# CPSC 4810 W01 Project Report

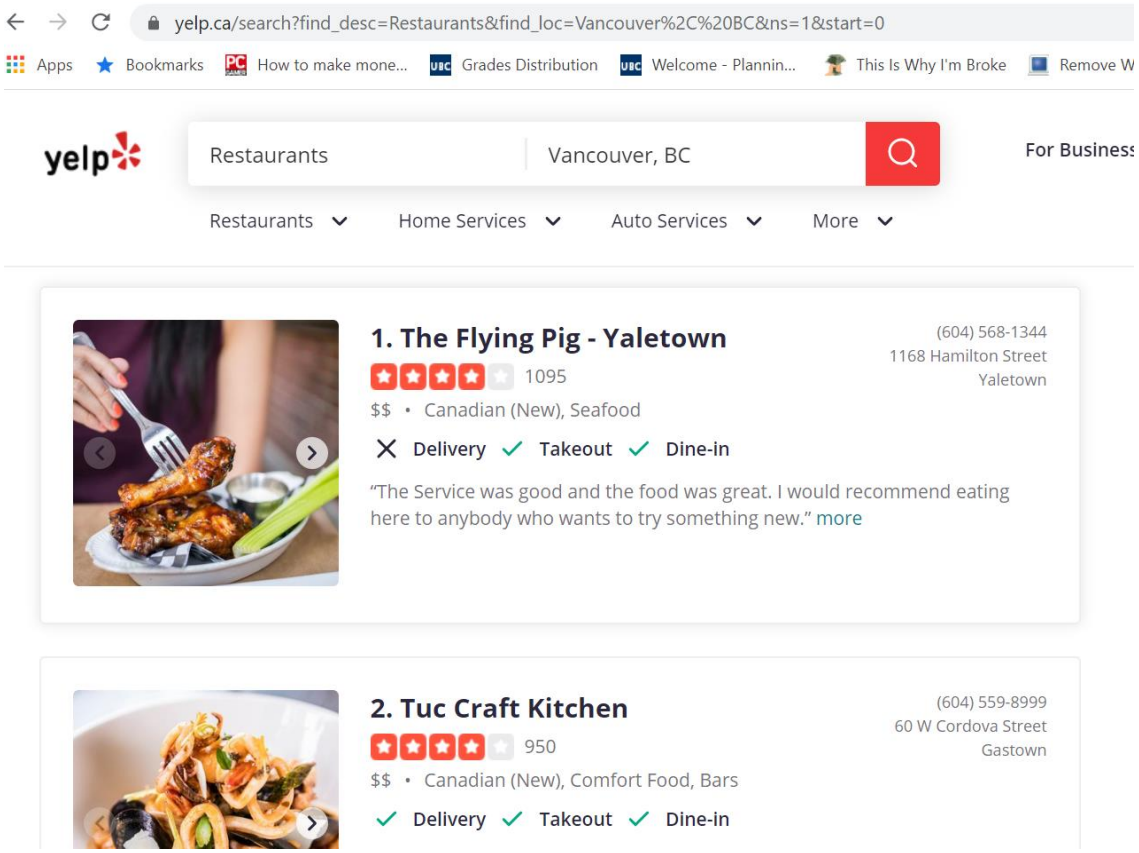
## Group 1 (Allen Jawahar Masilamani, Chandy George, Chun Ching Look, David Shih)

## Project Topic: Web Scraping of Yelp with Data Visualization

Web Scraping Info from Yelp.ca Tutorial:

```
# all the libraries we used for this project

from pymongo import MongoClient
import pandas as pd
import numpy as np
import requests
import re
import bs4 as bs
import urllib
import itertools
import seaborn as sns
from matplotlib import pyplot as plt
```

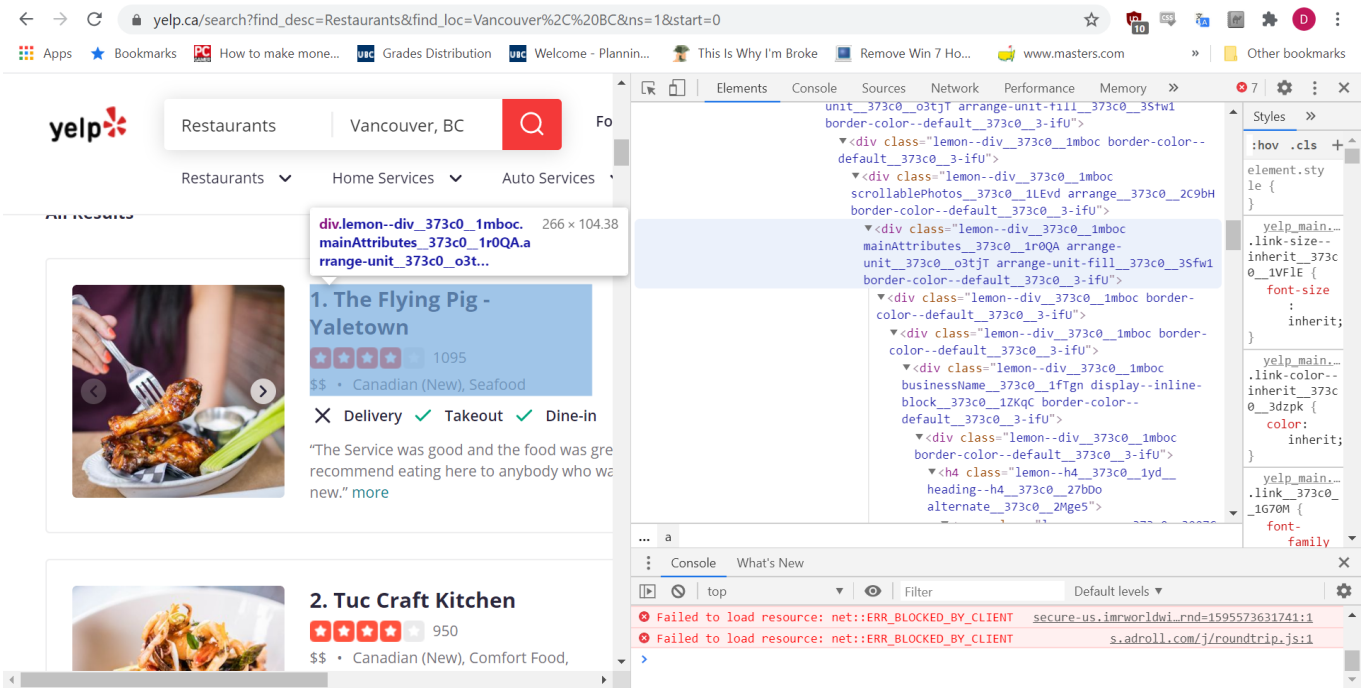
```
#the url of yelp that we will be using starting at page 1
source = url.urlopen('https://www.yelp.ca/search?find_desc=Restaurants&find_loc=Vancouver%2C%20BC&ns=1&start=0')
```




```
#passing on the html.parser to beautifulsoup
page_soup = bs.BeautifulSoup(source, 'html.parser')
```

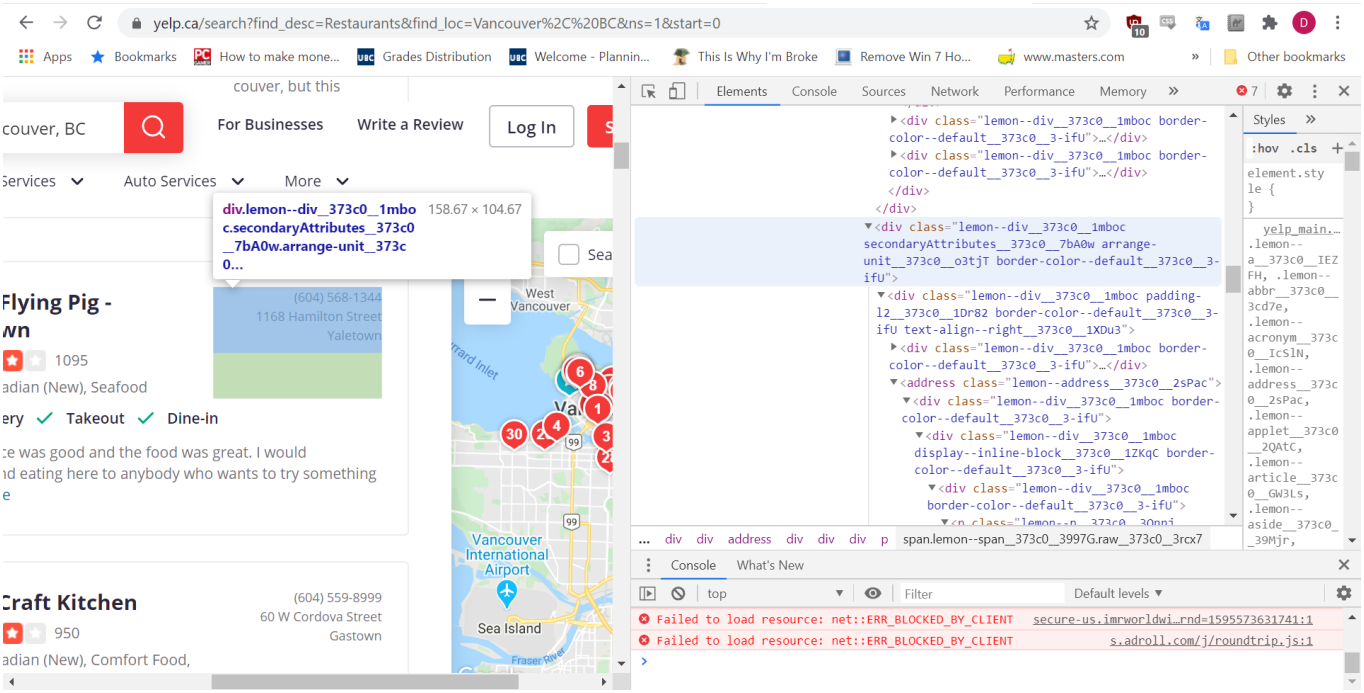
## Main Attributes:

```
#the info we want to scrape is divided into 2 parts: Main attributes & Secondary attributes
#main contains restaurant name, cuisine, rest rating, # of reviews, price
#secondary contains address, phone# and neighbourhood
mains = page_soup.find_all("div",
{"class": "lemon--div__373c0__1mboc mainAttributes__373c0__1r0QA arrange-unit__373c0__o3tjT arrange-unit-fill__373c0__3Sfw1 borde
```

