Programming for Data Science

BUAN 6340.002

Yingjie Zhang

Group Project Final Report

Predictive Models for NYC Airbnb

Group 7

Daniel Liao

Kashish Vermani

Gowtham Konda

Pradeep Munikrishnan

TABLE OF CONTENTS

➢ INTRODUCTION

Data science is a "concept to unify statistics, data analysis, machine learning and their related methods" in order to "understand and analyse actual phenomena" with data. It employs techniques and theories drawn from many fields within the context of mathematics, statistics, computer science, and information science. Turing award winner Jim Gray imagined data science as a "fourth paradigm" of science (empirical, theoretical, computational and now data-driven) and asserted that "everything about science is changing because of the impact of information technology" and the data deluge. In 2015, the American Statistical Association identified database management, statistics and machine learning, and distributed and parallel systems as the three emerging foundational professional communities.

➢ PROJECT DESCRIPTION

Airbnb is an online marketplace for arranging or offering lodging, primarily homestays, or tourism experiences. The company does not own any of the real estate listings, nor does it host events; it acts as a broker, receiving commissions from each booking. The company is based in San Francisco, California, United States. It is a platform that connects people who want to rent out their homes with people who are looking for accommodations in that locale. It currently covers more than 81,000 cities and 191 countries worldwide. The company's name comes from "air mattress Bed & Breakfast."

Context: Since 2008, guests and hosts have used Airbnb to expand on traveling possibilities and present more unique, personalized way of experiencing the world. This dataset describes the listing activity and metrics in NYC, NY for 2019.

Content: The data file includes all needed information to find out more about hosts, geographical availability, necessary metrics to make predictions and draw conclusions.

Acknowledgment: This public dataset is part of Airbnb, and the original source can be found on this website , from which we need to collect the data.

Objective: The Objective is to create a best model to predict the rental prices of the Airbnb listings throughout New York State & also create clusters based on all the variables to create a data structure and segment the data.

Planning ,execution & implementation: Here we will be using various machine learning algorithms to predict the best model for rental prices & segmentation then we will select the best suited one based on the metrics for implementation to achieve the goal of the project.

➢ PROCESS DESCRIPTION

1) Data Acquisition: The process of collecting the data from the source.

2) Data Exploration: Data exploration is an approach which uses visual exploration to understand the dataset, rather than through traditional data management systems.

3) Exploratory Data Analysis: Primarily EDA is for seeing what the data can tell us beyond the formal modelling or hypothesis testing task.

4) Data Wrangling: Data wrangling is the process of cleaning (skewness, outliers, duplicates) and unifying messy and complex data sets for easy access and analysis. It is also known as Data pre-processing and manipulation.

5) Data Partitioning: The process of splitting the data into train, validation and test for building a model.

6) Data Modelling: Modelling is the statistical technique that is chosen to find trends, patterns, and relationships within existing data, staged in the first phase of the data wrangling process.

7) Model Selection: Based on various metrics the best model will be chosen for implementation.

DATA ACQUISITION

- The data can be downloaded from the Kaggle website which has been hosted by Airbnb
- The data dictionary of the dataset which contains the detailed explanation of the various features present in it, also needs to be explored.
- The dimensions of the dataset are **48,895 x 16**.

Data Dictionary:

1. id: listing ID
2. name: name of the listing
3. host_id: host ID
4. host_name: name of the host
5. neighbourhood_group: location
6. neighbourhood: area
7. latitude: latitude coordinates
8. longitude: longitude coordinates
9. room_type: listing space type
10. price: price in dollars
11. minimum_nights: amount of nights minimum
12. number_of_reviews: number of reviews
13. last_review: latest review date
14. reviews_per_month: number of reviews per month
15. calculated_host_listings_count: amount of listing per host
16. availability_365: number of days when listing is available for booking

DATA EXPLORATION

- After acquiring the dataset from the source, we can explore it to understand the various features present in the dataset and its characteristics.
- In python, there is a special package to do this all exploration of data in to a single step known as pandas-profiling.
- The **pandas-profiling Python package** is a great tool to create HTML profiling reports. For a given dataset, it computes the following statistics:
    - Essentials: type, unique values, missing values.
    - Quantile statistics like minimum value, Q1, median, Q3, maximum, range, interquartile range.
    - Descriptive statistics like mean, mode, standard deviation, sum, median absolute deviation, coefficient of variation, kurtosis, skewness.
    - Most frequent values.
    - Histogram.

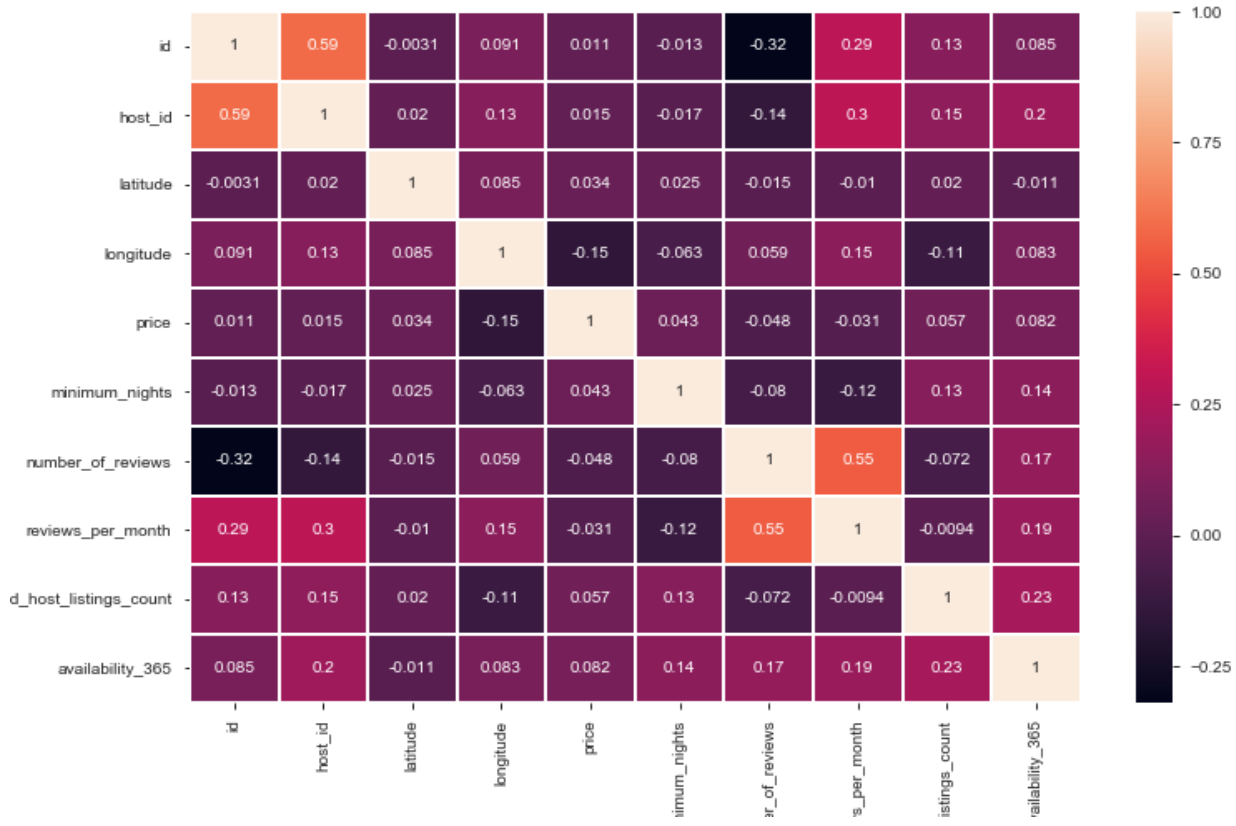
- Correlations highlighting of highly correlated variables, Spearman and Pearson matrixes.

```
In [3]: import pandas_profiling# pip install pandas-profiling
        display(df.profile_report(style={'full_width':True}))
```

EXPLORATORY DATA ANALYSIS

- At first, we will try to find the strength of the correlation between target variable and all the other variables by using the heat map.

```
In [5]: plt.figure(figsize=(12,8))
        fig= sns.heatmap(df.corr(),linecolor='white',linewidths=1,annot=True)
        plt.show()
```



- Now, we will check the insignificant features with respect to the target variable based on p-values by using as user-defined function.

	Pearson-Coeff	P-Value	P-Ind	C-Ind
id	-0.006696	1.870667e-01	Insignificant	Negative
host_id	0.006263	2.172186e-01	Insignificant	Positive
latitude	0.031344	6.529758e-10	Strong	Positive
longitude	-0.155298	4.228450e-208	Strong	Negative
minimum_nights	0.025501	5.030048e-07	Strong	Positive
number_of_reviews	-0.035924	1.438829e-12	Strong	Negative
reviews_per_month	-0.030623	1.591501e-09	Strong	Negative
calculated_host_listings_count	0.052895	1.831175e-25	Strong	Positive
availability_365	0.078276	8.034109e-54	Strong	Positive

From the above results, we can see that the **id and host_id** have insignificant relationship with the target variable **price**.

- Now, we will check whether any multi-correlation is present between the independent variables in the dataset by using user-defined functions.

```
# Function of multi-Correlation
def multicorrelation(data, threshold=0.80):
    col_corr = set()  # Set of all the names of deleted columns
    corr_matrix = data.corr()

    print('  \n    Correlation with more than :',threshold)
    print('\n\nCorr Value',"\t",'Fearture1', "\t\t\t\t", 'Feature 2')
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if (abs(corr_matrix.iloc[i, j]) >= threshold) and (corr_matrix.columns[j] not in col_corr):
                print(f'{corr_matrix.iloc[i, j].round(5):<{16}} {corr_matrix.columns[i]:{39}}
                      {corr_matrix.columns[j]:{40}}')

    print('\n\n The above mentioned correlations only are present in the given Dataset')
```

```
In [77]: multicorrelation(df,threshold=0.5)
```

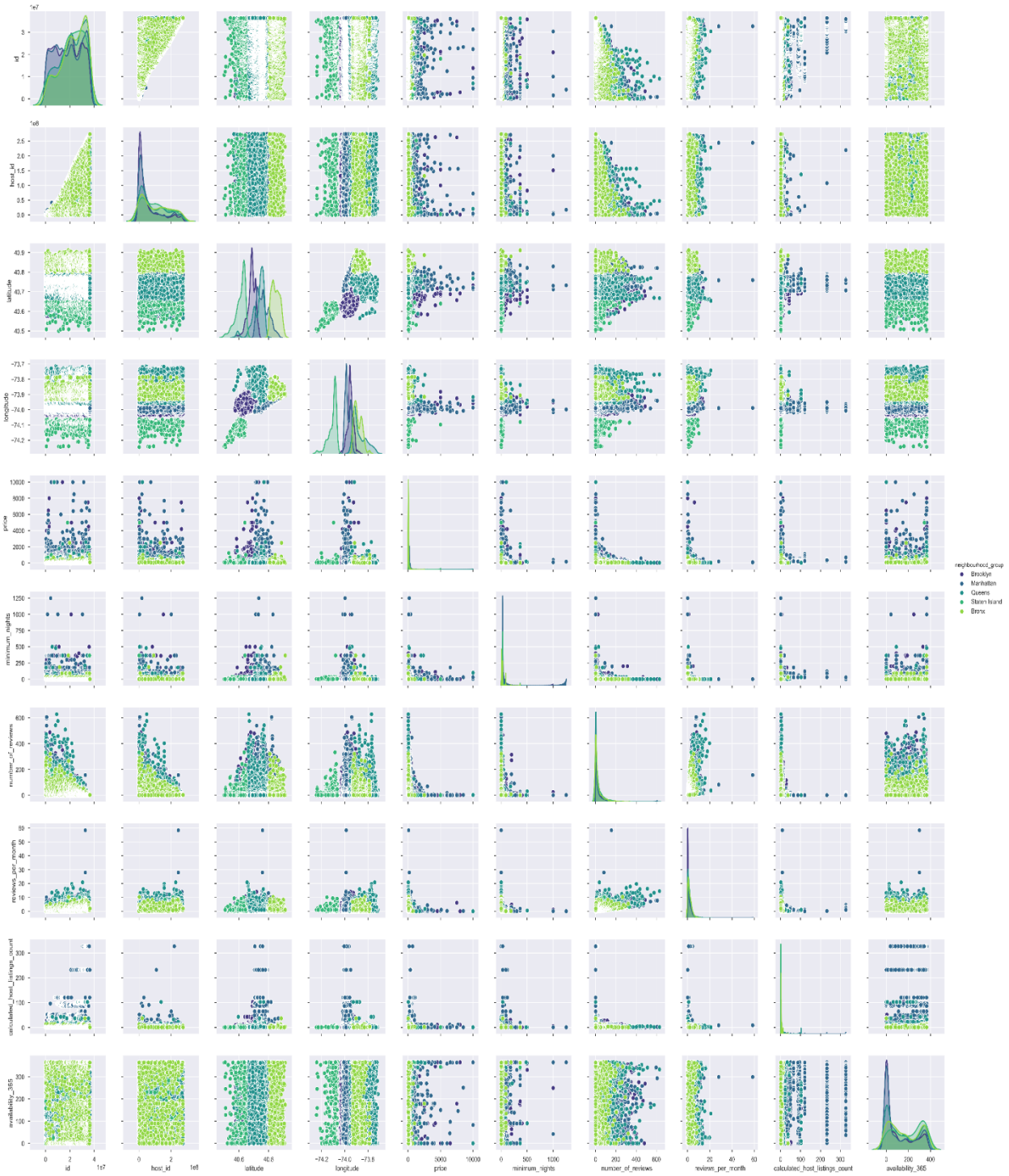
```
    Correlation with more than : 0.5


Corr Value      Fearture1                       Feature 2
0.58829         host_id                         id
0.54987         reviews_per_month               number_of_reviews

 The above mentioned correlations only are present in the given Dataset
```

The correlation within independent variables are less than 0.7, hence we don't remove them.

- Pairplot will be used to find, whether any variable can be able to differentiate the price distribution. Here we will be using the 'neighbourhood_group' feature.

```
In [48]: plt.figure(dpi=500)
         fig=sns.pairplot(df,hue='neighbourhood_group',palette='viridis')
         plt.show()
         fig.savefig('pair.png',dpi=400)
```

Pairplot:


id
host_id
latitude
longitude
price
minimum_nights
number_of_reviews
reviews_per_month
calculated_host_listings_count
availability_365
neighbourhood_group
Brooklyn
Manhattan
Queens
Staten Island
Bronx

- Now, we will explore about the ‘neighbourhood_group’ and ‘neighbourhood’ variables and its explanation.

