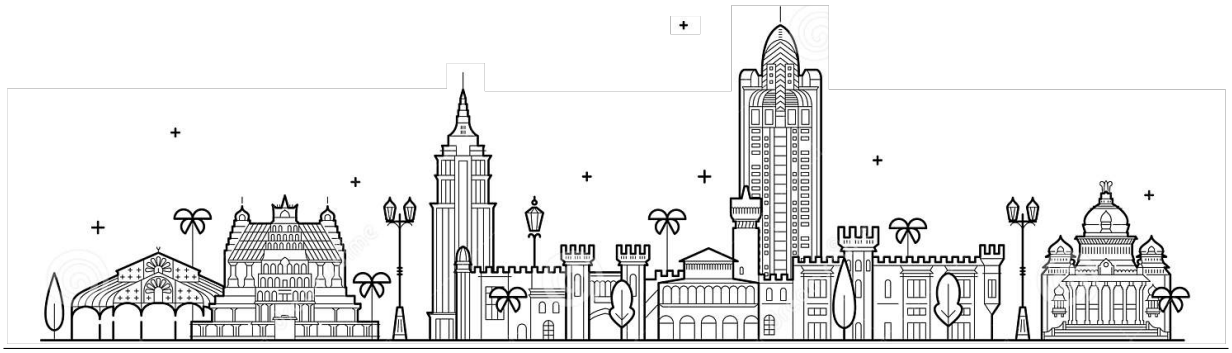


The Battle of Neighborhoods of Bangalore Central

A Case-Study for Commercial Viability of Mobile Food-Truck in the Neighborhood of Bangalore City

IBM Professional Certificate in Data Science by Coursera

Capstone Project Report
submitted towards the requirement of
IBM Coursera Applied Data Science by

Kumar Hemant

Dated : February 20, 2020

This report is available at: https://github.com/khemanta/Coursera_Capstone/

//Table of Contents

1 Introduction **3**
- 1.1 Problem Statement . . . 3
- 1.2 Data Sources . . . 4
  - 1.2.1 Data from Foursquare APIs . . . 4
  - 1.2.2 Data from Zomato APIs . . . 4

2 Data Collection and APIs **5**
- 2.1 GeoPy and Other Python Package . . . 5
- 2.2 Foursquare API . . . 5
- 2.3 Zomato API . . . 6

3 Data Cleaning and Transformation **7**
- 3.1 Visualization of the Data . . . 7
- 3.2 After Merging and Cleaning of the Data . . . 7
- 3.3 The Cleaned Data . . . 8

4 Methodology: Approach and Solution **9**
- 4.1 Re-Visit Objective to Carve Out Approach . . . 9
- 4.2 Data Aggregation . . . 9
- 4.3 Data Exploration . . . 9
- 4.4 Data Analysis . . . 9

5 Data Exploration and Analysis **10**
- 5.1 Analysis of Categories of Foods . . . 10
- 5.2 Analysis of Ratings of Restaurants . . . 10
- 5.3 Analysis of Average Price of Food . . . 11

6 Results and Discussions **12**
- 6.1 Results . . . 12
- 6.2 Discussions . . . 13
- 6.3 Clustering of Neighborhoods . . . 13
  - 6.3.1 Cluster of Venues/Restaurants with Color Codes . . . 14
  - 6.3.2 Cluster of Venues/Restaurants with Color Codes . . . 14

7 Conclusions and Recommendations **15**
- 7.1 Conclusion . . . 15
- 7.2 Commercial Viability of Mobile Food Truck in The Neighborhood . . . 15
- 7.3 Recommendation for Mobile Food Truck . . . 15
- 7.4 Final Recommendations . . . 15

8 Appendix **16**
- 8.1 References . . . 16
- 8.2 The Complete Source Code . . . 16

1 Introduction

The present exercise aim to study the neighborhood of Bangalore Central District. Bangalore is thriving metropolis, an IT and technology hub and usually referred as the Silicon Valley of India.

In this case study, we study to address if there is a specific neighborhood which has food preferences over other? Are there neighborhood which are price sensitive? Are there any neighborhood which enjoy better quality restaurants than other? And finally some recommendation for a entrepreneurial venture to cater the need of mobile food truck and which segment and region and customer segment would be who could be targeted from commercial viability. The study is divided largely into seven parts which is illustrated below as per table of contents.

1.1 Problem Statement

Bangalore is a big metropolis. Till few decades back it was a beautiful city of people wishing to spend the time of after retirement from work. Old Bangalore which is city center has remained largely same with lot of eateries and restaurants, ice-cream parlors, cafe and pubs. But now a thriving metropolis is an IT hub.

A. The immediate objective of this exercise is to analyze and data analysis to:
    1. Identify the neighborhood of city center of Bangalore.
    2. Ratings of the restaurants in these neighborhoods.
    3. Price sensitivity neighborhoods i.e., average cost of diner across the neighborhoods.
    4. Dominant categories of food and restaurants in across neighborhoods.

B. Recommendation for opening a new restaurant :
    1. Whether a mobile food truck would be good option?
    2. What area of the neighborhoods the food truck should target?
    3. What category of foods, which price range in which neighborhoods food truck should target?

1.2 Data Sources

For this exercise to know the neighborhoods we need geographic data. As well for food and restaurant related information. TO cater this need we rely on the the followings APIs

- Foursquare APIs for locations and neighborhoods places of interest as restaurant.
- Zomato APIs for the identified locations foods and restaurants.
- GeoPy Package for retrieving the geo-code (longitude, latitude) of the city center and neighborhoods.

1.2.1 Data from Foursquare APIs

The following data sources and attributes will be extracted from Foursquare APIs.

https://developer.foursquare.com/

```
foursquare_data_attributes = ['venue.name', 'venue.categories',
  'venue.location.lat', 'venue.location.lng']
```

- Venue name in the Neighborhood
- Group or Category of the Venue
- Geo-Code of the Venue as (lat, lng)

1.2.2 Data from Zomato APIs

The following data sources and attributes will be extracted from Zomato APIs.

https://developers.zomato.com/

```
zomato_data_attributes = ['venue', 'latitude', 'longitude',
  'price_for_two', 'price_range', 'rating', 'address']
```

- Venue of the Restaurant from that Neighborhood
- Geo-Code: (Latitude, Longitude)
- Price of Foods for Dinning of Two People
- Price Range of Foods from the Restaurant
- Ratings of the Restaurant
- Address of the Restaurant

2 Data Collection and APIs

We begin by fetching a total of all venues in Bangalore upto a range of 10 Kilometers using the Foursquare API. The Foursquare API has the explore API which allows us to find venue recommendations within a given radius from the given coordinates. We will use this API to find all the venues we need.

2.1 GeoPy and Other Python Package

In order to get the geo-code of Bangalore, the city center, we could have used just ”Google” to search get the coordinates. But wanted to make a generic approach for searching and retrieving any city or location from a geo-graphical information system like OpenStreetMap.

