Practical Data Science
Final Project
Group 13

James Hu
Miller Ke
Jung Lee

Exploring Yelp's Treasure Trove of Data

INTRODUCTION

Yelp is a massive data aggregator of restaurant data. Founded in 2004, it now has an estimated 117 million monthly unique visitors, with over 47 million reviews of local businesses. They're also a major community, doing events for their community and fostering discussion within their forum. They're well known for providing restaurant suggestions to users to find the latest and greatest restaurants.

We selected Yelp based on their data feature-set of restaurant reviews and reviewers. Not only do they have a lot of restaurants, but they also have a lot of information that we can parse, with a very rich data set of many attributes of restaurants. In particular, we were interested in restaurants with reviews and the Yelp Elites, those in the Yelp community that possess the following traits:

Authenticity: you are a real person and you keep things real
Contribution: more reviews the better
Connection: review other reviewers

Does acquiring the Elite status have an impact on how Yelpers rate restaurants? Is there a measurable difference in ratings between the Yelp Elite and the general audience? Is one a better predictor of a restaurant's overall rating?

In addition to comparing the different types of users on Yelp, we wanted to get an understanding of what Yelp's data tells about restaurants in general in NYC. Do some restaurants possess certain attributes that indicate their ratings? Can restaurateurs take action on Yelp to possibly drive higher ratings?

DATA CAPTURE AND PROCESSING

So while Yelp has a lot of data, most of the data was locked in the site itself. Yelp provides its own API, which has some data, but it is extraordinarily limited compared to the data on the site. It only provides snippets of the first 3 reviews, deal information, coordinate information and overall rating. The information is incredibly limited from the API, so the decision was made to develop a scraper to traverse their site.

Unfortunately as found out later, this is technically against Yelp's Terms of Service. There was also information found that indicated that Yelp very aggressively pursued scrapers and IP banned them, so that forced us to be relatively careful, and would limit the extent of how much data we could reasonably and reliably scrape.

This was done using python with Beautifulsoup and urllib2. Beautifulsoup is a powerful python library that can analyze HTML and parse certain tags and their information with ease. urllib2 was used to open the sites for processing of the data, so Beautifulsoup could then strip it down the the essential data.

To store this data, it was decided to use csv file format. CSV is a relatively easy to use, easy to transport and easy to set up data source. While it does lack the power of a more powerful database like SQL, the data is easier to setup, easier to manipulate, and easier to transfer between the group.

Our idea to scrape through the site was to go to a search page, process that search page and get all the restaurants, process all the restaurants, then go to the next page, and continue until all the data was processed.

So first we had to quantify how we would scrape through the site. Initially we looked at doing a blanket search for restaurant in Manhattan, but there was an issue. The maximum number of results that Yelp would show was only up to 1000 results. That size data set would be hard to analyze at a large scale and could prove to extremely bias our results to the best restaurants instead of across all of Manhattan. Given this limitation, a factor was found that could better limit the result sets.

The neighborhood filter was able to strictly define and limit the amount of restaurants displayed to only those in that region. This would increase the number of restaurants by sectioning off locations, then processing all the locations. In the end this resulted in a total of around 8000 unique restaurant results.

So to iterate through all of these, for loops were used, with this URL structure.

```
searchurl =
"http://www.yelp.com/search?find_desc=restaurants&start=" +
```

James Hu
Miller Ke
Jung Lee

```
str(num*10) + "&l=p:NY:New_York:Manhattan:" + searchLocation
```

So for example for Greenwich Village:

http://www.yelp.com/search?find_desc=restaurants&start=0&l=p:NY:New_York:Manhattan:Greenwich_Village

Would give the first page of search results that are located within Greenwich Village.

So then the loop would iterate through, the URL would search for restaurants, start at page num, and iterate through the list of neighborhoods (searchLocation).

```
searchLocations =
['Alphabet_City','Battery_Park','Chelsea','Chinatown','Civic_Cent
er','East_Harlem','East_Village','Financial_District','Flatiron',
'Gramercy','Greenwich_Village','Harlem','Hell\'s_Kitchen','Inwood
','Kips_Bay','Koreatown','Little_Italy','Lower_East_Side','Manhat
tan_Valley','Marble_Hill','Meatpacking_District','Midtown_East','
Midtown_West','Morningside_Heights','Murray_Hill','NoHo','Nolita'
,'Roosevelt_Island','SoHo','South_Street_Seaport','South_Village'
,'Stuyvesant_Town','TriBeCa','Two_Bridges','Union_Square','Upper_
East_Side','Upper_West_Side','Washington_Heights','West_Village']
```

The list of searchLocations covers all of Manhattan. While the full list is slightly longer, including Theater District and Yorkville, these were completely inside the regions of Midtown West and Upper East Side respectively, so we deemed them redundant.

Iterating through the search results, we would find the listings, and then go through each of the individual listings. The listings would be structured as "/biz/gotham-bar-and-grill-new-york" so we made sure to append yelp.com to it so it would process.

```
"http://www.yelp.com" + listingurl
```

This was then a restaurants yelp page, with all the standard information.

Now that we had this, we used a standard page with multiple factors we wanted to look at to try and understand the coding sturcture:

James Hu
Miller Ke
Jung Lee

http://www.yelp.com/biz/gotham-bar-and-grill-new-york

Using this page and looking at the source code, multiple factors were identified and chosen to be added to the data collection. Conveniently enough, Yelp uses significant class and id variables which made it relatively easy for BeautifulSoup to iterate through and discover the information we needed. To confirm the variables available, we chose a wide variety of restaurant pages were checked to see if they had any other relevant information. With that, some new variables were discovered and added into our scraper. After that, code was written to take the information off the page. This information was then written to a row of a restaurants CSV.

```
fr.writerow([resturl, title, latitude, longitude, rating,
reviewCount, categories, photos, URL, neighborhood, menu,
reservable, yelpDelivery, slides, sponsor, claim, eliteReviews,
transit, hours, attire, creditCards, parking, price, groups,
kids, reservations, deal, delivery, takeout, service,
outdoorSeating, wifi, meals, bestNights, happyHour, alcohol,
smoking, coatCheck, noise, goodForDancing, ambience, tv, caters,
wheelchairAccessible])
```

Since reviews were also listed on a restaurants page, the reviews were parsed later in the code. The reviews segment was identified, common factors were pulled then it would be written to a reviews CSV.

```
frev.writerow([resturl, eliteStatus, friendCount, reviewCount,
userPhoto, reviewInfo, reviewRating, publish, description,
reviewPix, reviewSeated, reviewDeal, reviewCheckIn, useful,
funny, cool])
```

We chose not to parse all the reviews and just the reviews on the initial page. This was because the amount of time required to parse through every single page of reviews for every restaurant (including Ippudo with 5000+ reviews!), would be immense, and would make us a bigger target for IP ban.

Given that, we focused on two locations and worked on parsing all the data.

Debugging

Our initial pass throughs worked, but there were some significant issues that had to be addressed before we could use the code reliably to scrape information off of Yelp's site.

Making sure viable restaurant data is acquired

We made sure of this by running it on multiple restaurant pages and double checking the results. Many of the bugs were ironed out through strict checking with the data set.

Preventing errors from significantly slowing down data collection

Initially the plan was to run the code at once and parse all the information. This was eventually seen as unviable. Any coding error or internet connection drop would invariably crash the program which would then stop the data collection. To help mitigate this problem, the CSV files were subdivided into their neighborhood.

```
filenamer = "yelpr_" + searchLocation + ".csv"
filenamerev = "yelprev_" + searchLocation + ".csv"
```

This then allowed for easier recovery if a crash happened and identification of stop locations, preventing one major error from potentially complicating or corrupting the data set.

Preventing anomalies from stopping data collection

This proved to be a significant part of initial runs. Occasionally anomalies would occur out of the blue due to semi-inconsistent pages or what seemed to be AB Testing strategies.

One of the first ones encountered was a completely new Yelp style which was extremely frustrating to code through as the error message nor the page could indicate what was wrong. This is when one of the first AB Tests were identified. Instead of the same structure that Yelp restaurant pages normally used, it would occasionally become this:

James Hu
Miller Ke
Jung Lee

This would then proceed to break and error the script. This was resolved with a try catch statement to identify what the page looked like and its characteristics, then a if statement was placed so that if an identifying characteristic of the alternate page was found, it would run the function again until it switched. The occurrence of the alternate site seemed to be 1 out of 50, so it was unreliable to find manually through the browser.

Issues after this became significantly easier to debug as they were more immediately visible.

The next issue that arose was occasionally a site with a rating, but no reviews would appear. This was because Yelp acquired Qype, which had many reviews in alternate languages but would not display. This was fixed through a check at the beginning of the script.

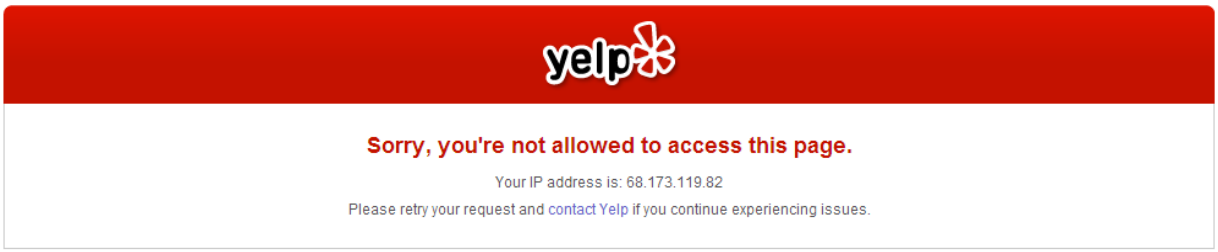
Another issue very much tied to that was that Qype lacked certain data points that the script was searching for. This was resolved by using if statements and setting default values.

The last issue was that there was another incidence of AB testing. Instead of a massive change, it would subtly change one of the classes my code was looking for into something else. Fortunately, the data would error instead of giving bad data, which made detection relatively simple. Using the try catch code again, it was relatively easy to identify the code that was different. This happened about 1 every 15 pages.

<u>Avoiding IP Ban</u>

This was perhaps the biggest issue that had to be dealt with. There were at least 4 IP bans from Yelp for scraping their site.

James Hu
Miller Ke
Jung Lee



The data processing already took a significant amount of time, with each restaurant page scrape taking around 5 seconds, so I was under the impression that this would potentially give enough downtime to not receive an IP ban. I was mistaken.

The first IP ban was rather sudden. After scraping for two hours, the script would throw an error and stop immediately. After researching, it seemed that Yelp aggressively protected their data against scraping. According to a Yelp employee, they even went as far as to completely ban Tor since users were disproportionately using Tor to scrape Yelp.

In order to continue to collect data, I had to circumvent the IP Ban. A MAC Address clone was used to generate new IPs and continue scraping.

Unfortunately this was found unsustainable too. Subsequently the script kept getting IP banned faster, in about 1 hour, resulting in data collection stoppage. In addition, the sustainability of constantly refreshing the IP was not guaranteed so a longer term solution had to be developed.

Eventually, through the built in time module of Python, sleep states were implemented into the code. With the IPs still getting banned, the numbers had to be fairly high and were incremented up to 1 minute per search page, 10 minutes per neighborhood and on average 5 seconds wait per restaurant page.

With this still not working, a proxy was attempted.

```
import urllib2

proxy = urllib2.ProxyHandler({'http': '173.213.113.111:8089'})
opener = urllib2.build_opener()
opener.addheaders = [('User-agent', 'Mozilla/5.0 (X11; Linux
i686; rv:5.0) Gecko/20100101 Firefox/5.0')]
```

```
urllib2.install_opener(opener)
```

While this script proved successful, when applied to Yelp, very often it would result in the home IP being used, or the proxy being denied on Yelp, so that was not a feasible solution.

In the end though, this did help identify a key feature that was added to the code that made it possible to run for a long time.

urllib2's default user-agent was Python/urllib2, which was extremely easy to determine that most likely the user was trying to scrape the site. This was incremented to Firefox and Chrome also. While they lasted longer, the code was not able to perpetually run without getting IP banned.

In the end, the most successful user-agent was IE9.

```
opener.addheaders = [('User-agent', 'IE 9/Windows: Mozilla/5.0
(compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)')]
```

This allowed the code to run for 20 straight hours scraping Yelp. This strategy allowed the information to be scraped to completion.

Data Cleanup

The data was now a little unformatted and split across the various neighborhoods.

To merge the data, a script was written, concat.py, to merge all the data together. Simply it would read through the lines and output them to a new document. This resulted in a restaurant document with 10,000+ restaurants and 277,000+ reviews.

Restaurant sample data:

```
http://www.yelp.com/biz/pylos-new-york,Pylos,40.7260964,-
73.9841525,4.5,597,Greek,Has photos,pylosrestaurant.com,"['East
Village', 'Alphabet City']",Has menu,None,None,None,None,None,Has
Elites,"2 Ave. (F)
1 Ave. (L)
Astor Pl (4, 6, 6X)","Mon-Thu, Sun 5 pm - 12 am
```

```
Wed-Sun 11:30 am - 4 pm
Fri-Sat 5 pm - 1
am",Casual,Yes,Street,$$$,Yes,No,Yes,None,No,Yes,Yes,No,No,Dinner
,None,None,Beer & Wine
Only,None,None,Average,None,Romantic,No,No,Yes
```

Review sample data:

```
http://www.yelp.com/biz/pylos-new-york,None,0,9,Has photo,"East
Village, Manhattan, NY",5.0,2013-11-07,"A
Boutique...restaurants.",None,None,None,None,1,0,0
```

Then another script was written to sterilize the data, to make all the relevant binary into numbers instead of their values. Using input from team members, two scripts, rsterilization.py and revsterilization.py, were created and used to sterilize the data to a format that can easily be used in data manipulation programs such as Tableau etc.
For the last step, the removal of duplicates, the decision was to allow Excel to do the processing. While it is possible to write a python script to manipulate the documents and remove duplicates, the overhead, time and difficulty to do so is excessive and there was little information on removing duplicates quickly. So the decision was made to use Excel to expedite the process. This then cut down the restaurants to 8,600+ and reviews to 220,000+.

