# Coursera Capstone Project: IBM Data Science

Farrukh Naveed Anjum
anjum.farurkh@gmail.org
Rawalpindi, Pakistan

## 1. Introduction

Rawalpindi is the fourth-largest city in Pakistan by population, while the larger Islamabad Rawalpindi metropolitan area is the country's third-largest metropolitan area. People across the Pakistan settle her or come here daily either to visit Job, Travel Tour and Hospitalization.

According to the average hotel fair for one-night stay is $25 /night. Along with Restaurant charging at least $2 /meal.

Train stations are ideal locations for small businesses to set up shops, because they are hubs of human interaction where hundreds or even thousands of people day and night come and go. Each person in this flow of foot traffic is a potential customer who might need a specific item or purchase on impulse while waiting for a train. To succeed with retail at a train station, one must provide an accessible and affordable shopping experience offering merchandise or services that travelers might not quickly find elsewhere enroot while travelling.

## 2. Business Problem

Passengers coming out of Railway station needs a hotel for short stays, as well as station and train employees need to eat breakfast, lunch, dinner and snacks.

Although food sales are forbidden in some railway stations, many do offer merchants the opportunity to sell food. Foods that attract busy people on the go include egg sandwiches, fries, pizza, burgers, microwaveable or cold prepared meals.

Beverages such as coffee, tea, wraps, bottled water, soda and juice also sell well. Thus, the main objective of the project will be to find ideal spots in the city where fast food retail chains can be put up, aiming at the above demographic, thereby helping the **owners of the outlets to extract maximum profits** out of them.

## 3. Data

The data for this project has been retrieved and processed through multiple sources, giving careful considerations to the accuracy of the methods used.

### 3.1 Neighborhoods

The data of the neighborhoods in Rawalpindi can be extracted out by web scraping using BeautifulSoup library for Python. The neighborhood data is scraped from a **Paktive.com** webpage.

**Code**

```
#! /usr/bin/python
url_to_scrape = "https://www.paktive.com/postalcodes/Rawalpindi-Punjab.html"
```

```
page = requests.get(url_to_scrape)
text = []
data = dict()
data['Neighborhood'] = []
data['PostalCode'] = []
data['Borough'] = []
headers = []
if page.status_code==200:
   soup = BeautifulSoup(page.content, 'html.parser')

   city_areas = soup.find_all('span', {'class' : 'pcn'})
   city_postal_code = soup.find_all('span', {'class' : 'pcc'})
   city_list = [span.get_text() for span in city_areas]
   post_code_list = [span.get_text() for span in city_postal_code]
   headers = [city_list[0], post_code_list[0]]
   i=1
   for index, city in enumerate(city_list):
      try:
         data['Neighborhood'].append(city_list[i])
         data['PostalCode'].append(post_code_list[i])
         data['Borough'].append("Islamabad")
         i = i + 1
      except IndexError:
         data['Neighborhood'].append("Not assigned")
         data['PostalCode'].append("Not assigned")
         data['Borough'].append("Rawalpindi Village")
rawalpindi_df.to_csv('rawalpindi_data.csv')
```

## 3.2 Geocoding

The file contents from **rawalpindi_data.csv** is retrieved into a Pandas DataFrame. The latitude and longitude of the neighborhoods are retrieved using Google Maps Geocoding API. The geometric location values are then stored into the initial dataframe.

**Code**

```
#! /usr/bin/python
latitudes = []
longitudes = []
for nbd in rawalpindi_df ["Neighbourhood"] :
            place_name = nbd + ",Rawalpindi, Pakistan"
            url = 'https://maps.googleapis.com/maps/api/geocode/json?address={}&key={}'
            .format(place_name, API_KEY)
            obj = json.loads(requests.get(url).text)
            results = obj['results']
            lat = results[0]['geometry']['location']['lat']
            lng = results[0]['geometry']['location']['lng']
            latitudes.append(lat)
            longitudes.append(lng)
            rawalpindi_df ['Latitude'] = latitudes
            rawalpindi_df ['Longitude'] = longitude
```

### 3.3 Venue Data

From the location data obtained after Web Scraping and Geocoding, the venue data is found out by passing in the required parameters to the FourSquare API, and creating another DataFrame to contain all the venue details along with the respective neighborhoods.

