement'].str.split(' -&- ') # The
 ↪  splitter we used for the joint categories was pretty much handy.
enforcement_since_2021 =
 ↪  enforcement_since_2021.explode('category_enforcement')
enforcement_since_2021 = enforcement_since_2021.reset_index(drop = True)

# Filtering to only have the two groups
condition_two_categories = [category in ['Criminal and Civil Actions', 'State
 ↪  Enforcement Agencies'] for category in
 ↪  enforcement_since_2021['category_enforcement']]
enforcement_crim_vs_state = enforcement_since_2021[condition_two_categories]

# Plotting the two categories now.
graph_two_categories =
 ↪  alt.Chart(enforcement_crim_vs_state).mark_line().encode(
  alt.X(
    'yearmonth(date_enforcement):T',
    title = 'Month + Year',
    axis = alt.Axis(labelAngle = 270)
  ),
  alt.Y('count()', title = 'Number of enforcement actions per month'),
  alt.Color('category_enforcement:N', title = 'Categories of enforcement')
).properties(
  title = '"Criminal and Civic actions" vs "State Enforcement Agencies" over
 ↪  the period 2021-2024',
  width = 500,
  height = 200
)
graph_two_categories
```



For the next step, we asked Perplexity to create different lists of 10 key words related to each specific subcategories we have, while asking Perplexity to make sure that there is no common key word for those lists. We push the prompt given to Perplexity and the output in a text document. The text document was also pushed into the repo.

```
# Adjusting the list of the keywords provided by Perplexity
healthcare_fraud_keywords = [
    "Health", "Medicare", "Medicaid", "Prescription", "Diagnosis",
    "Hospice", "Telemedicine", "Copayment", "Experiments", "Medical",
    "Medecine", "Patients", "Preventive", "Hospital", "Illness",
    "Practioner", "Nurse", "Doctors", "Pharmacist", "Pharmaceutical",
    "Surgery", "Surgeon", "Physician", "Testing", "Pharmacy"
]

financial_fraud_keywords = [
    "Financial", "Embezzlement", "Ponzi", "Securities", "Insider",
    "Mortgage", "Identity", "Bankruptcy", "Laundering", "Wire",
    "Accounting", "Pump", "Dump", "Predatory", "Lending",
    "Falsifying", "Credit", "Check", "Investment", "Skimming",
    "Bank", "Stock", "Cash", "Accounting", "Accountant",
    "Racketeering", "Ponzi", "Pyramidal", "Arbitrage", "Finance"
]

drug_enforcement_keywords = [
    "Narcotics", "Trafficking", "Smuggling", "Cartel", "Controlled",
```

```
    "Substances", "Opioid", "Fentanyl", "Methamphetamine", "Cocaine",
    "Heroin", "Prescription", "Manufacturing", "Distribution", "Addiction",
    "Seizure", "Enforcement", "Illicit", "Cultivation", "Synthetic",
    "Drug", "Narcotics", "Analgesics", "Stimulants", "Sedatives",
    "Psychotropic", "Counterfeit", "Precursors"
]

