SI 206
Final Project Report

Github Repository: https://github.com/kyungrimK/final_project.git

PRESENTED BY TEAM TWO KIMS

Insun Kim

Kyungrim Kim

Contents

1. Project Goals

Most college students live on a budget. While students want to save their money, they also want to enjoy good food at the same time. However, every platform's ratings and price levels are different, and it creates confusion. Also, most of the platforms do not provide "cost-effective" filter options, making it hard for users to see which restaurants are both high-rated and budget-friendly. Thus, our team came up with a project finding the most cost-effective food in Ann Arbor using three different platforms: Yelp, Zomato, and Google Places. We hope this project will allow the students in Ann Arbor to find the best and cost-effective restaurant and ultimately spend their money in a smarter way.

2. Goals that were achieved

By calculating the rating-to-cost ratio for every restaurant, we could sort 123 restaurants from the most to least cost-effective. Among those restaurants, we found out that "Pilar's Tamales" ranked first with a price range of 1 and ratings of 4.47.

3. Problems Faced

1. **Yelp.py**
    a. Adding only 25 items to the database at a time, while also making sure the toal is more than 100 items.

       Solution:

        i. Limited the search results to 25.
        ii. Dropped a table if there are more than 101 items. Then, create a new table.
2. **Zomato.py**
    a. Since the main zomato API has limitations for a day, we needed to challenge ourselves to limit when we were using the data.
    b. To search restaurant names that were already saved in the database, we needed to get the location's entity_id and entity_title first and then search restaurants.
        i. The solutions to the problems are provided in section 8.
    c. There were some restaurants that have empty values for rating and price range.
        i. We substituted values to 0.
3. **Google_places.py**
    a. When searching the restaurants' info, some restaurants did not have either rating or price level.

Solution:

i. When there was no price level or rating, it returned nothing or as an empty list. When that happened, we replaced them as 0.

b. Also had issues of adding only 25 items to the database as we could not limit the number of returned results.

Solution:

i. Count the number, and if it reaches 25, 50, or 100, it breaks and adds items to the table.
ii. Just like Yelp.py, it dropped the existing table and created a new one if there are more than 101 items.

c. Due to uploading files that contain my Google API keys to Github, we got emails from Github and we were warned to not expose my API keys in public sources from Google.

Solution:

i. Made repo private.

4. **Calculation.py**
    a. Joining tables to load items we want.
    b. The solution to the problem is provided in section 8.
5. **Visualization.py**
    a. Changing details of the visualizations.
    b. The solution to the problem is provided in section 8.

