Project 1: Yelp!

## Mayuresh and the Three Musketeers

**Data Exploration:**

In order to retrieve data from Yelp Fusion (Yelp’s developer platform) as a preliminary step, we had to obtain an API key and understand the documentation well enough to work around any constraints and limitations. Fortunately, obtaining a key was straight-forward (from [www.yelp.com/fusion](http://www.yelp.com/fusion)). The only prerequisites were to create an account and then an application.

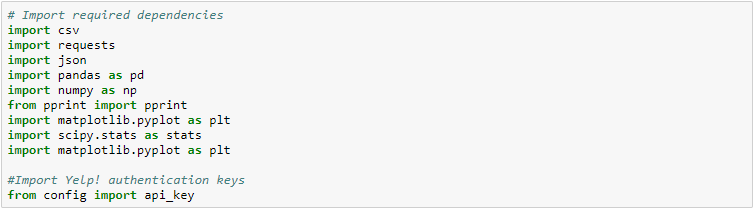
The limitation of using this free API key is that we are limited to 5000 API calls per day. One downside is that the documentation (<https://www.yelp.com/developers/documentation/v3/get_started>) itself didn’t have much sample code or explanation. A bit of exploring had to be done to know how to authenticate your API key per each request. This was done by passing credentials within a ‘header’ variable:

headers = {'Authorization': f"Bearer {api\_key}"}

Once we got authenticated and begun testing and using the API to retrieve various data, we understood what we could work with. For our purposes, the basic “Yelp Search” API was used. We aren’t developing an interactive or comparative tool, and we don’t need to perform searches to get business hours, phone numbers, photos, or transactions. The Search API allowed us to retrieve the following data for each business: location (city and state), price ($-$$$$), closed status, # total reviews, category, and average rating.

Another discovery was that we found that reviews are limited to 3 per business, which isn’t sufficient to perform a sentiment analysis, so that idea was discarded. In addition, the number of businesses we were limited to obtaining per API search was 50. This seemed a bit sparse, but we found a query addition (not in Yelp’s API) called offset=x, from which we were able to loop through pages of results to get the next 50, 100, etc.

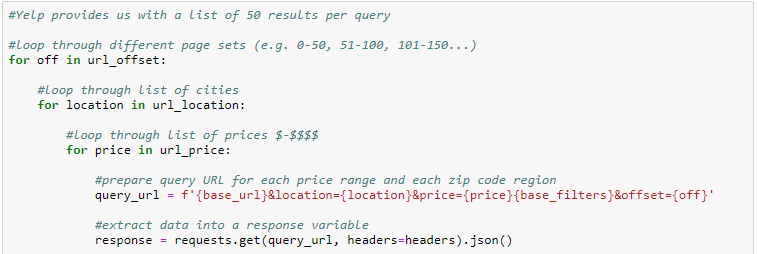
**Retrieving Data:**

****

Dependencies were built as we needed them. Our team each took a ‘divide-and-conquer’ approach, where after we’ve defined our primary dataset, we each used necessary modules to accomplish the objective.

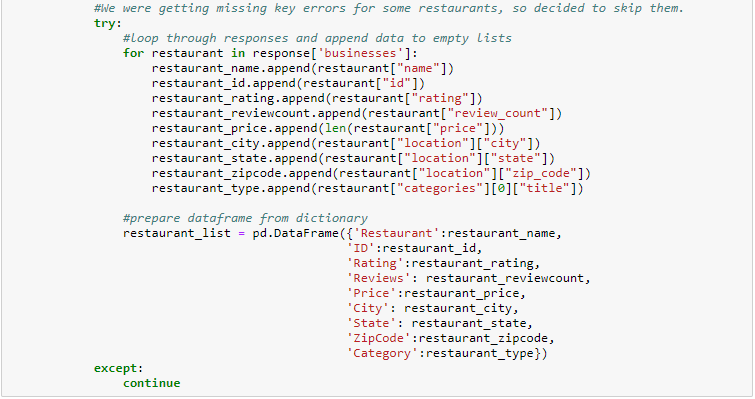


We’ve identified the type of data we needed and prepared empty lists to store and later retrieve them. The next piece is to define the base URL needed to retrieve data. The ‘base\_url’ is the root and ‘base\_filters’ define additional constraints we can impose to derive a more relevant dataset. For example, categories = restaurants allow us to focus on restaurants. Yelp has standards as to which categories are acceptable to pass within the ‘categories’ parameter. Possibly the most important is the ‘headers’ parameter, which is needed in every API call. Without that, an error would be prompted. Dependent factors, such as price, location, and the previously mentioned offset will be described later.

