SI 206 Final Project Report

Github Link: https://github.com/ProgrammerNick/SI206-Final-Project.git

Goal:

Create a travel app to compare travel data between two different cities using TripAdvisor API to get restaurants, Ticketmaster API to get events and API to get weather.
Display and compare weather data, restaurants data and events data for the two cities for the desired travel date.
We wanted to create an app that allowed a user to be more informed about the events, the weather, and the restaurant offerings in a location that they were interested in traveling to. The goal was to create an app that managed data and created visualizations to aid users in making better travel plans, and being informed about the places they wanted to visit.

Achieved Goal:

Created a travel app that displays valuable tourist information for one city at a time using TripAdvisor API to get restaurants, Ticketmaster API to get events and API to get weather.
Moreover, we could display two additional visualizations from our data.
Display average temperature and humidity data over a 5 day forecast within 15 days from current date
Displays average rating, average number of user reviews, highest rated restaurant and popularity for different cuisines in the city using data such as name, rating, number of user reviews and cuisine.
Displays the number of events in a given city over a specified period of time, and the mix of venues where these events were taking place.

Fixes after Presentation:

For restaurant data, it threw an error when the database was deleted or renamed and it had to start from scratch. This was because I had a line of code trying to access the Restaurants table in the database before the table was actually created in the database and this led to an error. So, I debugged it and it worked.

When it came to gathering the TicketMaster data, I initially used a for loop with the upper bound value being 25 to limit the amount of rows being inserted into the table to 25 for each API call. After receiving feedback, I implemented a counter to increment each time a row was added to the table and limited this counter to 25.

The event code had also run into an error when the old database was deleted and created again. This was a small error with the date being an integer, I fixed this issue by making sure I casted the date values to a string.

Problems faced:

Issues with merging files through github
Weather API's have a 15 day forecast limit from the current date, so it was only possible to show weather data within a 15 day period
TripAdvisor API(used to get restaurants) had limited API calls allowed so had to change API keys a couple of times
Initially we wanted to use a flight price API to help users identify areas that they want to fly to. Unfortunately after realizing the Google Flight Prices API was discontinued we struggled to find another API that was affordable to use for this project. We pivoted to getting events data for different locations, and focused less on the price and more on the overall experience of the destination for the user.

Instructions for running code:

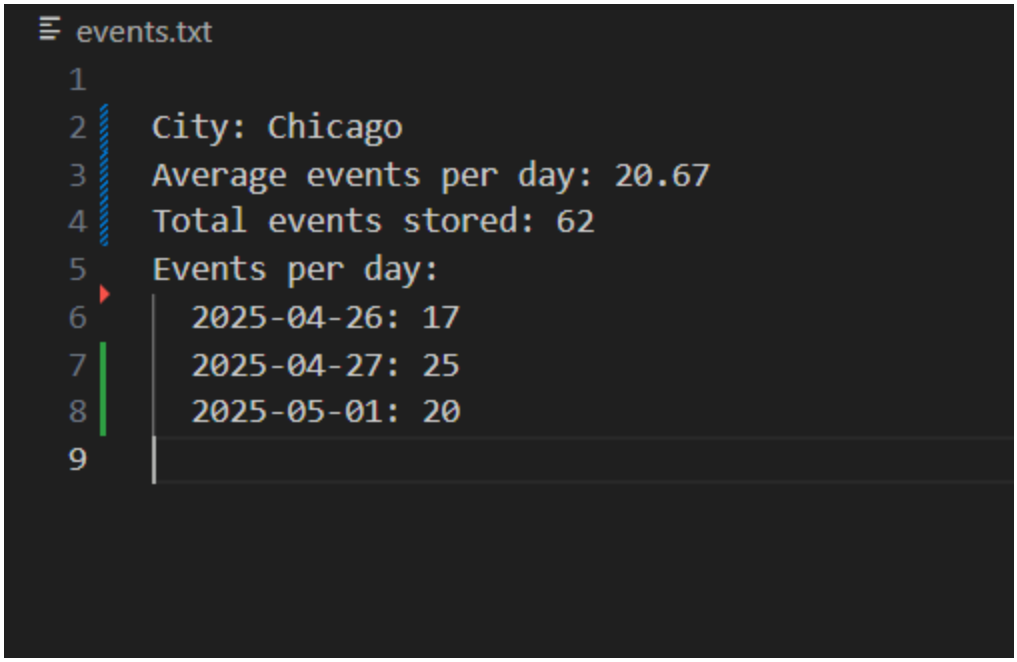
When prompted for a city, enter the city name. (Note that not all cities will work if they are not in the cities csv file. Please input popular cities that are in the file like Chicago or New York City to ensure the program works.)
When prompted for the date, enter the date in yyyymmdd format. Date must be within 10 days from the current date.
Example Input:
Enter the city: Chicago
Enter the date (yyyymmdd): 20250422

