CS109A Introduction to Data Science

Homework 1: Data Collection, Parsing, and Quick Analyses

Harvard University
Fall 2021
Instructors: Pavlos Protopapas and Natesh Pillai

In [1]:

```
## RUN THIS CELL TO GET THE RIGHT FORMATTING
import requests
from IPython.core.display import HTML
styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2021-CS109A/m
HTML(styles)
```

Out[1]:

Overview

In this homework, your goal is to learn how to acquire, parse, clean, and analyze data. Toward this goal, we will address certain questions about COVID, and you will scrape data directly from a website. For the remainder of the semester, we will provide you data files directly; however, since real-world problems often require gathering information from a variety of sources, including the Internet, web scraping is a highly useful skill to have.

Instructions

- To submit your assignment, follow the instructions given in Canvas.

Learning Objectives

- Get started using Jupyter Notebooks (https://jupyter.org/), which are incredibly popular, powerful, and will be our medium of programming for the duration of CS109A and CS109B.
- Become familiar with how to access and use data from various sources (i.e., web scraping and directly from files).
- Gain experience with data exploration and simple analysis.
- Become comfortable with pandas (https://pandas.pydata.org/) as a means of storing and working with data.
- Reflect on what further analysis you may wish to do with this data. For example, given the material we've covered so far, what *more* do you wish you had the ability to do (e.g., modelling, prediction, etc). That is, think about questions you may have about the data, and try to imagine what types of tools you might need to help answer your questions.

Notes

- Exercise **responsible scraping**. Web servers can become slow or unresponsive if they receive too many requests from the same source in a short amount of time. In your code, use a delay of 2 seconds between requests. This helps to not get blocked by the target website -- imagine how frustrating it would be to have this occur. Section 1 of this homework involves saving the scraped web pages to your local machine. Thus, after completing Section 1, you do not need to re-scrape any of the pages, unless you wish to occasionally grab the latest data.
- **Web scraping requests can take several minutes**. This is another reason why you should not wait until the last minute to do this homework.
- As you run a Jupyter Notebook, it maintains a running state of memory. Thus, the order in which you run cells matters and plays a crucial role; it can be easy to make mistakes based on *when* you run different cells as you develop and test your code. Before submitting every Jupyter Notebook homework assignment, be sure to restart your Jupyter Notebook and run the entire notebook from scratch, all at once (i.e., "Kernel -> Restart & Run All"). Just make sure to not re-run the time intensive tasks unnecessarily. In this notebook for example, you could declare a variable to act as a 'setting' and use some controll logic to prevent a re-scrap from happening when not desired.
- We will be working with COVID data. COVID has impacted everyone in the world, and naturally some people have been greatly more affected than others. We, the teaching staff, are sensitive to this, empathize, and understand that working with COVID data may be unsettling to some. We apologize for any discomfort this may cause. Our intent with this assignment is purely pedagogical, and we'd like to remind students that data science and machine learning can be used to provide insights that can be used for good and invoke change. Toward this goal, parts of the homework are intended to shed light on the unfortunate, widespread inequality that exists. So, while this data may be unsettling, our aim is for the learned skills addressed here -- and in all future assignments -- to provide you with knowledge and confidence to do good work.

1. Obtaining Data (17 points)

For any given situation or scenario that we wish to understand, we will rely on having relevant data. Here, we are interested in the degree to which the SARS-CoV-2 virus has affected United States citizens (SARS-CoV-2 is the virus that causes the COVID-19 disease). The Centers for Disease Control and Prevention (CDC) provides relevant data from USAFacts.org that includes the number of confirmed COVID-19 cases on a per-county basis. Visit https://usafacts.org/visualizations/coronavirus-covid-19-spread-map/ (https://usafacts.org/visualizations/coronavirus-covid-19-spread-map/). At the bottom of the web page, in a blue table, you should see a list of every state, each of which has its own web page.

In this exercise, we will focus on automating the downloading of each state's data with Requests (https://docs.python-requests.org/en/master/) and then manipulating it with BeautifulSoup (https://www.crummy.com/software/BeautifulSoup/bs4/doc/).

But first, as we will do for every Jupyter Notebook, let's import necessary packages that we will use throughout the notebook (i.e., run the cell below).

In [2]:

```
# import the necessary libraries
import re
import requests
import pandas as pd
import numpy as np
from time import sleep
from bs4 import BeautifulSoup
import pickle # for loading a dictionary from disk
from typing import Optional # typehint that value can also be None

# NOTE: files will be saved to this directory, so you need to ensure
# that it exists on your system first (it should be visible from the
# directory of where you are running this Notebook file)
# i.e.,
# >> ls
# cs109a_hw1_student.ipynb
# data/
# state_data/
state_dir = "state_data/"
```

In [3]:

```
# we define this for convenience, as every state's url begins with this prefix
base_url = 'https://usafacts.org/visualizations/coronavirus-covid-19-spread-map/
```

> **Exercise 1.1 [1 pt]: Fetching Website data via Requests**
>
> Fetch the web page located at `base_url` and save the request's returned object (a Response object) to a variable named `home_page`.

In [4]:

```
# YOUR CODE HERE
home_page = requests.get(base_url)
# END OF YOUR CODE HERE
```

> **Exercise 1.2 [2 pts]:** In the cell below:
>
> - Write a line of code that prints to the screen the status of `home_page` (the web page's returned object). You should receive a code of 200 if the request was successful; then,
> - **When working with Jupyter Notebooks, avoiding unnecessarily long output in is essential.** Write code that prints the first 10,000 characters from the contents of `home_page` and enable scolling output for the cell (https://www.youtube.com/watch?v=U4usAUZCv_c&t=1s).