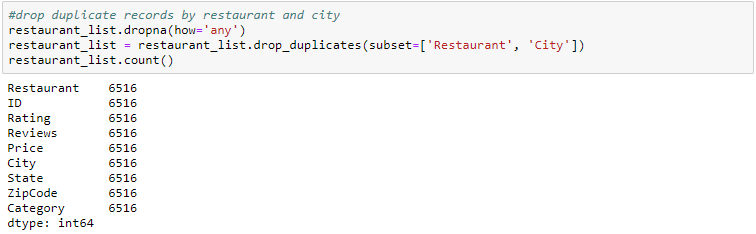


Since we ultimately had 4 cities and 4 different price points based on the data we wanted, the URL had to be dynamically customized per each API request. As defined by lists, ‘url\_offset’ allows us to loop through additional pages, so we aren’t constrained to 50 results per city, ‘url\_location’ allows us to loop through each location, and ‘url\_price’ allows us to loop through each price point.

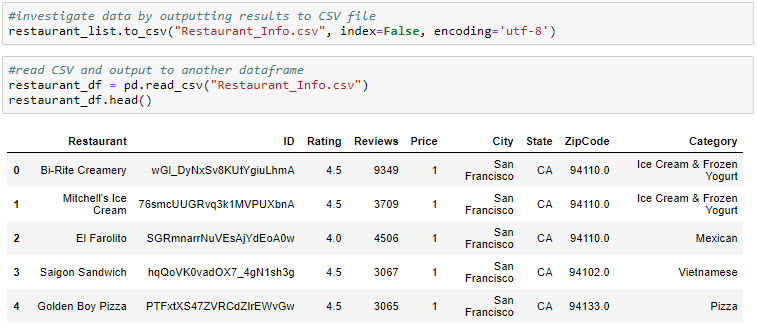
Finally, to pass the URL to the API, we defined ‘response’ to parse the data into JSON.