PyPI is used to install and load the geopy package.

```
# Conditional installation of Python libraries and packages
!pip install beautifulsoup4
!pip install lxml
!pip install requests
!pip install html5lib
!pip install geopy
!pip install folium
!pip install pandas
!pip install numpy
!pip install scipy
!pip install scikit-learn
!pip install -r requirement.txt
```

```
from geopy.geocoders import Nominatim # convert an address into latitude and longitude values
address = 'Bangalore, India', # 'Whitefield, Bangalore, India'
geolocator = Nominatim(user_agent="application") #geolocator = Nominatim()
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinates Bangalore City are {}, {}.'.format(latitude, longitude))
```

```
The geograpical coordinates Bangalore City are 12.9791198, 77.5912997.
```

2.2 Foursquare API

We have used the [name, lat, ng] values of various venues fetched from Foursquare API to use the search API and get more information regarding each venue.

The developer account has to be signed here. https://developer.foursquare.com/

```
FOURSQUARE_CLIENT_ID = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
FOURSQUARE_CLIENT_SECRET = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
RADIUS = 15000 # 15 Km
NO_OF_VENUES = 100
VERSION = '20200220' # Current date
```

2.3 Zomato API

The Zomato API allows using its search API to search for any given venue based on certain search filters such as query, latitude, longitude and more. Zomato also requires a Zomato user key which can be accessed with a developer account.

The developer account has to be signed here. https://developers.zomato.com/ and a key has to be generated.

```
headers = {'user-key': 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'}
venues_information = []

for index, row in foursquare_venues.iterrows():
    print("Fetching data for venue: {}".format(index + 1));
    venue = []
    url = ('https://developers.zomato.com/api/v2.1/search?q={}' +
           '&start=0&count=1&lat={}&lon={}&sort=real_distance').format(row['name'], row['lat'], row
['lng'])
    result = requests.get(url, headers = headers).json()
    if (len(result['restaurants']) > 0):
        venue.append(result['restaurants'][0]['restaurant']['name'])
        venue.append(result['restaurants'][0]['restaurant']['location']['latitude'])
        venue.append(result['restaurants'][0]['restaurant']['location']['longitude'])
        venue.append(result['restaurants'][0]['restaurant']['average_cost_for_two'])
        venue.append(result['restaurants'][0]['restaurant']['price_range'])
        venue.append(result['restaurants'][0]['restaurant']['user_rating']['aggregate_rating'])
        venue.append(result['restaurants'][0]['restaurant']['location']['address'])
        venues_information.append(venue)
    else:
        venues_information.append(np.zeros(6))

zomato_venues = pd.DataFrame(venues_information,
                             columns = ['venue', 'latitude',
                                        'longitude', 'price_for_two',
                                        'price_range', 'rating', 'address'])
```

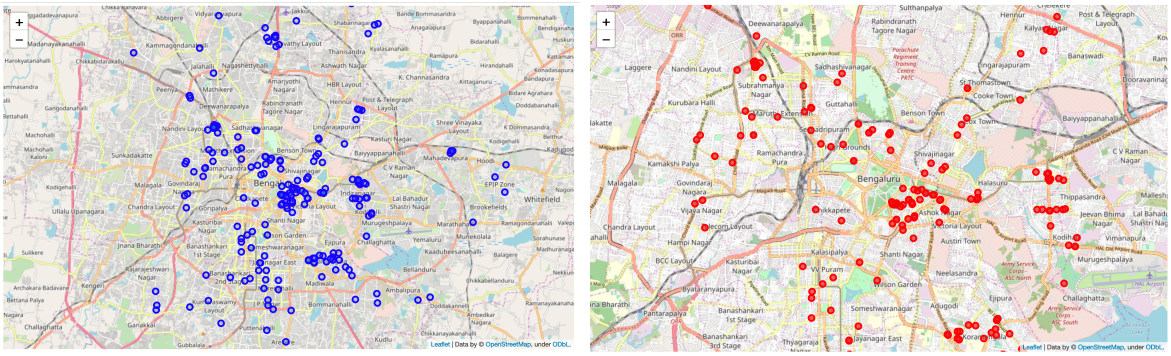
- The query will be the name of the venue.
- The start defines from what offset we want to start, so we'll keep it at 0.
- The count defines the number of restaurants we want to fetch. As we have the exact location coordinates, we'll fetch only one.
- We will supply the latitude and longitude values.
- We will set the sorting criteria as real_distance so each time we get the venue we're searching based on location coordinates.

3 Data Cleaning and Transformation

The data from multiple resources might not always align. Thus, it is important to combine the data retrieved from multiple resources properly.

We'll first plot the two data points on the map. We'll then try to combine data points that have their latitude and longitude values very close to one another. From the remaining selected venues, we will inspect the venues to ensure that any remaining mismatched venues are also removed from the final dataset of venues before we begin any analysis.

3.1 Visualization of the Data



Foursquare and Zomato on Map

3.2 After Merging and Cleaning of the Data

1. After merging the data there are many information which are redundant like: geo-code(lat, lng) is same as geo-code(latitude, longitude). We will deleted them.

2. We will compute a distance measure of neighborhood with geo-code and if the difference in latitude and longitude will be less than 0.0004, we will drop them as the locations would be very close.

```
foursquare_venues['lat'] = foursquare_venues['lat'].apply(lambda lat: round(float(lat), 4))
foursquare_venues['lng'] = foursquare_venues['lng'].apply(lambda lng: round(float(lng), 4))
zomato_venues['latitude'] = zomato_venues['latitude'].apply(lambda lat: round(float(lat), 4))
zomato_venues['longitude'] = zomato_venues['longitude'].apply(lambda lng: round(float(lng), 4))
```

```
dataset = pd.concat([foursquare_venues, zomato_venues], axis = 1)
dataset['lat_diff'] = dataset['latitude'] - dataset['lat']
dataset['lng_diff'] = dataset['longitude'] - dataset['lng']
```

```
# Creating a subset of the dataset as selected_venues
selected_venues = dataset[(abs(dataset['lat_diff']) <= 0.0004) & (abs(dataset['lng_diff']) <= 0.00
04)].reset_index(drop = True)
selected_venues.head()
```

This is to capture more of the data which are not at same place as we are not comparing two restaurants of adjacent places but interested in span of locations and neighborhoods.