In [5]:

```
# YOUR CODE HERE
print(home_page.status_code)
print(home_page.text[:10000])
# END OF YOUR CODE HERE
```

```
200
<!doctype html><html lang="en"><head><script type="text/javascript">wi
ndow.NREUM||(NREUM={});NREUM.info = {"agent":"","beacon":"bam-cell.nr-
data.net","errorBeacon":"bam-cell.nr-data.net","licenseKey":"NRJS-c11b
817f31177e0b4d1","applicationID":"1475026924","applicationTime":2161.5
54995,"transactionName":"ZwZaNUEFVhZZAkNRWl5Mdg5BCVkJURtSXGBCChdL","qu
eueTime":0,"ttGuid":"50a3b7d4afb96077","agentToken":null}; (window.NRE
UM||(NREUM={})).loader_config={licenseKey:"NRJS-c11b817f31177e0b4d1",a
pplicationID:"1475026924"};window.NREUM||(NREUM={}),__nr_require=funct
ion(t,e,n){function r(n){if(!e[n]){var i=e[n]={exports:{}};t[n][0].cal
l(i.exports,function(e){var i=t[n][1][e];return r(i||e)},i,i.exports)}
return e[n].exports}if("function"==typeof __nr_require)return __nr_req
uire;for(var i=0;i<n.length;i++)r(n[i]);return r}({1:[function(t,e,n)
{function r(){}function i(t,e,n){return function(){return o(t,[u.now
()].concat(f(arguments)),e?null:this,n),e?void 0:this}}var o=t("handl
e"),a=t(8),f=t(9),c=t("ee").get("tracer"),u=t("loader"),s=NREUM;"undef
ined"==typeof window.newrelic&&(newrelic=s);var d=["setPageViewNam
e","setCustomAttribute","setErrorHandler","finished","addToTrace","inl
ineHit","addRelease"],p="api-",l=p+"ixn-";a(d,function(t,e){s[e]=i(p+
```

Exercise 1.3 [1 pt]:

In the cell below, create a new BeautifulSoup object that parses the `home_page` as an HTML document (can be done with 1 line of code)

In [6]:

```
# YOUR CODE HERE
home_page = BeautifulSoup(home_page.text, 'html.parser')
# END OF YOUR CODE HERE
```

Exercise 1.4 [8 pts]:

In the cell below, write code that uses the BeautifulSoup object to parse through the home page in order to extract the link for every state. Feel free to use Regular Expressions, in conjunction with any BeautifulSoup parsing. Specifically, the goal is to populate a `state_urls` dictionary by setting each key to be the state name and the value to be the full URL. When complete, there will be 51 keys (50 states + 1 for DC).

AS A CRITICAL EXAMPLE:

Within `state_urls`, one of your <key, value> pairs should be:

`"District of Columbia" : "https://usafacts.org/visualizations/coronavirus-covid-19-spread-map/state/district-of-columbia"`

The casing here is **incredibly** important because later, in Exercise 4, you will merge your data with another dataset that has casing of this form. Thus, our key here should be `District of Columbia` and not `District Of Columbia` or `district-of-columbia`.

NOTES:

- There are *many* solutions, but you may find it easiest to use Regular Expression(s)
- Pay attention to the casing example above, so that your later exercises go smoothly.
- Some HTML tag attributes may change over time. It your code stops working, make sure you are not targeting such ephemeral elements ('jss' class attributes are a common culprit)

In [7]:

```
state_urls = {}

# YOUR CODE HERE
list_state_name = [i.text for i in home_page.select("tbody a")]
state_url = ["https://usafacts.org"+i.get("href") for i in home_page.select("tbo

for i,d in enumerate(list_state_name):
    state_urls[d] = state_url[i]
# END OF YOUR CODE HERE
```

Run the cell below to help ensure your formatting is correct and has 51 <key, value> pairs.

In [8]:

```
# SANITY CHECK
if len(state_urls.keys()) != 51 or \
state_urls["District of Columbia"] != "https://usafacts.org/visualizations/coron
    print("** 1.4 is incorrect")
else:
    print("** 1.4 might be correct")
```

```
** 1.4 might be correct
```

We wish to use the data without having to re-download it every time. So, let's save each webpage to our local hard drive. **NOTE: It's probably okay to download all of the state web pages a few times a day, but it's safer to keep it to a minimum.**

Exercise 1.5 [5 pts]:

In the cell below, we will iterate through all <key, value> items in `state_urls` . Your job is to make a web request for each URL and save the **contents** out to a file on your hard drive (use `state_dir` , defined above, as the prefix to the path.)

NOTES:

- **Leave a 2 second pause between requests**
- You should be saving to a file the actual content of the webpage, not a BeautifulSoup object. That is, you should be able to open the saved files in an editor and see the HTML code, just as you could if you were to view the webpage in your browser and click 'View Page Source'.
- See official Python documentation (https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files) for details on how to read/write files to disk
- You should have saved 51 different files to your hard drive.
- **Once you have written the files you can comment out this cell. This will save time and prevent you from making unnecessary requests when you restart the kernel & re-run all cells in the noteboook before submitting (as you should!)**

In [9]:

```
# 1.5 (5 pts) -- save each webpage to disk
for state, url in state_urls.items():

    # YOUR CODE HERE
    r = requests.get(url)
    with open(state_dir+state+".txt", 'w') as file:
        file.write(r.text)
    # END OF YOUR CODE HERE

    sleep(2) # LEAVE THIS IN
```

2. Loading and Exploring Data (22 pts)

Now, let's actually use the data! Fortunately, it's saved to our local machine, so we don't need to re-crawl the data every time we wish to access it. We want you to understand that pandas (https://pandas.pydata.org/) is a library of useful data structures and operations, but we also wish to remind you that it isn't magic and it isn't the *only* way to do Data Science; it's just a tool to help, and you could do the same operations without pandas. Thus, here we ask you to perform a few operations without using pandas, and then in Exercise 3 we will use pandas.

Terminology Notice: In the United States, every state is comprised of many **counties.** You can think of a **county** as being a pretty large district.

First, run the cell below to construct `state_info` This is an example of a Python list comprehension (https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions).

In [10]:

```
state_info = [(state, state_dir + state) for state in state_urls.keys()]
```

Exercise 2.1 [10 pts]: Parsing and storing data

Complete the `load_covid_data()` function, which:

- Takes as input `state_info` , which is a list of tuples (https://docs.python.org/3.3/library/stdtypes.html?highlight=tuple#tuple): (state name, path to the corresponding file)
- Parses the contents of the file and extracts for **each county**:
  - 7 day average case
  - 7 day average deaths
  - # of confirmed cases (total)
  - # of deaths
  - Stores the above 4 pieces of data above as well as **population** in a **non-pandas** data structure named `covid_data` **for every county across every state**
- Returns `covid_data`

NOTES:

- **Attention: the population variable not in `state_info` . More on info on where to get this value is found in the green block below**

