1. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)

Our initial goal was to analyze and compare user ratings between manga and their anime adaptations using data from the Mangadex API and AniDB API. Specifically, we aimed to gather user ratings for manga and their respective anime adaptations and store this data in a database. From the collected data, we planned to calculate the average ratings for each and compare them to determine whether the anime adaptation improved on, maintained, or worsened the perception of the original manga. For visualizations, we planned to use Matplotlib to create charts representing these comparisons, categorized by genre or studio.

2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

While our initial project plan focused on manga and anime ratings, we pivoted to analyzing weather data due to challenges with accessing reliable anime and manga APIs. We successfully used the Weatherbit and Visual Crossing APIs to gather weather data for various cities in the United States and Europe. Specifically, we collected average, maximum, and minimum temperatures for each city and stored this data for analysis. Using this data, we achieved our goal of visualizing trends, creating heatmaps with Seaborn to compare average temperatures between regions. Although our project goals shifted, we still adhered to our objective of integrating API data into meaningful visualizations.

3. The problems that you faced (10 points)

We faced several challenges during the project. Initially, we struggled to use the Mangadex and AniDB APIs due to difficulties in obtaining API access and parsing the data efficiently. These issues led us to change the scope of our project to focus on weather data. Once we pivoted, we encountered new challenges. For example, we had to navigate API rate limits and discrepancies in how data was formatted between the Weatherbit and Visual Crossing APIs. Additionally, visualizing the data required overcoming errors in the data pivot process, especially when formatting it for a Seaborn heatmap. Debugging these issues improved our understanding of data wrangling and visualization tools.

4. The calculations from the data in the database (i.e. a screen shot) (10 points)

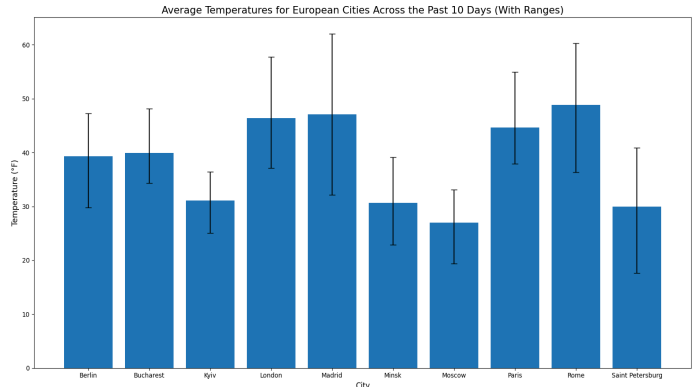
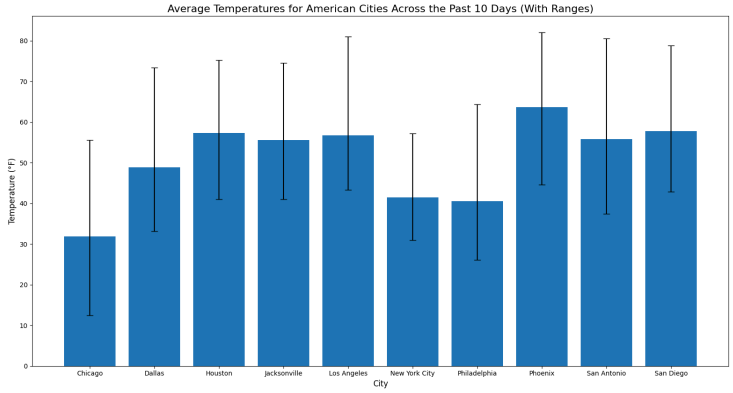
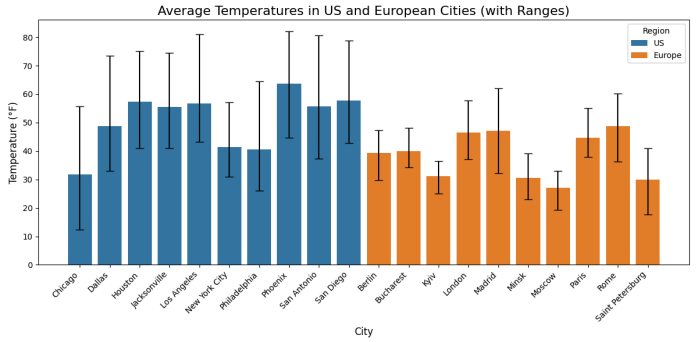
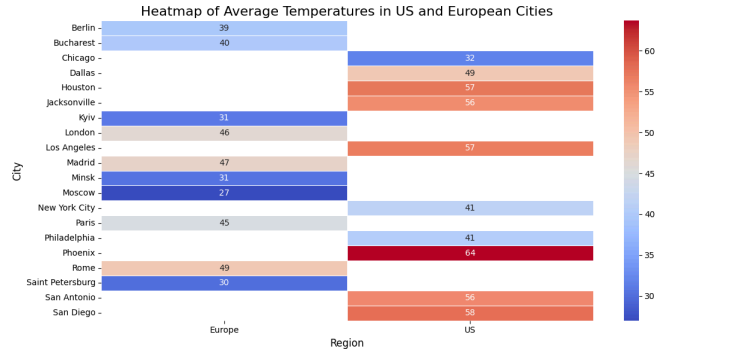
european_weather_summary.txt

```
City    Region  Avg Temp    Highest Temp    Lowest Temp
Berlin  Germany 39.339999999999996  47.3    29.8
Bucharest   Romania 39.88   48.1    34.3
Kyiv    Ukraine 31.110000000000003  36.4    25.1
London  UK  46.43   57.8    37.1
Madrid  Spain   47.05   62.0    32.1
Minsk   Belarus 30.619999999999997  39.1    22.9
Moscow  Russia  26.98   33.1    19.4
Paris   France  44.62   55.0    37.9
Rome    Italy   48.85   60.3    36.3
Saint Petersburg    Russia  29.99   40.9    17.6