One additional step then taken was to remove restaurants with under 10 reviews. The data for restaurants with under 10 reviews was very questionable and was not nearly as accurate as it should be. Therefore another script, g10.py, was used to remove any restaurants with less than 10 reviews. This reduced it down to 6,200+ results.

VISUALIZING RESTAURANT FEATURES

Histograms

To get a sense of the data we were working with, we plotted a few histograms in Python, starting with ratings (see Visual 1 in Appendix). What we see is somewhat of a normally distributed bell-curve, but with a longer left tail and a right-biased peak. People tend to be more positive with their ratings overall but there is still some hating going, which anecdotally makes sense. You really need to have a polarizing experience to want to rate a restaurant. Yelp

has an additional variable with the Elite factor, which will be explored later on.

Next, we plotted the frequency of restaurant's price range. Originally, we counted all restaurants in this plot, which showed a usually large amount of $$$$ restaurants. As we dove deeper into the dataset, we discovered that there were hundreds of restaurants with only a single digit count of reviews, all around East and West Harlem, which were skewing our analysis.

Yelp apparently allows users to calculate the price range for the restaurant, if the restaurant has not been "claimed" by the owner. To offset this, we filtered out all restaurants that had less than 10 reviews for all future analysis. Upon filtering the data, we saw again a fairly normally distributed histogram of restaurants by price range, with the majority of restaurants in the $$ category (see Visual 2 in Appendix).

Identifying Significant Variables

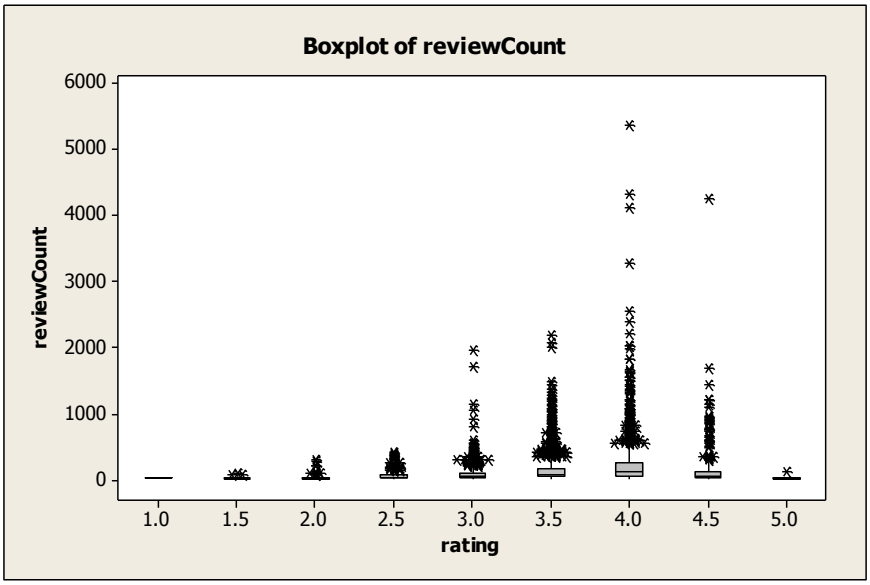
We began our analysis with ratings as the target variable and price range as the initial independent variable. But we quickly came to the conclusion that a restaurant's price range does not necessary drive ratings (see Visual 3 in Appendix). The original hypothesis arose from the notion of confirmation bias - in order to justify spending more money for an expensive restaurant, Yelpers would tend to rate expensive restaurants higher. While we do see that the more expensive restaurants have slightly higher average ratings, they were not statistically significant.

To find additional relationships within Yelp's data, we looked for other features that were strongly correlated to ratings. We ran a few linear regression models in Python and Minitab to identify variables that showed stronger correlations and offer us some better visuals. Out of the approximately 40 features for restaurants, we found three - the total count of reviews for each restaurant, whether or not a restaurant's Yelp page was claimed by the restaurateur and whether or not the restaurant offered deals.

Boxplots

To look at the relationship between ratings and the count of reviews, we plotted a few boxplots in Minitab and Python.

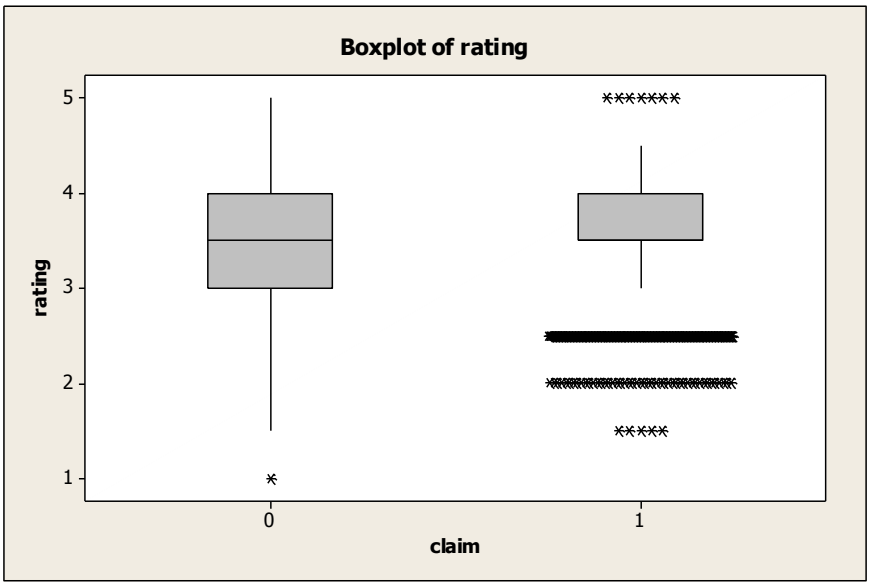
James Hu
Miller Ke
Jung Lee



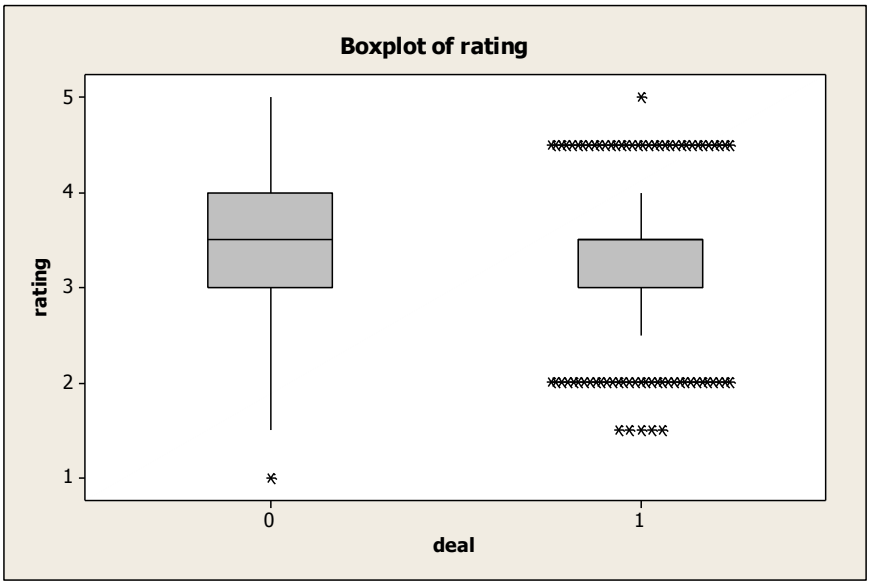
The average count of reviews that a restaurant has is around 140. And as you can see from the boxplot above, the number of reviews is fairly consistent across all ratings levels. However, a greater number of restaurants with higher ratings have extreme review counts, peaking at 4. Ippudo (a ramen restaurant) ranked first in the count of reviews at 5,381. The extreme review counts are from destination restaurants like Katz Deli and Shake Shack. But it mirrors the overall distribution of rating seen in the histogram.

Looking at the boxplot comparing ratings for restaurants that have been claimed by the owner and restaurants that have not been claimed, we see claimed restaurants have an overall tighter range in the ratings they receive with a slightly higher average. Claimed restaurants also have a greater number of outliers.

James Hu
Miller Ke
Jung Lee



Finally, looking at the boxplot of restaurants that offer deals vs. those that do not below, we see a similar distribution of no deals to no claims, with a broad range of ratings and a average of 3.5. However, restaurants that offer deals appear have a narrower band of opinion driving a slightly lower rating with a large number of outliers on both sides. Further analysis would be required to identify trends in both the positive and negative outliers.



Heatmaps

One of the variables we were interested in was whether or not higher rated restaurants were more heavily clustered around certain locations. Does the location make the restaurant or will people flock to wherever a good restaurant resides?

Given the cost of leasing space in New York City, neighborhoods most likely have significant biases in terms of pricing and awareness that draws certain types of people. This is probably one of the reasons why you won't find many ramen joints around Columbus Circle or Tribeca. But you will find plenty down in the East Village.

Using Tableau, we created heatmaps to plot out the location of restaurants using their geo-coordinates captured from the Yelp data-scrape and compared the distribution of restaurants by ratings, price, review count and Yelp page claimage.

Ratings vs. Price

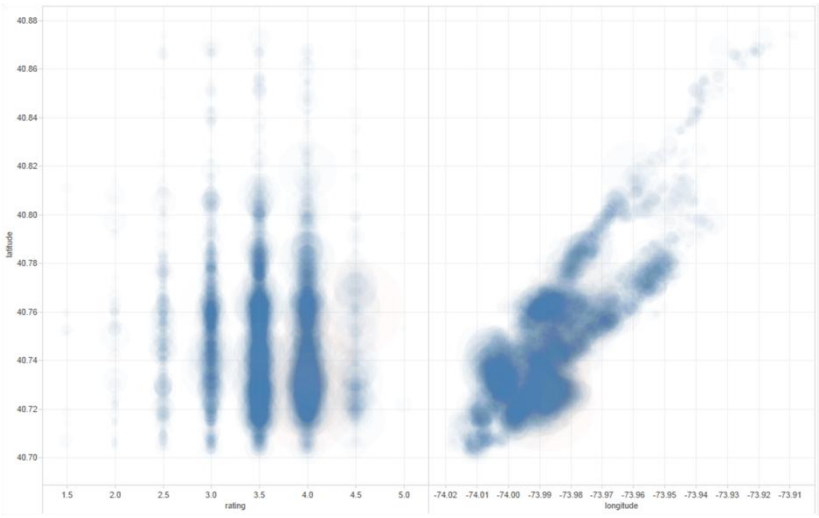
We realized that for restaurant ratings, the 1 - 5 scale did not offer a lot of depth for us to create a refined heatmap. What we saw was a blurry distributed blob of restaurants (see Visual 4 in Appendix).

We then created small multiples of the heatmap by the price category (see Visual 5 in Appendix). We saw a similar representation of the distribution of restaurants in the histogram of price above where the majority of restaurants fell into the $$ category.

Review Count vs. Rating

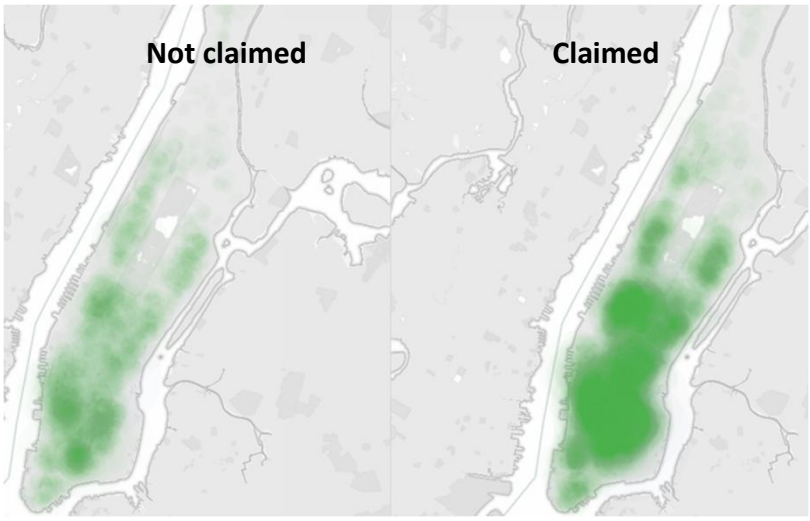
Since review count was significantly correlated to ratings, and because it had a greater range of data, we plotted that against rating. This heatmap shows that restaurants with a 3.5 | 4 garner the greatest number of reviews.

James Hu
Miller Ke
Jung Lee



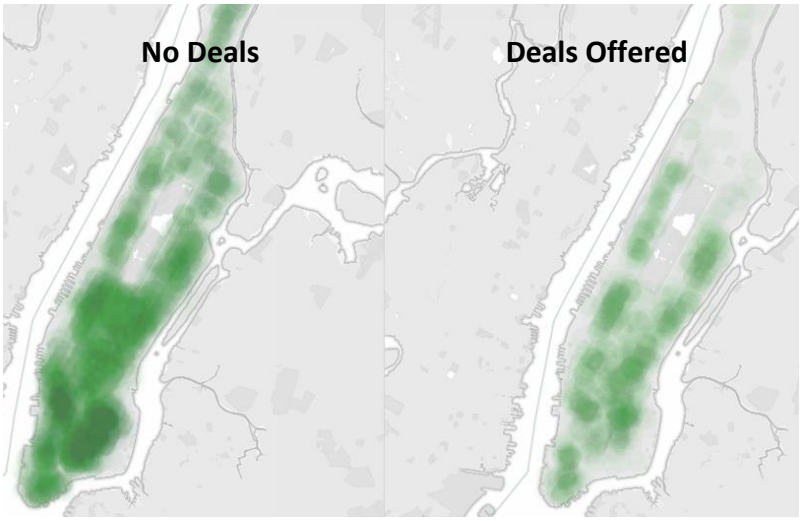
Review Count vs. Claims and Deals

Finally, we looked at a few heatmaps that compared restaurants Claims and Deals status to explore the distribution of review counts on a map.