- To be clear, as of September 7, 2021, the webpage for Alabama currently lists 67 counties. District of Columbia has 1 county, and Wyoming has 23. Here we are asking you to store in `covid_data` *all counties* across every state. So, later, if we were wished to access just Wyoming's information, you could easily retrieve such for each of its 23 counties, or the info for any of the 67 counties in Alabama.
- `covid_data` **must not be a PANDAS data structure;** it must use a combination of lists and/or dictionaries. It's up to you to decide how to organize this, e.g., a lists of lists of lists, or a list of dictionaries, or a dictionary of dictionaries, or a dictionary of lists of lists, etc. A guiding decision should be ease of access for computing basic stats (Exercises 2.2, 2.3, and 2.4)
- For the duration of our using this data for the homework, be sure to **properly store the data with the correct data types;** that is, counts should be represented as Integers and rates should be represented as Floats. For example:
  - # of confirmed cases (total) should be an **Integer**
  - # of deaths should be an **Integer**
  - # of confirmed cases (per 100k) should be a **Float** (we haven't created this feature yet!)
  - 7 day average cases should be an **Integer** (you'd think an average should be a float but the values you scrapped were rounded to the nearest int)

Injecting population data

The table on usafacts.org you've just scrapped originally had additional columns related to county population. But these have recently been removed! We'd like you to be able to utilize the population data in the following section but also use up-to-date COVID data (so the Internet Archive (https://archive.org/) was not an option). And, though this information is available elsewhere on usafacts.org, we've decided that you've already done enough web scraping for one HW. So below we've provided a kludge (https://en.wikipedia.org/wiki/Kludge#Computer_science).

`population_dict` is a nested dictionary. The keys are states whose values are *themselves* dictionaries. Those '*inner*' dictionaries' keys are counties and their values are populations. It looks like this:

```
{'Alabama': {'Autauga County': 55869,
             'Baldwin County': 223234,
            ...
'Wyoming': {'Albany County': 38880,
            'Big Horn County': 11790,
            ...
```

To get at a population you could use double dictionary indexing like `population_dict['Alabama']['Autauga County']`

But not all of the counties you've scrapped have population data in this dictionary. So we've provided a helper function, `get_pop`, that will return `None` if the county data was not found. Use `get_pop` to inject popoulation data into your `covid_data` as you build it up in the `load_covid_data` function you'll implement below.

Final Note: you should *ignore counties with missing population data or populations of 0*. Simply do not add them to `covid_data` as it is constructed.

In [11]:

```
# load additional county population data as a nested dictionary
# you can read about this strange .pkl 'pickle' file here
# https://docs.python.org/3/library/pickle.html
with open('population.pkl', 'rb') as f:
    population_dict = pickle.load(f)

# not sure what's happening with the data types in the function header?
# check out: https://docs.python.org/3/library/typing.html#module-typing
def get_pop(state: str, county: str) -> Optional[int]:
    '''
    returns population of country, state (int)
    If county or state not found, returns None
    Example: get_pop('Alabama', 'Autauga County')
    '''
    try:
        return population_dict.get(state).get(county)
    except AttributeError:
        print('incorrect state name!')
        return None
```

In [12]:

```
def load_covid_data(state_info):
    covid_data = {}
    # YOUR CODE HERE
    for (state, state_path) in state_info:
        covid_data[state] = []
        with open(state_path, 'r') as f:
            soup = BeautifulSoup(f.read(), 'html.parser')
            counties = soup.find_all('a', href=re.compile('county/'))

            for c in counties:
                row = c.find_parent('tr')

                cols = [col.text.replace(',','') for col in row.find_all('td')]

                county_name = c.text
                pop = get_pop(state, county_name)
                if ((pop := get_pop(state, county_name)) is None) or (pop == 0):
                    continue
                covid_data[state].append({'county_name': county_name,
                                          'population': pop,
                                          '7_day_avg_cases': float(cols[0]),
                                          '7_date_ave_deaths': float(cols[1]),
                                          'cases': int(cols[2]),
                                          'deaths': int(cols[3])})
    # END OF YOUR CODE HERE
    return covid_data
```

Run the cell below (no changes necessary) to execute your code above

In [13]:

```
covid_data = load_covid_data(state_info)
```

Exercise 2.2 [4 pts]: Simple analytics

Complete the `calculate_county_stats()` function, which calculates:

1. The single county (and the state to which it belongs) that has the **lowest rate** of COVID cases per 100k people
2. The single county (and the state to which it belongs) that has the **highest rate** of COVID cases per 100k people

NOTES:

- Place your resulting variables within the blanks of the `print()` statements that we provide
- These values you report should be Floating point numbers (e.g., 3.4), not Integers (e.g., 3).
- If there are ties, return any one of the tied counties (see if you can do it in an unbiased way!)

In [14]:

```
# QUESTION: what is the output? Any variable should be returned? what do you mea

def calculate_county_stats(covid_data):

    # YOUR CODE HERE
    min_county_count = 999999
    min_county_name = ""
    max_county_count = -1
    max_county_name = ""

    # looks through every county in every state, while checking
    # to see if we have a new low or high
    for state in covid_data.keys():
        for county in covid_data[state]:
            if ((pop := county['population']) is None) or (pop == 0):
                continue
            covid_rate = round(county['cases'] / (pop/100000),2)
            if covid_rate < min_county_count:
                min_county_count = covid_rate
                min_county_name = county['county_name'] + " (" + state + ")"
            if covid_rate > max_county_count:
                max_county_count = covid_rate
                max_county_name = county['county_name'] + " (" + state + ")"

    print(min_county_name + " has the lowest COVID cases per 100k: " + str(float
    print(max_county_name + " has the highest COVID cases per 100k: " + str(floa

    # END OF YOUR CODE HERE

```

Run the cell below (no changes necessary) to execute your code above

In [15]:

```
calculate_county_stats(covid_data)
```

```
Lake and Peninsula Borough (Alaska) has the lowest COVID cases per 100
k: 0.0
Bristol Bay Borough (Alaska) has the highest COVID cases per 100k: 727
27.27
```

Exercise 2.3 [4 pts]: Simple analytics

Complete the `calculate_state_deaths()` function, which calculates:

1. The state that has the **lowest number** of deaths
2. The state that has the **highest number** of deaths

NOTES:

- Place your resulting variables within the blanks of the `print()` statements that we provide (don't just manually type your textual answers in the blanks)
- These values you report should be Integers, not Floating point numbers.
- If there are ties, return any of the tied states

