**Airbnb Insights**

For this final project of our course we have worked on Airbnb dataset to get some insights of it, which would eventually help the user who wants to host their property on Airbnb by making the price decisions easy to get maximum profit out of it using machine learning.

Our dataset contains 3 files, listing, review and calendar. Listing file contains all the property listed in the Boston area and its details like description, exact location, price, amenities, number of beds, different charges, cancellation policy. Review file contains the review given by for all the listing property. And the calendar file contains the change in price of listing according to date.

Our goal for this project is to help the hosting user, which could be achieved by first helping them decide where he should invest in Boston to get good returns And to the one who already has a property, what price they should put to attract customers.

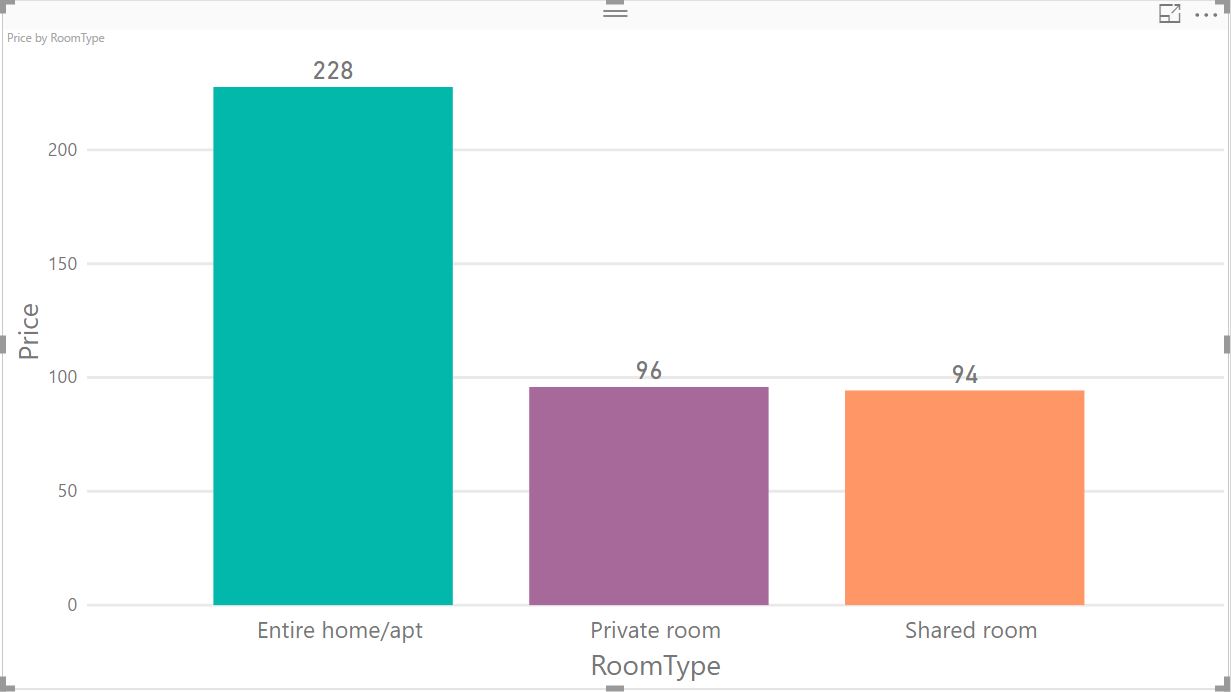
The first task was to do some data cleaning so that we minimize any absurd and unexpected output for this project. We handled some missing data, so that it would be helpful during any kind of computations. We handled data in different language, as you can see in there is this review written in Spanish, which can create problems when we want to perform sentiment analysis over it. And at last we had to convert this string values of amenities to integer, which taking numeric value can be helpful to us when we want to perform machine learning computations on it.

The first analysis we are going to demonstrate you is been generated with pig and Mapreduce. The reason why we have selected pig is to that when we want to perform some join functionalities at that time pig can outperform Mapreduce in terms of easiness. This analysis will be helpful to one who is planning to buy a property just for the sake of investment and want to generate some income from this new property.