As we already saw in the boxplots, we see more activity and higher ratings for restaurants that have been claimed. But we see less activity and lower ratings for restaurants that offer deals. The former could be due to the restaurant owners' commitment to their establishment and customers which ultimately manifests in Yelp's user ratings. The latter could suggest that restaurants that are not doing well in general are more likely to offer deals in a desperate attempt to capture more customers.

James Hu
Miller Ke
Jung Lee



YELP ELITES

Our second area of insight we looked into was the potential relationship between Elite Status review ratings, Non-Elite Status review ratings, average review ratings, and other user criteria. The original impression we had was that Elite Status Yelpers were considered to be foodies and would thus judge food more critically. Non-Elite Status Yelpers were considered to be voluntary reviewers that had a polarizing experience at the restaurant. Thus, we believed that Elite Status Yelp review ratings would more accurately represent the average rating of the restaurant.

Obtaining Ratings

After collecting the data from Yelp we needed a method to easily analyze the data. We decided on using the pandas library to do the analysis. The pandas library allowed us to easily import the data into a DataFrame and call the .describe method to give us relevant statistical information. We primarily decided to look at the mean rather than the median since the presence of integer values trivialized the importance of the median. In addition we decided to plot our data using side by side box and whisker plots based on Elite Status in order to easily compare the distribution between the two groups.

Aggregate and Group Ratings

There were a total of 48,853 entries under Elite Status and 115,915 entries under Non-Elite Status. The mean review rating for Elite Status was 3.454273 with a standard deviation of 1.295525 and the mean review rating for Non-Elite Status was 3.475795 with a standard

deviation of 1.003664. In addition, the average review rating is 3.460654 with a standard deviation of 1.216352 . We expected the average rating to be closer to 3.0, but this was not the case.

Aggregate Rating Interpretation

Looking at the difference between the number of Elite Status entries and Non-Elite Status entries did not surprise us. We knew that the Elite Status was supposed to carry some prestige and that there would likely be more Non-Elite Status reviews. The average aggregate review rating of 3.460654 is interesting due to its distance from 3.0. On the Yelp review scale from 1-5 one would naturally assumed that the average should be 3.0. A higher value has many implications. One conclusion may be that the restaurants being reviewed are simply on average better than the norm. New York City being a renowned location for food does not put this out of the realm of possibility, however without data about other regions it is difficult to validate this conclusion. Another conclusion may be that consumers tend to post reviews if they are satisfied with the restaurant. This promotes the idea of the obvious volunteer bias when looking at reviews. However, it suggests that there is a skew towards positive experiences rather than negative experiences. The standard deviations being close to 1 show that the distribution of ratings adheres in some capacity to the 1-5 scale. This means that the mean review rating is not arrived at through a combination of 1s and 5s. Rather it suggests a rough bell distribution centered around 3.460654.

Group Ratings Interpretation

The staggering similarity between the group ratings lead us to believe that there is little difference between Elite Status Yelper reviews and Non-Elite Status Yelper reviews. A box and whisker plot shows that the overlap between the two is very apparent [6]. Our original belief that Non-Elite Status Yelpers had relatively more polarizing views turned out to be incorrect since the standard deviation is lower than that of Elite Status Yelpers. This suggests that there is less variance in Non-Elite Status Yelp reviews. The similarity between these two pools of users pointed us in the direction that the distinguishing factor between them was simply quantity. If Non-Elite Status Yelpers match Elite Status Yelpers in their reviews, then the only difference between the two are the number of reviews they have written. Rather than thinking about it as Elite vs. Non-Elite it is more of an analysis into all of the review ratings on Yelp. A new user on Yelp will on average arrive at the same rating for a restaurant that a seasoned user will. This is important because it detracted from the notion that Elite users were foodies. In fact, users who

post reviews on Yelp are more homogenous than we had originally thought.

Dealing with Bias and Relevancy

There are a few significant instances of bias in our data's collection and processing. We opted not to scrape every review for a restaurant and capped the reviews for a given restaurant at 40. This took the most recent 40 ratings, which means that the data may fail to capture changes in management or policy. This sampling is not random and adds some bias to our method in collecting data. The issue of volunteer bias is also ever present in reviews as people only tend to write a review if they had a significant enough of an experience to care about. There is no real way around this and we accept that the conclusions we draw are formed around Yelp's voluntary user base.

The issue of relevancy is also difficult to address due to the scope of the data. By processing the data in an aggregate capacity we ignore what restaurants the reviews came from and instead are looking at a review of the overall quality of restaurants in Manhattan. We can instead opt to randomly sample restaurants that have a significant amount of Elite vs. Non-Elite reviews in order to gain a better picture of the difference between these two groups. This method is also prone to bias, since restaurants that have enough reviews to constitute being used are likely going to be popular. Rarely are there any poor quality restaurants with a high amount of reviews. Thus, our conclusions drawn are only relevant to the single restaurant in question.

Comparing Reviews for a Single Restaurant

After understanding how broad the scope was of our previous analysis, we decided to analyze a single restaurant in order to preserve the same topic of review between an Elite and Non-Elite review. Our randomly selected restaurant needed to have a significant amount of both Elite and Non-Elite reviews. We ended up arriving at Silom, a Thai restaurant in Chelsea. The restaurant had 35 Elite reviews and 73 Non-Elite reviews. The average review rating for the restaurant was 2.805556. The average Elite review rating was 2.571429 with a standard deviation of 1.092372. The average Non-Elite review rating was 2.890411 with a standard deviation of 1.161436. Immediately we see there is a significant difference between the two group ratings. The box and whisker plot[7] shows us that distribution of values for the two groups vary drastically. The lower average rating from Elite reviewers coincides with our original belief in a more critical group of users. On the other hand, the Non-Elite reviewers found Silom to be slightly better. Interestingly enough we saw that there we no ratings of 5 from Elite reviewers for Silom.

Further Improvements and Goals

The results from Silom are only an example of a single instance. While we cannot draw any broad conclusions from it, the groundwork is laid for a more expansive forage into the Yelp data. In order to better judge if there is a significant difference between Elite and Non-Elite reviewers, we would expand the data scraping to encapsulate more than 40 reviews. Furthermore, we would only select restaurants that had n>30 for both Elite and Non-Elite reviews. From here we could look at the difference between the two group's reviews relative to the average restaurant rating. Conducting a hypothesis test to test whether or not there is a significant difference between the two average ratings would then most accurately answer our question.

We could also combine both of our insights and investigate the relevance of location and rating. The location of a restaurant is extremely important, since Yelp users tend to review restaurants that are easily accessible to them. Thus, an additional layer of bias arises due to a non-random selection of Yelp users reviewing the restaurant. This relates to the possibility of restaurants in certain areas may be more heavily reviewed by Elite users due to the demographic of the location. Looking at the crossover between location and user demographic could lead to interesting insights about the relationship between a restaurants rating and its area. There is also the idea that Elite users may be bias towards reviewing higher quality restaurants. We could investigate this by looking at the distribution of the amount of Elite reviews against the average rating of the restaurants.

Predicting User Status

We also decided to see if there were any correlations between user review qualities and user status. By looking at if pictures were taken, the user checked-in, and how useful, funny, or cool the review was we attempted to predict if the user was an Elite. The first criteria we looked at was whether or not the user incorporated pictures in their review[8]. We saw that on average, an Elite user submitted 0.458672 pictures with their review while a Non-Elite user only submitted 0.163200 pictures. While the standard variation is extremely high relative to the averages, this is understandable due to reviewers usually taking many pictures should they decide to do so. The same was true for check-ins as well. Elite users tended to check-in at every 0.460614 locations they reviewed. On the other hand, Non-Elite users only checked-in at 0.176531 locations they reviewed. While the standard deviations were high again due to checking-in being a binary value, it showed us that Elite users were more prone to checking in over Non-Elite users.

We also found that reviews marked as useful, funny, or cool had a higher tendency to be from an Elite user. In all categories, an Elite user's review on average had higher points. Elite users on average had 1.645850 useful points, 0.860063 funny points, and 0.965838 cool points per review. Non-Elite users on average had 1.013697 useful points, 0.443200 funny points, and 0.449872 cool points. The standard deviation values for each category were very high at 2.746813 for useful, 2.295888 for funny, and 2.156238 for cool for Elite users and 2.298212 for useful, 1.517773 for funny, and 1.671170 for cool for Non-Elite users. This again suggests a polarizing effect where reviews are considered very useful, funny, or cool or not at all.

Predicting Rating via Review Qualities

The last information we looked at was whether or not there was correlation between the characteristics of a review and the rating given[9]. We first looked at average rating vs. average review pictures taken. The scatterplot shows a nice positive linear trend between the two criteria[10]. This suggests that as average rating increases, average review pictures taken increases as well. The next comparison we looked at was average rating vs. average check-in rate[11]. This presented a bell-curve which led us to two possible conclusions. One was that the relation between the two followed a bell-curve. The other was that there is no relation since with only 5 points to look at it is possible the points are simply random. Finally, we looked at average ratings v. total useful/funny/cool ratings. The scatterplot looked like an inverse bell-curve[12]. We followed the same conclusions from our previous comparison in that it was difficult to say if the relationship followed a true inverse bell-curve or was just random.

WHAT THIS MEANS FOR YELP

So given this data we've received, we can make some conclusions and perhaps suggest Yelp down a path.

A significant finding is that it seems like there are strategies that restaurants can use to increase their ratings, with some of the factors being important to the improvement of Yelp score.

One factor of particular important that we noticed that the regressions showed that if the Yelp Restaurant has a deal, it is negatively correlated with their review rating. Part of Yelp's business is to provide these deals and redirect people to them, but perhaps it needs to be understood and perhaps tweaked so that restaurants can see real, positive results from using Yelp's deal system, which would help improve Yelp revenue and usage. Given that restaurants with deals

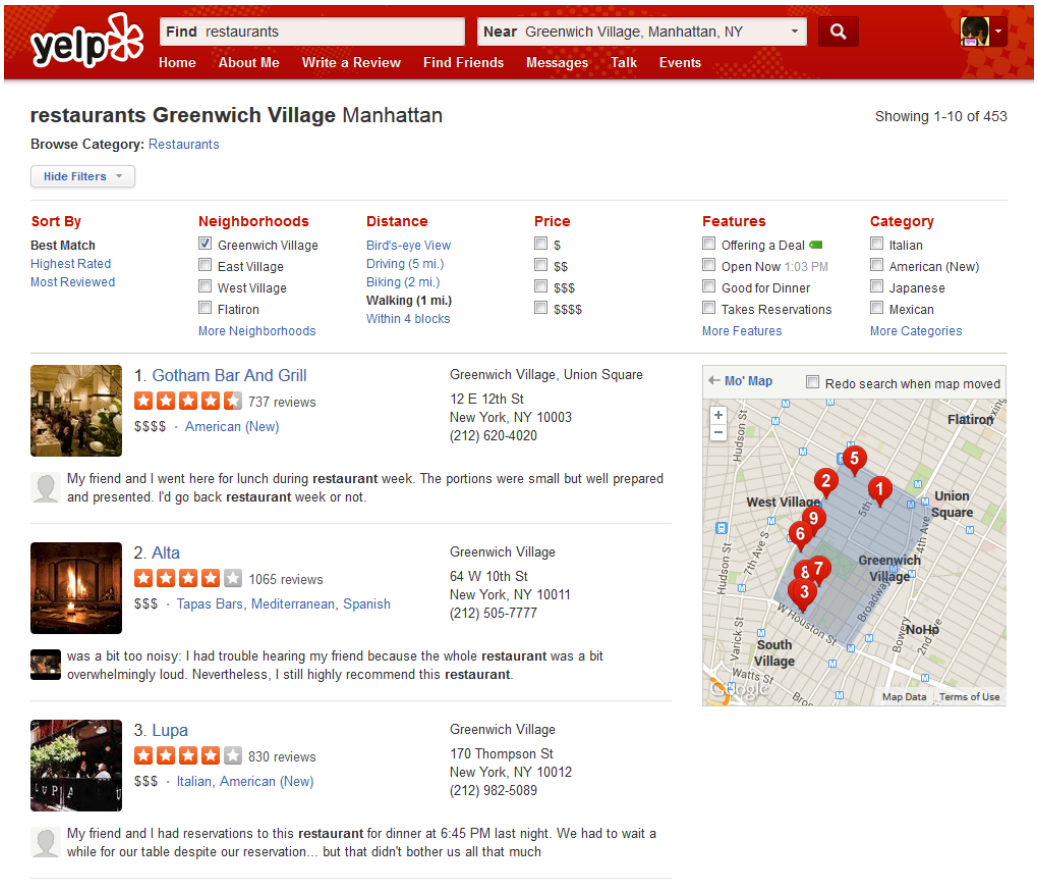
are on average worse, doesn't help the average Yelp user's perception of the restaurant.

We also believe from the statistics we found that perhaps at a larger scale at Yelp with proprietary data, there is an opportunity for Yelp to create a consulting type of business. They have the data necessary to potentially better mine out factors. Given our relatively limited data set (only New York City), and the data collection limitations we had, we still were able to find fairly significant data results that Yelp could use to help restaurants improve their score and thusly improve their restaurants.