- The New York City encompasses Five county-level administrative divisions called `"Boroughs"`:
  1. Bronx
  2. Brooklyn
  3. Manhattan
  4. Queens
  5. Staten Island

Each borough is coterminous with a respective county of New York State.

In [241]:
```
from IPython.display import Image
display(Image("5 boroughs.gif"))
print('\n• The number of unique neighbourhood_group in dataset is :\n\n',df['neighbourhood_group'].value_counts())
print('\n\n• The Number of unique neighbourhood is : ',df['neighbourhood'].nunique(),'\n')
one=pd.DataFrame(df[['neighbourhood_group','neighbourhood']].groupby(['neighbourhood_group']).nunique())
two=pd.DataFrame(df[['neighbourhood_group','neighbourhood']].groupby(['neighbourhood_group']).count())
two.rename(columns={'neighbourhood':'n2'},inplace=True)
three= pd.concat([one,two],axis=1)
three.drop('neighbourhood_group',axis=1,inplace=True)
three.rename(columns={'neighbourhood':'Unique_neighborhood','n2':'Total number of Airbnb Listings'},inplace=True)
display(three)
print('\t     Total\t\t',np.sum(48+47+32+51+43),'   \t\t',np.sum(1091+20104+21661+5666+373))
```

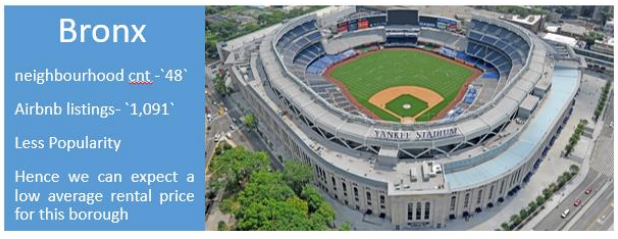
- The number of unique neighbourhood_group in dataset is :

```
 Manhattan        21661
Brooklyn         20104
Queens            5666
Bronx             1091
Staten Island      373
Name: neighbourhood_group, dtype: int64
```

- The Number of unique neighbourhood is : 221

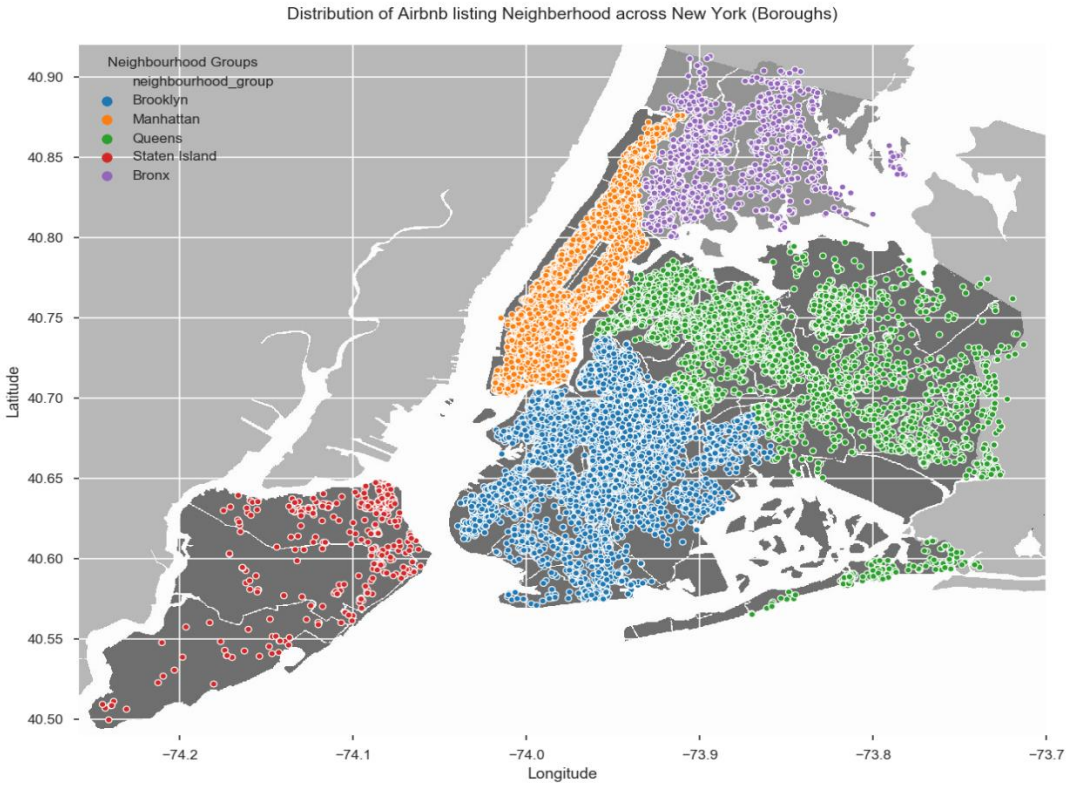
neighbourhood_group	Unique_neighborhood	Total number of Airbnb Listings
Bronx	48	1091
Brooklyn	47	20104
Manhattan	32	21661
Queens	51	5666
Staten Island	43	373
Total	221	48895

INSIGHTS



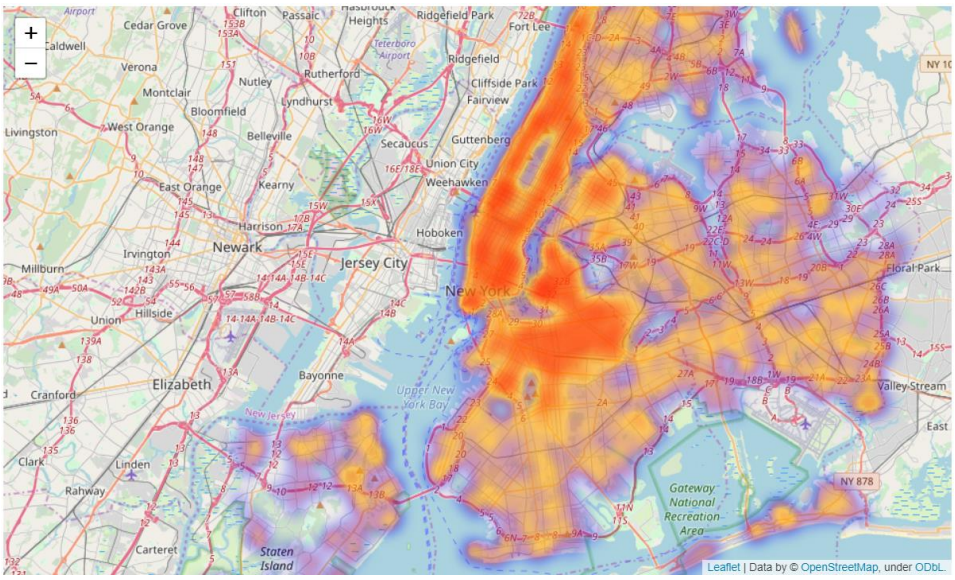
- Let's Visualize the number of Airbnb listings on New York map to know the density of all the listing in each 'neighbourhood_group'(boroughs)

In [9]:
```
fig, ax = plt.subplots(figsize=(12,12),dpi =85)
img=plt.imread('New_York_City_.png', 0)# importing the Background image
coordenates_to_extent = [-74.258, -73.7, 40.49, 40.92]
ax.imshow(img, zorder=0, extent=coordenates_to_extent)
nymap = sns.scatterplot(x='longitude', y='latitude', hue='neighbourhood_group',s=20, ax=ax, data=df)
nymap.set(xlabel='Longitude', ylabel='Latitude', title='\n\nDistribution of Airbnb listing Neighberhood across New York (Boro
ax.grid(True)
plt.legend(title='Neighbourhood Groups')
plt.show()
grp = nymap.get_figure()
grp.savefig('ny_grp.png',dpi=120)
```



As still the number of points plotted looks cluttered in the above plot we will be using the heatmap from the folium package to illustrate the density in each boroughs

In [3]:
```
import folium
import folium.plugins
map = folium.Map([40.7128,-74.0060],zoom_start=11)
folium.plugins.HeatMap(df[['latitude','longitude']].dropna(),radius=8,
                       gradient={0.2:'blue',0.4:'purple',0.6:'orange',1.0:'red'}).add_to(map)
display(map)
plt.show()
```

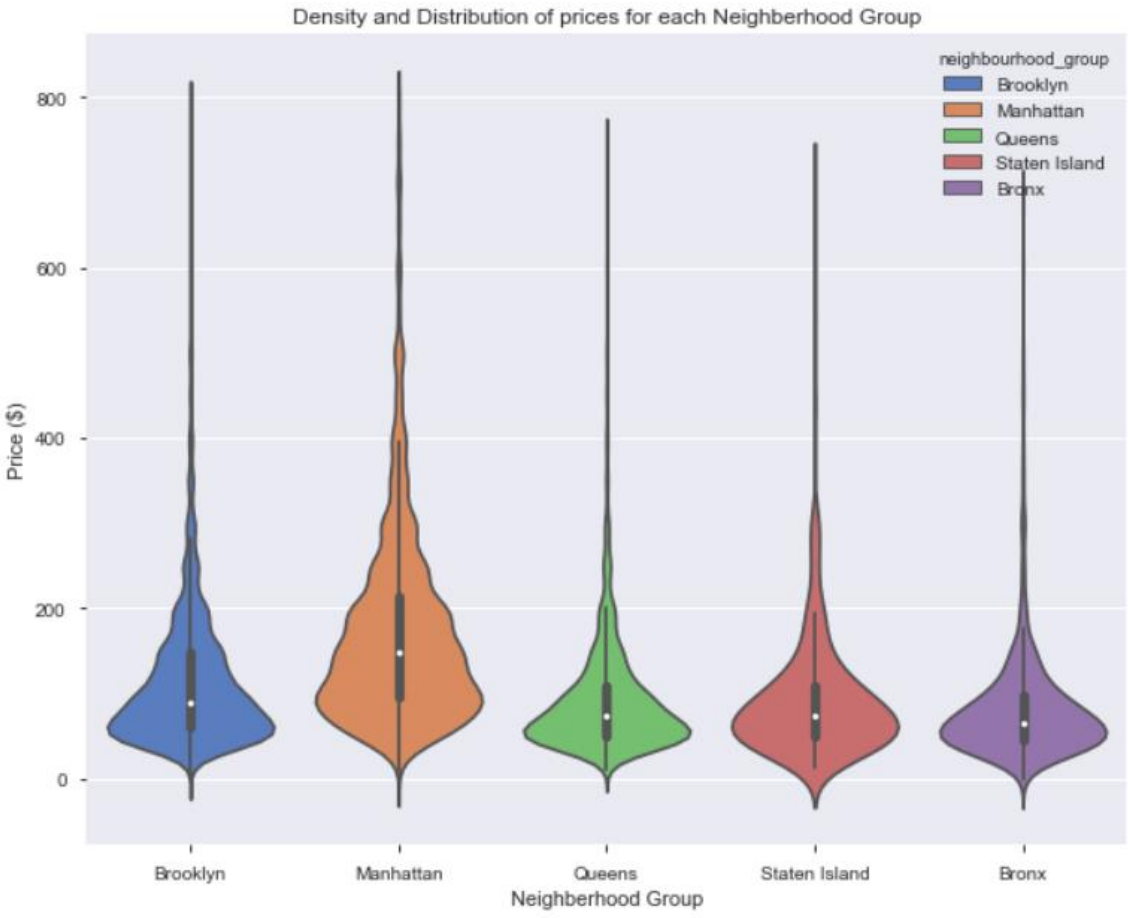


- Let's visualize the Price Distribution of Airbnb listings based on neighborhood_group(boroughs) to verify the above insight by using violin plot

```
plt.figure(figsize=(10,8))
vplot=sns.violinplot( x="neighbourhood_group", y="price",hue="neighbourhood_group", data=df[df['price'] < 800],
                     palette="muted", dodge=False)
vplot.set(xlabel='Neighberhood Group', ylabel='Price ($)',
          title='Density and Distribution of prices for each Neighberhood Group')
plt.show()
avgprice= pd.DataFrame((df[['neighbourhood_group','price']].groupby(['neighbourhood_group']).mean()))
avgprice.rename(columns={'price':'Average-Price'},inplace=True)
print('\n\nThe Average price of each Boroughs :')
display(avgprice)
```

The Average price of each Boroughs :

neighbourhood_group	Average-Price
Bronx	87.496792
Brooklyn	124.383207
Manhattan	196.875814
Queens	99.517649
Staten Island	114.812332



And each `neighbourhood_group`(boroughs) has been drilled down into fine locations as `neighbourhood` whose count is 221. Among the neighbourhood groups:

1. `'Queens'` borough have highest number of unique neighbourhood locations(`51`) but less number of Airbnb listings(`5666` - rental places),which indicates that the Queens is not much popular site therefore less tourists and less rental spaces.Hence we can expect a low average rental price for this borough

2. `'Manhattan'` have least number of unique neighbourhood locations(`32`) but have the maximum number of Airbnb listings(highest-`21,661`) in the entire state, which means that this is a popular site, thus more travelers and tourists and thus the large number of rental places. Therefore we can expect that the average rental price to be higher than the other boroughs