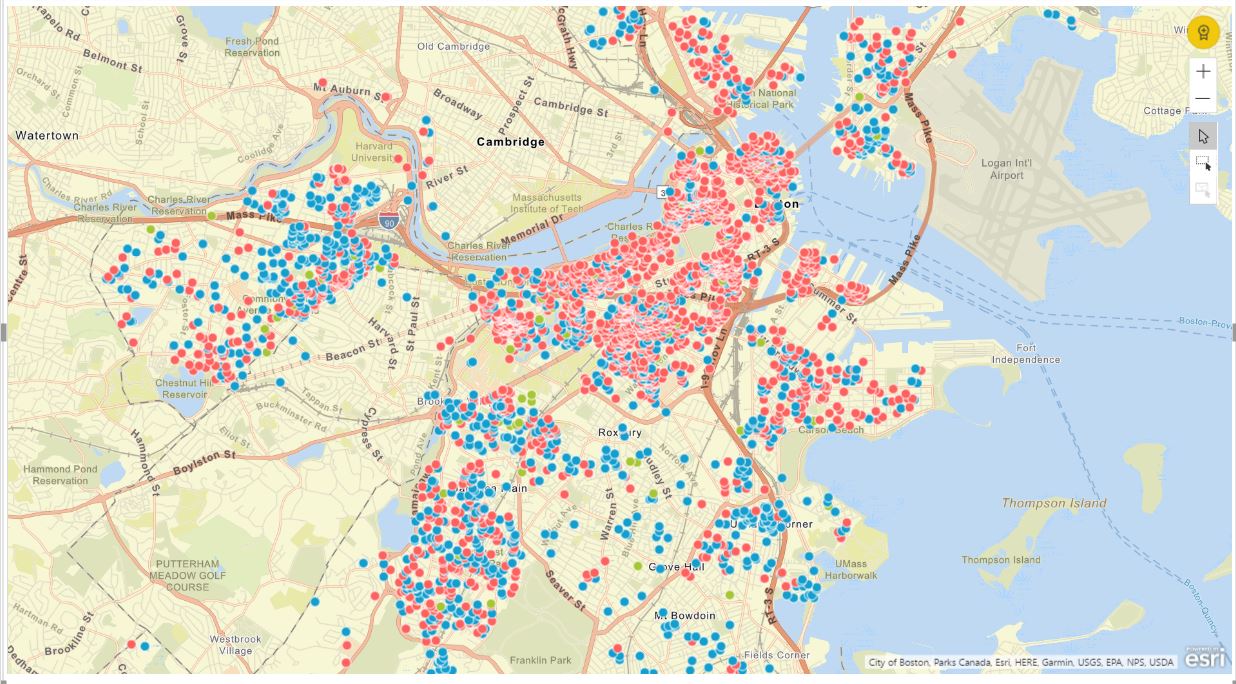
**Analysis**

Which area of Boston is most Profitable for hosting a property?

To backup the profitability, First we need to check which Room Type gets good return.