James Hu
Miller Ke
Jung Lee

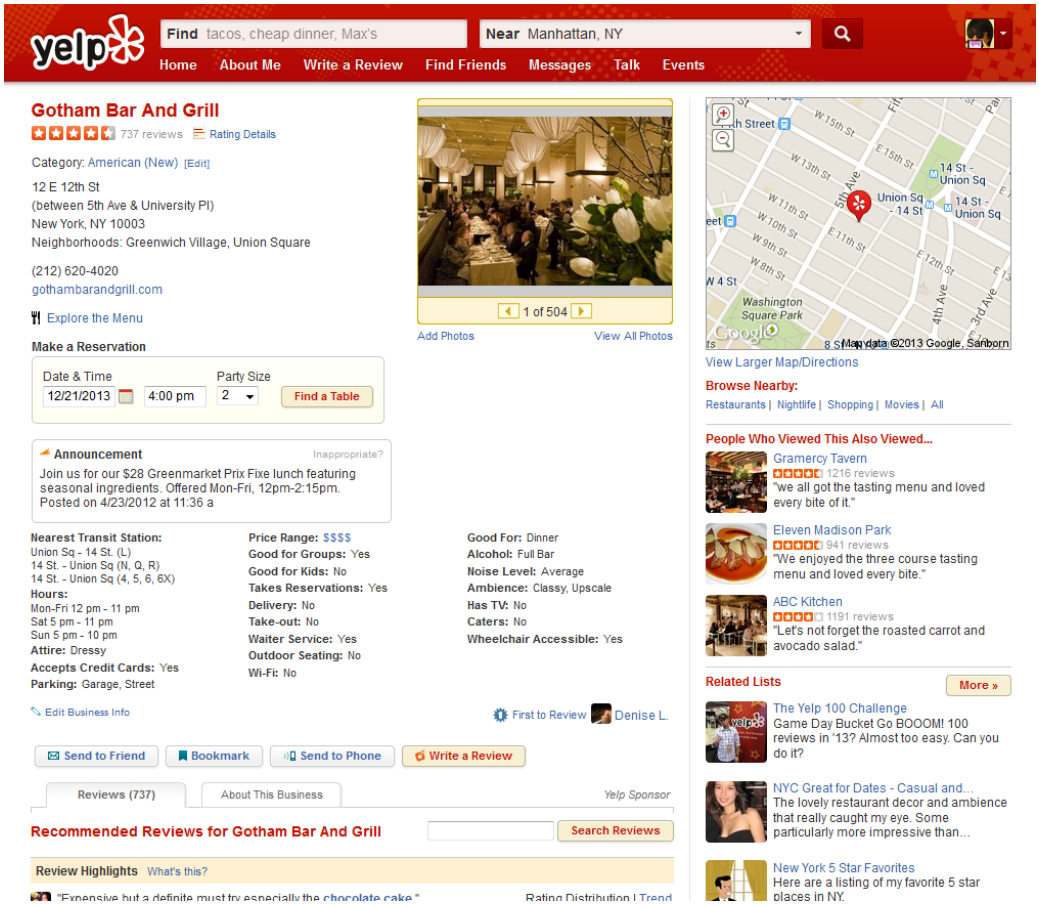
APPENDIX

http://www.yelp.com/search?find_desc=restaurants&start=0&l=p:NY:New_York:Manhattan:Greenwich_Village



http://www.yelp.com/biz/gotham-bar-and-grill-new-york

James Hu
Miller Ke
Jung Lee



James Hu
Miller Ke
Jung Lee

Code

Yelp.py

```
from bs4 import BeautifulSoup
import urllib2
import csv
from pyparsing import Literal, quotedString, removeQuotes, delimitedList
import time
from random import randint


# Scrape restaurant information
def restscrape(resturl, filenamersc, filenamerevsc):

    time.sleep(randint(2,8))
    # Read the url
    response = urllib2.urlopen(resturl)
    soup = BeautifulSoup(response.read())
    response.close()


    # Check if it is rated
    if soup.find(itemprop="ratingValue") == None:
        return

    # Anamoly
    if soup.find(class_="container no-reviews") != None:
        return

    # Check if it is not the alternate version
    if soup.find(id="mapbox") != None:
        print "alt version"
        restscrape(resturl, filenamersc, filenamerevsc)
        return

    # Check if it is not an alternate version
    if soup.find(class_="friend-count miniOrange") == None:
        print "alt version rev"
        restscrape(resturl, filenamersc, filenamerevsc)
        return

#### ##    ## ########  #######
 ##  ###   ## ##       ##     ##
 ##  ####  ## ##       ##     ##
 ##  ## ## ## ######   ##     ##
 ##  ##  #### ##       ##     ##
 ##  ##   ### ##       ##     ##
#### ##    ## ##        #######

    # Key Yelp information
    title = soup.find(property="og:title").get("content").encode('utf-8')
    latitude = soup.find(property="place:location:latitude").get("content")
    longitude = soup.find(property="place:location:longitude").get("content")
    rating = soup.find(itemprop="ratingValue").get("content")
    reviewCount = soup.find(itemprop="reviewCount").get_text()
```

```
53.
54.     if soup.find(id="cat_display") != None:
55.         categories = soup.find(id="cat_display").get_text().strip()
56.         categories = ' '.join(categories.split())
57.     else:
58.         categories = "None"
59.
60.     if soup.find(class_="photo-box-img")['src'] != "http://s3-
    media1.ak.yelpcdn.com/assets/2/www/img/5f69f303f17c/default_avatars/business_medium_squ
    are.png":
61.         photos = "Has photos"
62.     else:
63.         photos = "None"
64.
65.     if soup.find(id="bizUrl") != None:
66.          URL = soup.find(id="bizUrl").get_text().strip().encode('utf-8')
67.     else:
68.         URL = "None"
69.
70.     # Get Neighborhoods
71.     # Particularly special code because it has to be stripped from javascript script
72.     # automatically strip quotes from quoted strings
73.     # quotedString matches single or double quotes
74.     neighborhood = ""
75.     quotedString.setParseAction(removeQuotes)
76.
77.     # define a pattern to extract the neighborhoods: entry
78.     neighborhoodsSpec = Literal('\"neighborhoods\":') + '[' + delimitedList(quotedStrin
    g)('neighborhoods') + ']'
79.
80.     for hoods in neighborhoodsSpec.searchString(soup):
81.         neighborhood = str(hoods.neighborhoods)
82.
83.
84.     # Yelp Interaction/Information
85.     if soup.find(class_="yelp-menu") != None:
86.         menu = "Has menu"
87.     else:
88.         menu = "None"
89.
90.     if soup.find(id="opentable-reservation-actions") != None:
91.         reservable = "Reservable"
92.     else:
93.         reservable = "None"
94.
95.     if soup.find(class_="media-story offer-detail") != None:
96.         deal = "Has deal"
97.     else:
98.         deal = "None"
99.
100.    if soup.find(id="delivery-address-form") != None:
101.        yelpDelivery = "Delivery system"
102.    else:
103.        yelpDelivery = "None"
104.
105.    if soup.find(id="bizSlide") != None:
106.        slides = "Has slides"
107.    else:
108.        slides = "None"
```

```
109.
110.
111.    # Restaurant status
112.    if soup.find(id="bizSupporter") != None:
113.        sponsor = "Sponsors"
114.    else:
115.        sponsor = "None"
116.
117.    if soup.find(id="bizClaim") != None:
118.        claim = "Unclaimed"
119.    else:
120.        claim = "None"
121.
122.    if soup.find(style="color:#999999;") == None:
123.        eliteReviews = "Has Elites"
124.    else:
125.        eliteReviews = "None"
126.
127.
128.    # Restaurant attributes from attributes section
129.    if soup.find(class_="attr-transit") != None:
130.        transit = soup.find(class_="attr-transit").get_text().strip()
131.    else:
132.        transit = "None"
133.
134.    if soup.find(class_="attr-BusinessHours") != None:
135.        hours = soup.find('dd', class_="attr-BusinessHours").get_text()
136.    else:
137.        hours = "None"
138.
139.    if soup.find(class_="attr-RestaurantsAttire") != None:
140.        attire = soup.find('dd', class_="attr-RestaurantsAttire").get_text()
141.    else:
142.        attire = "None"
143.
144.    if soup.find(class_="attr-BusinessAcceptsCreditCards") != None:
145.        creditCards = soup.find('dd', class_="attr-
    BusinessAcceptsCreditCards").get_text()
146.    else:
147.        creditCards = "None"
148.
149.    if soup.find(class_="attr-BusinessParking") != None:
150.        parking = soup.find('dd', class_="attr-BusinessParking").get_text()
151.    else:
152.        parking = "None"
153.
154.    if soup.find(class_="attr-RestaurantsPriceRange2") != None:
155.        price = soup.find('dd', class_="attr-
    RestaurantsPriceRange2").get_text().strip()
156.    else:
157.        price = "None"
158.
159.    if soup.find(class_="attr-RestaurantsGoodForGroups") != None:
160.        groups = soup.find('dd', class_="attr-RestaurantsGoodForGroups").get_text()
161.    else:
162.        groups = "None"
163.
164.    if soup.find(class_="attr-GoodForKids") != None:
165.        kids = soup.find('dd', class_="attr-GoodForKids").get_text()
```

```
    else:
        kids = "None"

    if soup.find(class_="attr-RestaurantsReservations") != None:
        reservations = soup.find('dd', class_="attr-
RestaurantsReservations").get_text()
    else:
        reservations = "None"

    if soup.find(class_="attr-RestaurantsDelivery") != None:
        delivery = soup.find('dd', class_="attr-RestaurantsDelivery").get_text()
    else:
        delivery = "None"

    if soup.find(class_="attr-RestaurantsTakeOut") != None:
        takeout = soup.find('dd', class_="attr-RestaurantsTakeOut").get_text()
    else:
        takeout = "None"

    if soup.find(class_="attr-RestaurantsTableService") != None:
        service = soup.find('dd', class_="attr-RestaurantsTableService").get_text()
    else:
        service = "None"

    if soup.find(class_="attr-OutdoorSeating") != None:
        outdoorSeating = soup.find('dd', class_="attr-OutdoorSeating").get_text()
    else:
        outdoorSeating = "None"

    if soup.find(class_="attr-WiFi") != None:
        wifi = soup.find('dd', class_="attr-WiFi").get_text()
    else:
        wifi = "None"

    if soup.find(class_="attr-GoodForMeal") != None:
        meals = soup.find('dd', class_="attr-GoodForMeal").get_text()
    else:
        meals = "None"

    if soup.find(class_="attr-BestNights") != None:
        bestNights = soup.find('dd', class_="attr-BestNights").get_text()
    else:
        bestNights = "None"

    if soup.find(class_="attr-HappyHour") != None:
        happyHour = soup.find('dd', class_="attr-HappyHour").get_text()
    else:
        happyHour = "None"

    if soup.find(class_="attr-Alcohol") != None:
        alcohol = soup.find('dd', class_="attr-Alcohol").get_text()
    else:
        alcohol = "None"

    if soup.find(class_="attr-Smoking") != None:
        smoking = soup.find('dd', class_="attr-Smoking").get_text()
    else:
        smoking = "None"