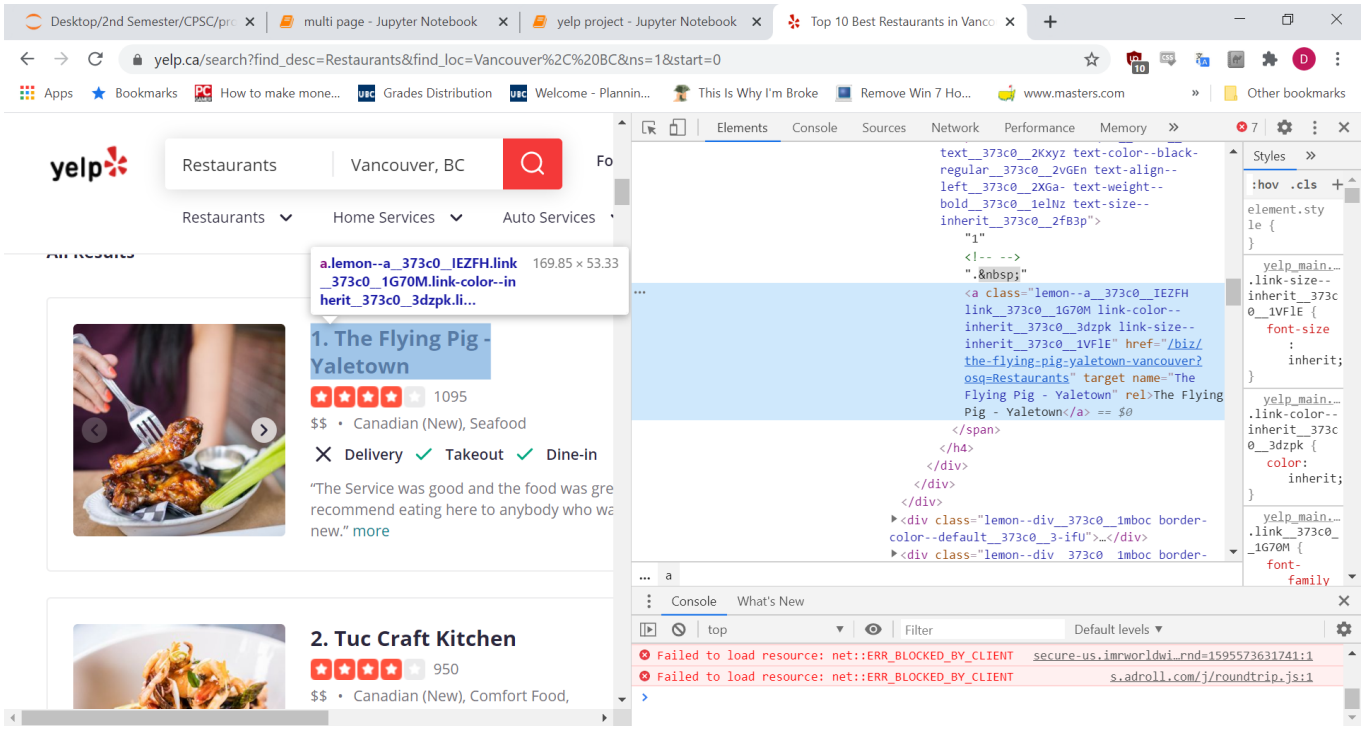



## Secondary Attributes:

```
#secondary attributes
second = page_soup.find_all("div", {"class":"lemon--div__373c0__1mboc secondaryAttributes__373c0__7bA0w arrange-unit__373c0__o3tj
```



Restaurant name under ‘a’ class:



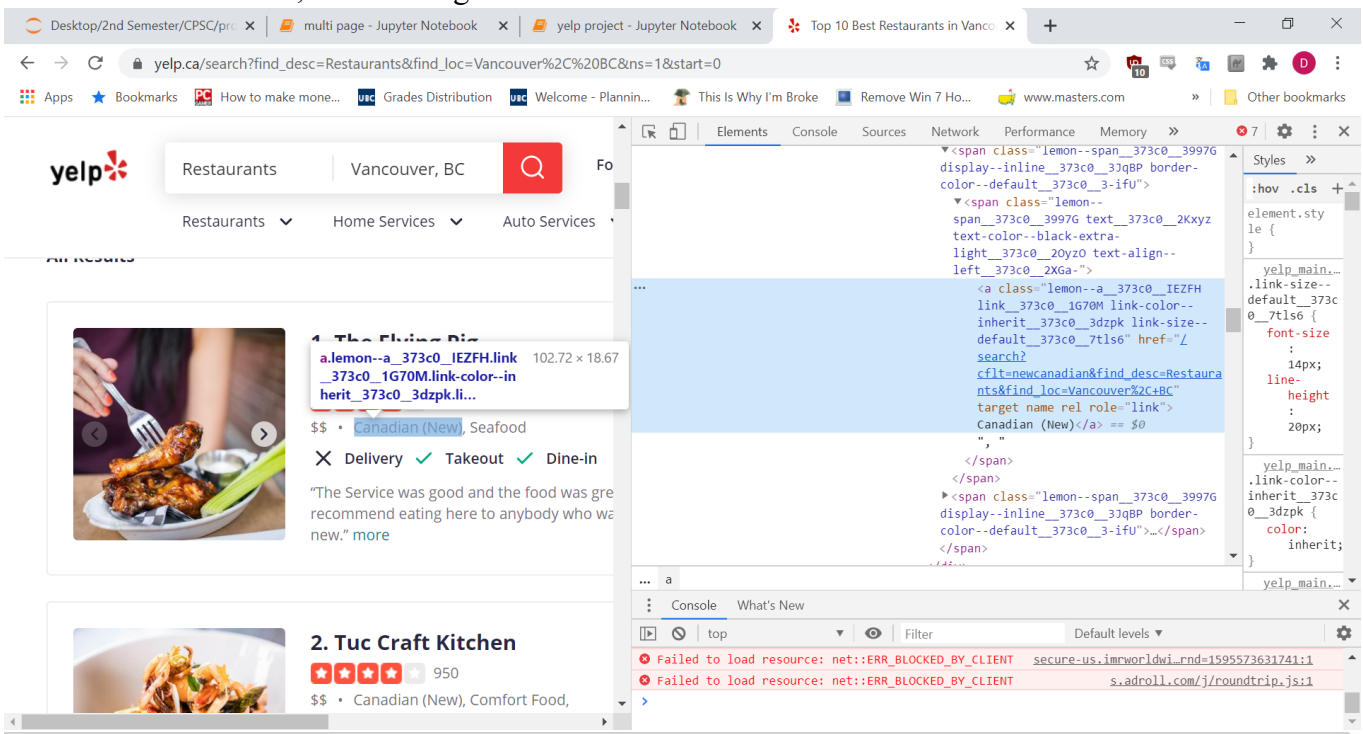

```
#restaurant name is under the "a class". Ran code to do a quick test to see if we can scrape business name of restaurant #1
main = mains[1]
name = main.find('a').text
print(name)
```

```
The Flying Pig - Yaletown
```

```
#using a for loop to print all restaurant names. We want to be able to debug quickly if we cant scrape the restaurant name.
#if we are unable to find a restaurant name, the code will print "none" allowing us to see which row the code went wrong.
#we were able to scrape all the names in page1
for main in mains:
    try:
        name = main.find('a').text
        print('rest name: '+ name)
    except:
        print(none)
```

```
rest name: Spicy 6 Fine Indian Cuisine
rest name: The Flying Pig - Yaletown
rest name: Fable
rest name: Tuc Craft Kitchen
rest name: Saku
rest name: The Flying Pig - Gastown
rest name: Guu with Garlic
rest name: Lunch Lady
rest name: Kokoro Tokyo Mazesoba
rest name: Dinesty Dumpling House
rest name: Phnom Penh
rest name: Chambar
rest name: Fanny Bay Oyster Bar & Shellfish Market
rest name: Blue Water Cafe
rest name: Elisa
rest name: Alibi Room
rest name: Miku
rest name: La Taqueria Pinche Taco Shop
```

Cuisine under ‘a’ class, different tag:

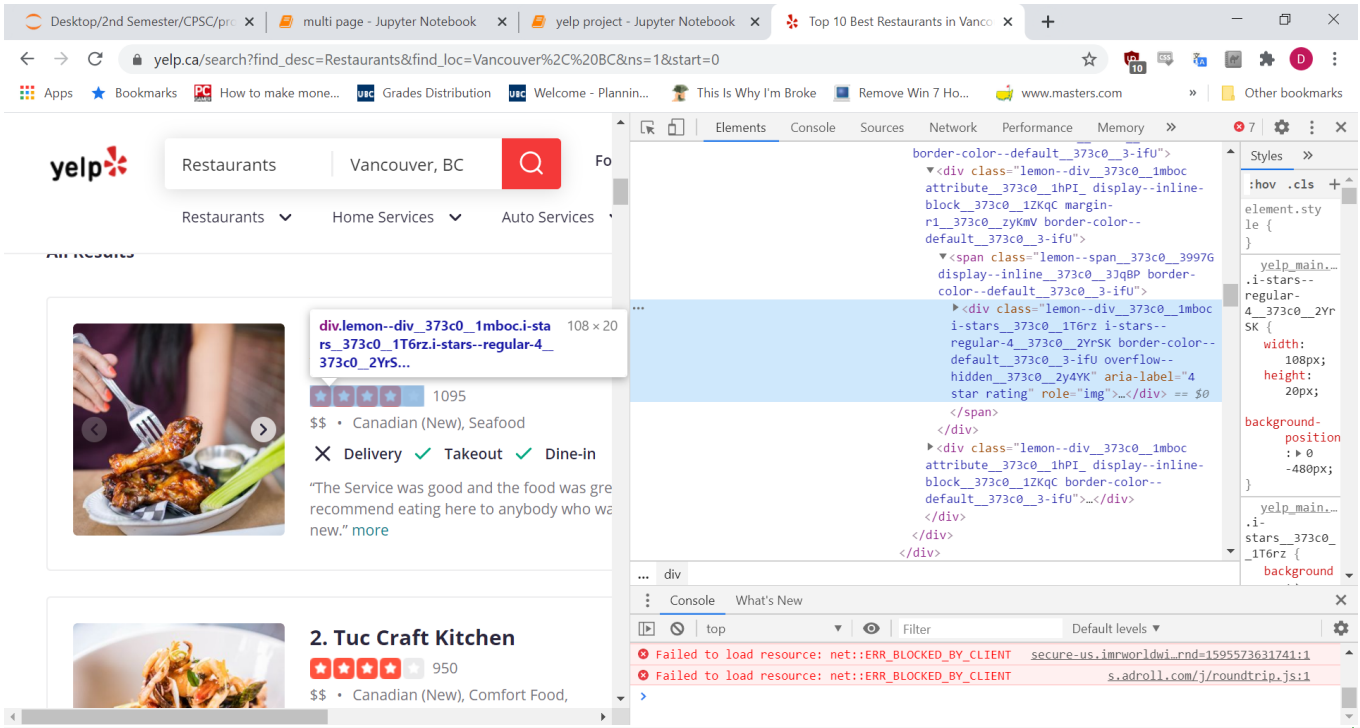



```
#cuisine tag
<a class="lemon--a__373c0__IEZFH link__373c0__1G70M link-color--inherit__373c0__3dzpk link-size--default__373c0__7tls6" href="/se
```

```
#using a for loop to print all the cuisine type for restaurants.
# same idea: we want to be able to debug quickly if we cant scrape the type of cuisine.
#if we are unable to find a cuisine type, the code will print "none" allowing us to see which row the code went wrong.
for main in mains:
    try:
        cuisine = main.find('a', {'class': 'lemon--a__373c0__IEZFH link__373c0__1G70M link-color--inherit__373c0__3dzpk link-size
        print('Cuisine: '+ cuisine)
    except:
        print('none')
```

```
Cuisine: Coffee & Tea
Cuisine: Canadian (New)
Cuisine: Japanese
Cuisine: Canadian (New)
Cuisine: Canadian (New)
Cuisine: Vietnamese
Cuisine: Chinese
Cuisine: Cambodian
Cuisine: Japanese
Cuisine: Canadian (New)
Cuisine: Chicken Shop
```

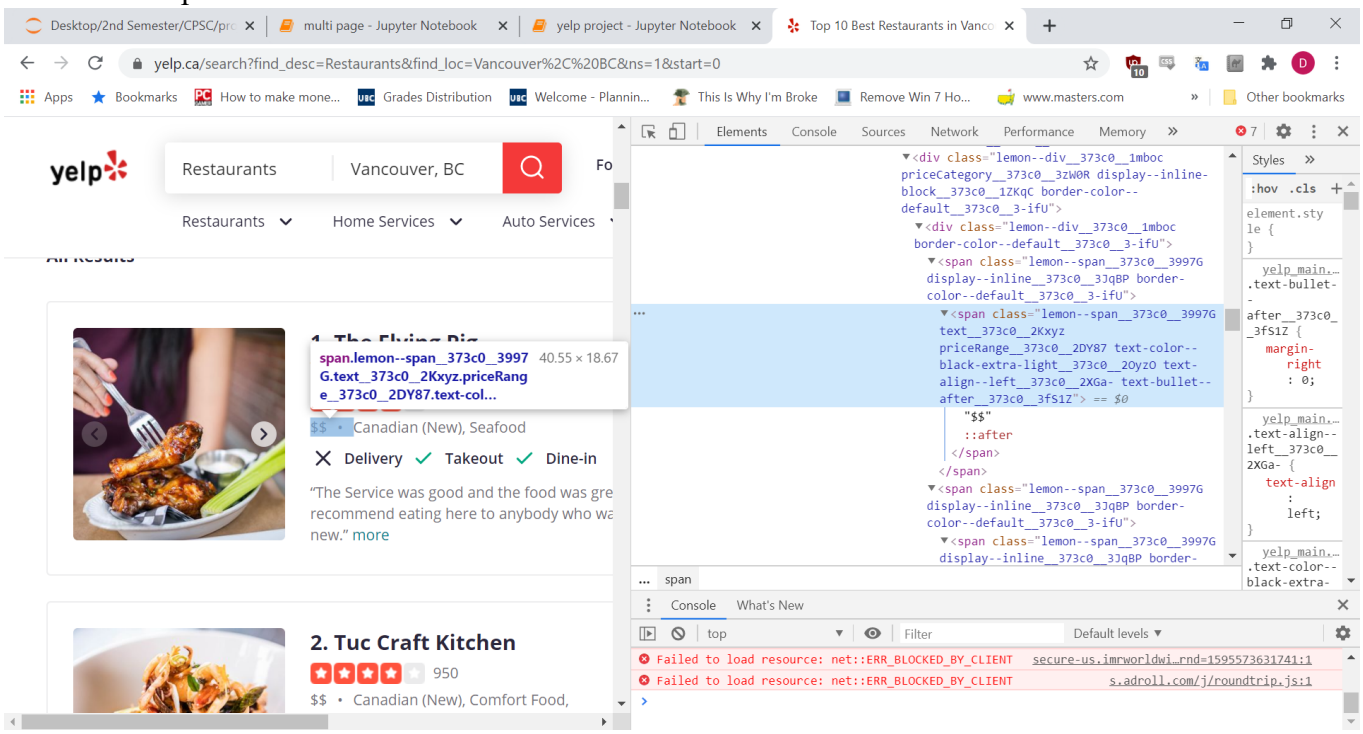
Number of reviews under same class but after ‘aria-label’:




```
#now for num reviews
for main in mains:
    try:
        reviews=(main.find("span", {"class": "lemon--span__373c0__3997G text__373c0__2Kxyz reviewCount__373c0__2r4xT text-color--
        print('Num of reviews: '+ reviews)
    except:
        print(None)
```

```
None
Num of reviews: 950
Num of reviews: 114
Num of reviews: 1095
Num of reviews: 676
Num of reviews: 10
Num of reviews: 776
Num of reviews: 1303
Num of reviews: 928
Num of reviews: 931
Num of reviews: 200
Num of reviews: 6
Num of reviews: 12
Num of reviews: 900
Num of reviews: 30
Num of reviews: 326
Num of reviews: 254
Num of reviews: 10
```

Price under ‘span’:

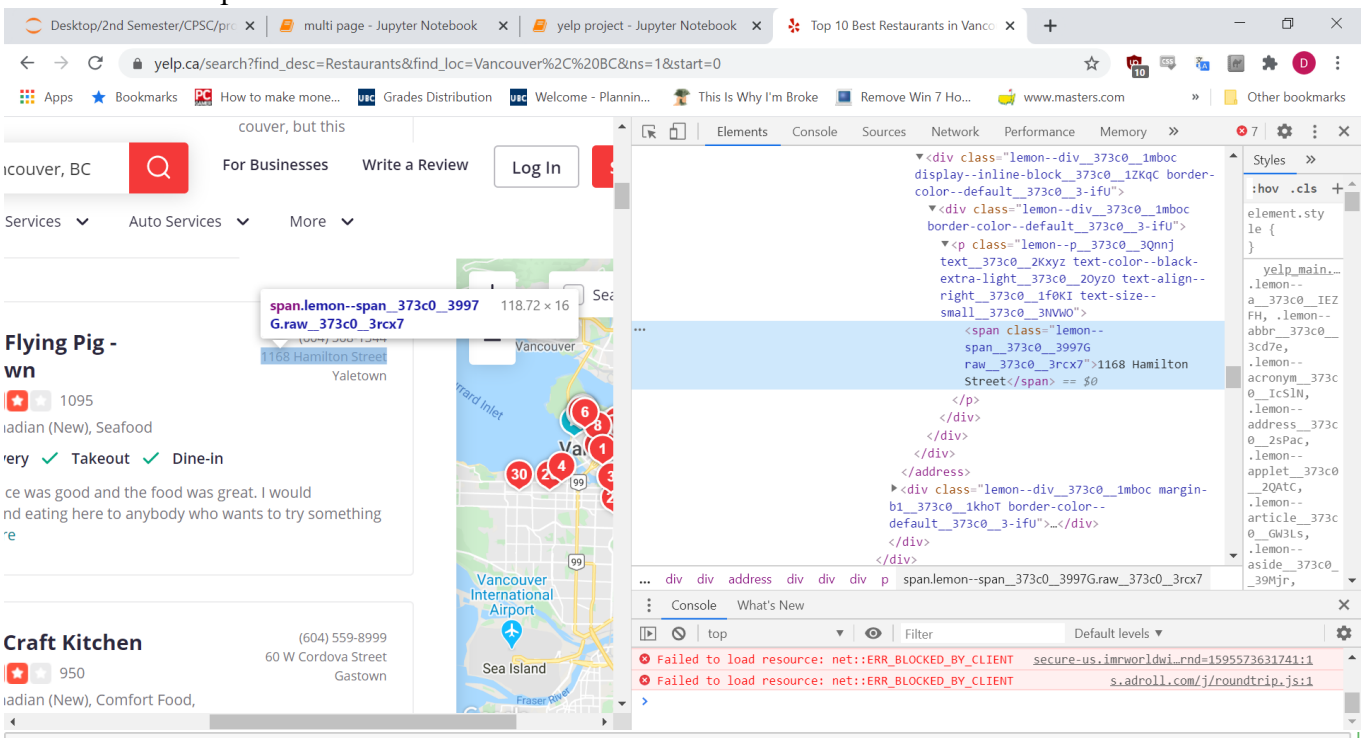



```
#and for price
#here, a result of none does not mean a bug in our code. From looking at the yelp page, a new restaurant will not have enough
#reviews to determine a price point.
for main in mains:
    try:
        price=(main.find("span", {"class": "lemon--span__373c0__3997G text__373c0__2Kxyz priceRange__373c0__2DY87 text-color--bla
        print('Price: '+ price)
    except:
        print(None)
```

```
None
Price: $$
Price: $$
Price: $$
Price: $$
None
Price: $$
Price: $$
Price: $$
Price: $$
Price: $$
```

