Leonardo Lindemberg, Wilson Chen SI 206 W24

Final Project: AltEdu

GitHub Repository: Link

1. Original data and APIs we planned to use

We gathered data from two APIs: ScorecardAPI and GeoDB Cities. For the ScorecardAPI we gathered data from US universities from a variety of US states. Some of the data collected from the Scorecars API were: avg SAT, graduation rate, admission rate, and more. From the GeoDB Cities API, we collected details about cities, such as: population, elevation, state, etc.

2. Goals achieved and data gathered

We worked with both APIs that we originally planned since both were able to supply us with good data. For the Scorecard API, the data we gathered was: School name, ID, SAT avg, graduation rate, Admissions Rate, student size, zip, city, and state. For the GeoDB Cities API, we gathered: cities, states, latitude, longitude, elevation, and population. We managed to use the data from both API to create the SQL tables and then the calculations for our visuals later.

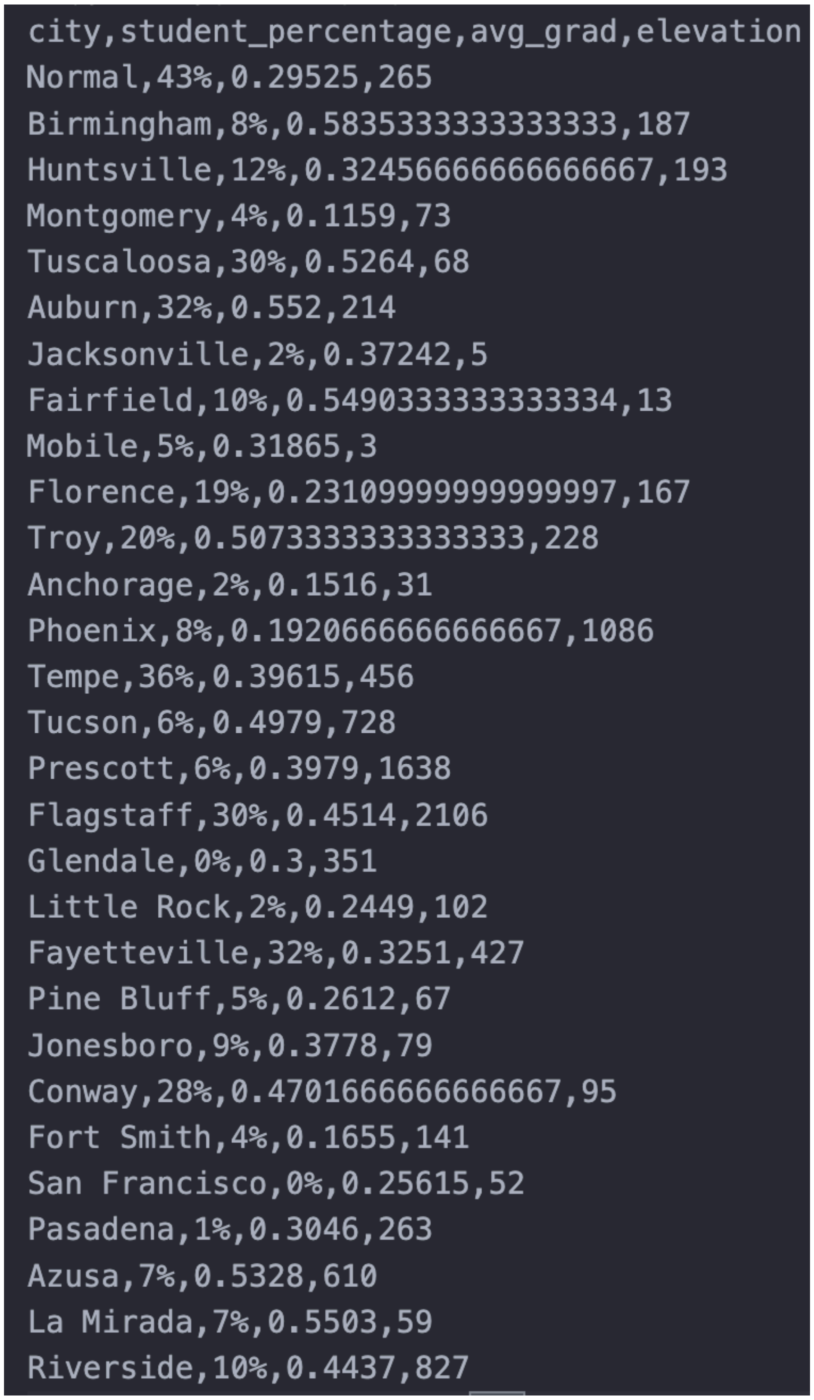
3. Problems we encountered

The scorecard API had poor documentation making it difficult to understand what data was returned and how to set the fields for specific data. The workaround for this was to just request all of the data available and see its formatting. Additionally, some of the schools in the scorecardAPI had missing data for SAT avg, graduation rate, and Admissions rate; therefore, decreasing our data count during calculations and visualization.