4. Calculations

calculation_avgs.txt

```
Frita Batidos, 4.7, 2.0
Northside Grill, 4.17, 1.67
Sava's, 4.2, 1.67
Juicy Kitchen, 4.27, 1.67
Zingerman's Delicatessen, 4.33, 2.33
Tomukun Noodle Bar, 4.23, 2.0
Avalon Cafe and Kitchen, 4.1, 1.67
Zingerman's Roadhouse, 3.97, 2.67
Dimo's Deli and Donuts, 4.4, 1.0
Biercamp, 4.27, 1.67
Barry Bagels Ann Arbor, 4.3, 1.67
Jolly Pumpkin Cafe & Brewery, 4.3, 2.0
Anna's House - Ann Arbor, 4.0, 2.0
Tomukun Korean Barbeque, 4.23, 2.0
Cafe Zola, 4.07, 2.33
Isalita, 4.2, 2.0
Mani Osteria & Bar, 4.33, 2.0
Nick's Original House of Pancakes, 3.97, 2.0
Village Kitchen, 4.07, 1.33
Afternoon Delight, 4.17, 1.67
Detroit Street Filling Station, 4.2, 1.67
Momo Sushi, 4.2, 2.33
Chela's, 4.37, 1.33
Knight's Steak House, 4.1, 2.67
Lan City Hand Pulled Noodle, 4.43, 1.67
Jim Brady’s, 4.13, 1.67
Bell's Diner, 4.27, 1.0
Maize N Blue Deli, 4.23, 1.67
Mediterrano, 4.1, 2.33
The Lunch Room - Bakery & Cafe, 4.3, 1.33
Fresh Forage, 4.07, 1.67
The West End Grill, 4.1, 3.67
Fleetwood Diner, 4.0, 1.0
Kang's Korean Restaurant, 4.07, 1.33
Broadway Cafe & Hoagie, 4.27, 1.0
Seva Restaurant, 3.97, 2.0
Share Wine Lounge and Small Plate Bistro, 4.23, 2.33
The Jefferson Market, 4.3, 1.33
Everest Sherpa Restaurant, 4.2, 2.0
Ashley's Restaurant, 3.73, 2.0
KOSMO'S Bop Shop, 4.23, 1.0
Zola Bistro, 3.3, 2.33
Shake Shack, 3.57, 2.33
Haymaker Public House, 4.2, 2.0
TAQ - Taqueria Restaurant & Bar, 4.2, 1.67
Zamaan Cafe, 4.13, 1.67
Star's Cafe, 4.57, 1.33
Palm Palace, 4.0, 2.33
Metzger's German Restaurant, 4.37, 2.33
Jamaican Jerk Pit, 4.1, 1.67
RoosRoast Coffee, 4.4, 1.67
Madras Masala, 4.23, 1.67
Mark's Midtown Coney Island Restaurant, 4.1, 1.0
Pilar's Tamales, 4.47, 1.0
Bigalora Wood Fired Cucina, 3.77, 2.0
Benny's Family Dining, 4.1, 1.0
El Harissa Market Cafe, 4.27, 1.67
Grizzly Peak Brewing Company, 4.03, 2.0
NeoPapalis, 4.27, 1.0
Carlyle Grill, 4.07, 2.0
```