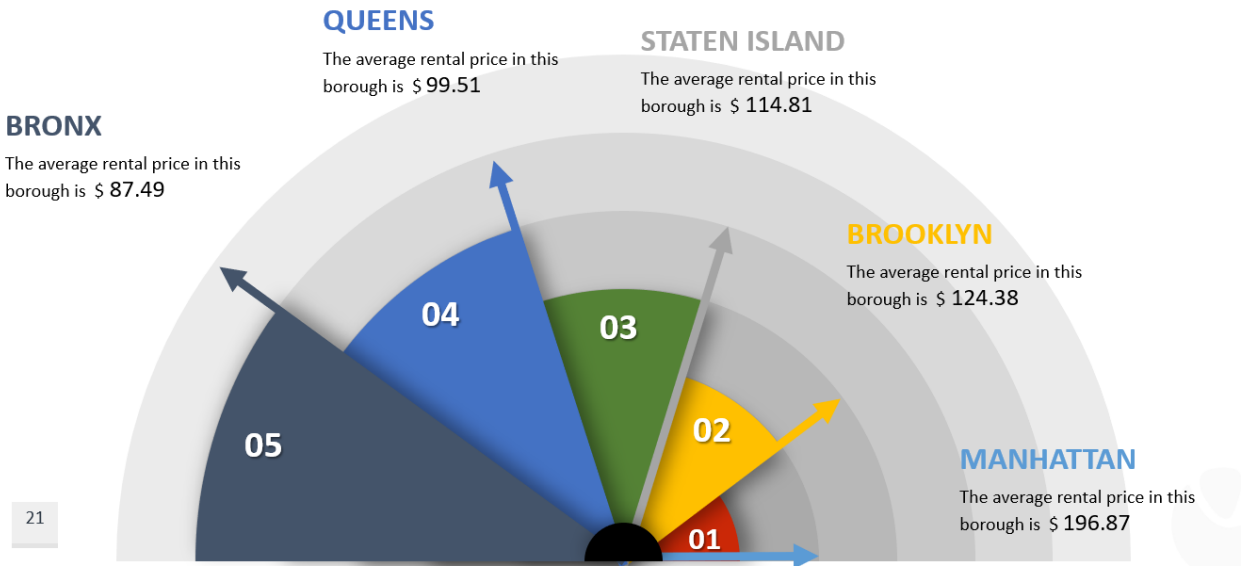
3. `Brooklyn` have the third highest number of unique neighbourhood locations(`47`) and also have the maximum number of Airbnb listings(2nd highest-`20,104`) in the entire state , which means that this site is also as popular as Manhattan. Therefore we can expect that the average rental price to be higher than the other boroughs but as same as Manhattan borough

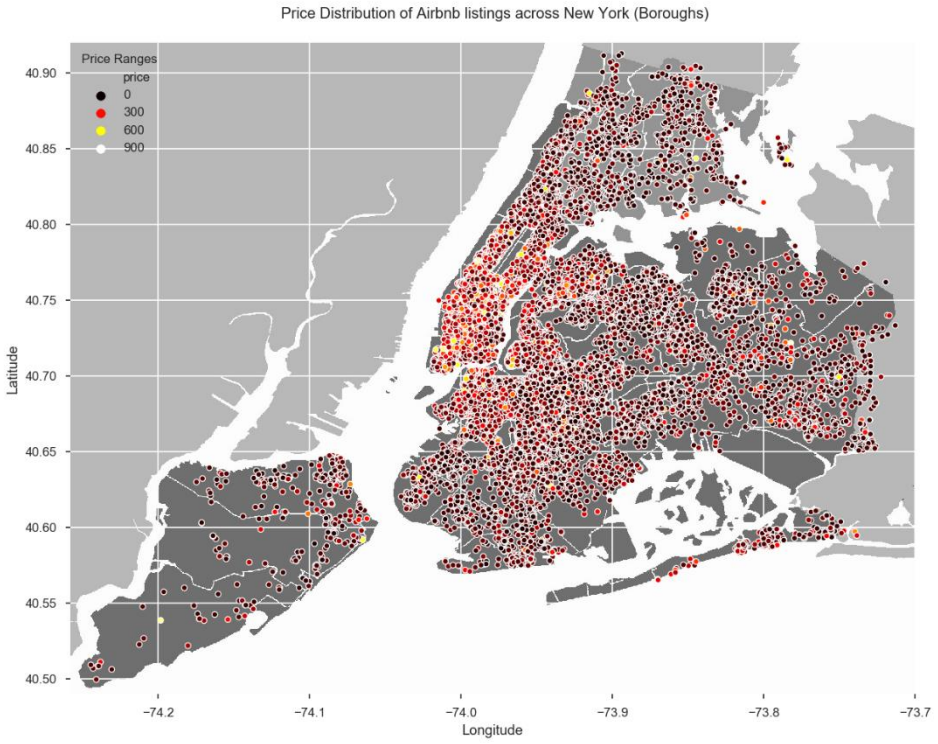
4. `Bronx` have the second highest number of unique neighbourhood locations(`48`) but have the less number of Airbnb listings(`1,091`) , which means that this site is less popular. Hence we can expect a low average rental price for this borough

5. `Staten Island` the fourth highest number of unique neighbourhood locations(`43`) but have very very less number of Airbnb listings(`373`) , which means that this site is not popular at all. Hence we can expect a low average rental price for this borough

We can see that the number of Airbnb Listings in each borough is what influences the price and there are more number of Airbnb listing in Manhattan and Brooklyn, lets visualize the same below one by one using scatterplot & Heatmap

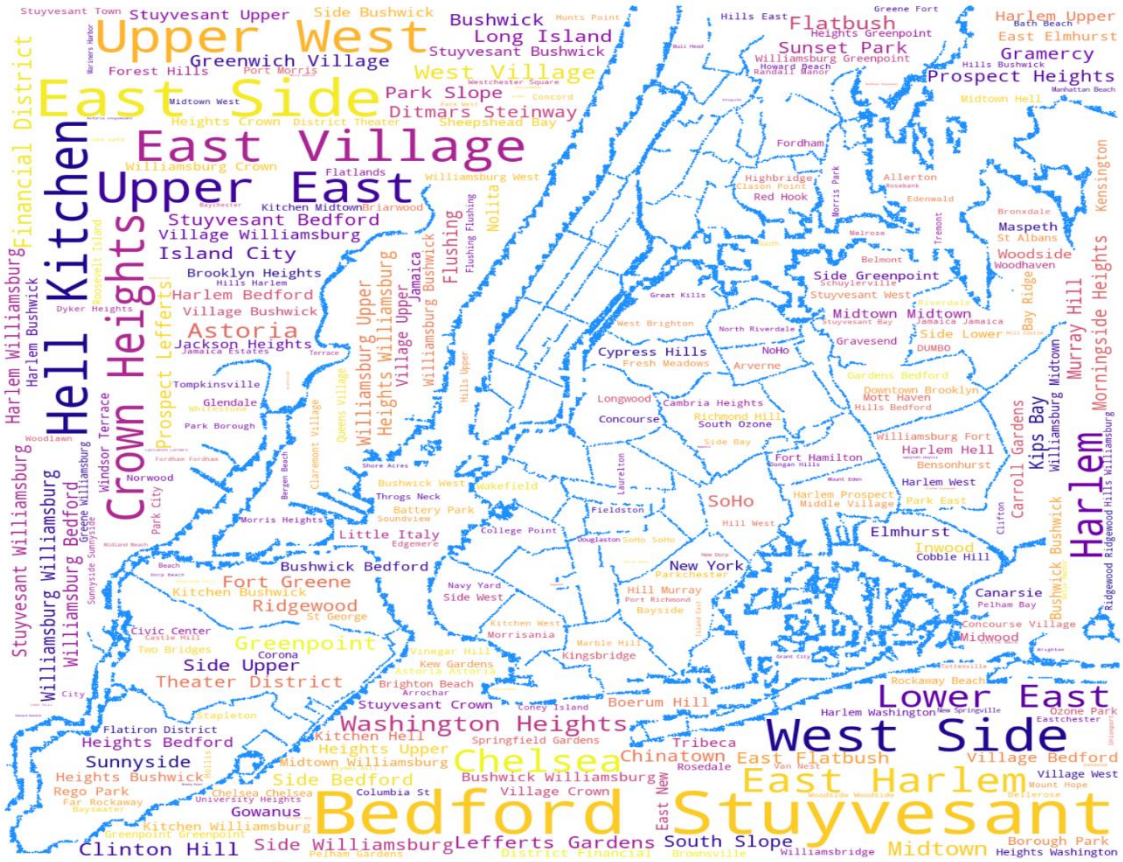
RANKING OF BOROUGHS





- Among all the neighbourhoods (Airbnb listings-221), which neighbourhood is the most popular ones(frequency of names in the dataset) by using Word cloud
- if we simply use the value_counts() over the dataset we will be getting 221 columns in bar charts of crunching numbers
- Hence, we can visualize the same in wordcloud, as the better form of representing the data.
- Here we will be plotting the word cloud on the map of New York itself!

```
import wordcloud
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from PIL import Image
# Generate a word cloud image
wc = " ".join([neighbourhood for neighbourhood in df["neighbourhood"].dropna()])
stopwords = set(STOPWORDS) #setting the stop words
mask = np.array(Image.open("New_York_City_.png"))
wordcloud_por = WordCloud(stopwords=stopwords, background_color="white", max_words=1000,
                          mask=mask,colormap="plasma",
                          contour_width=3, contour_color='dodgerblue').generate(wc)
plt.figure(figsize=(10,10),dpi=200)
fig=plt.imshow(wordcloud_por, interpolation="bilinear")
plt.axis("off")
plt.show()
grp = fig.get_figure()
grp.savefig('wcloud.png',dpi=200)
```

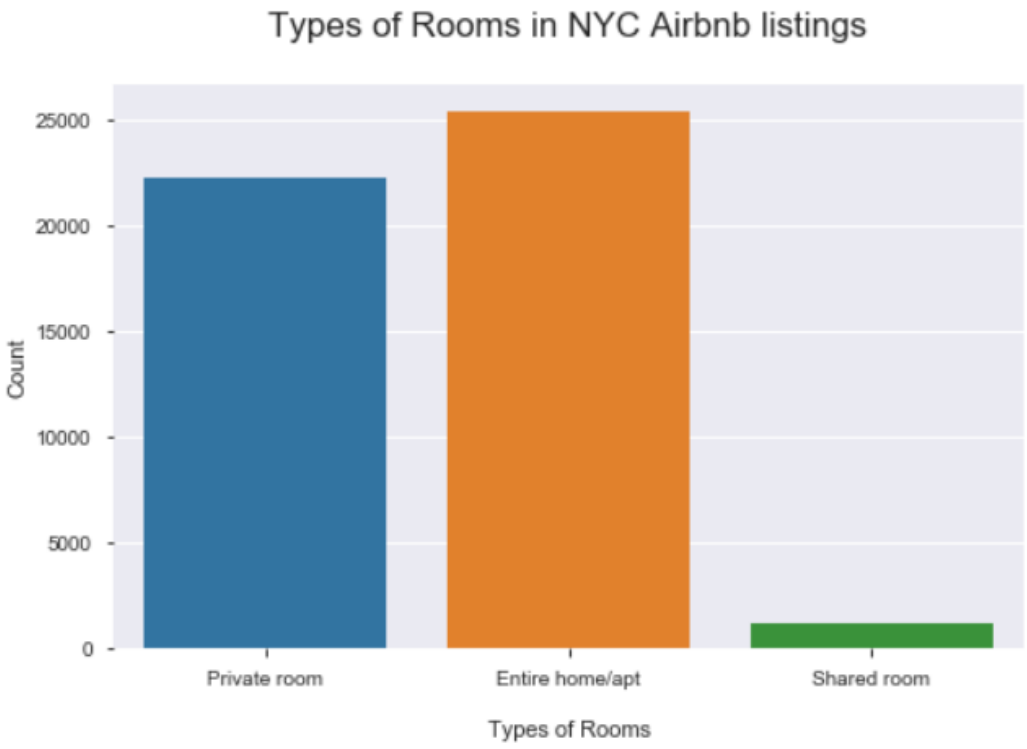


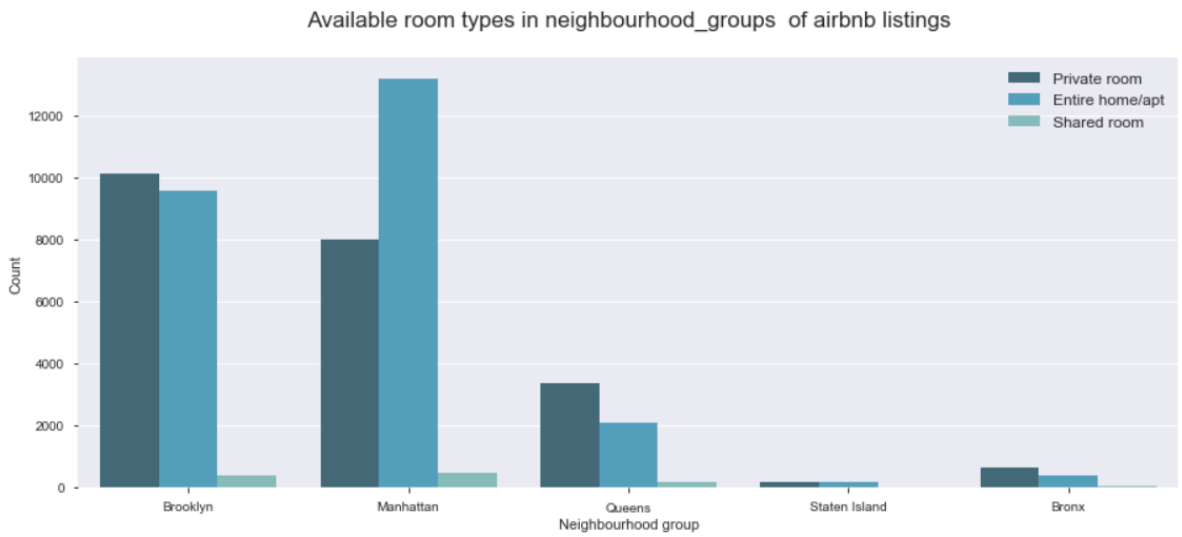
Let's analyse the other categorical features:

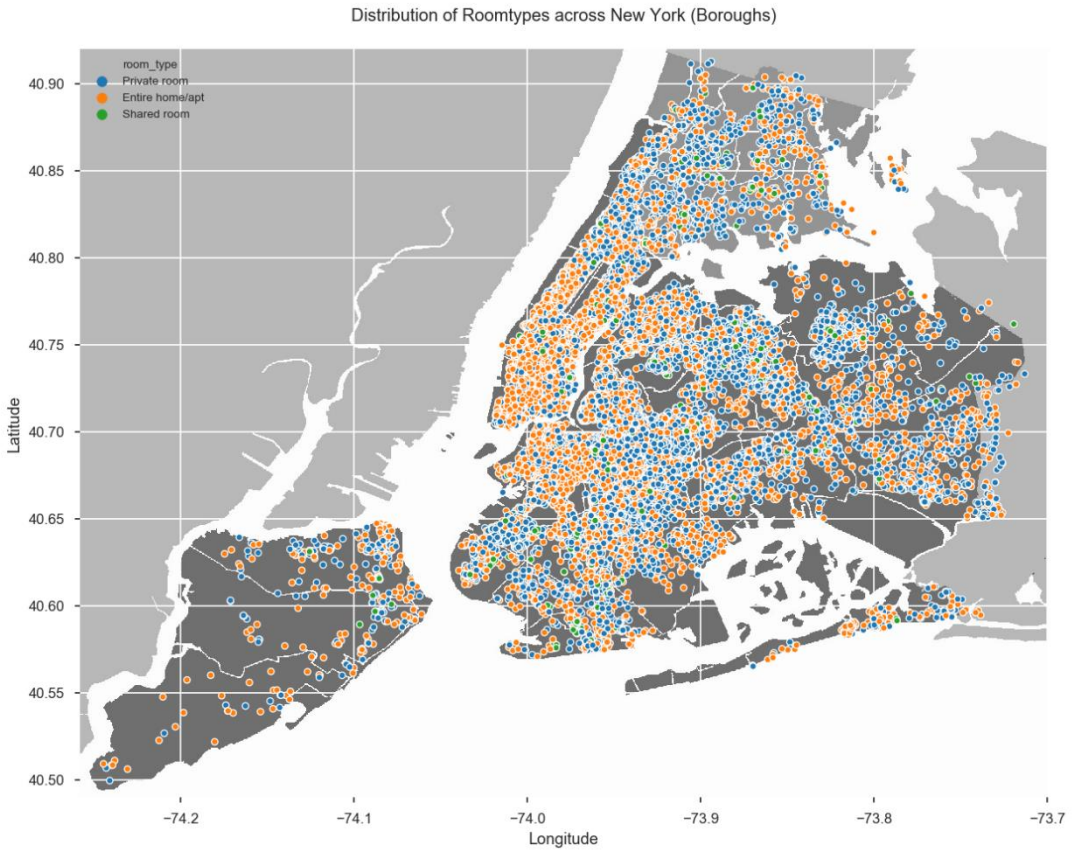
There are 3 types of Rooms available in the above-mentioned neighbourhoods (221)

1. Entire home/apt
2. Private room
3. Shared room

let's visualize the dataset and verify how many rooms are there in each type of rooms and how much rooms are available in each type in each neighbourhood_group (boroughs)







From the above spatial analysis we can see that the `'Manhattan'` has more number of `'Entire home/apt'` (orange) compared with `'Private rooms'` and the `'Brooklyn'` has more number of `'Private Rooms'` (blue) than `'Entire home/apt'`

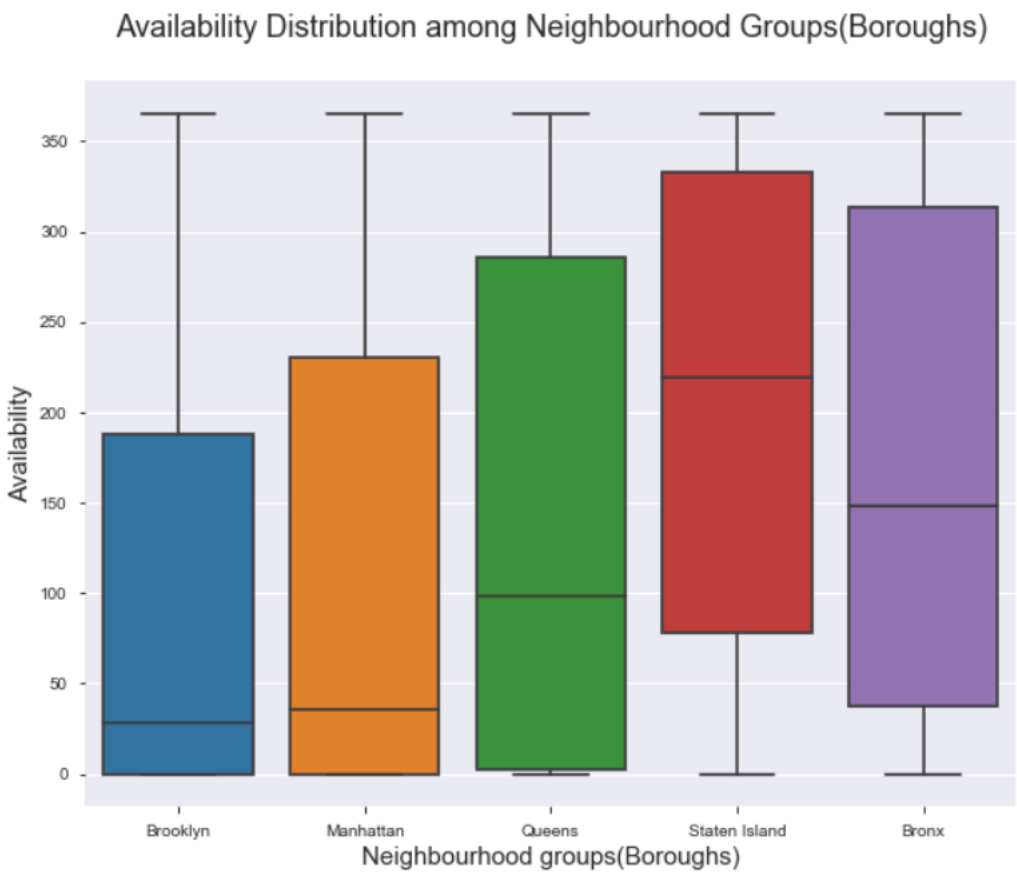
Distribution of room types in each neighbourhood_group (boroughs)

```
In [60]: display(df[['neighbourhood_group','room_type','id']].groupby(['neighbourhood_group','room_type']).count())
```

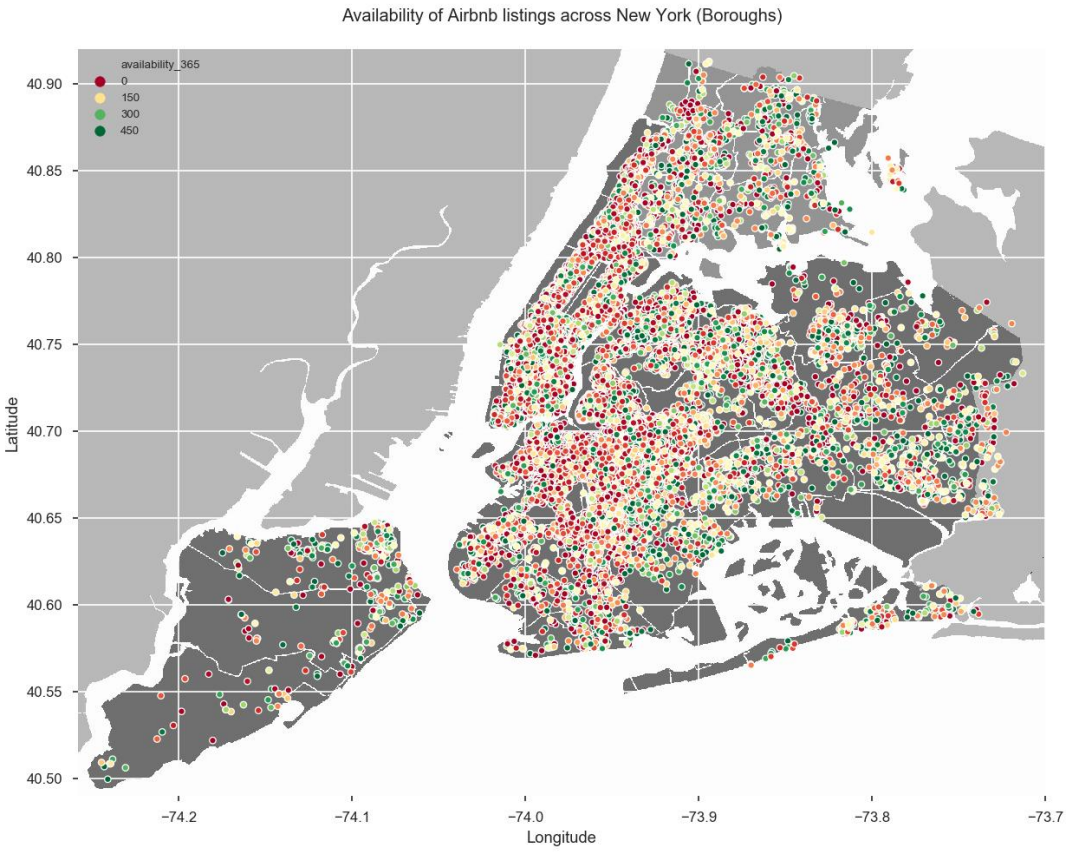
neighbourhood_group	room_type	
Bronx	Entire home/apt	379
	Private room	652
	Shared room	60
Brooklyn	Entire home/apt	9559
	Private room	10132
	Shared room	413
Manhattan	Entire home/apt	13199
	Private room	7982
	Shared room	480
Queens	Entire home/apt	2096
	Private room	3372
	Shared room	198
Staten Island	Entire home/apt	176
	Private room	188
	Shared room	9

Let's explore the availability of the same rooms based on `neighbourhood_groups` (boroughs)

```
plt.figure(figsize=(10,8))
sns.boxplot(x='neighbourhood_group',y='availability_365',data=df)
plt.xlabel('Neighbourhood groups(Boroughs)',fontsize=15)
plt.ylabel('Availability',fontsize=15)
plt.title('\n\nAvailability Distribution among Neighbourhood Groups(Boroughs)\n',fontsize=18)
plt.show()
```

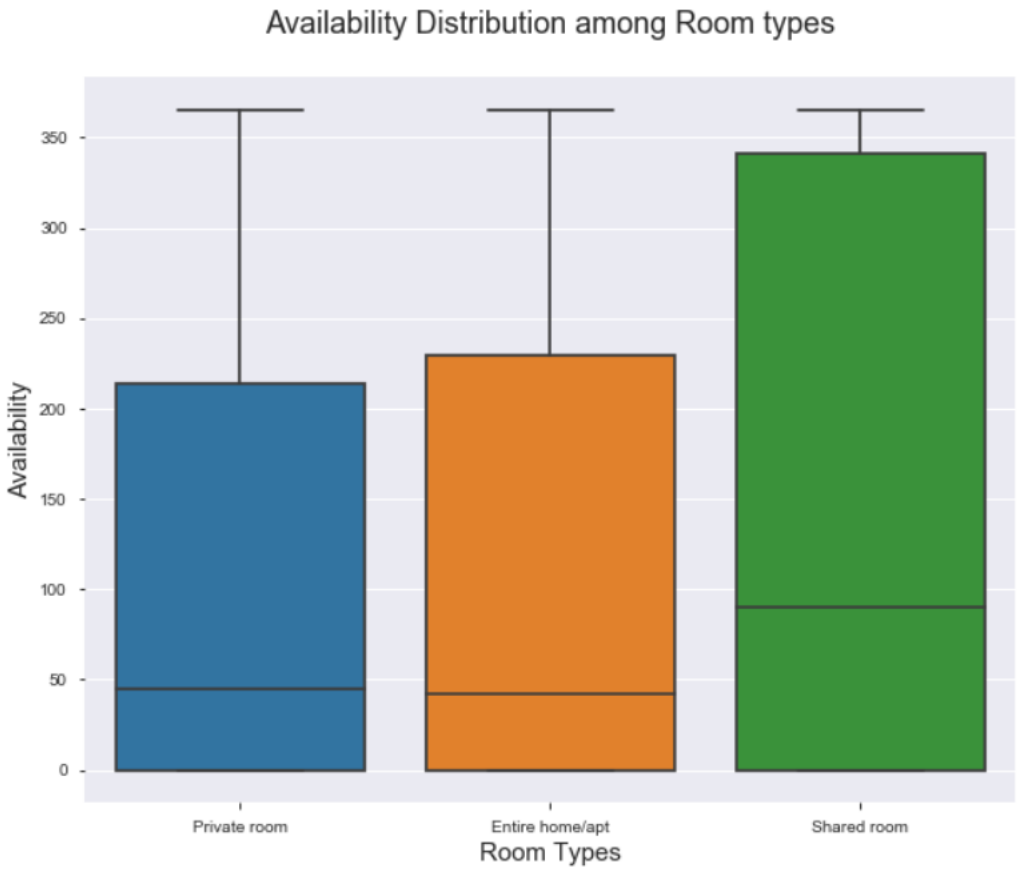


From above we can see that there are less availability in the `Manhattan` and `Brooklyn` and there are more availability in `Bronx` and `Staten Island` , let's visualize the same on map



Let's explore the availability of the same rooms based on `room_type`

In [3]:
```
plt.figure(figsize=(10,8))
sns.boxplot(x='room_type',y='availability_365',data=df)
plt.xlabel('Room Types',fontsize=15)
plt.ylabel('Availability',fontsize=15)
plt.title('\n\nAvailability Distribution among Room types\n',fontsize=18)
plt.show()
```

