Coursera Capstone Project: IBM Data Science  
Farrukh Naveed Anjum  
anjum.farurkh@gmail.org  
Rawalpindi, Pakistan

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

3.1 NeighborhoodsThe 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

|  |
| --- |
| 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 GeocodingThe 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

|  |
| --- |
| 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 DataFrom 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

|  |
| --- |
| 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]['items']  *# 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 MethodologyA 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 APIIn 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

|  |
| --- |
| 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 FoliumFolium 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

|  |
| --- |
| 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'], neighborhoods['Longitude'], neighborhoods['Borough'], neighborhoods['Neighborhood']):  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 encodingOne 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

|  |
| --- |
| 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 venuesDue 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 clustersSilhouette 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

|  |
| --- |
| 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() |

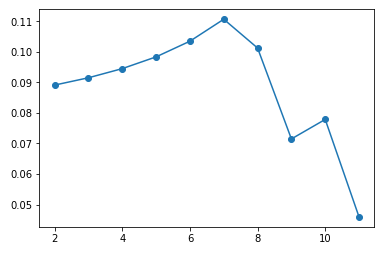
****

Fig: Silhouette Score vs No. of Clusters

4.6 K-means clusteringThe 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

|  |
| --- |
| 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\_clustering)  *# check cluster labels generated for each row in the dataframe*  kmeans.labels\_[0:10] |

5 ResultsThe 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 |

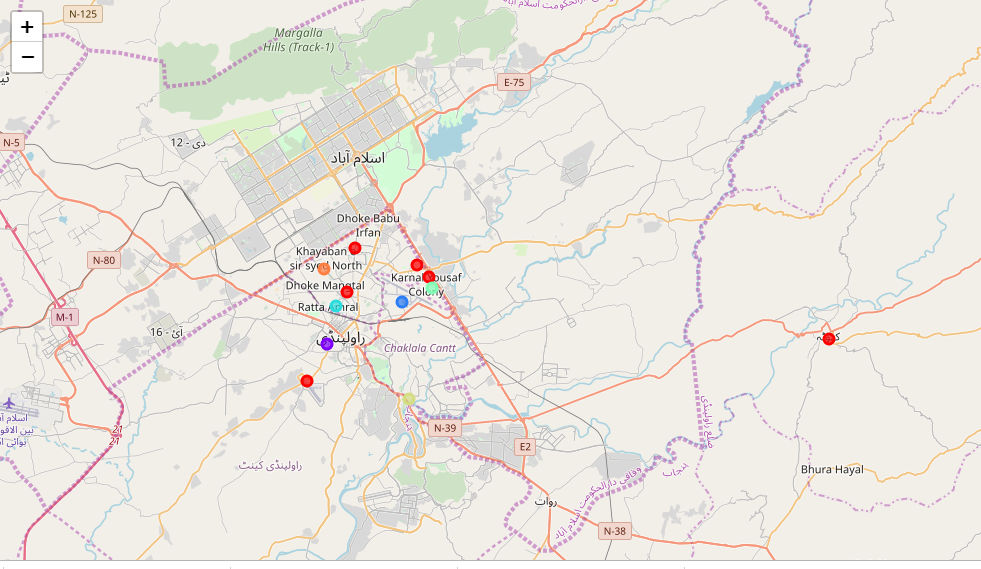
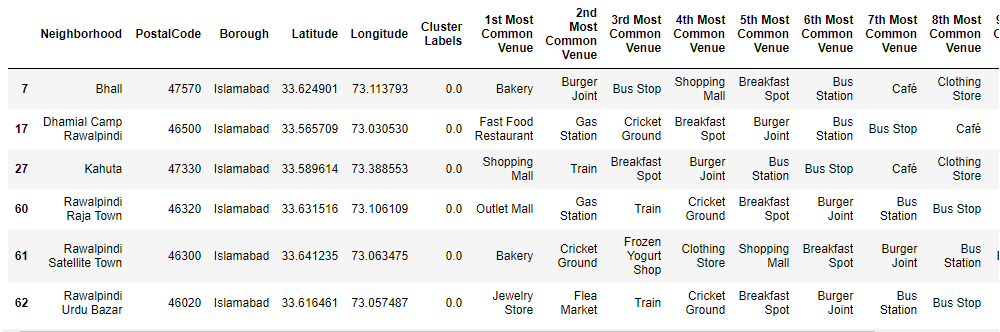
Code  


Figure 3: Neighborhoods of Rawalpindi (Clustered).

6. DiscussionAfter 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.

  
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 ConclusionThe 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.