Predictive Data Analysis

Ranking of YELP

Hyerim Hwang

Table of Contents

Table of Contents **1**

Introduction **2**

Cleaning data **3**

Findings **4**

Conclusion **5**

Appendix **6**

References **6**

Introduction

Yelp is one of the most popular search engines in restaurants that users can get lots of helpful information at a glance containing the store's location; opening hours; menus with prices and actual customers' reviews; pictures; ratings. A few times, I found that some restaurants, which keep over 4.5 ratings placed on top of the results page, were not the best selections and I wondered which factors of the restaurant proceed Yelp to optimize in the search results page. The objective of the project is to examine which factors from restaurants determine their rankings in the search result page.

Yelp Ranking Factors

According to the Local Visibility System, on Yelp's search result page, there are some major factors lead the ranking of the businesses listed on the top (Yelp Ranking Factors). To find the factors that determine the Yelp ranking, the report applied several prominent variables from the list below.

1. Existence of reviews.
2. Keyword-relevance of reviews.
3. Business categories specified.
4. Name of business.
5. A number of reviews.
6. Reviews by "Elite" members.
7. Check-ins via smartphone.
8. Ratings of reviewers.

Also, the local brand management service Charmeter assumed that the content page needs to work by adding more photos (free), managing the specialties section with a good description (free), applying the call to action button (paid), offering customers check-in (free) to get more customers attention. These were needed to web scraping from each page, so in this case, fetching the API or downloading existing JSON, or CSV files would be more fit rather than applying web scraping method per each page.

Hypothesis

A higher ranking will take a large number of reviews. Or a higher ranking has less expensive menus. There needs to be a linear relationship between the two variables, “the number of reviews ” and “ranking” or “prices” and “ranking”.

Methodology

Since Yelp doesn’t offer any official API to the public - only offers to businesses or valid JSON datasets[1], there is a great Python library for web scraping to gather raw information online, Beautiful soup (figure. 1). To conduct an accurate ranking on the search results page, the main end-point page https://www.yelp.com/search?cflt=restaurants&find_loc=nyc was picked because the ranking system on this page was not influenced by the distance between stores and the location of the device itself. Later on, San Fransisco and Chicago were also managed to see the analytics on NYC dataset is accurate.

```
from splinter import Browser
from bs4 import BeautifulSoup
import os
import pandas as pd
from datetime import datetime
import platform
import matplotlib.pyplot as plt
```

Figure. 1: Imports the libraries for scraping data online.

[1] There is an existing JSON file offering non-officially, but the dataset is so huge and enables to convert to CSV file to be a readable dataset (https://www.yelp.com/dataset).