In [16]:

```
def calculate_state_deaths(covid_data):

    # YOUR CODE HERE
    min_state_deaths = 999999
    min_state_name = ""
    max_state_deaths = -1
    max_state_name = ""
    for state in covid_data.keys():
        cur_state_count = 0
        for county in covid_data[state]:
            cur_state_count += county['deaths']

        if cur_state_count < min_state_deaths:
            min_state_deaths = cur_state_count
            min_state_name = state
        if cur_state_count > max_state_deaths:
            max_state_deaths = cur_state_count
            max_state_name = state

    print(min_state_name + " has the fewest COVID deaths: " + str(min_state_deat
    print(max_state_name + " has the most COVID deaths: " + str(max_state_deaths
    # END OF YOUR CODE HERE

```

Run the cell below (no changes necessary) to execute your code above

In [17]:

```
calculate_state_deaths(covid_data)
```

```
Hawaii has the fewest COVID deaths: 185
California has the most COVID deaths: 67794
```

Exercise 2.4 [4 pts]: Simple analytics

Complete the `calculate_state_deathrate()` function, which calculates:

1. The state that has the **lowest rate** of deaths based on its entire population
2. The state that has the **highest rate** of deaths based on its entire population

NOTES:

- To calculate a state's population, we are asserting that is sufficient to sum the population over all counties, and that each county's population can be calculated simply from the data fields stored within `covid_data`.
- **If a county has reported 0 COVID cases,** then we should ignore this county as we estimate its county population. Thus, that county would contribute 0 to its state population total.
- Round your results to the a single person (e.g., "1 out of every 2703 people has died" not 2703.4)
- Place your resulting variables within the blanks of the `print()` statements that we provide (don't just manually type your textual answers in the blanks)

In [18]:

```
def calculate_state_deathrate(covid_data):

    # YOUR CODE HERE
    min_state_death_rate = -1
    min_state_name = ""
    max_state_death_rate = 9999999
    max_state_name = ""

    for state in covid_data.keys():
        cur_state_deaths = 0
        cur_state_population = 0
        for county in covid_data[state]:
            if (county['cases'] > 0) and ((pop := county['population']) is not N
                cur_state_population += pop
                cur_state_deaths += county['deaths']

        cur_state_deathrate = float(cur_state_population) / cur_state_deaths

        if cur_state_deathrate > min_state_death_rate:
            min_state_death_rate = cur_state_deathrate
            min_state_name = state
        if cur_state_deathrate < max_state_death_rate:
            max_state_death_rate = cur_state_deathrate
            max_state_name = state

    print(min_state_name + " has the lowest COVID death rate; 1 out of every " +
    print(max_state_name + " has the highest COVID death rate; 1 out of every "

    # END OF YOUR CODE HERE

```

Run the cell below (no changes necessary) to execute your code above

In [19]:

```
calculate_state_deathrate(covid_data)
```

```
Hawaii has the lowest COVID death rate; 1 out of every 2385 people has
died
Mississippi has the highest COVID death rate; 1 out of every 310 peopl
e has died
```

3. PANDAS (36 pts)

What if we wanted to observe more than just the single-most extreme counties and states? What if we wanted to inspect all states, after having sorted the data by some feature? As you saw in the above exercises, doing the most basic analytics is possible, but it can quickly become cumbersome. As we learned in class, PANDAS is a great library that provides data structures that are highly useful for data analysis.

> **Exercise 3.1 [10 pts]: Converting to PANDAS**
>
> In Exercise 2, we worked with `covid_data`, which is comprises of some combination of lists and/or dictionaries.
>
> Complete the `convert_to_pandas()` function, which converts `covid_data` to a PANDAS DataFrame, whereby:
>
> - Each row corresponds to a unique county
> - The 4 columns are:
>   - county
>   - state
>   - # total covid cases (Integer)
>   - # case per 100k (Integer)
>   - # covid deaths (Integer)
> - The columns should be titled **exactly** as listed above
>
> **NOTE:**
>
> - If there exists multiple counties with the same name, each of which belonging to a different state, then there should be a distinct row for each.
> - The 2 columns that correspond to COVID counts should all be Integers (e.g., 1498), not Floating point digits (e.g., 1498.0)

In [20]:

```
def convert_to_pandas(covid_data):

    # YOUR CODE HERE
    covid_data_flipped = []
    for state, counties in covid_data.items():
        for county in counties:
            if ((pop:= county['population']) is None) or (pop == 0):
                continue
            cases = county['cases']
            cur_dict = {"county":county['county_name'], "state":state,
                        "# total covid cases": cases,
                        "# covid cases per 100k": cases/(pop/100000),
                       "# covid deaths": county['deaths']}
            covid_data_flipped.append(cur_dict)
    covid_df = pd.json_normalize(covid_data_flipped)
    # END OF YOUR CODE HERE
    return covid_df

```

Run the cell below (no changes necessary) to execute your code above and inspect the results.

In [21]:

```
covid_df = convert_to_pandas(covid_data)
```

In [22]:

```
covid_df.head()
```

Out[22]:

	county	state	# total covid cases	# covid cases per 100k	# covid deaths
0	Autauga County	Alabama	9744	17440.799012	140
1	Baldwin County	Alabama	36447	16326.814016	509
2	Barbour County	Alabama	3490	14137.567852	71
3	Bibb County	Alabama	4131	18446.905421	83
4	Blount County	Alabama	9818	16978.521772	160

In [23]:

```
covid_df.shape
```

Out[23]:

```
(3081, 5)
```

> **Exercise 3.2 [5 pts]: Simple analytics**
>
> Complete the `calculate_county_stats2()` function, **which should obtain identical information (other than ties) as problem 2.2, but now using the PANDAS `covid_df` DataFrame.**
>
> That is, it should calculates:
>
> 1. the single county (and the state to which it belongs) that has the **lowest rate** of COVID cases per 100k people
> 2. the single county (and the state to which it belongs) that has the **highest rate** of COVID cases per 100k people
>
> **NOTES:**
>
> - If there are ties, return any of the tied counties
> - Place your resulting variables within the `print()` statements that we provide (don't just manually type your textual answers in the blanks)
> - The values you report should be Floating point numbers (e.g., 3.4), not Integers (e.g., 3).