**Code**

```
def getNearbyVenues(names, latitudes, longitudes, limit=100, radius=500)
:

    venues_list=[]
    for name, lat, lng in zip(names, latitudes, longitudes):
        print(name)

        # create the API request URL
        url = 'https://api.foursquare.com/v2/venues/explore?&client_id={
}&client_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(
            CLIENT_ID,
            CLIENT_SECRET,
            VERSION,
            lat,
            lng,
            radius,
            limit)

        # make the GET request
        results = requests.get(url).json()["response"]['groups'][0]['ite
ms']
        # return only relevant information for each nearby venue
        venues_list.append([(
            name,
            lat,
            lng,
            v['venue']['name'],
            v['venue']['location']['lat'],
            v['venue']['location']['lng'],
            v['venue']['categories'][0]['name']) for v in results])

    nearby_venues = pd.DataFrame([item for venue_list in venues_list for
item in venue_list])
    nearby_venues.columns = ['Neighborhood',
                  'Neighborhood Latitude',
                  'Neighborhood Longitude',
                  'Venue',
                  'Venue Latitude',
                  'Venue Longitude',
                  'Venue Category']

    return nearby_venues
rawalpindi_venues = getNearbyVenues(names=neighborhoods['Neighborhood'],
                                    latitudes=neighborhoods['Latitude'],
                                    longitudes=neighborhoods['Longitude']
```

```
                                        )
print(rawalpindi_venues.shape)
rawalpindi_venues.head()
```

# 4 Methodology

A thorough analysis of the principles of methods, rules, and postulates employed g=have been made in order to ensure the inferences to be made are as accurate as possible.

## 4.1 Accuracy of the Geocoding API

In the initial development phase with Open Cage Geocoder API, the number of erroneous results were of an appreciable amount, which led to the development of an algorithm to analyze the accuracy of the Geocoding API used. In the algorithm developed, Geocoding API from various providers were tested, and in the end, Google Maps Geocoder API turned out to have the least number of collisions (errors) in our analysis.

**Code**

```
#! /usr/bin/python
col = 0
explored_lat_lng = []
df = rawalpindi_df
for lat, lng, neighbourhood in zip(df['Latitude'], df['Longitude'],
df['Neighbourhood']):
        if (lat, lng) in explored_lat_lng:
            col = col + 1
        else:
            explored_lat_lng.append((lat, lng))
print("Collisions : ", col)
```

## 4.2 Folium

Folium builds on the data wrangling strengths of the Python ecosystem and the mapping strengths of the leaflet.js library. All cluster visualization are done with help of Folium which in turn generates a Leaflet map made using OpenStreetMap technology.

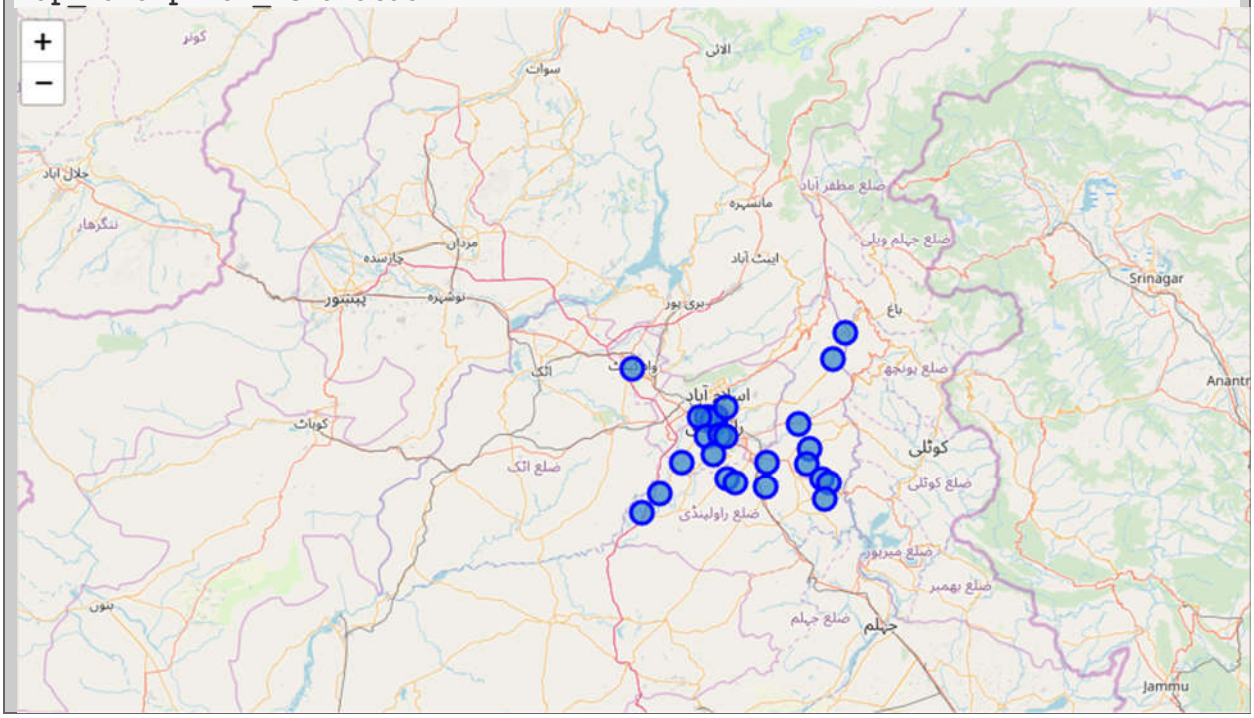
**Code**

```
map_rawalpindi_islamabad = folium.Map(location=[islamabad_location['lat'
], islamabad_location['lon']], zoom_start=8)

# add all postcoe markers on te map
for lat, lng, borough, neighborhood in zip(neighborhoods['Latitude'], ne
ighborhoods['Longitude'], neighborhoods['Borough'], neighborhoods['Neigh
borhood']):
    label = '{}, {}'.format(neighborhood, borough)
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=8,
```

```
        popup=label,
        color='blue',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=False).add_to(map_rawalpindi_islamabad)

map_rawalpindi_islamabad
```