Web Scraping to collect datasets

For this project, the report proceeded to conduct analytics on the Yelp search result page that holds a number of reviews and prices. These major two variables were selected to analyze the relationship to the Yelp ranking. The prices were displayed by a dollar sign, not a numeric value, and both numbers of reviews and prices stood out on the result page. There was able to capture other information about the restaurants so the report also scrapped their address, names, and types of the restaurants.

```python
def init_browser():
    if platform.system().lower() == 'windows'.lower():
        executable_path = {
            'executable_path':
            os.path.join(os.getcwd(), 'chromedriver.exe')}
        return Browser('chrome', **executable_path, headless=False)
    else:
        return Browser('chrome')

def get_html(browser, url):
    browser.visit(url)
    html = browser.html
    return html

def get_data(html):
    soup = BeautifulSoup(html, "html.parser")
    ol_lists = soup.find('div', class_='lemon--div__373c0__1mboc mapColumnTransition__373c0__10KHB arrange-unit__373c0_

    ranks = ol_lists.find_all('p', class_='lemon--p__373c0__3Qnnj text__373c0__2pB8f text-color--black-regular__373c0_
    title = ol_lists.find_all('a', class_='lemon--a__373c0__IEZFH link__373c0__29943 link-color--blue-dark__373c0__1mh
    target_bf_rating = ol_lists.find_all('div', class_='lemon--div__373c0__1mboc attribute__373c0__1hPI_ display--inli
    reviews = ol_lists.find_all('span', class_='lemon--span__373c0__3997G text__373c0__2pB8f reviewCount__373c0__2r4xT
    infos = ol_lists.find_all('div', class_='lemon--div__373c0__1mboc mainAttributes__373c0__1r0QA arrange-unit__373c0_
    addresses = ol_lists.find_all('address', class_='lemon--address__373c0__2sPac')
    time = datetime.now()

    if len(ranks) > 30 or len(ranks) < 33:
        diff = len(ranks) - 30
        ranks = ranks[diff:]
        target_bf_rating = target_bf_rating[diff:]
        infos = infos[diff:]
    if len(ranks) == 33:
        ranks = ranks[2:-1]
        target_bf_rating = target_bf_rating[2:-1]
        infos = infos[2:-1]

    for i in range(len(ranks)):
        rank = [p.text.split('.')[0] for p in ranks]
        title = [a.a.text for a in ranks]
        rating = [a.span.div for a in target_bf_rating if a.span]
        num_review = [a.text.split()[0]for a in reviews]
        price = [a.div.div.find_next_sibling('div').find_next_sibling('div') for a in infos]
        types = [b.find_all('a', class_='lemon--a__373c0__IEZFH link__373c0__29943 link-color--inherit__373c0__15ymx l

        address = [a.div.div.p.span.text for a in addresses]
        neighbourhood = [a.find_next_sibling('div') for a in addresses]

        data[rank[i]] = {'title': title[i],
                         'rating': rating[i]['aria-label'].split(' ')[0],
                         'num_review': num_review[i],
                         'price': ''.join([a for a in price[i].div.div.span.span.text if a is '$' or a is '$$' or a is
                         'types': [b.text for b in types[i]],
                         'address': address[i],
                         'neighbourhood': ''.join([a.div.div.p.text for a in neighbourhood[i]]),
                         'time': time}
    return data

def scrape(browser, url):
    html = get_html(browser, url)
    datas = get_data(html)
    return len(datas), datas

def main():
    browser = init_browser()
    url = "https://www.yelp.com/search?cflt=restaurants&find_loc=New+York%2C+NY"
#    url = "https://www.yelp.com/search?cflt=restaurants&find_loc=San+Francisco%2C+CA"
#    url = "https://www.yelp.com/search?cflt=restaurants&find_loc=Chicago%2C+IL"
    datas = scrape(browser, url)
    return datas
```

Figure. 2 : The yellow line points out five functions to scrap Yelp data.

To scrap any information of Yelp on Jupyter notebook, the report wrote functionalized codes and run the final function **def main()** to call other functions simultaneously (Figure. 2). Since Jupyter notebook doesn't work well with direct scraping by beautiful soup, Codes for opening the new Chrome window was a priority which refers to **def init_brower()**. Then, a new window opened with given URL from **def get_html(browser, url)** and returned the dataset in Html (Find the \<tag> divs or spans, a, p, address with class names) and stored them into a dictionary using beautiful soup library by **def get_data(html)**. To handle both browsing URL function and get stored data function at once, **def scrape(browser, url)** were call the two functions get_html(browser, url) and get_data(html).

The Get_data function collected actual scraping information of Yelp by pointing out their tag and class name of the restaurants' ranking; name; rating; a number of reviews; average price; address; neighborhood; types and stored them into a new dictionary. After calling the final main() function, a data frame was created and called them using the panda's library[2] (Figure. 3).

```
main()
df = pd.DataFrame(data).T
df.head()
```

	title	rating	num_review	price	types	address	neighbourhood	time
1	Amélie	4.5	1	$$	[French, Wine Bars]	22 W 8th St	Greenwich Village	2019-10-25 21:20:23.057085
2	Upstate	4.5	2593	$$	[Seafood, Wine Bars, Beer Bar]	95 1st Ave	East Village	2019-10-25 21:20:23.057085
3	LoveMama	4.5	1796	$$	[Thai, Malaysian, Vietnamese]	174 2nd Ave	East Village	2019-10-25 21:20:23.057085
4	Burger & Lobster	4	4647	$$	[Seafood, Burgers, American (New)]	39 W 19th St	Flatiron	2019-10-25 21:20:23.057085
5	Thai Villa	4.5	5298	$$	[Thai, Asian Fusion]	5 E 19th St	Flatiron	2019-10-25 21:20:23.057085

Figure. 3 : The dataset transfer to data frame from the panda's library.