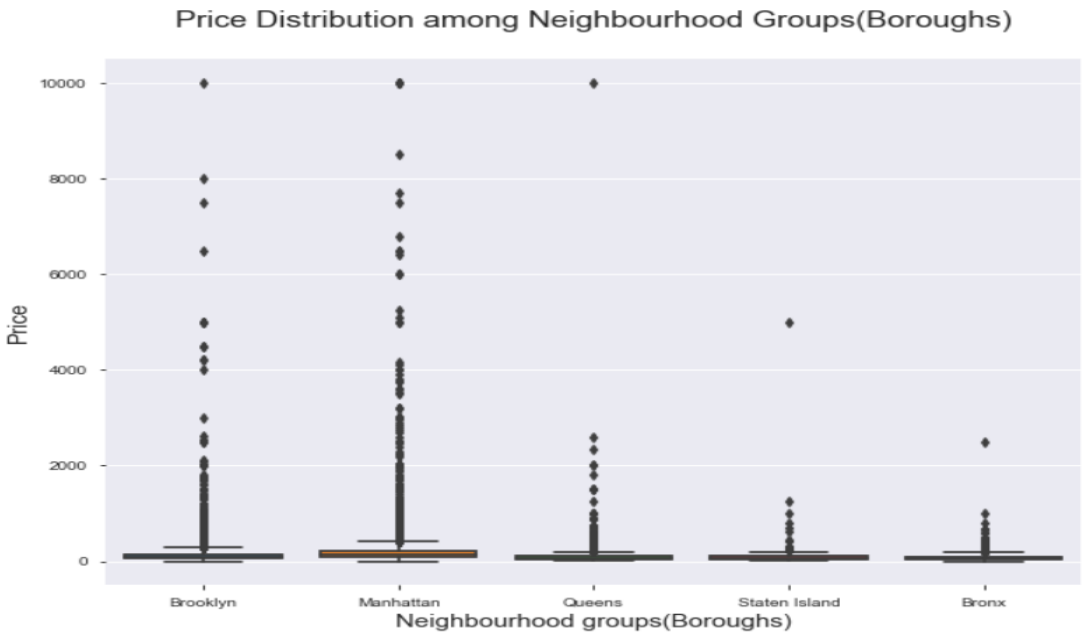


- From above we can see that there is less availability in Private rooms and more availability in shared rooms.
- Let's see how many rooms available in each type in each neighborhood with average availability of each rooms.
- Let's explore how many rooms are available in each type of rooms in each neighbourhoods & average availability of each rooms

```
a=pd.DataFrame(df[['neighbourhood_group','room_type','id']]
               .groupby(['neighbourhood_group','room_type']).count())
a.rename(columns={'id':'Number of Rooms'},inplace=True)
b=pd.DataFrame(df[['neighbourhood_group',
                   'room_type','availability_365']]
               .groupby(['neighbourhood_group','room_type']).mean())
b.rename(columns={'availability_365':'Average availability'},inplace=True)
c=pd.concat([a,b['Average availability']],axis=1)
display(c)
```

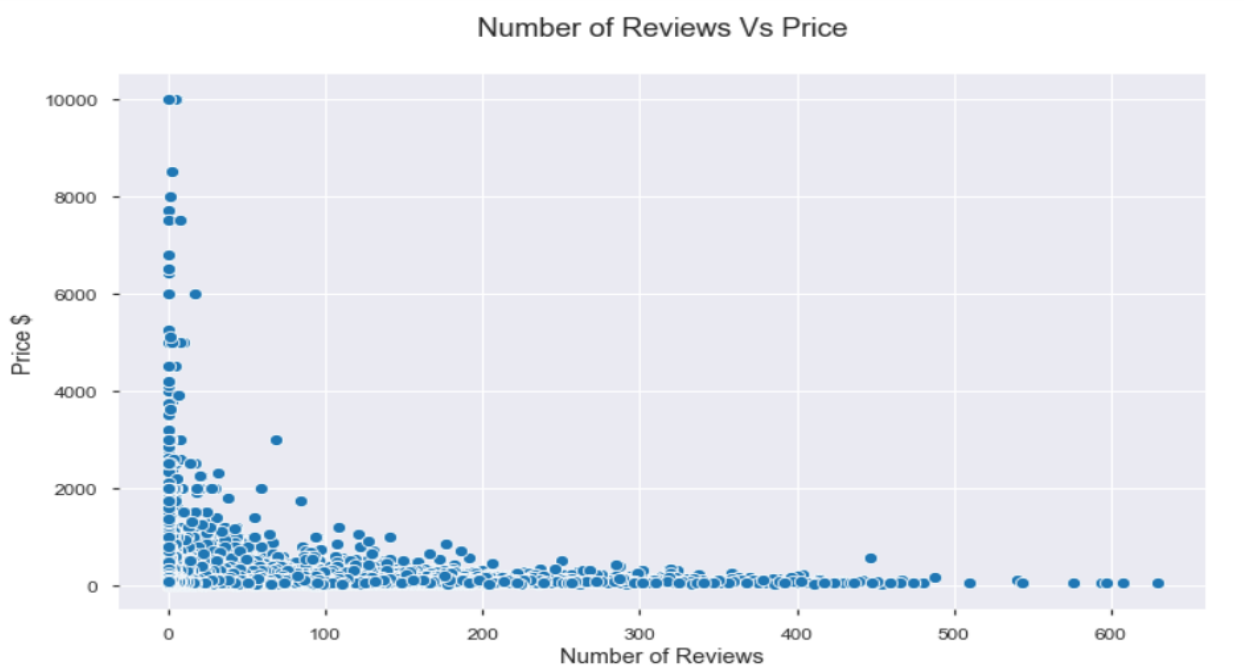
neighbourhood_group	room_type	Number of Rooms	Average availability
Bronx	Entire home/apt	379	158.000000
	Private room	652	171.331288
	Shared room	60	154.216667
Brooklyn	Entire home/apt	9559	97.205147
	Private room	10132	99.917983
	Shared room	413	178.007264
Manhattan	Entire home/apt	13199	117.140996
	Private room	7982	101.845026
	Shared room	480	138.572917
Queens	Entire home/apt	2096	132.267176
	Private room	3372	149.222716
	Shared room	198	192.186869
Staten Island	Entire home/apt	176	178.073864
	Private room	188	226.361702
	Shared room	9	64.777778

- Distribution of price among various neighbourhood_group to check for outliers (Box-plot) and skewness (Histogram)



From above we can see that the price variable is heavily skewed towards right(which needs to be normalized)

- Let's check whether, there is any relationship between number_of_reviews and price variable



As most of our Airbnb locations have 0 reviews because they stay for minimum number of days but sometimes longer stay ends up with good reviews(its not guaranteed), thus we will be neglecting Number of reviews as it is not a good predictor of Price

DATA WRANGLING

- Identify and handling missing values
- Data Formatting (categorical to numerical, changing datatypes)
- Categorical data transformation (One-Hot Encoding)
- Duplicates Removal
- Skewness
- Outliers

MISSING VALUES

```
In [6]: display(pd.concat([df.isnull().sum().sort_values(ascending =False),
                          ((df.isnull().sum() / len(df))*100).sort_values(ascending = False)],
                         axis = 1,keys = ['Total','Percent %']))
```

```
In [7]: #filling the NaN values with 0 for 'reviews_per_month' variable
        df['reviews_per_month'].fillna(0,inplace=True)
```

```
In [8]: df.drop(['id','name','host_id','host_name','last_review'],axis=1,inplace=True)
```

```
In [9]: # for the rest of the variables we will just drop the NaN entries
        df.dropna(inplace=True)
```

```
In [10]: # Checking the shape of the dataset after dropping the NaN values
         df.shape
```

```
Out[10]: (48895, 11)
```

	Total	Percent %
reviews_per_month	10052	20.558339
last_review	10052	20.558339
host_name	21	0.042949
name	16	0.032723
availability_365	0	0.000000
calculated_host_listings_count	0	0.000000
number_of_reviews	0	0.000000
minimum_nights	0	0.000000
price	0	0.000000
room_type	0	0.000000
longitude	0	0.000000
latitude	0	0.000000
neighbourhood	0	0.000000
neighbourhood_group	0	0.000000
host_id	0	0.000000
id	0	0.000000

From above we can see that the `last_review` and `reviews_per_month` are missing about 20% of the data, hence we will be replacing null values in the column `reviews_per_month` with '0' in the dataset(instead of dropping as its logic to conclude that if there aren't any guests for the corressponding location the number of reviews will drop to 0) but not for `last_review` as it indicates the just only the date of latest review(last) which won't be quite useful for Regression analysis.

And we will also be dropping the features `id` , `name` , `host_name` , `host_id` and `last_review` as they possess insignificant relationship to the Target variable `price` and also these are textual data. Hence we will be dropping these features while modelling but for Exploratiry Data Analysis(EDA) we will make use of it to understand the dataset better.

CATEGORICAL DATA TRANSORMATION

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48895 entries, 0 to 48894
Data columns (total 11 columns):
neighbourhood_group               48895 non-null object
neighbourhood                     48895 non-null object
latitude                          48895 non-null float64
longitude                         48895 non-null float64
room_type                         48895 non-null object
price                             48895 non-null int64
minimum_nights                    48895 non-null int64
number_of_reviews                 48895 non-null int64
reviews_per_month                 48895 non-null float64
calculated_host_listings_count    48895 non-null int64
availability_365                  48895 non-null int64
dtypes: float64(3), int64(5), object(3)
memory usage: 4.5+ MB
```

As we can see that there are 3 categorical variables which needs to be one-hot encoded first for regression analysis as the model wont take any categorical values.

1. `neighbourhood_group` contains only 5 unique values hence it can be one-hot encoded from 0 to 4
2. `neighbourhood` contains 221 unique values hence it cannot be transformed into one-hot encoding as it will create 221 more columns, but we don't have a choice because if we just transformed it into numerical values(without one-hot encoding) the reggression model will consider that a unit increase/decrease in neighbourhood will leads to some amount of increase/decrease in price, which is not a case here as these variable is categorical in nature
3. `room_type` has only 3 unique values hence it can be one-hot encoded from 0 to 2

ONE-HOT ENCODING

```
In [11]: df1= pd.get_dummies(df)
         df1.head()
```

Out[11]:

	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_count	availability_365	neighbourhood_gro
0	40.64749	-73.97237	149	1	9	0.21	6	365	
1	40.75362	-73.98377	225	1	45	0.38	2	355	
2	40.80902	-73.94190	150	3	0	0.00	1	365	
3	40.68514	-73.95976	89	1	270	4.64	1	194	
4	40.79851	-73.94399	80	10	9	0.10	1	0	

5 rows × 237 columns

after one-hot encoding we will be having 237 columns

```
In [12]: df1.shape
```

Out[12]: (48895, 237)

SKEWNESS:

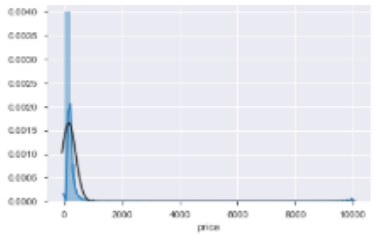
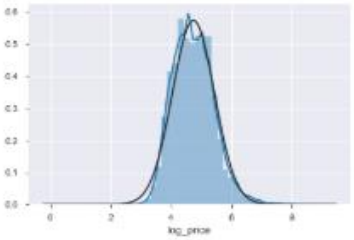
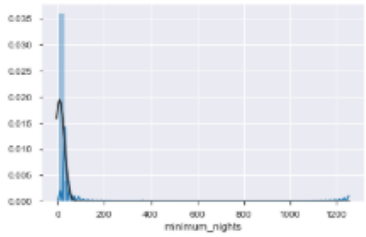
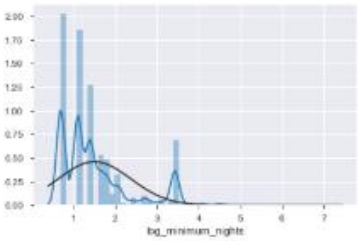
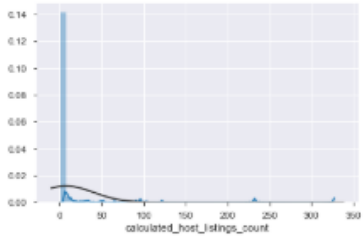
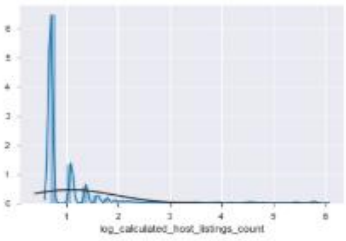
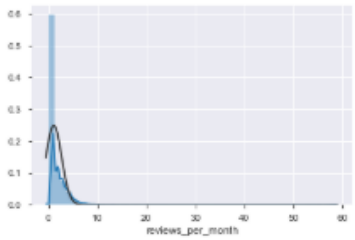
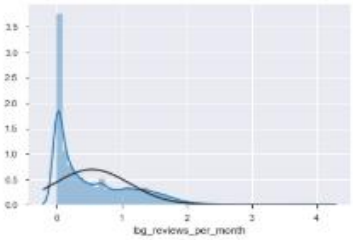
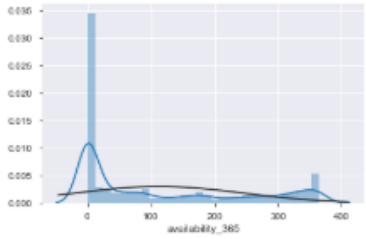
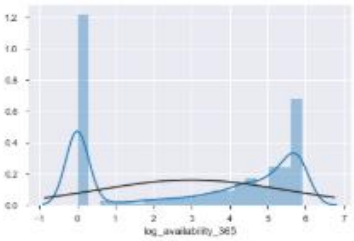
- If a independent/dependent variable has a right skew (mean > median). Taking the log would make the distribution of your transformed variable appear more symmetric (more normal). However, this is not a very good reason to log your variable. There are no regression assumptions that require your independent or dependent variables to be normal. However, if you have outliers in your dependent or independent variables, a log transformation could reduce the influence of those observations.
- The variance of your regression residuals is increasing with your regression predictions. Taking the log of your dependent and/or independent variables may eliminate the heteroscedasticity of residuals and thereby improves the fitness of the model.