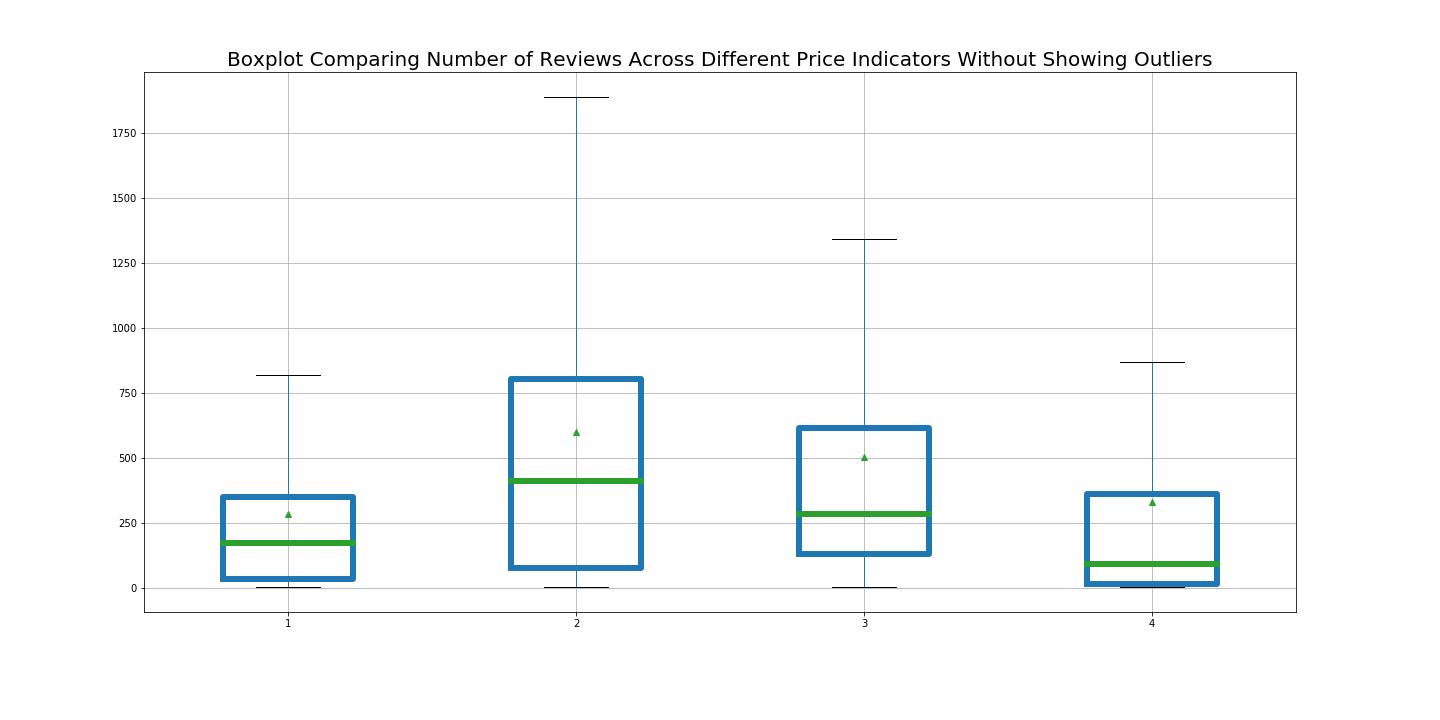
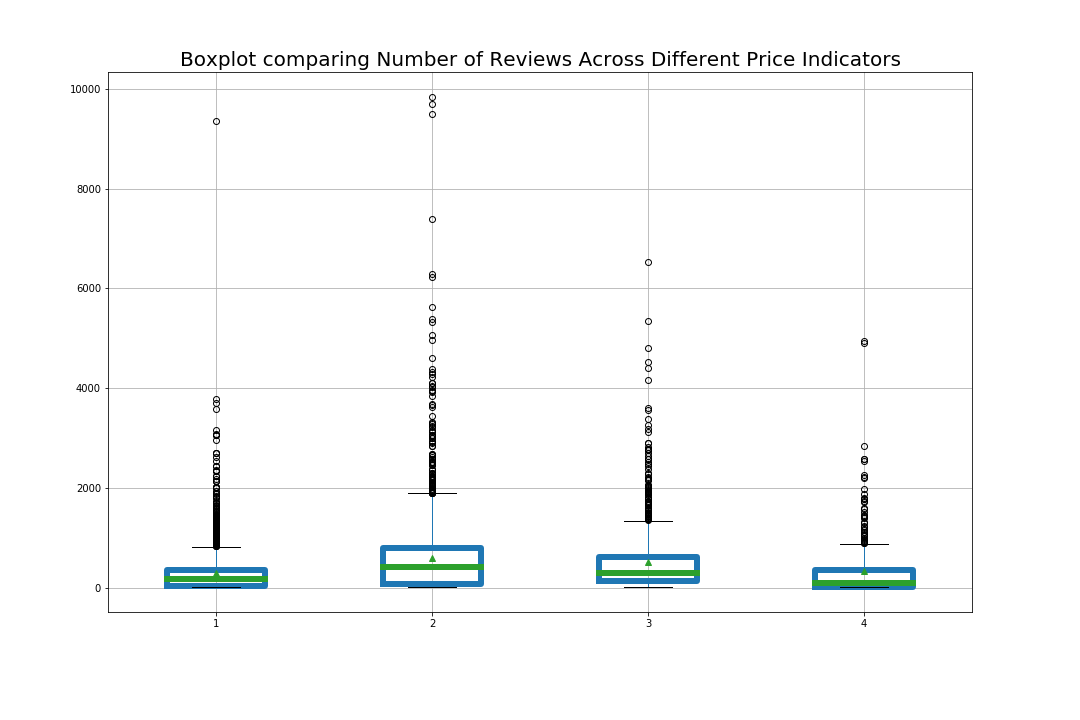
Looping through the responses, we are able to extract and build our dictionary. The first step would be to append each type of data into their respective lists, and the next is to append them to a dictionary. Exception handling was used here because we found some businesses returned an empty JSON response, prompting an error of ‘invalid key: businesses’, where ‘businesses’ is the root key of each response.