```

us_weather_summary.txt

```
City    Region  Avg Temp    Highest Temp    Lowest Temp
Chicago IL  31.830000000000002  55.6    12.4
Dallas  TX  48.839999999999996  73.4    33.1
Houston TX  57.33   75.2    41.0
Jacksonville    FL  55.58   74.5    41.0
Los Angeles CA  56.73   81.0    43.3
New York City   NY  41.45   57.2    30.9
Philadelphia    PA  40.54   64.4    26.1
Phoenix AZ  63.69   82.0    44.6
San Antonio TX  55.779999999999994  80.6    37.4
San Diego   CA  57.8    78.8    42.8

```

5. The visualization that you created (i.e. screen shot or image file) (10 points)

Average European City Temperatures	Average US City Temperatures
	
Average US and European City Temperatures (Combined)	**Heatmap of Average US and European City Temperatures**
	

6. Instructions for running your code (10 points)

- **Run european_weather.py** and **us_weather.py** 4 times each to make the API requests for gathering data, creating the database tables, and inputting the necessary information into each designated table within the database
- **Run calculations.py** to calculate the average temperatures for each region and then write them into a file so we can use it to make visualizations later
- **Run create_plot.py** to create a bar chart for each region individually, **Visualization.py** to create a combined bar chart for both regions, and **SecondVisualization.py** to create a heatmap

7. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)

us_weather.py

- *load_data(cities)*
  - load_data takes one argument: a list of dictionaries
  - Each dictionary has two entries:
    - "city", which represents the name of the city
    - "state", which represents the state in which the city is found
  - load_data will make an API request using the list of dictionaries as locations, and will return the weather data for the past 10 days (at the time, set to be the period between 12/02/2024 and 12/12/2024) of each city in the list.
- *create_tables(conn)*
  - create_tables takes one argument: the database connection object
  - create_tables will create two tables as long as they do not exist:
    - "USCities", which will have the following columns:
      - cityid (datatype: integer and primary key)
      - city (datatype: text)
      - state (datatype: text)
    - "USWeatherData", which will have the following columns:
      - weatherid (datatype: integer and primary key)
      - cityid (datatype: integer)
      - date (datatype: date)
      - temp (datatype: real)
      - max_temp (datatype: real)
      - min_temp (datatype: real)
      - precipitation (datatype: real)
    - Both tables will ensure that there are no duplicate entries using the UNIQUE keyword.
  - create_tables will then commit these changes to the database and return nothing.
- *input_sql_data(conn, data)*
  - input_sql_data takes two arguments: the database connection object and a list of dictionaries representing a city and its past ten days of weather data
  - input_sql_data will take the weather data created by the load_data function and insert it into the database.
  - The final result will have each of the ten cities and the states in which they are found in the USCities table, and the dates, temperatures, and precipitation in the USWeatherData table. Both tables will have the cityid column filled so that we can use it to join later when calculating values.
  - input_sql_data will then commit these changes to the database and return nothing
- *main()*
  - main takes no arguments

    - main will initialize a list of dictionaries that have the top ten cities in the US by population
    - main will then use all of the functions to initialize the database

EuropeanWeather.py

- *load_data(cities)*
    - Description: This function loads weather data from the Visual Crossing API for a given list of cities and returns it in a structured format.
    - Input: cities (list): A list of dictionaries where each dictionary contains a "city" and "country" key, representing the locations to fetch weather data for.
    - Output: Returns a list of dictionaries where each dictionary represents a city and contains the city, country, and the fetched weather data. The weather data is stored in a nested "data" key, which holds the API response.
- *create_tables(conn)*
    - Description: This function creates the necessary tables (EuropeanCities and EuropeanWeatherData) in the SQLite database to store city and weather data.
    - Input: conn (sqlite3.Connection): A connection object to the SQLite database.
    - Output: No output is returned. The function performs actions to create the tables in the database.
- *input_sql_data(conn, data)*
    - Description: This function inserts weather data into the SQLite database. It inserts the city data into the EuropeanCities table and the daily weather data into the EuropeanWeatherData table. It limits the number of records inserted per run, based on MAX_ITEMS_PER_RUN.
    - Input: conn (sqlite3.Connection): A connection object to the SQLite database. data (list): A list of dictionaries, where each dictionary contains city weather data, including the city name, country, and daily weather data (temperature, description, etc.).
    - Output: No output is returned. The function inserts data into the database and prints a summary of how many items were inserted.
- *main()*
    - Description: This is the main function that runs the overall program. It defines a list of locations, loads the weather data using load_data, creates the necessary tables using create_tables, and then inserts the weather data into the database using input_sql_data.
    - Input: No direct input. The function defines a list of cities to fetch data for.
    - Output: No output is returned.