```
In [13]: df1[['price','minimum_nights',
               'calculated_host_listings_count',
               'reviews_per_month','availability_365'
              ]].skew()
```

Out[13]:

```
price                             19.118939
minimum_nights                    21.827275
calculated_host_listings_count     7.933174
reviews_per_month                  3.300723
availability_365                   0.763408
dtype: float64
```

```
#applying log transformation
df1['log_price'] = np.log1p(df1['price'])
df1['log_minimum_nights'] = np.log1p(df1['minimum_nights'])
df1['log_calculated_host_listings_count'] = np.log1p(df1['calculated_host_listings_count'])
df1['log_reviews_per_month'] = np.log1p(df1['reviews_per_month'])
df1['log_availability_365'] = np.log1p(df1['availability_365'])
```

	Before Log Transformation	After Log Transformation
price		
minimum_nights		
calculated_host_listings_count		
reviews_per_month		
availability_365		

DATA PARTITIONING

- We will split/partition the data in to train and test split with 80% and 30% respectively
- Here we will be using RobustScaler to scale the data to nullify the effect of outliers present in the dataset

```
In [10]:
x=df.drop(['log_price'],axis=1)
y=df['log_price']
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split( x, y, test_size=0.30, random_state=42)
from sklearn.preprocessing import RobustScaler
scale= RobustScaler()
x_train_sc=scale.fit_transform(x_train)
x_test_sc=scale.transform(x_test)
```

Data Modelling :

1. KNN REGRESSOR

Based on the closest nearest neighbor distance(Manhattan or Euclidean) it will classify or assign the datapoints.

2. LINEAR REGRESSION

To capture the linear relationship between the dependent and independent variable.

3. RIDGE REGRESSION

Regularized Linear model which penalizes the higher coefficients by using alpha parameter to eliminate the overfitting issues.

4. LASSO REGRESSION

Least Absolute Shrinkage Selective Operator which penalizes the higher coefficients by using the parameter alpha to eliminate the overfitting issues and feature selection

5. SVR – RBF KERNEL

Support Vector Regressor with the Radial Bias Function Kernel.

6. DECISION TREES

Based on Gini index a split will be made at each node to completely classify the data.

7. RANDOM FOREST

Utilizes bagging (bootstrapping and aggregating) strategy with base estimator as decision tree and creates a n number of trees based on which the model predicts. Here all the trees are independent to each other.

8. GRADIENT BOOSTING

Stochastic Gradient Boosting Regressor in which all the weak learners will be combined to format a strong learner. Here decision tree is the base estimator and all estimators are dependent on each other(learns by residuals of the previous decision tree)

9. LINEAR SVR

Linear Support Vector Regressor for the regression tasks.
It implements an original proprietary version of a cutting plane algorithm for designing a linear support vector machine

10. ARTIFICIAL NEURAL NETWORK

Artificial Neural networks (ANN) are computational algorithms. It intended to simulate the behavior of biological systems composed of "neurons" and inspired by an central nervous systems. It is capable of machine learning as well as pattern recognition

11. POLYNOMIAL REGRESSION

Polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial in x. it fits a nonlinear relationship between x and y, denoted E(y |x)

12. K MEANS CLUSTERING

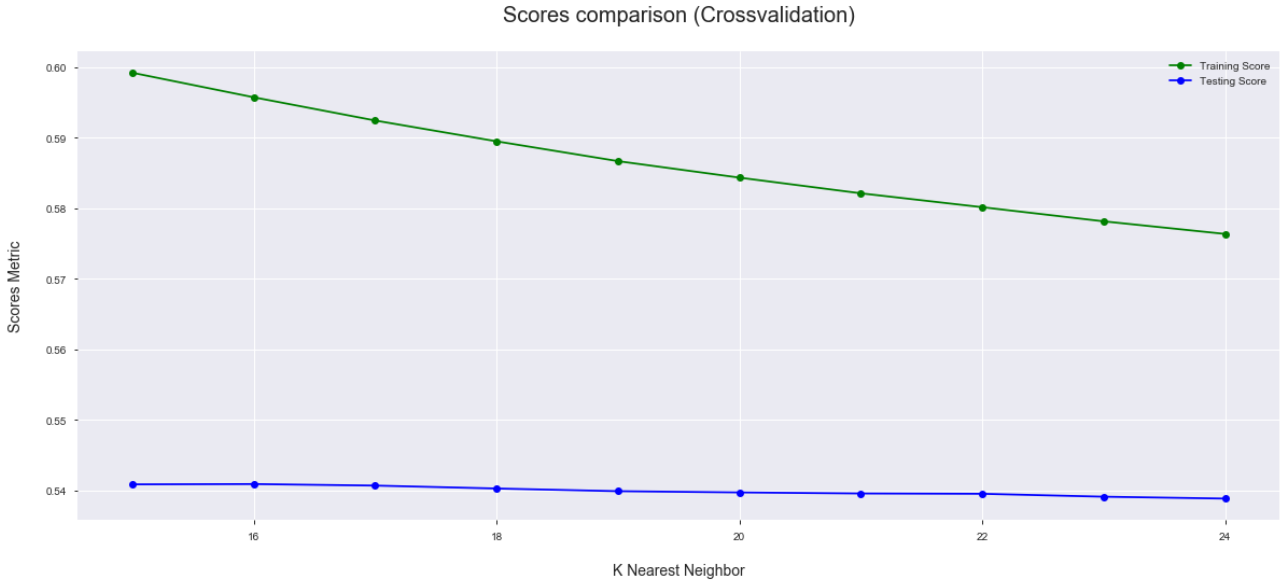
K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

KNN REGRESSOR

Grid search parameters:

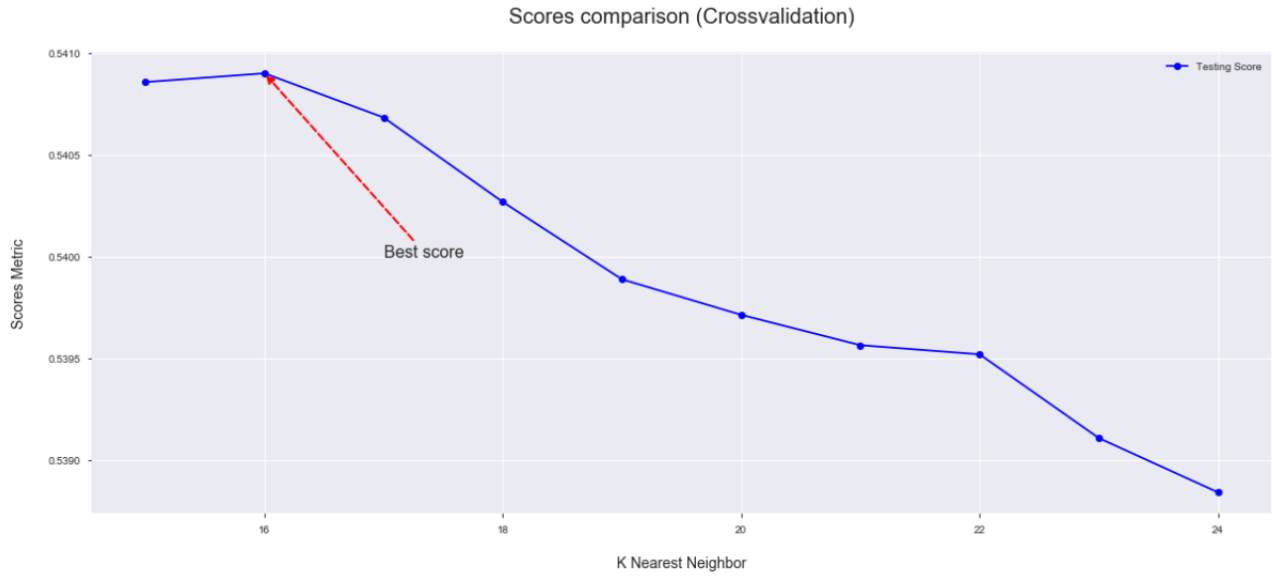
knnparameters = {'n_neighbors':range(15,25)}

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
knnparameters = {
    'n_neighbors':range(15,25)
}
reg = GridSearchCV(KNeighborsRegressor(),
                   knnparameters,return_train_score=True, cv = 5,n_jobs=-1)
reg.fit(x_train_sc, y_train)
```



Grid Search Results(embedded):

mean_fit_time	std_fit_time	mean_score_time	std_score_time	params	mean_test_score	mean_train_score	std_train_score
5.339925671	0.330100892	47.90711107	0.331898168	{'n_neighbors': 15}	0.540857527	0.599218368	0.003017687
5.082831812	0.09452334	48.37598076	0.647328934	{'n_neighbors': 16}	0.540900871	0.595739247	0.003044617
4.995753622	0.277021324	51.05363369	0.492124166	{'n_neighbors': 17}	0.540683751	0.592447228	0.002927303
4.762053871	0.619945969	51.79148946	1.043357495	{'n_neighbors': 18}	0.540268118	0.589496699	0.002939102
3.990099907	0.483285815	42.29691157	2.811984916	{'n_neighbors': 19}	0.539887787	0.586687443	0.003098054
4.708196211	0.497449293	48.19317527	0.882501844	{'n_neighbors': 20}	0.539713152	0.584348701	0.003111389
4.11592741	0.543780544	42.33138123	1.766722685	{'n_neighbors': 21}	0.539563951	0.582126278	0.003397383
4.036872149	0.187834514	44.43656645	1.512150017	{'n_neighbors': 22}	0.539519241	0.580156052	0.003507676
5.529732561	0.361458614	59.02943397	0.68877691	{'n_neighbors': 23}	0.539107321	0.578153334	0.003402129
5.63770957	0.913390966	60.46733656	4.061065017	{'n_neighbors': 24}	0.538840564	0.576366763	0.003349648



Results:

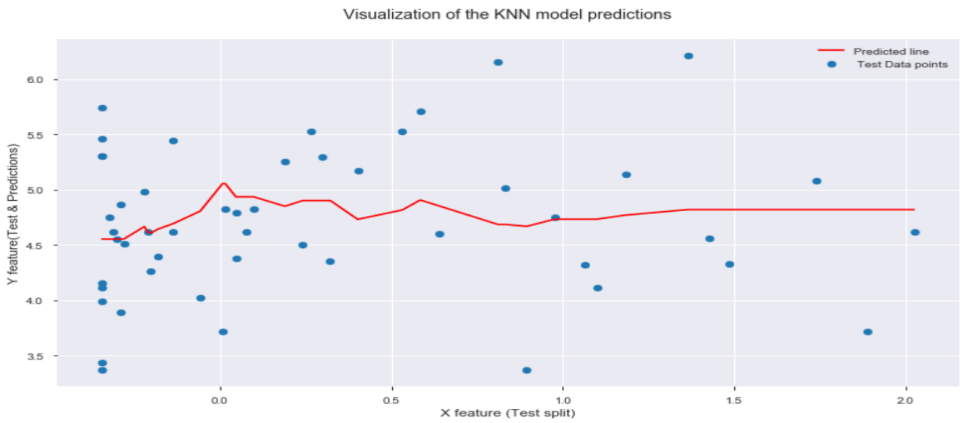
Best Parameters { 'n_neighbors': 16}

Metrics:

- › Best Score(accuracy) : 0.5409
- › Test Score (accuracy) : 0.5626
- › Mean Squared Error : 0.4546
- › R2 Score : 0.5626
- › Mean Absolute Error : 0.3285

```
from sklearn.neighbors import KNeighborsRegressor
X_b = x_train_sc[:50,234].reshape(-1,1)
y_b = y_train[:50]
knn_reg = KNeighborsRegressor(16)
knn_reg.fit(X_b, y_b)
cm=pd.DataFrame({'x':x_test_sc[:50,234].tolist(),
                 'y':y_test[:50].tolist()}).sort_values(by='x')
y_predict = knn_reg.predict(cm['x'].values.reshape(-1,1))
plt.figure(figsize=(13,7))
plt.plot(cm['x'].tolist(), y_predict, c = 'r',label='Predicted line')
plt.scatter(cm['x'].tolist(), cm['y'].tolist(),label=' Test Data points')
plt.title('\nVisualization of the KNN model predictions\n',fontsize=14)
plt.xlabel('X feature (Test split)',fontsize=12)
plt.ylabel('Y feature(Test & Predictions)',fontsize=12)
plt.legend()
plt.show()
```

The best fitted line by the KNN Model



LINEAR REGRESSION

Metrics:

- › Train Score (accuracy) : 0.5540
- › Test Score (accuracy) : -3.2041
- › Mean Squared Error : 123059676.092
- › R2 Score : **-3.2041**
- › Mean Absolute Error : 1801289.535

```
from sklearn.linear_model import LinearRegression
lnreg=LinearRegression()
lnreg.fit(x_train_sc,y_train)
print('\n\n"Linear Regression Model and it\'s Metrics" :\n')
print('\t• Training score of the Linear Regression Model is :',
      lnreg.score(x_train_sc,y_train))
print('\n\t• Testing score of the Linear Regression Model is :',
      lnreg.score(x_test_sc,y_test))
pred= lnreg.predict(x_test_sc)
print(f'\n\t• Mean Squared Error:
      {np.sqrt(mean_squared_error(y_test, pred))}
      \n\n\t• R2 Score: {r2_score(y_test,pred)} \n\n\t•
      Mean Absolute Error: {mean_absolute_error(y_test,pred)}')
```

R^2 is negative only when the chosen model does not follow the trend of the data, so fits worse than a horizontal line.(ie) the relationship between the independent variable and the dependent variable is not a simple linear relationship and it might be a complex linear relationship which cann't be captured by as a simple Linear regression model like as above.

A negative R^2 is not a mathematical impossibility or the sign of a computer bug. It simply means that the chosen model (with its constraints) fits the data really poorly.