### 4.3 One hot encoding

One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. For the K-means Clustering Algorithm, all unique items under Venue Category are one-hot encoded.

**Code**

```
#! /usr/bin/python
df_onehot = pd.get_dummies(rawalpindi_venues[['Venue Category']], prefix
="", prefix_sep="")

# add neighborhood column back to dataframe
df_onehot['Neighborhood'] = rawalpindi_venues['Neighborhood']

# move neighborhood column to the first column
fixed_columns = [df_onehot.columns[-1]] + list(df_onehot.columns[:-1])
df_onehot = df_onehot[fixed_columns]
df_onehot.head()
```

## 4.4 Top 10 most common venues

Due to high variety in the venues, only the top 10 common venues are selected and a new DataFrame is made, which is used to train the K-means Clustering Algorithm.

**Code**

```
def return_most_common_venues(row, num_top_venues):
    row_categories = row.iloc[1:]
    row_categories_sorted = row_categories.sort_values(ascending=False)

    return row_categories_sorted.index.values[0:num_top_venues]


rawalpindi_downtown_grouped = df_onehot.groupby('Neighborhood').mean().reset_index()

num_top_venues = 10

#Because we will look only at 4-top most common venues let's simpliy the columns naming
columns = ['1st Most Common Venue',
        '2nd Most Common Venue',
        '3rd Most Common Venue',
        '4th Most Common Venue',
        '5th Most Common Venue',
        '6th Most Common Venue',
        '7th Most Common Venue',
        '8th Most Common Venue',
        '9th Most Common Venue',
        '10th Most Common Venue']

# create columns according to number of top venues
columns = ['Neighborhood'] + columns

# create a new dataframe
neighborhoods_venues_sorted = pd.DataFrame(columns=columns)
neighborhoods_venues_sorted['Neighborhood'] = rawalpindi_downtown_grouped['Neighborhood']

for ind in np.arange(rawalpindi_downtown_grouped.shape[0]):
    neighborhoods_venues_sorted.iloc[ind, 1:] =
return_most_common_venues(rawalpindi_downtown_grouped.iloc[ind, :], num_top_venues)

neighborhoods_venues_sorted
```

## 4.5 Optimal number of clusters

Silhouette Score is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from -1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. Based on the Silhouette Score of various clusters below 20, the optimal cluster size is determined.