calculation_ratio.txt

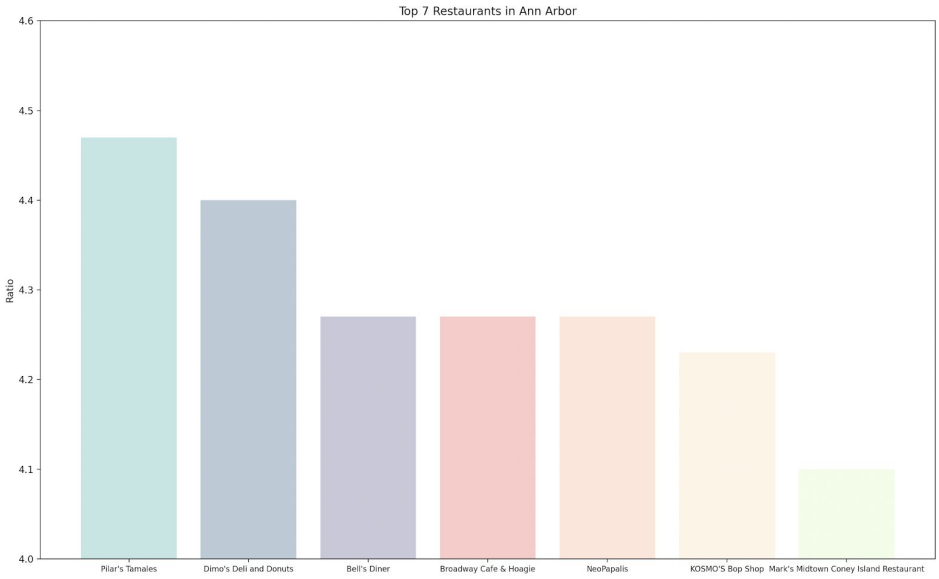
```
Pilar's Tamales, 4.47
Dimo's Deli and Donuts, 4.4
Bell's Diner, 4.27
Broadway Cafe & Hoagie, 4.27
NeoPapalis, 4.27
KOSMO'S Bop Shop, 4.23
Mark's Midtown Coney Island Restaurant, 4.1
Benny's Family Dining, 4.1
Fleetwood Diner, 4.0
Star's Cafe, 3.436
Chela's, 3.286
The Lunch Room - Bakery & Cafe, 3.233
The Jefferson Market, 3.233
Village Kitchen, 3.06
Kang's Korean Restaurant, 3.06
Lan City Hand Pulled Noodle, 2.653
RoosRoast Coffee, 2.635
Barry Bagels Ann Arbor, 2.575
Juicy Kitchen, 2.557
Biercamp, 2.557
El Harissa Market Cafe, 2.557
Maize N Blue Deli, 2.533
Madras Masala, 2.533
Sava's, 2.515
Detroit Street Filling Station, 2.515
TAQ - Taqueria Restaurant & Bar, 2.515
Northside Grill, 2.497
Afternoon Delight, 2.497
Jim Brady’s, 2.473
Zamaan Cafe, 2.473
Avalon Cafe and Kitchen, 2.455
Jamaican Jerk Pit, 2.455
Fresh Forage, 2.437
Frita Batidos, 2.35
Mani Osteria & Bar, 2.165
Jolly Pumpkin Cafe & Brewery, 2.15
Tomukun Noodle Bar, 2.115
Tomukun Korean Barbeque, 2.115
Isalita, 2.1
Everest Sherpa Restaurant, 2.1
Haymaker Public House, 2.1
Carlyle Grill, 2.035
Grizzly Peak Brewing Company, 2.015
Anna's House - Ann Arbor, 2.0
Nick's Original House of Pancakes, 1.985
Seva Restaurant, 1.985
Bigalora Wood Fired Cucina, 1.885
Metzger's German Restaurant, 1.876
Ashley’s Restaurant, 1.865
Zingerman's Delicatessen, 1.858
Share Wine Lounge and Small Plate Bistro, 1.815
Momo Sushi, 1.803
Mediterrano, 1.76
Cafe Zola, 1.747
Palm Palace, 1.717
Knight's Steak House, 1.536
Shake Shack, 1.532
Zingerman's Roadhouse, 1.487
Zola Bistro, 1.416
The West End Grill, 1.117
```

calculation_avgs.txt returns a dictionary with restaurant names as keys and their averaged ratings and price ranges as values.

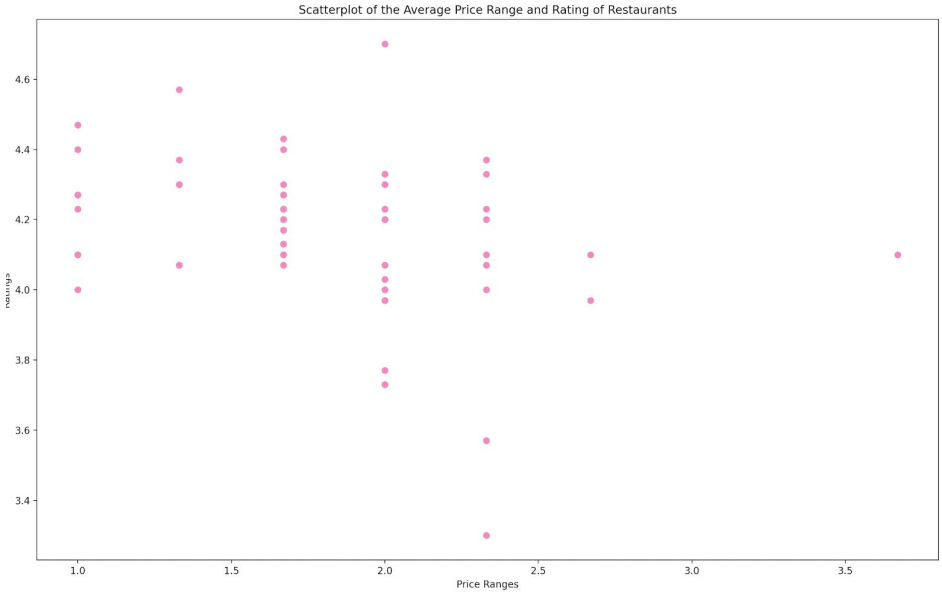
calculation_ratio.txt returns a dictionary with restaurant names as keys and their ratios as values.
Ratio = avg rating/ avg price range

Calculation functions are documented in more detail in section 7.4.