bribery_corruption_keywords = [
    "Bribery", "Corruption", "Extortion", "Kickbacks", "Collusion",
    "Nepotism", "Cronyism", "Graft", "Embezzlement", "Misappropriation",
    "Clientelism", "Favoritism", "Lobbying", "Influence", "Peddling",
    "Slush", "Patronage", "Kleptocracy", "Malfeasance", "Quid",
    "Coercion", "Profiteering"
]
```

Next, we will filter the previous dataset generated ("enforcement_crim_vs_state") to only have the cases where the category is "Criminal and Civil Actions". We will then create the subcategories based on the keyowrds we created.

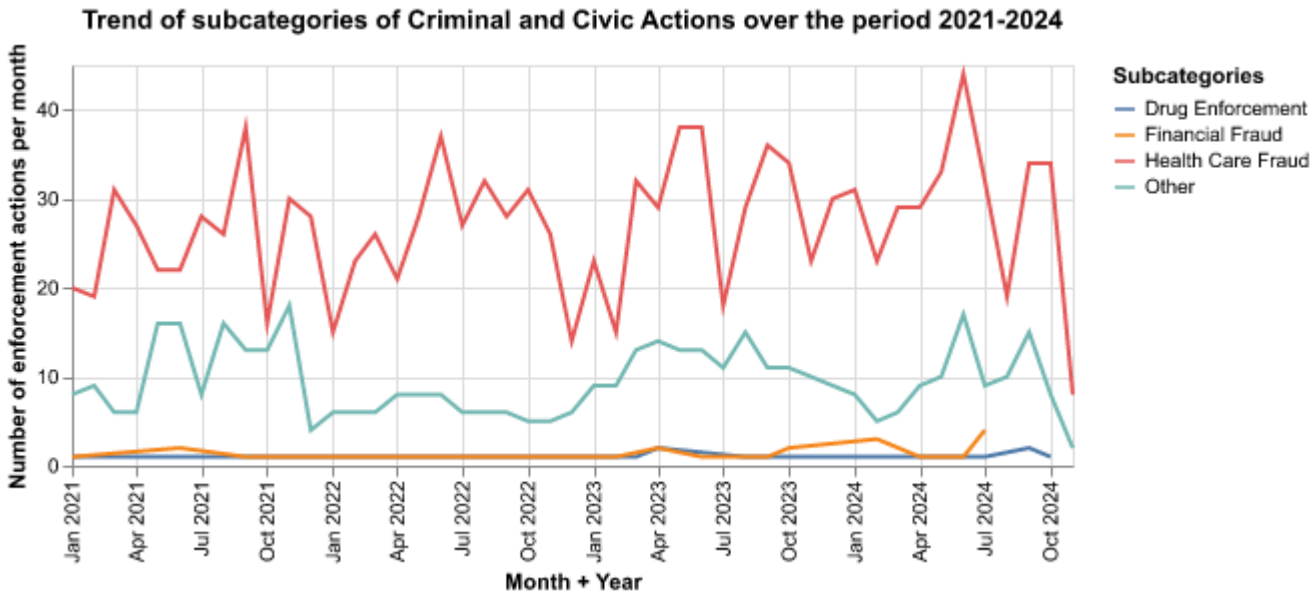
For each title, we are going to check the number of keywords detected from each subcategory list. The subcategory that has more keywords detected will be the final category. When there is 0 keywords from all subcategories in the title, then the subcategory will be "Other".

```
# Filtering the previous dataset to have only the category of "Criminal and
 ↪  Civil Actions"
only_criminal_civil =
 ↪  enforcement_crim_vs_state[enforcement_crim_vs_state['category_enforcement']
 ↪  == 'Criminal and Civil Actions'].copy()

# Constructing the subcategories
  # We are building a function to calculate the number of keywords from each
   ↪  subcategory detected in a title (as we will this more than once).
def check_num_keywords(list_keywords, title_in_string):
  keyword_in_title = [keyword in title_in_string for keyword in
 ↪  list_keywords]
  num_keyword = sum(keyword_in_title)
  return num_keyword

  # We can use this function to finalize the process of creating
   ↪  subcategories
subcategories_enforcement = [] # Initializing the list that will be use to
 ↪  create the new varibale in the dataframe.
```

```
for title in only_criminal_civil['title_enforcement']:
  num_keywords_health = check_num_keywords(healthcare_fraud_keywords, title)
  num_keywords_financial = check_num_keywords(financial_fraud_keywords,
 ↪  title)
  num_keywords_drug = check_num_keywords(bribery_corruption_keywords, title)
  num_keywords_bribery_corruption =
 ↪  check_num_keywords(bribery_corruption_keywords, title)
  dict_num_keys = {
    'Health Care Fraud' : num_keywords_health,
    'Financial Fraud' : num_keywords_financial,
    'Drug Enforcement' : num_keywords_drug,
    'Bribery/Corruption' : num_keywords_bribery_corruption
  }
  subcategory_max_keywords = max(dict_num_keys, key = dict_num_keys.get)
  if dict_num_keys[subcategory_max_keywords] == 0:
    subcategories_enforcement.append('Other')
  else:
    subcategories_enforcement.append(subcategory_max_keywords)

# Appending the list to the dataframe
only_criminal_civil['subcategory_enforcement'] = subcategories_enforcement
```

We can create the plot now.

