Group Name: CA
Name: Angel Huang, Cecilia Huynh
Github Link: https://github.com/angelxhuang/final_project206
Course: SI 206: Final Project Report

1. Project Goals

Our goal for the project was to determine if there was an effect on Yelp ratings of restaurants and cafes based on the population in California cities. By calculating the average Yelp ratings of restaurants and the average Yelp ratings of cafes in select cities, we compared this to the population sizes. We grabbed 25 data points each from San Francisco, Silicon Valley, Southern California, Bay Area, Los Angeles, and overall California for restaurants and did the same for cafes for a total of 300 ratings. We matched these data points with their respective cities to examine our findings. We also compared the ratings of cafes to restaurants in a city. To visualize this, we planned to have four charts: one bar graph for average ratings of cafes and restaurants by city, one scatterplot for city population versus their average cafe and restaurant ratings, one bar graph for average price levels of cafes and restaurants by city, and one scatterplot for city population versus their average cafe and restaurant price levels.

2. Achieved Project Goals

In the beginning, we planned to look at Yelp and TripAdvisor APIs to compare ratings of restaurants and the city location. However, we revised our plan due to complications with getting the latter API and looked at a website (www.california-demographics.com/cities_by_population) that had California city population instead. The Yelp API provided valuable data of restaurant and cafe ratings and price levels, while the website gave us information about California city populations. By extracting information from these two sources, we were able to calculate the average ratings and price levels, collect the population sizes, and make graphs to represent our findings. We found that cafes generally have better ratings than restaurants do in the same city, and there is not much correlation between a city's population and their average cafe and restaurant ratings. Our conclusion comes from our two graphs about average ratings: a bar graph and a scatterplot. We also discovered that average cafe and restaurant price levels are generally the same in each city, and there was no correlation between the city population versus average cafe and restaurant price levels. Overall, this project allowed us to learn more about web scraping and APIs, understand that there is a very small relation between restaurant and cafe ratings in California cities, and lastly, visualize how there is no correlation between price levels of restaurants and cafes in addition to their city population.

3. Problems We Faced

Problem 1: Unable to access TripAdvisor API for free.
Solution: Revised our project plan to find a different API/website instead which also meant we had to change our goals for the project. Found a California city population website to compare restaurants and cafes with the population rather than Yelp ratings with overall TripAdvisor ratings.

Problem 2: Extracting randomized data from Yelp API gave cities that did not appear on California cities population website.
Solution: Specifically choose parts/regions of California for the majority of data points (25 from San Francisco, 25 from Silicon Valley, 25 from Southern California, 25 from Bay Area, 25 from Los Angeles, 25 California) to find ratings that correspond with population website. The latter of the 25 points are to account for some randomization.

Problem 3: Running code without duplicates in the data.
Solution: Used an offset query that allowed us to grab the data once without having repeated data over and over again.

4. Calculations from the Database File

Average Ratings of Cafes and Restaurants vs. Population of California Cities

cafes_population

Average Ratings of California Cities Places vs. Population	
Population	Average Ratings
117145	4.0
122989	4.5
544510	4.5
3849297	4.4
433823	4.24
148338	4.75
815201	4.3
127151	4.33
152258	4.14

res_population

Average Ratings of California Cities Places vs. Population	
Population	Average Ratings
117145	4.0
122989	4.33
544510	4.5
3849297	4.4
433823	4.14
148338	4.5
815201	4.26
983489	4.5
127151	4.33
152258	4.11

Average Price Levels of Cafes and Restaurants vs. Population of California Cities

