SI 206 FINAL PROJECT
POPULATION & POVERTY RATE

Jacob Negroni and Umang Bhojani

Github Link: https://github.com/Jacobnegroni/SI-206-UBJN-Final-Project

Initial Goals

Our initial goals were to see if there was any type of correlation between state populations and some other type of metric. We were originally thinking about comparing state populations and the prices of popular restaurants in that state. We wanted to use a TripAdvisor API and another government API site for the state populations.

Goals Achieved

The goals we achieved in our project was to gather the poverty rate of all 50 US states (plus 2 territories) and compare these results with the total population in each of the US states. We extracted data from the first API to gather poverty rates from the years 2013-2019, and from the second API the state total population was extracted. From this information we wanted to see how many people are below the poverty line, the states with the largest poverty rates and if population size and poverty have a correlation.

At the end, the total numbers (not rates) across all states are summed and entered into the results.txt file for 7 years from 2013-2019.

Problems Faced

Problem 1: Storing only 20 items per execution without duplicate entries
We struggled to not include duplicate entries in the database. Initially each time we would run our code, there would be duplicate entries in our database during multiple runs. This would be an issue because it would make it so that the database would become unnecessarily larger than it needed to be with data that was constantly repeated.

Solution: Our solution to this problem was to modify our SQL query syntax in order to remove the duplicate entries. We did this by making sure that the number of additional entries was checked every time before exiting the program. In the SQL query, this removal of duplicate entries was done using the "OR IGNORE" syntax.

Problem 2: Difficulties with displaying visualizations.
Another problem we encountered was that we were finding it very difficult to display our visualizations. Originally, we were using Visual Studio Code as our main coding environment

but we found out that our visualizations were not appearing when we ran our code. There were no errors in the code itself so we knew that it was an environment issue.

Solution: We did some research and an environment named Spyder seemed good for our goals. We decided to change our coding environments specifically for our visualizations. The platform, Spyder, has a separate window where you can view the plots.

Problem 3: Trouble with multiple instances of the same year in our "years" list
One of our functions (find_years) is supposed to take a cursor and select the "year" field from our Population table. The issue we were having was that we were getting multiple instances of the same year in the list (ex. 50 instances of "2018"). We only wanted one instance of each year in the list.

Solution: What we did is that we converted the list of tuples that are extracted from the Population table into a set. A set only allows unique elements to be retained within it so it only allows one instance of each year to be retained. We then converted the set back into a list with the proper number of years included.

Instructions

1) First the file should be unzipped. If you would like, feel free to delete the database and results.txt file
2) Open part1.py and part2p.py in your environment
3) Run part1.py. Every time you run the python script 20 entries are added to the database (population and poverty rate table). The file will be created in the folder, feel free to view the database and see that every time the file is run items are added.
4) Run part1.py until the terminal shows 364/364 items
5) Open up part2.py and in the folder the results.txt file will be created with calculations
6) Next is the visualizations. We used an environment called Spyder in Anaconda. Open up Spyder if you are not using it already.
7) Open visualizations.py and run the script
8) View the graphs by hitting the PLOTS button in the top right corner

Documentation

The modules required for the script include:

- requests
- os
- json
- bs4 (BeautifulSoup)
- sqlite3

1. part1.py

setUpDatabase()
Parameters:
- db_name (str) The name of the database to be created

Returns:
- cur (sql object) Cursor for the database created
- conn (sql object) Connection to the database

Explanation:

The function looks at the current path and sets up a database. A connection is made to the database using the connect() method of the sqlite3 module. A cursor object is also defined for the same connection and is returned in order to be used for the execution of queries later.

getdataAPI()
Parameters:
- url (str)

Returns:
- data (list of dict)

Explanation:

The function takes a url string and uses the get() method from requests module to retrieve the webpage. The content is parsed using a html parser of BeautifulSoup. The text content is already in the json format. The json module's loads() method is used to convert the text into a list of dictionaries that contains the relevant information. This list is then returned by the function.

setUpPopulationTable()
Parameters:
- data_json (list of dict) Dictionary containing the data
- cur (SQL obj) Cursor for the database
- conn(SQL Obj) Connection to the database
- limit (int) limit for the number of new entries added to the table. Default value is 20

Returns:

-None

Explanation:

First, the table Population is created in the project.db database. But the table is created only if a new table does not exist. The fields of the table include

Field name	id	year	state	population

Field type	TEXT	INTEGER	TEXT	INTEGER

The id field is set as the primary column.