```
graph_subcategories_crimandcivic =
 ↪  alt.Chart(only_criminal_civil).mark_line().encode(
  alt.X(
    'yearmonth(date_enforcement):T',
    title = 'Month + Year',
    axis = alt.Axis(labelAngle = 270)
  ),
  alt.Y('count()', title = 'Number of enforcement actions per month'),
  alt.Color('subcategory_enforcement:N', title = 'Subcategories')
).properties(
  title = 'Trend of subcategories of Criminal and Civic Actions over the
 ↪  period 2021-2024',
  width = 500,
  height = 200
)
graph_subcategories_crimandcivic
```



Step 4: Create maps of enforcement activity

1. Map by State (PARTNER 1)

For this section, we are going to plot a choropleth of the number of enforcement actions for each state.

Let's first filter by agencies, to have only the observations for which there is "State of" in the the agency name. We will clean the agency name after this filter.

```
# Filtering the "enforcement_since_2021" dataset to have only case where the
 ↪  agency is a state
condition_stateof_in = ['State of' in agency for agency in
 ↪  enforcement_since_2021['agency_enforcement']]
agency_with_state = enforcement_since_2021[condition_stateof_in]

# Let's clean and creating a new column
agency_with_state['state_enforcement'] = [state.replace('State of ', '') for
 ↪  state in agency_with_state['agency_enforcement']]
```

Now, we can read the us state shape file to create the mapping.

```
# Reading the census state file
state_shapefile_path = "N:/3 MES DOSSIERS SECONDAIRES/MASTER PREPARATION
 ↪  PROGRAM/University of Chicago/DAP
 ↪  II/problem-set-5-RalphValiere/cb_2018_us_state_500k/cb_2018_us_state_500k.shp"
state_geo_data = gpd.read_file(state_shapefile_path)

# Grouping agency_with_state by state
group_state = agency_with_state.groupby('state_enforcement')
num_enforcement_state = group_state.apply(lambda group: len(group))
num_enforcement_state = num_enforcement_state.reset_index()
num_enforcement_state.columns = ['state_enforcement', 'number_enforcement']

# Before merging, let's check if all the states names in
 ↪  num_enforcement_state are also in state_geo_data. If there are difference
 ↪  in the way the state name is written, we will correct them to have them
 ↪  match. This will avoid generating missing values just because of thoses
 ↪  mismatch.
all_state_in_geo = all(state in list(state_geo_data['NAME']) for state in
 ↪  list(num_enforcement_state['state_enforcement']))
print(f'Results for checking if all state names in
 ↪  num_enforcement_state\nmatch the state name in state_geo_data:
 ↪  {all_state_in_geo}')

# Importing the geometry by merging agency_with_state into a geodataframe
agency_with_state_geo = num_enforcement_state.merge(
  state_geo_data,
  how = 'right',
  left_on = 'state_enforcement',
  right_on = 'NAME'
)

# Let's fill the missing values with 0. The rationale being is that if a cell
 ↪  end up having NaN after the merging, it only implies that this state
 ↪  didn't have any enforcement action in the period 2021-2021 because we
 ↪  already checked for mismatch of names and there were not.
agency_with_state_geo['number_enforcement'] =
 ↪  agency_with_state_geo['number_enforcement'].fillna(0)
agency_with_state_geo['number_enforcement'] =
 ↪  agency_with_state_geo['number_enforcement'].astype(int) # Converting from
 ↪  float to integer

# Converting the new dataframe into a geo dataframe
```

```
agency_with_state_geo = gpd.GeoDataFrame(agency_with_state_geo, geometry =
 ↪  'geometry')

# Finishing the plot
fig, graph_enforcement_ax = plt.subplots(figsize=(12, 6))
graph_enforcement_perstate = agency_with_state_geo.plot(
  column = 'number_enforcement',
  cmap = 'viridis',
  legend = True,
  ax = graph_enforcement_ax
)
plt.axis("off")
plt.title(
  'Overall number of enforcement actions per states for 2021-2024',
  fontsize = '15',
  loc = 'center'
)
legends = plt.legend(
  title ='Number of enforcement',
  title_fontsize = '13',
  fontsize = '10',
  loc = 'right',
  bbox_to_anchor = (1.2, 0.5)
)
legends.get_title().set_rotation(90)
plt.gcf().set_facecolor('lightgray')
graph_enforcement_perstate
```

```
Results for checking if all state names in num_enforcement_state
match the state name in state_geo_data: True
```

2. Map by District (PARTNER 2)

Now, let's do the same process at district level. Let's first filter by agencies, to have only the observations for which there is "District" in the the agency name. We will clean the agency name after this filter. For some observations, there might be two districts involved, however as a fast pass method, we will only consider one of those districts in those cases.

