SI 206: Data-oriented Programming

Final Project Report

Names: Kim Nguyen, Tiffany Tam, Ke Qing Wong

Uniqnames: hkimngu, itsteep, qscarlet

Team: Bao Buddies

Dr. Barbara Ericson

Table of Contents

Project Original Goals

We originally wanted to focus on evaluating the impact of nutritional content, eco-scores, and food-processing scores on the customer ratings of fast-food restaurants in Ann Arbor. Our team wanted to utilize data from the Yelp, Nutritionix, and Edamam APIs, and analyze the data in terms of the relationship between menu items, calorie count, and key nutritional metrics.

Project Achieved Goals

The hospitality sector, particularly restaurants, operates in an increasingly data-driven environment. Understanding customer preferences and behaviors is crucial for success. This project aimed to analyze the relationship between online ratings, customer engagement, and online booking using data from Yelp, Google Maps, and Opentable.

Project Objectives

1. **Yelp Ratings vs. Opentable Daily Bookings**

We aimed to uncover whether there is a direct correlation between the ratings a restaurant receives on Yelp and the number of bookings it garners on Opentable. This analysis is pivotal in understanding how online perceptions, reflected through Yelp ratings, translate into actual customer actions, in this case, restaurant bookings. The hypothesis driving this investigation was that higher Yelp ratings, indicative of positive customer experiences and satisfaction, would correlate with a greater volume of bookings on Opentable. To evaluate this, we gathered data on restaurants' Yelp ratings and compared them with their corresponding booking data from Opentable, aiming to identify trends or patterns that might suggest a relationship between these two variables.

2. **Yelp Review Word Count vs. Ratings**

The focus of this objective was to explore the relationship between the length of reviews on Yelp and the ratings provided by reviewers. The premise was that the word count of a review might reflect the intensity of the customer's experience, whether positive or negative. Longer reviews could indicate more detailed feedback, suggesting a stronger sentiment towards the restaurant, which in turn might correlate with either higher or lower ratings. By analyzing the word count of reviews and comparing them with the given ratings, we sought to understand whether there is a notable pattern – such as longer reviews correlating with more extreme ratings (either very high or very low) – and what this pattern might imply about customer engagement and satisfaction.

3. Frequency of Ratings for Yelp and Google Maps

We were interested in seeing overall, how well restaurants are rated and comparing the data between the searches for restaurants between Google Maps and Yelp and see any differences. The idea is to see if reviewers on Yelp and reviewers on Google Maps would rate restaurants around the same score and to see if the websites themselves impact the kinds of reviews each site produces. If the results are different, as in the ratings found on Google and Yelp are not the same, it could mean that reviewers on one site were more critical than the other and it is something to further investigate whether it is the environment around these review sites have an impact on the perception on the individual restaurants.

4. Streets with the Most Restaurants

One of our goals was to see what street has the most restaurants according to Google Maps, or if they were evenly distributed between the various streets of Ann Arbor. Location is important to the success of a restaurant, and knowing what street is most populated by other restaurants is important to people like business owners who may want to avoid competition, or customers who would want to know where there are the most options to eat at. Knowing this information can also lead to further investigations; if there is a street that is more densely packed with restaurants, that could mean that there are other factors of why there are more restaurants on a certain street than average.

Problems Faced

1. Lack of Common Variables

A significant hurdle we encountered was the absence of a standardized variable across our original choices for APIs, Edamam, Nutritionix, and Yelp. The data sets from these platforms lacked a common link that could be used for effective cross-comparison. For instance, the dish names in Yelp reviews often did not match the food items listed in Edamam and Nutritionix databases, making it challenging to accurately pair dishes with their nutritional profiles. This caused us to change the APIs we were using into ones where there were more common factors.

2. API Retrieval Difficulties

Accessing and retrieving data from these APIs posed another significant challenge. The original APIs, particularly for Edamam and Nutritionix, had complex structures and stringent usage limits. This complexity made it difficult to extract the required information efficiently and within the constraints of our resources and timelines. This is another reason why we switched to using the Google Maps API and using BeautifulSoup for Opentable instead. Even then, the APIs we were now using, Yelp Fusion API and Google Maps API, still had limitations on how often we

could request information. To get around this, we created multiple accounts to get the API keys from both websites and often switched around the keys when the limit for the day was reached.

3. Long Processing Times From APIs

The Google Maps API takes in a large load of information for every run of the execution code, so the wait time before the code finish running was around three to four minutes, not to mention it drained a lot of the remaining requests available from the API, so to work around it, we made it so that when requests were received from the Google Maps API, it was written down into a text file, so that the next time the code is executed, it would read from the text file to get information instead of having to request Google Maps again. This sped up the processing time significantly.

4. Running into Minor Details that Didn't Work

We had invisible files that made their way to our project folder and we weren't sure how to get rid of them. They were causing errors in our code.

5. Working with the Tables

Some of the data weren't able to show up in our table, and when they did, other parts wouldn't work. We had to switch what data we would get for each API. Additionally, we had some duplicate data, so we had to create a way to only get one of each restaurant.