3.3 The Cleaned Data

	categories	venue	latitude	longitude	price_range	rating	address	average_price
0	Hotel	Merak-JW Marriott Hotel	12.9724	77.5951	3.0	3.6	JW Marriott, 24/1, Vittal Mallya Road, Lavelle...	600.0
1	Shopping Mall	Shiro	12.9718	77.5959	4.0	4.3	2nd Floor, UB City Mall, Vittal Mallya Road, L...	1500.0
2	Italian Restaurant	Toscano	12.9719	77.5964	4.0	4.4	2nd Floor, UB City, Vittal Mallya Road, Lavell...	1200.0
3	Tea Room	Infinitea Tea Room & Tea Store	12.9871	77.5948	3.0	4.4	2, Shah Sultan Complex, Cunningham Road, Banga...	650.0
4	American Restaurant	Hard Rock Cafe	12.9760	77.6016	4.0	4.5	40, Opposite LIC Building, Off MG Road, St. Ma...	1250.0
5	Burger Joint	Truffles	12.9718	77.6011	2.0	4.4	22, St. Marks Road, Bangalore	450.0
6	Deli / Bodega	Smoke House Deli	12.9717	77.5983	3.0	4.6	52/ 53, Ground Floor, Lavelle Road, Bangalore	800.0
7	Ice Cream Shop	Corner House Ice Cream	12.9732	77.6000	1.0	4.4	4, Madras Bank Road, Lavelle Road, Bangalore	175.0
8	Sushi Restaurant	Harima	12.9675	77.5999	4.0	4.3	131, 4th Floor, Devatha Plaza, Residency Road,...	1000.0
9	Lounge	Skyye	12.9716	77.5964	4.0	4.2	Uber Level, 16th Floor, UB City, Vittal Mallya...	1250.0

4 Methodology: Approach and Solution

The methodology followed in this project is multi prong approach :

- To study the food pricing across city in within the old Bangalore city limit from center of the town.
- The price sensitivity analysis for cost of same cuisine varies across city because of the real-estate pricing of the food joints and restaurants serving the need.
- With this help, identify an option for Food Truck to cater the need of area where pricing are higher relative to other area.

4.1 Re-Visit Objective to Carve Out Approach

- Identify if there is a gap in pricing as food trucks are relatively cheaper and does not need permanent renting place.
- The area where the food truck could be target.
- Type of food popular with the end user based on Zomato data.

4.2 Data Aggregation

As a first step, we retrieved the data from two APIs (Foursquare and Zomato). We extract venue information from the center of Chandigarh, upto a distance of 4 Km. The latitude and longitude values are then used to fetch venue rating and price from Zomato.

4.3 Data Exploration

Secondly, we then explored the data retrieved from the two APIs on the map and identified the top category types. The data from the two sources is carefully combined based on the name, latitude and longitude values from the two sources. The final dataset would include the rating and price values for each venue.

4.4 Data Analysis

Next, we'll analyze the data that we created based on the ratings and price of each venue. We'll identify places where many venues are located so that any visitor can go to one place and enjoy the option to choose amongst many venue options. We'll also explore areas that are high rated and those that are low rated while also plotting the map of high and low priced venues. Lastly, we'll cluster the venues based on the available information of each venue. This will allow us to clearly identify which venues can be recommended and with what characteristics.

Finally, we'll discuss and conclude which venues to be explored based on visitor requirement of rating and cost.

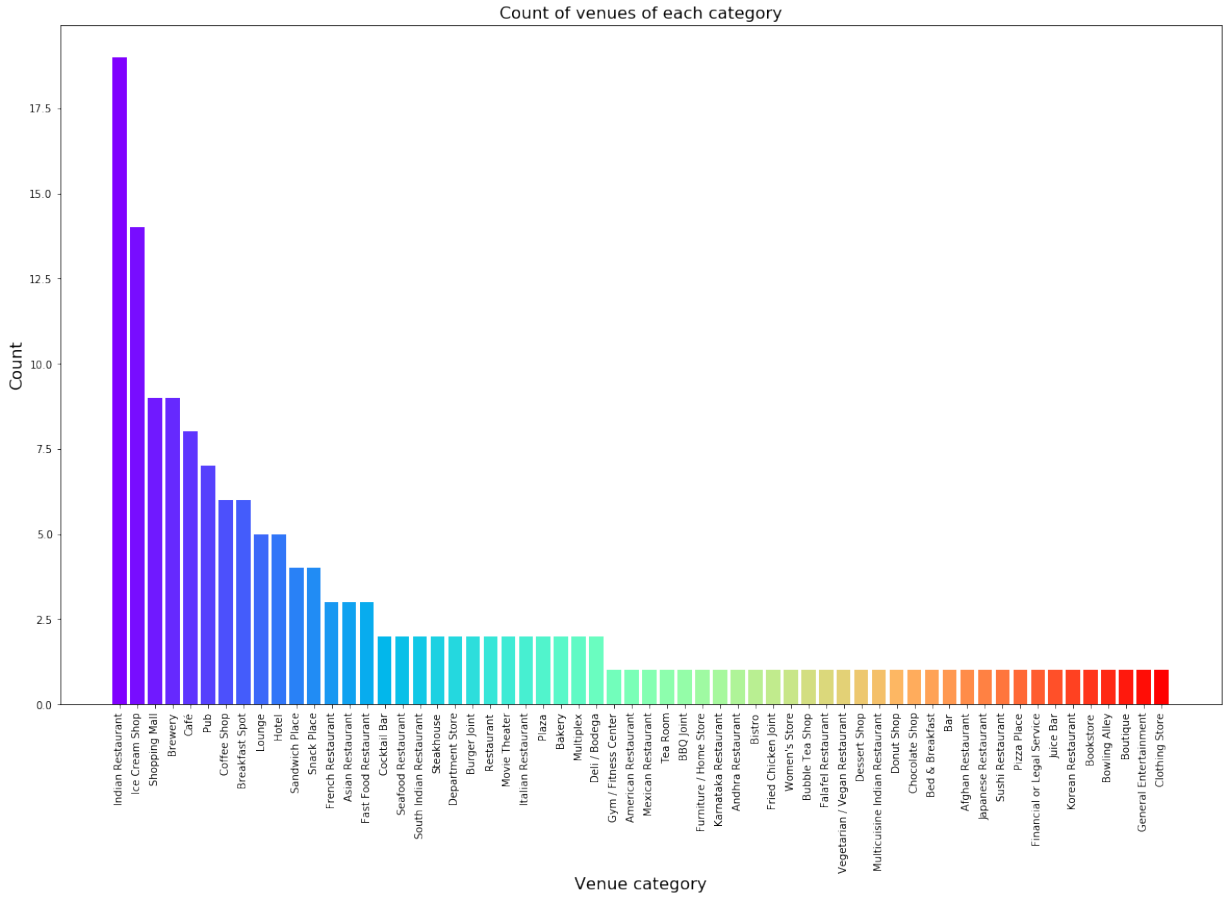
5 Data Exploration and Analysis

The complete dataset is now in its final form. Now we will do some exploratory data analysis (EDA).

We will inspect the neighborhood venues based on their rating. The rating of a venue are based on user reviews and belongs to a range from 1 to 5. We will also analyze the venues based on their price per person as well as the price range.