Now scrapping address, tel# and neighbourhood under secondary attributes:

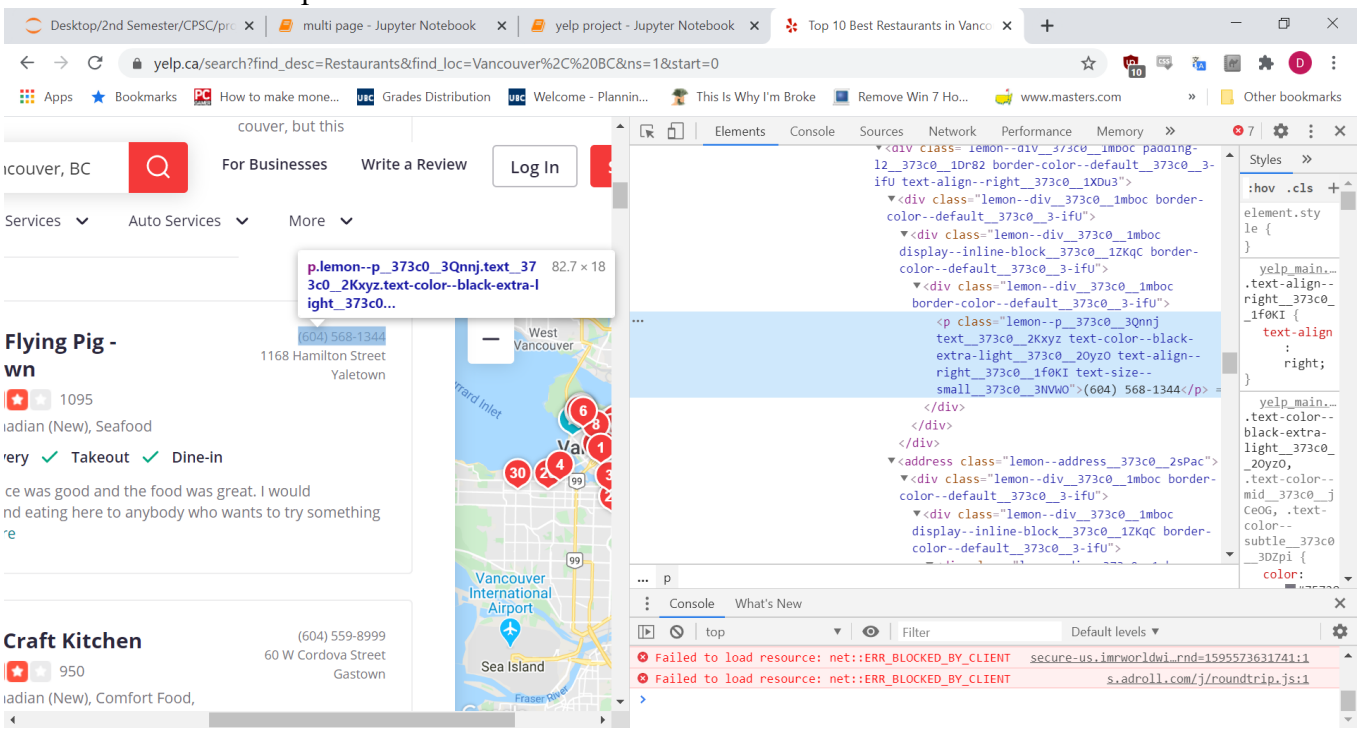
Address under ‘span’:




```
#same idea as before, now for secondary attributes. Find class + tag and scrape data
#run for loop to obtain addresses
for sec in second:
    try:
        address = sec.find("span", {"class": "lemon--span__373c0__3997G raw__373c0__3rcx7"}).text
        print('Address: '+ address)
    except:
        print(None)
```

```
Address: 1116 Robson Street
Address: 1168 Hamilton Street
Address: 1944 W 4th Avenue
Address: 60 W Cordova Street
Address: 548 W Broadway
Address: 102 Water Street
Address: 1698 Robson Street
Address: 1046 Commercial Drive
Address: 551 Seymour Street
Address: 1719 Robson Street
Address: 244 E Georgia Street
Address: 568 Beatty Street
Address: 762 Cambie St
Address: 1095 Hamilton St
Address: 1109 Hamilton Street
Address: 157 Alexander Street
Address: 200 Granville Street
```

Phone numbers under ‘p’:




```
# for loop to obtain tel#
#some phone# werent in the right tags and classes, will have to manually remove and filter after
tel = []
for sec in second:
    try:
        tel.append(sec.find("p" , {"class": "lemon--p__373c0__3Qnnj text__373c0__2Kxyz text-color--black-extra-light__373c0__2Oyz
    except:
        tel.append("None")
```

```
tel
```

```
['(604) 566-9666',
 '(604) 559-8999',
 '548 W Broadway',
 '(604) 568-1344',
 '(604) 732-1322',
 '(604) 559-5938',
 '(604) 669-7769',
 '(604) 428-8414',
 '(604) 682-5777',
 '(604) 685-8678',
 '(604) 559-7968',
 '(604) 331-0058',
```

To remove unwanted values from our list of telephone numbers, we used a simple regex line to keep all telephone #s and replace all none telephone #’s with ‘None’ values:

```
#regex to filter proper phone#s
new_tel = [x if (bool(re.search(r'[0-9]{10}|[0-9]{3}-[0-9]{3}-[0-9]{4}|\([0-9]{3}\)[0-9]{3}-[0-9]{4}|
```

```
\(\d\d\d\)\s\d\d\d-\d\d\d\d', x)) == True) else "None" for x in tel]
```

Noticed how 3rd line is replaced with ‘None’:

```
new_tel
```

```
['(604) 566-9666',
 '(604) 559-8999',
 'None',
 '(604) 568-1344',
 '(604) 732-1322',
 '(604) 559-5938',
 '(604) 669-7769',
 '(604) 428-8414',
 '(604) 682-5777',
 '(604) 685-8678',
 '(604) 559-7968',
 '(604) 331-0058',
 '(604) 283-1385',
```

Now, we had an issue with scrapping neighbourhood as it was under the same tag + class as telephone numbers, have to derive alternative to scrape neighbourhood information.

i.e.:

Telephone #:

(604) 568-1344

Neighbourhood:

Yaletown

First step, scrape data for everything under ‘p’:

```
#neighbourhood class
# there is problem with neighbourhood class. The class contains info for phone #, address, and neighbourhood.
#we will have to filter out neighbourhood and remove unwanted items
[item.get_text() for item in page_soup.find_all('p', class_ =
'lemon--p__373c0__3Qnnj text__373c0__2Kxyz text-color--black-extra-light__373c0__2OyzO text-align--right__373c0__1f0KI text-size-
```

```
['(604) 566-9666',
 '1778 Columbia Street',
 'Mount Pleasant',
 '(604) 559-8999',
 '60 W Cordova Street',
 'Gastown',
 '548 W Broadway',
 'Fairview Slopes',
 '(604) 568-1344',
 '1168 Hamilton Street',
 'Yaletown',
 '(604) 732-1322',
 '1944 W 4th Avenue',
 'Kitsilano',
 '(604) 559-5938',
 '1046 Commercial Drive',
 'Grandview-Woodlands',
 '(604) 669-7769',
 '1719 Robson Street',
 'West End',
```

We used a simple regex line to replace telephone numbers and addresses with 'None'.

```
n = [item.get_text() for item in page_soup.find_all('p', class_ = 'lemon--p__373c0__3Qnnj text__373c0__2Kxyz text-color--black-e
#n1 to change phone num and address to None
n1 = [x if (bool(re.search(r'\d+', x)) == False) else "None" for x in n]
n1
```

```
['None',
 'None',
 'Mount Pleasant',
 'None',
 'None',
 'Gastown',
 'None',
 'Fairview Slopes',
 'None',
 'None',
 'Yaletown',
 'None',
 'None',
 'Kitsilano',
 'None',
 'None',
 'Grandview-Woodlands',
 'None',
 'None',
 'West End',
```

