Carson Billig and Zachary Granados

GitHub Repo: https://github.com/zachgranados/206-final

Planned Goals:

Originally, we planned to work with the College Football Data API and the D1 Ticker Attendance Tracker website to draw insights into collegiate football. We had planned on pulling game data (i.e home team, away team, and score) from the College Football API along with media data (i.e where the game was being streamed from) in order to calculate the impact that TV had on teams' win rates. In other words, we originally wanted to see the impact that media played on win rates for FBS teams. As athletes ourselves, we were also interested in seeing the impact that attendance has on win rates. So, we planned on pulling attendance data (Team Name, 2022 Average, and 2023 Average) to investigate the impact that attendance has on win rates.

Achieved Goals:

In reality, we were able to work with both of the resources that we originally planned on working with. We pulled data for the teams and individual games from College Football Data API and pulled attendance data from the D1 Ticker Attendance Tracker. For each game we tracked home team ID, home team score, away team ID and away team score. This allowed us to match teams to their home games and calculate each team's home win rate. In terms of attendance, we were able to store each team's 2023 average attendance, 2022 average attendance, and the percent change over the year. This allowed us to count the number of teams who increased attendance from 2022 to 2023 and the average change in attendance from 2022-2023.

Problems Faced:

The biggest problem that we faced was figuring out how to post requests with the API. In class, we had learned how to use a request URL and append our API key to that URL in order to make requests for data. But, this API used a specific format of cURLs to verify API keys. I had never interacted with cURLs before so it took some research through the API Documentation and using LLMs like Chat GPT to help figure out the best way to format the requests.

Another cumbersome problem occurred when we tried to input our attendance data into the attendance table, but the string names did not always match the string name already stored in the database. This made it hard to match the team_id to the attendance data to avoid duplicate string data. For instance, in the teams table Army West Point was simply labeled as Army because that is how the College Football API referred to the school. But, when we were scraping attendance data from D1 Ticker, Army was referred to as Army West Point, therefore when we tired to run a SQL query on each team to get their team_id before inputting the data into the attendance table, a lot of queries were not returning any matches. In order to fix this without simply adding duplicate strings we had to handle each exception individually. While this may

not have worked for a larger dataset, we knew that the list of teams was finite so we handled each exception on its own using if and elif statements.

Another problem that we encountered was that the attendance data is dependent on the team data because we need the team_id to exist before we can input the attendance data.

Calculations:

Game Data (Home Team Win Rates)
Because our data was somewhat distinct, we created two different calculations. For our game data, we first used a dictionary to collect the school name and the count of home games. We then used SQL to get a list of all home games where the home team outscored the away team (i.e a home win). Using a second SQL query, we used a join to get the school name and total number of home games. Using this information, we were able to calculate the home win rate of each team to see which teams are most successful at home. Screenshots of the results are shown below:

```
Home Team Name : Home Win Percentage
Auburn : 0.43
UAB : 0.67
South Alabama : 0.67
Arkansas : 0.38
Arizona State : 0.25
Arizona : 0.83
San Diego State : 0.43
San José State : 0.67
Stanford : 0.00
California : 0.50
UCLA : 0.67
USC : 0.57
Colorado State : 0.67
Colorado : 0.33
Connecticut : 0.17
Florida State : 1.00
Jacksonville State : 0.83
Florida : 0.57
South Florida : 0.67
Georgia Tech : 0.43
Georgia : 1.00
Hawai'i : 0.57
Iowa State : 0.50
Boise State : 0.83
Northwestern : 0.71
Indiana : 0.43
Notre Dame : 0.86
Kentucky : 0.57
Louisville : 0.86
Western Kentucky : 0.67
LSU : 1.00
Boston College : 0.43
UMass : 0.17
```