cafe_price_pop

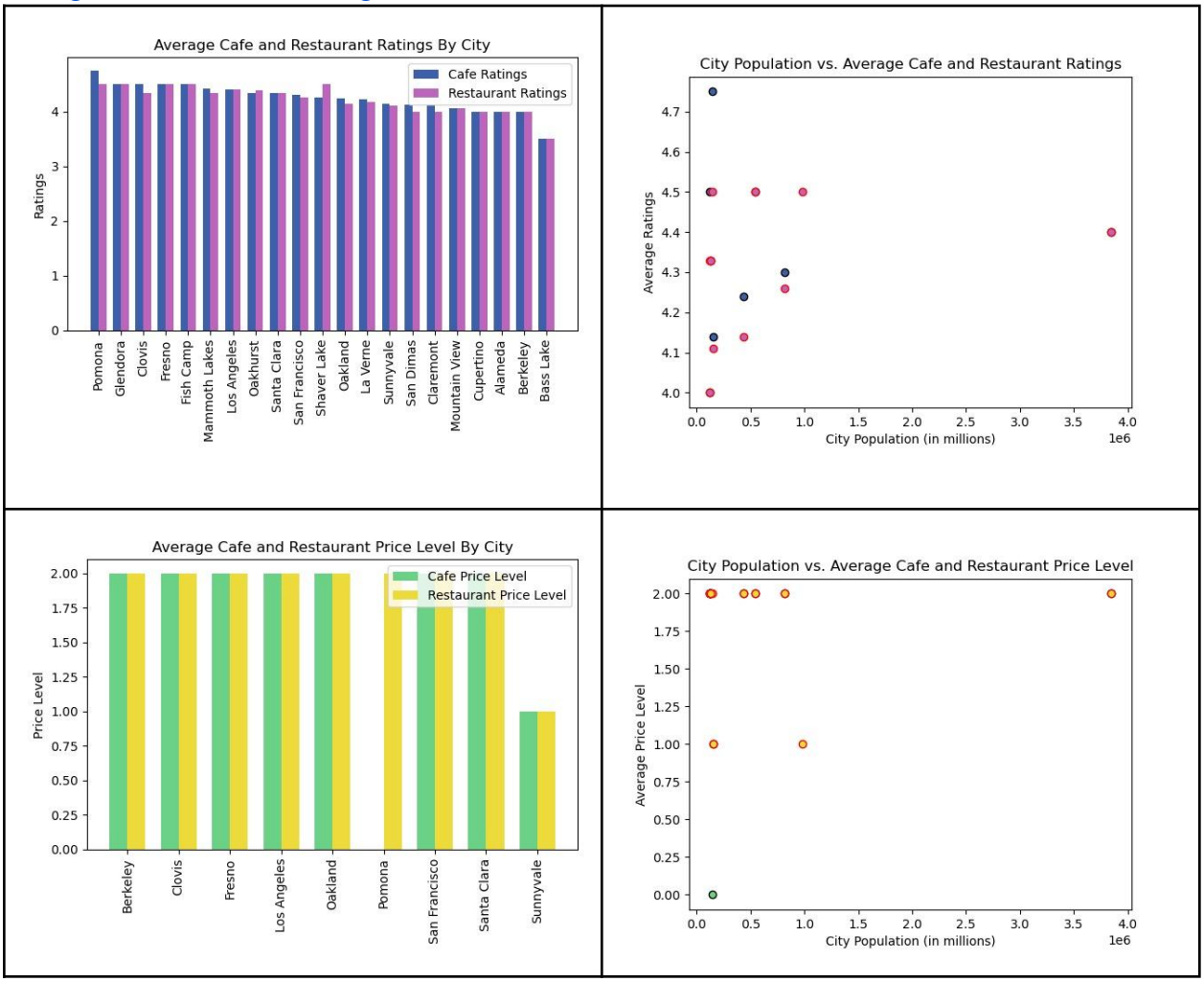
Average Price Levels of California Cities Places vs. Population	
Population	Average Price Levels
117145	2
122989	2
544510	2
3849297	2
433823	2
148338	0
815201	2
127151	2
152258	1

res_price_pop

Average Price Levels of California Cities Places vs. Population	
Population	Average Price Levels
117145	2
122989	2
544510	2
3849297	2
433823	2
148338	2
815201	2
983489	1
127151	2
152258	1

5. Created Data Visualizations

[Enlarged Versions of the Images]



6. Instructions For Running Code

Our code does not require any special instructions to run. Compile and run the code to have the CSV files and data visualizations appear.