```

```
    if soup.find(class_="attr-CoatCheck") != None:
        coatCheck = soup.find('dd', class_="attr-CoatCheck").get_text()
    else:
        coatCheck = "None"

    if soup.find(class_="attr-NoiseLevel") != None:
        noise = soup.find('dd', class_="attr-NoiseLevel").get_text()
    else:
        noise = "None"

    if soup.find(class_="attr-GoodForDancing") != None:
        goodForDancing = soup.find('dd', class_="attr-GoodForDancing").get_text()
    else:
        goodForDancing = "None"

    if soup.find(class_="attr-Ambience") != None:
        ambience = soup.find('dd', class_="attr-Ambience").get_text()
    else:
        ambience = "None"

    if soup.find(class_="attr-HasTV") != None:
        tv = soup.find('dd', class_="attr-HasTV").get_text()
    else:
        tv = "None"

    if soup.find(class_="attr-Caters") != None:
        caters = soup.find('dd', class_="attr-Caters").get_text()
    else:
        caters = "None"

    if soup.find(class_="attr-WheelchairAccessible") != None:
        wheelchairAccessible = soup.find('dd', class_="attr-
WheelchairAccessible").get_text()
    else:
        wheelchairAccessible = "None"

    if soup.find(class_="attr-DogsAllowed") != None:
        dogsAllowed = soup.find('dd', class_="attr-DogsAllowed").get_text()
    else:
        dogsAllowed = "None"


    with open(filenamersc, "ab") as filer:
        fr = csv.writer(filer)
        # Writing to CSV
        fr.writerow([resturl, title, latitude, longitude, rating, reviewCount, categori
es, photos, URL, neighborhood, menu, reservable, yelpDelivery, slides, sponsor, claim,
eliteReviews, transit, hours, attire, creditCards, parking, price, groups, kids, reserv
ations, deal, delivery, takeout, service, outdoorSeating, wifi, meals, bestNights, happ
yHour, alcohol, smoking, coatCheck, noise, goodForDancing, ambience, tv, caters, wheelc
hairAccessible])

########  ######## ##     ## #### ######## ##      ##  ######
##     ## ##       ##     ##  ##  ##       ##  ##  ## ##    ##
##     ## ##       ##     ##  ##  ##       ##  ##  ## ##
########  ######   ##     ##  ##  ######   ##  ##  ##  ######
##   ##   ##        ##   ##   ##  ##       ##  ##  ##       ##
##    ##  ##         ## ##    ##  ##       ##  ##  ## ##    ##
##     ## ########    ###    #### ########  ###  ###   ######
```

```
277.
278.    # Parsing top 40 Reviews
279.    reviews = soup.findAll(itemprop="review")
280.    for review in reviews:
281.
282.        # Get user data
283.        if review.find(title="User is Elite") != None:
284.            eliteStatus = "Elite"
285.        else:
286.            eliteStatus = "None"
287.
288.        friendCount = review.find(class_="friend-count miniOrange").get_text()[:-
    8].strip()
289.        reviewCount = review.find(class_="review-count miniOrange").get_text()[:-
    8].strip()
290.
291.        if review.find(class_="photo-box-img")['src'] != "http://s3-
    media4.ak.yelpcdn.com/assets/2/www/img/78074914700f/default_avatars/user_small_square.p
    ng":
292.            userPhoto = "Has photo"
293.        else:
294.            userPhoto = "None"
295.
296.        reviewInfo = review.find(class_="reviewer_info").get_text().encode('utf-8')
297.
298.
299.        # Get review data
300.        reviewRating = review.find(itemprop="ratingValue").get("content")
301.        publish = review.find(itemprop="datePublished").get("content")
302.        description = review.find(itemprop="description").get_text().encode('utf-8')
303.
304.
305.        # Get review attributes
306.        if review.find(class_="i-wrap ig-wrap-common i-camera-common-wrap badge photo-
    count") != None:
307.            reviewPix = review.find(class_="i-wrap ig-wrap-common i-camera-common-
    wrap badge photo-count").get_text()[:-6].strip()
308.        else:
309.            reviewPix = "None"
310.
311.        if review.find(class_="i-wrap ig-wrap-common i-opentable-badge-common-
    wrap badge opentable-badge-marker") != None:
312.            reviewSeated = "Seated"
313.        else:
314.            reviewSeated = "None"
315.
316.        if review.find(class_="i ig-common i-deal-price-tag-common") != None:
317.            reviewDeal = "Purchased Deal"
318.        else:
319.            reviewDeal = "None"
320.
321.        if review.find(class_="i-wrap ig-wrap-common i-checkin-burst-blue-small-common-
    wrap badge checkin checkin-irregular") != None:
322.            reviewCheckIn = review.find(class_="i-wrap ig-wrap-common i-checkin-burst-
    blue-small-common-wrap badge checkin checkin-irregular").get_text()[:-14].strip()
323.        else:
324.            reviewCheckIn = "None"
325.
326.
```

```
        # Special Qype users lack stats
        if review.find(class_="count"):
            usefulfunnycool = review.findAll(class_="count")
            # Get useful, funny, cool statistics
            if usefulfunnycool[0].get_text() != "":
                useful = usefulfunnycool[0].get_text()
            else:
                useful = 0

            if usefulfunnycool[1].get_text() != "":
                funny = usefulfunnycool[1].get_text()
            else:
                funny = 0

            if usefulfunnycool[2].get_text() != "":
                cool = usefulfunnycool[2].get_text()
            else:
                cool = 0
        else:
            useful = 0
            funny = 0
            cool = 0

        with open(filenamerevsc, "ab") as filerev:
            frev = csv.writer(filerev)
            # Writing to CSV
            frev.writerow([resturl, eliteStatus, friendCount, reviewCount, userPhoto, r
    eviewInfo, reviewRating, publish, description, reviewPix, reviewSeated, reviewDeal, rev
    iewCheckIn, useful, funny, cool])

##     ##    ###    #### ##    ##
###   ###   ## ##    ##  ###   ##
#### ####  ##   ##   ##  ####  ##
## ### ## ##     ##  ##  ## ## ##
##     ## #########  ##  ##  ####
##     ## ##     ##  ##  ##   ###
##     ## ##     ## #### ##    ##

# 'Alphabet_City','Battery_Park','Chelsea','Chinatown','Civic_Center','East_Harlem','Ea
    st_Village','Financial_District','Flatiron','Gramercy','Greenwich_Village','Harlem','He
    ll\'s_Kitchen','Inwood','Kips_Bay','Koreatown','Little_Italy','Lower_East_Side','Manhat
    tan_Valley','Marble_Hill','Meatpacking_District','Midtown_East','Midtown_West','Morning
    side_Heights','Murray_Hill','NoHo','Nolita','Roosevelt_Island','SoHo','South_Street_Sea
    port','South_Village','Stuyvesant_Town','Theater_District','TriBeCa','Two_Bridges','Uni
    on_Square','Upper_East_Side','Upper_West_Side','Washington_Heights','West_Village', 'Yo
    rkville'

# 'Alphabet_City','Battery_Park','Chelsea','Chinatown','Civic_Center','East_Harlem','Ea
    st_Village','Financial_District','Flatiron','Gramercy','Greenwich_Village','Harlem','He
    ll\'s_Kitchen','Inwood','Kips_Bay','Koreatown','Little_Italy','Lower_East_Side','Manhat
    tan_Valley','Marble_Hill','Meatpacking_District','Midtown_East','Midtown_West','Morning
    side_Heights','Murray_Hill','NoHo','Nolita','Roosevelt_Island','SoHo','South_Street_Sea
    port','South_Village','Stuyvesant_Town','TriBeCa','Two_Bridges','Union_Square','Upper_E
    ast_Side','Upper_West_Side','Washington_Heights','West_Village'

# Yorkville entirely encompassed by Upper East Side
# Theater District entirely encompassed by Midtown West

```

```
370.# Unerrored:'Alphabet_City','Battery_Park','Civic_Center','East_Harlem','East_Village',
    'Flatiron','Gramercy','Kips_Bay','Koreatown','Little_Italy','Lower_East_Side','Manhatta
    n_Valley','Marble_Hill','Meatpacking_District','Midtown_East','Midtown_West','Morningsi
    de_Heights','Murray_Hill','NoHo','Nolita','Roosevelt_Island','SoHo','South_Street_Seapo
    rt','South_Village','Stuyvesant_Town','TriBeCa','Two_Bridges','Union_Square','Upper_Eas
    t_Side','Upper_West_Side','Washington_Heights','West_Village'
371.# Errored:'Chelsea','Chinatown','Financial_District','Greenwich_Village',
372.
373.# List all the locations
374.searchLocations = ['Alphabet_City','Battery_Park','Chelsea','Chinatown','Civic_Center',
    'East_Harlem','East_Village','Financial_District','Flatiron','Gramercy','Greenwich_Vill
    age','Harlem','Hell\'s_Kitchen','Inwood','Kips_Bay','Koreatown','Little_Italy','Lower_E
    ast_Side','Manhattan_Valley','Marble_Hill','Meatpacking_District','Midtown_East','Midto
    wn_West','Morningside_Heights','Murray_Hill','NoHo','Nolita','Roosevelt_Island','SoHo',
    'South_Street_Seaport','South_Village','Stuyvesant_Town','TriBeCa','Two_Bridges','Union
    _Square','Upper_East_Side','Upper_West_Side','Washington_Heights','West_Village']
375.
376.# proxy = urllib2.ProxyHandler({'http': '173.213.113.111:8089'})
377.# opener = urllib2.build_opener(proxy)
378.# opener.addheaders = [('User-
    agent', 'Mozilla/5.0 (X11; Linux i686; rv:5.0) Gecko/20100101 Firefox/5.0')]
379.# urllib2.install_opener(opener)
380.
381.opener = urllib2.build_opener()
382.opener.addheaders = [('User-
    agent', 'IE 9/Windows: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Triden
    t/5.0)')]
383.urllib2.install_opener(opener)
384.
385.# Iterate through
386.for searchLocation in searchLocations:
387.    # Print for reference
388.    print searchLocation
389.    filenamer = "yelpr_" + searchLocation + ".csv"
390.    filenamerev = "yelprev_" + searchLocation + ".csv"
391.
392.    # File setup
393.    with open(filenamer, "ab") as filer:
394.        fr = csv.writer(filer)
395.        # Write reference row
396.        fr.writerow(['resturl', 'title', 'latitude', 'longitude', 'rating', 'reviewCoun
    t', 'categories', 'photos', 'URL', 'neighborhood', 'menu', 'reservable', 'yelpDelivery'
    , 'slides', 'sponsor', 'claim', 'eliteReviews', 'transit', 'hours', 'attire', 'creditCa
    rds', 'parking', 'price', 'groups', 'kids', 'reservations', 'deal', 'delivery', 'takeou
    t', 'service', 'outdoorSeating', 'wifi', 'meals', 'bestNights', 'happyHour', 'alcohol',
     'smoking', 'coatCheck', 'noise', 'goodForDancing', 'ambience', 'tv', 'caters', 'wheelc
    hairAccessible'])
397.
398.    with open(filenamerev, "ab") as filerev:
399.        frev = csv.writer(filerev)
400.        # Write reference row
401.        frev.writerow(['resturl', 'eliteStatus', 'friendCount', 'reviewCount', 'userPho
    to', 'reviewInfo', 'reviewRating', 'publish', 'description', 'reviewPix', 'reviewSeated
    ', 'reviewDeal', 'reviewCheckIn', 'useful', 'funny', 'cool'])
402.
403.    # Build number to iterate search #
404.    for num in range(0,100):
405.
406.        print num
```

```
        # Build URL
        searchurl = "http://www.yelp.com/search?find_desc=restaurants&start=" + str(num*10) + "&l=p:NY:New_York:Manhattan:" + searchLocation

        # Read URL
        responseMain = urllib2.urlopen(searchurl)
        soupMain = BeautifulSoup(responseMain.read())
        responseMain.close()

        # If there are no more entries, break out of the loop and go to the next search Location
        if soupMain.find(class_="broaden-search-suggestions") != None:
            break

        # Otherwise get listings
        listings = soupMain.find_all(class_="search-result natural-search-result biz-listing-large")
        listingurls = []

        # Iterate through listings
        for listing in listings:

            # Check if no ratings
            if listing.find(class_="biz-rating biz-rating-large clearfix") != None:
                # Pull the url if it has ratings and add to list
                listingurls.append(listing.find('a')['href'])

        # Then get the url list
        for listingurl in listingurls:

            print listingurl
            # Then use scrapers on the urls
            restscrape(str("http://www.yelp.com" + listingurl), filenamer, filenamerev)

        time.sleep(60)

    time.sleep(600)
```

proxytest.py

```
import urllib2

# proxy = urllib2.ProxyHandler({'http': '173.213.113.111:8089'})
opener = urllib2.build_opener()
opener.addheaders = [('User-agent', 'Mozilla/5.0 (X11; Linux i686; rv:5.0) Gecko/20100101 Firefox/5.0')]
urllib2.install_opener(opener)

my_ip = urllib2.urlopen('http://whatthehellismyip.com/?ipraw').read()
my_ua = urllib2.urlopen('http://whatsmyua.com/').read()[750:900]
print my_ip
print my_ua
```

Concat.py

```
# ['Alphabet_City','Battery_Park','Chelsea','Chinatown','Civic_Center','East_Harlem','East_Village','Financial_District','Flatiron','Gramercy','Greenwich_Village','Harlem','H
```

```
    ell\'s_Kitchen','Inwood','Kips_Bay','Koreatown','Little_Italy','Lower_East_Side','Manha
    ttan_Valley','Marble_Hill','Meatpacking_District','Midtown_East','Midtown_West','Mornin
    gside_Heights','Murray_Hill','NoHo','Nolita','Roosevelt_Island','SoHo','South_Street_Se
    aport','South_Village','Stuyvesant_Town','TriBeCa','Two_Bridges','Union_Square','Upper_
    East_Side','Upper_West_Side','Washington_Heights','West_Village']

locations = ['Alphabet_City','Battery_Park','Chelsea','Chinatown','Civic_Center','East_
Harlem','East_Village','Financial_District','Flatiron','Gramercy','Greenwich_Village','
Harlem','Hell\'s_Kitchen','Inwood','Kips_Bay','Koreatown','Little_Italy','Lower_East_Si
de','Manhattan_Valley','Marble_Hill','Meatpacking_District','Midtown_East','Midtown_Wes
t','Morningside_Heights','Murray_Hill','NoHo','Nolita','Roosevelt_Island','SoHo','South
_Street_Seaport','South_Village','Stuyvesant_Town','TriBeCa','Two_Bridges','Union_Squar
e','Upper_East_Side','Upper_West_Side','Washington_Heights','West_Village']

frout=open("allr.csv","ab")
frevout=open("allrev.csv","ab")
# first file:
for line in open("yelpr_"+locations[0]+".csv"):
    frout.write(line)

for line in open("yelprev_"+locations[0]+".csv"):
    frevout.write(line)

# now the rest:
for num in range(1,len(locations)):

    fr = open("yelpr_"+locations[num]+".csv")
    fr.next() # skip the header
    for line in fr:
         frout.write(line)

    fr.close()

    frev = open("yelprev_"+locations[num]+".csv")
    frev.next() # skip the header
    for line in frev:
         frevout.write(line)

    frev.close()

frout.close()
frevout.close()
```

rsterilization.py

```
import csv
inputFileName = "allr.csv"
outputFileName = "allr_edited.csv"

with open(inputFileName) as infile, open(outputFileName, "wb") as outfile:
    r = csv.reader(infile)
    w = csv.writer(outfile)
    w.writerow(next(r))  # Writes the header unchanged
    for row in r:
        if row[7] == "None":
            row[7] = 0
        else:
            row[7] = 1

