Best Location to Start an Asian Restaurant in Jozi

Derrick Mashwama

16 June 2021

1. Introduction

a. City Background

The city of Johannesburg is located in Gauteng province (one of 9 provinces in the country). It is often referred to as Jozi, Joburg or Egoli and it's the largest city in South Africa. As of 2021, the population of Joburg is estimated to be 5.7 million with 76.4% are Black African, 12.3% are White people, 5.6% are coloured people, and 4.9% are Indian/Asian.

Johannesburg has the largest man-made forest in the world, it is also one of the largest 50 urban agglomerations in the world and the largest city in the world that is not located on near a water body.

The city is divided into 7 distinct regions or municipalities:

- Regions or municipalities can be viewed as boroughs
- Each municipality is constructed from locations or suburbs
    - Location or suburbs can be viewed as neighbourhoods

Regions	Areas/Municipality
A	Diepsloot, Midrand, Ivory Park
B	Rosebank, Parktown
C	Roodepoort, North Gate
D	Soweto, Diepkloof
E	Woodmead, Wynberg, Alexandra, Bruma
F	Johannesburg CBD, Aeroton, South Gate
G	Lenasia, Ennerdale, Orange Farm

b. Problem Description

Joburg is very diverse in its population distribution and there is a fair mix of ethnic groups within each location/suburb. We will use clustering and segmentation methods to investigate and find a good location for establishing an Asian restaurant within the Midrand municipality.
Special emphasis will be on an area where the restaurant will be first of its kind or increase on a currently small footprint.

2. Analytic Approach

Data Description

This means identifying what type of patterns will be needed to address the question most effectively. If the question is to determine probabilities of an action, then a predictive model might be used. If the question is to show relationships, a descriptive approach maybe be required.

This project will look into the municipality/Region data and the locations/suburbs within each municipality in Joburg.

For each location in Joburg, we will look at the venues (i.e. restaurants) that exist within that location.

a. Locations within Joburg:

This data is retrieved from Johannesburg postal code data (on: https://www.southafricapostcode.com/location/gauteng/city-of-johannesburg/):

- This data will be scrapped over 57 pages of data within above url
- This data does not contain the Region/Municipality data, this data will be retrieved from other sources

Browse Location - City of Johannesburg, Gauteng - Page 1

Location	Municipal	District	State	Postcode
Abbotsford		City of Johannesburg	Gauteng	2192
Aeroton		City of Johannesburg	Gauteng	2013
Aeroton		City of Johannesburg	Gauteng	2190
Airdlin		City of Johannesburg	Gauteng	2157
Alan Manor		City of Johannesburg	Gauteng	2091
Albertskroon		City of Johannesburg	Gauteng	2195
Albertsville		City of Johannesburg	Gauteng	2195
Aldara Park		City of Johannesburg	Gauteng	2194
Alexandra		City of Johannesburg	Gauteng	2014

b. Region/Municipality data in Joburg:

Municipality data is retrieved from (https://www.joburg.org.za/about_/regions/Pages/City-of-Johannesburg-regions.aspx):

- Data is arranged in 7 different Regions (A to G)
- Data is scrapped from each Region, however the format within each page is not common and some pages present data in html while others in pdf
    - Will be explained further in Section 3

- **Venues data within each location in Joburg:**

This data will be retrieved from Foursquare by using predefined credentials for the Foursquare API

3. Methodology

Methodology section which represents the main component of the report where you discuss and describe any exploratory data analysis that you did, any inferential statistical testing that you performed, if any, and what machine learnings were used and why.

a. Data Requirements

- The data required consists of location/suburb information that contains postal codes
- Postal code data is used to translate to latitude and longitude data
- The Region/Municipality of each location is then concatenated with the location data to provide the final dataset as shown below

	Location	District	State	PostCode	Lat	Long	Region
0	Barbeque Downs	City of Johannesburg	Gauteng	1684	-26.006963	28.072761	Diepsloot_Midrand
1	Beverley	City of Johannesburg	Gauteng	2191	-26.007799	28.017145	Diepsloot_Midrand
2	Brendavere	City of Johannesburg	Gauteng	2021	-26.015073	27.985670	Diepsloot_Midrand
3	Broadacres	City of Johannesburg	Gauteng	2021	-25.994151	28.012097	Diepsloot_Midrand
4	Buccleuch	City of Johannesburg	Gauteng	2090	-26.057261	28.107827	Diepsloot_Midrand

b. Data Collection and preparation

- Locations within Joburg is strapped from https://www.southafricapostcode.com/location/gauteng/city-of-johannesburg/ over 57 pages of content.

```
#Defining the scrappper function
def scrapper(webpage, first_page, page_total):

    JoziPages = np.arange(first_page, (page_total+1), 1) #
    for page in JoziPages:
        next_page = webpage + str(page)
        print("URL: ",next_page)
        response = requests.get(str(next_page))
        Jozi_soup = BeautifulSoup(response.text,'html5lib')
        Jozi_data = Jozi_soup.find_all("tr")

        for row in Jozi_data:
            col = row.find_all('td')
            if (col != []): #ignore blank columns
                loc = col[0].text #assign first column to loc
                location.append(loc) #add loc to location list
                #print('location: ',loc)

                muni = col[1].text
                municipal.append(muni)

                distr = col[2].text
                district.append(distr)

                stat = col[3].text
                state.append(stat)

                postc = col[4].text
                postcode.append(postc)

        sleep(randint(2,50))
```

```
  #page has 57 pages (https://www.southafricapostcode.com/location/gauteng/city-of-johannesburg/?page=57)
scrapper('https://www.southafricapostcode.com/location/gauteng/city-of-johannesburg/?page=', 1,8)
```

ace time delays (2mins) between each scrapping function so not to overload the server

```
#the lists continues being populated from the last entry
sleep(120) #wait 2mins
scrapper('https://www.southafricapostcode.com/location/gauteng/city-of-johannesburg/?page=', 9,16)
sleep(120) #wait 2mins
scrapper('https://www.southafricapostcode.com/location/gauteng/city-of-johannesburg/?page=', 17,24)
sleep(120) #wait 2mins
scrapper('https://www.southafricapostcode.com/location/gauteng/city-of-johannesburg/?page=', 25,32)
sleep(120) #wait 2mins
scrapper('https://www.southafricapostcode.com/location/gauteng/city-of-johannesburg/?page=', 33,40)
sleep(120) #wait 2mins
scrapper('https://www.southafricapostcode.com/location/gauteng/city-of-johannesburg/?page=', 41,48)
sleep(120) #wait 2mins
scrapper('https://www.southafricapostcode.com/location/gauteng/city-of-johannesburg/?page=', 49,57)
```