Calculations

1. calculations.py Python file screenshot

Below are screenshots of the calculations.py Python file that we created to conduct calculations and create the calculations JSON file.

SI206finalproject > calculations.py > calculate_num_restarants_in_street

```python
import json
import sqlite3
import plotly.graph_objects as go

def visualization_yelp_ratings_vs_word_count(cur, conn):
    try:
        cur.execute(
            """
            SELECT rating, word_count
            FROM YelpData
            WHERE word_count IS NOT NULL
            """
        )
        res = cur.fetchall()
        if not res:
            print("No data found.")
            return

        ratings, word_counts = zip(*res)

        fig = go.Figure(data=[go.Scatter(name="Yelp Ratings vs. Word Count", x=ratings, y=word_counts, mode='markers', marker_color='rgb(55,83,109)')])

        title_str = "Yelp Ratings vs. Word Count"
        fig.update_layout(title=title_str, xaxis_title="Yelp Ratings", yaxis_title="Word Count")

        fig.show()

        fig.write_html("yelp_ratings_vs_word_count.html")
        fig.write_image("yelp_ratings_vs_word_count.png")

    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        conn.commit()

def visualization_yelp_ratings_vs_num_bookings(cur, conn):
    try:
        cur.execute(
            """
            SELECT yelp_rating, opentable_bookings
            FROM RatingNumBookings
            """
        )
        res = cur.fetchall()
        if not res:
            print("No data found.")
            return

        yelp_rating, opentable_bookings = zip(*res)

        fig = go.Figure()

        fig.add_trace(go.Scatter(x=yelp_rating, y=opentable_bookings, mode='markers', name='Yelp Ratings', marker=dict(size = 12)))
```