calculations.txt

```
Maryland : 0.57
Michigan State : 0.43
Michigan : 1.00
Minnesota : 0.57
Missouri : 0.88
Ole Miss : 1.00
Duke : 0.86
East Carolina : 0.17
NC State : 0.71
North Carolina : 0.88
Wake Forest : 0.50
Nebraska : 0.57
Rutgers : 0.71
New Mexico State : 0.83
New Mexico : 0.33
Syracuse : 0.71
Bowling Green : 0.60
Miami (OH) : 0.80
Ohio State : 1.00
Ohio : 0.83
Oklahoma State : 0.86
Oklahoma : 1.00
Tulsa : 0.33
Oregon State : 0.83
Penn State : 0.86
Temple : 0.43
Pittsburgh : 0.50
Clemson : 0.86
Memphis : 0.67
Vanderbilt : 0.33
Baylor : 0.12
Rice : 0.57
Texas A&M : 0.86
Houston : 0.43
North Texas : 0.50
```

```
Texas : 0.88
BYU : 0.67
Utah : 0.86
James Madison : 0.83
Virginia : 0.33
Virginia Tech : 0.67
Washington : 1.00
Washington State : 0.67
Wisconsin : 0.57
Marshall : 0.83
West Virginia : 0.83
Fresno State : 0.83
Georgia Southern : 0.83
Old Dominion : 0.67
Louisiana : 0.67
Coastal Carolina : 0.67
Texas State : 0.83
Utah State : 0.50
Alabama : 0.88
Mississippi State : 0.50
Army : 0.57
Illinois : 0.43
Air Force : 0.67
Akron : 0.40
Appalachian State : 0.83
Arkansas State : 0.67
Ball State : 0.50
Buffalo : 0.17
UCF : 0.67
Central Michigan : 0.80
Cincinnati : 0.14
Eastern Michigan : 0.83
Florida Atlantic : 0.33
Florida International : 0.33
Georgia State : 0.50
Iowa : 0.75
Kansas : 0.71
Kansas State : 0.86
Kent State : 0.20
Liberty : 1.00
Louisiana Tech : 0.33
Miami : 0.71
Middle Tennessee : 0.67
Navy : 0.67
Charlotte : 0.17
Louisiana Monroe : 0.33
UNLV : 0.71
Nevada : 0.17
Northern Illinois : 0.50
Oregon : 1.00
Purdue : 0.43
Sam Houston State : 0.33
SMU : 1.00
Southern Mississippi : 0.33
South Carolina : 0.71
TCU : 0.57
Tennessee : 0.88
UT San Antonio : 0.83
UTEP : 0.17
Texas Tech : 0.67
Toledo : 0.86
Troy : 0.86
Tulane : 0.75
Western Michigan : 0.60
Wyoming : 1.00
```

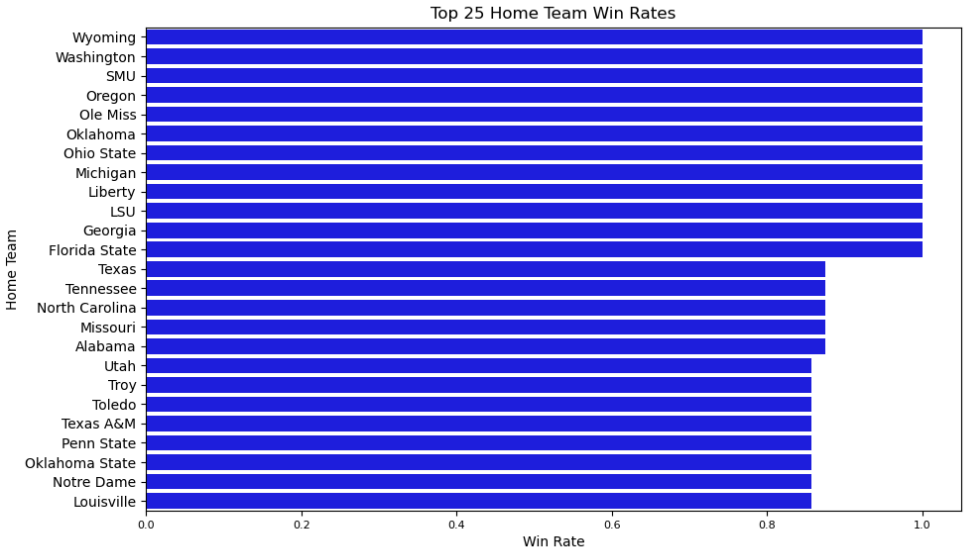
Attendance Data (Average Increase in Attendance)

For the attendance data we calculated the number of teams that increased attendance from the 2022 season to the 2023 season, and also calculated the average change in attendance over all schools in the FBS. To calculate the number of teams that increased attendance, we used a simple SQL query that counted the number of rows where percent change was positive (indicating an increase in attendance over the years). We then took that list of teams that increased attendance and were able to calculate the average increase in attendance that occurred from 2022 to 2023. The results are shown in the screenshots below.