7. Documentation For Each Function (input and output)

```
# Input: Database name
# Output: Creates a database
def setUpDatabase(db_name):
    path = os.path.dirname(os.path.abspath(__file__))
    conn = sqlite3.connect(path+'/'+db_name)
    cur = conn.cursor()
    return cur, conn
```

```
# Input: Dictionaries that you want to combine
# Output: List
def combine(dict1, dict2, dict3, dict4, dict5, dict6):
    combine_list = []
    for i in dict1['businesses']:
        combine_list.append(i)
    for i in dict2['businesses']:
        combine_list.append(i)
    for i in dict3['businesses']:
        combine_list.append(i)
    for i in dict4['businesses']:
        combine_list.append(i)
    for i in dict5['businesses']:
        combine_list.append(i)
    for i in dict6['businesses']:
        combine_list.append(i)
    return combine_list

# Input: Location, Offset
# Output: Dictionary
def yelp_cafes(location, offset):
    url =
f'https://api.yelp.com/v3/businesses/search?location={location}&categories=cafe&sort_by=best_match&limit=25&offset={offset}'
    headers = {
        "accept": "application/json",
        "Authorization": "Bearer
NHV7eR1EelF7RQYwG61LN1ODa4pLvqxqs6LMCpoxaQ4vEboO91EEyH7jdTgpgrG2GkT5p8dagosuw9FGeNF_rAwlvgKS0UeMyIpr2fqE74o4Lj6Hk6suedQx2cqQY3Yx"
    }
    response = requests.get(url, headers=headers)
    data = response.text
    cafes_lst = json.loads(data)
    return cafes_lst

# Input: Dictionary, Cur, Conn
# Output: A table based off Yelp cafes data, sorted by city, cafe name, rating, price level
def create_yelp_cafes(cafes, cur, conn):
    cur.execute("DROP TABLE IF EXISTS cafes")
    cur.execute("CREATE TABLE cafes (city TEXT, name TEXT, rating INTEGER, price TEXT)")
```

```
    # print(len(cafes))
    for i in cafes:
        city = i["location"]["city"]
        name = i["name"]
        rating = i["rating"]
        price = i.get("price", "none")
        cur.execute("INSERT INTO cafes (city,name,rating,price) VALUES
(?,?,?,?)",(city, name, rating, price))
    conn.commit()

# Input: Location, Offset
# Output: Dictionary
def yelp_restaurants(location, offset):
    url =
f'https://api.yelp.com/v3/businesses/search?location={location}&categories=restaurants
&sort_by=best_match&limit=25&offset={offset}'
    headers = {
        "accept": "application/json",
        "Authorization": "Bearer
NHV7eR1EelF7RQYwG61LN1ODa4pLvqxqs6LMCpoxaQ4vEboO91EEyH7jdTgpgrG2GkT5p8dagosuw9FGeNF_rA
wlvgKS0UeMyIpr2fqE74o4Lj6Hk6suedQx2cqQY3Yx"
    }
    response = requests.get(url, headers=headers)
    data = response.text
    restaurant_lst = json.loads(data)
    return restaurant_lst

# Input: Dictionary, Cur, Conn
# Output: A table based off Yelp restaurants data, sorted by city, restaurant name,
# rating, price level
def create_yelp_restaurants(restaurants, cur, conn):
    cur.execute("DROP TABLE IF EXISTS restaurants")
    cur.execute("CREATE TABLE restaurants (city TEXT, name TEXT, rating INTEGER, price
TEXT)")
    # print(len(cafes))
    for i in restaurants:
        city = i["location"]["city"]
        name = i["name"]
        rating = i["rating"]
        price = i.get("price", "none")
        cur.execute("INSERT INTO restaurants (city,name,rating,price) VALUES
(?,?,?,?)",(city, name, rating, price))
```

```
    conn.commit()

# Input: None
# Output: Dictionary
# About: We extracted the first 100 ranked california cities & their population
def cali_ratings():
    url = "https://www.california-demographics.com/cities_by_population"
    r = requests.get(url)
    soup = BeautifulSoup(r.text, "html.parser")
    cali_dict = {}
    tag = soup.find("table", class_= "ranklist")
    info = tag.find_all("tr")
    for tag in info[1:101]:
        lst = tag.find_all("td")
        city = lst[1].text.strip()
        pop = int(lst[2].text.strip().replace(",", ""))
        cali_dict[city] = pop
    return cali_dict