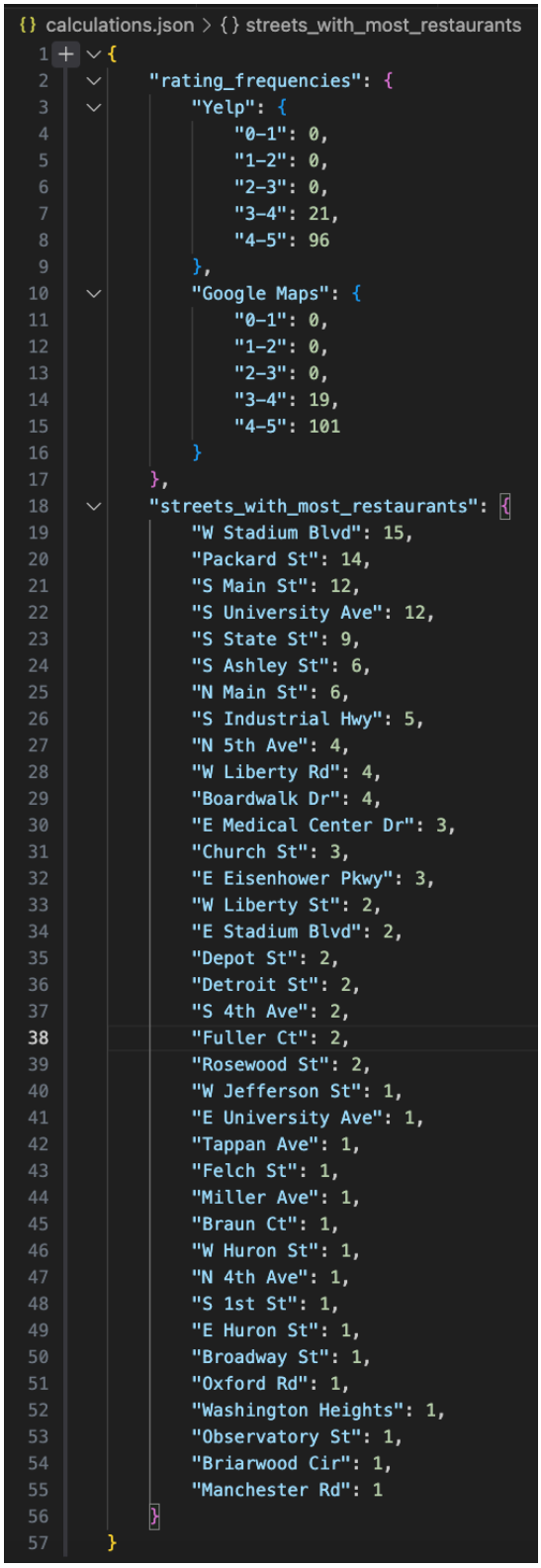
```python
        title_str = "Yelp Ratings vs Number of Bookings"
        fig.update_layout(title=title_str, xaxis_title="Number of Bookings", yaxis_title="Yelp Rating")

        fig.show()

        fig.write_html("yelp_ratings_vs_num_bookings.html")
        fig.write_image("yelp_ratings_vs_num_bookings.png")

    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        conn.commit()

def calculate_num_restarants_in_street(cur):
    cur.execute(
        """
        SELECT name, street FROM GoogleMapsData JOIN StreetID
        ON GoogleMapsData.street_id = StreetID.id
        """
    )
    street_count_dict = {}
    for row in cur:
        if row[1] not in street_count_dict:
            street_count_dict[row[1]] = 1
        else:
            street_count_dict[row[1]] += 1

    sorted_results = sorted(street_count_dict.items(), key=lambda x:x[1], reverse=True)
    return sorted_results

def visualization_googlemaps_vs_streetid(sorted_results):
    labels_list = []
    values_list = []

    for i in range(6):
        labels_list.append(sorted_results[i][0])
        values_list.append(sorted_results[i][1])

    fig = go.Figure(data=[go.Pie(labels=labels_list, values=values_list, hole=.3)])

    title_str = "Streets with the Most Restaurants"
    fig.update_layout(title=title_str)

    fig.show()

    # Save the figure
    fig.write_html("streets_with_the_most_restaurants.html")  # Save as interactive HTML
    fig.write_image("streets_with_the_most_restaurants.png")  # Save as static image (PNG)

def calculate_rating_frequencies(cur, table_name, rating_column):
    rating_ranges = [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5)]
    frequencies = []
```

```
    for lower, upper in rating_ranges:
        query = f"""
            SELECT COUNT(*)
            FROM {table_name}
            WHERE {rating_column} >= ? AND {rating_column} < ?
        """
        cur.execute(query, (lower, upper))
        count = cur.fetchone()[0]
        frequencies.append(count)

    return frequencies

def visualization_rating_frequencies(cur, conn):
    yelp_freq = calculate_rating_frequencies(cur, "YelpData", "rating")
    google_maps_freq = calculate_rating_frequencies(cur, "GoogleMapsData", "rating")

    rating_labels = ["0-1", "1-2", "2-3", "3-4", "4-5"]

    fig = go.Figure(data=[
        go.Bar(name='Yelp', x=rating_labels, y=yelp_freq),
        go.Bar(name='Google Maps', x=rating_labels, y=google_maps_freq)
    ])

    fig.update_layout(barmode='group', title="Rating Frequencies in Yelp and Google Maps",
                      xaxis_title="Rating Range", yaxis_title="Frequency")

    fig.show()

    fig.write_html("rating_frequencies.html")
    fig.write_image("rating_frequencies.png")

# Assuming calculate_rating_frequencies is as you defined in your previous message
def calculate_rating_frequencies(cur, table_name, rating_column):
    rating_ranges = [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5)]
    frequencies = []

    for lower, upper in rating_ranges:
        query = f"""
            SELECT COUNT(*)
            FROM {table_name}
            WHERE {rating_column} >= ? AND {rating_column} < ?
        """
        cur.execute(query, (lower, upper))
        count = cur.fetchone()[0]
        frequencies.append(count)

    return frequencies
```

```
def create_json_file(cur):
    restaurant_freq = dict(calculate_num_restarants_in_street(cur))
    yelp_freq = calculate_rating_frequencies(cur, "YelpData", "rating")
    google_maps_freq = calculate_rating_frequencies(cur, "GoogleMapsData", "rating")

    rating_labels = ["0-1", "1-2", "2-3", "3-4", "4-5"]
    rating_data = {
        "Yelp": dict(zip(rating_labels, yelp_freq)),
        "Google Maps": dict(zip(rating_labels, google_maps_freq))
    }

    json_data = {"rating_frequencies": rating_data,
                 "streets_with_most_restaurants": restaurant_freq}

    # Write to JSON file
    with open("calculations.json", 'w') as file:
        json.dump(json_data, file, indent=4)


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

# Create the JSON file
create_json_file(cur)

# Visualizations
visualization_yelp_ratings_vs_word_count(cur, conn)
visualization_googlemaps_vs_streetid(cur, conn)
visualization_yelp_ratings_vs_num_bookings(cur, conn)
visualization_rating_frequencies(cur, conn)

# Close the database connection
conn.close()
```