Calculations:

```
Weather data for Chicago

Maximum temperature for 42/22/2025 in Chicago: 70.
Minimum temperature for 42/22/2025 in Chicago: 42.

Maximum temperature for 42/23/2025 in Chicago: 58.
Minimum temperature for 42/23/2025 in Chicago: 54.

Maximum temperature for 42/24/2025 in Chicago: 58.
Minimum temperature for 42/24/2025 in Chicago: 50.

Maximum temperature for 42/25/2025 in Chicago: 61.
Minimum temperature for 42/25/2025 in Chicago: 47.

Maximum temperature for 42/26/2025 in Chicago: 51.
Minimum temperature for 42/26/2025 in Chicago: 45.

Average temperature for 42/22/2025 in Chicago: 56.0.
Average humidity for 42/22/2025 in Chicago: 62.7.
Average wind speed for 42/22/2025 in Chicago: 7.5.
Average UV index for 42/22/2025 in Chicago: 1.8.
Average chance of precipitation for 42/22/2025 in Chicago: 13.5.
Most common conditions for 42/22/2025 in Chicago: Partially cloudy.

Average temperature for 42/23/2025 in Chicago: 53.8.
Average humidity for 42/23/2025 in Chicago: 78.6.
Average wind speed for 42/23/2025 in Chicago: 6.3.
Average UV index for 42/23/2025 in Chicago: 2.0.
Average chance of precipitation for 42/23/2025 in Chicago: 26.6.
Most common conditions for 42/23/2025 in Chicago: Partially cloudy.

Average temperature for 42/24/2025 in Chicago: 51.9.
Average humidity for 42/24/2025 in Chicago: 76.3.
Average wind speed for 42/24/2025 in Chicago: 6.9.
Average UV index for 42/24/2025 in Chicago: 2.8.
Average chance of precipitation for 42/24/2025 in Chicago: 13.8.
Most common conditions for 42/24/2025 in Chicago: Partially cloudy.

Average temperature for 42/25/2025 in Chicago: 52.0.
Average humidity for 42/25/2025 in Chicago: 77.5.
Average wind speed for 42/25/2025 in Chicago: 8.5.
Average UV index for 42/25/2025 in Chicago: 0.5.
Average chance of precipitation for 42/25/2025 in Chicago: 55.1.
Most common conditions for 42/25/2025 in Chicago: Rain.

Average temperature for 42/26/2025 in Chicago: 46.6.
Average humidity for 42/26/2025 in Chicago: 73.4.
Average wind speed for 42/26/2025 in Chicago: 10.4.
Average UV index for 42/26/2025 in Chicago: 2.1.
Average chance of precipitation for 42/26/2025 in Chicago: 20.9.
Most common conditions for 42/26/2025 in Chicago: Partially cloudy.
```