5.1 Analysis of Categories of Foods

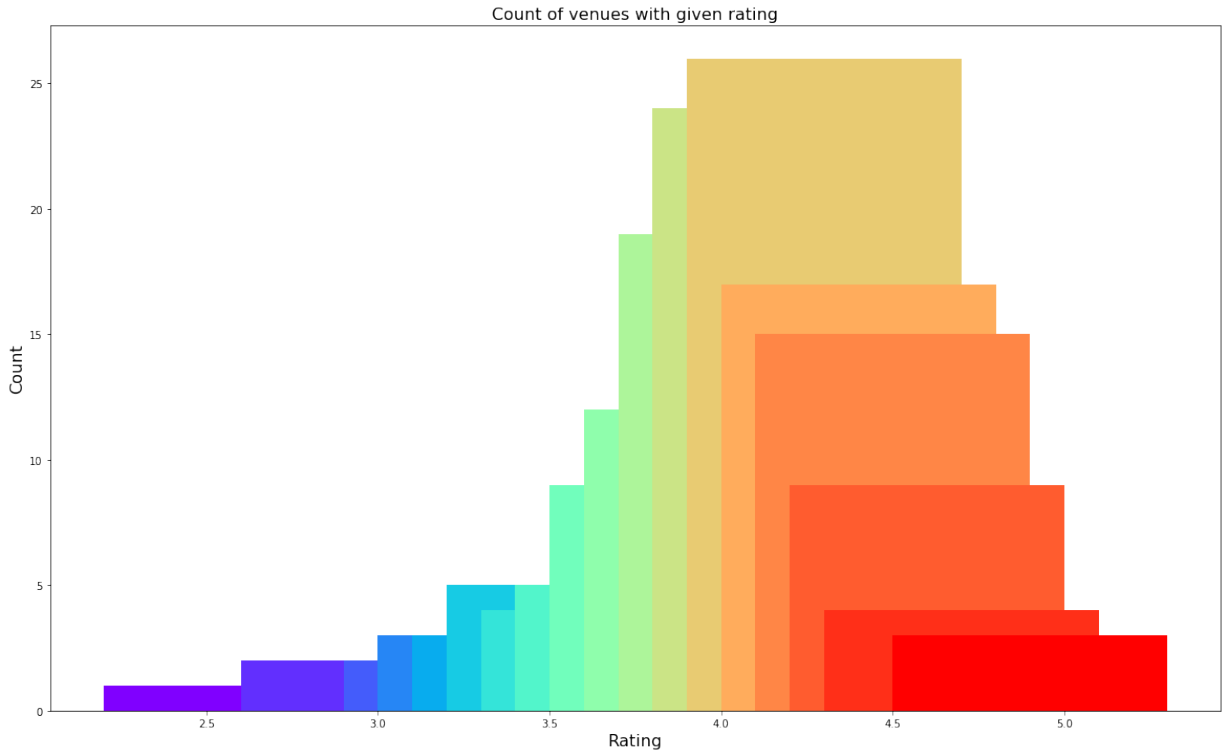
Graphical analysis data visualized for various categories of food.



5.2 Analysis of Ratings of Restaurants

Rating of a venue is an important factor on which a visitor decides whether it is worth it to visit the place. To cater to this, we will first see what is the average rating for all the venues in the city. Next, we will plot the venues on the map and color code them.

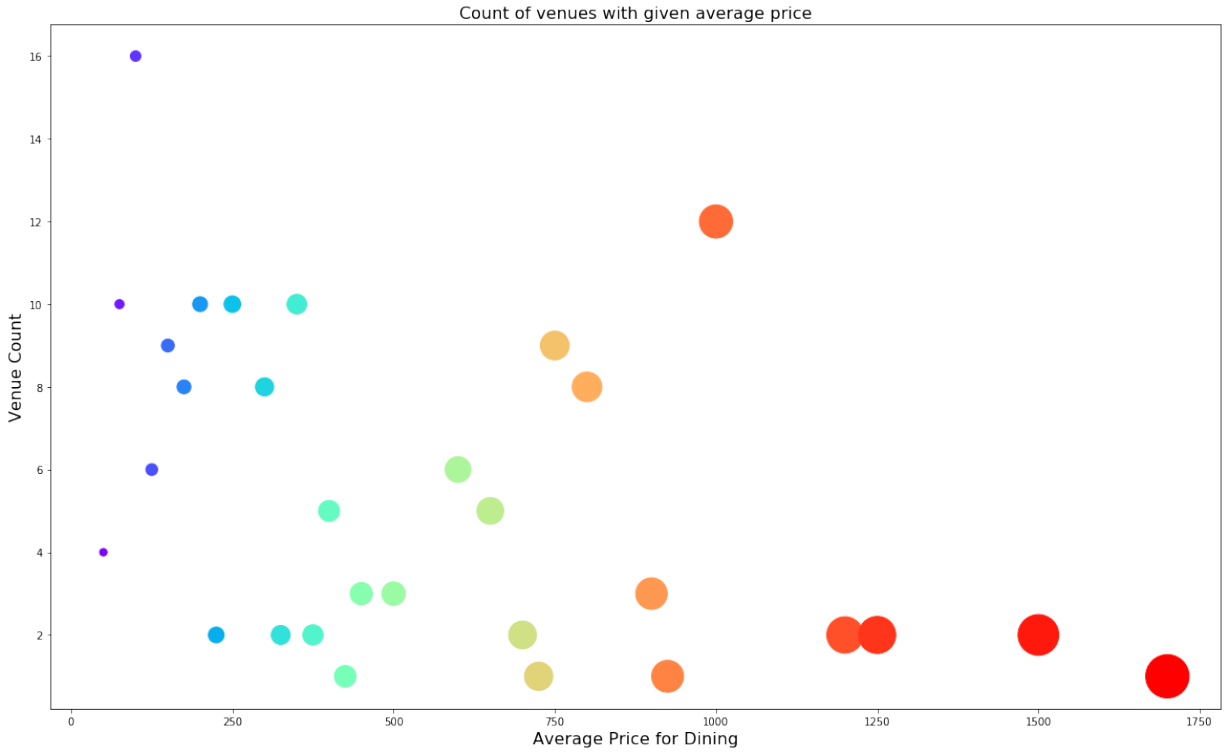
We'll first identify the various rating values and plot them as a bar plot with their counts to see the most common rating.



5.3 Analysis of Average Price of Food

We will now take a look the venues based on the price values. We have two price features for our venues, one is average price which defines the average cost for one person and the other is price range which determines the price range as defined by Zomato.