5. Visualizations



The bar graph "Top 7 Restaurants in Ann Arbor" compares the ratio of ratings to price ranges per restaurant and shows the top 7 restaurants. The higher the bar, the higher the ratio of ratings to price ranges for the restaurant is.



The scatterplot of the average price range and restaurants' rating illustrates how ratings are correlated to price range. The trend line naturally decreases. It can be interpreted as that the lower the price range, the higher the rating. Through the graph, it tells Ann Arbor has cost-effective and good restaurants or does not.

6. Instructions for Running Code

You should only run three files in this order:

1. **Main.py**

   You only need to run Main.py to run yelp.py, zomato.py and google_places.py simultaneously. You have to run the code for 5 times to get all the data and it will approximately take 3 mins for each run.

   Explanations for each file:

   a. **Yelp.py**

      Yelp.py searches for more than 100 restaurants that match the search criteria "Ann Arbor." It stores the returned items into two tables, restaurants_names and yelp_restaurants_info in database, restaurants.db. This file should be run before running zomato.py and google_places.py as those two files retrieve restaurants' names from the "restaurants_names" table, which is created by running this file.

   b. **Zomato.py**

      Zomato.py will put content into the database called restaurants.db by running the main zomato API and searching restaurants' names that have been already retrieved in the database through yelp.py. The code creates and puts retrieved data into a new table, 'zomato_restaurants_info' into the restaurants.db.

   c. **Google_places.py**

      Google_places.py will retrieve restaurant names from the table "restaurants_names." Using the retrieved data and Google Places API, it will search each restaurants' price ranges and ratings and store them into the table, "google_restaurants_info."

2. Calculation.py

Calculation.py will calculate the average ratings and price ranges and the rating-to-price ratio for each restaurant stored in the database. It will return text files with the computed data.

3. Visualization.py

Visualization.py will create a bar graph and a scatter plot based on the text documents returned from calculation.py.

7. Documentation for Functions

1. Main.py

def main

a) Input
   i) Runs yelp, zomato, and google_places python files to put data into the database restaurants.db.
b) Output: restaurants.db with four tables.

2. Yelp.py

def get_business_data

a) Input: location, offset=0
   i) Location indicates the geographic area to be used when searching for restaurants. The offset parameter is to offset the list of returned business results by written amount.
   ii) Requests to Yelp API.
   iii) Converts from JSON to the dictionary.
b) Output: business_dict
   i) Creates new dictionary - restaurant name as key and price for value
   ii) If the price is not indicated, it is listed as "0."
   iii) Returns new dictionary, business_dict.

def setUpDatabase

a) Input: db_name
   i) Takes a string filename argument as input and sets up a database connection and cursor.
b) Output: cur, conn
   i) Returns the database connection and cursor.

def adding_item_database

a) Input: dictionary, curr_id_num
   i) Takes any dictionary and current id number as an input.
   ii) Inserts current id number and restaurant names in the table, restaurants_names.
   iii) Select and insert restaurant id, and insert rating, review count, and price range in the table, yelp_restaurants_info.
b) Output: n/a (only inserts item to the database)

def adding_25_items_database

a) Input: location
    i) Location is used when calling business_dict function and searching for restaurants.
    ii) It checks whether the table exists. If it exists, it returns 1. If not, it will return 0.
    iii) If the table exists, it selects restaurants' ids and finds its length.
    iv) If the length of items are greater than 0 and less than 101, add items to the existing table.
    v) If there are more than 101 items, drop table.
    vi) If there is no existing table, create a table and add items to the table.
b) Output: n/a (only inserts item to the database)