# Input: Dictionary, Cur, Conn
# Output: A table based off our california city population data, sorted by city and
population
def population_table(pop_dict, cur, conn):
    cur.execute("DROP TABLE IF EXISTS population")
    cur.execute("CREATE TABLE population (city TEXT, pop INTEGER)")
    for i in pop_dict:
        cur.execute("INSERT INTO population (city,pop) VALUES (?,?)",(i, pop_dict[i]))
    conn.commit()

# Input: Cur, Conn
# Output: Dictionary
# About: We selected city + cafe rating from our cafe data and calculated the average
# ratings of cafes in each city
def avg_yelp_cafes(cur, conn):
    cur.execute('SELECT city,rating FROM cafes ORDER BY rating DESC')
    avg_cafes_ratings = cur.fetchall()
    conn.commit()
    avg_ratings_dict = {}
    for i in avg_cafes_ratings:
        if i[0] in avg_ratings_dict.keys():
            avg_ratings_dict[i[0]].append(i[1])
        else:
```

```
            avg_ratings_dict[i[0]] = [i[1]]
    for x in avg_ratings_dict:
        avg_ratings_dict[x] = round((sum(avg_ratings_dict[x])) /
len(avg_ratings_dict[x]), 2)
    avg_ratings_dict = dict(sorted(avg_ratings_dict.items(), key=lambda item: item[1],
reverse=True))
    return avg_ratings_dict

# Input: Cur, Conn
# Output: Dictionary
# About: We selected city + restaurant rating from our restaurant data and calculated
the average
# ratings of restaurants in each city
def avg_yelp_restaurants(cur, conn):
    cur.execute('SELECT city,rating FROM restaurants ORDER BY rating DESC')
    avg_res_ratings = cur.fetchall()
    conn.commit()
    avg_res_dict = {}
    for i in avg_res_ratings:
        if i[0] in avg_res_dict.keys():
            avg_res_dict[i[0]].append(i[1])
        else:
            avg_res_dict[i[0]] = [i[1]]
    for x in avg_res_dict:
        avg_res_dict[x] = round((sum(avg_res_dict[x])) / len(avg_res_dict[x]), 2)
    avg_res_dict = dict(sorted(avg_res_dict.items(), key=lambda item: item[1],
reverse=True))
    return avg_res_dict

# Input: Cur, Conn
# Output: List
# About: Joins restaurants data and population data in a tuple format: (City name,
Population,
# Average restaurant ratings)
def res_pop_join(cur, conn):
    cur.execute('SELECT restaurants.city, population.pop,
ROUND(AVG(restaurants.rating), 2) FROM restaurants JOIN population ON restaurants.city
= population.city GROUP BY restaurants.city')
    result = cur.fetchall()
    # print(result)
    return result
```

```
# Input: Cur, Conn
# Output: Dictionary
# About: Joins restaurants data and population data in a dictionary format: {City
name: (Population,
# Average restaurant price level) + also changes all price level from string format to
integer
# format
def res_price_pop_join(cur, conn):
   cur.execute('SELECT restaurants.city, population.pop, restaurants.price FROM
restaurants JOIN population ON restaurants.city = population.city GROUP BY
restaurants.city')
   result = cur.fetchall()
   res_price_dict = {}
   for i in result:
       if i[2].strip() == '$':
           res_price_dict[i[0]] = (i[1], 1)
       elif i[2].strip() == '$$':
           res_price_dict[i[0]] = (i[1], 2)
       elif i[2].strip() == '$$$':
           res_price_dict[i[0]] = (i[1], 3)
       elif i[2].strip() == '$$$$':
           res_price_dict[i[0]] = (i[1], 4)
       else:
           res_price_dict[i[0]] = (i[1], 0)
   # print(res_price_dict)
   return res_price_dict

# Input: Cur, Conn
# Output: List
# About: Joins cafe data and population data in a tuple format: (City name,
Population,
# Average cafe ratings)
def cafe_pop_join(cur, conn):
   cur.execute('SELECT cafes.city, population.pop, ROUND(AVG(cafes.rating), 2) FROM
cafes JOIN population ON cafes.city = population.city GROUP BY cafes.city')
   result = cur.fetchall()
   # print(result)
   return result

# Input: Cur, Conn
# Output: Dictionary
```

```
# About: Joins cafe data and population data in a dictionary format: {City name:
(Population,
# Average cafe price level) + also changes all price level from string format to
integer