Here we construct a scrapper function that retrieves data from each page and stores it in a dataframe called Jozi_df

```
Jozi_df = pd.DataFrame({'Location':location,'Municipality':municipal,'District':district,'State':state,'PostCode':postcode})
```

```
Jozi_df.head()
```

	Location	Municipality	District	State	PostCode
0	Abbotsford	NaN	City of Johannesburg	Gauteng	2192
1	Aeroton	NaN	City of Johannesburg	Gauteng	2013
2	Aeroton	NaN	City of Johannesburg	Gauteng	2190
3	Airdlin	NaN	City of Johannesburg	Gauteng	2157
4	Alan Manor	NaN	City of Johannesburg	Gauteng	2091

a. Assigning coordinates to Joburg locations requires the use of geocoder library. Here we use the Jozi_df data's postal codes to translate to lat/long data

```
for index, col_name in Jozi_df.iterrows():
    print(col_name['Location'])
    Lat = geocoder.google("{}, {}, Johannesburg".format(col_name['PostCode'],col_name['Location']), key=API_KEY).lat
    Long = geocoder.google("{}, {}, Johannesburg".format(col_name['PostCode'],col_name['Location']), key=API_KEY).lng

    LatsIn.append(Lat)
    LongsIn.append(Long)
```

create DF with coords

```
Jozi_lats = pd.DataFrame({'Lat':LatsIn})
#Jozi_lats=Jozi_lats.drop(Jozi_lats.index[1123])
Jozi_longs = pd.DataFrame({'Long':LongsIn})
```

```
#join the dataframes
Jozi_combined = Jozi_lats.join(Jozi_longs)
Jozi_combined.head()
```

```
#join coords to the main data
Jozi_df = Jozi_df.join(Jozi_combined)
Jozi_df.head()
```

	Location	District	State	PostCode	Lat	Long
0	Abbotsford	City of Johannesburg	Gauteng	2192	-26.143112	28.066176
1	Aeroton	City of Johannesburg	Gauteng	2013	-26.249736	27.982456
2	Aeroton	City of Johannesburg	Gauteng	2190	-26.249736	27.982456
3	Airdlin	City of Johannesburg	Gauteng	2157	-26.026442	28.060636
4	Alan Manor	City of Johannesburg	Gauteng	2091	-26.278238	27.993722

b. Region/Municipality data is retrieved from (https://www.joburg.org.za/about_/regions/Pages/City-of-Johannesburg-regions.aspx), here the data for each Region is retrieved with a specialized function as the format of the data varies from Region to Region

Function for retrieving Regions A, C, D and G

a. For Regions A, C, D and G the format is similar and a single function, Regioner, is used to retrieve the locations within each municipality:

Suburbs in Region A

Airdlin,

Barbeque, Barbeque Downs, Barbeque Downs Extensions, Beverley, Beverley Ext.1 & 2, Beverley Extensions, Bloubosrand Extension, Blue Hills, Blue Hills Extension, Bo
408-Jr Ext.408 , Brendavere, Broadacres, Broadacres Ext.1 & 2, Broadacres Extensions, Broadacres Park Ext.1 & 10, Buccleuch,

Carlswald, Carlswald Estate, Chartwell, Country View, Country View Extensions, Craigavon, Craigavon Ext.1, Crowthorne ,

Dainfern, Dainfern Extensions, Dainfern Ridge, Diepsloot 388-Jr Ext.388, Diepsloot, Diepsloot Wes, Diepsloot Wes Extensions,

Ebony Park, Ebony Park Extensions, Erand, Erand Ext.1 & 2, Erand Gardens Extensions,

Farmall, Farmall Ext.1, Fourways Extensions,

Function for retrieving Regions A, C, D and G

```
def Regioner(input_url,municipality):
    region_response = requests.get(input_url)
    region_soup = BeautifulSoup(region_response.text,'html.parser')
    region_rows = region_soup.findAll('div',{'class':'article-content description
    Loc = []
    Region = []
    for x in region_rows:
        item = x.find_all("td")
        Loc.append(item[0].text.strip().strip(' '))
        #Region.append(item[1].text.strip().replace('Suburbs in ',''))

    #make the two lists into dataframes
    test_df = pd.DataFrame(Loc,columns=['Locations'])
    region_df = pd.DataFrame(Region,columns=['Region'])

    #Combine the dataframes
    combined_df = test_df.join(region_df)

    a_df = pd.DataFrame(columns=['Loc','Region'])
    for index, col_name in combined_df.iterrows():
        loc = col_name['Locations'].strip().split(',')
        #reg = col_name['Region'].strip()
        #a_df = a_df.append({'Loc':loc,'Region':reg},ignore_index=True)
        a_df = a_df.append({'Loc':loc},ignore_index=True)

    a = pd.DataFrame.from_records( a_df['Loc']).transpose()
    #a[0].str.strip() #remove leading/trailling spaces

    b = a.join(a_df['Region'])

    #replace NaN values
    b['Region'].fillna(municipality,inplace=True)
    b.rename(columns={0:'Location'},inplace=True)

    #remove whitespaces
    whitespace_remover(b)
    return b[['Location','Region']]
```

b. For Region F a special function (RegionerF) is created to retrieve data as it's data is not the same as preceding regions:

Suburbs		
Aeroton,	Aspen Hills,	Bassonia,
Bellevue East,	Bellevue,	Benrose,
Berea,	Bertrams,	Braamfontein,
City and Suburban,	City and Suburban Industrial,	City Deep,
City West,	Crown Gardens,	Denver,
Doornfontein,	Droste Park,	Elandspark,
Elcedes,	Fairview,	Ferreirasdorp,