Then we simply removed the 'None' values from the list using a quick filter:

```
n = [item.get_text() for item in page_soup.find_all('p', class_ = 'lemon--p__373c0__3Qnnj text__373c0__2Kxyz text-color--black-e
#n1 to change phone num and address to None
n1 = [x if (bool(re.search(r'\d+', x)) == False) else "None" for x in n]
n2 = list(filter(lambda x: x != 'None', n1))
n2
```

```
['Mount Pleasant',
 'Gastown',
 'Fairview Slopes',
 'Yaletown',
 'Kitsilano',
 'Grandview-Woodlands',
 'West End',
 'West End',
 'Strathcona',
 'West End',
 'Gastown',
 'West End',
 'Grandview-Woodlands',
 'West End',
 'Yaletown',
 'Downtown',
```

Now to combine everything into a dataframe, we first create empty list and used the codes above to append values into the list:

```
#appending everything to a list
name = []
cuisine = []
ratings = []
reviews = []
price = []
add = []
tel = []
#dont need list for neighbourhood because it is already in a list
```

```
for main in mains:
    try:
        name.append(main.find('a').text)
    except:
        name.append('none')
    try:
        cuisine.append(main.find('a', {'class': 'lemon--a__373c0__IEZFH link__373c0__1G70M link-color--inherit__373c0__3dzpk link
    except:
        cuisine.append('none')
    try:
        ratings.append(main.find('span', {'class': 'lemon--span__373c0__3997G display--inline__373c0__3JqBP border-color--default
    except:
        ratings.append('none')
    try:
        reviews.append(main.find("span", {"class": "lemon--span__373c0__3997G text__373c0__2Kxyz reviewCount__373c0__2r4xT text-
    except:
        reviews.append('none')
    try:
        price.append(main.find("span", {"class": "lemon--span__373c0__3997G text__373c0__2Kxyz priceRange__373c0__2DY87 text-colo
    except:
        price.append('none')
```

```
#just combining all the loops together for secondary attributes
for sec in second:
    try:
        add.append(sec.find("span", {"class": "lemon--span__373c0__3997G raw__373c0__3rcx7"}).text)
    except:
        add.append("None")
    try:
        tel.append(sec.find("p" , {"class": "lemon--p__373c0__3Qnnj text__373c0__2Kxyz text-color--black-extra-light__373c0__2Oyz
    except:
        tel.append("None")

n = [item.get_text() for item in page_soup.find_all('p', class_ = 'lemon--p__373c0__3Qnnj text__373c0__2Kxyz text-color--black-ex
n1 = [x if (bool(re.search(r'\d+', x)) == False) else "None" for x in n]
n2 = list(filter(lambda x: x != 'None', n1))
new_tel = [x if (bool(re.search(r'[0-9]{10}|[0-9]{3}-[0-9]{3}-[0-9]{4}|\([0-9]{3}\)[0-9]{3}-[0-9]{4}|\(\d\d\d\)\s\d\d\d-\d\d\d\d
```

Using the information we scrapped to create a dataframe:

```
#combining everything to a dataframe
data = {}
#giving values
data = {'Restaurant_Name': name, 'Cuisine': cuisine, 'Restaurant_Ratings': ratings, 'Restaurant_Reviews': reviews,
        'Restaurant_Price': price, 'Restaurant_Address': add, 'Restaurant_Telephone': new_tel, 'Neighbourhood': n2}
rest = pd.DataFrame(data)
#giving headers for our csv output if needed
header = ["Restaurant_Name", "Cuisine", "Restaurant_Ratings", "Restaurant_Reviews", "Restaurant_Price", "Restaurant_Address", "Re
#extra line to convert dataframe to an csv file if needed:
#rest.to_csv("Output.csv", columns = header)
```

```
rest
```

| | Restaurant_Name | Cuisine | Restaurant_Ratings | Restaurant_Reviews | Restaurant_Price | Restaurant_Address | Restaurant_Telephone | Neighbourhood |
|---|---|---|---|---|---|---|---|---|
| 0 | Beeryani Indian Bistro & Bar | Indian | 5 star rating | 10 | none | 1184 Denman Street | (604) 609-9999 | West End |
| 1 | The Flying Pig - Yaletown | Canadian (New) | 4 star rating | 1095 | $$ | 1168 Hamilton Street | (604) 568-1344 | Yaletown |
| 2 | Saku | Japanese | 4.5 star rating | 116 | $$ | 548 W Broadway | None | Fairview Slopes |
| 3 | Tuc Craft Kitchen | Canadian (New) | 4 star rating | 950 | $$ | 60 W Cordova Street | (604) 559-8999 | Gastown |
| 4 | Fable | Canadian (New) | 4.5 star rating | 676 | $$ | 1944 W 4th Avenue | (604) 732-1322 | Kitsilano |
| 5 | Lunch Lady | Vietnamese | 4.5 star rating | 10 | none | 1046 Commercial Drive | (604) 559-5938 | Grandview-Woodlands |
| 6 | The Flying Pig - Gastown | Canadian (New) | 4 star rating | 932 | $$ | 102 Water Street | (604) 559-7968 | Gastown |
| 7 | Dinesty Dumpling House | Chinese | 4 star rating | 777 | $$ | 1719 Robson Street | (604) 669-7769 | West End |
| 8 | Kosoo | Korean | 4.5 star rating | 6 | none | 1128 Robson Street | (604) 428-8414 | West End |
| 9 | Phnom Penh | Cambodian | 4 star rating | 1303 | $$ | 244 E Georgia Street | (604) 682-5777 | Strathcona |
| 10 | Blue Water Cafe | Seafood | 4.5 star rating | 902 | $$$ | 1095 Hamilton St | (604) 688-8078 | Yaletown |

Now, for multi page scraping.
Defined a quick function for BeautifulSoup:

```
def make_soup(url):
    source = urllib.request.urlopen(url)
    page_soup = bs.BeautifulSoup(source, 'html.parser')
    return page_soup
```

Yelp Url for page one:
https://www.yelp.ca/search?find_desc=Restaurants&find_loc=Vancouver%2C%20BC&ns=1&start=0

Page 2: https://www.yelp.ca/search?find_desc=Restaurants&find_loc=Vancouver%2C%20BC&ns=1&start=30

Page 3: https://www.yelp.ca/search?find_desc=Restaurants&find_loc=Vancouver%2C%20BC&ns=1&start=60

The pattern is that the ending jumps by 30. We created a quick for loop to scrape multiple pages at a time:

```
for x in range(0, 61, 30):
    print(x)
```

```
0
30
60
```

```
for i in range(0, 61, 30):
    soup = make_soup('https://www.yelp.ca/search?find_desc=Restaurants&find_loc=Vancouver%2C%20BC&ns=1&start=' + str(i))
```

The biggest issue was with scrapping neighbourhood. Since the values for neighbourhood was scrapped different compared to the rest of the information, the codes for neighbourhood scrapping is different.

For each page that we had to scrape, the values was stored in an nested list. When we scrapped 2 pages worth of Yelp data, instead of giving 62 values (31 per page), the list inside neighbour only contained 2 values each containing another 31 values inside.

We used itertools to flatten the list:

```
#had issue with length of neighbourhood not being the same as the other lists
#realized that neighbourhood was a nested list
#had to un-nest the list to make all the list the same length
neigh = []
for i in range(0, 31, 30):
    soup = make_soup('https://www.yelp.ca/search?find_desc=Restaurants&find_loc=Vancouver%2C%20BC&ns=1&start=' + str(i))
    n = [item.get_text() for item in soup.find_all('p', class_ = 'lemon--p__373c0__3Qnnj text__373c0__2Kxyz text-color--black-ext
    n1 = [x if (bool(re.search(r'\d+', x)) == False) else "None" for x in n]
    n2 = list(filter(lambda x: x != 'None', n1))
    neigh.append(n2)
```

```
len(neigh)
```

```
2
```

```
flat  = list(itertools.chain(*neigh))
len(flat)
```

```
62
```

itertools.chain(*iterables) Make an iterator that returns elements from the first iterable until it is exhausted, then proceeds to the next iterable, until all of the iterables are exhausted. Used for treating consecutive sequences as a single sequence. Roughly equivalent to:

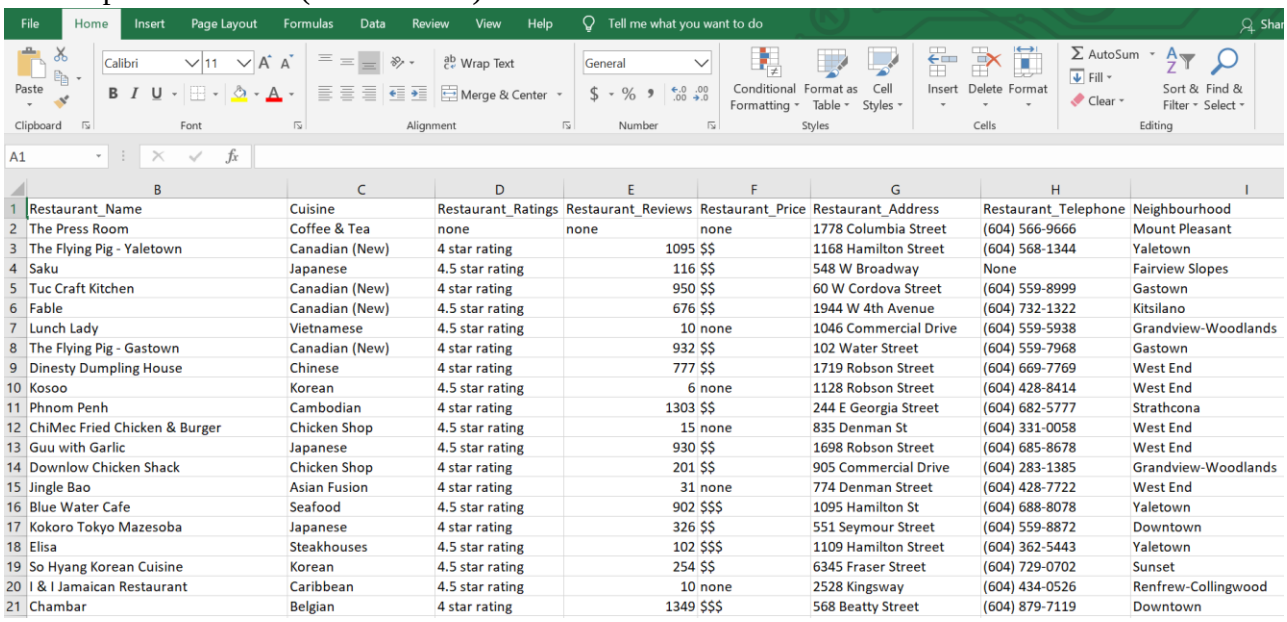
```
def chain(*iterables):
    # chain('ABC', 'DEF') --> A B C D E F
    for it in iterables:
        for element in it:
            yield element
```

