IBM DS - Capstone - Rent Pricing (1)

July 4, 2020

1 IBM CAPSTONE - Analysing Rent Pricing

1.1 1. Introduction

For this final project, we're going to tackle the infamous problem of Rent Pricing. Knowing that real estate stakeholders need to be well informed about rent pricing so they can take better decisions, either for finding a place to rent, or to put a property for renting. In this whole renting market, the hospitality has found a new branch in which homeowners rent their house for tourists, for a, usually, cheaper price. However, the real estate is an unstable market, varying a lot on seasonality, random happenings, but mostly, to the surroundings of a property and what it can offer in the vicinities. It is important to have a well established method to evaluate the price of a house/apartment and take a decision based on that value.

So, our main question will be:

How does the venues influence the price of an accomodation? Can we predict the price of a home/apartment for renting based on the venues and locations?

This question is of interest for both sides of this transaction. The homeowner, who needs to know the best areas and price to put on their property, and the visitor, who will be looking for a balance o price and location.

Now, hence the importance of location, we must pick at least a city to evaluate these prices. For this project, the city will be Rio de Janeiro in Brasil. Being a city with high contrasts like, being the main touristic route in Brasil and hosting a World Cup, but also with high violence ad poverty rates. It will be interesting to investigate how the prices fluctuate in such conditions.

1.2 2. Data

It is a requirement that the Foursquare API is used, so this is one of the Databases that I will be using, specially to retrieve venues and their location. Another dataset that will be useful is the Airbnb dataset provided by their Inside Airbnb, since it is, currently, the bigest platform for C2C renting market. The dataset retrieved for this project was collected on 24 May, 2020.

1.2.1 2.1 Data Scraping

Importing the referred libraries and collecting the necessary data. Here I applied a Pandas Copy, so I won't need to load the whole dataset again everytime I screw the database.

[ ]:
```
import pandas as pd
import numpy as np


import pandas as pd # library for data analsysis
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

import json # library to handle JSON files

#!conda install -c conda-forge geopy --yes # uncomment this line if you haven't␣
 ↪completed the Foursquare API lab
from geopy.geocoders import Nominatim # convert an address into latitude and␣
 ↪longitude values

import requests # library to handle requests
from pandas.io.json import json_normalize # tranform JSON file into a pandas␣
 ↪dataframe

# Matplotlib and associated plotting modules
import matplotlib.cm as cm
import matplotlib.colors as colors

# import k-means from clustering stage
from sklearn.cluster import KMeans

#!conda install -c conda-forge folium=0.5.0 --yes # uncomment this line if you␣
 ↪haven't completed the Foursquare API lab
import folium # map rendering library

print('Libraries imported.')
```

[ ]:
```
df_listingsAirbnb = pd.read_csv("http://data.insideairbnb.com/brazil/rj/
 ↪rio-de-janeiro/2020-05-24/data/listings.csv.gz",compression='gzip')
df_listingsAirbnb.head()
```

[ ]:
```
df_listings = df_listingsAirbnb.copy()
```

1.2.2 2.2 Knowing and Cleansing the Airbnb Database

As expected, the database has values that does not interest the question we asked in the Introduction.

So the following steps will be taken to make the database more friendly for future EDA and Modelling.

Among the procedures taken are: selecting features, one-hot enconding, removing outliers

[ ]:
```
#knowing our dataset
df_listings.describe()
```

[ ]:
```
#removing columns filled with Nan values, irrelevant data like 'id', or a 75% amount equals to '0'
df_listings = df_listings[['neighbourhood', 'latitude','longitude', 'accommodates', 'bathrooms', 'bedrooms', 'beds', 'square_feet', 'guests_included', 'minimum_nights', 'maximum_nights', 'price']]
df_listings.head()
```

[ ]:
```
df_listings.shape
```

[ ]:
```
# Checking for amount on Nan values in other features. If there is more than 10% of Nan we will drop the column, if not, we will substitute by the mean.
print("number of NaN values for the column bedrooms :", df_listings['bedrooms'].isnull().sum())
print("number of NaN values for the column bathrooms :", df_listings['bathrooms'].isnull().sum())
print("number of NaN values for the column beds :", df_listings['beds'].isnull().sum())
print("number of NaN values for the column square_feet :", df_listings['square_feet'].isnull().sum())
print("number of NaN values for the column guests_included :", df_listings['guests_included'].isnull().sum())
print("number of NaN values for the column minimum_nights :", df_listings['minimum_nights'].isnull().sum())
print("number of NaN values for the column maximum_nights :", df_listings['maximum_nights'].isnull().sum())
print("number of NaN values for the column neighbourhood :", df_listings['neighbourhood'].isnull().sum())
```