Function for retrieving Region F

```
def RegionerF(input_url,municipality):
    region_response = requests.get(input_url)
    region_soup = BeautifulSoup(region_response.text,'html.parser')
    region_rows = region_soup.findAll('div',{'class':'article-content description'})

    Loc = []
    output_df = pd.DataFrame(columns=['Location','Region'])
    for x in region_rows:
        item = x.find_all("td")
        for i in range(len(item)): #range iterates from 0 to number in range brackets
            Loc.append(item[i].text.strip(',').strip(', ,'))
    #make dataframe from Loc data
    test_df = pd.DataFrame(Loc,columns=['Locations'])

    a_df = pd.DataFrame(columns=['Loc','Region'])
    for index, col_name in test_df.iterrows():    #if you don't do this, items come out in s
        loc = col_name['Locations'].strip().split(',')
        a_df = a_df.append({'Loc':loc},ignore_index=True)

    output_df = pd.DataFrame.from_records( a_df['Loc'])
    output_df.rename(columns={0:'Location'},inplace=True)
    output_df = output_df.append({'Region':municipality}, ignore_index=True)
    output_df['Region'].fillna(municipality,inplace=True)

    return output_df
```

c. For Region B, we create a special function (RegionerB) that retrieves the data for all locations that fall within Region B:

Suburbs in Region B
ALBERTSKROON
ALBERTVILLE
ALBERTVILLE EXT.1
ALBERTVILLE EXT.10
ALBERTVILLE EXT.11
ALBERTVILLE EXT.12
ALBERTVILLE EXT.2
ALBERTVILLE EXT.4
ALBERTVILLE EXT.5
ALBERTVILLE EXT.6
ALBERTVILLE EXT.7
ALBERTVILLE EXT.8
ALBERTVILLE EXT.9
ALDARAPARK
AUCKLAND PARK
AUCKLAND PARK EXT.2
AUCKLAND PARK EXT.3
BERARIO
BERGBRON
BERGBRON EXT.1
BERGBRON EXT.10
BERGBRON EXT.11
BERGBRON EXT.12

Function for retrieving Region B

```
def RegionerB(input_url,municipality):
    region_response = requests.get(input_url)
    region_soup = BeautifulSoup(region_response.text,'html.parser')
    region_rows = region_soup.findAll('div',{'class':'ms-rtestate-field'})

    output_df = pd.DataFrame(columns=['Location','Region'])
    Loc = []
    for x in region_rows:
        item = x.find_all("div") #oddly elements are placed in 'div' and not 'td'
        for i in range(len(item)): #range iterates from 0 to number in range brackets
            Loc.append(item[i].text.strip())

    #make dataframe from Loc data
    output_df = pd.DataFrame(Loc,columns=['Location'])
    output_df = output_df.append({'Region':municipality}, ignore_index=True)
    output_df['Region'].fillna(municipality,inplace=True)

    return output_df
```

d. Region E data is presented as pdf format. For this we use camelot library to scrap data from Region E url:

REGION E WARD LOCATION & SUBURBS

Regional Manager: Andile Seeko (011 582 1584/073 195 5143): andilese@joburg.org.z

WARD	LOCATION
32	Linbro Park, Portion of Frankenwald, Greenstone, Buccleuch, Modderfontein X 2,Founders Hill ,Sebenza X6 , Linbro Park AH ,Linbro Park EXT 1, Greenstone Klipfontein View & Ext, Long Meadow, Waterval, Westlake, Westfiled, Longlak
72	Dunhill, Fairmount ,Fairmount Ridge EXT 1,2 Fairvale, Fairvale EXT 1, Glenka Glensan, Linksfield EXTs 1, 2, 3, 4, 5, Linksfield North, Linksfield Ridge EXT 1 Sandringham, Silvamonte EXT1,Talbolton, Sunningdale, Sunningdale Ext 1,2,3,4,5, 7,8,11,12,Sunningdale Percelia, Percelia Estate, Percelia Ext, Sydenham, Glenhazel, and Orange Grove North of 14th Street,Viewcrest
73	Bellevue, Fellside, Houghton Estate, Mountain View, Norwood, Oaklands, Orchards, Riviera, Victoria EXT2,The Gardens , Orange Groove, Killarney
74	Wanderers, Waverley, Bagleyston, Birdhaven, Birnam, Bramley Gardens, Cheltondale, Chetondale EXT1, 2, 3, Elton Hill EXTs 1, 2, 3, 4,Gresswold, Hawkins Estate, Hawkins Estate EXT1, Highlands North EXT2, 3, 4, 5, 6, 9, Highlands North Extension, 1,Kentview,Kew,Maryvale, Melrose, Melrose Estat Melrose Ext 1,2, Melrose North Ext 1,2,3,4,57,8,Orchards From Hamlen to African Street(Highroad border), 1,2,Raedene Estate, Raedene Estate Ext 1,Raumarais Park ,Rouxville, Savoy Estate, Winston Ridge, Ext1,2,The Gardens, Abbotsford.

Function for retrieving Region E data. This is pdf data that is extracted using camelot

```
#import libraries
import camelot
import ghostscript
```

```
def RegionerE(input_url,municipality):
    #file = "https://www.joburg.org.za/about_/regions/Documents/Region%20E/Region%20E%20Ward%20Location%20%20suburbs%20
    Table_df = pd.DataFrame()
    #read the tables on the file
    tables = camelot.read_pdf(input_url, pages='all' )
    tables

    #check shape of the tables
    for x in range (3):
        Table_df = Table_df.append(tables[x].df,ignore_index = True)

    #clean rows and columns
        #rename columns
    Table_df.rename(columns={0:'Ward',1:'Location'},inplace=True)
    Table_df = Table_df[~Table_df['Ward'].str.contains('WARD')]

    #from https://www.mikulskibartosz.name/how-to-split-a-list-inside-a-dataframe-cell-into-rows-in-pandas/
    # convert 'Location' data into a list (i.e. has [])
    Split_df = pd.DataFrame()
    for col_name in Table_df['Location']:    #if you don't do this, items come out in series
        loc = col_name.split(',')
        Split_df = Split_df.append({'Loc':loc},ignore_index=True)

    #Split the comma separated 'Loc' column into new columns
    Split_df = Split_df.Loc.apply(pd.Series)


    #merge split and original data
    output_df = Split_df.merge(Table_df, left_index = True, right_index = True).drop(['Location'], axis = 1)

    output_df = output_df.melt(id_vars = ['Ward'], value_name = "Location").dropna().drop("variable", axis = 1)
    output_df = output_df.drop('Ward',axis=1)
    #add region
    output_df['Region'] = municipality
    whitespace_remover(output_df)
    return(output_df)
```