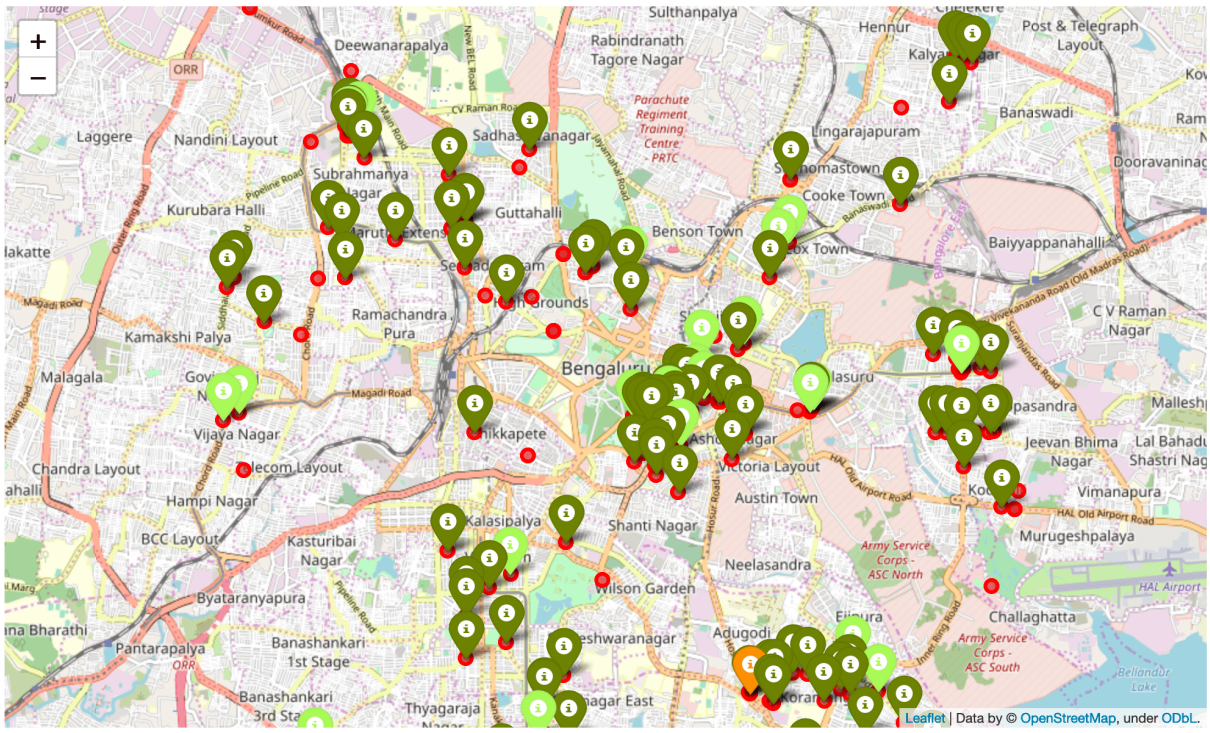
We will first explore the average price using a scatter plot between the price and the count of venues with that average price. We'll size the points based on the price to highlight their price.



6 Results and Discussions

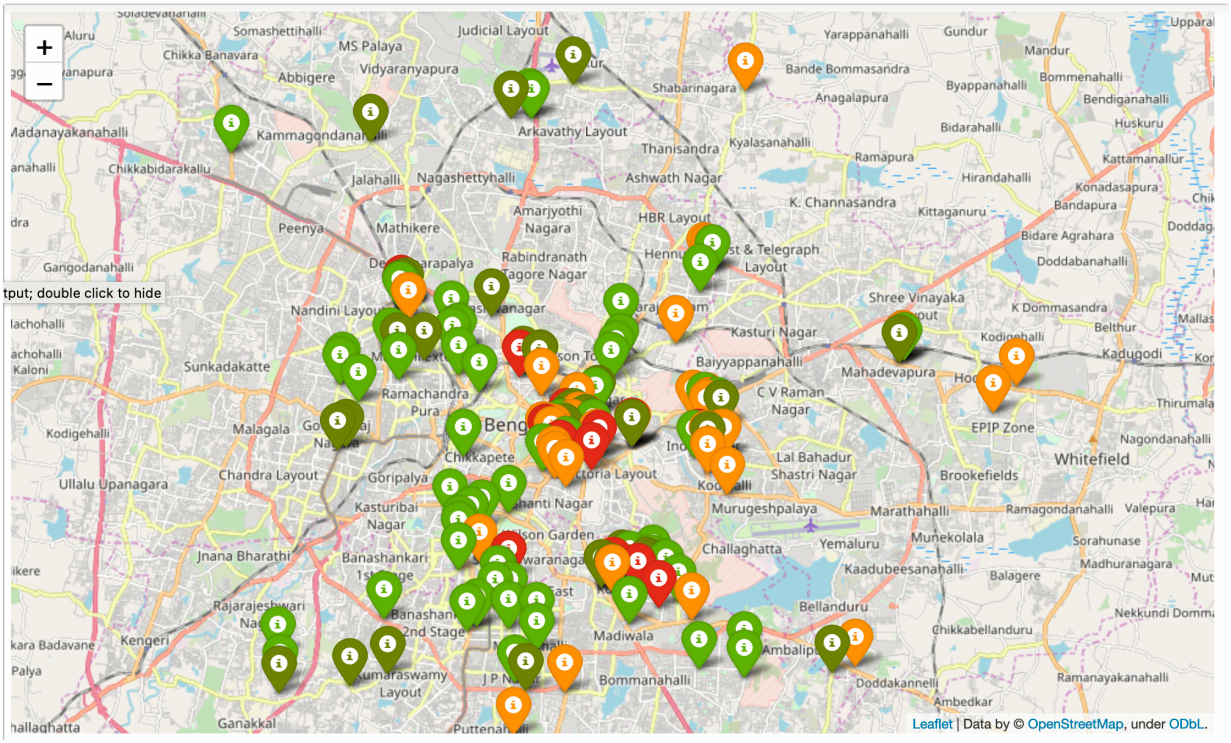
6.1 Results

- After collecting data from the Foursquare and Zomato APIs, we got a list of 239 different venues. But after cleaning we were left with 176 with ratings 1 and above it further came down to 163.
- Observation are shared as an immediate but here would like to highlights that contrary to usual believe that higher price of restaurants in specific area and to some extent it was true but largely all sorts of restaurant on afford-ability or pricing index was across all the geographic clusters.
- While the complete range of ratings range from 1 to 5, the majority venues have ratings close to 4. This means that most restaurants provide good quality food which is liked by the people of the city, thus indicating the high rating.
- CBD, Indira Nagar, and Koramangala are posh area but Koramangala surprisingly has lot of price variations but very good ratings almost 4+ while the Malleswaram are the price is cheapest.
- The Jaya Nagar and JP Nagar being famous with family outings and the neighborhood has many ice cream and cool drink and dessert joints which is evident from data.