3. Zomato.py

def setUpData

a) Input: db_name
    i) Takes a string filename argument as input and sets up a database connection and cursor.
b) Output: cur, conn
    i) Returns the database connection and cursor.

def getZomatoInfo

a) Input: api_key, location, restaurants_name, rating_reviews
    i) Takes a global variable (api_key), string (location), and global lists (restaurants_name, rating_reviews) as input.
    ii) Extracts restaurant names from the database 'restaurants' returned by yelp.py and appends them into the global list 'restaurants_name.'
    iii) Retrieves entity_id and entity_type in the JSON object from the zomato API using api_key.
    iv) Loads the restaurant's name, rating, review counts, and price range in the JSON object of the zomato API into a tuple looping through the list 'restaurants_name' and using api_key, entity_id, and entity_type.
        1) If rating and price range have empty value, the code substitutes them to 0.
    v) Appends the list 'rating_reviews' with the tuples.
b) output : rating_reviews
    i) Returns the list of tuples.

def database

a) Input: rating_reviews
    i) Takes the list of tuple ‘rating_reviews’ returned by getZomatoInfo.
    ii) Connects to a database called restaurant.db.
    iii) Creates a table zomato_restaurants_info, if it does not exist within the database.
    iv) Inserts restaurant_id that matches with the name in a table 'restaurant_names' in the database restaurants created in yelp.py, name, rating, reviews_count, and price_range.
        1) Adds only 25 items each time the code runs using iterator
b) Output: zomato_restaurants_info.db
    i) Commits the database.

4. Google_places.py

def setUpDatabase

a) Input: db_name
    i) Takes a string filename argument as input and sets up a database connection and cursor.
b) Output: cur, conn
    i) Returns the database connection and cursor.

def get_restaurant_names

a) Input: n/a
    i) Get restaurant names from the table, restaurants_names in the database.
b) Output: restaurant_names
    i) Returns restaurant names as a list.

def restaurant_search

a) Input: input
    i) Takes restaurant name as an input.
    ii) Requests to Google Places API and gets a list of price level and rating.
b) Output:
    i) If both price level and rating doesn't exist, returns [0,0,0].
    ii) If only rating exists, replace price level as 0 and return price level and rating as a list.

def get_restaurant_info

a) Input: n/a
    i) Retrieve list of restaurant names by calling function, get_restaurant_names.

ii) Searches each restaurant's price level and rating by calling function, restaurant_search

b) Output: restaurant_dict
   i) Returns all of the restaurants' info in a dictionary.

def adding_items_database

a) Input: dictionary
   i) Take any dictionary as an input.
   ii) Select restaurant names in table, restaurants_names and insert restaurant_id, price, and rating in table, google_restaurants_info.
b) Output: n/a (only inserts item to the database)

def adding_25_items_database

a) Input: n/a
   i) retrieve restaurants' price level and rating by calling function, get_restaurant_info.
   ii) It checks whether the table exists. If it exists, it returns 1. If not, it will return 0.
   iii) If the table exists, it selects restaurants' ids and finds its length.
   iv) If the length of items are greater than 0 and less than 101, add items to the existing table.
   v) If there are more than 101 items, drop tables.
   vi) If there is no existing table, create a table and add items to the table.
b) Output: n/a (only inserts item to the database)

5. Calculation.py

def setUpDatabase

a) Input: db_name
   i) Takes a string filename argument as input and sets up a database connection and cursor.
b) Output: cur, conn
   i) Returns the database connection and cursor.

def average

a) Input: avgdict
   i) Takes a list as input.
   ii) Calculates the average of values for each key using iterator and for loop.
b) Output: avg_dict

i) Returns a dictionary - restaurants names as keys and calculated values as values.