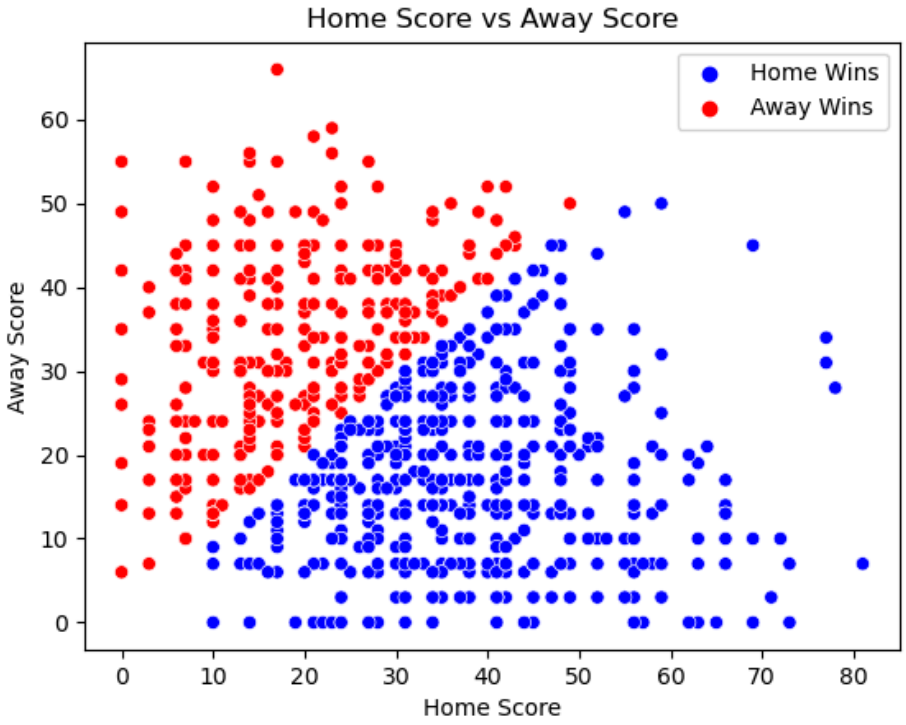
```
95 Teams Increased Their Attendance
The Average Increase in Attendance is 7.326315789473684%
```

Visualizations:

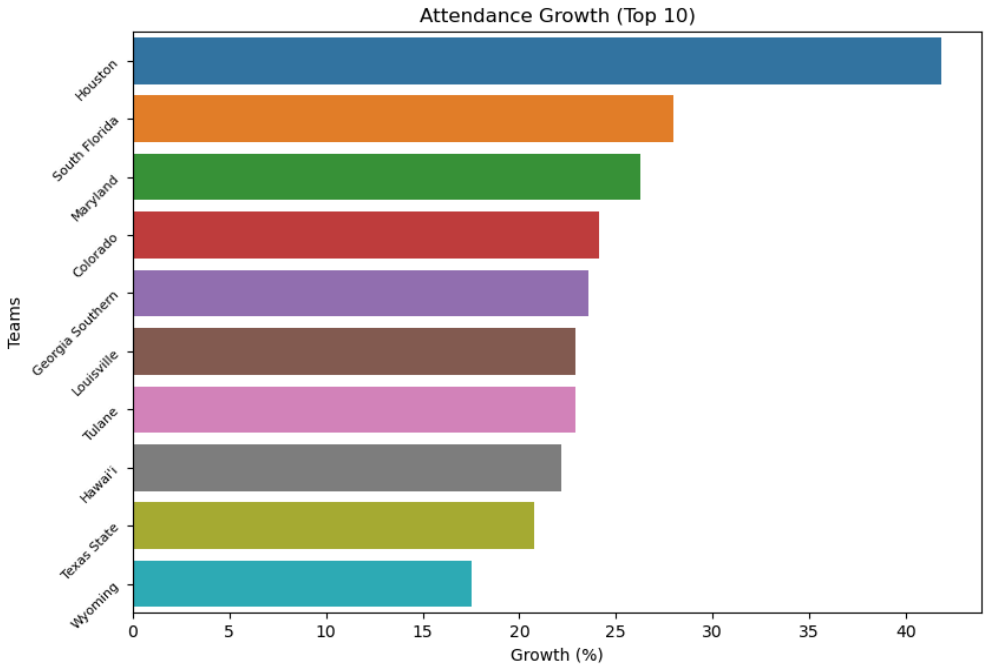
Game Data (Top 25 Home Team Win Rate):