[2] It offers data structures and operations for manipulating numerical tables and time series (Wikipedia).

Cleaning Data

Check data types of the data frame

```
df.dtypes
```

```
ranking                 object
title                   object
rating                  object
num_review              object
price                   object
address                 object
neighbourhood           object
time            datetime64[ns]
type_1                  object
type_2                  object
type_3                  object
dtype: object
```

Change numuric values to be a int or float

```
df['ranking'] = df['ranking'].astype(int)
df['rating'] = df['rating'].astype(float)
df['num_review'] = df['num_review'].astype(int)
```

Figure. 4 : Check data types and use default function to change the types of variables at ease.

Since a data frame has lots of variables with both numeric and string values, the report checked each variable's types of the data frame (Figure. 4). Two variables of price and types which were needed to change - a price column had to transfer to numeric values and a types column contained listed multiple strings that were necessary to change to multiple columns with each string value (Figure. 5).

Make the multiple columns with string value from a column with list value

- [types] -> 'type_1', 'type_2', 'type_3'
- Drop the column [types]
- Make a ranking column from index

```
df2 = pd.DataFrame(df['types'].values.tolist())
df = df.assign(**{'type_1': df2[0].values, 'type_2': df2[1].values, 'type_3': df2[2].values})
df = df.drop(['types'], axis=1)
df = df.reset_index()
df = df.rename(columns = {'index':'ranking'})
df.head()
```

	ranking	title	rating	num_review	price	address	neighbourhood	time	type_1	type_2	type_3
0	1	Amélie	4.5	1	$$	22 W 8th St	Greenwich Village	2019-10-25 21:20:23.057085	French	Wine Bars	None
1	2	Upstate	4.5	2593	$$	95 1st Ave	East Village	2019-10-25 21:20:23.057085	Seafood	Wine Bars	Beer Bar
2	3	LoveMama	4.5	1796	$$	174 2nd Ave	East Village	2019-10-25 21:20:23.057085	Thai	Malaysian	Vietnamese
3	4	Burger & Lobster	4	4647	$$	39 W 19th St	Flatiron	2019-10-25 21:20:23.057085	Seafood	Burgers	American (New)
4	5	Thai Villa	4.5	5298	$$	5 E 19th St	Flatiron	2019-10-25 21:20:23.057085	Thai	Asian Fusion	None

Figure. 5 : A listed multiple strings in a column transferred to three columns with one string.

Applying the linear model in R Studio

The report stated the hypothesis that a higher ranking will take a large number of reviews. To make a linear model with the ny data, use num_review column to be a dependent variable and ranking is the independent variable (Figure.6 - 1). To predict the number of reviews on the highest ranking, use data.frame("ranking" = 1) and predict it within the linear model ny data (Figure.6 - 2). Then, use ggplot to plot the predicted values with actual values (Figure.6 - 3).