All the location data within each municipality is then gathered into Municipality_df dataframe:

Construct and populate Municipalities table

```
#delete the contents of df
Municipality_df = Municipality_df.iloc[0:0]


Municipality_df = Municipality_df.append(Regioner(RegionA_url,'Diepsloot_Midrand'),ignore_index=True)
Municipality_df = Municipality_df.append(RegionerB(RegionB_url,'Northcliff_Randburg'),ignore_index=True)
Municipality_df = Municipality_df.append(Regioner(RegionC_url,'Roodepoort'),ignore_index=True)
Municipality_df = Municipality_df.append(Regioner(RegionD_url,'Soweto'),ignore_index=True)
Municipality_df = Municipality_df.append(Regioner(RegionG_url,'Ennerdale_OrangeFarm'),ignore_index=True)
Municipality_df = Municipality_df.append(RegionerF(RegionF_url,'InnerCity'),ignore_index=True)
Municipality_df = Municipality_df.append(RegionerE(RegionE_url,'Fourways'),ignore_index=True)
```

```
Municipality_df.head()
```

	Location	Region
0	Airdlin	Diepsloot_Midrand
1	Barbeque	Diepsloot_Midrand
2	Barbeque Downs	Diepsloot_Midrand
3	Barbeque Downs Extensions	Diepsloot_Midrand
4	Beverley	Diepsloot_Midrand

We merge the location and municipality dataframes to generate final data as per Data Requirements section:

Merge Municipalities and Locations

```
Muni_Loc_df = pd.merge(Jozi_data,Municipality_df,on='Location',how='left')
```

```
#drop all non-matched Locations
Muni_Loc_df = pd.DataFrame(Muni_Loc_df.dropna(subset=['Region']))
Muni_Loc_df.head()
```

	Location	District	State	PostCode	Lat	Long	Region
1	Aeroton	City of Johannesburg	Gauteng	2013	-26.249736	27.982456	InnerCity
10	Allen's Nek	City of Johannesburg	Gauteng	1709	-26.133470	27.909717	Roodepoort
14	Amorosa	City of Johannesburg	Gauteng	1724	-26.100808	27.870703	Roodepoort
15	Anchorville	City of Johannesburg	Gauteng	1827	-26.340688	27.828945	Ennerdale_OrangeFarm
18	Armadale	City of Johannesburg	Gauteng	2013	-26.153074	28.095781	Soweto

```
Muni_Loc_df.to_csv('Muni_loc.csv',index=False)
```

c. Data Understanding

To better understand the locations and distribution of suburbs within Joburg, we use folium library to display the data in a map:

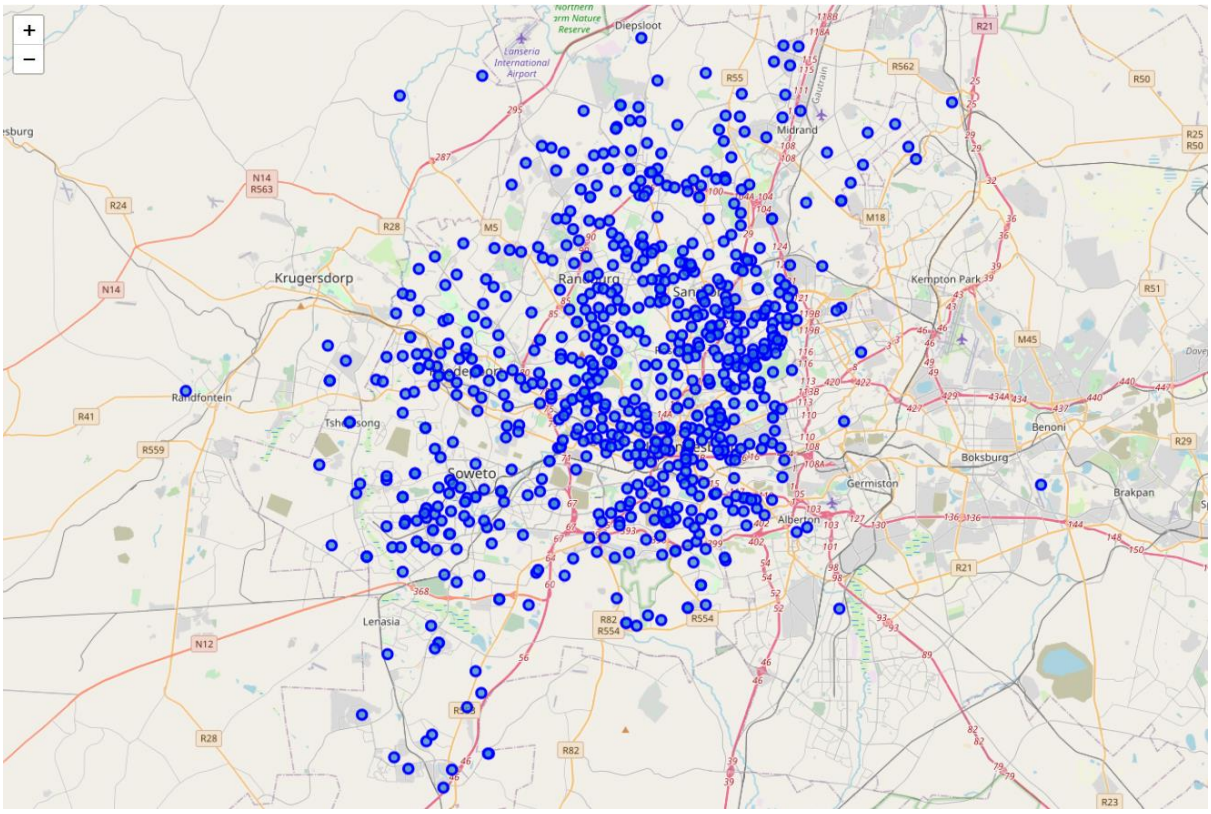
Get Joburg coordinates

```
Jozi_Lat = geocoder.google("Johannesburg, South Africa", key=API_KEY).lat
Jozi_Long = geocoder.google("Johannesburg, South Africa", key=API_KEY).lng

print('The geograpical coordinate of Johannesburg City are {}, {}.'.format(Jozi_Lat, Jozi_Long))
```

```
The geograpical coordinate of Johannesburg City are -26.2041028, 28.0473051.
```