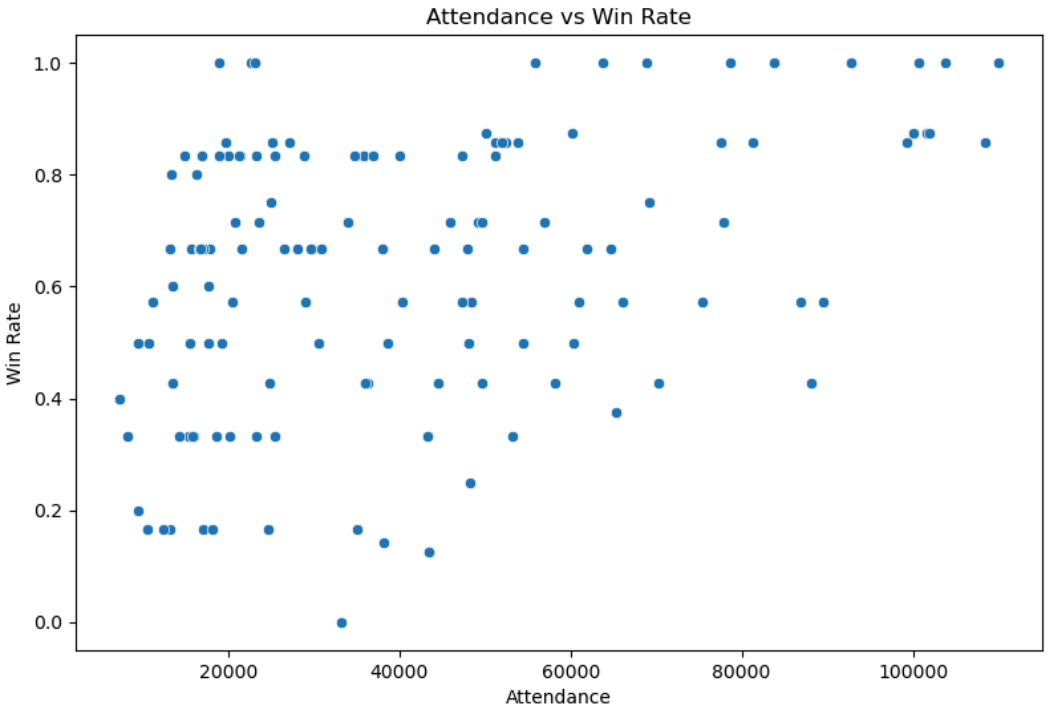
Home Scores versus Away Scores:



Attendance Growth:



Home Team Win Rate vs Attendance:



Instructions For Running:

In order to run our code and set up our database from scratch you will need to run the **main.py** file as a python file. Once this file is executed, you will see a cue printed to the terminal. If there is still data that needs to be inputted into the database, **the cue will ask you to execute the file again.** You will execute the file until you get the cue "Database is Complete" at that point, the calculations and visualizations will be generated automatically and you will be able to view the results of our work.

***Disclaimer:** There are over 850 games that were played in the 2023 FBS season, therefore in order to collect all the data necessary for the calculations you will need to execute the main.py file roughly 35 times.

Documentation:

def setup_db(db_name):

- Establishes a connection to a SQLite database file located in the same directory

Parameters:
- db_name (str): The name of the SQLite database file to connect to.

Returns:
- cur (sqlite3.Cursor): Cursor object used to execute SQL queries.
- conn (sqlite3.Connection): Connection object representing the database connection.

Example:

cur, conn = setup_db("cfb.db")

- This function connects to a SQLite database file specified by `db_name` (string) located in the same directory as the script. If the file does not exist, it creates a new database file with the provided name. It returns a tuple containing a cursor object (`cur`) and a connection object (`conn`) for interacting with the database.

def create_team_table(cur, conn):

- Creates a SQLite table named 'teams' if it does not already exist in the database.

Parameters:
- cur (sqlite3.Cursor): Cursor object used to execute SQL queries.
- conn (sqlite3.Connection): Connection object representing the database connection.

Returns:
None

Example:
```
create_teams_table(cur, conn)
```

This function executes an SQL query to create a table named 'teams' in the connected SQLite database if it does not already exist. The table contains two columns:
- team_id: An INTEGER column representing the unique identifier for each team (PRIMARY KEY).
- school: A TEXT column representing the name of the school (UNIQUE).