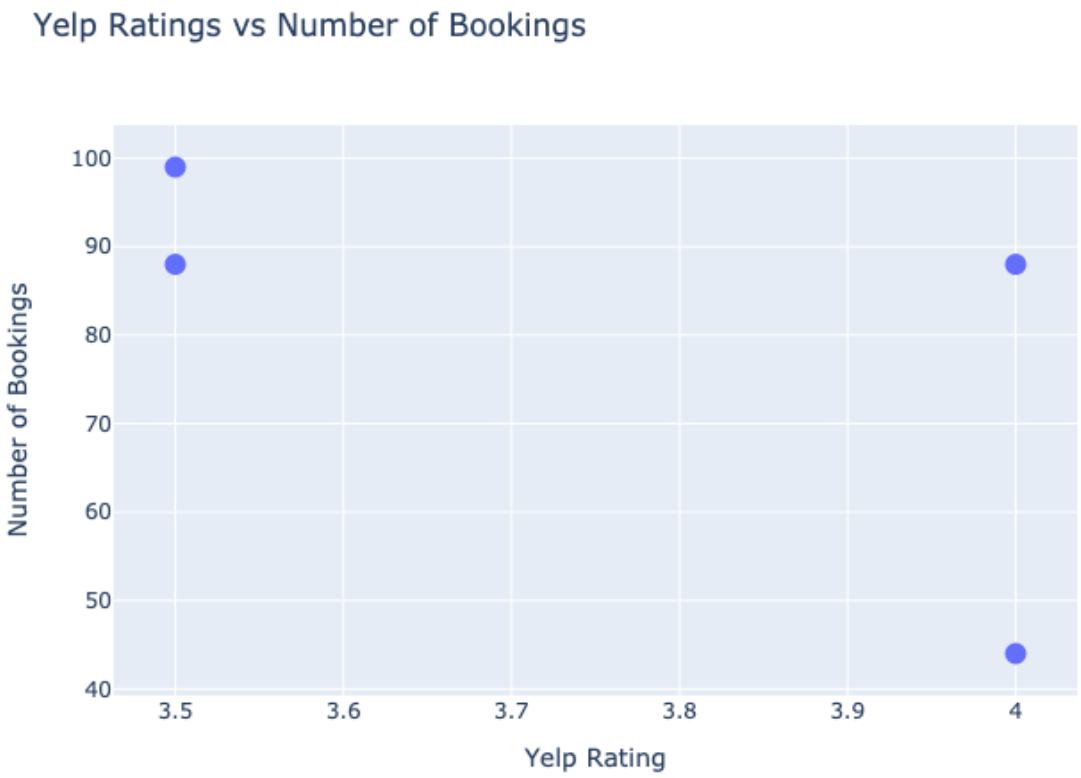
2. calculations.json JSON file screenshot

Below is a screenshot of the calculations.json file that was created based on the calculations.py Python file above.



Visualizations

1. Yelp Ratings vs. OpenTable Number of Bookings



We created a scatter plot with the title 'Yelp Ratings vs Number of Bookings'. The horizontal axis is labeled "Yelp Ratings" and it ranges from 3.5 to 4.0. The vertical axis is labeled 'Number of Bookings' and it ranges from 40 to 90. There are a few data points plotted on the graph:

- One data point is at approximately (3.5, 98), indicating that a Yelp Rating of 3.5 corresponds to a number of bookings of 98.
- Another point is around (4, 43), suggesting that the Yelp Rating increases to 4, while the number of bookings drops to 43.
- The other points are scattered between the above two points, without forming a clear trend or correlation between the two variables.

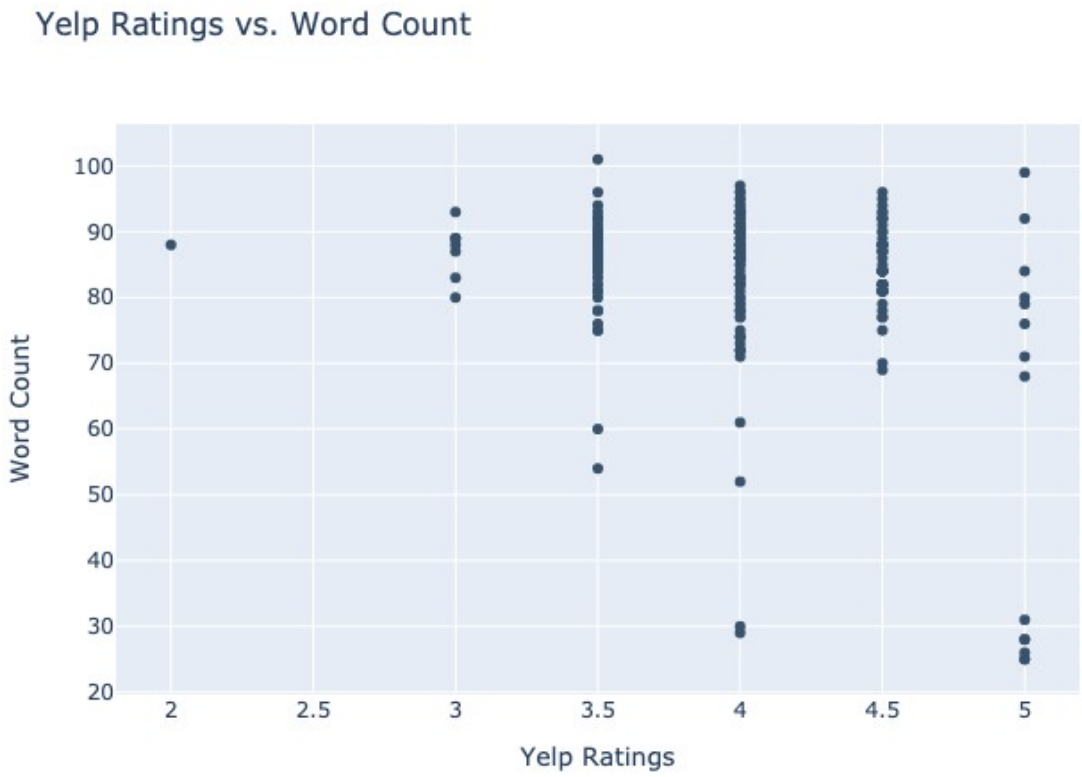
Usefulness

This graph could help in identifying trends, such as whether a higher customer satisfaction score directly influences their decision to book or visit a restaurant. Moreover, by observing the concentration and distribution of data points, businesses can gauge the impact of customer

ratings on their sales volume and can adjust their service quality strategies accordingly to optimize bookings.