When working with the GeoDB API, the primary issue was dealing with the limited number of results returned per request. It only returns 10 results per request and the request rate is also limited to 75/min. The data set for the cities was about 1500 in size (the public API only has cities with population of over 40k) and so the time to request all the information will take over a minute. This restriction in the results returned coupled with the restriction of the assignment, which is to obtain <=25 entries per run of the assignment. To avoid requesting all 1500 results and then checking the database to see where we last “left off”, we keep track of a GraphQL cursor in the database that is returned by the API when we request data. This cursor acts like a bookmark for us so we know where to start fetching from on each run of the program.

4. Calculations using data from database

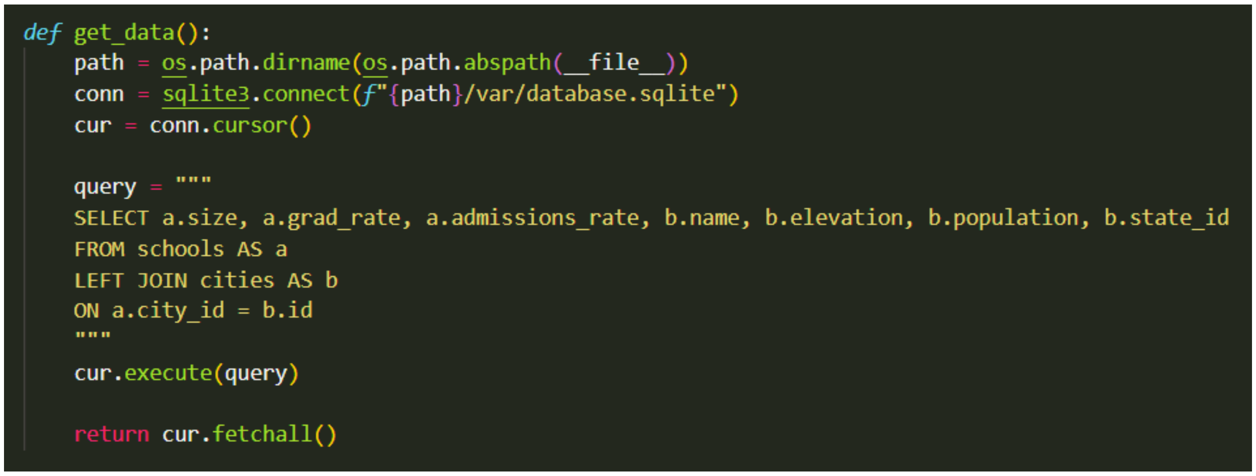
This is the CSV file containing the calculated data from the database. It includes the city name, student percentage, average graduation rate, and elevation.



```
city,student_percentage,avg_grad,elevation
Normal,43%,0.29525,265
Birmingham,8%,0.5835333333333333,187
Huntsville,12%,0.3245666666666667,193
Montgomery,4%,0.1159,73
Tuscaloosa,30%,0.5264,68
Auburn,32%,0.552,214
Jacksonville,2%,0.37242,5
Fairfield,10%,0.5490333333333334,13
Mobile,5%,0.31865,3
Florence,19%,0.23109999999999997,167
Troy,20%,0.5073333333333333,228
Anchorage,2%,0.1516,31
Phoenix,8%,0.1920666666666667,1086
Tempe,36%,0.39615,456
Tucson,6%,0.4979,728
Prescott,6%,0.3979,1638
Flagstaff,30%,0.4514,2106
Glendale,0%,0.3,351
Little Rock,2%,0.2449,102
Fayetteville,32%,0.3251,427
Pine Bluff,5%,0.2612,67
Jonesboro,9%,0.3778,79
Conway,28%,0.4701666666666667,95
Fort Smith,4%,0.1655,141
San Francisco,0%,0.25615,52
Pasadena,1%,0.3046,263
Azusa,7%,0.5328,610
La Mirada,7%,0.5503,59
Riverside,10%,0.4437,827
```