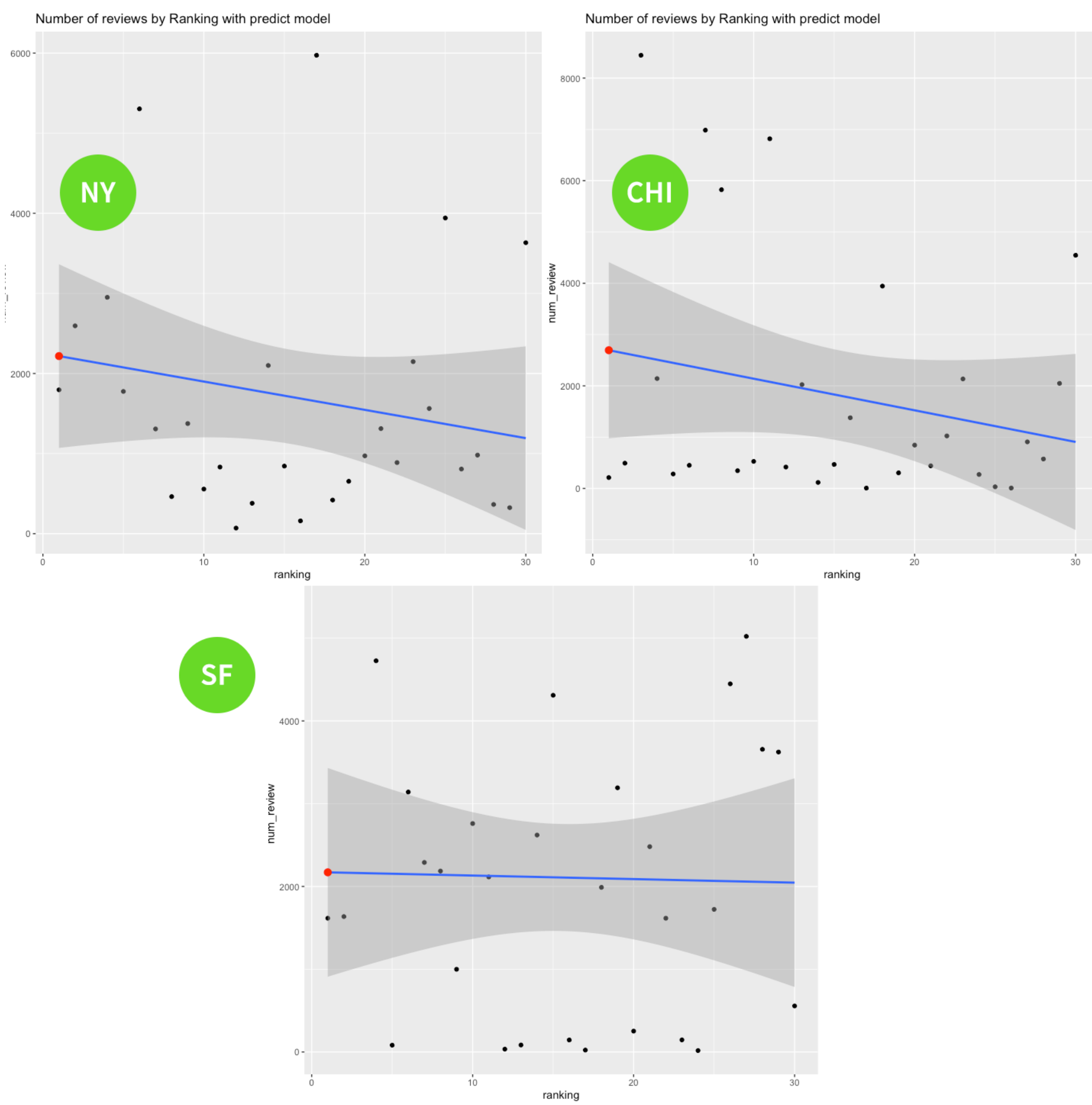
```r
## 1 --- Using raw_ny dataset, pick a variable "num_review" --- ##
ny <- raw_ny[, c(1, 3, 4, 5)]
glimpse(ny)

lm_ny <- lm(num_review ~ ranking, data=ny)
summary(lm_ny)

fitted_ny <- fitted.values(lm_ny)
fitted_ny
residuals_ny <- residuals(lm_ny)
residuals_ny

lm_matrix_ny <- broom::augment(lm_ny)
head(lm_matrix_ny)
lm_matrix_ny$.resid_abs <- abs(lm_matrix_ny$.resid)
lm_matrix_ny %>% arrange(desc(.resid_abs)) %>% head()

new_ny <- data.frame("ranking" = 1)
new_ny
predict(lm_ny, newdata=new_ny)

myny <- broom::augment(lm_ny, newdata = new_ny)
myny

ggplot(data=ny, aes(x=ranking, y=num_review)) + geom_point() +
  geom_smooth(method = 'lm') + geom_point(data=myny, aes(y=.fitted), size = 3, color = 'red') +
  labs(title='Num_review by Ranking with predict model')

ggpairs(data = ny, columns = 1:4)
cor(ny$ranking, ny$num_review)
```