# format
def cafe_price_pop_join(cur, conn):
    cur.execute('SELECT cafes.city, population.pop, cafes.price FROM cafes JOIN
population ON cafes.city = population.city GROUP BY cafes.city')
    result = cur.fetchall()
    cafe_price_dict = {}
    for i in result:
        if i[2].strip() == '$':
            cafe_price_dict[i[0]] = (i[1], 1)
        elif i[2].strip() == '$$':
            cafe_price_dict[i[0]] = (i[1], 2)
        elif i[2].strip() == '$$$':
            cafe_price_dict[i[0]] = (i[1], 3)
        elif i[2].strip() == '$$$$':
            cafe_price_dict[i[0]] = (i[1], 4)
        else:
            cafe_price_dict[i[0]] = (i[1], 0)
    print(cafe_price_dict)
    return cafe_price_dict

# Input: Data table, File name
# Output: CSV File
# About: Writes the csv file for dot plot of "Average Ratings of California Cities
# Places vs. Population"
def write_csv_dot(data1, filename):
    f = open(filename, "w")
    f.write("Average Ratings of California Cities Places vs. Population")
    f.write('\n')
    f.write("Population, Average Ratings")
    f.write('\n')
    for i in data1:
        f.write(str(i[1]) + "," + str(i[2]))
        f.write('\n')

# Input: Data table, File name
# Output: CSV File
# About: Writes the csv file for dot plot of "Average Price Levels of California
Cities
```

```
# Places vs. Population"
def write_csv_dot_price(data1, filename):
    f = open(filename, "w")
    f.write("Average Price Levels of California Cities Places vs. Population")
    f.write('\n')
    f.write("Population, Average Price Levels")
    f.write('\n')
    for i in data1.keys():
        f.write(str(data1[i][0]) + "," + str(data1[i][1]))
        f.write('\n')

# Input: Data table, File name
# Output: CSV File
# About: Writes the csv file for bar plot of "Average Ratings of California Cities
# Cafes vs. Restaurants"
def write_csv_bar(data1, data2, filename):
    f = open(filename, "w")
    f.write("Average Ratings of California Cities Cafes vs. Restaurants")
    f.write('\n')
    f.write("City, Average Cafe Rating, Average Restaurant Rating")
    f.write('\n')
    for i in data1:
        if (i in data1.keys()) and (i in data2.keys()):
            f.write(i + "," + str(data1[i]) + "," + str(data2[i]))
            f.write('\n')

# Input: Data table, File name
# Output: CSV File
# About: Writes the csv file for bar plot of "Average Price Levels of California
Cities
# Cafes vs. Restaurants"
def write_csv_bar_price(data1, data2, filename):
    f = open(filename, "w")
    f.write("Average Price Levels of California Cities Cafes vs. Restaurants")
    f.write('\n')
    f.write("City, Average Cafe Price Levels, Average Restaurant Price Levels")
    f.write('\n')
    for i in data1.keys():
        if (i in data1.keys()) and (i in data2.keys()):
            f.write(i + "," + str(data1[i][1]) + "," + str(data2[i][1]))
            f.write('\n')
```

```
#ratings

# Input: File
# Output: Bar graph
# About: Creates a bar graph of 'Average Cafe and Restaurant Ratings By City'
def cali_bar_graph(file):
   f = open(file)
   lines = f.readlines()
   city = []
   cafe_ratings = []
   res_ratings = []
   for row in lines[2:]:
       value = row.split(",")
       city.append(value[0].strip())
       cafe_ratings.append(float(value[1].strip()))
       res_ratings.append(float(value[2].strip()))

   x = np.arange(len(city))  # the label locations
   width = 0.35  # the width of the bars

   fig, ax = plt.subplots()
   rects1 = ax.bar(x - width/2, cafe_ratings, width, label='Cafe Ratings',
color=['#3e60ab'])
   rects2 = ax.bar(x + width/2, res_ratings, width, label='Restaurant Ratings',
color=['#bd64bd'])

   # Add some text for labels, title and custom x-axis tick labels, etc.
   ax.set_ylabel('Ratings')
   ax.set_title('Average Cafe and Restaurant Ratings By City')
   ax.set_xticks(x, city)
   ax.set_xticklabels(ax.get_xticklabels(), rotation = 90)

   ax.legend()