Conclusion
The scatter plot does not show a clear relationship between the number of bookings and the Yelp ratings, as the data points are few and do not form any obvious pattern. This visualization would indicate that further data might be necessary to determine any potential correlation between the two variables.

2. Yelp Ratings vs. Yelp Review Word Count



We created a scatter plot titled ‘Yelp Ratings vs. Word Count’. The horizontal axis represents ‘Yelp Ratings’, which range from 3 to 5. The vertical axis represents ‘Word Count’, which is the total number of words in each Yelp review and extends from 30 to 100. The plot is populated with a considerable number of data points spread across the range of ratings.

- There are clusters of data points at each half and whole rating value (e.g., 3, 3.5, 4, 4.5, 5), suggesting that the Yelp ratings are discrete values rather than a continuous range.

- For ratings of 3.5, 4, and 4.5, the word counts vary widely but generally stay below 100 words.
- At rating 4 and 4.5, there is a significant concentration of reviews with higher word counts.
- Ratings of 3.5 show a similar pattern to 4.5, with reviews having higher word counts, although there is a more dispersed spread, ranging from 54 to 102 words.

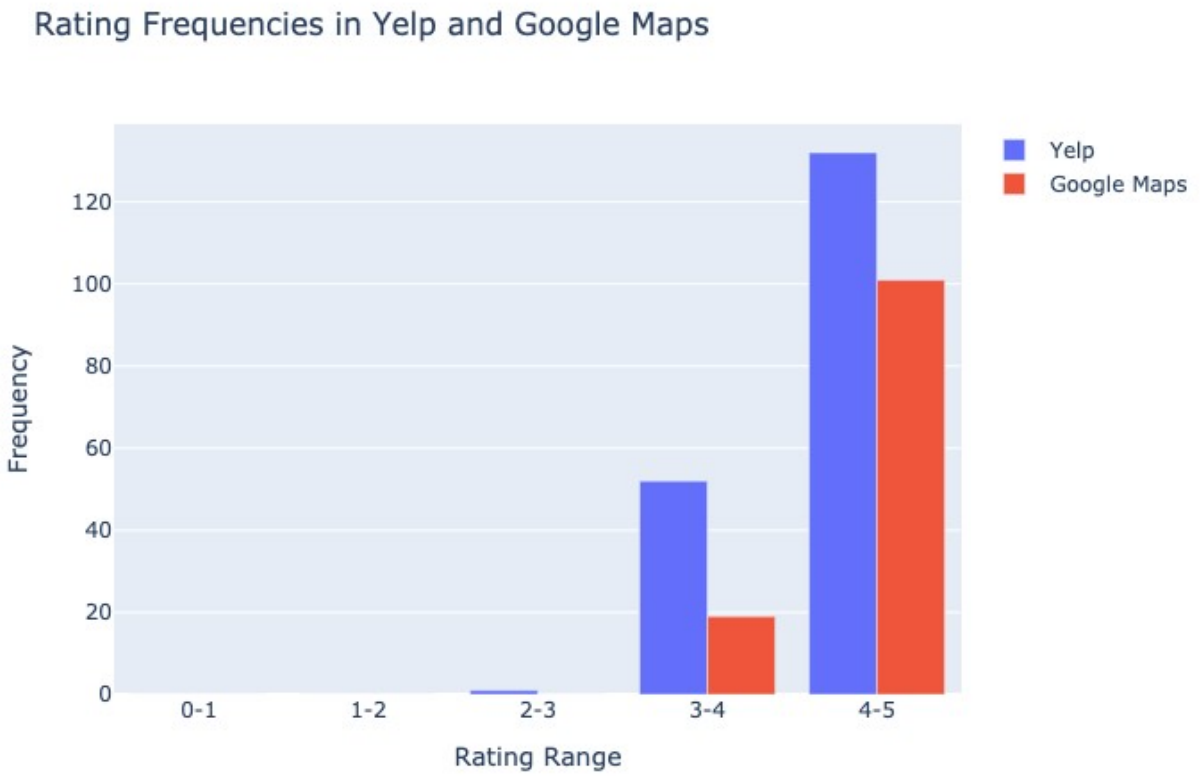
Usefulness

The graph could be used in assessing the review quality and informing strategic business decisions. It indicates that longer reviews may be associated with higher ratings, suggesting they may be perceived as more trustworthy and informative. Businesses can also leverage this insight to encourage customers to leave more detailed reviews and feedback, which, in turn, could positively influence potential customers by providing comprehensive perspectives on products and services.