Create map on folium

```
Jozi_Map = folium.Map(location=[Jozi_Lat, Jozi_Long], zoom_start=12)
# adding in the markers for Jozi neighborhoods
for lat, lng, location in zip(
        Jozi_data['Lat'],
        Jozi_data['Long'],
        Jozi_data['Location']):
    label = '{}'.format(location)
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        popup=label,
        color='blue',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=False).add_to(Jozi_Map)
# display map
Jozi_Map
```



First we filter the Joburg municipality and locations data such that all entries that were not matched with a municipality are discarded:

Group data by Municipality: Northcliff and Randburg excluded

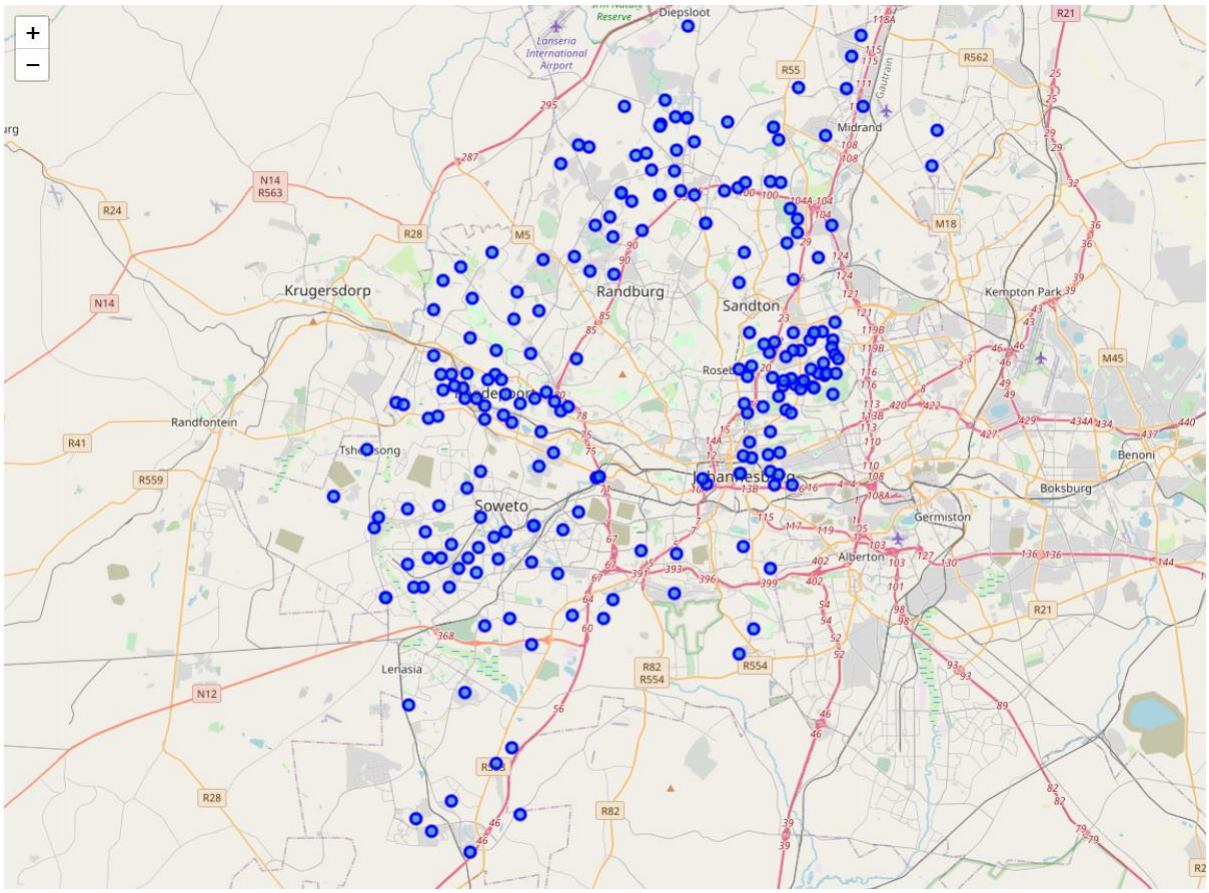
```
MuniLoc_group = Muni_Loc_df.groupby('Region').count()
MuniLoc_group.head()
```

Region	Location	District	State	PostCode	Lat	Long
Diepsloot_Midrand	29	29	29	29	29	29
Ennerdale_OrangeFarm	13	13	13	13	13	13
Fourways	62	62	62	62	62	62
InnerCity	18	18	18	18	18	18
Roodepoort	62	62	62	62	62	62

On visualization the location data is reduced to:

Visualize matched Joburg Municipality Locations

```
Jozi_Map_municipalities = folium.Map(location=[Jozi_Lat, Jozi_Long], zoom_start=12)
# adding in the markers for Toronto neighborhoods
for lat, lng, municipality, location in zip(
        Muni_Loc_df['Lat'],
        Muni_Loc_df['Long'],
        Muni_Loc_df['Region'],
        Muni_Loc_df['Location']):
    label = '{}, {}'.format(location, municipality)
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        popup=label,
        color='blue',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=False).add_to(Jozi_Map_municipalities)

# display map
Jozi_Map_municipalities
```