```

James Hu
Miller Ke
Jung Lee

```
15.         if row[8] == "None":
16.             row[8] = 0
17.         else:
18.             row[8] = 1
19.
20.         if row[10] == "None":
21.             row[10] = 0
22.         else:
23.             row[10] = 1
24.
25.         if row[11] == "None":
26.             row[11] = 0
27.         else:
28.             row[11] = 1
29.
30.         if row[12] == "None":
31.             row[12] = 0
32.         else:
33.             row[12] = 1
34.
35.         if row[13] == "None":
36.             row[13] = 0
37.         else:
38.             row[13] = 1
39.
40.         if row[14] == "None":
41.             row[14] = 0
42.         else:
43.             row[14] = 1
44.
45.         if row[15] == "None":
46.             row[15] = 1
47.         else:
48.             row[15] = 0
49.
50.         if row[16] == "None":
51.             row[16] = 0
52.         else:
53.             row[16] = 1
54.
55.         if row[17] == "None":
56.             row[17] = 0
57.         else:
58.             row[17] = 1
59.
60.         if row[18] == "None":
61.             row[18] = 0
62.         else:
63.             row[18] = 1
64.
65.         if row[19] == "Dressy":
66.             row[19] = 1
67.         else:
68.             row[19] = 0
69.
70.         if row[20] == "Yes":
71.             row[20] = 1
72.         else:
73.             row[20] = 0
```

```
74.
75.         if row[21] == "None" or row[21] == "Street":
76.             row[21] = 0
77.         else:
78.             row[21] = 1
79.
80.         row[22] = len(row[22].strip())
81.
82.         if row[23] == "Yes":
83.             row[23] = 1
84.         else:
85.             row[23] = 0
86.
87.         if row[24] == "Yes":
88.             row[24] = 1
89.         else:
90.             row[24] = 0
91.
92.         if row[25] == "Yes":
93.             row[25] = 1
94.         else:
95.             row[25] = 0
96.
97.         if row[26] == "Has deal":
98.             row[26] = 1
99.         else:
100.            row[26] = 0
101.
102.        if row[27] == "Yes":
103.            row[27] = 1
104.        else:
105.            row[27] = 0
106.
107.        if row[28] == "Yes":
108.            row[28] = 1
109.        else:
110.            row[28] = 0
111.
112.        if row[29] == "Yes":
113.            row[29] = 1
114.        else:
115.            row[29] = 0
116.
117.        if row[30] == "Yes":
118.            row[30] = 1
119.        else:
120.            row[30] = 0
121.
122.        if row[31] == "Free":
123.            row[31] = 1
124.        else:
125.            row[31] = 0
126.
127.        if row[34] == "Yes":
128.            row[34] = 1
129.        else:
130.            row[34] = 0
131.
132.        if row[35] == "Full Bar":
```

```
            row[35] = 2
        elif row[35] == "Beer & Wine Only":
            row[35] = 1
        else:
            row[35] = 0

        if row[36] == "Yes":
            row[36] = 2
        elif row[36] == "Outdoor Area/ Patio Only":
            row[36] = 1
        else:
            row[36] = 0

        if row[37] == "Yes":
            row[37] = 1
        else:
            row[37] = 0

        if row[38] == "Very Loud" or row[38] == "Loud":
            row[38] = 2
        elif row[38] == "Average":
            row[38] = 1
        else:
            row[38] = 0

        if row[39] == "Yes":
            row[39] = 1
        else:
            row[39] = 0

        if row[41] == "Yes":
            row[41] = 1
        else:
            row[41] = 0

        if row[42] == "Yes":
            row[42] = 1
        else:
            row[42] = 0

        if row[43] == "Yes":
            row[43] = 1
        else:
            row[43] = 0
        w.writerow(row)
```

revsterilization.py

```
import csv
inputFileName = "allrev.csv"
outputFileName = "allrev_edited.csv"

with open(inputFileName) as infile, open(outputFileName, "wb") as outfile:
    r = csv.reader(infile)
    w = csv.writer(outfile)
    w.writerow(next(r))  # Writes the header unchanged
    for row in r:
        if row[1] == "Elite":
            row[1] = 1
```

```
        else:
            row[1] = 0

        if row[4] == "None":
            row[4] = 0
        else:
            row[4] = 1

        if row[9] == "None":
            row[9] = 0

        if row[10] == "None":
            row[10] = 0
        else:
            row[10] = 1

        if row[11] == "None":
            row[11] = 0
        else:
            row[11] = 1

        if row[12] == "None":
            row[12] = 0
        else:
            row[12] = 1

        w.writerow(row)
```

g10.py

```
# To only keep restaurants with greater than 10 reviews.
import csv
inputFileName = "allr_edited_deduped.csv"
outputFileName = "allr_edited_deduped_g10.csv"

with open(inputFileName) as infile, open(outputFileName, "wb") as outfile:
    r = csv.reader(infile)
    w = csv.writer(outfile)
    w.writerow(next(r))  # Writes the header unchanged
    for row in r:
        if int(row[5]) >= 10:
            w.writerow(row)
```

Jung iPython Notebook

```
In [95]:

#histogram of ratings

%matplotlib inline
import matplotlib.pyplot as plt
import csv

ratings_count = {"1": 0,
"1.5": 0,
"2": 0,
"2.5": 0,
```

```
"3": 0,
"3.5": 0,
"4": 0,
"4.5": 0,
"5": 0}

infile  = open('allr_edited_deduped_g10.csv', "rb")
reader = csv.reader(infile)

reader.next()
for row in reader:
    ratings = row[4]
    ratings_count[ratings] = ratings_count[ratings] + 1

infile.close()

ratings_index = {"1": 0,
"1.5": 1,
"2": 2,
"2.5": 3,
"3": 4,
"3.5": 5,
"4": 6,
"4.5": 7,
"5": 8}

ratings_arr = [0, 0, 0, 0, 0, 0, 0, 0, 0]

keys = range(9)
for key, value in ratings_count.iteritems():
    index = ratings_index[key]
    ratings_arr[index] = value

plt.bar(keys,ratings_arr,color='b')
plt.show()

In [97]:

#histogram of price

price_count = {"1": 0,
"2": 0,
"3": 0,
"4": 0}

infile  = open('allr_edited_deduped_g10.csv', "rb")
reader = csv.reader(infile)

reader.next()
for row in reader:
    price = row[22]
    price_count[price] = price_count[price] + 1

infile.close()

price_index = {"1": 0,
"2": 1,
"3": 2,
"4": 3}
```

```
72.
73. price_arr = [0, 0, 0, 0]
74.
75. keys = range(4)
76. for key, value in price_count.iteritems():
77.     index = price_index[key]
78.     price_arr[index] = value
79.
80. plt.bar(keys,price_arr,color='r')
81. plt.show()
82.
83. In [88]:
84.
85. #attempt at a cluster chart
86.
87. import csv
88. import numpy as np
89. from pylab import plot,show
90. from numpy import vstack,array
91. from numpy.random import rand
92. from scipy.cluster.vq import kmeans,vq
93.
94. infile  = open('allr_edited_deduped.csv', "rb")
95. reader = csv.reader(infile)
96. #data = vstack((rand(150,2) + array([.5,.5]),rand(150,2)))
97.
98. ratings_list = []
99. for row in reader:
100.     ratings = int(float(row[4]))
101.     price = int(float(row[22]))
102.
103.     ratings_list.append([ratings,price])
104.     #p = np.asarray(ratings, price)
105.
106. p = vstack(ratings_list)
107.
108. # computing K-Means with K = 3 (3 clusters)
109. centroids,_ = kmeans(p,3)
110. # assign each sample to a cluster
111. idx,_ = vq(p,centroids)
112.
113. # some plotting using numpy's logical indexing
114. plot(p[idx==0,0],p[idx==0,1],'ob',
115.      p[idx==1,0],p[idx==1,1],'or',
116.      p[idx==2,0],p[idx==2,1],'og')
117. plot(centroids[:,0],centroids[:,1],centroids[:,2],'sg',markersize=8)
118. show()
119.
120. ---------------------------------------------------------------------------
121. IndexError                                Traceback (most recent call last)
122. <ipython-input-88-ae7c317bbd18> in <module>()
123.      26
124.      27 # some plotting using numpy's logical indexing
125. ---> 28 plot(p[idx==0,0],p[idx==0,2],'ob',
126.      29      p[idx==1,0],p[idx==1,2],'or',
127.      30      p[idx==2,0],p[idx==2,2],'og')
128.
129. IndexError: index 2 is out of bounds for axis 1 with size 2
130.
```

```
In [75]:

#attempt at a boxplot

from pylab import *

# fake up some data
spread= rand(50) * 100
center = ones(25) * 50
flier_high = rand(10) * 100 + 100
flier_low = rand(10) * -100
data =concatenate((spread, center, flier_high, flier_low), 0)

print data


# fake up some more data
spread= rand(50) * 100
center = ones(25) * 40
flier_high = rand(10) * 100 + 100
flier_low = rand(10) * -100
d2 = concatenate( (spread, center, flier_high, flier_low), 0 )
data.shape = (-1, 1)
d2.shape = (-1, 1)
#data = concatenate( (data, d2), 1 )
# Making a 2-D array only works if all the columns are the
# same length.  If they are not, then use a list instead.
# This is actually more efficient because boxplot converts
# a 2-D array into a list of vectors internally anyway.
data = [data, d2, d2[::2,0]]


# multiple box plots on one figure
figure()
boxplot(data)

show()

[  63.79546152   19.65157435   32.63003327   91.97750172   65.06890263
   11.1174677    82.57265482   52.26190629   26.91672317   15.63198348
   65.97502866   67.39583805   37.99084017    9.6420859    94.18799126
   29.10139972   40.51556865    4.85790462    3.85263129   32.84172293
   86.47713579    5.64763582   86.39398244   65.73331889   40.97693479
   21.35507367   64.49553411   49.17026711   30.12449077    2.0095419
   67.27264579   29.99458345   99.13562905   49.78354538    3.77188196
   82.74870614   38.22563021   34.53909834   50.66782223   50.21074449
   78.08356527   94.25060163   62.16788951   19.04474011   90.68360956
   45.49576673    7.40531061   51.75721172   15.75123556   96.81151693
   50.           50.           50.           50.           50.           50.
   50.           50.           50.           50.           50.           50.
   50.           50.           50.           50.           50.           50.
   50.           50.           50.           50.           50.           50.
   50.          114.23848175  137.16782724  149.83022405  129.36547062
  102.91117314  158.70471456  131.81505752  113.28382836  137.40310604
  148.56522517  -38.85887058  -72.52140826  -31.64348736   -6.45848084
  -67.5212848   -25.16578098  -23.60681635  -92.05346249  -92.60459022
  -15.02384999]

In [107]:
```

```
190.
191.#linear regression for review count
192.
193.from scipy import stats
194.
195.infile  = open('allr_edited_deduped_g10.csv', "rb")
196.reader = csv.reader(infile)
197.
198.ratings_list = []
199.reviews_list = []
200.reader.next()
201.for row in reader:
202.    ratings = int(float(row[4]))
203.    reviewcount = int(float(row[5]))
204.
205.    ratings_list.append(ratings)
206.    reviews_list.append(reviewcount)
207.
208.slope, intercept, r_value, p_value, std_err = stats.linregress(reviews_list,ratings_lis
    t)
209.
210.print "P-value", p_value #significant
211.print "R-squared", r_value**2
212.print "Slope", slope
213.
214.P-value 7.97070891518e-57
215.R-squared 0.0397748655747
216.Slope 0.00052694268763
217.
218.In [108]:
219.
220.#linear regression for price
221.
222.from scipy import stats
223.
224.infile  = open('allr_edited_deduped_g10.csv', "rb")
225.reader = csv.reader(infile)
226.
227.ratings_list = []
228.price_list = []
229.reader.next()
230.for row in reader:
231.    ratings = int(float(row[4]))
232.    price = int(float(row[22]))
233.
234.    ratings_list.append(ratings)
235.    price_list.append(price)
236.
237.slope, intercept, r_value, p_value, std_err = stats.linregress(price_list,ratings_list)

238.
239.print "P-value", p_value #significant
240.print "R-squared", r_value**2
241.print "Slope", slope
242.
243.P-value 3.55606573266e-12
244.R-squared 0.00774710286578
245.Slope 0.0689953657474
246.
```

James Hu
Miller Ke
Jung Lee

```
In [111]:

#linear regression for claims

from scipy import stats

infile  = open('allr_edited_deduped_g10.csv', "rb")
reader = csv.reader(infile)

ratings_list = []
claim_list = []
reader.next()
for row in reader:
    ratings = int(float(row[4]))
    claim = int(float(row[15]))

    ratings_list.append(ratings)
    claim_list.append(claim)

slope, intercept, r_value, p_value, std_err = stats.linregress(claim_list,ratings_list)

print "P-value", p_value #significant
print "R-squared", r_value**2
print "Slope", slope

P-value 7.19556603492e-33
R-squared 0.0226739762926
Slope 0.190908992096

In [112]:

#linear regression for deals

from scipy import stats

infile  = open('allr_edited_deduped_g10.csv', "rb")
reader = csv.reader(infile)

ratings_list = []
deal_list = []
reader.next()
for row in reader:
    ratings = int(float(row[4]))
    deal = int(float(row[26]))

    ratings_list.append(ratings)
    deal_list.append(deal)

slope, intercept, r_value, p_value, std_err = stats.linregress(deal_list,ratings_list)

print "P-value", p_value #significant
print "R-squared", r_value**2
print "Slope", slope

P-value 1.50657558112e-17
R-squared 0.0116252818137
Slope -0.147325808319
```