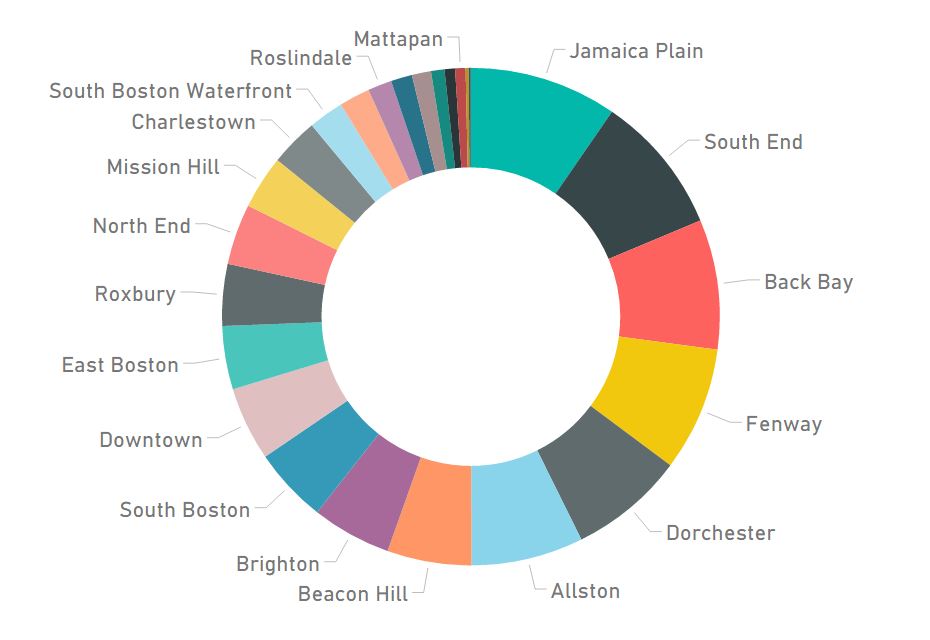
****

Once we have decided which type of room to buy, then we need the select in which general area of Boston this has to be

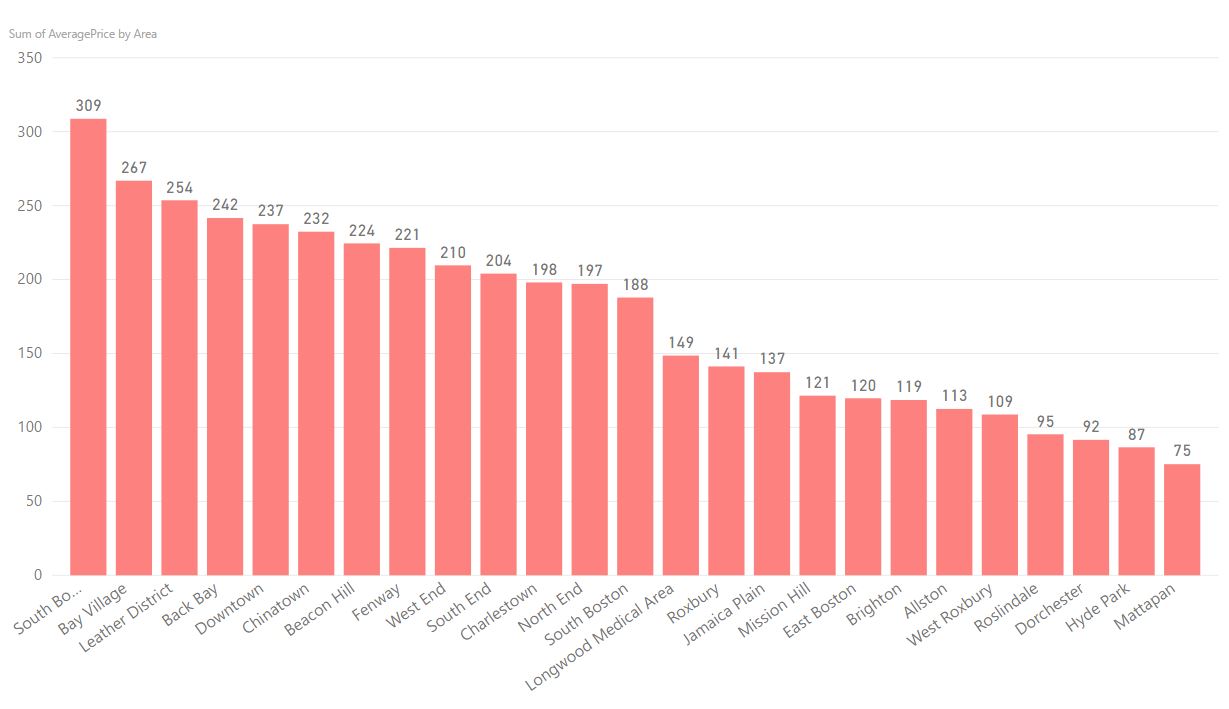


* Red Spot is Entire Home/Apt
* Blue Spot is Private Room
* Green Spot is Shared Room