6.2 Discussions

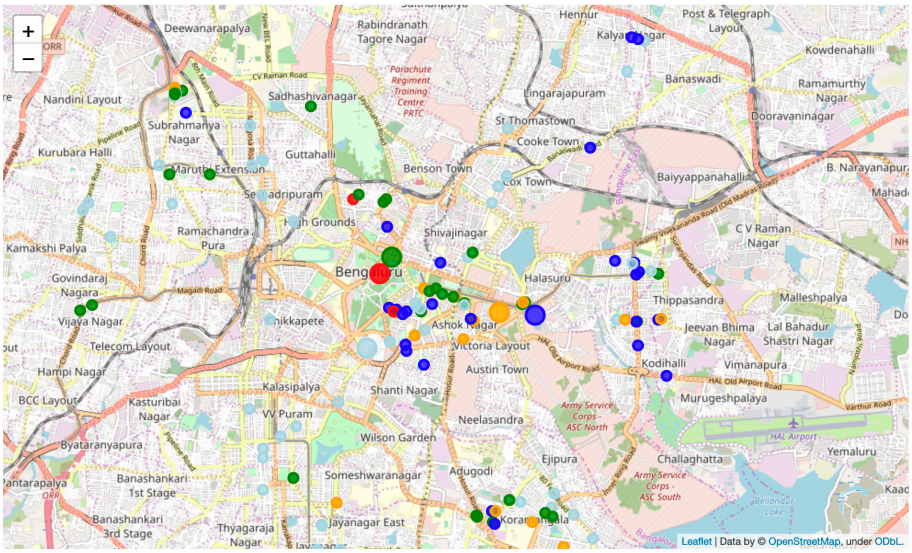
- If one is looking for cheap places with relatively high rating, one should check Malleswaram and Jaya Nagar.
- If one is looking for the best places, with the highest rating but might also carry a high price tag, you should CBD and Indira Nagar is the place to go.
- If you're looking to explore the city and have no specific criteria to decide upon the places you want to visit, you should try Koramangala and Jaya Nagar specially 4th Block and Jaya Nagar East End.



6.3 Clustering of Neighborhoods

The clustering has however has not distinctively across geography but across geography. The clusters are as followings:

6.3.1 Cluster of Venues/Restaurants with Color Codes



6.3.2 Cluster of Venues/Restaurants with Color Codes

```
print("These venues for cluster#1 have mean price range of {:.02f} and rating spread around {:.02
f}".
      format(result['average_price'].mean(), result['rating'].astype(float).mean()))
```

C1

```
These venues for cluster#1 have mean price range of 130.77 and rating spread around 4.15
```

```
print("These venues for cluster#2 have mean price range of {:.02f} and rating spread around {:.02
f}".
      format(result['average_price'].mean(), result['rating'].astype(float).mean()))
```

C2

```
These venues for cluster#2 have mean price range of 713.71 and rating spread around 4.31
```

```
print("These venues for cluster#3 have mean price range of {:.02f} and rating spread around {:.02
f}".
      format(result['average_price'].mean(), result['rating'].astype(float).mean()))
```

C3

```
These venues for cluster#3 have mean price range of 342.61 and rating spread around 3.98
```

```
print("These venues for cluster#4 have mean price range of {:.02f} and rating spread around {:.02
f}".
      format(result['average_price'].mean(), result['rating'].astype(float).mean()))
```

C4

```
These venues for cluster#4 have mean price range of 1026.25 and rating spread around 4.37
```

```
print("These venues for cluster#5 have mean price range of {:.02f} and rating spread around {:.02
f}".
      format(result['average_price'].mean(), result['rating'].astype(float).mean()))
```

C5

```
These venues for cluster#5 have mean price range of 1566.67 and rating spread around 4.27
```

7 Conclusions and Recommendations

Here the conclusively what is the recommendation, would be discussed as take away points for business users of this study.

7.1 Conclusion

The purpose of this project was to explore the places in Central Bangalore and it's close neighborhood with 10 km range but with zomato data range has been much lower within 5km of neighborhood. Bangalore being very big almost 750 square km area is vast with many suburbs.

The charm of old rustic Bangalore and its street and food-joints are worth trying. This resonates with the experience one would have interacting with local or Bangaloreans, who would knows well - Jaya Nagar and Koramangala area are best to try awesome food.

7.2 Commercial Viability of Mobile Food Truck in The Neighborhood

The objective is also from a commercial point of which kind of mobile food-truck could be targeted. As I started this capstone exercise, a couple of friends who want to looking for recommendation what could be offered as food truck/mobile restaurant and in which area?

7.3 Recommendation for Mobile Food Truck

As evident, the Jaya Nagar and Koramangala are quite famous for it.
However to find the gap of temporary food truck I suggest : Indira Nagar, CBD and Koramangala area. The reason being these places are a little costlier and have opportunity for price sensitive customer of these area to offer some options. To bridge these gaps, there could be options for **'Pick-and-Go'** like 'Biryani' kind of food. The food-trucks, though there is no temporal data (time based), **late evening 'Dessert and Ice Creams'** offerings as an options or both.

Though, there was no temporal information in the data, but based on evidence of two categories of food, the second highest was Ice Cream, this does correlate to the real life experience.

7.4 Final Recommendations

1. Indira Nagar for late evening Ice Cream and Dessert Options
2. CBD and Indira Nagar for Day Time Packed Lunch
3. Pick-and-Go Food Option like Biryani in Koramangala, Indira Nagar in the evening time.

Disclaimer : However, more analysis with temporal aspect of the data has to be factored in for more accurate analysis.

8 Appendix

8.1 References

- Foursquare Developer https://developer.foursquare.com/
- Zomato Developer https://developers.zomato.com/
- PyPI BeautifulSoup and other libraries
- IBM Developers and CognitiveClasse.ai https://www.cognoitiveclasses.ai
- Coursera courses in IBM Applied Data Science https://www.coursera.org