So, now we have our dataset, but we noticed that API searches extend outside the boundary of your search. We assume this behavior is intentional, similar to how a user is able to search for restaurants in a particular zip code on Yelp.com but retrieves results from other nearby zip codes. However, the issue is primarily getting duplicate results. Before omitting duplicates, there were around 10k results, which would have skewed our data quite a bit.



To mitigate the necessity of retrieving API results again (due to the 5000 API request daily limit) as well as to have a consistent data analysis, we have saved the unique dataset to a csv file and converted this to a Pandas dataframe to populate our charts.



**Objective:**

Testing the null hypothesis that the average number of reviews received by restaurants are no different across 4 different price indicators in Yelp. Four price indicators are displayed as “$”, “$$”,”$$$” and “$$$$” respectively in Yelp, which indicates gradually increasing price. The average number of reviews by restaurants could be interpreted as the degree of attachment/participation of the customers towards the restaurants.

**Data Management and Plotting process:**

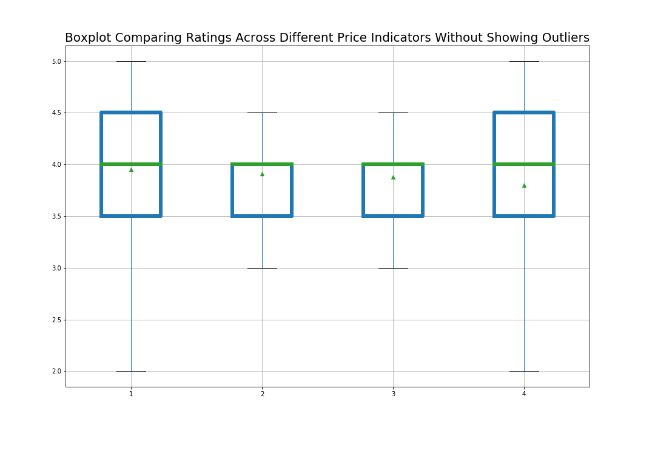
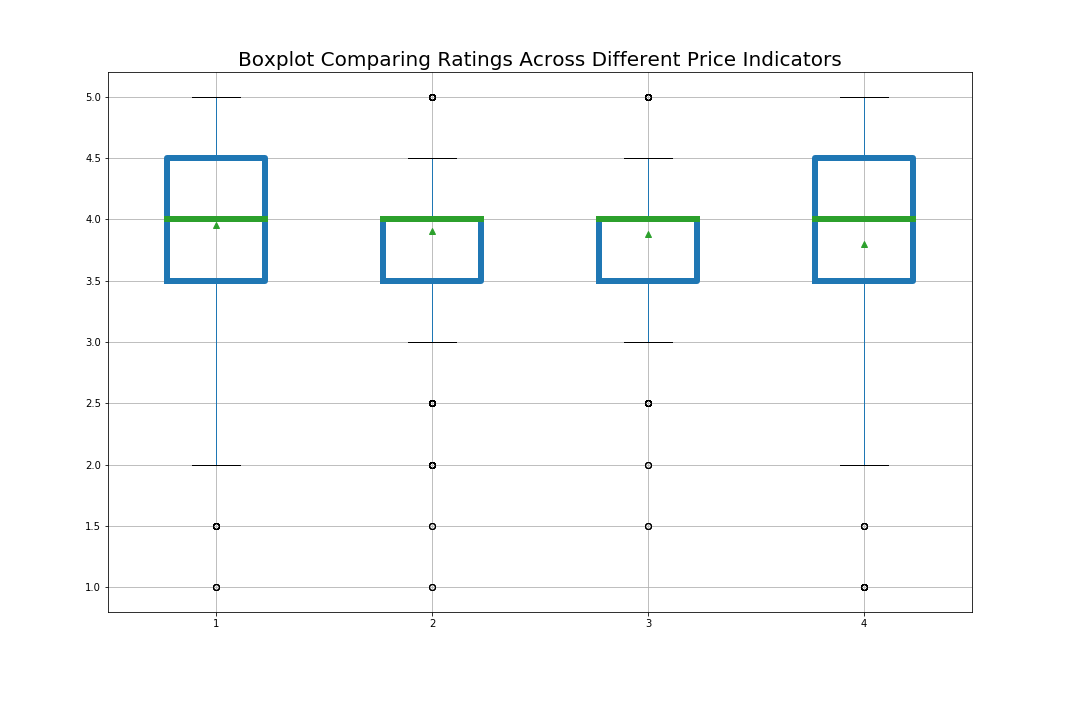
Utilizing the function boxplot() embedded in Pandas library and the comprehensive data frame that contains all the information, I generated two box plots as shown above to visualize the distribution of the review counts, mean of review counts (denoted as the triangle in each box) and median of review counts (denoted as the green horizontal line in each box) for each price indicator. The first and second boxplots are essentially showing the same data, whereas the second one is only showing boxplot itself without the outliers.