4. Results

We pick the Midrand municipality as our area of interest and filter all locations that contain Midrand as the municipality into dataframe Midrand_Municipality:

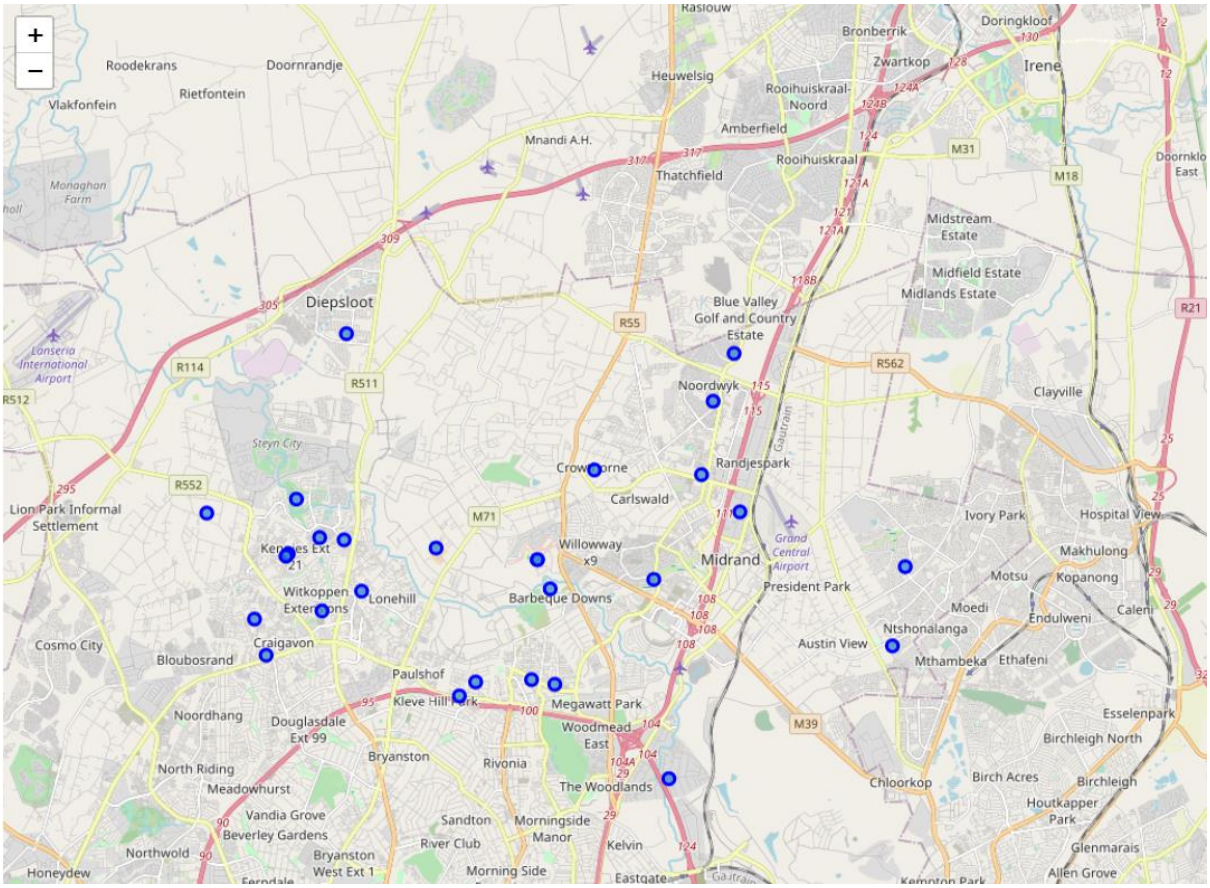
Filter data to only show Midrand municipality data

```
Midrand_Municipality = Muni_Loc_df[Muni_Loc_df['Region'].str.contains("Midrand")].reset_index(drop=True)
Midrand_Municipality.head()
```

	Location	District	State	PostCode	Lat	Long	Region
0	Barbeque Downs	City of Johannesburg	Gauteng	1684	-26.006963	28.072761	Diepsloot_Midrand
1	Beverley	City of Johannesburg	Gauteng	2191	-26.007799	28.017145	Diepsloot_Midrand
2	Brendavere	City of Johannesburg	Gauteng	2021	-26.015073	27.985670	Diepsloot_Midrand
3	Broadacres	City of Johannesburg	Gauteng	2021	-25.994151	28.012097	Diepsloot_Midrand
4	Buccleuch	City of Johannesburg	Gauteng	2090	-26.057261	28.107827	Diepsloot_Midrand

We then visualize all locations within Midrand:

```
Midrand_map = folium.Map(location=[Vorna_Lat, Vorna_Long], zoom_start=12)
# adding in the markers for Toronto neighborhoods
for lat, lng, region, location in zip(
        Midrand_Municipality['Lat'],
        Midrand_Municipality['Long'],
        Midrand_Municipality['Region'],
        Midrand_Municipality['Location']):
    label = '{}, {}'.format(location, region)
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        popup=label,
        color='blue',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=False).add_to(Midrand_map)
# display map
Midrand_map
```



The Foursquare API is used to retrieve all venues within each location in Midrand and the data is stored in Midrand_venues_sorted dataframe:

```
num_top_venues = 10

indicators = ['st', 'nd', 'rd']

# create columns according to number of top venues
columns = ['Location']
for ind in np.arange(num_top_venues):
    try:
        columns.append('{}{} Most Common Venue'.format(ind+1, indicators[ind]))
    except:
        columns.append('{}th Most Common Venue'.format(ind+1))

# create a new dataframe
Midrand_venues_sorted = pd.DataFrame(columns=columns)
Midrand_venues_sorted['Location'] = Midrand_group['Location']

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

Midrand_venues_sorted.head()
```

	Location	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
0	Barbeque Downs	Park	African Restaurant	Pharmacy	Historic Site	Indian Restaurant	Italian Restaurant	Japanese Restaurant	Mexican Restaurant	Office	Optical Shop
1	Beverley	Indian Restaurant	Brazilian Restaurant	Pub	Portuguese Restaurant	Chinese Restaurant	Pet Store	African Restaurant	Pharmacy	Italian Restaurant	Japanese Restaurant
2	Brendavere	Office	African Restaurant	Pharmacy	Historic Site	Indian Restaurant	Italian Restaurant	Japanese Restaurant	Mexican Restaurant	Optical Shop	Other Great Outdoors
3	Broadacres	Shopping Mall	Asian Restaurant	Supermarket	Pharmacy	Pizza Place	Gym / Fitness Center	Restaurant	Seafood Restaurant	Bakery	Steakhouse
4	Buccleuch	Pizza Place	Indian Restaurant	Shopping Mall	Dog Run	Pharmacy	Italian Restaurant	Japanese Restaurant	Mexican Restaurant	Office	Optical Shop

a. Modelling

The segmentation of the locations in Midrand will be conducted using an unsupervised clustering algorithm called k-mean method.