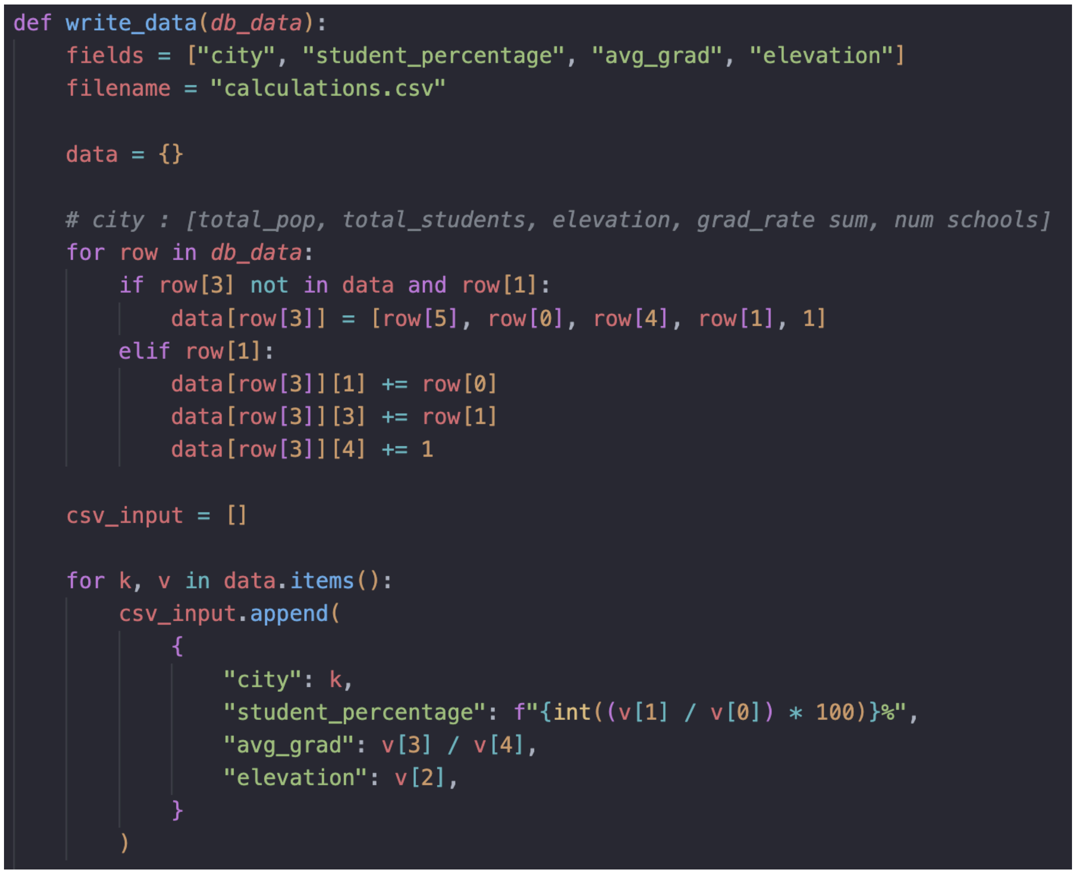
Below is how we calculated this data:

We have a function that connects to our database and executes a query to retrieve the data we need from the appropriate columns. We used LEFT JOIN to combine our data from our normalized tables.



```python
def get_data():
    path = os.path.dirname(os.path.abspath(__file__))
    conn = sqlite3.connect(f"{path}/var/database.sqlite")
    cur = conn.cursor()

    query = """
    SELECT a.size, a.grad_rate, a.admissions_rate, b.name, b.elevation, b.population, b.state_id
    FROM schools AS a
    LEFT JOIN cities AS b
    ON a.city_id = b.id
    """
    cur.execute(query)

    return cur.fetchall()
```

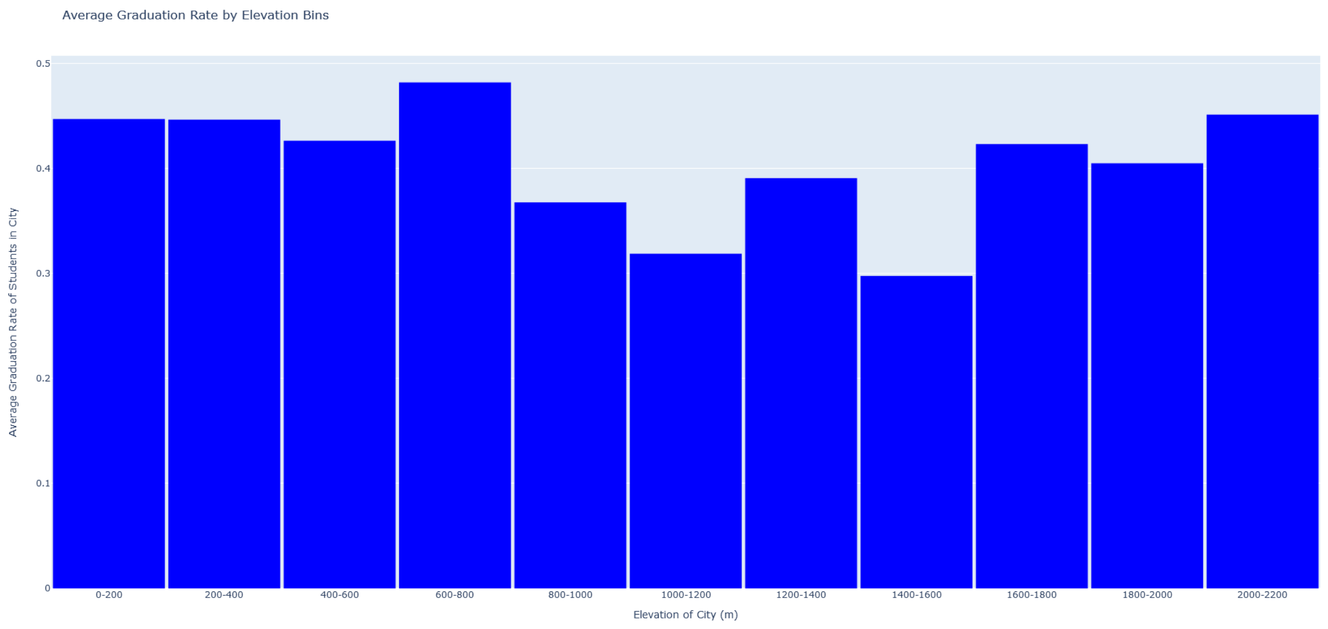
We then have another function that



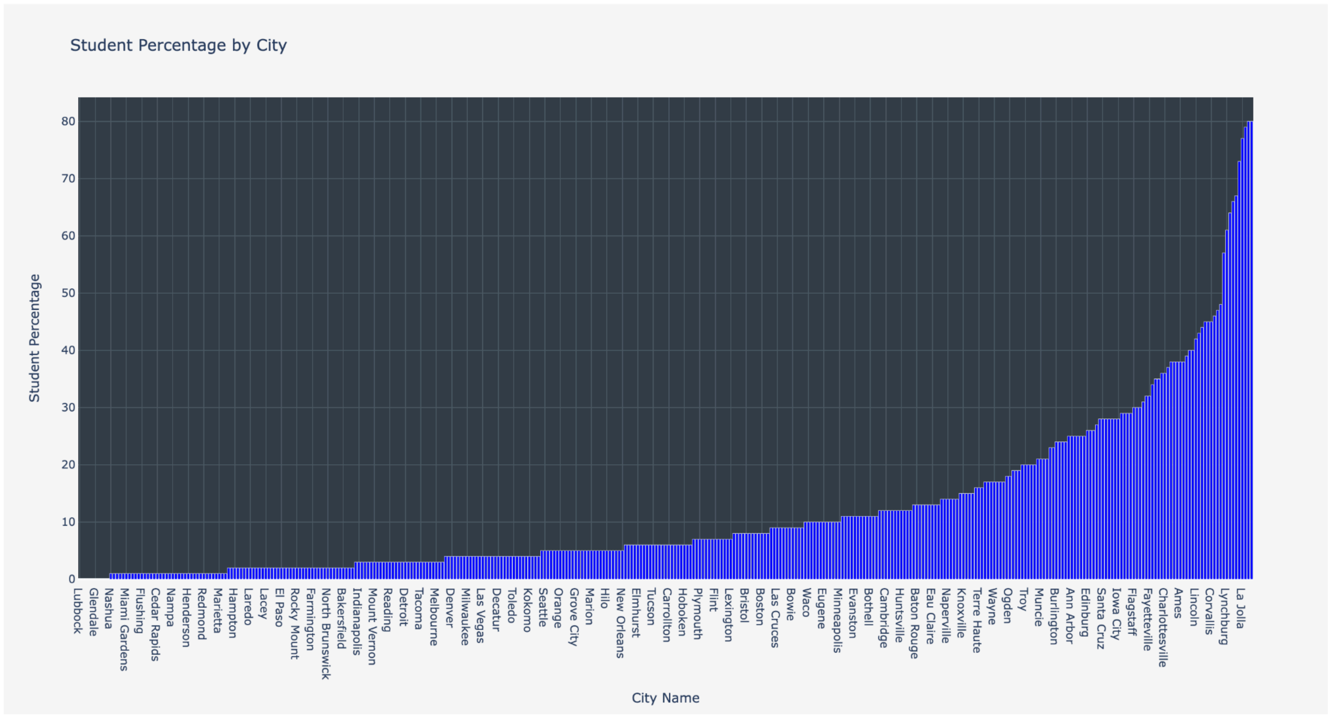
```python
def write_data(db_data):
    fields = ["city", "student_percentage", "avg_grad", "elevation"]
    filename = "calculations.csv"

    data = {}

    # city : [total_pop, total_students, elevation, grad_rate sum, num schools]
    for row in db_data:
        if row[3] not in data and row[1]:
            data[row[3]] = [row[5], row[0], row[4], row[1], 1]
        elif row[1]:
            data[row[3]][1] += row[0]
            data[row[3]][3] += row[1]
            data[row[3]][4] += 1

    csv_input = []

    for k, v in data.items():
        csv_input.append(
            {
                "city": k,
                "student_percentage": f"{int((v[1] / v[0]) * 100)}%",