If an instance of the Population already exists, the old table is retained with the existing entries. Now, the total number of entries are recorded with the nrows variable. Each entry in the data list is looped through. Each dictionary inthe list contains the keys (ID State, Year, State, Population). The last 4 characters of the value for the key ‘ID State’ and the year are concatenated to create the id which is unique to each entry. It is important that the entries to this field are unique because it is required that the primary field entries are unique in the database. The new entry row for the table is now fetched from the dictionary’s values and inserted into the database. Here, the syntax of SQL is taken advantage of. To prevent duplicate entries to the database during multiple runs of the script, the option ‘OR IGNORE’ in the SQL Query while inserting the new row takes care of skipping the insertion if the entry already exists in the database. Now, the new total number of entries in the database is fetched by selecting all rows of the database using the cur object and counting the length of the output of the fetchall() method that returns a list of tuples. If the new total number of rows equals the nrows+limit number, the for loop is exited by using the break command. The changes made to the table are committed using the conn.commit() function. To return information to the user of the current total entries in the table, the same is fetched using the cursor object and printed as a fraction of the total number of entries expected. This information is to let the user know the final result when all of the required data is updated in the table.

SetUpPovertyTable()
Parameters:
- data_json (list of dict) Dictionary containing the data
- cur (SQL obj) Cursor for the database
- conn(SQL Obj) Connection to the database
- limit (int) limit for the number of new entries added to the table. Default value is 20

Returns:
-None

Explanation:
This function creates a table containing the poverty rate information in the same database. The fields in the table include:

Field name	id	year	state	poverty_rate
Field type	TEXT	INTEGER	TEXT	REAL

The id field is set as the primary column.

The function is identical to the setUpPopulationTable() function. Existing entries are ignored using the 'OR IGNORE' syntax in the query. New entries up to the limit value are updated before the for loop exits. The changes made to the table are committed using the conn object. Relevant information about total entries in the table is also printed before the function ends.

<u>2. part2.py</u>

find_years()

Parameters:

- cur (SQL object) The cursor to the database that is returned from the function setUpDatabase(), same as the one used in Part 1

Returns:

- year (list) A list of the years that was considered in the data in descending order

Explanation:

The function takes a cursor and selects the 'year' field from the Population table. The entries in rows are returned as a list of tuples from which only the first element of the tuple is extracted and converted to a set to retain only unique elements. This set contains only the years that are present in the data. The set is converted to a list and sorted in reverse order and returned by the function

find_total_population()

Parameters:

- cur (SQL Object) Cursor object to the project.db database
- years (list) List of years sorted in descending order

Returns:

- total_population (dict) Dictionary containing year as the key and the total population across all states in the US during that particular year as it's value

Explanation:

The function loops through each year in the list and the SQL Query selects the population field of the row only where the year is the same as the year considered within the loop. Using the fetchall() command with the cursor object, the population of each state for that particular year is summed up in a dummy variable (population_sum) that is initiated at 0 for every loop. The total sum across all states is then appended as a value in the dictionary with the corresponding year as its key

find_total_poverty()

Parameters:

- cur (SQL Object) Cursor object to the project.db database
- years (list) List of years sorted in descending order

Returns:

- total_poverty (dict) Dictionary containing year as the key and the population in poverty for that particular year as it’s value

Explanation:

The function takes loops through each year in the years list. For each year, the tables Population and Poverty are joined using the ‘id’ column. Only the entries which have the same year as the year considered in the loop are filtered and fetched by the cursor object. The columns fetched are population from the Population table and poverty_rate from the Poverty table. The total population in poverty for each state is then calculated by multiplying the poverty rate for that state by the state’s population and rounding it to an integer. All such numbers for each state are summed over and the total population in poverty all through the USA for that particular year is appended to the dictionary as a value with the year as its key. This dictionary is then returned by the function.

write_results()

Parameters:

- years (list) List of years in descending order
- total_population (dict) Dictionary containing year as key and the total population for that year as it’s value
- total_poverty (dict) Dictionary containing year as key and the population under poverty line for that year as it’s value

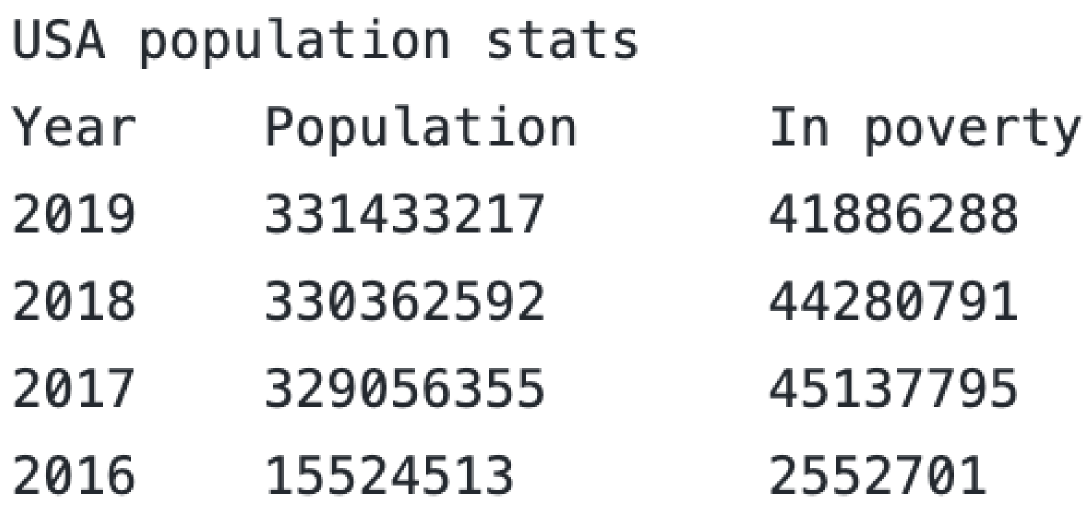
Returns:

- None

Explanation:

The function opens a text file in write format and writes an introductory line and header in the file. The header consists of 3 columns, the year, population and population in poverty. Each year is looped through from the list “years”. The corresponding total population is retrieved from the total_population dictionary and the corresponding population in poverty is retrieved using the total_poverty dictionary. Each entry is written in the text file with proper tab spacing. After the years list is exhausted, the file is closed and the function terminates.

Calculations



```
USA population stats
Year    Population      In poverty
2019    331433217       41886288
2018    330362592       44280791
2017    329056355       45137795
2016    15524513        2552701
```

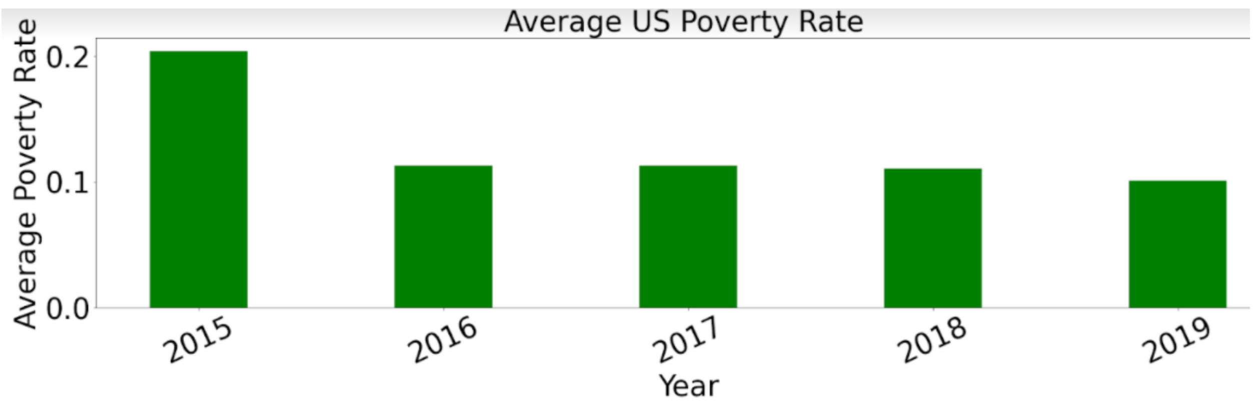
Visualizations

<u>**Visualizations.py**</u>

bar_plotter()
This function simply sets the sizing for various elements including font size, width of bars, and rotation of the x-axis labels.

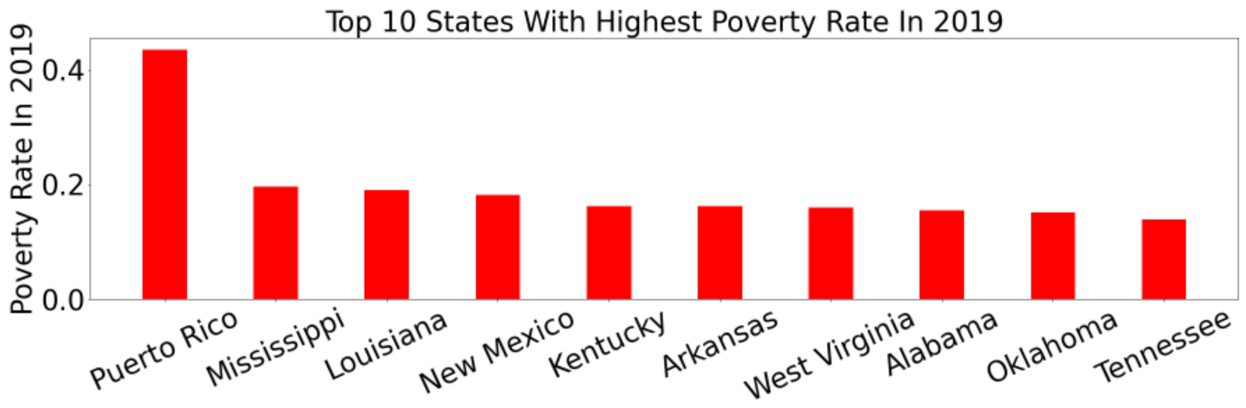
visualizations()
Query selects the poverty rate for the given year for each state. cur.fetchall() contains the poverty rate of each state for each year. Then the average poverty rate as calculated to the avg_pov list is added. The figure creates three axes that can be accessed as ax[0], ax[1], and ax[3]