Example: restaurant\_df.boxplot("Reviews",by="Price",figsize=(15,10),showmeans=True,boxprops=boxprops,medianprops=medianprops)

**Statistical Analysis:**

Utilizing the data frame that contains all the data downloaded from Yelp, I extracted the number of reviews for each price indicator by creating 4 variables, i.e review1, review2, review3 and review4 and applied the One-Way ANOVA (example code: stats.f\_oneway(review1, review2, review3, review4) ).

The result of ANOVA with P-value pvalue being 4.57108165194e-68 indicates that the number of reviews submitted by customers are different based on the price indicators of the restaurants. According to the boxplot, Yelp users are most likely to leave comments for medium expensive restaurants ($$).



**Objective:**

Testing the null hypothesis that the average rating received by restaurants are no different across 4 different price indicators in Yelp. Four price indicators are displayed as “$”, “$$”,”$$$” and “$$$$” respectively in Yelp, which indicates gradually increasing price. The average rating by restaurants could be interpreted as the degree of popularity of the customers towards the restaurants.

**Data Management and Plotting process:**

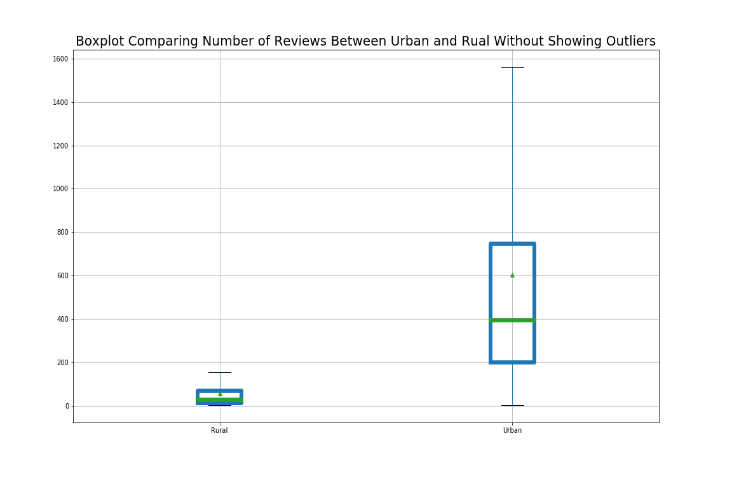
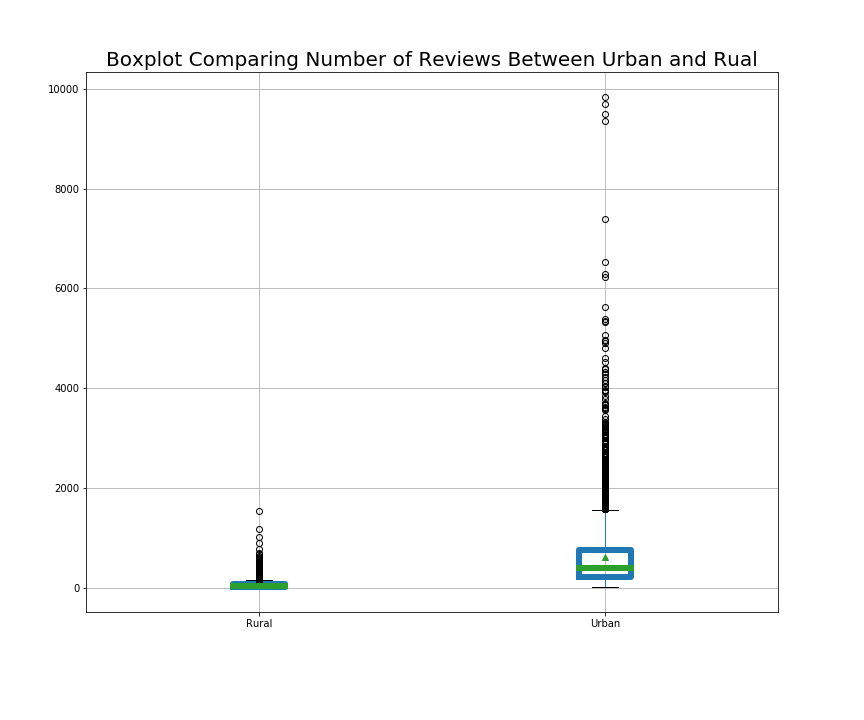
Utilizing the function boxplot() embedded in Pandas library and the comprehensive data frame that contains all the information, I generated two box plots as shown above to visualize the distribution of the ratings, mean of the ratings (denoted as the triangle in the box) and median of the ratings (denoted as the green horizontal line inside the box) for each price indicator. The first and second boxplots are essentially showing the same data, however the second one is only showing boxplot itself without the outliers.

Example: restaurant\_df.boxplot("Ratings",by="Price",figsize=(15,10),showmeans=True,boxprops=boxprops,medianprops=medianprops)