def calculate

a) Input
    i) Connects to a database called restaurant.db.
    ii) Loads the restaurant name and rating from three tables yelp_restaurants_info, zomato_restaurans_info, and google_restaurants_info by joining restaurants_id with restaurants_names into a dictionary rating_dict - restaurant names as keys and lists of ratings from three different tables as values.
        1) Removes items that contain 0 as value.
    iii) Averages three ratings for each restaurant through the function average.
    iv) Loads the restaurant name and price range from three tables yelp_restaurants_info, zomato_restaurans_info, and google_restaurants_info by joining restaurants_id with restaurants_names into a dictionary price_dict - restaurant names as keys and lists of price ranges from three different tables as values.
        1) Removes items that contain 0 as value.
    v) Averages three price ranges for each restaurant through the function average.
b) Output: name_r_p_dict
    i) Combines and returns the averaged rating_dict and the averaged price_dict into a new dictionary - restaurant names as keys and averaged ratings and price ranges as values.

def find_ratio

a) Input
    i) Runs the function ‘calculate’ and takes the output of the function.
    ii) Calculates ratio of averaged rating to averaged price range for each restaurant and inserts result into a new dictionary final_dict.
b) Output: final_dict

def write_txt

a) Input: filename, dictionaries
    i) Takes a string filename and a dictionary as input.
    ii) Inserts the key and value of the dictionary for each restaurant in the text, filename.
b) Output: text file

6. Visualization.py

def txtreader

a) Input: filename
   i) Take a text file as input.
   ii) Loads items in the text file into a dictionary.

b) Output: result

   iii) Returns the dictionary.

def bargraph

a) Input
   i) Gets data by calling txtreader with a file name “calculation_ratio.”
   ii) Creates visualization.
b) Output: a bar graph

def scatterplot

a) Input
   i) Gets data by calling txtreader with a file name “calculation_avgs.”
   ii) Creates visualization.
b) Output: a scatter plot

8. Documentation of Resources

Date	Issue Description	Location Resource	Result (did it solve the issue?)
12/05/20	We needed help on how to use the zomato API and how to retrieve data with a restaurants’ names list from the API/ documentation for the zomato API.	https://developers.zomato.com/documentation https://stackoverflow.com/questions/34038673/zomato-api-request-with-python-requests-library https://requests.readthedocs.io/en/master/user/quickstart/	Yes, we were able to figure out how to use header and parameters for the API url and to document the zomato API.

12/07/20	We needed help on how to join one table into three different tables in the database and to load joined items.	http://zetcode.com/db/sqlite/joins/	Yes, we were able to figure out how to join tables and to get items we want from tables using JOIN and WHERE if statements.
12/07/20	We needed help on how to put float values into a text file.	https://www.guru99.com/reading-and-writing-files-in-python.html	Yes, we were able to figure out how to insert string and float values in a dictionary with the write function and then percent s and f using an iterator.
12/08/20	We needed help on how to make visualizations and to change details in visualizations.	https://matplotlib.org/3.1.0/tutorials/introductory/pyplot.html https://pythonspot.com/matplotlib-bar-chart/	Yes, we were able to figure out how to create two different graphs and to change labels, titles, and colors.
12/5/20	Struggled using the Yelp Fusion API, especially how to retrieve restaurant data based on search criteria.	https://www.yelp.com/developers/documentation/v3/business_search	Yes, we learned how to use parameters and interpret the response body.
12/5/20	Struggled using the headers in Yelp Fusion API.	https://www.youtube.com/watch?v=GJf7ccRIK4U&ab_channel=SigmaCoding	Yes, we learned what to put in the headers.
12/9/20	Did not know how to check whether the table exists.	https://stackoverflow.com/questions/1601151/how-do-i-check-in-sqlite-whether-a-table-exists	Yes, we learned the complete query. We also knew it would return 1 if the table exists, and 0 if it does not.
12/7/20	Struggled using the Google Places API	https://developers.google.com/places/web-service/search?hl=en_US	Yes, we got the request url and learned how to use parameters.