In [24]:

```
def calculate_county_stats2(covid_df):

    # YOUR CODE HERE
    sorted_df = covid_df.sort_values(by=['# covid cases per 100k'])
    lowest = sorted_df.iloc[0]
    highest = sorted_df.iloc[-1]

    print(f"{lowest['county']} ({lowest['state']}) has the lowest rate of confir
    print(f"{highest['county']} ({highest['state']}) has the highest rate of cor

    # END OF YOUR CODE HERE
```

Run the cell below (no changes necessary) to execute your code above

In [25]:

```
calculate_county_stats2(covid_df)
```

```
Kalawao County (Hawaii) has the lowest rate of confirmed COVID cases p
er 100k: 0.00
Bristol Bay Borough (Alaska) has the highest rate of confirmed COVID c
ases per 100k: 72,727.27
```

Exercise 3.3 [5 pts]: Simple analytics

Complete the `calculate_state_deaths2()` function, **which should obtain identical information as problem 2.3 (other than ties), but now using the PANDAS `covid_df` DataFrame.**

1. the state that has the **lowest number** of deaths
2. the state that has the **highest number** of deaths

NOTES:

- If there are ties, return any of the tied states
- Place your resulting variables within the `print()` statements that we provide (don't just manually type your textual answers in the blanks)
- The values you report should be Integers, not Floating point numbers.

In [26]:

```
def calculate_state_deaths2(covid_df):

    # YOUR CODE HERE
    state_deaths = covid_df.groupby('state').sum().sort_values(by=['# covid deat
    lowest = state_deaths.iloc[0]
    highest = state_deaths.iloc[-1]

    print(lowest.name + " has the fewest COVID deaths: " + str(lowest['# covid c
    print(highest.name + " has the most COVID deaths: " + str(highest['# covid c

    # END OF YOUR CODE HERE
```

Run the cell below (no changes necessary) to execute your code above

In [27]:

```
calculate_state_deaths2(covid_df)
```

```
Hawaii has the fewest COVID deaths: 185.0
California has the most COVID deaths: 67794.0
```

> **Exercise 3.4 [5 pts]: Simple analytics**
>
> Complete the `calculate_state_deathrate2()` function, **which should obtain identical information as problem 2.4, but now using the PANDAS `covid_df` DataFrame.** That is, return:
>
> 1. The state that has the **lowest rate** of deaths based on its entire population
> 2. The state that has the **highest rate** of deaths based on its entire population
>
> **NOTES:**
>
> - Just as in, 2.4, to calculate a state's population, we are asserting that is sufficient to sum the population over all counties -- and that each county's population can be calculated simply from the data fields stored within `covid_data` .
> - Just as in 2.4, counties with 0 COVID cases should contibute 0 to the total population of the state.
> - Round your results to the a single person (e.g., "1 out of every 2703 people has died" not 2703.4)
> - Place your resulting variables within the blanks of the `print()` statements that we provide (don't just manually type your textual answers in the blanks)

In [28]:

```
def calculate_state_deathrate2(covid_df):

    # YOUR CODE HERE
    covid_df2 = covid_df
    covid_df2['population'] = 100000*covid_df2['# total covid cases'] / covid_df
    covid_df2 = covid_df2.groupby('state').sum()
    covid_df2['death_rate'] = covid_df2['population'] / covid_df2['# covid deaths
    covid_df2 = covid_df2.sort_values(by=['death_rate'])

    print(covid_df2.iloc[-1].name + " has the lowest COVID death rate; 1 out of e
    print(covid_df2.iloc[0].name + " has the highest COVID death rate; 1 out of e

```

Run the cell below (no changes necessary) to execute your code above

In [29]:

```
calculate_state_deathrate2(covid_df)
```

```
Hawaii has the lowest COVID death rate; 1 out of every 2384 people has
died
Mississippi has the highest COVID death rate; 1 out of every 310 peopl
e has died
```

These are highly alarming and tragic statistics, and doing calculations like this can really put the severity of the virus into a grounded perspective. In order to perfectly understand the virus and its spread, everyone would be tested and we would have contact tracing. Without getting into socio-political issues, our point is that (1) we

wish to better understand the virus' effects; (2) naturally, any real-world data is messy, and thus we will never have *perfect* data.

Let's now attempt to understand *some* of the uncertainty around our COVID data. It's reasonable to believe that the # of COVID deaths is fairly reliable. That is, there are inevitably some false negatives -- people who died of COVID but were not accounted for, as other conditions were listed as the cause. However, the number of false positives is probably minimal -- if someone was denoted as dying from COVID, it's probably true. It's also the case that every disease has a mortality rate. For example, if 1,000 randomly-selected people contracted COVID, $N\%$ of them will die. We'd imagine that this percentage should be pretty constant throughout all people in the United States. Of course, we can think of reasons for this rate to not be perfectly consistent, as some people are at higher risk (e.g., older folks, people with pre-existing conditions, etc). Yet, we can imagine that this natural *variance* in the population to be fairly uniform throughout the USA at large. To this end, if all counties were equal in their **testing**, we ought to see a consistent ratio between: (a) the # of people who died from COVID; and (b) the # of people who tested positive for COVID. Within the medical domain, this ratio is referred to as the `case_fatality_rate`. For example, if 750 people tested positive for COVID, and 75 of those people died, then our `case_fatality_rate` would be 0.1 (meaning 10%).

Exercise 3.5 [5 pts]: Further analytics

Complete the `add_death_stats()` function below, which should add 3 new columns:

- `case_fatality_rate`
- `# covid deaths per 100k` and
- `population`

And return the updated DataFrame **sorted by `case_fatality_rate` in ascending order**

NOTES:

- `add_death_stats()` should return a new DataFrame that has 8 columns:
  - county
  - state
  - population
  - # total covid cases
  - # covid cases per 100k
  - # covid deaths
  - # covid deaths per 100k
  - case_fatality_rate
- DataFrame should be sorted by `case_fatality_rate` in ascending order
- Again, the values for `case_fatality_rate` should be < 1. A value of 1 would mean that 100% of people who tested positive for COVID also died.
- `# covid deaths per 100k` is simply defined as the # of COVID deaths for every 100,000 people. We calculate this on a per-county basis.
- Make sure you inspect your results thoroughly. You may have to address the results of divisions by zero (or prevent these divisions in the first place).