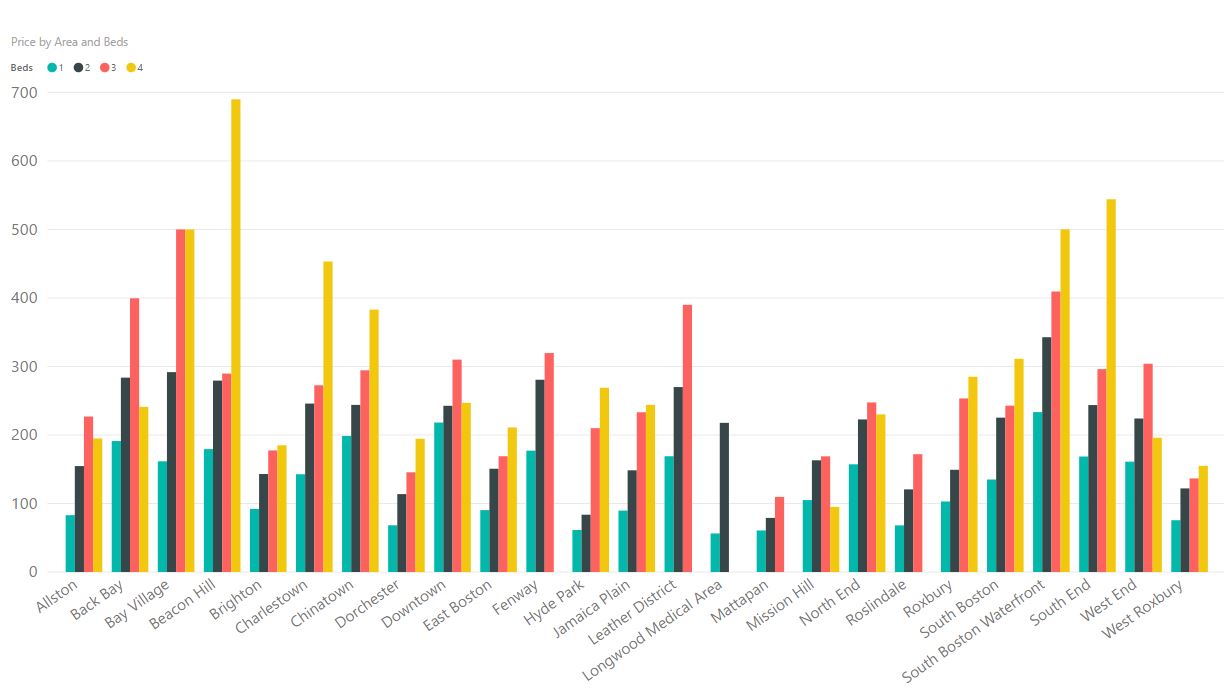
Now we need to have a look which area has maximum listings