**Code**

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
def plot(x, y, xlabel, ylabel):
    plt.plot(x, y, 'o-')
    plt.figure(figsize = (20,10))
    plt.xlabel("No. of clusters")
    plt.ylabel("Silhouette Score")
    plt.show()
```

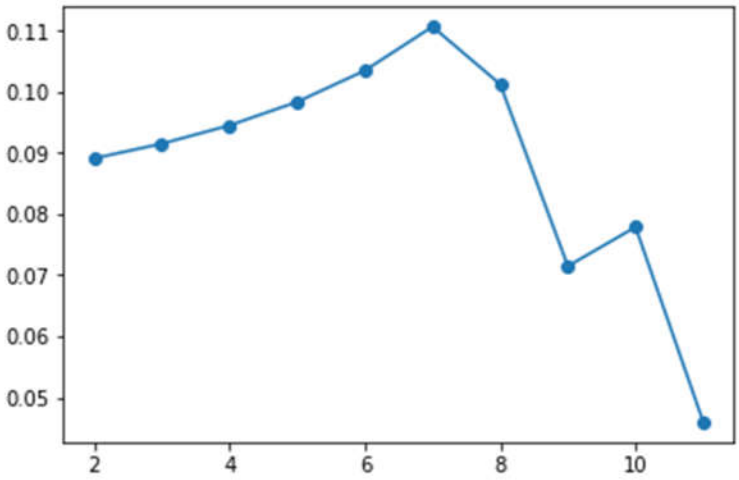



**Fig:** Silhouette Score vs No. of Clusters

### 4.6 K-means clustering

The venue data is then trained using K-means Clustering Algorithm to get the desired clusters to base the analysis on. K-means was chosen as the variables (Venue Categories) are huge, and in such situations K-means will be computationally faster than other clustering algorithms.

**Code**

```
# set number of clusters
kclusters = optimal_value

grouped_clustering = rawalpindi_downtown_grouped.drop('Neighborhood', 1)

# run k-means clustering
kmeans = KMeans(n_clusters=kclusters, random_state=0).fit(grouped_cluste
ring)

# check cluster labels generated for each row in the dataframe
kmeans.labels_[0:10]
```

## 5 Results

The neighborhoods are divided into n clusters where n is the number of clusters found using the optimal approach. The clustered neighborhoods are visualized using different colors so as to make them distinguishable.

**Code**

```
# create map
cluster_map = folium.Map(location=[islamabad_location['lat'], islamabad_location['lon']], zoom_
start=10)

# set color scheme for the clusters
x = np.arange(kclusters)
ys = [i + x + (i*x)**2 for i in range(kclusters)]
colors_array = cm.rainbow(np.linspace(0, 1, len(ys)))
rainbow = [colors.rgb2hex(i) for i in colors_array]

# add markers to the map
markers_colors = []
for lat, lon, poi, cluster in zip(rawalpindi_merged['Latitude'],
                                  rawalpindi_merged['Longitude'],
                                  rawalpindi_merged['Neighborhood'],
                                  rawalpindi_merged['Cluster Labels']):
    if not math.isnan(cluster):
        label = folium.Popup(str(poi) + ' Cluster ' + str(cluster), parse_html=True)
        folium.CircleMarker(
            [lat, lon],
            radius=5,
            popup=label,
            color=rainbow[int(cluster)-1],
            fill=True,
            fill_color=rainbow[int(cluster)-1],
            fill_opacity=0.7).add_to(cluster_map)