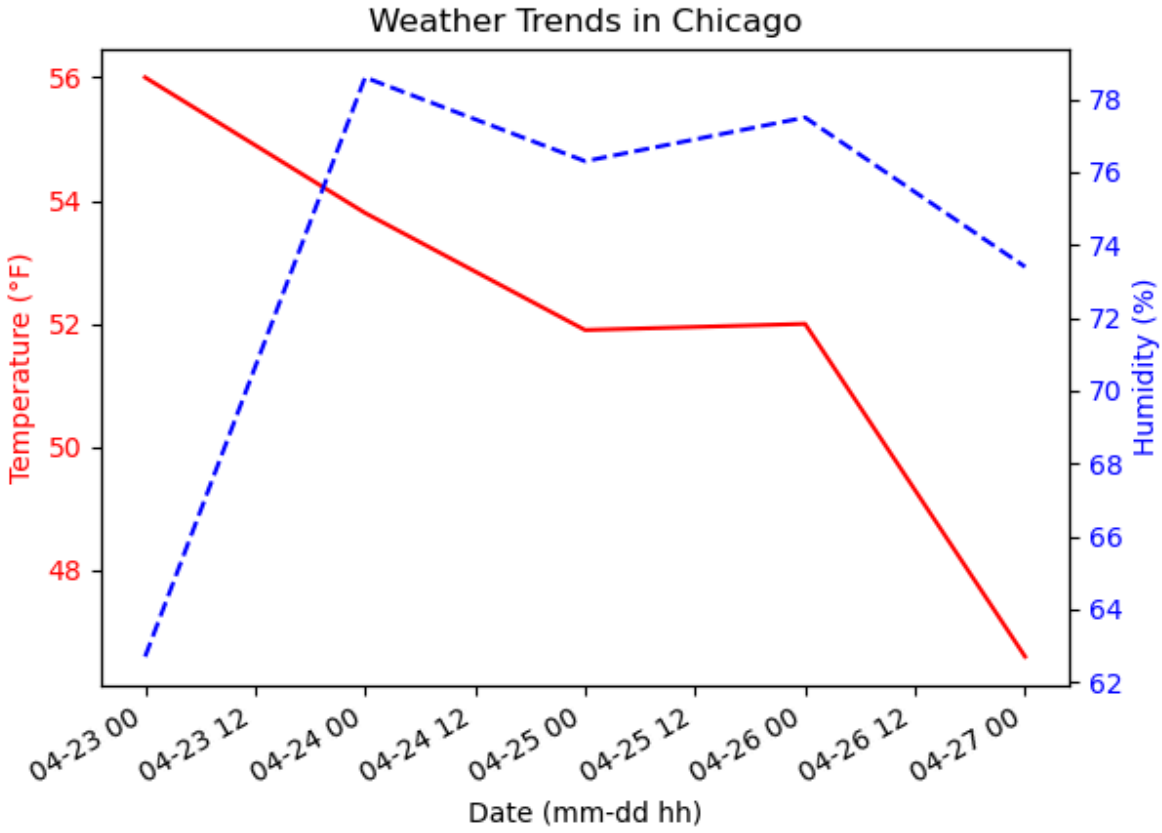
Calculations done from weather data to find the average weather data and the high and low temperatures over a 5 day period

Calculations done from Restaurants data to find average rating and average number of user reviews for different cuisines in a city

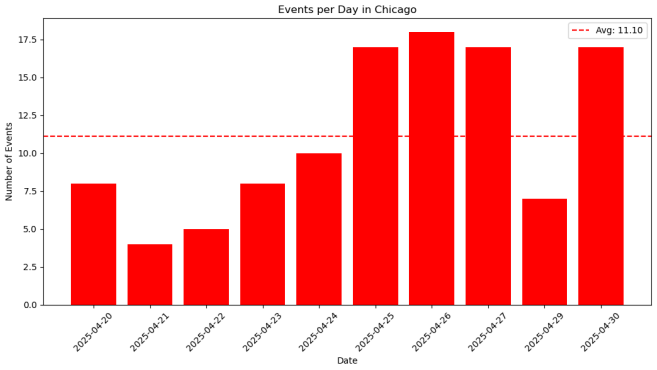


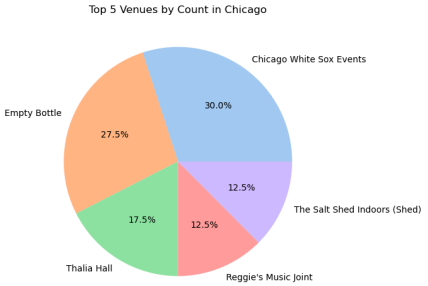
Calculation to store number of events per day and average events per day for a specific city.