[ ]:
```
mean=df_listings['bedrooms'].mean()
df_listings['bedrooms'].replace(np.nan,mean, inplace=True)

mean=df_listings['bathrooms'].mean()
df_listings['bathrooms'].replace(np.nan,mean, inplace=True)

mean=df_listings['beds'].mean()
df_listings['beds'].replace(np.nan,mean, inplace=True)

df_listings.drop(['square_feet'], axis=1, inplace = True)

df_listings.dropna(axis=0, inplace = True)
```

[ ]:
```
df_listings.shape
```

[ ]:
```
# building a function to convert price to float so it can be used as a target
def convert_currency(val):
    """
    Convert the string number value to a float
     - Remove $
     - Remove commas
     - Convert to float type
    """
    new_val = val.replace(',','').replace('$', '')
    return float(new_val)
```

[ ]:
```
df_listings['price'] = df_listings['price'].apply(convert_currency)
```

1.2.3 2.3 Using the Foursquare API

[ ]:
```
address = 'Rio de Janeiro'

geolocator = Nominatim(user_agent="rj_explorer")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinate of Rio de Janeiro are {}, {}.'.
 ↪format(latitude, longitude))
```

[ ]:
```
# Getting Top 500 venues in a 1000 meters radius
LIMIT = 100
radius = 500
url = 'https://api.foursquare.com/v2/venues/explore?
 ↪&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(
    CLIENT_ID,
    CLIENT_SECRET,
    VERSION,
    latitude,
    longitude,
    radius,
    LIMIT)
```

[ ]:
```
results = requests.get(url).json()
results
```

[ ]:
```
# function that extracts the category of the venue
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']
```

[ ]:
```
venues = results['response']['groups'][0]['items']

nearby_venues = json_normalize(venues) # flatten JSON

# filter columns
filtered_columns = ['venue.name', 'venue.categories', 'venue.location.lat',␣
↪'venue.location.lng']
nearby_venues =nearby_venues.loc[:, filtered_columns]

# filter the category for each row
nearby_venues['venue.categories'] = nearby_venues.apply(get_category_type,␣
↪axis=1)

# clean columns
nearby_venues.columns = [col.split(".")[-1] for col in nearby_venues.columns]

nearby_venues.head()
```

[ ]:
```
print('{} venues were returned by Foursquare.'.format(nearby_venues.shape[0]))
```

[ ]:
```
nearby_venues.head()
```

[ ]:
```
#slicing the dataframe so i can run the query with my sand account
df_listingsSample = df_listings.sample(100)
df_listingsSample.shape
```

[ ]:
```
def getNearbyVenues(names, latitudes, longitudes, 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)
```

[ ]:
```
rJ_venues = getNearbyVenues(names=df_listingsSample['neighbourhood'],
                                    latitudes=df_listingsSample['latitude'],
                                    longitudes=df_listingsSample['longitude']
                                   )
```

[ ]:
```
print(rJ_venues.shape)
rJ_venues.head()
```

1.3 3. Methodology

Now that we have our database with both the Airbnb and Foursquare API, we will do through EDA, Modelling and Model Evaluation.

1.3.1 3.1 Exploratory Data Analysis

Importing the necessary libraries. We will use Scatterplots, Heatmaps and Geospatial Visualization, so we can understand better how the features relate to our target *price*.

[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

[ ]:
```
from pandas.plotting import scatter_matrix

continuous_cols =␣
 ↪["bedrooms","bathrooms","beds","guests_included","minimum_nights","maximum_nights",␣
 ↪"price"]
scatter_matrix(df_listings.sample(1000)[continuous_cols], figsize=(15, 10))
plt.show()
```

[ ]:
```
tmp = df_listings.groupby('neighbourhood')['price'].sum().sort_values(ascending␣
 ↪= False)
plt.figure(figsize=(30,6))
_ = tmp.plot(kind='bar')
```

[ ]:
```
plt.figure(figsize=(20,5))
x_data, y_data = (df_listings["neighbourhood"].values, df_listings["price"].
 ↪values)
plt.plot(x_data, y_data, 'ro')
plt.ylabel('Price')
plt.xlabel('Neighbourhood')
plt.xticks(rotation=90)
plt.show()
```

1.4 3.2 Modelling and Evaluating

Since this a regression problem and we are dealing with a distribution that, taken the outliers, could be linear, we will try a polynomial regression.