In [30]:

```
def add_death_stats(covid_df):

    # can add an infintesimal or fillna after the fact to handle nans from divic

    # YOUR CODE HERE
    covid_df['population'] = 100000*covid_df['# total covid cases'] / (covid_df[
#     covid_df.fillna(0, inplace=True)
    covid_df["population"] = covid_df["population"].astype('int32')

    covid_df['# covid deaths per 100k'] = 100000*covid_df['# covid deaths'] / (c
#     covid_df.fillna(0, inplace=True)
    covid_df["# covid deaths per 100k"] = covid_df["# covid deaths per 100k"].as

    covid_df['case_fatality_rate'] =  covid_df['# covid deaths'] / (covid_df['#
#     covid_df.fillna(0, inplace=True)
    covid_df = covid_df.sort_values(by=['case_fatality_rate'])
    # END OF YOUR CODE HERE
    return covid_df
```

Run the cell below (no changes necessary) to execute your code above

In [31]:

```
covid_updated = add_death_stats(covid_df)
covid_updated
covid_updated[["case_fatality_rate","# covid deaths per 100k"]][:-5].agg("mean")
```

Out[31]:

```
case_fatality_rate           0.017991
# covid deaths per 100k    241.443108
dtype: float64
```

> **Reflection:** Data Analysis allows us to better understand a system or scenario.

> **Exercise 3.6.1 [2 pts] Trends**
>
> Having looked at the results from Exercises 3.3, 3.4, and 3.5, what are some trends you've noticed and any conclusions you have? (2-3 sentences)?

Some counties (e.g. Harding County in New Mexico) have low number or rate of cases but high case fatality rate. While some counties (e.g. Rich County in Utah) have high number or rate of cases but low case fatality rate. This may imply the effectiveness of virus control actions and the performance of local healthcare system.

> **Exercise 3.6.2 [2 pts]: Data Reliability**
>
> Having looked at the results from Exercise 3.5 (i.e., `covid_updated` DataFrame), do you think the original data is reliable and accurate? Are there any potential biases that you're aware of or concerned about? Please explain (3-5 sentences).

The case fatality rate is not constant throughout the country. I doubt the reliability and accuracy of the original data because there may be biases when reporting deaths -- people who died of COVID but were not accounted for. And it is likely that not all the people are tested, so some cases were not accounted for. And there may be variations in the testing availability and capability among the counties.

Exercise 3.6.3 [1 pt]: Relationships Between Variables

If a county has 15 confirmed deaths, how many cases would you expect? What would you expect its population to be? Explain why (1-2 sentences in total)?

NOTE: For this question, we aren't evaluating the accuracy of your answer but your thought-process and reasoning.

Since the mean case fatality rate is 0.018, if a county has 15 confirmed deaths, it is expected that there are 15/0.018 = 833 cases.

Exercise 3.6.4 [1 pt]: Further Questions

What further questions do you wish to answer about COVID, including ones that may not be possible to answer from this data alone (e.g., Is there a correlation between the average age of people in a county and the # of COVID deaths)? Write at least 3 of your questions.

1. How vaccination rate influences the case rate and the case fatality rate? 2. Is keeping social distance effective on reduing the case rate? 3. Is there a correlation between the gender and the COVID deaths/cases?

4. MORE DATA (25 pts)

In order to better understand how COVID (and the testing thereof) has impacted our world, we could look at how it relates to demographics, income, education, health, and political voting. For this exercise, we will make use of `election2020_by_county.csv`.

Exercise 4.1 [4 pts]: Load more data

Complete the `merge_data()` function, which should:

1. First, load `election2020_by_county.csv` as a new DataFrame.
2. Then, using the state and county names (case-sensitive) in both DataFrames, merge this new DataFrame with your existing `covid_updated`.
3. Return the merged DataFrame

The returned `merged` DataFrame should contain all 8 columns from `covid_updated`:

- county
- state
- # total covid cases
- # covid cases per 100k
- # covid deaths
- population
- # covid deaths per 100k
- case_fatality_rate

along with these 15 columns from `election2020_by_county.csv`:

- hispanic
- minority
- female
- unemployed
- income
- nodegree
- bachelor
- inactivity
- obesity
- density
- cancer
- voter_turnout
- voter_gap
- trump
- biden

NOTES:

- We are dropping two columns from `election2020_by_county.csv`:
  - fipscode
  - population
- Do not attempt to manually fix any of the state or county names. That is, **our merging should require the state and county names to be identical (case-sensitive) between the two DataFrames.** If there is a discrepancy between the two, do not worry about adjusting these names to find a perfect match.

HINT: there are many ways to solve this, but you may find the pandas.merge() (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.merge.html) function can be really helpful

EXTRA INFORMATION: In case you're wondering what the different features/columns are in `election2020_by_county.csv`:

- state: the state in which the county lies
- fipscode: an ID to identify each county
- county: the name of each county
- population: total population
- hispanic: percent of adults that are hispanic
- minority: percent of adults that are nonwhite
- female: percent of adults that are female
- unemployed: unemployment rate, as a percent
- income: median income
- nodegree: percent of adults who have not completed high school
- bachelor: percent of adults with a bachelor's degree

- inactive: percent of adults who do not exercise in their leisure time
- obesity: percent of adults with BMI > 30
- density: population density, persons per square mile of land
- cancer: prevalence of cancer per 100,000 individuals
- voter_turnout: percentage of voting age population that voted
- voter_gap: percentage point gap in 2020 presidential voting: trump-briden

In [32]:

```
def merge_data(covid_updated, filepath):

    # YOUR CODE HERE
    election_data = pd.read_csv(filepath).drop(columns = ["population", "fipscoc
    new_df = pd.merge(election_data, covid_updated,  how='inner', on=['state', '
    # END OF YOUR CODE HERE
    return new_df
```

Run the cell below (no changes necessary) to execute your code above

In [33]:

```
merged = merge_data(covid_updated, 'election2020_by_county.csv')
```

In [34]:

```
merged.head()
```

Out[34]:

	state	county	hispanic	minority	female	unemployed	income	nodegree	bachelor	i
0	Alabama	Autauga County	2.7	24.6	51.124718	5.2	54366	13.8	21.9	
1	Alabama	Baldwin County	4.4	16.9	51.058714	5.5	49626	11.0	28.6	
2	Alabama	Barbour County	4.2	54.3	46.658524	8.9	34971	25.4	13.6	
3	Alabama	Bibb County	2.4	25.4	45.744021	6.6	39546	22.1	10.2	
4	Alabama	Blount County	9.0	12.6	50.595661	5.4	45567	21.9	12.3	

5 rows × 23 columns

In [35]:

```
merged.shape
```

Out[35]:

```
(3012, 23)
```

As mentioned above, the merging requires exact matching between the two DataFrames' `state` and

`county` columns. Thus, some mismatches will occur, yielding our `merged` DataFrame to have fewer rows than `covid_updated` and `election2016_by_county.csv`.