Visualizations:

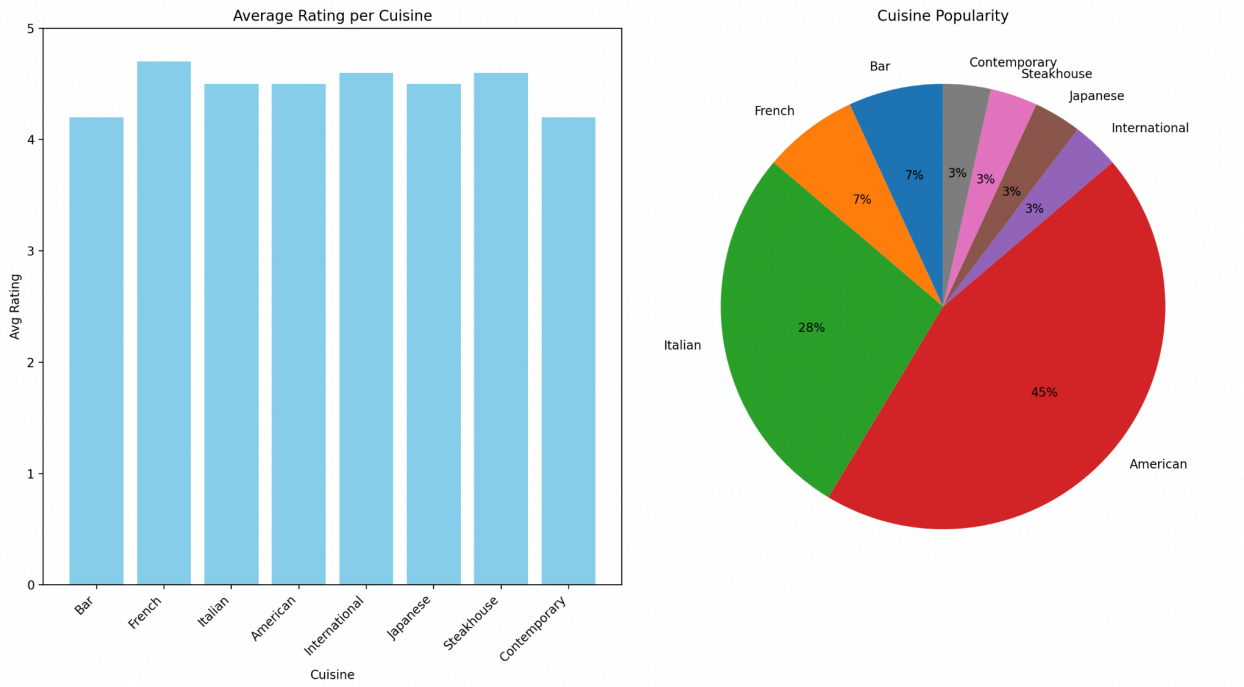


Line graph depicting average temperature and humidity over a 5 day period in Chicago





Bar chart depicting events per day and daily event counts in Chicago
Pie chart showing venues and the share of events that they are hosting over the same period in Chicago



Visualizations done from Restaurants data to visualize average rating and popularity for different cuisines in a city

Code documentation:

Weather.py

Python file used to request data from a weather api, create a database and store the data from the api request into the database.

Read_data_from_file(filename):

Reads data from a file with the given filename

Parameters

Filename: str
    The name of the file to read
Returns

Dictionary:
    Parsed JSON data from the file

setup_weather_database(db_name):

Sets up a SQLite database connection and cursor.

Parameters
---
db_name: str
    The name of the SQLite database
Returns

Tuple (cursor, connection):
    A tuple containing the database cursor and connection objects.

search_for_weather(city):

Sends a request to a weather api and returns the data retrieved from the api call

Parameters
---
city: str
    The name of the city the user wants to look up weather information for
Returns

Dictionary:
    Parsed JSON data from the api request if the request was successful
String:
    Error message with api response code if the request was unsuccessful

days_from_date(date1_int, date2_int):

Calculates the difference between two dates.