Now, combining everything together, scrapping all available pages/restaurants in Vancouver and putting it into a csv file:

```
# multi page scrapping to CSV output

name = []
cuisine = []
ratings = []
reviews = []
price = []
add = []
tel = []
neigh = []

for i in range(0, 211, 30):
    soup = make_soup('https://www.yelp.ca/search?find_desc=Restaurants&find_loc=Vancouver%2C%20BC&ns=1&start=' + str(i))
    mains = soup.find_all("div", {"class": "lemon--div__373c0__1mboc mainAttributes__373c0__1r0QA arrange-unit__373c0__o3tjT arra
    second = soup.find_all("div", {"class":"lemon--div__373c0__1mboc secondaryAttributes__373c0__7bA0w arrange-unit__373c0__o3tj
    for main in mains:
        try:
            name.append(main.find('a').text)
        except:
            name.append('none')
        try:
            cuisine.append(main.find('a', {'class': 'lemon--a__373c0__IEZFH link__373c0__1G70M link-color--inherit__373c0__3dzpk
        except:
            print('none')
        try:
            ratings.append(main.find('span', {'class': 'lemon--span__373c0__3997G display--inline__373c0__3JqBP border-color--de
        except:
            ratings.append('none')
        try:
            reviews.append(main.find("span", {"class": "lemon--span__373c0__3997G text__373c0__2Kxyz reviewCount__373c0__2r4xT te
        except:
            reviews.append('none')
        try:
            price.append(main.find("span", {"class": "lemon--span__373c0__3997G text__373c0__2Kxyz priceRange__373c0__2DY87 text-
        except:
            price.append('none')

    for sec in second:
        try:
            add.append(sec.address.find("p", {"class": "lemon--p__373c0__3Qnnj text__373c0__2Kxyz text-color--black-extra-light__
        except:
            add.append("None")
        try:
            tel.append(sec.div.div.text)
        except:
            tel.append("None")
    n = [item.get_text() for item in soup.find_all('p', class_ = 'lemon--p__373c0__3Qnnj text__373c0__2Kxyz text-color--black-ext
    n1 = [x if (bool(re.search(r'\d+', x)) == False) else "None" for x in n]
    n2 = list(filter(lambda x: x != 'None', n1))
    neigh.append(n2)
    new_tel = [x if (bool(re.search(r'[0-9]{10}|[0-9]{3}-[0-9]{3}-[0-9]{4}|\([0-9]{3}\)[0-9]{3}-[0-9]{4}|\(\d\d\d\)\s\d\d\d-\d\d\
```

```
flat  = list(itertools.chain(*neigh))
data = {}
data = {'Restaurant_Name': name, 'Cuisine': cuisine, 'Restaurant_Ratings': ratings, 'Restaurant_Reviews': reviews,
        'Restaurant_Price': price, 'Restaurant_Address': add, 'Restaurant_Telephone': new_tel, 'Neighbourhood': flat[0:248]}
rest = pd.DataFrame(data)
header = ["Restaurant_Name", "Cuisine", "Restaurant_Ratings", "Restaurant_Reviews", "Restaurant_Price", "Restaurant_Address", "Re
rest.to_csv("ProjectFinal.csv", columns = header)
```

The output in a csv file(first 20 rows):



| | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| 1 | Restaurant_Name | Cuisine | Restaurant_Ratings | Restaurant_Reviews | Restaurant_Price | Restaurant_Address | Restaurant_Telephone | Neighbourhood |
| 2 | The Press Room | Coffee & Tea | none | none | none | 1778 Columbia Street | (604) 566-9666 | Mount Pleasant |
| 3 | The Flying Pig - Yaletown | Canadian (New) | 4 star rating | 1095 | $$ | 1168 Hamilton Street | (604) 568-1344 | Yaletown |
| 4 | Saku | Japanese | 4.5 star rating | 116 | $$ | 548 W Broadway | None | Fairview Slopes |
| 5 | Tuc Craft Kitchen | Canadian (New) | 4 star rating | 950 | $$ | 60 W Cordova Street | (604) 559-8999 | Gastown |
| 6 | Fable | Canadian (New) | 4.5 star rating | 676 | $$ | 1944 W 4th Avenue | (604) 732-1322 | Kitsilano |
| 7 | Lunch Lady | Vietnamese | 4.5 star rating | 10 | none | 1046 Commercial Drive | (604) 559-5938 | Grandview-Woodlands |
| 8 | The Flying Pig - Gastown | Canadian (New) | 4 star rating | 932 | $$ | 102 Water Street | (604) 559-7968 | Gastown |
| 9 | Dinesty Dumpling House | Chinese | 4 star rating | 777 | $$ | 1719 Robson Street | (604) 669-7769 | West End |
| 10 | Kosoo | Korean | 4.5 star rating | 6 | none | 1128 Robson Street | (604) 428-8414 | West End |
| 11 | Phnom Penh | Cambodian | 4 star rating | 1303 | $$ | 244 E Georgia Street | (604) 682-5777 | Strathcona |
| 12 | ChiMec Fried Chicken & Burger | Chicken Shop | 4.5 star rating | 15 | none | 835 Denman St | (604) 331-0058 | West End |
| 13 | Guu with Garlic | Japanese | 4.5 star rating | 930 | $$ | 1698 Robson Street | (604) 685-8678 | West End |
| 14 | Downlow Chicken Shack | Chicken Shop | 4 star rating | 201 | $$ | 905 Commercial Drive | (604) 283-1385 | Grandview-Woodlands |
| 15 | Jingle Bao | Asian Fusion | 4 star rating | 31 | none | 774 Denman Street | (604) 428-7722 | West End |
| 16 | Blue Water Cafe | Seafood | 4.5 star rating | 902 | $$$ | 1095 Hamilton St | (604) 688-8078 | Yaletown |
| 17 | Kokoro Tokyo Mazesoba | Japanese | 4 star rating | 326 | $$ | 551 Seymour Street | (604) 559-8872 | Downtown |
| 18 | Elisa | Steakhouses | 4.5 star rating | 102 | $$$ | 1109 Hamilton Street | (604) 362-5443 | Yaletown |
| 19 | So Hyang Korean Cuisine | Korean | 4.5 star rating | 254 | $$ | 6345 Fraser Street | (604) 729-0702 | Sunset |
| 20 | I & I Jamaican Restaurant | Caribbean | 4.5 star rating | 10 | none | 2528 Kingsway | (604) 434-0526 | Renfrew-Collingwood |
| 21 | Chambar | Belgian | 4 star rating | 1349 | $$$ | 568 Beatty Street | (604) 879-7119 | Downtown |

To input the file into PyMongo, all we had to do was save the info in a dataframe, and instead of passing the info into a CSV file, we pushed it into PyMongo:

```
busname = []
cuisine = []
ratings = []
reviews = []
price = []
add = []
tel = []
neigh = []

for i in range(0, 211, 30):
    soup = make_soup('https://www.yelp.ca/search?find_desc=Restaurants&find_loc=Vancouver%2C%20BC&ns=1&start=' + str(i))
    mains = soup.find_all("div", {"class": "lemon--div__373c0__1mboc mainAttributes__373c0__1r0QA arrange-unit__373c0__o3tjT arra
    second = soup.find_all("div", {"class":"lemon--div__373c0__1mboc secondaryAttributes__373c0__7bA0w arrange-unit__373c0__o3tj
    for main in mains:
        try:
            busname.append(main.find('a').text)
        except:
            busname.append('none')
        try:
            cuisine.append(main.find('a', {'class': 'lemon--a__373c0__IEZFH link__373c0__1G70M link-color--inherit__373c0__3dzpk
        except:
            print('none')
        try:
            ratings.append(main.find('span', {'class': 'lemon--span__373c0__3997G display--inline__373c0__3JqBP border-color--de
        except:
            ratings.append('none')
        try:
            reviews.append(main.find("span", {"class": "lemon--span__373c0__3997G text__373c0__2Kxyz reviewCount__373c0__2r4xT te
        except:
            reviews.append('none')
        try:
            price.append(main.find("span", {"class": "lemon--span__373c0__3997G text__373c0__2Kxyz priceRange__373c0__2DY87 text-
        except:
            price.append('none')

    for sec in second:
        try:
            add.append(sec.address.find("p", {"class": "lemon--p__373c0__3Qnnj text__373c0__2Kxyz text-color--black-extra-light__
        except:
            add.append("None")
        try:
            tel.append(sec.div.div.text)
        except:
            tel.append("None")
    n = [item.get_text() for item in soup.find_all('p', class_ = 'lemon--p__373c0__3Qnnj text__373c0__2Kxyz text-color--black-ext
    n1 = [x if (bool(re.search(r'\d+', x)) == False) else "None" for x in n]
    n2 = list(filter(lambda x: x != 'None', n1))
    neigh.append(n2)
    new_tel = [x if (bool(re.search(r'[0-9]{10}|[0-9]{3}-[0-9]{3}-[0-9]{4}|\([0-9]{3}\)[0-9]{3}-[0-9]{4}|\(\d\d\d\)\s\d\d\d-\d\d\
```