Figure. 6

Findings

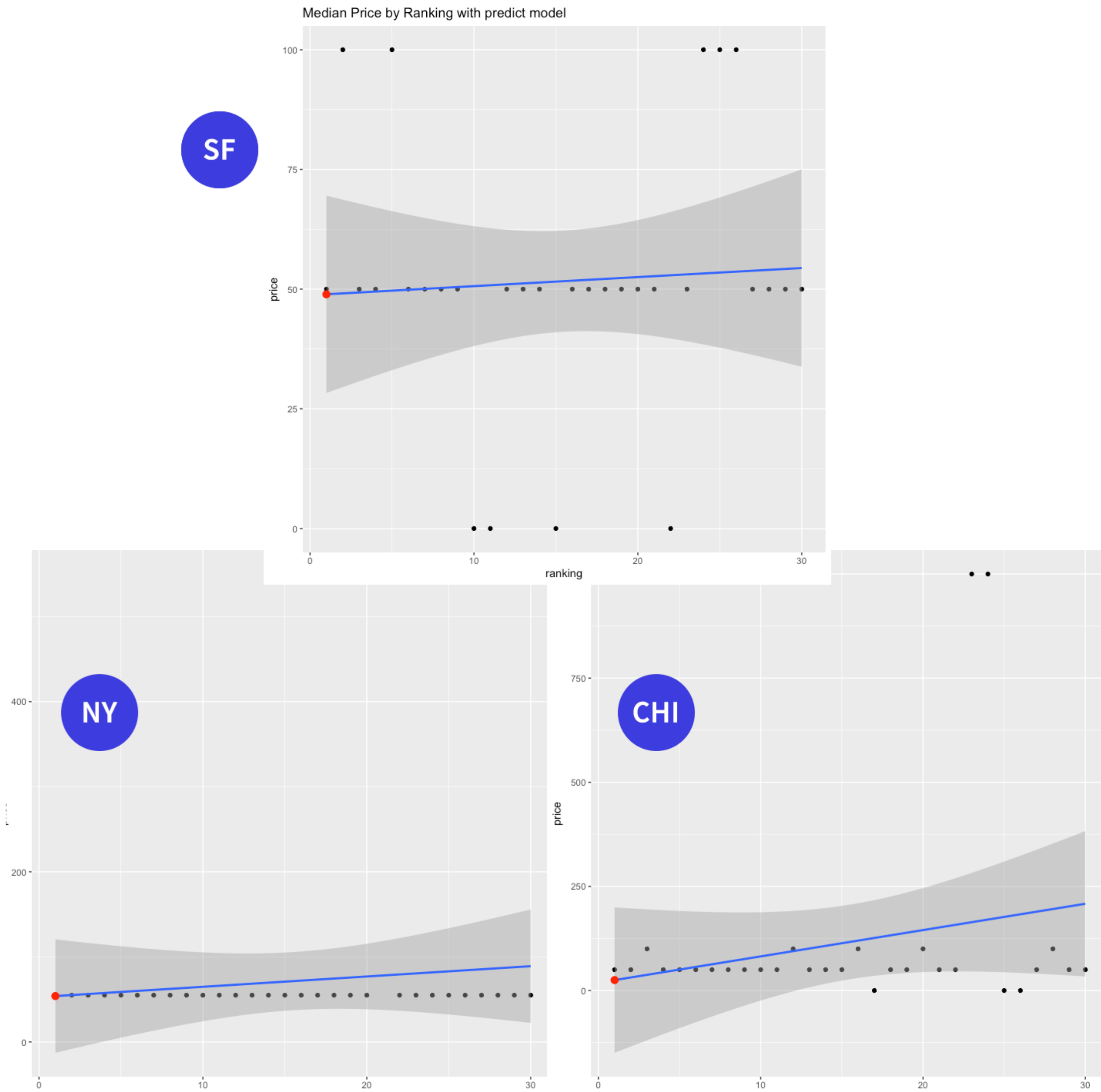
A higher ranking has a larger number of reviews, but San Fransisco has a non-linear regressional aspect.

- Correlation between num_reviews and ranking is strong and but not too dense.
- Unlikely, San Fransico has a wide variety number of reviews from ranking 1 to 30.