For our cluster modelling on the Midrand data, we use 3 clusters to evaluate each observation within each cluster that falls within the nearest mean (i.e. groups common clusters).

Clustering of Locations

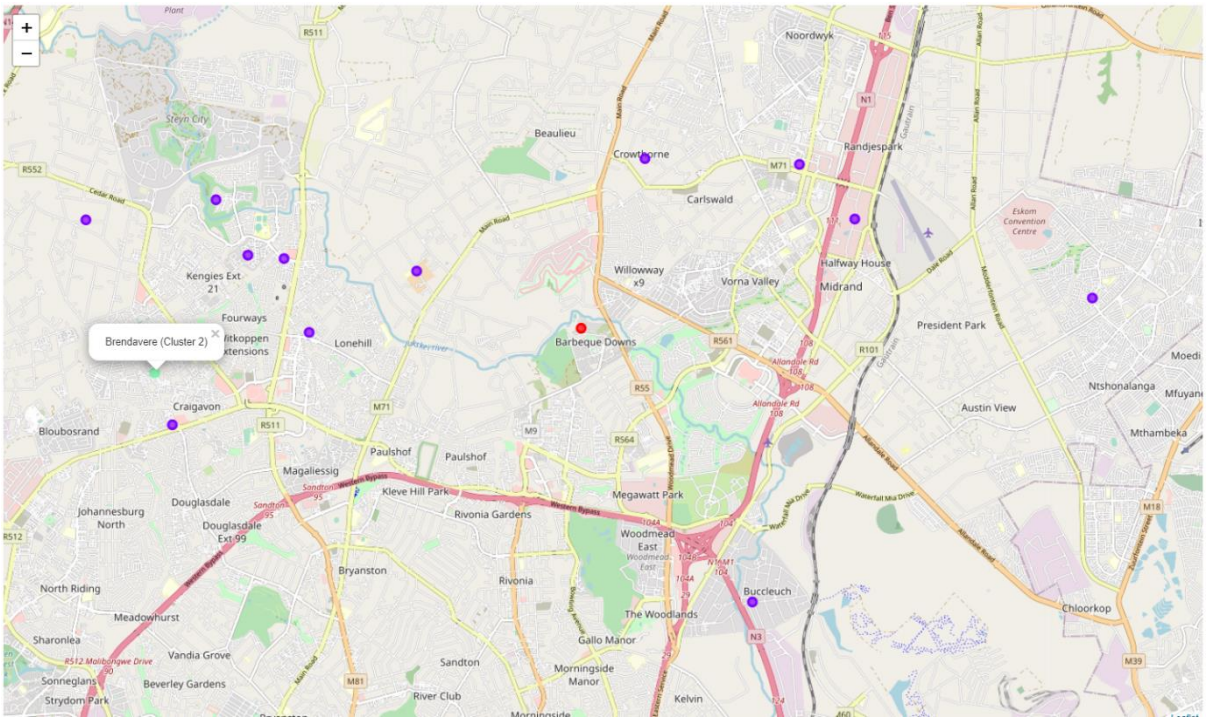
```
# set number of clusters to 3, these will be High, Medium and Low density venue areas
kclusters = 3

Midrand_group_clustering = Midrand_group.drop('Location', 1)

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

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

```
array([0, 1, 2, 1, 1, 1, 1, 1, 1, 1])
```

```
# add clustering labels
Midrand_venues_sorted.insert(0, 'Cluster Labels', kmeans.labels_)

Midrand_merged = Midrand_Municipality

# merge Jozi_grouped with Jozi_df to add latitude/longitude for each Location
Midrand_merged = Midrand_merged.join(Midrand_venues_sorted.set_index('Location'), on='Location')