Conclusion

This plot may suggest that higher Yelp ratings correlate with longer reviews, as indicated by the greater number of word counts. However, this is not a definitive conclusion and further statistical analysis would be needed to determine the correlation strength and significance.

3. Rating Frequencies in Yelp vs. Google Maps



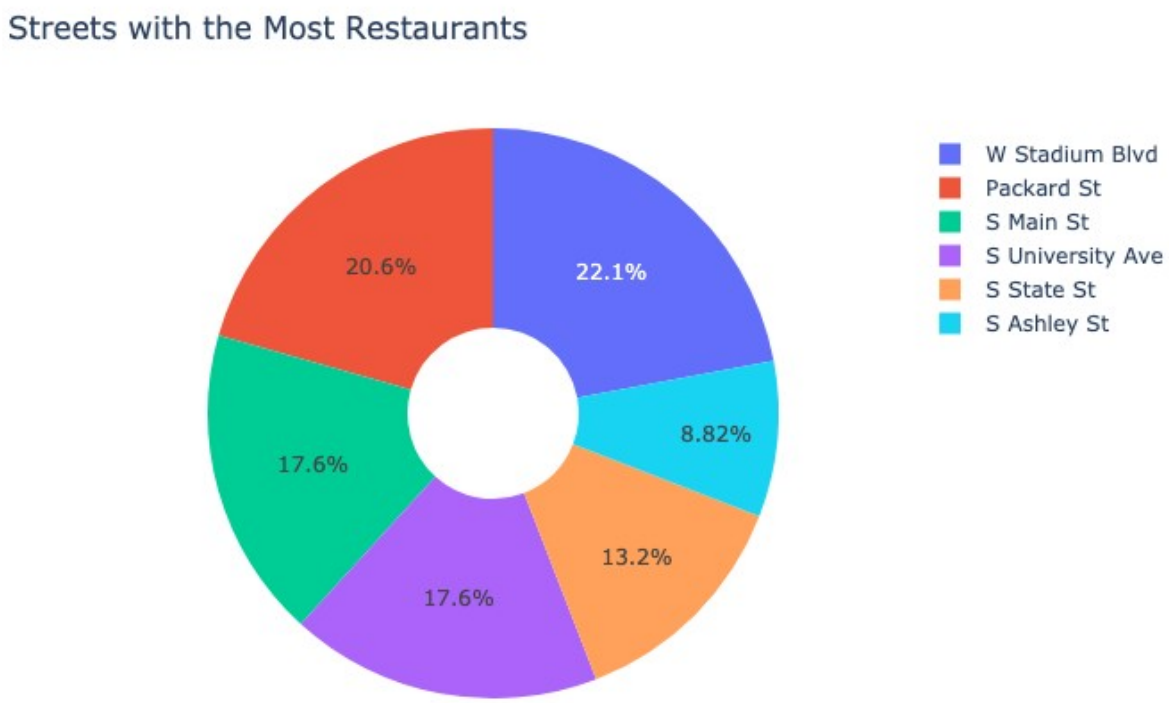
We created a clustered bar chart titled ‘Rating Frequencies in Yelp and Google Maps’. The horizontal axis represents the ‘Rating Range’ with the following categories: 0-1, 1-2, 2-3, 3-4, and 4-5. The vertical axis measures the ‘Frequency’ of ratings within each range, with values from 0 to 100. Two sets of bars represent the number of ratings within each rating range for Yelp and Google Maps. The first set, in blue, represents Yelp ratings. The frequency of ratings increases significantly with the higher rating ranges, with the 4-5 range having the highest frequency, and a count close to 96. The second set, in red, represents Google Maps ratings. The distribution of ratings on Google Maps shows a similar trend, with the highest frequency also in the 4-5 range, but the count is slightly more than that of Yelp, around 100.

Usefulness

This bar chart may be useful in several circumstances, for example, business analysis and platform comparison. For business analysis, businesses can see which platforms are more favorable for their reputation and adjust their marketing strategies accordingly. For platform comparison, the visual comparison of user engagement and rating behaviors on the two different platforms could be important for researchers or marketers looking to understand which platform has a more active or positive user base.

Conclusion
The chart demonstrates that both Yelp and Google Maps have the majority of their ratings in the 4-5 range, indicating that higher ratings are more common on these platforms. The lower rating ranges (0-1, 1-2, 2-3) have significantly fewer counts, suggesting that users tend to leave more positive reviews.

4. Streets with the Most Restaurants



We created a pie chart titled ‘Streets with the Most Restaurants’. The chart is divided into six segments, each representing a different street and its proportional share of restaurants. The segments are color-coded and labeled with both the name of the street and the percentage of restaurants on that street. The top 3 streets are as follows: W Stadium Blvd (22.1%), Packard St (20.6%), and S Main St (17.6%). S Ashley St has the smallest share with less than 9%.