1.4.1 3.2.1 Regression

Since we are trying to predict price, this is a Regression problem. We will use the Statsmodel package. Its a very extensive package that allow us to implement different types of regression, displays the features by weight, treats categorical features and prints a series of evaluations.

[ ]:
```
import statsmodels.formula.api as smf
import sklearn, matplotlib, statsmodels
```

[ ]:
```
results = smf.ols('price ~ accommodates + np.square(bedrooms) + np.
 ↪square(bathrooms) + np.square(beds) + guests_included + minimum_nights +␣
 ↪maximum_nights + C(neighbourhood)', data=df_listings).fit()
results.summary()
```

1.4.2 3.2.1 Clustering

Our model didn't perform so well with regression. Lets try clustering so we can better visualize the inflence of location in a rent price.

[ ]:
```
# one hot encoding
rJ_onehot = pd.get_dummies(rJ_venues[['Venue Category']], prefix="",␣
 ↪prefix_sep="")

# add neighborhood column back to dataframe
rJ_onehot['Neighborhood'] = rJ_venues['Neighborhood']

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

rJ_onehot.head()
```

[ ]:
```
rJ_grouped = rJ_onehot.groupby('Neighborhood').mean().reset_index()
rJ_grouped.head()
```

[ ]:
```
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]
```

[ ]:
```
num_top_venues = 10

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

# create columns according to number of top venues
columns = ['Neighborhood']
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
neighborhoods_venues_sorted = pd.DataFrame(columns=columns)
neighborhoods_venues_sorted['Neighborhood'] = rJ_grouped['Neighborhood']

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

neighborhoods_venues_sorted.head()
```

[ ]:
```
# Seting number of clusters
kclusters = 5
```

```
rJ_grouped_clustering = rJ_grouped.drop('Neighborhood', 1)

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

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

[ ]:
```
# add clustering labels
neighborhoods_venues_sorted.insert(0, 'Cluster Labels', kmeans.labels_)

rJ_merged = df_listings

# merge toronto_grouped with toronto_data to add latitude/longitude for each␣
 ↪neighborhood
rJ_merged = rJ_merged.join(neighborhoods_venues_sorted.
 ↪set_index('Neighborhood'), on='neighbourhood')

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

[ ]:
```
rJ_merged.dropna(axis = 0, inplace = True)
```

[ ]:
```
# create map
map_clusters = folium.Map(location=[latitude, longitude], zoom_start=11)

# 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(rJ_merged['latitude'],␣
 ↪rJ_merged['longitude'], rJ_merged['neighbourhood'], rJ_merged['Cluster␣
 ↪Labels']):
    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(map_clusters)

map_clusters
```

[ ]:
```
rJ_merged.loc[rJ_merged['Cluster Labels'] == 0, rJ_merged.columns[[1] +␣
↪list(range(5, rJ_merged.shape[1]))]]
```

[ ]:
```
rJ_merged.loc[rJ_merged['Cluster Labels'] == 1, rJ_merged.columns[[1] +␣
↪list(range(5, rJ_merged.shape[1]))]]
```

1.5 Results

After analysing the results in EDA and the Models, it became clear that we can use location as an important feature to predict rent pricing, since it represents a significant weight in price variation, whilst other variables have an influence, but are not as sensible.

For Modeling, it's better to implement more complex models. Since the relation between location and price is not linear, and requires more versatile models.

1.6 Discussion

It can be recommended for owners to try and rent properties in costal areas, as they are better valued and can generate more revenue.

For tourists, if they don't have a problem with price, than the costal area is the most recommended, as it has a higher concentration of venues. However, if one is trying to save, it can still find properties close to costal areas but are not as expensive.

1.7 Conclusion

Afer analysing these datas. It is clear that we can predict rent price using location and venues. The following model is a first attemp and can be better improvef through more Feature Engineering and trying to implement other models that can better access these problem.

[ ]:
```

```