A higher ranking has moderately less expensive menus.

- Correlation between prices and ranking is moderate and dense.
- San Fransico takes place in higher average median prices out of thee cities.



Conclusion

Overall linear regression charts didn’t show a strong and dense relationship between two variables, though they are in a linear relationship. The predicted model doesn’t seem to capture an accurate prediction were a limitation of the dataset since each city has only 30 data points. The further studies on Yelp ranking will be needed with more volumes in the dataset and to compare multiple variables through gathering on the content page, such as a number of photos and the existence of the “Elite” reviewer.

Appendices

Appendix 1: Scraping data on Yelp using Beautiful soup on Jupyter notebook

Import the Libraries

```
from splinter import Browser
from bs4 import BeautifulSoup
import os
import pandas as pd
from datetime import datetime
import platform
import matplotlib.pyplot as plt
```

Create five functions to scrap data

- def init_browser(): Open a new Chrome window (For Window user, you need to download a file named 'chromedriver.exe')
- def get_html(browser, url): Get the html of url through new Chrome window
- get_data(html): Scrap the data in html (Find the
- def scrape(browser, url): Call two functions get_html(browser, url) and get_data(html)
- def main(): Call two functions init_browser() and scrape(browser, url) and return the dictionary that contain all dataset from url

```
data = {}

def init_browser():
    if platform.system().lower() == 'windows'.lower():
        executable_path = {
            'executable_path':
            os.path.join(os.getcwd(), 'chromedriver.exe')}
        return Browser('chrome', **executable_path, headless=False)
    else:
        return Browser('chrome')

def get_html(browser, url):
    browser.visit(url)
    html = browser.html
    return html

def get_data(html):
    soup = BeautifulSoup(html, "html.parser")
    ol_lists = soup.find('div', class_='lemon--div__373c0__1mboc mapColumnTransition__373c0__10KHB arrange-unit__373c0_

    ranks = ol_lists.find_all('p', class_='lemon--p__373c0__3Qnnj text__373c0__2pB8f text-color--black-regular__373c0__
    title = ol_lists.find_all('a', class_='lemon--a__373c0__IEZFH link__373c0__29943 link-color--blue-dark__373c0__1mhJ
    target_bf_rating = ol_lists.find_all('div', class_='lemon--div__373c0__1mboc attribute__373c0__1hPI_ display--inlin
    reviews = ol_lists.find_all('span', class_='lemon--span__373c0__3997G text__373c0__2pB8f reviewCount__373c0__2r4xT
    infos = ol_lists.find_all('div', class_='lemon--div__373c0__1mboc mainAttributes__373c0__1r0QA arrange-unit__373c0_
    addresses = ol_lists.find_all('address', class_='lemon--address__373c0__2sPac')
    time = datetime.now()

    if len(ranks) > 30 or len(ranks) < 33:
        diff = len(ranks) - 30
        ranks = ranks[diff:]
        target_bf_rating = target_bf_rating[diff:]
        infos = infos[diff:]
    if len(ranks) == 33:
        ranks = ranks[2:-1]
        target_bf_rating = target_bf_rating[2:-1]
        infos = infos[2:-1]

    for i in range(len(ranks)):
        rank = [p.text.split('.')[0] for p in ranks]
        title = [a.a.text for a in ranks]
        rating = [a.span.div for a in target_bf_rating if a.span]
        num_review = [a.text.split()[0]for a in reviews]
        price = [a.div.div.find_next_sibling('div').find_next_sibling('div') for a in infos]
        types = [b.find_all('a', class_='lemon--a__373c0__IEZFH link__373c0__29943 link-color--inherit__373c0__15ymx li

        address = [a.div.div.p.span.text for a in addresses]
        neighbourhood = [a.find_next_sibling('div') for a in addresses]

        data[rank[i]] = {'title': title[i],
                         'rating': rating[i]['aria-label'].split(' ')[0],
                         'num_review': num_review[i],
                         'price': ''.join([a for a in price[i].div.div.span.span.text if a is '$' or a is '$$' or a is
                         'types': [b.text for b in types[i]],
                         'address': address[i],
                         'neighbourhood': ''.join([a.div.div.p.text for a in neighbourhood[i]]),
                         'time': time}
    return data

def scrape(browser, url):
    html = get_html(browser, url)
    datas = get_data(html)
    return len(datas), datas

def main():
    browser = init_browser()
    url = "https://www.yelp.com/search?cflt=restaurants&find_loc=New+York%2C+NY"
#     url = "https://www.yelp.com/search?cflt=restaurants&find_loc=San+Francisco%2C+CA"
#     url = "https://www.yelp.com/search?cflt=restaurants&find_loc=Chicago%2C+IL"
    datas = scrape(browser, url)
    return datas

main()
df = pd.DataFrame(data).T
df.head()
```