When the test score is relatively poor than the training score , it indicates that the model is suffering from the problem of `over-generalization` or `over-fitting` and it can be eliminated by either `'Ridge'` or `'LASSO'` model which is also known as the Regularized Linear model

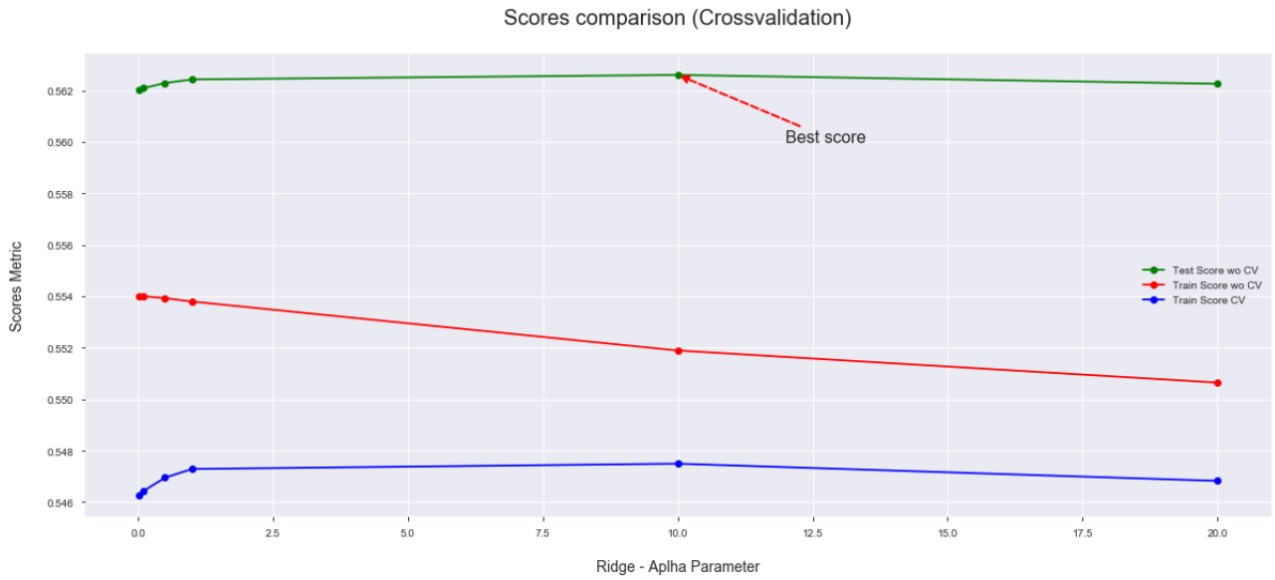
RIDGE REGRESSION

Best Parameters: { 'alpha': 10 }

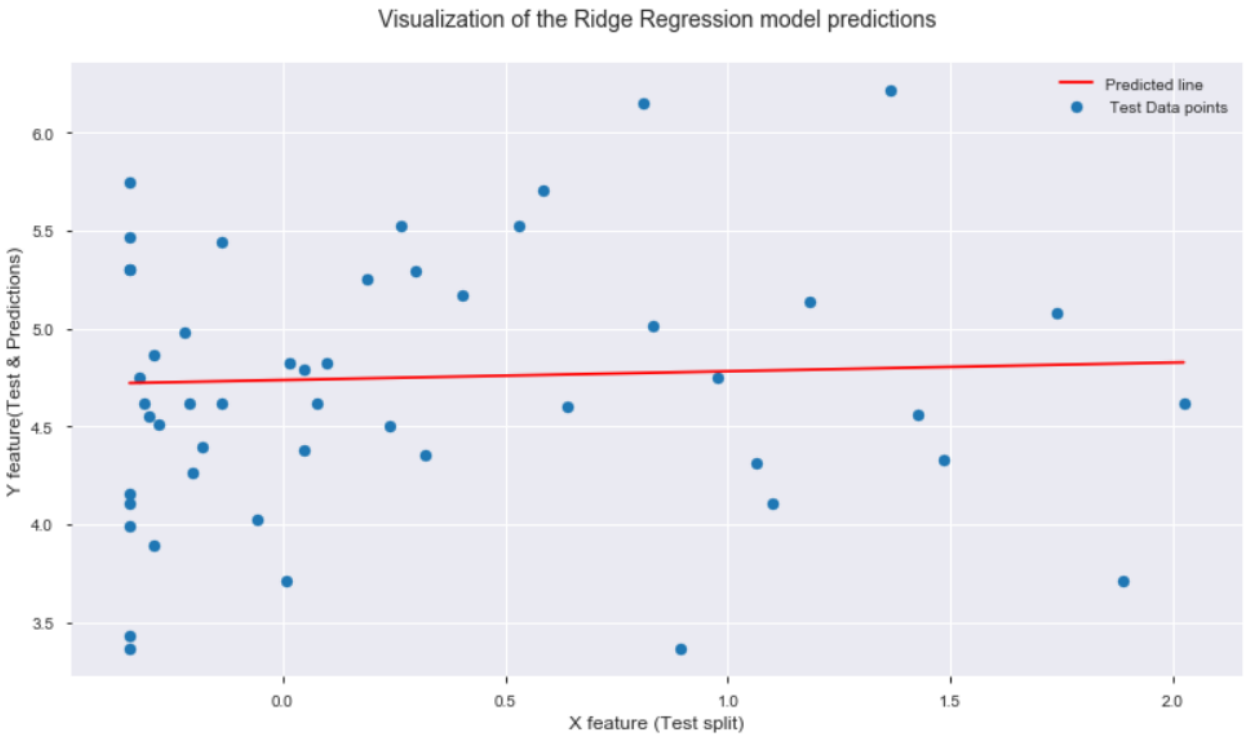
Metrics:

- › Train Score (accuracy) : 0.5474
- › Test Score (accuracy) : 0.5626
- › Mean Squared Error : 0.2067
- › R2 Score : 0.5626
- › Mean Absolute Error : 0.3289

```
from  sklearn.linear_model import Ridge
train_score_list = []
test_score_list = []
cross_score_list=[]
for alpha in [0.01,0.1,0.5,1,10,20]:
    ridge = Ridge(alpha,random_state=41)
    scores = cross_val_score(ridge, x_train_sc, y_train, cv=5)
    print('\nFor alpha =',alpha)
    cross_score_list.append(scores.mean())
    ridge.fit(x_train_sc,y_train)
    y_pred=ridge.predict(x_test_sc)
    train_score_list.append(ridge.score(x_train_sc,y_train))
    test_score_list.append(ridge.score(x_test_sc, y_test))
    print('\n\t•Training score without Cross validation: ',train_score_list[-1])
    print('\n\t•Cross validation Training score mean:',scores.mean())
    print('\n\t•Testing score without Cross validation: ',test_score_list[-1])
    print(f'\n\t• Mean Squared Error: {(mean_squared_error(y_test, y_pred))}
          \n\n\t• R2 Score: {r2_score(y_test,y_pred)} \n\n\t
          • Mean Absolute Error: {mean_absolute_error(y_test,y_pred)}')
    print('-------------------------------------------------------------------------
```



The best fitted line by the Ridge Model



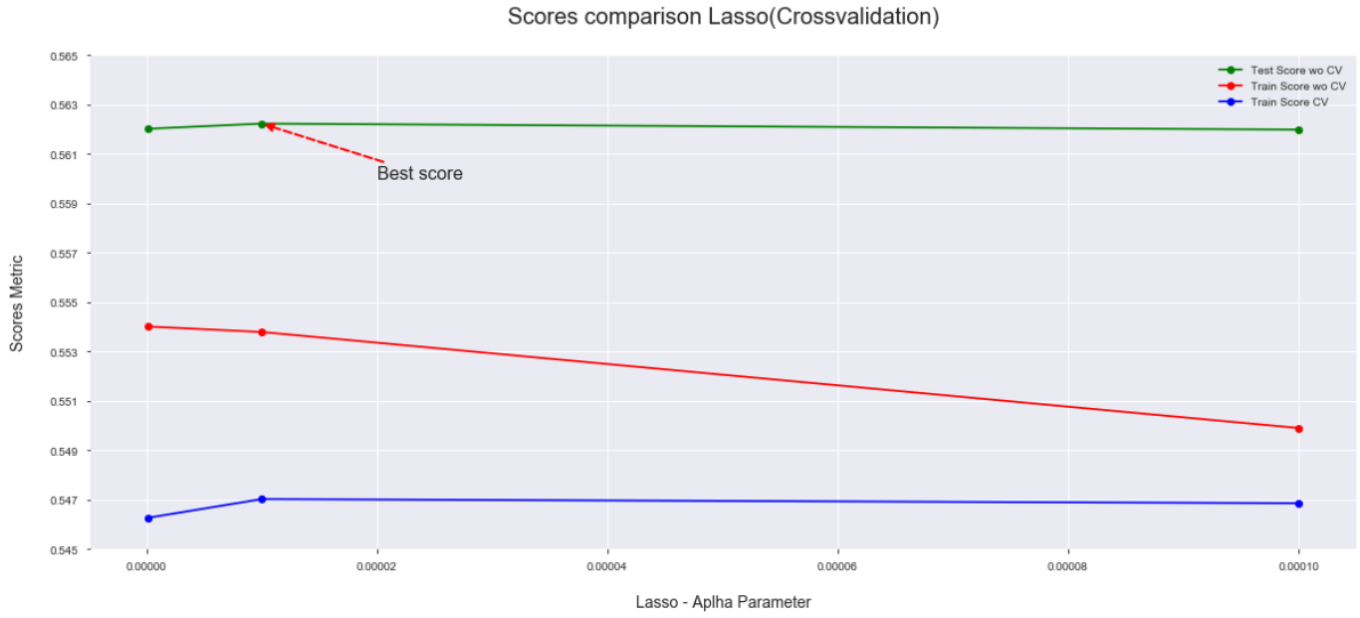
LASSO REGRESSION

Best Parameters: { 'alpha': 1e-05 }

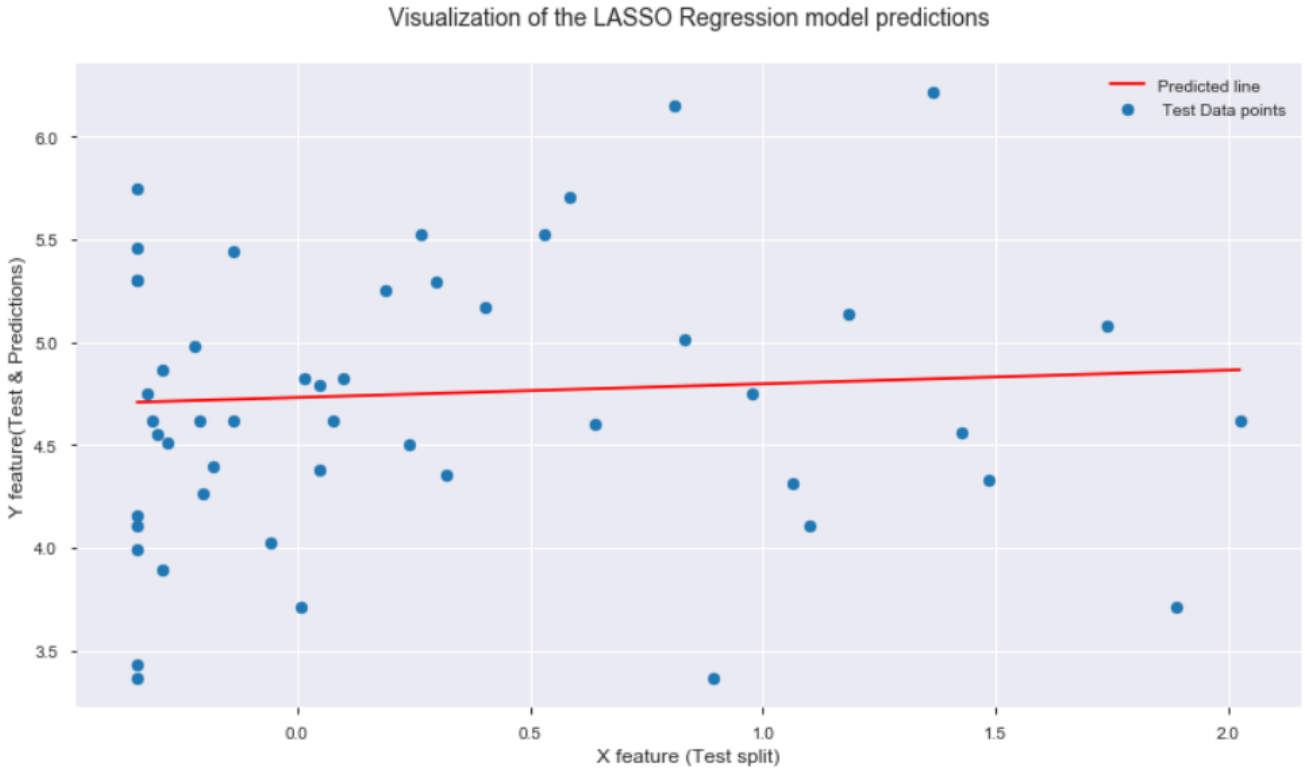
Metrics:

- Train Score (accuracy) : 0.5470
- Test Score (accuracy) : 0.5622
- Mean Squared Error : 0.2069
- R2 Score : 0.5622
- Mean Absolute Error : 0.3292

```
train_score_list = []
test_score_list = []
cross_score_list=[]
for alpha in [0.0000001,0.00001,0.0001,0.001,0.01,0.1,0.5,0.8,1]:
    lasso = Lasso(alpha,random_state=41,max_iter=3000)
    scores = cross_val_score(lasso, x_train_sc, y_train, cv=5)
    print('\nFor alpha =',alpha)
    cross_score_list.append(scores.mean())
    lasso.fit(x_train_sc,y_train)
    y_pred=lasso.predict(x_test_sc)
    train_score_list.append(lasso.score(x_train_sc,y_train))
    test_score_list.append(lasso.score(x_test_sc, y_test))
    print('\n\t•Training score without Cross validation: ',train_score_list[-1])
    print('\n\t•Cross validation Training score mean:',scores.mean())
    print('\n\t•Testing score without Cross validation: ',test_score_list[-1])
    print(f'\n\t• Mean Squared Error: {(mean_squared_error(y_test, y_pred))}
          \n\n\t• R2 Score: {r2_score(y_test,y_pred)} \n\n\t•
          Mean Absolute Error: {mean_absolute_error(y_test,y_pred)}')
    print('-----------------------------------------------------------------------
```



The best fitted line by the LASSO Model



POLYNOMIAL REGRESSION

For Degree = 2

Metrics:

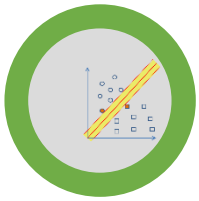
- Train Score (accuracy) : 0.6296
- Test Score (accuracy) : -494221579336.88
- Mean Squared Error : 233582513324.7001
- Root Mean Squared Error : 483303.748
- R2 Score : -494221579336.88
- Mean Absolute Error : 21123.2228

```
from  sklearn.preprocessing  import PolynomialFeatures
poly = PolynomialFeatures(degree = 2)
print('Degree initialized')
X_train_poly = poly.fit_transform(x_train[:20000])
print('train set is transormed')
X_test_poly = poly.transform(x_test[:20000])
print('test set is transformed')
from sklearn.linear_model import LinearRegression
lreg = LinearRegression()
print('Linear regression initialized and now fitting')
lreg.fit(X_train_poly, y_train[:20000])
print('model fitted and now predicting')
y_pred=lreg.predict(X_test_poly)
print('Training score',lreg.score(X_train_poly, y_train[:20000]))
print('Testing score',lreg.score(X_test_poly, y_test[:20000]))
print(f'Polynomial Linear Regression Metrics:\n \n\t•
      Mean Squared Error: {(mean_squared_error(y_test[:20000], y_pred))}
      \n\n\t• Root Mean Squared Error:
      {np.sqrt(mean_squared_error(y_test[:20000], y_pred))} \n\n\t•
      R2 Score: {r2_score(y_test[:20000],y_pred)} \n\n\t•
      Mean Absolute Error: {mean_absolute_error(y_test[:20000],y_pred)}')
```

R^2 is negative only when the chosen model does not follow the trend of the data, so fits worse than a horizontal line.(ie) the relationship between the independent variable and the dependent variable is not a simple linear relationship and it might be a complex linear relationship which cann't be captured by as a simple Linear regression model like as above.