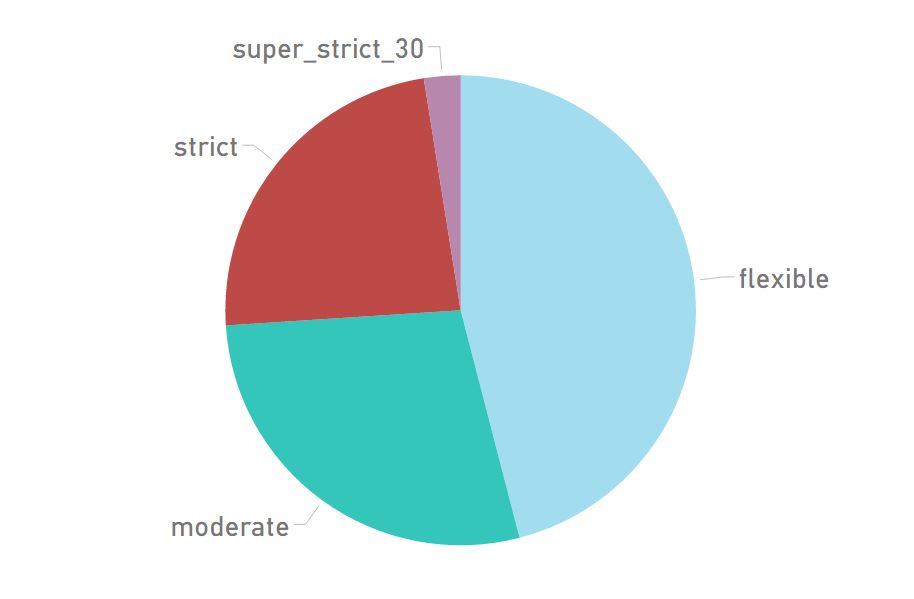
Area Vs Price



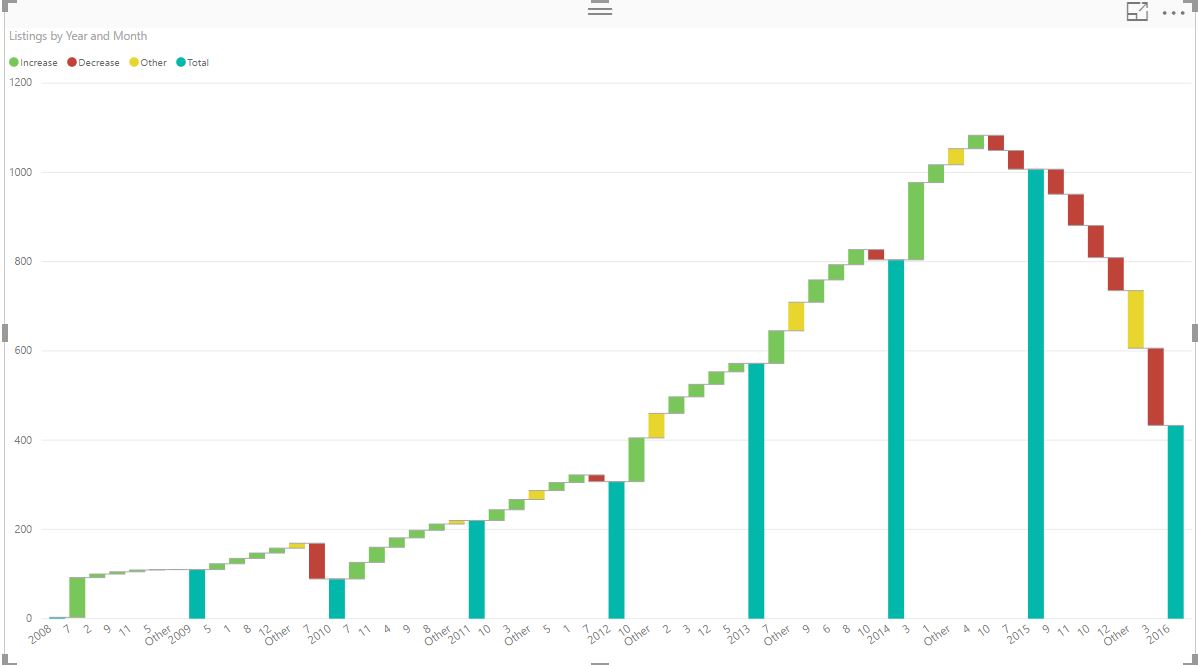
After the location is finalized, We can check how many beds listing could be in our favour



Cancellation Policy Vs Booking



Performance of Airbnb Until Now in Boston



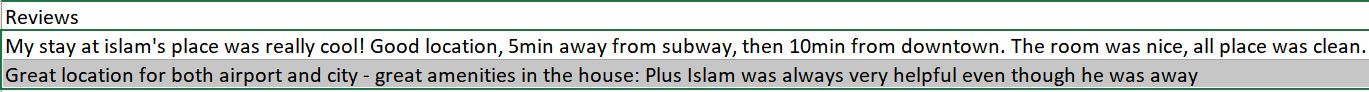
**Sentimental Analysis**

It is a way to evaluate written or spoken language to determine if the expression is favourable, unfavourable, or neutral, and to what degree.

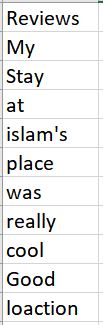
**Process**

Extracted the review column from the data

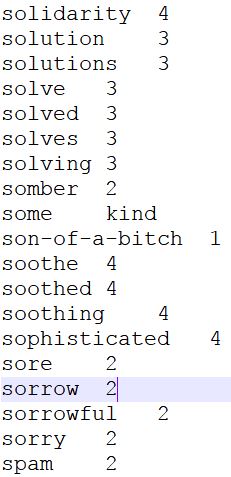
Split the data into arrays

****

Tokenized all the arrays

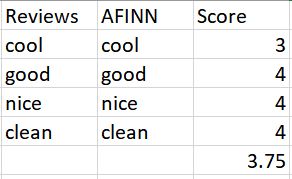
****

AFINN file



Inner join on our data and AFINN file

Perform Group by operation and calculate average rating of every review.