                "avg_grad": v[3] / v[4],
                "elevation": v[2],
            }
        )
```

5. Visualizations

Average Graduation Rate by Elevation Bins



Student Population Percentage in Cities



6. Running the code

Running ScorecardAPI.py: -> Calls Scorecard API and populates the DB
```
$ python3 scorecardAPI.py “<state-abbr>”
```

Running script to populate school table in DB
```
$ ./bin/insert
```

Running get_us_cities.py and get_us_states.py -> Calls geodb API and populates DB
```
$ python3 get_us_cities.py
$ python3 get_us_states.py
```

Running script to populate cities and states tables in DB
```
$ ./fill_geodb.sh
```

Running calculations.py: -> Calculates data using DB and writes to CSV
```
$ python3 calculations.py
```

Running visualizations.py -> Creates the visuals
```
$ python3 visualizations.py
```

7. Function Documentation

ScorecardAPI.py:

```
def main():
```

Main driver, accepts input arg and calls the other functions in the program.

```
def check_state(input):
```
Input: string, state abbreviation
Output: boolean, if the abbreviation exists or not

Checks the input against a list of state abbr, returns if it was found or not.

```
def api_call(state_in):
```
Input: string, state abbr
Output: Json, json data from the API

Calls the scorecard API and returns the response.

```
def db_setup():
```
Output: tuple(cur, conn), DB connection and cursor