Midrand_merged.head() # check the last columns!
```

	Location	District	State	PostCode	Lat	Long	Region	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	9th Most Common Venue	10th Mos Commo Venu
0	Barbeque Downs	City of Johannesburg	Gauteng	1684	-26.006963	28.072761	Diepsloot_Midrand	0.0	Park	African Restaurant	Pharmacy	Historic Site	Indian Restaurant	Italian Restaurant	Japanese Restaurant	Mexican Restaurant	Office	Optica Sho
1	Beverley	City of Johannesburg	Gauteng	2191	-26.007799	28.017145	Diepsloot_Midrand	1.0	Indian Restaurant	Brazilian Restaurant	Pub	Portuguese Restaurant	Chinese Restaurant	Pet Store	African Restaurant	Pharmacy	Italian Restaurant	Japanes Restauran
2	Brendavere	City of Johannesburg	Gauteng	2021	-26.015073	27.985670	Diepsloot_Midrand	2.0	Office	African Restaurant	Pharmacy	Historic Site	Indian Restaurant	Italian Restaurant	Japanese Restaurant	Mexican Restaurant	Optical Shop	Othe Grea Outdoor
3	Broadacres	City of Johannesburg	Gauteng	2021	-25.994151	28.012097	Diepsloot_Midrand	1.0	Shopping Mall	Asian Restaurant	Supermarket	Pharmacy	Pizza Place	Gym / Fitness Center	Restaurant	Seafood Restaurant	Bakery	Steakhous

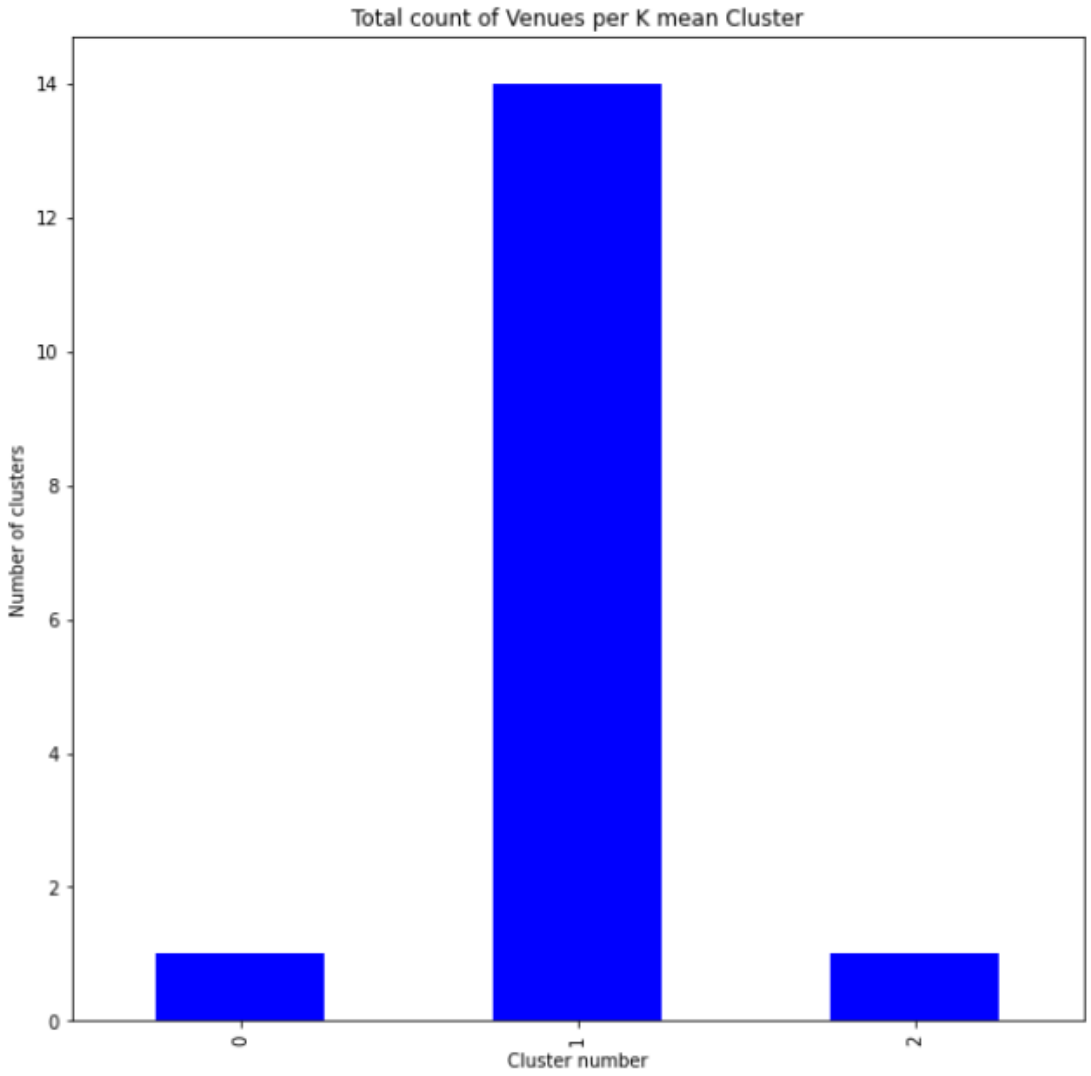
We visualize the clustered data on folium:



b. Evaluation

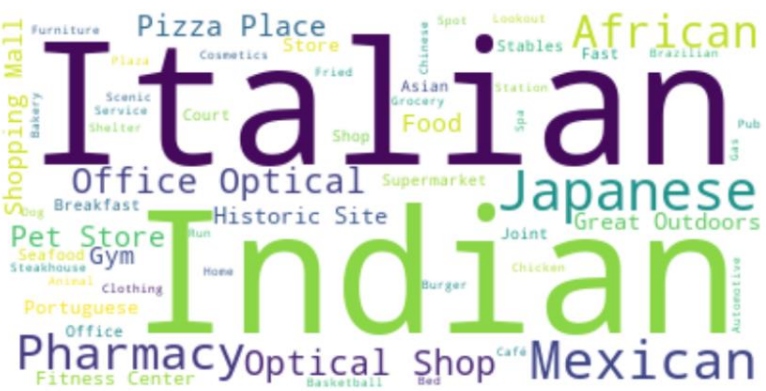
Out of the 3 clusters, the distribution per cluster is shown below:

```
# generate plot
cluster_df['Venues'].plot(kind='bar', figsize=(10,10), color='blue')
plt.xlabel('Cluster number')
plt.ylabel('Number of clusters')
plt.title('Total count of Venues per K mean Cluster')


plt.show()
```



Within cluster 0, the most famous restaurants are African, Indian and Italian

Within cluster 1, the most famous restaurants are Italian, Indian and Mexican



In cluster 2, the most famous restaurants are African, Indian and Italian

5. Discussion and Conclusion

Cluster 1 (purple dots) contains most of the establishments within the Midrand municipality and it is the most saturated cluster. It would not be the best area to establish new market.

Cluster 0 (red dots) and Cluster 2 (greed dots) are unsaturated markets and they would form a good base to deploy an Asian restaurant. Cluster 2, especially Brendavere suburb would be the most suitable location to establish an Asian restaurant as it has a larger distribution of traffic in terms of locality with office buildings, outdoor sites and historic areas.

6. References

[1]: https://www.joburg.org.za/media_/Newsroom/PublishingImages/2020%20News%20Images/March/Office-Johannesburg.jpg
[2]: https://en.wikipedia.org/wiki/Suburbs_of_Johannesburg
[3]: https://www.southafricapostcode.com/location/gauteng/city-of-johannesburg/
[4]: https://www.joburg.org.za/about_/regions/Pages/City-of-Johannesburg-regions.aspx