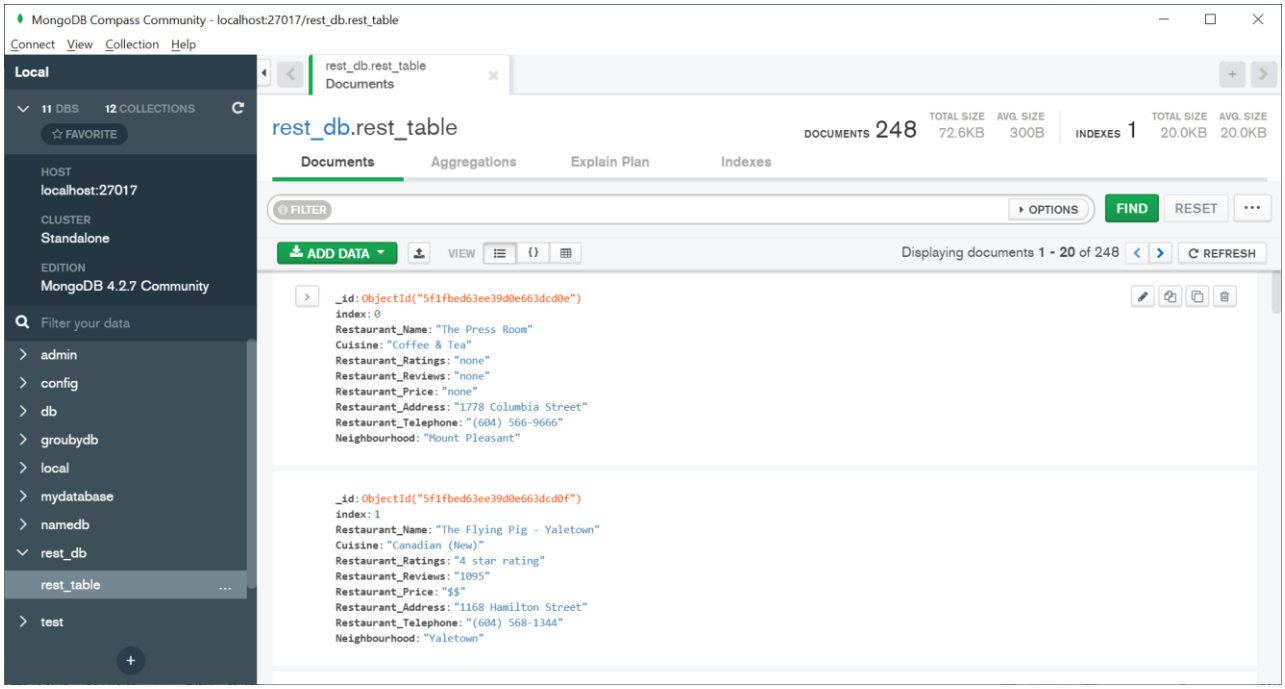
```
flat  = list(itertools.chain(*neigh))
data = {}
data = {'Rest_name': busname, 'Cuisine': cuisine, 'Rest_ratings': ratings, 'Rest_noreviews': reviews, 'Rest_price': price, 'Rest_
rest = pd.DataFrame(data)
header = ["Rest_name", "Cuisine", "Rest_ratings", "Rest_noreviews", "Rest_price", "Rest_add", "Rest_tel", "Neighbourhood"]
rest
```

```
client = MongoClient('localhost',27017)
yelp = client.rest_db.rest_table
```

```
rest.reset_index(inplace=True)
rest_dict = rest.to_dict("records")
```

```
yelp.insert_many(rest_dict)
```

```
<pymongo.results.InsertManyResult at 0x21563f25208>
```




The csv file shows that data type of all columns are object, except the unnamed column (Row ID).

```
df=pd.read_csv('ProjectFinal.csv')
```

```
df.dtypes
```

```
Unnamed: 0              int64
Restaurant_Name        object
Cuisine                object
Restaurant_Ratings     object
Restaurant_Reviews     object
Restaurant_Price       object
Restaurant_Address     object
Restaurant_Telephone   object
Neighbourhood          object
dtype: object
```

In the “Restaurant_Reviews” column, there is a “none” value inside. We replaced it with 0.

```
df['Restaurant_Reviews'].unique()
```

```
array(['none', '1095', '116', '950', '676', '10', '932', '777', '6',
       '1303', '15', '930', '201', '31', '902', '326', '102', '254',
       '1349', '11', '70', '336', '836', '28', '190', '1785', '12', '9',
       '4', '205', '184', '562', '14', '60', '371', '283', '1020', '432',
       '52', '170', '428', '3', '22', '37', '35', '7', '505', '460',
       '325', '13', '97', '155', '364', '96', '986', '18', '445', '537',
       '730', '134', '2', '214', '26', '449', '724', '421', '20', '344',
       '2282', '632', '8', '33', '146', '455', '1', '82', '16', '93',
       '128', '147', '355', '297', '275', '62', '17', '75', '380', '569',
       '681', '21', '194', '19', '249', '40', '1029', '282', '90', '738',
       '211', '27', '609', '53', '113', '104', '392', '46', '693', '29',
       '1067', '47', '678', '25', '167', '185', '61', '99', '271', '196',
       '209', '182', '187', '294', '181', '248', '488', '42', '140',
       '138', '5', '324', '272', '423', '84', '346', '491', '197', '171',
       '131', '269', '387', '71', '143', '30', '334', '473', '163', '947',
       '132', '118', '420', '166', '125', '123', '148', '273', '329',
       '39', '331', '614', '542', '286', '112', '229', '154', '50', '88',
       '195', '444', '172', '49', '54', '180', '51', '44', '45', '103',
       '578'], dtype=object)
```

```
df['Restaurant_Reviews']=df['Restaurant_Reviews'].replace('none',0)
```

```
df['Restaurant_Reviews'].unique()
```

```
array([0, '1095', '116', '950', '676', '10', '932', '777', '6', '1303',
       '15', '930', '201', '31', '902', '326', '102', '254', '1349', '11',
       '70', '336', '836', '28', '190', '1785', '12', '9', '4', '205',
       '184', '562', '14', '60', '371', '283', '1020', '432', '52', '170',
       '428', '3', '22', '37', '35', '7', '505', '460', '325', '13', '97',
       '155', '364', '96', '986', '18', '445', '537', '730', '134', '2',
       '214', '26', '449', '724', '421', '20', '344', '2282', '632', '8',
       '33', '146', '455', '1', '82', '16', '93', '128', '147', '355',
       '297', '275', '62', '17', '75', '380', '569', '681', '21', '194',
       '19', '249', '40', '1029', '282', '90', '738', '211', '27', '609',
       '53', '113', '104', '392', '46', '693', '29', '1067', '47', '678',
       '25', '167', '185', '61', '99', '271', '196', '209', '182', '187',
       '294', '181', '248', '488', '42', '140', '138', '5', '324', '272',
       '423', '84', '346', '491', '197', '171', '131', '269', '387', '71',
       '143', '30', '334', '473', '163', '947', '132', '118', '420',
       '166', '125', '123', '148', '273', '329', '39', '331', '614',
       '542', '286', '112', '229', '154', '50', '88', '195', '444', '172',
       '49', '54', '180', '51', '44', '45', '103', '578'], dtype=object)
```

In the “Restaurant_Reviews” column, there is a “none” value inside. We changed its data type from object to int32.

```
df['Restaurant_Reviews']=df['Restaurant_Reviews'].astype('int')
```

```
df.dtypes
```

```
Unnamed: 0              int64
Restaurant_Name        object
Cuisine                object
Restaurant_Ratings     object
Restaurant_Reviews      int32
Restaurant_Price       object
Restaurant_Address     object
Restaurant_Telephone   object
Neighbourhood          object
dtype: object
```

To find out the top 5 popular restaurants based on the restaurant reviews, we summed up the number of restaurant reviews for each restaurant name and sort out the top 5 restaurant in descending order. The horizontal bar graph shows all the names of the top 5 popular restaurants. Medina Cafe is the most popular restaurant in Vancouver.

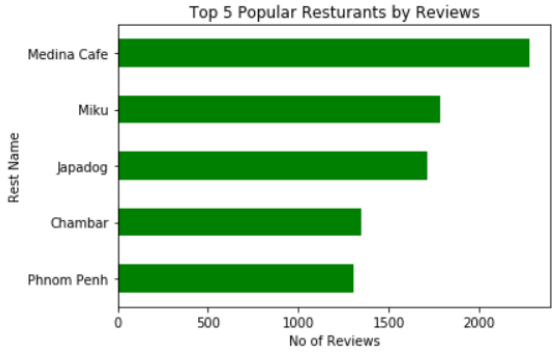
```
re=df.groupby('Restaurant_Name').sum()['Restaurant_Reviews'].sort_values(ascending=False).head(5)
re
```

```
Restaurant_Name
Medina Cafe    2282
Miku           1785
Japadog        1716
Chambar        1349
Phnom Penh     1303
Name: Restaurant_Reviews, dtype: int32
```

```
re.sort_values(inplace=True)
```

```
re.plot(kind='barh',color='green')
plt.xlabel("No of Reviews")
plt.ylabel("Rest Name")
plt.title("Top 5 Popular Resturants by Reviews")
```

```
Text(0.5, 1.0, 'Top 5 Popular Resturants by Reviews')
```




To find out the top 5 neighbourhoods by number of restaurant, we counted the number of restaurant based on each neighbourhood value and sorted out the top 5 neighbourhoods in descending order. The horizontal bar graph shows all the top 5 neighbourhoods. West End is the most common neighbourhood.