1)



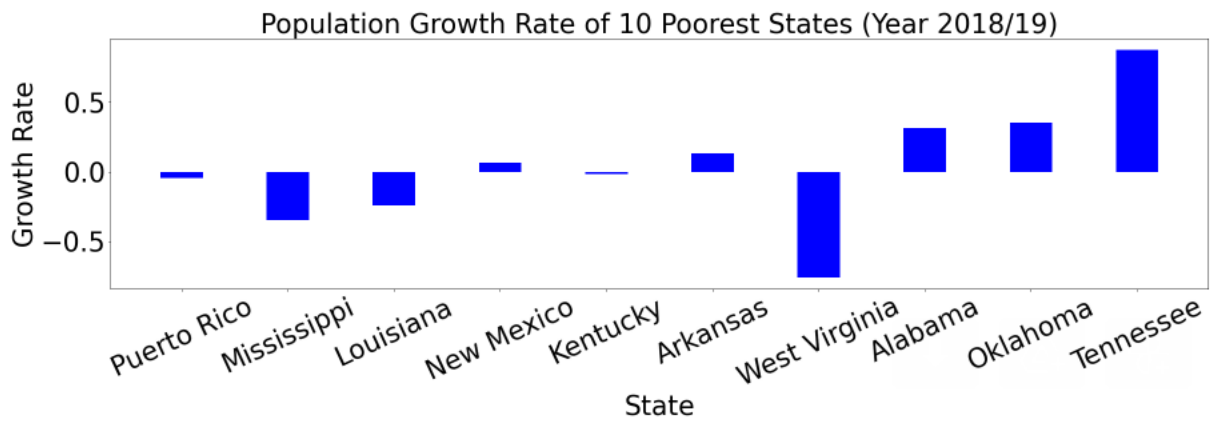
Axis of the graph is first set up and x is set to years, Y-axis is labeled Average Poverty Rate. cur.execute(), query selects state and the corresponding for the year 2019 and lists them in descending order by poverty rate. Unique key value pairs are created where the key is state and value is poverty_rate. Finally, the function creates a list of poverty rates.

2)



The query selects the population for the given year for the given state. pop1 now stores the population for the specific state in that current year (example y1 = 2018). pop2 will store the population for the state for the year (example y2 = 2019). Then growth_rate_dict[state] stores unique key value pairs in growth_rate_dict where the key is the state and the value is the growth rate calculated.

3)



The query selects the population values of the 10 states with the highest poverty rates in 2018 and 2019. The values of 2018 are stored in pop1 while the values of 2019 are stored in pop2. Then growth_rate_dict[state] calculates the growth rate between the years 2018 and 2019 by subtracting the population of 2019 from the population of 2018, dividing that number by the population of 2018 and then finally multiplying by 100 to output the population growth rate.

visualizations('project.db') - called to create all visualizations

Resources Used

Date	**Issue Description**	**Location of Resource**	**Result**
11/17	**Needed to find two API's with relevant and applicable data**	https://datausa.io/api/data?drilldowns=State&measures=Population	First API used to retrieve the population information for each state between years 2013-2019. Data is already in json format and does not need an authentication token to be accessed with the requests module.

11/17	**Needed to find two API's with relevant and applicable data**	https://datausa.io/api/data?drilldowns=State&measures=Poverty%20Rate	Second API used to retrieve the poverty rate information for each state between years 2013-2019. Data is also in json format and does not need an authentication token for access.
11/21	**Reminder regarding SQLite**	https://www.sqlitetutorial.net/	Basic reminder regarding sqlite which helped us
11/21	**Not include duplicate data**	https://stackoverflow.com/a/19343100	This answer was used to restructure the SQL query in order to not include duplicate entries in the database during multiple runs
11/23	**We were unable to figure out how to limit the output when our code was ran to less than 25 elements**	https://www.plus2net.com/sql_tutorial/sql_limit.php	Everytime code was ran, only 20 items were added to the database
12/1	**Formulate graphs**	https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html	Used to help formulate and design bar graphs
12/3	**Our visualizations were blank/were not responding**	towardsdatascience.com/, www.oreilly.com/, matplotlib.com	Did not solve our issue so we used a different environment and made some changes to our code