Usefulness
The pie chart would be useful in market analysis, strategic planning for new restaurant locations, and understanding the competitive landscape of the food industry in a specific area.

Conclusion
The chart effectively shows the distribution of restaurants based on their streets. To improve the clarity and effectiveness of the chart, it may be a good idea to include the actual number of restaurants for each street along with the percentages. This could provide a better understanding of the size differences between the categories.

Instructions

GitHub link: https://github.com/scarletkeqing/SI206finalproject.git

Before executing the code, the user must check the SI206FinalProject folder if googlemaps_api_data.txt and processed_files.json, and if those two files are present in the folder, the user must delete them. After ensuring the two mentioned files are deleted, the user would go to the startercode.py file and execute that file. The initial execution of the file will take a couple minutes to finish processing. This would create the database, called Baobuddies.db, and insert 25 rows of data in each of the tables in our database. The initial run of the code should produce another googlemaps_api_data.txt and processed_files.json, do not delete these files after the initial run. If the user decides to run the code again, an additional 25 or fewer rows of data will be added to the tables.

In the calculations.py, by running the file, the calculations we have done will be exported into a json file titled “calculations.json”. Alongside the exported file, the four visualizations will be opened on a browser.

Documentation

Yelp Fusion API

Function Name	Input	Output	Description
create_yelp_db	db_name: String (SQLite database file name)	None	Connects to the SQLite database specified by db_name. Creates a 'YelpData' table if it doesn't exist with columns for name, rating, reviews, address, number, and word count. Name is the primary key. Commits changes to the database and closes the cursor.
gather_yelp_data	keyword: String (search term for Yelp Fusion API) conn: SQLite connection object cur: SQLite cursor object db_name: String (SQLite database file name)	None	Performs Yelp Fusion API requests with keyword. Extracts and processes data including name, rating, address, phone number, and reviews. Calculates word count for reviews. Inserts data into 'YelpData' table in the SQLite database. Commits changes to the database.
save_data_to_db	db_name: String (SQLite database file name) source: String (data source identifier) restaurant_data: Dictionary (restaurant data) reviews: List of Strings (reviews) restaurant_address: String restaurant_number: String word_count: Integer	None	Connects to the SQLite database specified by db_name. Inserts restaurant data into 'YelpData' table. Commits changes and closes the database connection.

OpenTable Beautiful Soup

Function Name	Input	Output	Description

create_table	cursor: A SQLite cursor object, used to execute SQL commands. conn: A SQLite connection object, representing the database connection.	None	Creates a table named 'OpenTable' in the SQLite database if it doesn't already exist. The table includes columns for ID, restaurant name, number of bookings, address, and phone number. The ID is set as the primary key and auto-increments for each new entry. Commits the changes to the database.
extract_booking_number	text: a string containing the text from which the booking number is extracted.	An integer which represents the number of times booked	Extracts the booking number from a text string using regular expressions and returns it as an integer.
extract_phone_number	text: a string containing the text from which the phone number is extracted.	An string which represents the phone number	Extracts the phone number from a text string using regular expressions and returns it as a string.
get_processed_files	None	A set of processed file names	Reads a JSON file named ‘processed_files.json’ and returns a set of processed file names. If the file doesn’t exist, it returns an empty set.
save_processed_files	‘processed_files_set’: a set containing names of processed files	None	Saves the set of processed file names into a JSON file named ‘processed_files.json’.
get_daily_bookings	db_name: A string representing the name of the database file to connect to. ‘processed_files_set’: a set of names of already processed files.	Dictionary, Set	Reads HTML files from a directory, extracts restaurant data, and returns a dictionary with the extracted data and a set of processed files. Processes up to 25 new entries each run.

insert_into_table	cur: A SQLite cursor object conn: A SQLite connection object 'All_items_dict': a dictionary containing restaurant data to be inserted 'db_name': the name of the SQLite database file.	None	Inserts data from a dictionary into the 'OpenTable' table in the SQLite database, ensuring no duplicate data is inserted. Closes the database connection after committing changes.

Google Maps API

Function Name	**Input**	**Output**	**Description**
write_dict_into_file	A_dict (dict): a dict that would be written into the file, filename (string): name of the file to be written.	None, but writes a text file.	Writes a dictionary to a file. Is used in get_googlemaps_data to write restaurant_dict into a file.
read_dict_from_file	filename (string): name of the file to be read.	Dictionary read from the file.	Reads a dictionary from a specified file.
get_ann_arbor_locations	None	Dictionary of neighborhoods with coordinates.	Grabs Ann Arbor neighborhoods and their latitudes and longitudes. The latitudes and longitudes are used to find restaurants in get_googlemap_data
get_restaurant_details	place_id (string): Google Maps Place ID of the restaurant.	List containing the address and phone number of the restaurant.	Retrieves street address and phone number of a restaurant based on its place ID. Is used in get_googlemap_data.
get_googlemap_data	location_dict (dict): Dictionary of locations with coordinates.	None, but the dictionary with restaurant details (name, rating, etc.) is	Finds nearest and unique restaurants in Ann Arbor areas and places their information into a dictionary,