Parameters

Date1_int: int
    First date to the right of the subtraction operation
Date_2_int: int
    Second date to the left of the subtraction operation
Returns

Delta.days: int
    Difference between both dates
None

None if the dates aren't formatted properly

create_weather_table(data, cur, conn, days):

Creates weather table and adds weather data to the sql database for each hour in the day

Parameters
---
data: dictionary
Dictionary formed from the weather api call

cur: database cursor
conn: database connection object
days: int
Date corresponding to the weather data
Returns: None

create_conditions_table(cur, conn):

Creates table for all the different weather conditions in the sqlite database

Parameters:
---
cur: database cursor
conn: database connection object

Returns: None

create_cities_table(cur, conn):

Creates table from a cities.csv file with all of the city data into the sqlite database

Parameters:
---
cur: database cursor
conn: database connection object

Returns: None

weather_calc.py

Python file used to calculate data from the database, create visuals and a text file from the data, and condense all the methods into one function call.

daily_high_and_low(cur, date, city):

Calculates the highest and the lowest temperatures from a specified date and city in the database

Parameters:
---
cur: database cursor
Date: int
    Date to pull temperature data from the database
City: str
    City to pull temperature data from the database
Returns

Max_temp: int
    Maximum temperature for the specified date and city
Min_temp: int
    Minimum temperature for the specified date and city
Date: int
    Date corresponding to the temperature data

daily_avg(cur, day, city):

Calculates average weather data and most common weather conditions for specified date

Parameters:
---
cur: database cursor
day: int
    Date to pull temperature data from the database
City: str
    City to pull temperature data from the database
Returns
---
temp: float
    Average temperature for specified date and city
humidity: float
    Average humidity for specified date and city
wind_speed: float
    Average wind speed for specified date and city
uv_index: float
    Average UV index for specified date and city
precip: float

Average chance of precipitation for specified date and city
conditions: int
Most common weather conditions for specified date and city
day: int
Date corresponding to weather data

create_avg_chart(city, data, date_list):

Creates, saves and displays a line graph depicting the average temperature and humidity over a 5 day period for the given city

Parameters:
---
city: str
City corresponding to the temperature and humidity data
data: list
List of tuples containing the average weather data for the given city
date_list: list
List of integers containing the dates corresponding to the weather data
Returns: None

write_calculations(avg_data, temp_data, filename, city):

Writes all the weather calculations to a text file

Parameters:
---
avg_data: list
List of tuples containing the average weather data
temp_data: list
List of tuples containing the high and low temperature data
filename: str
Name of the file the data will be written to
city: str
Name of the city corresponding to the weather data
Returns: None

add_days_to_date(date_int, days):

Adds a specified number of days to a given date using the datetime python library to account for dates in other months

Parameters:
---

date_int: int
    Date in yyyymmdd format
days: int
    Number of days to be added to given date
Returns:
----
new_date: int
    Resulting date in yyyymmdd format if inputs are formatted correctly
None
    None if the date inputs are not formatted correctly
run_weather_app(city, date):

Method used to run methods in weather.py and weather_calc.py into one simple function call for seamless program integration

Parameters:
----
city: str
    Name of the city prompted from the user
date: int
    Date in yyyymmdd format prompted from user
Returns: None

Restaurants.py

search_restaurants(city)

Description:
Fetches restaurant data from the TripAdvisor API for a given city.

Input:

- city (str): Name of the city to search restaurants in.

Output:

- (dict): JSON response containing restaurant data.

create_cuisines_table(restaurants, cur, conn)

Description:
Creates a table for Cuisines and populates it with data from the restaurant JSON.

Inputs:

- restaurants (dict): JSON data containing a list of restaurant details.
- cur (sqlite3.Cursor): Cursor to execute SQL queries.
- conn (sqlite3.Connection): Connection to the SQLite database.

Output:

- None. (Table created and populated in database.)

create_restaurants_table(restaurants, cur, conn)

Description:
Creates a Restaurants table and populates it using JSON data.

Inputs:

- restaurants (dict): JSON data containing a list of restaurant details.
- cur (sqlite3.Cursor): Cursor to execute SQL queries.
- conn (sqlite3.Connection): Connection to the SQLite database.

Output:

- None. (Table created and populated in database.)

restaurant_calc(city, cur, conn)

Description:
Calculates and writes restaurant statistics for each cuisine in a city. Also generates visualizations.