```
# Filtering the "enforcement_since_2021" dataset to have only case where the
 ↪  agency is a state
condition_district_in = ['District' in agency for agency in
 ↪  enforcement_since_2021['agency_enforcement']]
agency_with_district = enforcement_since_2021[condition_district_in]

# Let's clean and creating a new column "district_enforcement". The districts
 ↪  are usually separated with other texts  with a "," or a ";"
agency_with_district['district_enforcement'] = [district.split(',')[-1] for
 ↪  district in agency_with_district['agency_enforcement']]
agency_with_district['district_enforcement'] = [district.split(';')[-1] for
 ↪  district in agency_with_district['district_enforcement']] # Second layer
 ↪  of cleaning for the new column
agency_with_district['district_enforcement'] =
 ↪  agency_with_district['district_enforcement'].str.strip() # To remove
 ↪  white space which could create issues with the merge coming next.
```

Now, we can read the us district shape file to create the mapping.

```
# Reading the US Attorney District shapefile file
usdistrict_shapefile_path = "N:/3 MES DOSSIERS SECONDAIRES/MASTER PREPARATION
 ↪  PROGRAM/University of Chicago/DAP II/problem-set-5-RalphValiere/US
 ↪  Attorney Districts Shapefile
 ↪  simplified_20241109/geo_export_8d61a61c-b22d-4587-8fb9-6c5739802da7.shp"
usdistrict_geo_data = gpd.read_file(usdistrict_shapefile_path)

# Grouping with_district_agency by disctrit

group_disctrict = agency_with_district.groupby('district_enforcement')
num_enforcement_district = group_disctrict.apply(lambda group: len(group))
num_enforcement_district = num_enforcement_district.reset_index()
num_enforcement_district.columns = ['district_enforcement',
 ↪  'number_enforcement']

# Before merging, let's check if all the district names in
 ↪  num_enforcement_district are also in usdistrict_geo_data. If there are
 ↪  difference in the way the district name is written, we will correct them
 ↪  to have them match. This will avoid generating missing values just
 ↪  because of thoses mismatch.
all_district_in_geo = all(district in list(usdistrict_geo_data['judicial_d'])
 ↪  for district in list(num_enforcement_district['district_enforcement']))
print(f'Results for checking if all state names in
 ↪  num_enforcement_state\nmatch the state name in state_geo_data:
 ↪  {all_district_in_geo}')
# There are mismatchs indeed.

# Let's check which districts have the mismatch
list_mismatched_name = []
  # Let's build the function to check the number of mismatch. We will use it
   ↪  often after cleaning the mismatch
def number_mismatch(dataset1, dataset2):
  for district in list(dataset1['district_enforcement']):
    if district in list(dataset2['judicial_d']):
      pass
    else:
      list_mismatched_name.append(district)
  print(list_mismatched_name)
```

```
  num_mismatch = len(list_mismatched_name)
  return num_mismatch
init_num_mismatch = number_mismatch(num_enforcement_district,
 ↪  usdistrict_geo_data) # To use as a reference to see if we are progressing
 ↪  in the cleaning process