The `cur` parameter should be a cursor object obtained from a database connection (`conn`).
After executing the query, changes are committed to the database using `conn.commit()`.

def create_attendance_table(cur, conn)

Parameters:
- cur (sqlite3.Cursor): Cursor object used to execute SQL queries.
- conn (sqlite3.Connection): Connection object representing the database connection.

Returns:
None

Example:
```
create_attendance_table(cur, conn)

```
This function creates a table named “attendance” in a SQLite database if it doesn’t already exist, if it finds that the table does exist then the function does nothing. The table has four columns:

- Team_id (TEXT)L represents the unique identifier for each team
- Average2023 (INTEGER): Represents the average attendance for the year 2023
- Average2022 (INTEGER): Represents the average attendance for the year 2022
- Percent_change (REAL) : Represents the percentage change in attendance from the years 2022 to 2023. This column stores real numbers (floating-point values)

The ‘CREATE TABLE IF NOT EXISTS’ statement ensures that the table creation operation is performed only if the “attendance” table does not already exist in the database. \

The `cur` parameter should be a cursor object obtained from a database connection (`conn`).
After executing the query, changes are committed to the database using `conn.commit()`.

def get_api_key(filename):

Parameters:
- filename: represents the path to the file containing the API key

Returns:
- Returns the content of the file (‘key’ variable) which is assumed to be the API key

Example:
filename = ‘api_key.txt’
api_key = get_api_key(filename)
print(“API Key:”, api_key)

This function facilitates retrieving an API key stored in a file by reading the contents of the file and returning it as a string.

def input_25_fbs_team_data(cur, conn, year):

Parameters:
- cur (sqlite3.Cursor): Cursor object used to execute SQL queries.
- conn (sqlite3.Connection): Connection object representing the database connection.
- year: the year for which data is requested

Returns:
Returns the count of teams added to the database (up to 25)

Example:
```
year = 2023
input_25_fbs_team_data(cur, conn, year)
cur.execute("SELECT * FROM fbs_teams WHERE year = ?", (year,))
teams = cur.fetchall()
for team in teams:
        print(team)

conn.close()
```

This function effectively retrieves FBS team data for a specific year from an API and adds it to a database, ensuring that duplicate entries are not added. It limits the addition of teams to 25 at a time and returns the count of teams added.

def home_team_wins(cur, conn):

Parameters:
- cur (sqlite3.Cursor): Cursor object used to execute SQL queries.
- conn (sqlite3.Connection): Connection object representing the database connection.

Return:
Returns a dictionary ('team_win_rate') containing the win rate for each team.

Example:
```
if __name__ == "__main__"

conn = sqlite3.connect("cfb.db")
cur = conn.cursor()

home_wins = home_team_wins(cur, conn)

print("Home teams that won their games:")
for team in home_wins:
        print(team[0]) #Print the school name
```

conn.close()

This function effectively calculates and returns the win rate for each team based on the number of home wins and the total number of home games played.

def write_data(data, file):

Parameters:

- data: a dictionary containing team names as keys and their corresponding win percentages as values
- file: a file object to which the data will be written

Return:
None

Example:

```
Sample_data = {
“Team a” : 0.75,
“Team b”: 0.65,
}

With open(“calculations.txt”, “w”) as f:
        write_data(sample_data, f)
```

This function is useful for writing data to a file in a specific format, making it suitable for storing or exporting data.

def scrape_data():

Parameters:
None

Return:
Returns the “school_data” dictionary containing the attendance data for each school

Example:

```
attendance_data = scrape_data()
If attendance_data:
        print(“attendance data scraped successfully!”)
        print(“sample data:”)
For school, data in attendance_data.items():
        print(f”school:{school}, 2023 Average: {data[‘2023_average’]}, 2022 Average:
{data[‘2022_average’]}. Percentage Change: {data[‘percentage_change’]}”)
```

Else:
    print("failed to scrape attendance data.")