Creates the connection to the DB and makes the cursor.

```
def insert_data(cur, conn, data):
```
Input:
- DB cursor
- DB connection
- Json data

Inserts the given data into the SQLite DB connected through conn.

get_us_regions.py

```
def main():
```

Main driver that runs the geodb_get_regions() function.

```
def connect_to_db():
```
Output: Returns a tuple holding the cursor and connection to the database "var/database.sqlite"

Creates a connection and cursor to the local database.

```
def geodb_get_regions():
```

Requests data of the next 10 regions from the GeoDB Cities API. Inserts results into the states table in the database.

```
def check_cursor():
```
Output: Returns the cursor of the last entry in the states database table. If the states table is empty, return None.

Checks the cursor of the last inserted entry in the database table. Similar to looking for a bookmark to keep track of where we were last reading from.

```
def check_in_db(data):
```

Output: Returns True if retrieved data is already in the database. Returns False if retrieved data is not already in the database. Returns None if retrieved data contains no results.

Checks if the data retrieved by the GeoDB Cities API is already in the database.

```
def update_db(data):
```
Input: data received from GeoDB Cities API

Inserts unique regions into the states table.

get_us_cities.py

```
def main():
```

Main driver that runs the geodb_get_cities() function.

```
def connect_to_db():
```
Output: Returns a tuple holding the cursor and connection to the database “var/database.sqlite”

Creates a connection and cursor to the local database.

```
def geodb_get_cities():
```

Requests data of the next 10 cities from the GeoDB Cities API. Inserts results into the cities table in the database.

```
def check_cursor():
```
Output: Returns the cursor of the last entry in the cities database table. If the cities table is empty, return None.

Checks the cursor of the last inserted entry in the database table. Similar to looking for a bookmark to keep track of where we were last reading from.

```
def check_in_db(data):
```
Output: Returns True if retrieved data is already in the database. Returns False if retrieved data is not already in the database. Returns None if retrieved data contains no results.

Checks if the data retrieved by the GeoDB Cities API is already in the database.

```
def update_db(data):
```
Input: data received from GeoDB Cities API

Inserts unique cities into the cities table.

calculations.py:

```
def main():
```

Main driver that retrieves data from the database, performs the calculations, and writes the data to a CSV file.

```
def get_data():
```
Output: list, all of the data to be used for the calculations

Gets the data used for the calculations from the SQLite DB.

```
def write_data(db_data):
```
Input: list, db_data

Parses through the DB data and calculates the student percentage and average graduation per city. Then writes to a CSV file.

visualization.py:

```
def main():
```

Main driver that reads the calculations CSV file and generates histogram and bar graph visualizations.

```
def histogram(df):
```
Input: DataFrame obj

Organizes the data from the CSV file and creates a line graph with it.

```
def bar_graph(df):
```
Input: DataFrame obj

Organizes the data from the CSV file and creates a bar graph with it.

8. Resources Used

Date	Issue Description	Location of Resource	Result (Did it solve the issue?)
4/16	How to use and request data from the GeoDB Cities API	http://geodb-cities-api.wirefreethought.com/	Provided documentation on how to use the API
4/18	How to do a LEFT JOIN on a DB	https://www.sqlitetutorial.net/sqlite-left-join/	It showed us how to perform a left join properly
4/20	How to use and request data from the ScorecardAPI	https://collegescorecard.ed.gov/data/documentation/	Provided documentation on how to use the API
4/20	How to use GeoDB Cities GraphQL API	https://wirefreethought.github.io/geodb-cities-graphql-schema-docs/query.doc.html	Provided the structure for the query to request data
4/25	How to keep track of offsets when making new requests	https://graphql.org/learn/pagination/	Resolved issue using pagination
4/26	How to access last entry in table in database	https://stackoverflow.com/questions/9902394/how-to-get-last-record-from-sqlite	Resolved through example code
4/29	How to create the visuals using Plotly	https://plotly.com/python-api-reference/	Provided reference for our code
4/30	How to create histograms and modify them using Plotly	https://plotly.com/python/histograms/	Found syntax and parameters to do this successfully