**Statistical Analysis:**

Utilizing the data frame that contains all the data downloaded from Yelp, I extracted the ratings for each price indicator by creating 4 variables, i.e rating1, rating2, rating3 and rating4 and applied the One-Way ANOVA (example code: stats.f\_oneway (rating1, rating2, rating3, rating4).

The result of ANOVA with P-value pvalue being 1.7936651013128483e-07 indicates that the average ratings given by yelp users to the restaurants different based on the price indicators of the restaurants.



**Objective:**

Testing the null hypothesis that the average number of reviews in Yelp are no different between restaurants located in Urban and Rural. Urban is defined as restaurants located in California and New York, and Rural is defined as restaurants located in other states. The average number of reviews could be interpreted as the degree of popularity of the customers towards the restaurants.

**Data Management and Plotting process:**

Utilizing the function boxplot() embedded in Pandas library and the comprehensive data frame that contains all the information, I generated two box plots as shown above to visualize the distribution of the reviews counts, mean of reviews counts (denoted as the triangle in the box) and median of the reviews counts (denoted as the green horizontal line inside the box) for urban and rural. The first and second boxplots are essentially showing the same data, however the second one is only showing boxplot itself without the outliers.

Example:

city\_urban\_t\_all=city\_df.boxplot("Reviews",by="State",figsize=(12,10),showmeans=True,boxprops=boxprops,medianprops=medianprops)

**Statistical Analysis:**

Utilizing the data frame that contains all the data downloaded from Yelp, I extracted the review counts for urban restaurants and rural restaurants and performed the t-test. (example code: stats.ttest\_ind(metro, rural, equal\_var=False)

The result of T test with P-value pvalue being 0 indicates that the average review counts given by yelp users to urban and rural restaurants are different. Urban customers are more likely to provide inputs to the restaurants they visit.