Data Construction / Understanding

Exercise 4.2.1 [1 pt]: Lost Rows

Compared to `covid_updated`, how many rows were lost during this merging process to create `merged`? Running the cell below should print to the screen your answer.

In [36]:

```
# YOUR CODE HERE
covid_updated.shape[0]-merged.shape[0]
# END OF YOUR CODE HERE
```

Out[36]:

```
69
```

Exercise 4.2.2 [2 pts]: Lost Counties

List the county and state of *at least 3* such rows that exist in `covid_updated` but didn't make it into `merged`. Running the cell below should print to the screen your answer.

In [37]:

```
# YOUR CODE HERE
covid_updated[(~covid_updated.state.isin(merged.state))&(~covid_updated.county.i
# END OF YOUR CODE HERE
```

Out[37]:

	county	state
1108	Cameron Parish	Louisiana
1133	Plaquemines Parish	Louisiana
1124	Lafayette Parish	Louisiana

Exercise 4.2.3 [2 pts]: Suggested Fixes

If we needed to be highly thorough and needed comprehensive data coverage, do you have any suggestions on how we could quickly, soundly fix most or all of them? (Write 2-3 sentences.)

NOTE: Please do not actually fix these mismatches; for this Exercise, it's okay that the `merged` DataFrame is smaller than `covid_updated`

To soundly fix most or all of them, I would check data state by state. I would check if the number of counties of each state matches between two DataFrame. If not, I would check carefully the names of the counties of that state and see why there is a mismatch, and then try to fix it. This approach may not be efficient or quick.

This past example demonstrates how easy it is for data to become messy. It also shows the importance of paying close attention to your data in order to understand what you are working with.

Our `case_fatality_rate` column can be viewed as an approximation of how effective and thorough *COVID testing* is for a given county.

Our `# covid deaths` column can be viewed as an extreme indication of how severe *COVID* has impacted a given county.

Our `# covid cases per 100k` column be viewed as middle-ground between the two aforementioned features. That is, it measures the impact of the disease and is influenced by the thoroughness of COVID testing.

Using these three informative features, we can inspect how impacted each county is, while correlating this with other features of each county, such as income-level, health metrics, demographics, etc.

> **Exercise 4.3 [2 pts]: Cleaning the data**
>
> Before we do any further analysis, we first notice that some counties haven't encountered a single COVID death (usually ones with very small populations), thus providing us with little information. Write code in the cell below to update the `merged` DataFrame so that all rows with 0 deaths are removed.

In [38]:

```
# YOUR CODE HERE
merged.drop(merged[merged["# covid deaths"] == 0].index, inplace = True)
# END OF YOUR CODE HERE
```

Running `.describe()` allows us to quickly see summary statistics of our DataFrame

In [39]:

```
1 merged.describe()
```

Out[39]:

	hispanic	minority	female	unemployed	income	nodegree	bac
count	2979.000000	2979.000000	2979.000000	2980.000000	2980.000000	2980.000000	2980.0(
mean	9.269923	22.517422	49.920178	5.487953	47048.403356	14.980302	19.9
std	13.932552	19.802011	2.357197	1.955011	11929.338986	6.745876	8.7
min	0.000000	0.200000	19.166215	1.800000	21658.000000	1.900000	4.4(
25%	2.000000	6.900000	49.465483	4.100000	38910.750000	9.900000	14.0(
50%	4.000000	15.200000	50.384248	5.300000	45186.000000	13.500000	17.9(
75%	9.500000	33.700000	51.068976	6.500000	52509.250000	19.200000	23.6(
max	99.200000	99.400000	56.633907	24.000000	122641.000000	53.300000	72.0(

8 rows × 21 columns

Using the information reported from `.describe()`, we can imagine dividing our DataFrame into 4 separate bins, based on the distribution for any given feature. Specifically, based on a particular feature:

- the 1^{st} bin will be the data that has values between the **min** and **25%**
- the 2^{nd} bin will be the data that has values between **25%** and **50%**
- the 3^{rd} bin will be the data that has values between **50%** and **75%**
- the 4^{th} bin will be the data that has values between **75%** and **max**

> **Exercise 4.4 [3 pts]: Partitioning our data**
>
> Complete the `partition_df()` function, which takes as input:
>
> - DataFrame to work with
> - feature (e.g., obesity) to filter by
> - minimum value
> - maximum value
>
> and outputs:
>
> - a subset of the DataFrame that has values between the passed-in minimum and maximum values (inclusively) for the passed-in feature.
>
> For example, if we called `partition_df(merged, 'obesity', 30, 45)`, it should return a subset of the `merged` DataFrame that has obesity values between 30 and 45 (and including the boundary values of 30 and 45).

In [40]:

```
def partition_df(df, column_name, minv, maxv):
    # YOUR CODE HERE
    return df[(df[column_name] >= minv) & (df[column_name] <= maxv)]
    # END OF YOUR CODE HERE
```

Exercise 4.5: [4 pts] Exploratory Data Analysis

Identify a few features that you're interested in, and inspect if there's any correlation with the COVID data. Specifically, simply run your `partition_df()` function below, many times, each with a different subset of the data -- select a range of values and a particular feature. For example, if I'm interested in **cancer**, I could look at the 4 quartiles (per `.describe()`) and use those ranges of values as I repeatedly execute `partition_df()`. For this exercise, after running the function several times, **write 3-5 sentences about any patterns or correlations you noticed or didn't notice but expected to find.**

In [41]:

```
# YOUR CODE HERE
partition_df(merged, 'income', merged['income'].min(), merged['income'].max())[[
# 0.018416
partition_df(merged, 'income', merged['income'].min(), 38895)[["income","case_fa
# 0.022442
partition_df(merged, 'income', 38895, 45192)[["income","case_fatality_rate", "#
# 0.019062
partition_df(merged, 'income', 45192, 52526)[["income","case_fatality_rate", "#
# 0.017060
partition_df(merged, 'income', 52526, merged['income'].max())[["income","case_fa
# 0.015099
# END OF YOUR CODE HERE

```

Out[41]:

	income	case_fatality_rate	# covid deaths	# covid cases per 100k
count	744.000000	744.000000	744.000000	744.000000
mean	62782.083333	0.014827	413.029570	12513.126235
std	10846.709298	0.007202	1354.125664	3925.955279
min	52526.000000	0.001666	1.000000	3119.250826
25%	55136.750000	0.010051	27.000000	10317.027003
50%	59182.500000	0.013516	95.000000	12647.279597
75%	66893.250000	0.018256	315.000000	14752.308555
max	122641.000000	0.062500	26093.000000	72727.272727

I expect county with high average income would have low case fatality rate. After runing for several times and check the mean value of case fatality rate, I find that county with higher average income has lower case fatality rate on average. One guess is that people living in the counties with higher average income probably have better access to healthcare system, thus lower rate of case fatality.