   # ax.bar_label(rects1, padding=10)
   # ax.bar_label(rects2, padding=10)

   fig.tight_layout()

   plt.show()

# Input: Cafe file and Restaurant file
```

```
# Output: Scatter plot
# About: Creates a scatterplot of 'City Population vs. Average Cafe and Restaurant
Ratings'
def cali_dot_plot(file1, file2):
    f1 = open(file1)
    lines1 = f1.readlines()
    cafes_population = []
    cafes_avg_ratings = []
    for row in lines1[2:]:
        value = row.split(",")
        cafes_population.append(int(value[0].strip()))
        cafes_avg_ratings.append(float(value[1].strip()))
    # print(cafes_population)
    # print(cafes_avg_ratings)
    f1.close()

    f2 = open(file2)
    lines2 = f2.readlines()
    res_population = []
    res_avg_ratings = []
    for row in lines2[2:]:
        value = row.split(",")
        res_population.append(int(value[0].strip()))
        res_avg_ratings.append(float(value[1].strip()))
    # print(res_population)
    # print(res_avg_ratings)
    f2.close()

    # First Scatter plot
    fig, ax = plt.subplots()
    ax.scatter(cafes_population, cafes_avg_ratings, edgecolor ="black", c=['#3e60ab'])

    # Second Scatter plot
    ax.scatter(res_population, res_avg_ratings, edgecolor ="red", c=['#bd64bd'])

    ax.set_title('City Population vs. Average Cafe and Restaurant Ratings')
    ax.set_xlabel('City Population (in millions)')
    ax.set_ylabel('Average Ratings')
    plt.show()

#price
```

```
# Input: File
# Output: Bar graph
# About: Creates a bar graph of 'Average Cafe and Restaurant Price Level By City'
def cali_price_bar_graph(file):
   f = open(file)
   lines = f.readlines()
   city = []
   cafe_price = []
   res_price = []
   for row in lines[2:]:
       value = row.split(",")
       city.append(value[0].strip())
       cafe_price.append(float(value[1].strip()))
       res_price.append(float(value[2].strip()))

   x = np.arange(len(city))  # the label locations
   width = 0.35  # the width of the bars

   fig, ax = plt.subplots()
   rects1 = ax.bar(x - width/2, cafe_price, width, label='Cafe Price Level',
color=['#6bd186'])
   rects2 = ax.bar(x + width/2, res_price, width, label='Restaurant Price Level',
color=['#eddb3b'])

   # Add some text for labels, title and custom x-axis tick labels, etc.
   ax.set_ylabel('Price Level')
   ax.set_title('Average Cafe and Restaurant Price Level By City')
   ax.set_xticks(x, city)
   ax.set_xticklabels(ax.get_xticklabels(), rotation = 90)

   ax.legend()

   # ax.bar_label(rects1, padding=10)
   # ax.bar_label(rects2, padding=10)

   fig.tight_layout()

   plt.show()


# Input: Cafe file and Restaurant file
# Output: Scatter plot
```

```
# About: Creates a scatterplot of 'City Population vs. Average Cafe and Restaurant
Price Level'
def cali_price_dot_plot(file1, file2):
    f1 = open(file1)
    lines1 = f1.readlines()
    cafes_population = []
    cafes_avg_price = []
    for row in lines1[2:]:
        value = row.split(",")
        cafes_population.append(int(value[0].strip()))
        cafes_avg_price.append(float(value[1].strip()))
    print(cafes_population)
    print(cafes_avg_price)
    f1.close()

    f2 = open(file2)
    lines2 = f2.readlines()
    res_population = []
    res_avg_price = []
    for row in lines2[2:]:
        value = row.split(",")
        res_population.append(int(value[0].strip()))
        res_avg_price.append(float(value[1].strip()))
    print(res_population)
    print(res_avg_price)
    f2.close()

    # First Scatter plot
    fig, ax = plt.subplots()
    ax.scatter(cafes_population, cafes_avg_price, edgecolor ="black", c=['#6bd186'])

    # Second Scatter plot
    ax.scatter(res_population, res_avg_price, edgecolor ="red", c=['#eddb3b'])

    ax.set_title('City Population vs. Average Cafe and Restaurant Price Level')
    ax.set_xlabel('City Population (in millions)')
    ax.set_ylabel('Average Price Level')
    plt.show()

# Calls all of our main functions
def main():
```