8.2 The Complete Source Code

```
# install all the requirements
!pip3 install -r requirement.txt

#!pip3 install geopy #uncomment this if geopy import throws error
from geopy.geocoders import Nominatim # convert an address into latitude and longitude values
address = 'Bangalore, India', # 'Whitefield, Bangalore, India'
geolocator = Nominatim(user_agent="application") #geolocator = Nominatim()
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinates Bangalore City are {}, {}.'.format(latitude, longitude))


# Getting Bangalore Central Geo-code
BLR_LAT = location.latitude
BLR_LON = location.longitude
print('The geograpical coordinates of Bangalore are {}, {}.'.format(BLR_LAT, BLR_LON))

# VISUALIZING THE NEIGHBORHOOD

#!pip install folium
import folium

bangalore_map = folium.Map(location = [BLR_LAT, BLR_LON], zoom_start = 12)
folium.Marker([BLR_LAT, BLR_LON]).add_to(bangalore_map)
#bangalore_map.save("maps/bangalore_map.html")
bangalore_map


# Cartesian coordinates system

#!pip3 install shapely
import shapely.geometry

#!pip install pyproj
import pyproj

import math

def lonlat_to_xy(lon, lat):
    proj_latlon = pyproj.Proj(proj='latlong',datum='WGS84')
    proj_xy = pyproj.Proj(proj="utm", zone=33, datum='WGS84')
    xy = pyproj.transform(proj_latlon, proj_xy, lon, lat)
    return xy[0], xy[1]

def xy_to_lonlat(x, y):
    proj_latlon = pyproj.Proj(proj='latlong',datum='WGS84')
    proj_xy = pyproj.Proj(proj="utm", zone=33, datum='WGS84')
    lonlat = pyproj.transform(proj_xy, proj_latlon, x, y)
    return lonlat[0], lonlat[1]

def calc_xy_distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    return math.sqrt(dx*dx + dy*dy)

bangalore_center=[latitude, longitude]
```

```
print('Coordinate transformation check')
print('-----------------------------------------')
print('Bangalore City Center latitude={}, longitude={}'.format(bangalore_center[0], bangalore_center[1]))
x, y = lonlat_to_xy(bangalore_center[1], bangalore_center[0])
print('Bangalore center UTM X={}, Y={}'.format(x, y))
lo, la = xy_to_lonlat(x, y)
print('Bangalore center longitude={}, latitude={}'.format(lo, la))


# The neighborhood in Cartesian coordinates

# City center in Cartesian coordinates
bangalore_center_x, bangalore_center_y = lonlat_to_xy(bangalore_center[1], bangalore_center[0])

k = math.sqrt(3) / 2 # Vertical offset for hexagonal grid cells
x_min = bangalore_center_x - 6000
x_step = 600
y_min = bangalore_center_y - 6000 - (int(21/k)*k*600 - 12000)/2
y_step = 600 * k

latitudes = []
longitudes = []
distances_from_center = []
xs = []
ys = []
for i in range(0, int(21/k)):
    y = y_min + i * y_step
    x_offset = 300 if i%2==0 else 0
    for j in range(0, 21):
        x = x_min + j * x_step + x_offset
        distance_from_center = calc_xy_distance(bangalore_center_x, bangalore_center_y, x, y)
        if (distance_from_center <= 6001):
            lon, lat = xy_to_lonlat(x, y)
            latitudes.append(lat)
            longitudes.append(lon)
            distances_from_center.append(distance_from_center)
            xs.append(x)
            ys.append(y)

print(len(latitudes), 'candidate neighborhood centers generated.')


# DATA GATHERING USING API
# Using Forsquare API

FOURSQUARE_CLIENT_ID = 'LFIRHYYRWVXXXXXXXXXXXXXXXXXXXXXXXXXVH425'
FOURSQUARE_CLIENT_SECRET = 'X5RDIJSLRXXXXXXXXXXXXXXXXXXXXXHI3N52E2'
RADIUS = 10000 # 10 Km
NO_OF_VENUES = 100
VERSION = '20200220' # Current date


def get_category_type(row):
    try:
        categories_list = row['categories']
    except:
        categories_list = row['venue.categories']

    if len(categories_list) == 0:
        return None
    else:
        return categories_list[0]['name']


# Using Foursqaure API for identified neighborhood coordinates to get the venue info
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.colors as colors

from pandas.io.json import json_normalize
import requests

pd.set_option('display.max_rows', None)

offset = 0
total_venues = 0
foursquare_venues = pd.DataFrame(columns = ['name', 'categories', 'lat', 'lng'])

while (True):
    url = ('https://api.foursquare.com/v2/venues/explore?client_id={}'
           '&client_secret={}&v={}&ll={},{}&radius={}&limit={}&offset={}').format(FOURSQUARE_CLIENT_ID,
                                                                                  FOURSQUARE_CLIENT_SECRET,
                                                                                  VERSION,
                                                                                  BLR_LAT,
                                                                                  BLR_LON,
                                                                                  RADIUS,
                                                                                  NO_OF_VENUES,
                                                                                  offset)
    result = requests.get(url).json()
    venues_fetched = len(result['response']['groups'][0]['items'])
    total_venues = total_venues + venues_fetched
    print("Total {} venues fetched within a total radius of {} Km".format(venues_fetched, RADIUS/1000))

    venues = result['response']['groups'][0]['items']
    venues = json_normalize(venues)
```

```
    # Filter the columns
    filtered_columns = ['venue.name', 'venue.categories', 'venue.location.lat', 'venue.location.lng']
    venues = venues.loc[:, filtered_columns]

    # Filter the category for each row
    venues['venue.categories'] = venues.apply(get_category_type, axis = 1)

    # Clean all column names
    venues.columns = [col.split(".")[-1] for col in venues.columns]
    foursquare_venues = pd.concat([foursquare_venues, venues], axis = 0, sort = False)

    if (venues_fetched < 100):
        break
    else:
        offset = offset + 100

foursquare_venues = foursquare_venues.reset_index(drop = True)
print("\nTotal {} venues fetched".format(total_venues))

# Using Zomato APIs

headers = {'user-key': '65xxxxxxxxxxxxxxx7b'}
venues_information = []

for index, row in foursquare_venues.iterrows():
    print("Fetching data for venue: {}".format(index + 1));
    venue = []
    url = ('https://developers.zomato.com/api/v2.1/search?q={}' +
           '&start=0&count=1&lat={}&lon={}&sort=real_distance').format(row['name'], row['lat'], row['lng'])
    result = requests.get(url, headers = headers).json()
    if (len(result['restaurants']) > 0):
        venue.append(result['restaurants'][0]['restaurant']['name'])
        venue.append(result['restaurants'][0]['restaurant']['location']['latitude'])
        venue.append(result['restaurants'][0]['restaurant']['location']['longitude'])
        venue.append(result['restaurants'][0]['restaurant']['average_cost_for_two'])
        venue.append(result['restaurants'][0]['restaurant']['price_range'])
        venue.append(result['restaurants'][0]['restaurant']['user_rating']['aggregate_rating'])
        venue.append(result['restaurants'][0]['restaurant']['location']['address'])
        venues_information.append(venue)
    else:
        venues_information.append(np.zeros(6))

zomato_venues = pd.DataFrame(venues_information,
                             columns = ['venue', 'latitude',
                                        'longitude', 'price_for_two',
                                        'price_range', 'rating', 'address'])



# visualize the forusquare data on map

bangalore_map = folium.Map(location = [BLR_LAT, BLR_LON], zoom_start = 13)

for name, latitude, longitude in zip(foursquare_venues['name'], foursquare_venues['lat'], foursquare_venues['lng']):
    label = '{}'.format(name)
    label = folium.Popup(label, parse_html = True)
    folium.CircleMarker(
        [latitude, longitude],
        radius = 5,