  # Let's start by removing the "†" in some of those names
num_enforcement_district['district_enforcement'] = [district.replace(' †††',
 ↪  '') for district in num_enforcement_district['district_enforcement']]
num_enforcement_district['district_enforcement'] = [district.replace(' ††',
 ↪  '') for district in num_enforcement_district['district_enforcement']]
  # Removing "Inspector General" in some names
num_enforcement_district['district_enforcement'] = [district.replace('
 ↪  Inspector General', '') for district in
 ↪  num_enforcement_district['district_enforcement']]
  # Removing 'U.S. Attorney's Office' in the names
for index in range(len(num_enforcement_district)):
  district = num_enforcement_district['district_enforcement'][index]
  district = district.replace('U.S. Attorney's Office ', ',')
  district = district.split(',')[-1]
  num_enforcement_district['district_enforcement'][index] = district
  # Cleaning for the case of "Pennsylvani"
for index in range(len(num_enforcement_district)):
  district = num_enforcement_district['district_enforcement'][index]
  if district == 'Eastern District of Pennsylvani':
    num_enforcement_district['district_enforcement'][index] = 'Eastern
 ↪  District of Pennsylvania'
  # Now we left with 5 cases that doesn't have clear mismatch. We had to
   ↪  check in the usdistrict_geo_data to see if the problem is not there
   ↪  too. We found that:
  # a- There is no District of Idaho Boise, just "District of Idaho"
  # b- In the usdistrict_geo_data, instead of having District of Columbia,
   ↪  there was "District of District of Columbia"
  # c- There are three different district in Florida. Since the number of
   ↪  enforcement for "District of Florida" is just one, we will reassign the
   ↪  name to any of those three district name.
  # d- There is no "Southern District of Pennsylvania". Since the number of
   ↪  enforcement for "Southern District of Pennsylvania" is just one, we
   ↪  will reassign the name to any of those three district name in
   ↪  Pennsylvania.
for index in range(len(num_enforcement_district)):
  district = num_enforcement_district['district_enforcement'][index]
```

```
  if district == 'District of Idaho Boise':
    num_enforcement_district['district_enforcement'][index] = 'District of
 ↪  Idaho'
  elif district == 'District of Florida':
    num_enforcement_district['district_enforcement'][index] = 'Middle
 ↪  District of Florida'
  elif district == 'Southern District of Pennsylvania':
    num_enforcement_district['district_enforcement'][index] = 'Middle
 ↪  District of Pennsylvania'
  # Let's complete this cleaning by solving the case of "District of
   ↪  Columbia"
for index in range(len(usdistrict_geo_data)):
  district = usdistrict_geo_data['judicial_d'][index]
  if district == 'District of District of Columbia':
    usdistrict_geo_data['judicial_d'][index] = 'District of Columbia'
# Data is fully cleaned now

# Export the geometry by merging and transforming "agency_with_district" into
 ↪  a geodataframe.
agency_with_district_geo = num_enforcement_district.merge(
  usdistrict_geo_data,
  how = 'left',
  left_on = 'district_enforcement',
  right_on = 'judicial_d'
)
agency_with_district_geo = gpd.GeoDataFrame(agency_with_district_geo,
 ↪  geometry = 'geometry')

# Let's plot now
fig, graph_enforcement_ax = plt.subplots(figsize=(20, 12))
graph_enforcement_perdistrict = agency_with_district_geo.plot(
  column = 'number_enforcement',
  cmap = 'plasma',
  legend = True,
  ax = graph_enforcement_ax
)
plt.axis("off")
plt.title(
  'Overall number of enforcement actions\nper district for 2021-2024',
  fontsize = '40',
  loc = 'center',
  pad = 75
```

```
)
legends = plt.legend(
  title ='Number of enforcement',
  title_fontsize = '20',
  fontsize = '15',
  loc = 'right',
  bbox_to_anchor = (1.2, 0.5)
)
legends.get_title().set_rotation(90)
plt.gcf().set_facecolor('lightgray')
graph_enforcement_perdistrict
```

```
Results for checking if all state names in num_enforcement_state
match the state name in state_geo_data: False
['District of Columbia', 'District of Columbia Inspector General', 'District
of Florida', 'District of Idaho Boise', 'District of Massachusetts †††',
'District of South Dakota ††', 'Eastern District of Pennsylvani', 'Southern
District of New York ††', 'Southern District of Pennsylvania', 'U.S.
Attorney's Office District of Connecticut', 'U.S. Attorney's Office District
of Rhode Island', 'U.S. Attorney's Office Eastern District of Pennsylvania',
'U.S. Attorney's Office Eastern District of Washington', 'U.S. Attorney's
Office Northern District of Indiana', 'Western District of Virginia ††']
```

Extra Credit

1. **Merge zip code shapefile with population**
2. **Conduct spatial join**
3. **Map the action ratio in each district**