**Complications**

* The scale of AFINN file usually ranges from -5 to +5 so there were discrepancies in the resultant score
* Input file given to Mahout should have the score in the range of 1 to 5
* Modified the AFINN file to change it’s range to 1 to 5 using MapReduce

**Modifying the AFINN file:**

* Modified the scale of the AFINN file from (-5 to +5) to (1 to 5) to remove discrepancy.



**Sentimental Analysis IN Hive**

**Step 1: Creating Table( reviews\_table ):-**

create table reviews\_table(listing\_id BIGINT,review\_id BIGINT,review\_date DATE,reviewer\_id BIGINT,reviewer\_name STRING,comments STRING) row format delimited fields terminated by ‘,’;

**Step 2: Loading Data in reviews\_table Table:-**

LOAD DATA LOCAL INPATH '/home/shubham/Desktop/FinalData/Reviews.csv' OVERWRITE INTO TABLE reviews\_table;

**Step 3: Creating a new table with comments data being split and saved in an array form**

create table comments\_split as select listing\_id as listing\_id,review\_id as review\_id,review\_date as review\_date,reviewer\_id as reviewer\_id,reviewer\_name as reviewer\_name,split(comments,' ') as words from reviews\_table;

**Step 4: Creating a new table with comments Column data being tokenized**

create table review\_word as select listing\_id as listing\_id,review\_id as review\_id,review\_date as review\_date,reviewer\_id as reviewer\_id,reviewer\_name as reviewer\_name,word from comments\_split LATERAL VIEW explode(words) w as word;

**Step 5: Creating a table for dictionary**

create table dictionary(word string,rating int) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

**Step 6: Loading data into the dictionary table**

LOAD DATA LOCAL INPATH '/home/shubham/Desktop/FinalData/AFINN.txt' into TABLE dictionary;

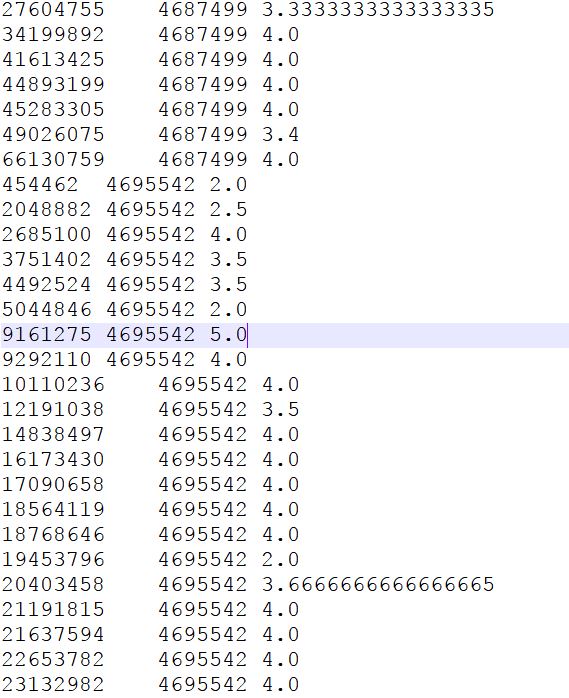
**Step 7: Creating a table by joing values from both the tables**

create table word\_join as select review\_word.listing\_id, review\_word.review\_id, review\_word.review\_date, review\_word.reviewer\_id, review\_word.reviewer\_name, review\_word.word,dictionary.rating from review\_word LEFT OUTER JOIN dictionary ON(review\_word.word =dictionary.word);