        popup = label,
        color = 'blue',
        fill = True,
        fill_color = '#3186cc',
        fill_opacity = 0.3,
        parse_html = False).add_to(bangalore_map)

bangalore_map.save("maps/venue-by-foursquare.html")
bangalore_map


# visualize the zomato data on map
bangalore_map = folium.Map(location = [BLR_LAT, BLR_LON], zoom_start = 13)

for venue, address, latitude, longitude in zip(zomato_venues['venue'], zomato_venues['address'],
                                               zomato_venues['latitude'], zomato_venues['longitude']):
    label = '{}, {}'.format(name, address)
    label = folium.Popup(label, parse_html = True)
    folium.CircleMarker(
        [latitude, longitude],
        radius = 5,
        popup = label,
        color = 'red',
        fill = True,
        fill_color = '#cc3535',
        fill_opacity = 0.7,
        parse_html = False).add_to(bangalore_map)

bangalore_map.save("maps/venue-by-zomato.html")
bangalore_map


# Merging of the Foursquare and Zomato Data¶

foursquare_venues['lat'] = foursquare_venues['lat'].apply(lambda lat: round(float(lat), 4))
foursquare_venues['lng'] = foursquare_venues['lng'].apply(lambda lng: round(float(lng), 4))
zomato_venues['latitude'] = zomato_venues['latitude'].apply(lambda lat: round(float(lat), 4))
zomato_venues['longitude'] = zomato_venues['longitude'].apply(lambda lng: round(float(lng), 4))

# cleaning of data
dataset = pd.concat([foursquare_venues, zomato_venues], axis = 1)
```

```
dataset['lat_diff'] = dataset['latitude'] - dataset['lat']
dataset['lng_diff'] = dataset['longitude'] - dataset['lng']


# Creating a subset of the dataset as selected_venues
selected_venues = dataset[(abs(dataset['lat_diff']) <= 0.0004) & (abs(dataset['lng_diff']) <= 0.0004)].reset_index(drop = True)
selected_venues.head()

# clean and subset the data
selected_venues['average_price'] = selected_venues['price_for_two']/2
selected_venues = selected_venues.drop(columns = ['name', 'lat', 'lng', 'lat_diff', 'lng_diff', 'price_for_two'])



selected_venues = selected_venues[selected_venues['rating'] != 0.0]
print("Total venues available: {}".format(selected_venues.shape[0]))



# exploratory data analysis

venue_distribution = selected_venues['categories'].value_counts()
colors = cm.rainbow(np.linspace(0, 1, len(venue_distribution.index)))
plt.figure(figsize = (20, 12))
plt.xticks(rotation = 90)
plt.xlabel("Venue category", fontsize = 16)
plt.ylabel("Count", fontsize = 16)
plt.title("Count of venues of each category", fontsize = 16)
plt.bar(venue_distribution.index, venue_distribution.values, color = colors)


# 4.1.2. Analysis of Ratings of Restaurants¶


selected_venues['rating'] = selected_venues['rating'].astype(float)
rating = selected_venues['rating'].value_counts().sort_index()
plt.figure(figsize = (20, 12))
plt.bar(rating.index, rating.values, color = cm.rainbow(np.linspace(0, 1, len(rating.index))))
plt.xlabel("Rating", fontsize = 16)
plt.ylabel("Count", fontsize = 16)
plt.title("Count of venues with given rating", fontsize = 16)


# 4.1.2.A. Visualizing the Restaurant Ratings on OpenStreetMap¶

bins = [1.0, 2.0, 3.0, 4.0, 5.0]
labels = ['Bad', 'Okay', 'Good', 'Awesome']
selected_venues['rating_bin'] = pd.cut(selected_venues['rating'].astype(float), bins = bins, labels = labels, include_lowest = True)


color_map = {'Bad': 'red', 'Okay': 'orange', 'Good': 'lightgreen', 'Awesome': 'darkgreen'}

chandigarh_map = folium.Map(location = [BLR_LAT, BLR_LON], zoom_start = 13)

for name, address, latitude, longitude, rating_bin in zip(selected_venues['venue'],
                                                          selected_venues['address'],
                                                          selected_venues['latitude'],
                                                          selected_venues['longitude'],
                                                          selected_venues['rating_bin']):
    label = '{}, {}'.format(name, address)
    label = folium.Popup(label, parse_html = True)
    folium.Marker(
        [latitude, longitude],
        icon = folium.Icon(color = color_map[rating_bin]),
        popup = label).add_to(bangalore_map)

bangalore_map.save("maps/venues-by-ratings.html")
bangalore_map

# 4.1.3 Analysis of Average Price of Food¶

average_prices = selected_venues['average_price'].value_counts().sort_index()
plt.figure(figsize = (20, 12))
plt.scatter(average_prices.index,
            average_prices.values,
            s = average_prices.index,
            c = cm.rainbow(np.linspace(0, 1, len(average_prices.values))))
plt.xlabel("Average Price for Dining", fontsize = 16)
plt.ylabel("Venue Count", fontsize = 16)
plt.title("Count of venues with given average price", fontsize = 16)


# CLUSTERING

from sklearn.cluster import KMeans
clustering = selected_venues.drop(['categories', 'venue', 'address', 'rating_bin'], 1)
kMeans = KMeans(n_clusters = NO_OF_CLUSTERS, random_state = 0).fit(clustering)
selected_venues.insert(0, 'cluster_labels', kMeans.labels_)
#selected_venues.head(10)


# 4.2.2. Visualizing The Cluster on OpenStreetMap

# def kClusterColorMap(k):
#         if(k==3):
#             color_map = {0:"green", 1:"orange" , 2:"red"} # if cluster is of size 3
#             return color_map
#         elif(k==5):
#             color_map = {0:"lightblue", 1:"blue", 2:"green", 3:"orange" , 4:"red"} # if cluster size 5
#             return color_map
```

```
#         else :
#             print("ERROR: Not a suitable choice of cluster. Choose again.")
#color_map

NO_OF_CLUSTERS = 5

if(NO_OF_CLUSTERS == 3):
    color_map = {0:"green", 1:"orange" , 2:"red"}
elif(NO_OF_CLUSTERS == 5):
    color_map = {0:"lightblue", 1:"blue", 2:"green", 3:"orange" , 4:"red"}


# add venues to the map
markers_colors = []
for venue, address, cluster, latitude, longitude in zip(selected_venues['venue'],
                                                        selected_venues['address'],
                                                        selected_venues['cluster_labels'],
                                                        selected_venues['latitude'],
                                                        selected_venues['longitude']):
    label = folium.Popup(str(venue) + ', ' + str(address), parse_html = True)
    folium.CircleMarker(
        [latitude, longitude],
        radius = 5,
        popup = label,
        color = color_map[cluster],
        fill = True,
        fill_color = color_map[cluster],
        fill_opacity = 0.7).add_to(bangalore_map)



# add cluster centers to the map
for index, cluster in enumerate(kMeans.cluster_centers_):
    latitude = cluster[0]
    longitude = cluster[1]
    label = folium.Popup("Cluster: " + str(index), parse_html = True)
    folium.CircleMarker(
        [latitude, longitude],
        radius = 10,
        popup = label,
        color = color_map[index],
        fill = True,
        fill_color = color_map[index],
        fill_opacity = 0.7).add_to(bangalore_map)

bangalore_map.save("maps/venues-clusters.html")
bangalore_map



# CHECK DIFFERENT CLUSTERS

result = selected_venues[selected_venues['cluster_labels'] == 0]
print("Cluster: C0")
result.head(10).reset_index(drop = True)


result = selected_venues[selected_venues['cluster_labels'] == 1]
print("Cluster: C2")
result.head(10).reset_index(drop = True)


result = selected_venues[selected_venues['cluster_labels'] == 2]
print("Cluster: C3")
result.head(10).reset_index(drop = True)


result = selected_venues[selected_venues['cluster_labels'] == 3]
print("Cluster: C3")
result.head(10).reset_index(drop = True)

result = selected_venues[selected_venues['cluster_labels'] == 4]
print("Cluster: C5")
result.head(10).reset_index(drop = True)
```