		written into a file.	which is then written into a file.
get_street_dict	restaurant_dict (dict): Dictionary containing restaurant data.	Dictionary of streets.	Creates a dictionary of all the unique streets in the given restaurant dictionary.
create_google_maps_table	db_name (string): Name of the SQLite database.	None (creates a table in the database).	Creates a table in the SQLite database to store Google Maps data.
create_street_id_table	db_name (string): Name of the SQLite database.	None (creates a table in the database).	Creates a table in the SQLite database for street IDs.
insert_into_street_id_table	street_dict (dict): Dictionary of streets, db_name, (string): Name of the SQLite database.	None (inserts data into the database).	Inserts data into the Street ID table in the SQLite database in intervals of 25 or less.
insert_into_google_maps_table	restaurant_dict (dict): Dictionary with restaurant details, db_name (string): Name of the SQLite database.	None (inserts data into the database).	Inserts restaurant data into the Google Maps table in the SQLite database in intervals of 25 or less.

Starter Code

Function Name	**Input**	**Output**	**Description**
create_db	None	None, but it creates the Baobuddies database, along with the Yelp data table, the Opentable data table, the Google Maps data table, and the StreetID data table in it.	Is used to create the main tables used for calculations, which are the tables containing data from Yelp, Opentable, Google Maps, and StreetID.

create_ratings_and_num_bookings_table	cur (cursor): A SQLite cursor object, used to execute SQL commands conn (connection): A SQLite connection object, representing the database connection	None, but it creates a table used for the visualization for the visualization_yelp_ratings_vs_num_bookings function.	It creates a table using the YelpData table and the OpenTable table.

Calculations

Function Name	**Input**	**Output**	**Description**
visualization_yelp_ratings_vs_word_count	cur (cursor): A SQLite cursor object, used to execute SQL commands conn (connection): A SQLite connection object, representing the database connection	None, but it creates a visualization and writes it as an html and img file in the project folder.	It creates a graph that plots restaurants based on their Yelp ratings and the word count of the reviews.
visualization_yelp_ratings_vs_num_bookings	cur (cursor): A SQLite cursor object, used to execute SQL commands conn (connection): A SQLite connection object, representing the database connection	None, but it creates a visualization and writes it as an html and img file in the project folder.	None, but it creates a graph that plots restaurants and the number of bookings they have and their ratings on Yelp.

calculate_num_restarants_in_street	cur (cursor): A SQLite cursor object, used to execute SQL commands	It returns a sorted_results, which is a list of streets and the number of restaurants on that street	This function calculates the number of restaurants on each street from the tables GoogleMapsData and StreetID. It is used in visualization_googlemaps_vs_streetid to calculate the graph.
visualization_googlemaps_vs_streetid	cur (cursor): A SQLite cursor object, used to execute SQL commands	None, but it creates a visualization and writes it as an html and img file in the project folder.	It creates a pie chart of the top six streets with the most restaurants using the Google Maps data.
calculate_rating_frequencies	cur (cursor): A SQLite cursor object, used to execute SQL commands, table_name: a string that records the name of the table, Rating_column: a string that refers to the specific column in the database that contains the rating data	List of frequencies	Calculates the frequency of ratings within specified ranges for a given table and rating column. It segments ratings into five ranges, and counts the number of entries in each range, returning the result as a list of frequencies.
visualization_rating_frequencies	cur (cursor): A SQLite cursor object, used to execute SQL commands	None, but it creates a visualization and writes it as an html and img file in the project folder.	Visualizes the frequency of ratings for Yelp and Google Maps data. After calling the previous function, it creates a grouped bar chart to compare the two frequencies. It then creates visualizations that can be saved as HTML and PNG files.

create_json_file	cur (cursor): A SQLite cursor object, used to execute SQL commands	None, but it creates a json file named “calculations.json”	This function exports the calculations made in calculate_num_restarants_in_street and calculate_rating_frequencies.

Resources

You must also clearly document all resources you used.

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
11/30/2023	We kept running into issues getting the data and it was because we could only get 500 API calls per day.	Yelp Fusion API Documentation	We had to create extra API keys in order to test our functions.
11/30/2023	We were unsure how to get more than the same 20 items per execution of code through the Google Maps API, and wondered if there was another way to do it.	Google Maps API documentation	We used the Google Maps location search to give 10 neighborhoods in Ann Arbor, then used the latitudes and longitudes, found 20 restaurants per location.
12/7/2023	We had trouble figuring out how to insert 25 items into the GoogleMapsData table and have it remember where we left off last time to insert 25 new items.	SQL tutorials and available functions	We used the MAX function to get the latest ID from the GoogleMapsData table to keep track of where we last left off and what row was last inserted.