This function effectively scrapes attendance data for FBS schools from a webpage and returns it in a structured format

def insert_25_attendance(school_data, cur, conn):

Parameters:
- school_data: A dictionary containing attendance data for schools, with school names as keys
- cur: cursor object for executing SQL commands
- conn: connection object to the SQLite database

Return:
Returns the count of rows added to the database (up to 25)

Example:

conn = sqlite3.connect("cfb.db")
cur = conn.cursor()

rows_added = insert_25_attendance(school_data, cur, conn)
print(f"Attendance data for {rows_added} schools inserted successfully."

conn.close()

This function effectively inserts attendance data for up to 25 schools into the database, ensuring that duplicate entries are avoided. It also handles exceptions for formatting school names and commits the changes to the database once all insertions are completed.

def create_barGraph(attendance_data):

Parameters:
attendance_data: a list of tuples where each tuple contains a team name and its corresponding attendance growth value.

Return:
None

Example:

create_barGraph(attendance_data)

This function effectively provides a visual representation of attendance growth for the top 10 teams

def extra_credit(cur, conn, team_win_rate):

Parameters:

- cur: cursor object for executing SQL commands
- conn: connection object to the SQLite database
- Team_win_rate: dictionary containing win rates for different teams

Return: None

Example:

```
example_team_win_rate = {
   "Team A": 0.65,
   "Team B": 0.75,
   "Team C": 0.60,
   "Team D": 0.70,
   "Team E": 0.80
}

conn = sqlite3.connect("cfb.db")
cur = conn.cursor()

extra_credit(cur, conn, example_team_win_rate)
```

def write_file(game_data, attendance_data):

Parameters:

game_data: a dictionary containing team names as keys and the corresponding home win rate as values

attendance_data: a list of tuples where each tuple contains a team name and its corresponding attendance growth value.

Return: None

Example:

write_file(game_results, attendance_results)

This function takes the given data, checks if the file has any data already, if so, the function returns, if not the data is written into a text file.

def num_of_increases(cur, conn):
Parameters:

- cur: cursor object for executing SQL commands
- conn: connection object to the SQLite database

Return: List of tuples with team names and the corresponding change in attendance

Example:

num_of_increases(cur, conn)

This function uses the cursor object to execute a SQL command that selects all teams and the corresponding attendance change where the 2022 Average was less than the 2023 average. This data is then used to write to files and/or run visualizations

def write_calcs(data, file):

Parameters:

- data: list of tuples with team names and the corresponding change in attendance
- file: a file object to which the data will be written

Return: None

Example:
write_calcs(attendance_data, file)

This function takes the data created from the attendance calculations and writes it to a given file. It is useful for writing data to a file in a specific format, making it suitable for storing or exporting data.

def create_barGraph(data):
Parameters:

- data: a dictionary containing team names and their corresponding win rate at home

Return: None

Example:
create_barGraph(game_results)

This function takes a dictionary of team names and win rate, sorts the teams by their win rate, takes the top 25 teams and their win rates and displays them in a blue bar graph.

def create_extra_credit(cur, conn):

Parameters:
- cur: cursor object for executing SQL commands
- conn: connection object to the SQLite database

Return: None

Example
create_extra_credit(cur, conn)

This function uses the cursor object to execute a SQL command that selects all home scores and away scores. This function then uses the list of tuples created (i.e (home_score, away_score) to create a scatter plot with home score on the x-axis and away_scores on the y axis. For games where home team wins the dot is blue and when the away team wins the dot is red

Resources Used:

Date	Issue	Resource Used	Issue Resolved?
4/15/2024	I was struggling to create a url because the API documentation had everything in cURL format and I could not figure out how to create the necessary parameters to make a request to the API. Using the cURL given in the documentation I was able to use Chat GPT to help me format my request using the Python request library	API Documentation Chat GPT	Yes
4/16/2024	I was struggling to figure out how to import a file to use because I was trying to clean up the code and create a setup file / a file that would actually put the data into the database.	Import Tutorial	Yes
4/25/2024	I was struggling to figure out how to	ChatGPT	Yes

	create a visualization without overlapping the team names. I used the Seaborn documentation and ChatGPT to learn how to structure the plot so that every team name could be read.	Seaborn Documentation	

Date	Issue	Resource Used	Issue Resolved?
4/22/2024	I was struggling on creating the database from VSCode to DB Browser and having the data be displayed in the way I wanted it. I was initially indexing from the second, third and fourth columns when I only needed the third, fourth and fifth.	Chat GPT SQL Tutorial - Essential SQL For The Beginners	Yes
4/23/2024	I was also running into duplicate data being added into the database every time the file was run. Being able to reference this website helped me resolve this problem.	SQL Tutorial - Essential SQL For The Beginners	Yes