8. Resources Used

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
12/2	Finding an API/website to replace our initial plan of using TripAdvisor	https://www.california-demographics.com/cities_by_population	Yes, we found and used this website to extract population size data using BeautifulSoup
12/3	We realized that we were getting restaurants/cafes from the same cities in California (that did not match the top 100 California cities we had in our other data table). We wanted a more spread-out set of data points that include restaurants/cafes from major cities in California	https://docs.developer.yelp.com/reference/v3_business_search	Yes, we realized that we can set the location to specific locations, such as "Silicon Valley" or "Bay Area", which included more major cities such as Berkeley or Santa Clara. This is why we decided to extract 25 data points each from different areas within California.
12/5	Because we could only retrieve 25 items each time we run our code, we wanted to combine all of the data (150 items) for restaurants and cafes.	https://www.geeksforgeeks.org/python-merging-two-dictionaries/	Yes, we ended up combining our cafes/restaurants data into one big dictionary that we can utilize for the rest of our functions.
12/7	We had trouble running the code without duplicating existing data (ex. Trying to not get the same 25 restaurants/cafes over and over again)	https://docs.developer.yelp.com/reference/v3_business_search	Yes, we realize that there was an offset query that allows us to get unique restaurants/cafes every time we run our code. This is also why we had offset as an input for our yelp_restaurants and yelp_cafes functions so we can control the uniqueness of our data.
12/8	We did not know how to set up database	Discussion 11	Yes, it worked. We grabbed the code from Discussion 11. def setUpDatabase(db_name): path = os.path.dirname(os.path.abspath(__file__)) conn = sqlite3.connect(path+'/'+db_name) cur = conn.cursor() return cur, conn
12/8	We had trouble joining our city population data and our calculated average ratings/price level data.	https://www.w3schools.com/python/python_mysql_join.asp	Yes, we figured out how to join the two data tables together. Because the price level data was in string forms (ex. "$" or "$$"), we had to convert the data to integer first (ex. "$" = 1, "$$" = 2, etc.) before calculating the average price levels and joining it with the population data into one table.

12/8	We had trouble making a grouped bar graph with labels.	https://matplotlib.org/stable/gallery/lines_bars_and_markers/barchart.html	Yes, we followed the code format for creating a grouped bar chart from this website and adjusted titles/data points based on our own project (ex. x-axis/y-axis labels + x-ticks/y-ticks).
12/9	We had trouble plotting multiple data from two different files onto the same scatterplot.	https://www.scaler.com/topics/matplotlib/scatter-plot-matplotlib/ https://www.geeksforgeeks.org/visualize-data-from-csv-file-in-python/	Yes, we followed the code format for implementing multiple scatterplots on the same graph from this website and adjusted titles/data points based on our own project (ex. x-axis/y-axis labels + x-scatter/y-scatter).
12/11	We were not able to change the color or edgecolor of the bar graphs.	https://www.python-graph-gallery.com/3-control-color-of-barplots https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html	Yes, we realized that we were using “c=” instead of “color=”. We also had to put the color code in [] for it to work.