Inputs:

- city (str): Name of the city to generate the report for.
- cur (sqlite3.Cursor): Cursor to execute SQL queries.
- conn (sqlite3.Connection): Connection to the SQLite database.

Output:

- None.
- Side Effects:
    - Creates a text file: RestaurantsReport.txt.
    - Saves a visualization image: RestaurantsVisual.png.

restaurants_call(cur, conn, city)

Description:
Wrapper function that fetches restaurant data, populates the database, and generates analysis and reports.

Inputs:

- cur (sqlite3.Cursor): Cursor to execute SQL queries.
- conn (sqlite3.Connection): Connection to the SQLite database.
- city (str): Name of the city to analyze.

Output:

- None.
- Side Effects:
    - Calls multiple functions to update the database and generate reports.

gather_ticketmaster.py

```
def init_database(db_name="weather.db"):
```

Initializes the tables into the database. It creates two tables, one for venues and another for events.

Input: db name - name of the database
Output: None

```
def fetch_ticketmaster_events(city, event_date, api_key, max_items=25):
```

Fetches events from Ticketmaster API for a city and date

Input:
city (str) - City name (e.g., 'New York')
event_date (str) - Event date in YYYY-MM-DD
api_key (str) - Ticketmaster API key
max_items (int) - Max events to fetch per call

Output: List of (name, date, venue_name, city, price_range) tuples

```
def store_events(events, db_name="weather.db"):
```

Store events in SQLite database, avoiding duplicates, using Cities table.

Input:
events (list) - List of (name, date, venue_name, city, price_range) tuples
db_name (str) - Database file

Output: Tuple of (new events stored, new venues stored, new cities stored)

```
def gather_events(city, event_date, db_name="weather.db"):
```

Gather and store Ticketmaster events for a city and date.

Input:
city (str) - City name (e.g., 'New York')
event_date (str) - Event date in YYYY-MM-DD
db_name (str) - Database file
Output: Tuple of (new events stored, new venues stored, new cities stored)

```
def fetch_and_visualize_events(city, event_date, db_name="weather.db"):
```

Fetch events and generate visualizations for the dashboard.

Input:
city (str) - City name (e.g., 'New York')
event_date (str) - Event date in YYYY-MM-DD
db_name (str) - Database file
Output: Tuple of (new_events, new_venues, new_cities, table_counts, sample_events, viz_images)

display_ticketmaster.py

```
def calculate_events_per_day(city_id, db_name="weather.db"):
```

Calculate average events per day and return counts.

Input:
city_id: shared city id throughout the program to filter cities
db_name (str) - Database file,
Output: Tuple of (average events per day, dict of date:count)

```
def get_venue_distribution(city_id, db_name="weather.db"):
```

Get count of events per venue.

Input: db_name (str) - Database file,
City_id: shared city id throughout the program to filter cities
Output: Dict of venue_name:count

```
def write_calculations(city_id, city_name, avg_events, counts,
db_name="weather.db", output_file="events.txt"):
```

Write calculations to the file that records average event count for the period and total event count per day.

Input:
city_id (int) - shared city id throughout the program to filter cities
city_name (str) - City name for output clarity
avg_events (float) - Average events per day
counts (dict) - Date:count
db_name (str) - Database file
output_file (str) - Output text file
Output: average events per day and total event count written to the events.txt file

```
def visualize_data(city_id, city_name, db_name="weather.db"):
```

Create two visualizations, a bar chart representing events per day and average events per day, and a pie chart representing the mix of venues for a specific city for the events entered into the database.

Input:
city_id: shared city id throughout the program to filter cities
db_name (str) - Database file
Output: two visualizations for the data

Resources:

Date	Issue Description	Location of Resource	Result
04/08/2025	Weather API	https://www.visualcrossing.com/	Used weather api to gather weather data for calculations and visualization
04/09/2025	Restaurants API	https://rapidapi.com/DataCrawler/api/tripadvisor16	Used TripAdvisor API to get restaurants data such as name,cuisine,rating,user reviews for calculations and visualizations
04/10/2025	Events API	https://developer.ticketmaster.com/products	Utilized the TicketMaster API to

		-and-docs/apis/discovery-api/v2/	fetch events and venues for a city. Outputted average daily event data and event count per day in the events.txt file.