Calculations.py

- *write_data_to_file(query, filename)*
    - Description: This function executes a SQL query, retrieves the results, and writes them to a text file. It formats the data into a readable table with tab-separated values for each row, including a header row.
    - Input: query (str): A SQL query string to be executed.

    - Output: filename (str): The name of the text file where the query results will be written.
- *main()*
    - Description: This is the main function that orchestrates the execution of the program. It calls the write_data_to_file function twice, once for European cities and once for US cities. Each call executes a specific SQL query to gather weather data averages and saves it to a text file.
    - Input: No direct input. It triggers the write_data_to_file function twice with predefined SQL queries for European and US cities.
    - Output: No output is returned. The function initiates the writing of weather data to two separate files (european_weather_summary.txt and us_weather_summary.txt).

Visualize.py

- *main()*
    - Description: The main function processes weather data for US and European cities from two text files, merges them into one dataset, and then creates a bar chart visualization comparing the average temperatures of cities from both regions. The chart includes error bars for the minimum and maximum temperatures, providing a range for each city's average temperature.
    - Input: No direct input. The function reads data from two files (us_weather_summary.txt and european_weather_summary.txt), processes them, and generates a visualization.
    - Output: A bar chart comparing the average temperatures of US and European cities, including error bars indicating the temperature range (minimum to maximum), saved as an image file (comparative_temperatures_bar_chart.png).

SecondVisualize.py

- *main()*
    - Description: The main function processes weather data for US and European cities from two text files, combines the data into one dataset, and creates a heatmap visualization comparing the average temperatures of cities from both regions. The heatmap shows the average temperature for each city in both regions, using color to represent temperature values.
    - Input: No direct input. The function reads data from two files (us_weather_summary.txt and european_weather_summary.txt), processes them, and generates a heatmap visualization.
    - Output: A heatmap image comparing the average temperatures for cities from both regions, saved as an image file (temperature_heatmap.png).

create_plot.py

- *get_data_from_file(filename)*
    - get_data_from_file takes one argument: a string called filename which represents the name of the file

    - get_data_from_file adds every line from a file and processes it by both removing the header and splitting the data so it can be used easier.
    - get_data_from_file then returns the list of lines from the file.
- *create_bar_chart_averages(data, region)*
    - create_bar_chart_averages takes two arguments: a list of data from the file, and a string representing the region which the data represents
    - create_bar_chart_averages creates a bar chart with the average temperature as the main data, and the min and max temperatures represented through error bars.
    - create_bar_chart_averages finishes by naming the bar chart based on the region string and adding axis labels, and then returns nothing.
- *main()*
    - main takes in no arguments
    - main uses the get_data_from_file with european_weather_summary.txt and us_weather_summary.txt to gather the data, and then creates bar charts with each using create_bar_charts_averages.
    - main shows the bar charts and returns nothing.

8. You must also clearly document all resources you used. The documentation should be of the following form (20 points)

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
12/11/2024	Figuring out how to use matplotlib	https://www.geeksforgeeks.org/bar-plot-in-matplotlib/	Yes
12/11/2024	Add error bars for min and max temperatures on the bar charts	https://chatgpt.com/	Yes
12/11/2024	Have two separate bar charts appear at the same time	https://stackoverflow.com/questions/6916978/how-do-i-create-a-second-new-plot-then-later-plot-on-the-old-one	Yes
12/11/2024	SQL help/debugging	https://chatgpt.com/	Yes
12/11/2024	Remembering how to read from files	runestone.academy	Yes
12/11/2024	Needed a way to create a heatmap to visualize temperature data for US and European cities.	https://seaborn.pydata.org/generated/seaborn.heatmap.html	Yes
12/11/2024	Needed a color scheme to visually represent temperature variations.	https://seaborn.pydata.org/tutorial/color_palettes.html	Yes
12/11/2024	Needed to read output of calculations.py files containing weather data and combine the datasets from US and Europe.	https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html	Yes
12/11/2024	Required a way to combine data from the US and European	https://pandas.pydata.org/pandas-docs/stable/reference/api/pan	Yes

	datasets into one unified dataset.	das.concat.html	
12/11/2024	Needed to pivot the data in a format suitable for generating a heatmap (cities as rows, regions as columns).	https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot.html	Yes

Notes/Changes

- We changed the datatype of the “date” column in USWeatherData in us_weather.py and the “datetime” column in WeatherData in european_weather.py from TEXT to DATE to account for duplicate string data following our grading session.