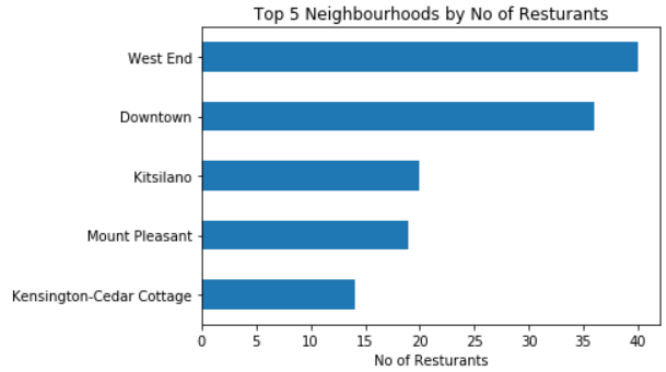
```
Top5Neigh=df['Neighbourhood'].value_counts().head(5)
Top5Neigh
```

```
West End                   40
Downtown                   36
Kitsilano                  20
Mount Pleasant             19
Kensington-Cedar Cottage   14
Name: Neighbourhood, dtype: int64
```

```
Top5Neigh.sort_values(inplace=True)
```

```
Top5Neigh.plot(kind='barh')
plt.xlabel("No of Resturants")
plt.title("Top 5 Neighbourhoods by No of Resturants")
```

```
Text(0.5, 1.0, 'Top 5 Neighbourhoods by No of Resturants')
```



To find out the top 5 cuisines in Vancouver, we counted the number of restaurant based on each cuisine value and sorted out the top 5 cuisines in descending order. The horizontal bar graph shows all the top 5 cuisines in Vancouver. Korean cuisine is the most common one in Vancouver.

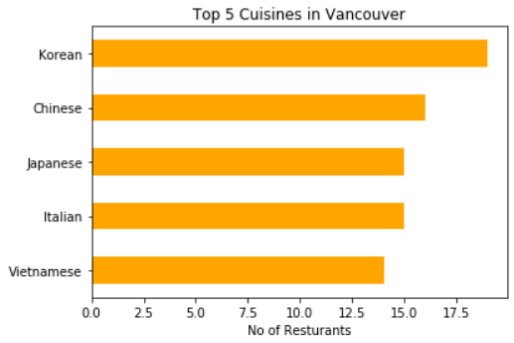
```
top5cuisine=df['Cuisine'].value_counts().head(5)
top5cuisine
```

```
Korean        19
Chinese       16
Italian       15
Japanese      15
Vietnamese    14
Name: Cuisine, dtype: int64
```

```
top5cuisine.sort_values(inplace=True)
```

```
top5cuisine.plot(kind='barh',color='orange')
plt.xlabel("No of Resturants")
plt.title("Top 5 Cuisines in Vancouver")
```

```
Text(0.5, 1.0, 'Top 5 Cuisines in Vancouver')
```



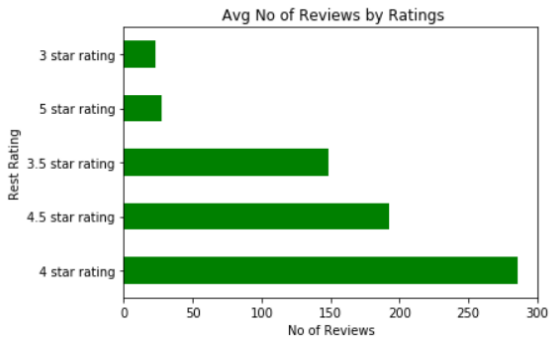

To find out the average number of reviews by ratings, we found the mean of restaurant ratings based on the number of restaurant reviews. Then, we sorted sorted out the top 5 ratings in descending order. The horizontal bar graph shows all the top 5 ratings. 4 star rating is the most common one while 3 star rating and 5 star rating are the least common ones.

```
rat=df.groupby('Restaurant_Ratings').mean()['Restaurant_Reviews'].sort_values(ascending=False).head()
rat
```

```
Restaurant_Ratings
4 star rating      285.977612
4.5 star rating    192.223684
3.5 star rating    148.176471
5 star rating       27.250000
3 star rating       23.333333
Name: Restaurant_Reviews, dtype: float64
```

```
rat.plot(kind='barh',color='green')
plt.xlabel("No of Reviews")
plt.ylabel("Rest Rating")
plt.title(" Avg No of Reviews by Ratings")
```

```
Text(0.5, 1.0, ' Avg No of Reviews by Ratings')
```

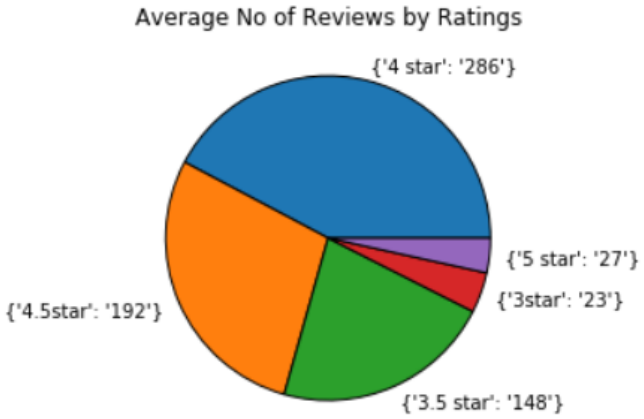


We also used a pie chart to show the same findings with number of reviews as labeling. Again, 4 star rating is the most common one while 3 star rating and 5 star rating are the least common ones.

```
labels=[{'4 star':'286'},{'4.5star':'192'},{'3.5 star':'148'},{'3star':'23'},{'5 star':'27'}]
```

```
plt.pie(rat, labels=labels ,wedgeprops={'edgecolor':'black'})
plt.title("Average No of Reviews by Ratings")
```

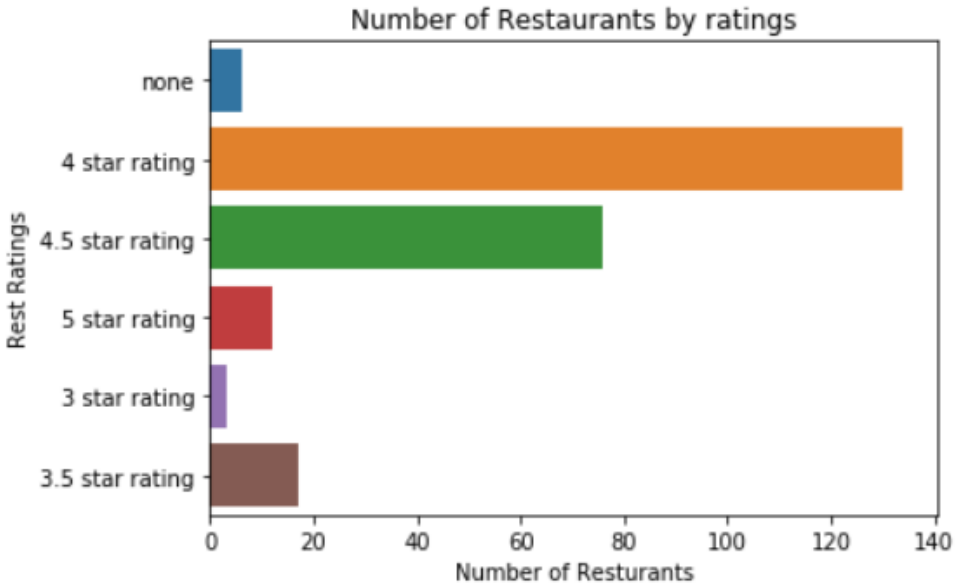
```
Text(0.5, 1.0, 'Average No of Reviews by Ratings')
```




To find the number of restaurants by ratings, we used a countplot from seaborn library. The horizontal bar graph shows that most of the restaurants have 4 star rating. Some restaurants do not have any rating because they are newly opened or restaurant information is not updated.

```
sns.countplot(y='Restaurant_Ratings',data=df,orient="v")
plt.title("Number of Restaurants by ratings")
plt.xlabel("Number of Resturants")
plt.ylabel("Rest Ratings")
```

```
Text(0, 0.5, 'Rest Ratings')
```



To count the number of restaurant by number of outlets, we counted all restaurant names first. Then, we used a countplot from seaborn library to show them all based on their number of outlets. The horizontal bar graph shows that most of the restaurants have 1 outlet only. It is uncommon to have 2+ outlets.

```
rest=df.groupby('Restaurant_Name').count()
rest
```

| Restaurant_Name | Unnamed: 0 | Cuisine | Restaurant_Ratings | Restaurant_Reviews | Restaurant_Price | Restaurant_Address | Restaurant_Telephone | Neighbourhood |
|---|---|---|---|---|---|---|---|---|
| Absinthe Bistro | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Afghan Horsemen Restaurant | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Ajs Brooklyn Pizza Joint | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Aleph Eatery | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Alibi Room | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| Z&W Shanghai | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Zaatar W Zeit | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ZamZam | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Zefferelli's | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Zocalo Modern Cantina | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

235 rows × 8 columns

```
sns.countplot(y='Restaurant_Ratings',data=rest)
plt.xlabel("Count of Restaurants")
plt.ylabel("No of outlets")
plt.title("Count of Restaurants by Number of outlets")
```

```
Text(0.5, 1.0, 'Count of Restaurants by Number of outlets')
```