Appendix 2: Using linear regression and ggpairs on Yelp on R studio

```r
install.packages("broom")
install.packages("GGally")

library(tidyverse)
library(lubridate)
library(broom)
library(GGally)

raw_ny <- read.csv("/Users/hh/Documents/Pratt/Data_Analytics/Data_analytics_labs/02_Predictive Data Analysis/1. Gathering
raw_sf <- read.csv("/Users/hh/Documents/Pratt/Data_Analytics/Data_analytics_labs/02_Predictive Data Analysis/1. Gathering
raw_chi <- read.csv("/Users/hh/Documents/Pratt/Data_Analytics/Data_analytics_labs/02_Predictive Data Analysis/1. Gathering



## 1 --- Using raw_ny dataset, pick two variables "price" and "num_review" --- ##
ny <- raw_ny[, c(1, 3, 4, 5)]
glimpse(ny)

lm_ny <- lm(price ~ ranking, data=ny)
summary(lm_ny)

fitted_ny <- fitted.values(lm_ny)
fitted_ny
residuals_ny <- residuals(lm_ny)
residuals_ny

lm_matrix_ny <- broom::augment(lm_ny)
head(lm_matrix_ny)
lm_matrix_ny$.resid_abs <- abs(lm_matrix_ny$.resid)
lm_matrix_ny %>% arrange(desc(.resid_abs)) %>% head()

new_ny <- data.frame("ranking" = 1)
new_ny
predict(lm_ny, newdata=new_ny)

myny <- broom::augment(lm_ny, newdata = new_ny)
myny

ggplot(data=ny, aes(x=ranking, y=price)) + geom_point() +
  geom_smooth(method = 'lm') + geom_point(data=myny, aes(y=.fitted), size = 3, color = 'red') +
  labs(title='Median Price by Ranking with predict model')

ggpairs(data = ny, columns = 1:4)
cor(ny$ranking, ny$price)



## --- 2 Using raw_sf dataset, pick two variables "price" and "num_review" --- ##
sf <- raw_sf[, c(1, 3, 4, 5)]
glimpse(sf)

lm_sf <- lm(price ~ ranking, data=sf)
summary(lm_sf)

fitted_sf <- fitted.values(lm_sf)
fitted_sf
residuals_sf <- residuals(lm_sf)
residuals_sf

lm_matrix_sf <- broom::augment(lm_sf)
head(lm_matrix_sf)
lm_matrix_sf$.resid_abs <- abs(lm_matrix_sf$.resid)
lm_matrix_sf %>% arrange(desc(.resid_abs)) %>% head()

new_sf <- data.frame("ranking" = 1)
new_sf
predict(lm_sf, newdata=new_sf)

mysf <- broom::augment(lm_sf, newdata = new_sf)
mysf

ggplot(data=sf, aes(x=ranking, y=price)) + geom_point() +
  geom_smooth(method = 'lm') + geom_point(data=mysf, aes(y=.fitted), size = 3, color = 'red') +
  labs(title='Median Price by Ranking with predict model')

ggpairs(data = sf, columns = 1:4)
cor(sf$ranking, sf$price)
```

References

Yelp Ranking Factors, Retrieved from
http://www.localvisibilitysystem.com/2012/08/23/yelp-ranking-factors/

Myth Busting Yelp for Business, Retrieved from
https://www.chatmeter.com/blog/myth-busting-yelp-for-business/

Jupyter/IPython Notebook Quick Start Guide. Retrieved from:
https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html