cluster_map
```

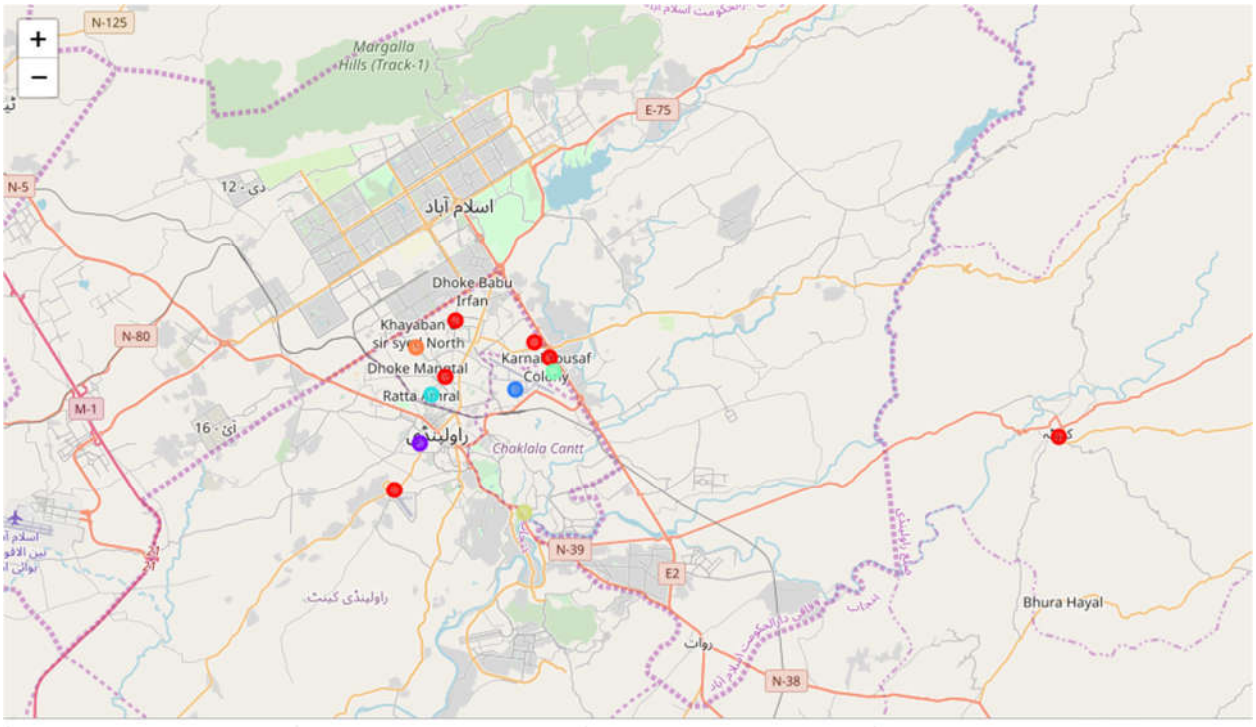



Figure 3: Neighborhoods of Rawalpindi (Clustered).

# 6. Discussion

After analyzing the various clusters produced by the Machine learning algorithm, cluster no.0 is a prime fit to solving the problem of finding a cluster with common venue as a train station mentioned before.

| | Neighborhood | PostalCode | Borough | Latitude | Longitude | Cluster Labels | 1st Most Common Venue | 2nd Most Common Venue | 3rd Most Common Venue | 4th Most Common Venue | 5th Most Common Venue | 6th Most Common Venue | 7th Most Common Venue | 8th Most Common Venue |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | Bhall | 47570 | Islamabad | 33.624901 | 73.113793 | 0.0 | Bakery | Burger Joint | Bus Stop | Shopping Mall | Breakfast Spot | Bus Station | Café | Clothing Store |
| 17 | Dhamial Camp Rawalpindi | 46500 | Islamabad | 33.565709 | 73.030530 | 0.0 | Fast Food Restaurant | Gas Station | Cricket Ground | Breakfast Spot | Burger Joint | Bus Station | Bus Stop | Café |
| 27 | Kahuta | 47330 | Islamabad | 33.589614 | 73.388553 | 0.0 | Shopping Mall | Train | Breakfast Spot | Burger Joint | Bus Station | Bus Stop | Café | Clothing Store |
| 60 | Rawalpindi Raja Town | 46320 | Islamabad | 33.631516 | 73.106109 | 0.0 | Outlet Mall | Gas Station | Train | Cricket Ground | Breakfast Spot | Burger Joint | Bus Station | Bus Stop |
| 61 | Rawalpindi Satellite Town | 46300 | Islamabad | 33.641235 | 73.063475 | 0.0 | Bakery | Cricket Ground | Frozen Yogurt Shop | Clothing Store | Shopping Mall | Breakfast Spot | Burger Joint | Bus Station |
| 62 | Rawalpindi Urdu Bazar | 46020 | Islamabad | 33.616461 | 73.057487 | 0.0 | Jewelry Store | Flea Market | Train | Cricket Ground | Breakfast Spot | Burger Joint | Bus Station | Bus Stop |

Figure 4: Cluster having Train Station as most common venue

The five places namely Bhall, Dhamial Camp, Kahula, Raja Town, Satellite Town and Urdu Bazar fall in the outskirts of the city of Rawalpindi, hence the demographic of the population in these areas fall under the middle class of the society.

# 7 Conclusion

The middle class in Pakistan can loosely be defined as the section of society that comprises households with a minimum monthly income of $320. A household on average consists of six members. If this categorization is correct in a broad sense, the size of the middle class in our country has grown to nearly 50 million of Pakistan's total population of 200 million. This estimate is not based on any scientific survey but on anecdotal evidence and social observations. However, one can argue that the size of Pakistan's upper middle class is smaller, not exceeding 20 million at best.

Hence opening the a Hotel (Along with Restaurant) near railway stations area can get one $1250 per day profit in case average of 50 people stay there.