**Step 8:Write a select query to get the rating for each review**

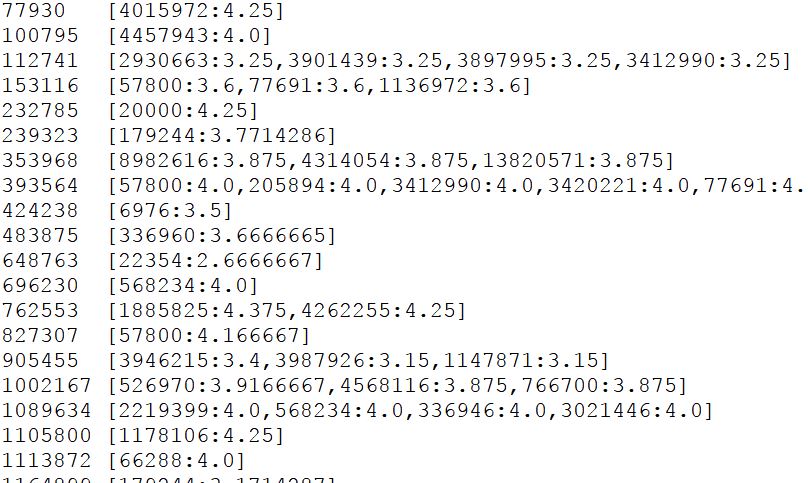
select reviewer\_id as reviewer\_id, listing\_id as listing\_id, AVG(rating) as rating from word\_join GROUP BY word\_join.listing\_id,word\_join.reviewer\_id order by rating DESC LIMIT 5000;

**Output**



**Recommendation**

* Performed recommendation using Mahout
* Output of sentimental analysis is used as input for recommendation



**Prediction:**

The price of a listing depends on multiple parameters from the neighbourhood area to the square feet area of an apartment. Here we have attempted to predict the price of a listing based on the data available. Also we studied the influence of different parameters on the price. The code is written in Python:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.grid\_search import GridSearchCV

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import r2\_score

listings = pd.read\_csv("./listings.csv")

listings = listings.iloc[:,12:]

rev = listings.columns

#print(rev)

temp = []

for r in rev:

if "review" in r:

temp.append(r)

temp.remove('number\_of\_reviews')

temp = ['price\_log','city', 'state' ,'zipcode', 'country', 'has\_availability', 'review\_scores\_checkin', 'review\_scores\_communication', 'review\_scores\_location', 'review\_scores\_value', 'market', "monthly\_price", "security\_deposit", "weekly\_price", "cleaning\_fee", "calendar\_updated", "calendar\_last\_scraped", "first\_review", "last\_review"] + temp

#print(temp)

listings.drop(temp, axis = 1, inplace=True)

listings['price'] = listings['price'].apply(lambda x: x.replace(",", ""))

listings['price'] = np.log(pd.to\_numeric(listings['price']))

listings['extra\_people'] = pd.to\_numeric(listings['extra\_people'])

listings['latitude'] = pd.to\_numeric(listings['latitude'])

listings['longitude'] = pd.to\_numeric(listings['longitude'])

listings = listings.fillna(listings.mean())

X = listings.drop('price', axis = 1)

Y = listings['price']

listings.head()

xx = ['property\_type', 'room\_type', "bed\_type",

"cancellation\_policy","is\_location\_exact"]

X = pd.get\_dummies(X, columns = xx)

#print(xx)

#print(X)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.33, random\_state = 42)

rf = RandomForestRegressor(oob\_score = True, n\_estimators = 1000, max\_depth = 30, min\_samples\_split = 15, max\_features = "sqrt")

print("\n Fitting Random Forest model...")

fit = rf.fit(X\_train, Y\_train)

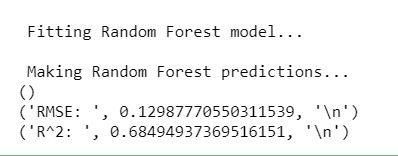
print("\n Making Random Forest predictions...")

predictions = rf.predict(X\_test)

print()

print("RMSE: ", mean\_squared\_error(Y\_test, predictions), "\n")

print("R^2: ", r2\_score(Y\_test, predictions), "\n")



feature\_names = X\_train.columns

importances = rf.feature\_importances\_

feats = {}

figComp= plt.figure(figsize=(50,20))

for feature, importance in zip(feature\_names, importances):

feats[feature] = importance #add the name/value pair

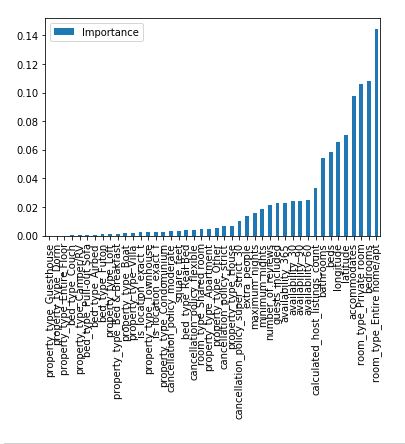
feature\_importance\_df = pd.DataFrame.from\_dict(feats, orient = 'index').rename(columns = {0: 'Importance'})

feature\_importance\_df = feature\_importance\_df.sort\_values(by = 'Importance')

feature\_importance\_df.plot(kind = 'bar')

plt.savefig("feature\_importance.png")

plt.show()