James Hu
Miller Ke
Jung Lee

.ipynb file

JungIPYNBWork.ipynb

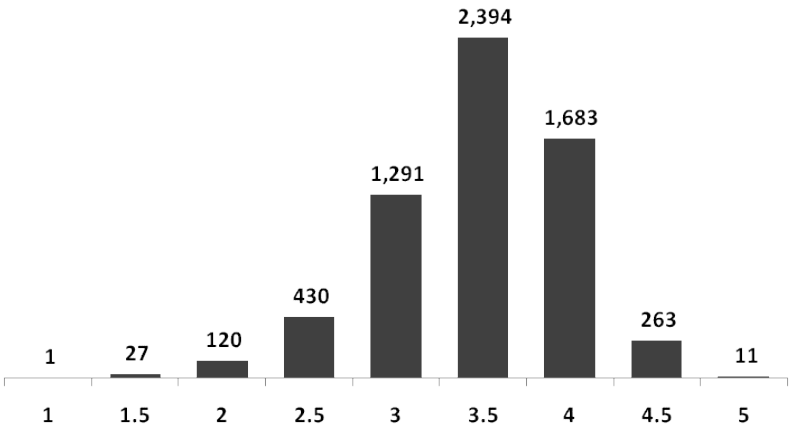
yelpinsight.py

```
1.  import matplotlib.pyplot as plt
2.  import pandas as pd
3.
4.  df=pd.read_csv('allrev_edited_deduped.csv', sep="[,\s]*")
5.  df
6.  g=df.describe()
7.  pd.set_option('display.max_columns',7)
8.  g
9.  print df
10. print g.to_string()
11. plt.figure();
12. bp = df.boxplot(column=['funny'],by =['eliteStatus'])
13. plt.show();
```

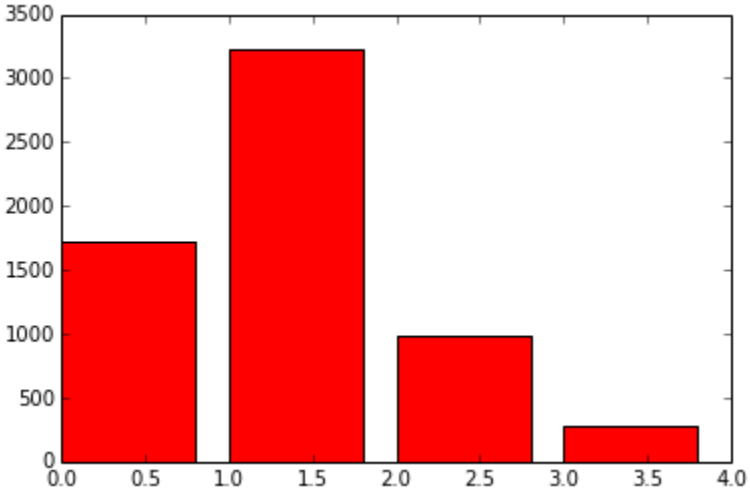
James Hu
Miller Ke
Jung Lee

Visualizations

Histogram of ratings distribution (Visual 1)

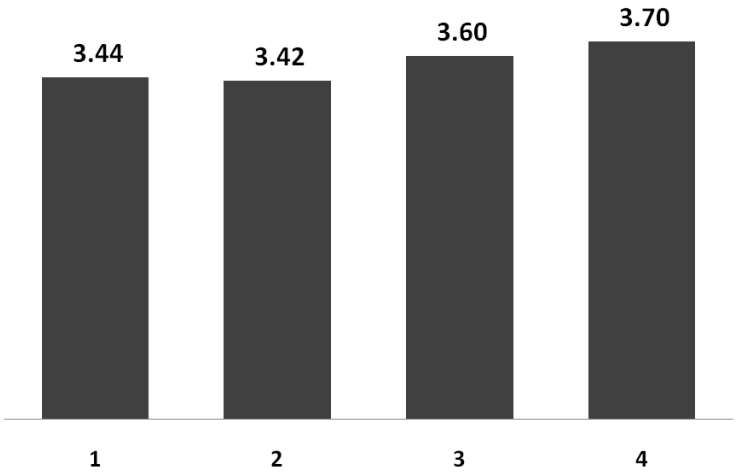


Histogram of price range (Visual 2)

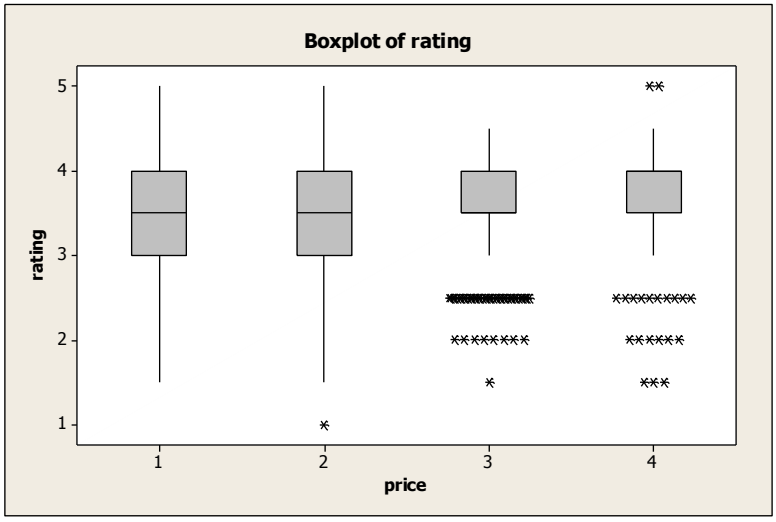


Average ratings by price range (Visual 3)

James Hu
Miller Ke
Jung Lee

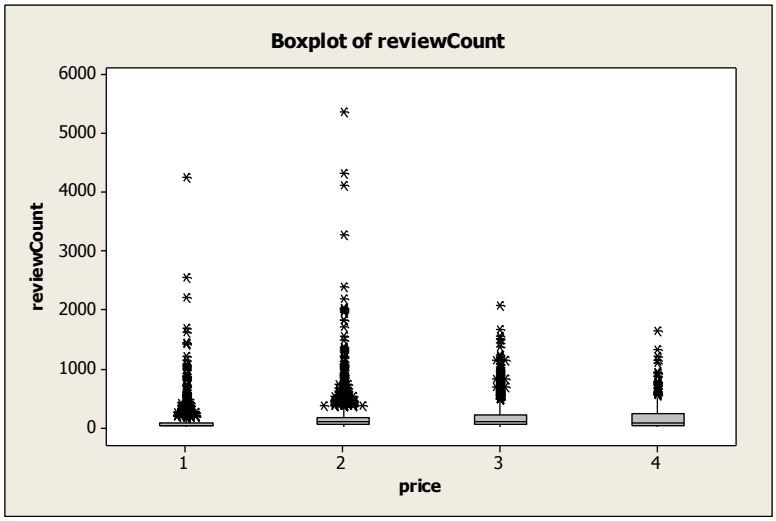


Boxplot of ratings and price range

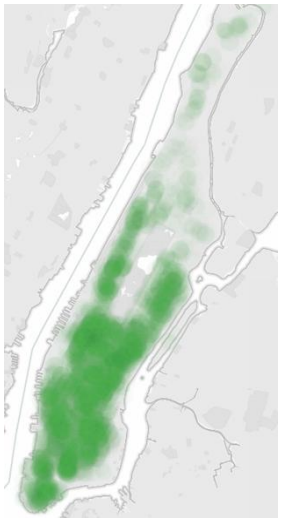


Boxplot of review count and price

James Hu
Miller Ke
Jung Lee



Heatmap of ratings (Visual 4)

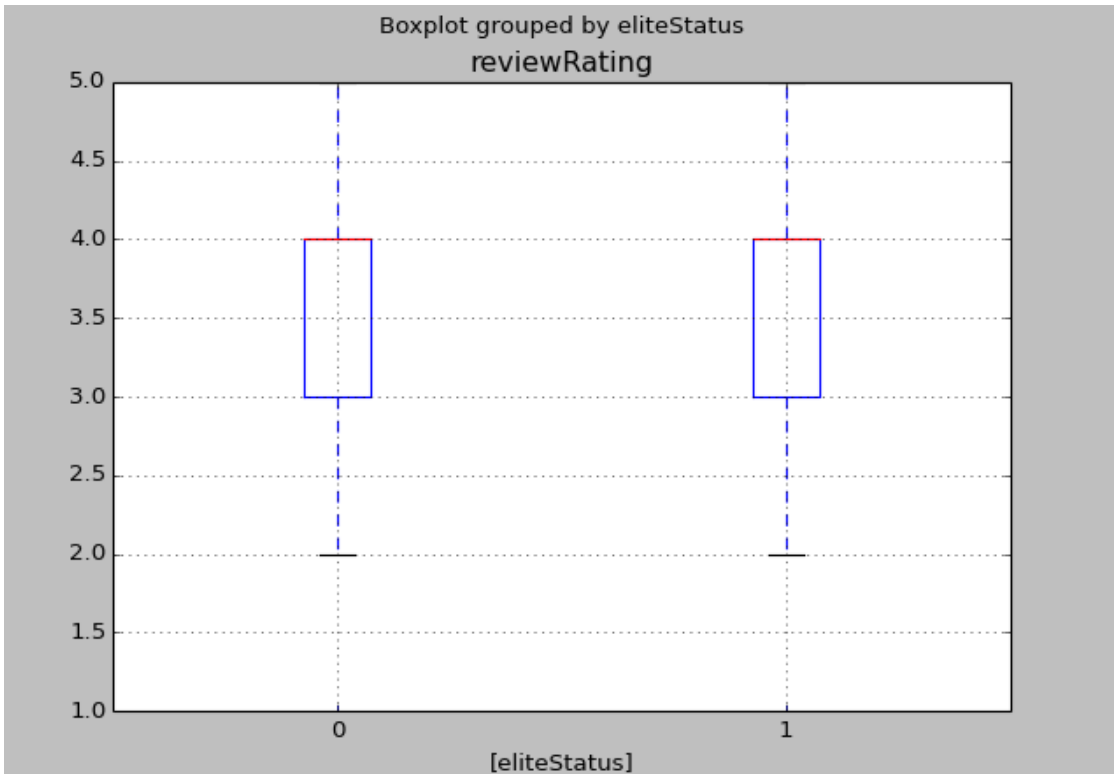


Heatmap of ratings by price range (Visual 5)

Heatmap File

Heatmaps.twb

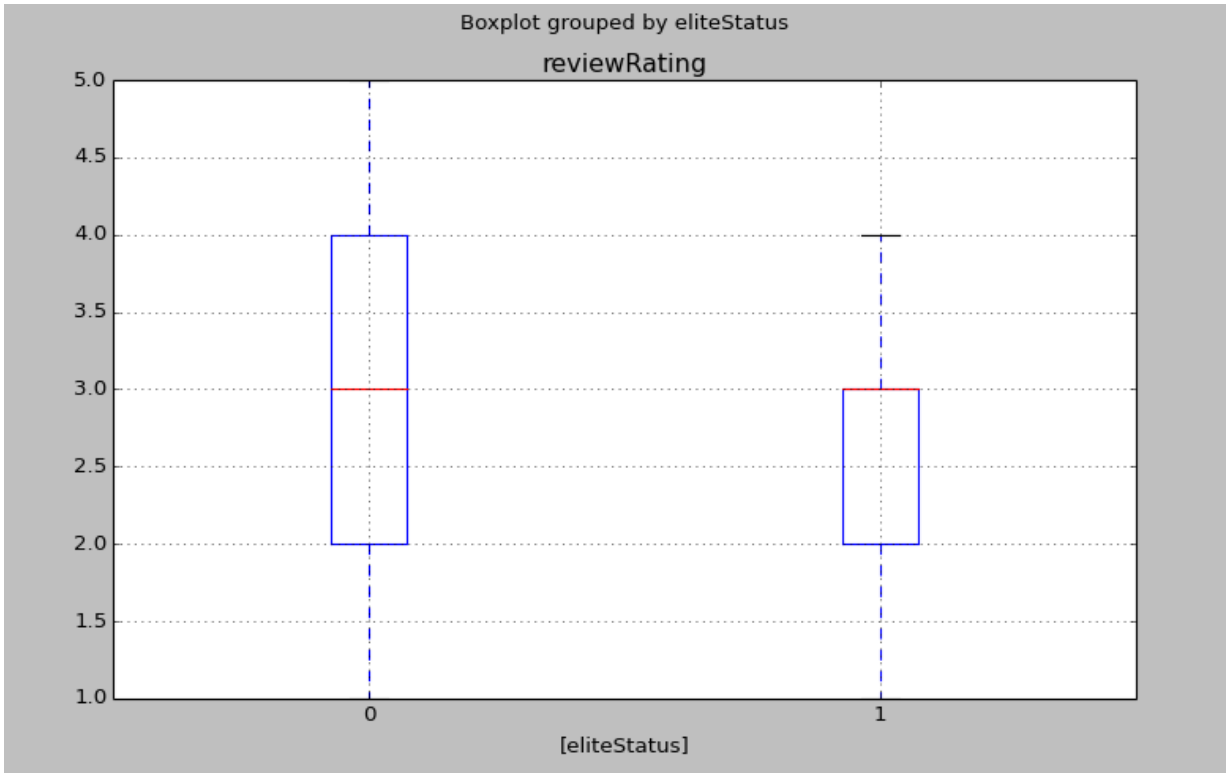
Appendix 6




```
$ python insight4.py
```