`.describe()` provides these nice summary statistics over any portion of data that we give it. Instead of iteratively inspecting several subsets of the data, let's actually split our DataFrame into new categories; instead of representing all features by floating point numbers, let's create new *categorical* names for feature(s) based on their numbers. The code below does just this. It creates a new column, `income group` that has 4 possible values, each one corresponding to a quartile of the original `income` values.

Run the cell below.

In [42]:

```
bins = [0, 38000, 45000, 52000, 200000]
names = ['income-group-1', 'income-group-2', 'income-group-3', 'income-group-4']
d = dict(enumerate(names, 1))
merged['income group'] = np.vectorize(d.get)(np.digitize(merged['income'], bins))
merged

```

Out[42]:

	state	county	hispanic	minority	female	unemployed	income	nodegree	bacl
0	Alabama	Autauga County	2.7	24.6	51.124718	5.2	54366	13.8	
1	Alabama	Baldwin County	4.4	16.9	51.058714	5.5	49626	11.0	
2	Alabama	Barbour County	4.2	54.3	46.658524	8.9	34971	25.4	
3	Alabama	Bibb County	2.4	25.4	45.744021	6.6	39546	22.1	
4	Alabama	Blount County	9.0	12.6	50.595661	5.4	45567	21.9	
...	...	...	...	...	...	...	...	...	
3007	Wyoming	Sweetwater County	16.0	20.4	48.388618	4.6	72604	9.5	
3008	Wyoming	Teton County	15.0	18.5	46.913580	3.8	75348	4.3	
3009	Wyoming	Uinta County	9.1	12.3	48.969072	4.9	56800	10.8	
3010	Wyoming	Washakie County	14.2	17.8	50.102993	4.0	50802	10.9	
3011	Wyoming	Weston County	1.4	8.4	47.224954	3.3	55520	11.4	

2980 rows × 24 columns

> **Exercise 4.6 [5 pts]: Aggregate data**
>
> Write code in the cell below to group (and display) the data according to the 4 income groups. Also, while we will still keep the same columns (i.e, features), the values of each should now represent the **average** value of all rows that were subsumed in the making of the aggregate income-group. Your resulting

DataFrame should have just 4 rows (income-group-1, income-group-2, income-group-3, income-group-4). See example in the cell below.

Since every feature (except for `# total cases`, `# covid deaths`, and `population`) was already an average value corresponding to a particular **county**, when we aggregate our data by income groups, we are effectively taking an average of an average. Many counties are being aggregated for each income-group row. This approach isn't as accurate as possible; it would be more accurate if we re-adjusted every value so that it was truly an average that was based on the total **population** of all counties that are subsumed within a given income-group row. That's okay, though. An average of averages will suffice for the purpose of this exercise.

In [43]:

```
# EXAMPLE: If our `merged` DataFrame were
# COUNTY    INCOME GROUP    BACHELOR ... (other columns, too)
#   A             2             50
#   B             1             20
#   C             1             30
#   D             2             70
#   E             3             95

# it should become
# INCOME GROUP    BACHELOR ... (other columns, too)
#   1                25
#   2                60
#   3                95

# YOUR CODE HERE
merged.groupby("income group").agg("mean")
# END OF YOUR CODE HERE
```

Out[43]:

income group	hispanic	minority	female	unemployed	income	nodegree	bachelor	ina
income-group-1	9.138779	32.366718	50.002613	7.272366	33921.184733	21.991145	13.595420	30.
income-group-2	8.869704	20.386946	49.813455	5.718696	41627.992620	15.617712	17.565068	26.
income-group-3	8.861281	17.989833	49.863410	4.850557	48494.079387	12.765460	20.440947	25.
income-group-4	10.156927	20.665365	50.012650	4.356045	62120.351385	10.546977	27.356423	22.

4 rows × 21 columns

Wrapping Up

Exercise 4.7.1 [1 pt]: Conclusions What are your conclusions/finding from this alternative view of the data? (2-4 sentences).

Counties with higher average income have lower case fatality rate. Since the higher income the larger population, it may be not meaningful to look into the number of total covid cases, cases per 100k, covid deaths, or covid deaths per 100k to find any pattern or correlation.

Exercise 4.7.2 [1 pt]: Possible Weaknesses What are some weaknesses from this view of the data? (2-4 sentences).

There might be confounders in the data. If we don't control for the potential confounders (e.g. a common cause of high income and effectiveness of COVID19 testing), there might be confounding that leads to biased estimation of the assocition between income and the effectiveness and thoroughness of COVID19 testing (case fatality rate).

Moving Forward

In this homework assignment, we've focused on gathering, parsing, and exploring data. However, what if we wanted to *predict* some behavior of the data. For example, imagine one is curious how a particular county will respond to COVID. Or, imagine we looked at counties' COVID data on a weekly basis, one could be interested in predicting the upcoming week's behavior.

Alternatively, one could be interested in *inference*, whereby we are more concerned with trying to understand **why** and **how** a system behaves the way it does. We might wish to understand which factors most correlate and cause a certain event to happen. This could give us insights into where certain inequalities persist.

For both *prediction* and *inference*, our computational method of solving such a task is referred to as a model. For the remainder of CS109, we will spend significant focus on various models.

Reflection

As a reminder, this is just **one** of the homework assignments in this course, the point of which is to assess your learning and to provide both you and us with an indication as to how aligned your knowledge and skills are with our learning objectives. To this end, we encourage you to reflect on your progress, strengths, and weaknesses and to make changes, if necessary, to accomplish your goals. Likewise, please reach out to the TFs and teaching staff if you need help. We want everyone to feel comfortable in being honest about these elements, with both herself/himself and us. For these purposes, we will ask you several times throughout the semester to complete an anonymous poll.

In [ ]:

```
1
```

Graded by: Mark Penrod