**Comparision of the models:**

* Linear Regression
* Decision Tree Regression
* Random Forests Regression

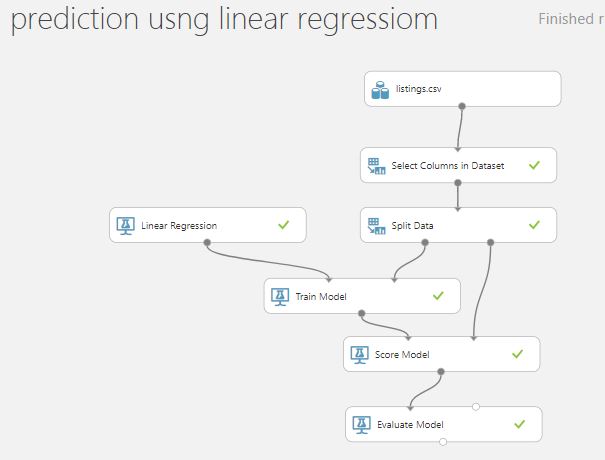
Studied the different models using Microsoft Azure Machine Learning studio

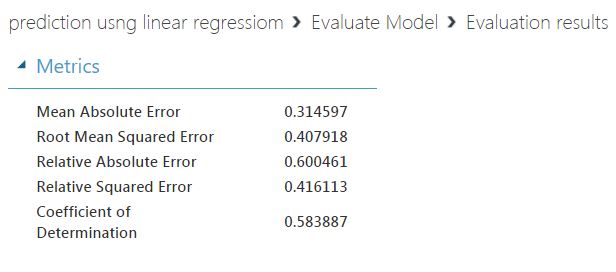
1. Linear Regression

Following is the model that we created using Machine learning studio. And the R-squared value of the model is 0.58 as we can see in the screenshot.

Reason for low efficiency:

The data here is non-linear.



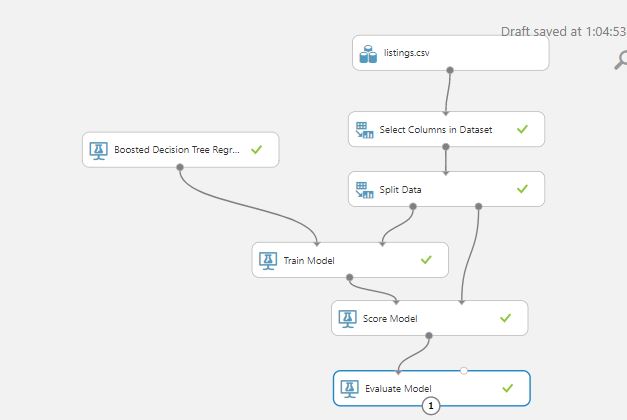


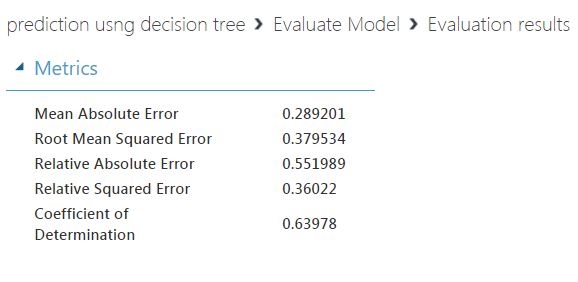
1. Decision Tree Regression

Following is the model that we created using Machine learning studio. And the R-squared value of the model is 0.63 as we can see in the screenshot.

Reason for low efficiency:

Over-fitting





1. Random Forest algorithm

Following is the model that we created using Machine learning studio. And the R-squared value of the model is 0.68 as we can see in the screenshot. The performance is best amongst the 3 algorithms because it overcomes the disadvantages of the other 2 algorithms.