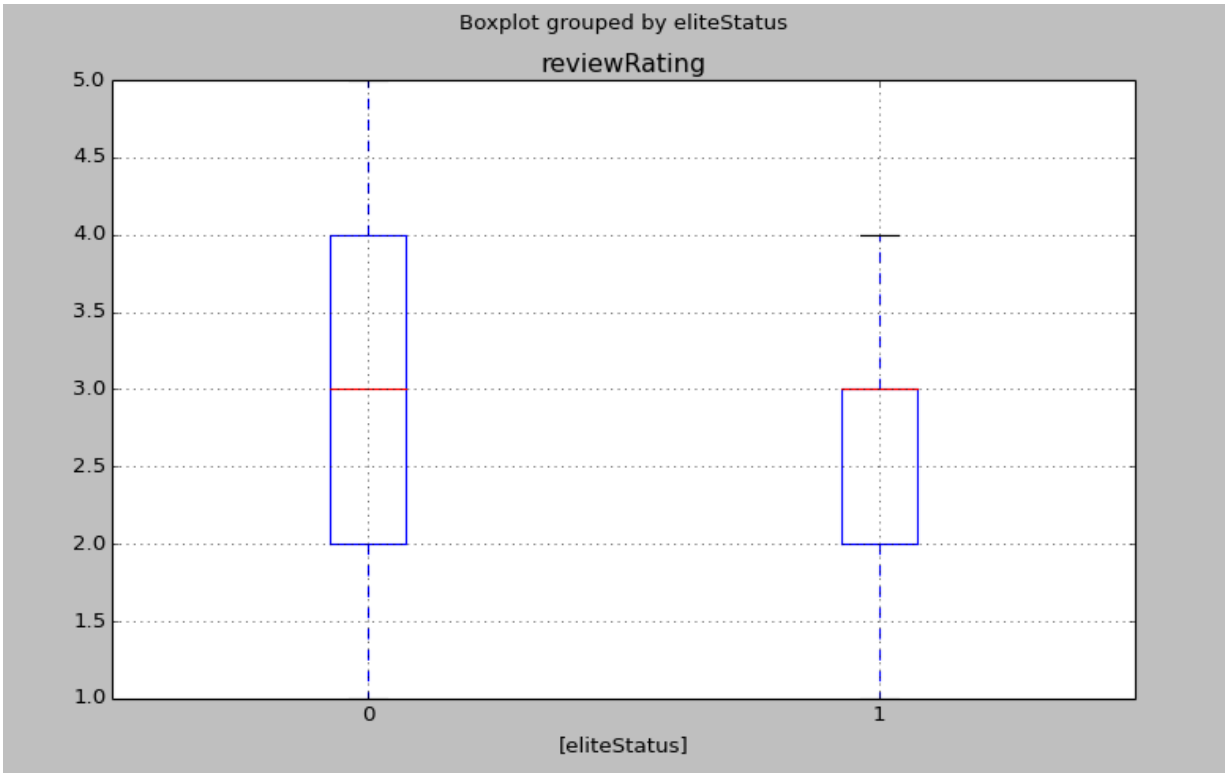
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 222268 entries, 0 to 222267
Data columns (total 2 columns):
eliteStatus     222268  non-null values
reviewRating    222268  non-null values
dtypes: int64(2)
                  eliteStatus   reviewRating
eliteStatus
0           count       158425  158425.000000
            mean             0       3.459309
            std              0       1.296230
            min              0       1.000000
            25%              0       3.000000
            50%              0       4.000000
            75%              0       4.000000
            max              0       5.000000
1           count        63843   63843.000000
            mean             1       3.485879
            std              0       1.003300
            min              1       1.000000
            25%              1       3.000000
            50%              1       4.000000
            75%              1       4.000000
            max              1       5.000000
```

Appendix 7




```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 108 entries, 0 to 107
Data columns (total 3 columns):
resturl         108  non-null values
eliteStatus     108  non-null values
reviewRating    108  non-null values
dtypes: int64(2), object(1)
                  eliteStatus  reviewRating
eliteStatus
```

```
0            count           73     73.000000
             mean             0      2.890411
             std              0      1.161436
             min              0      1.000000
             25%              0      2.000000
             50%              0      3.000000
             75%              0      4.000000
             max              0      5.000000
1            count           35     35.000000
             mean             1      2.571429
             std              0      1.092372
             min              1      1.000000
             25%              1      2.000000
             50%              1      3.000000
             75%              1      3.000000
             max              1      4.000000
```




```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 108 entries, 0 to 107
Data columns (total 3 columns):
resturl         108  non-null values
eliteStatus     108  non-null values
reviewRating    108  non-null values
dtypes: int64(2), object(1)
                eliteStatus  reviewRating
eliteStatus
0            count           73     73.000000
             mean             0      2.890411
             std              0      1.161436
             min              0      1.000000
             25%              0      2.000000
             50%              0      3.000000
             75%              0      4.000000
             max              0      5.000000
1            count           35     35.000000
```

mean	1	2.571429
std	0	1.092372
min	1	1.000000
25%	1	2.000000
50%	1	3.000000
75%	1	3.000000
max	1	4.000000

Appendix 8

dtypes: int64(6)

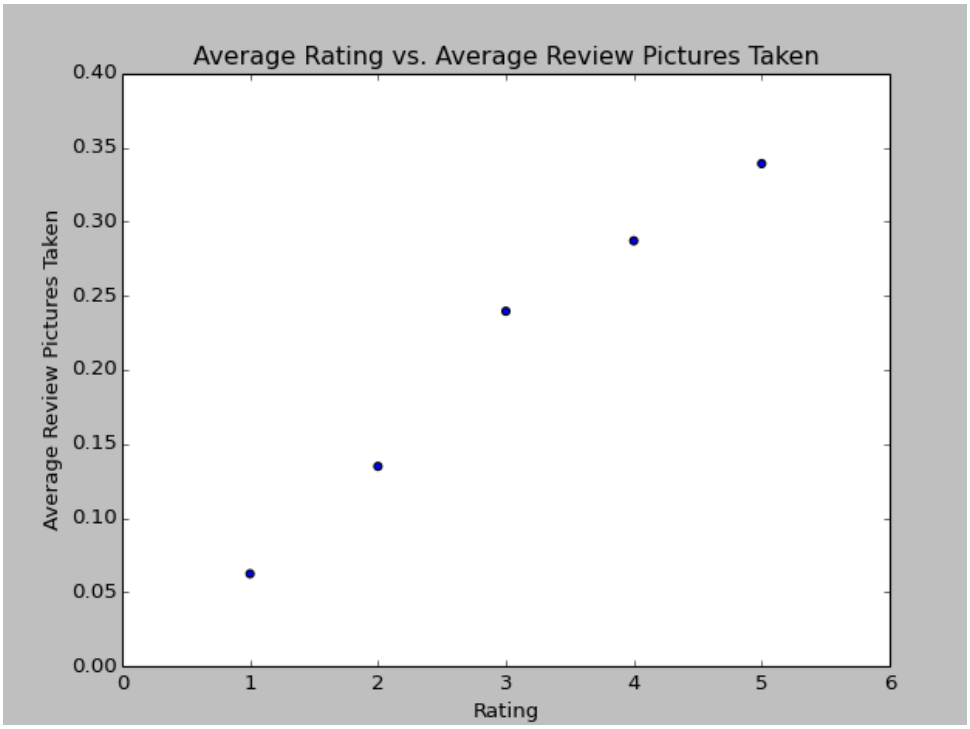
eliteStatus		eliteStatus	reviewPix	reviewCheckIn	useful	funny	cool
0	count	158425	158425.000000	158425.000000	158425.000000	158425.000000	158425.000000
	mean	0	0.163200	0.176531	1.013697	0.443200	0.449872
	std	0	0.890028	0.381273	2.298212	1.517773	1.671170
	min	0	0.000000	0.000000	0.000000	0.000000	0.000000
	25%	0	0.000000	0.000000	0.000000	0.000000	0.000000
	50%	0	0.000000	0.000000	0.000000	0.000000	0.000000
	75%	0	0.000000	0.000000	1.000000	0.000000	0.000000
	max	0	58.000000	1.000000	162.000000	76.000000	160.000000
1	count	63843	63843.000000	63843.000000	63843.000000	63843.000000	63843.000000
	mean	1	0.458672	0.460614	1.645850	0.860063	0.965838
	std	0	1.832375	0.498450	2.746813	2.295888	2.156238
	min	1	0.000000	0.000000	0.000000	0.000000	0.000000
	25%	1	0.000000	0.000000	0.000000	0.000000	0.000000
	50%	1	0.000000	0.000000	1.000000	0.000000	0.000000
	75%	1	0.000000	1.000000	2.000000	1.000000	1.000000
	max	1	95.000000	1.000000	99.000000	99.000000	97.000000

Appendix 9

reviewRating		reviewRating	reviewPix	reviewCheckIn	useful	funny	cool
1	count	21582	21582.000000	21582.000000	21582.000000	21582.000000	21582.000000
	mean	1	0.062459	0.109443	1.696182	0.886479	0.369660
	std	0	0.439459	0.312202	3.378772	2.224401	1.133441
	min	1	0.000000	0.000000	0.000000	0.000000	0.000000
	25%	1	0.000000	0.000000	0.000000	0.000000	0.000000
	50%	1	0.000000	0.000000	1.000000	0.000000	0.000000
	75%	1	0.000000	0.000000	2.000000	1.000000	0.000000
	max	1	15.000000	1.000000	119.000000	56.000000	33.000000
2	count	26063	26063.000000	26063.000000	26063.000000	26063.000000	26063.000000
	mean	2	0.134942	0.223727	1.234317	0.665388	0.418985
	std	0	0.770803	0.416749	2.110649	1.941956	1.273911
	min	2	0.000000	0.000000	0.000000	0.000000	0.000000
	25%	2	0.000000	0.000000	0.000000	0.000000	0.000000
	50%	2	0.000000	0.000000	1.000000	0.000000	0.000000
	75%	2	0.000000	0.000000	2.000000	1.000000	0.000000
	max	2	32.000000	1.000000	49.000000	76.000000	57.000000
3	count	48380	48380.000000	48380.000000	48380.000000	48380.000000	48380.000000
	mean	3	0.239603	0.307420	0.995494	0.512050	0.548243
	std	0	1.076466	0.461430	2.044216	1.580318	1.693294
	min	3	0.000000	0.000000	0.000000	0.000000	0.000000
	25%	3	0.000000	0.000000	0.000000	0.000000	0.000000
	50%	3	0.000000	0.000000	0.000000	0.000000	0.000000
	75%	3	0.000000	1.000000	1.000000	0.000000	1.000000
	max	3	29.000000	1.000000	151.000000	82.000000	150.000000
4	count	79473	79473.000000	79473.000000	79473.000000	79473.000000	79473.000000
	mean	4	0.287116	0.292590	1.131617	0.499302	0.678029
	std	0	1.268425	0.454955	2.265533	1.639771	1.900296
	min	4	0.000000	0.000000	0.000000	0.000000	0.000000
	25%	4	0.000000	0.000000	0.000000	0.000000	0.000000
	50%	4	0.000000	0.000000	0.000000	0.000000	0.000000
	75%	4	0.000000	1.000000	1.000000	0.000000	1.000000
	max	4	58.000000	1.000000	112.000000	54.000000	112.000000
5	count	46770	46770.000000	46770.000000	46770.000000	46770.000000	46770.000000
	mean	5	0.339170	0.236369	1.257195	0.517319	0.718965
	std	0	1.721847	0.424856	2.759751	1.888070	2.323070
	min	5	0.000000	0.000000	0.000000	0.000000	0.000000
	25%	5	0.000000	0.000000	0.000000	0.000000	0.000000
	50%	5	0.000000	0.000000	1.000000	0.000000	0.000000
	75%	5	0.000000	0.000000	2.000000	0.000000	1.000000
	max	5	95.000000	1.000000	162.000000	99.000000	160.000000

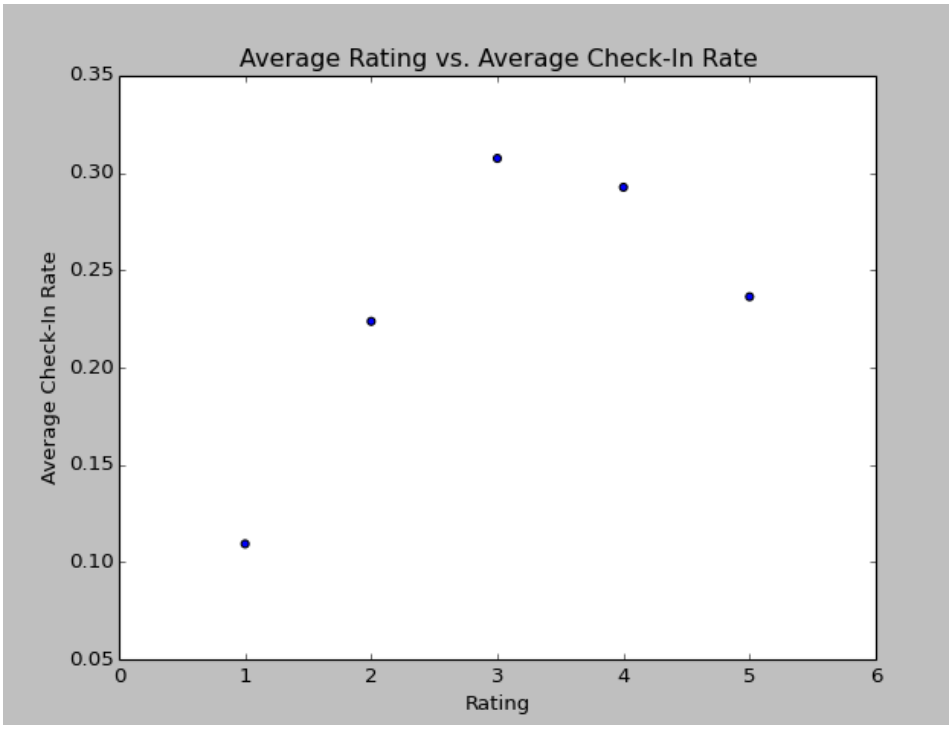
James Hu
Miller Ke
Jung Lee

Appendix 10



Appendix 11

James Hu
Miller Ke
Jung Lee



Appendix 12