A negative R^2 is not a mathematical impossibility or the sign of a computer bug. It simply means that the chosen model (with its constraints) fits the data really poorly.

When the test score is relatively poor than the training score , it indicates that the model is suffering from the problem of `over-generalization` or `over-fitting` and it can be eliminated by either `'Ridge'` or `'LASSO'` model which is also known as the Regularized Linear model



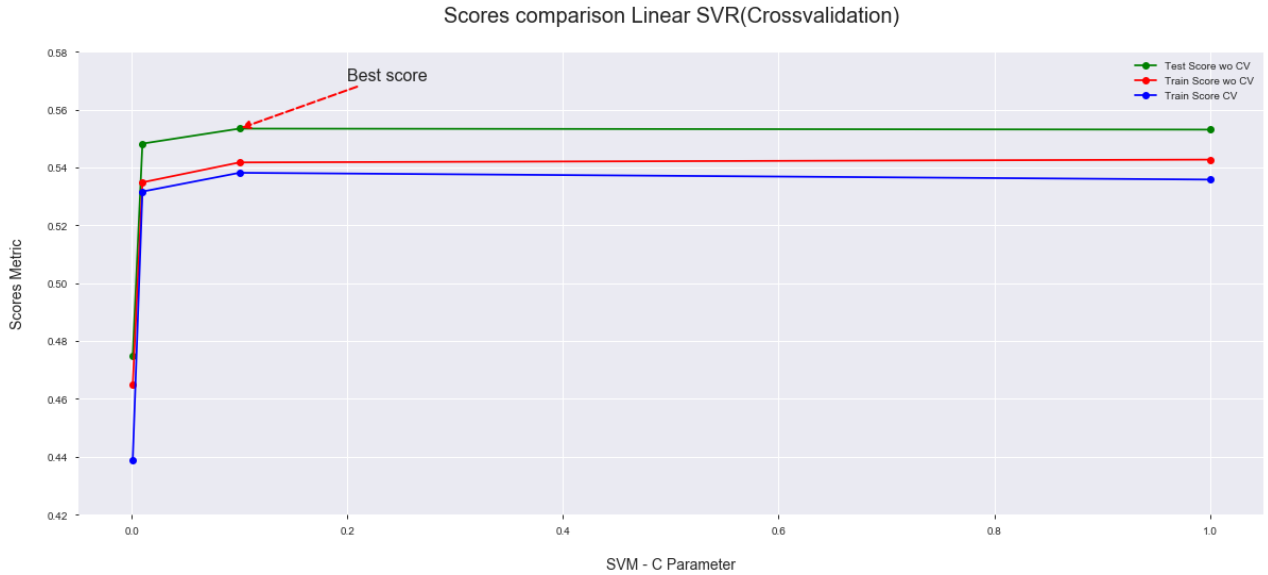
LINEAR SUPPORT VECTOR REGRESSOR

Best Parameters: { 'C': 0.1 }

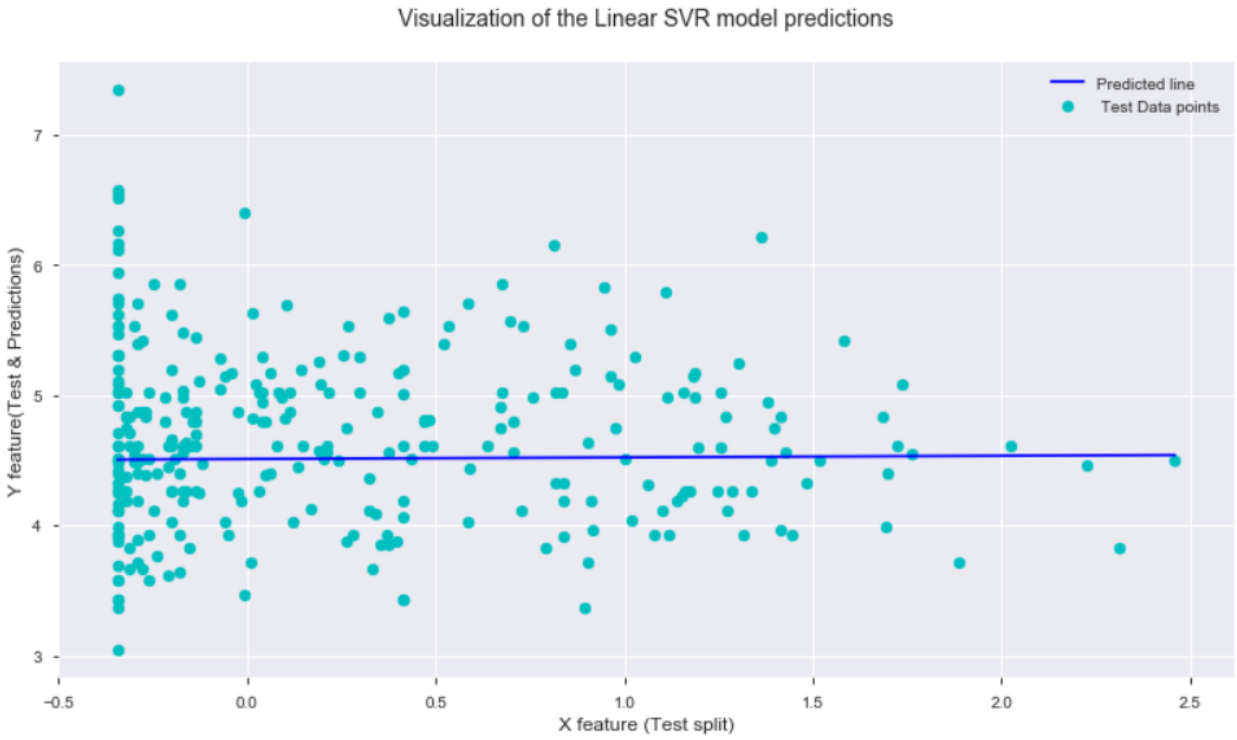
Metrics:

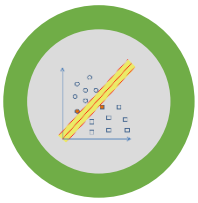
- Train Score (accuracy) : 0.5381
- Test Score (accuracy) : 0.5534
- Mean Squared Error : 0.2110
- R2 Score : 0.5534
- Mean Absolute Error : 0.3243

```
from sklearn.svm import LinearSVR
train_score_list = []
test_score_list = []
cross_score_list=[]
for C in [0.001, 0.01, 0.1, 1, 10, 100]:
    svreg = LinearSVR(random_state=41,C=C)
    scores = cross_val_score(svreg, x_train_sc, y_train, cv=5)
    print('\nFor C= ',C)
    cross_score_list.append(scores.mean())
    svreg.fit(x_train_sc,y_train)
    y_pred=svreg.predict(x_test_sc)
    train_score_list.append(svreg.score(x_train_sc,y_train))
    test_score_list.append(svreg.score(x_test_sc, y_test))
    print('\n\t•Training score without Cross validation: ',train_score_list[-1])
    print('\n\t•Cross validation Training score mean:',scores.mean())
    print('\n\t•Testing score without Cross validation: ',test_score_list[-1])
    print(f'\n\t• Mean Squared Error: {(mean_squared_error(y_test, y_pred))}
        \n\n\t• R2 Score: {r2_score(y_test,y_pred)} \n\n\t•
          Mean Absolute Error: {mean_absolute_error(y_test,y_pred)}')
```



The best fitted line by the Linear SVR Model





SUPPORT VECTOR REGRESSOR

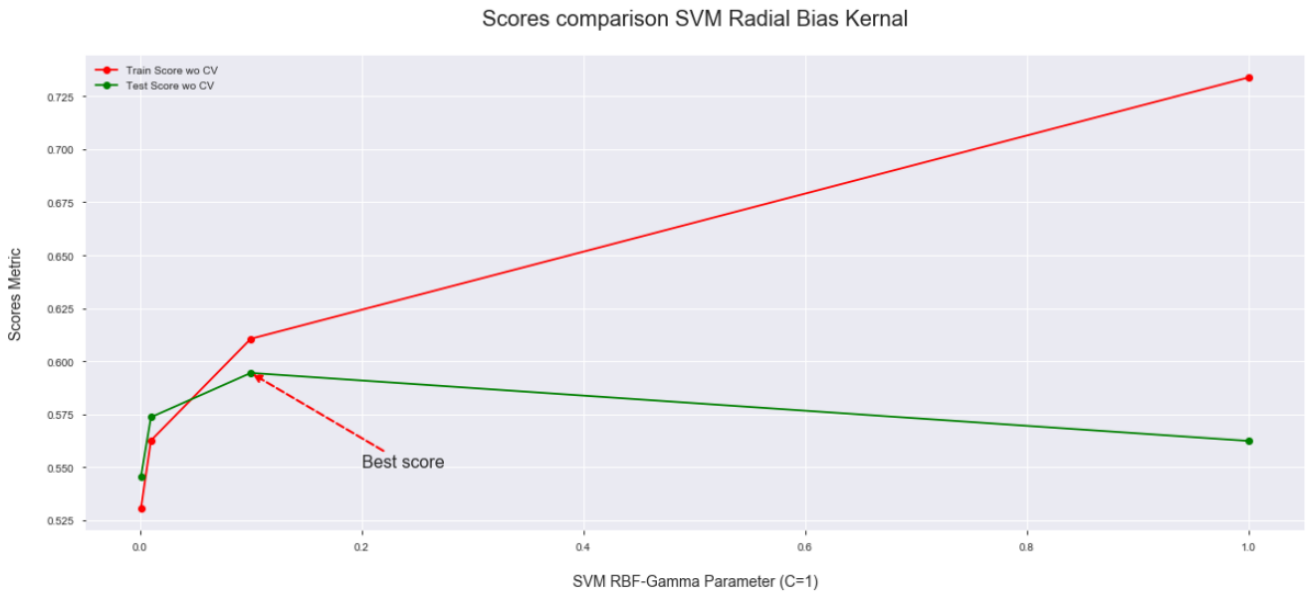
RADIAL KERNEL

Best Parameters: { 'C': 1 , 'gamma':0.1 }

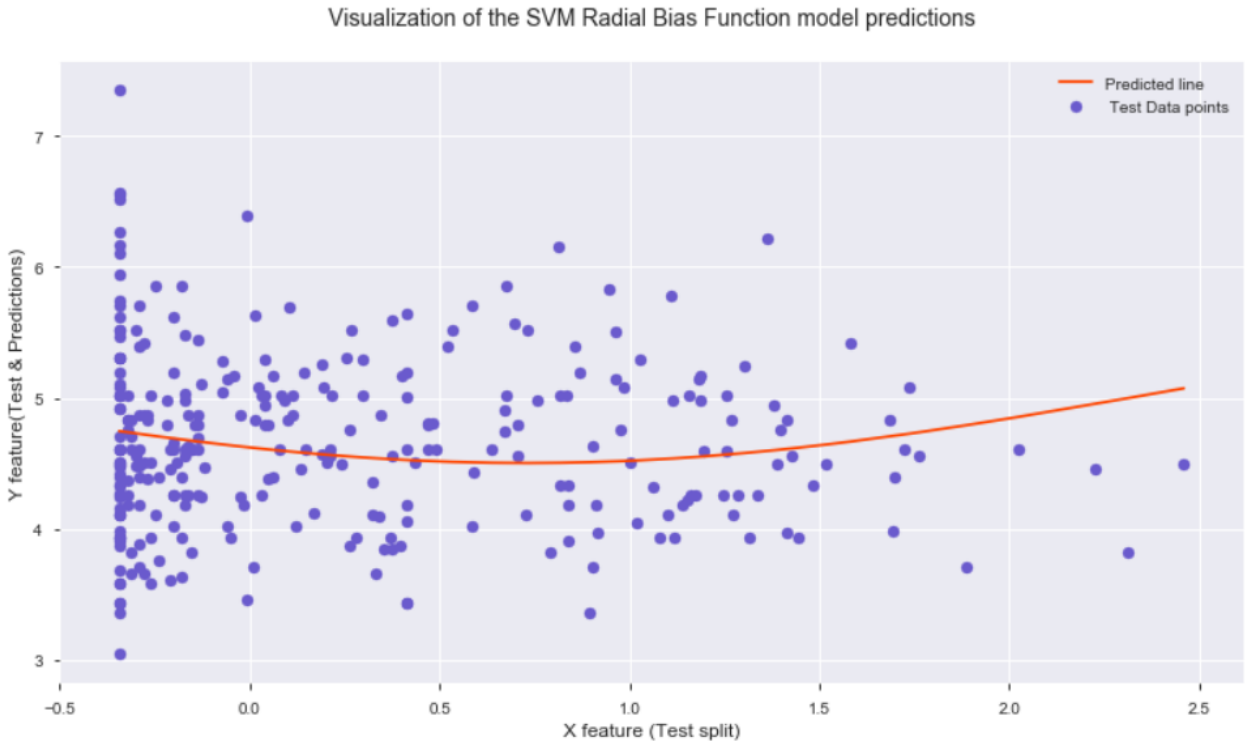
Metrics:

- Train Score (accuracy) : 0.6105
- Test Score (accuracy) : 0.5944
- Root Mean Squared Error : 0.4377
- R2 Score : 0.5944
- Mean Absolute Error : 0.3068

```
train_score_list = []
test_score_list = []
cross_score_list=[]
for C in [0.01, 0.1, 1, 10, 100]:
    for g in [0.001, 0.01, 0.1, 1, 10]:
            svreg = SVR(kernel='rbf',gamma=g,C=C)
            print('\nFor C= ',C,' and gamma = ',g)
            svreg.fit(x_train_sc,y_train)
            y_pred=svreg.predict(x_test_sc)
            train_score_list.append(svreg.score(x_train_sc,y_train))
            test_score_list.append(svreg.score(x_test_sc, y_test))
            print('\n\t•Training score without Cross validation: ',train_score_list[-1])
            print('\n\t•Testing score without Cross validation: ',test_score_list[-1])
            print(f'\n\t•Root Mean Squared Error:
                  {np.sqrt(mean_squared_error(y_test, y_pred))}
                  \n\n\t• R2 Score: {r2_score(y_test,y_pred)} \n\n\t•
                  Mean Absolute Error: {mean_absolute_error(y_test,y_pred)}')
            print('-------------------------------------------------------------------------
```



The best fitted line by the SVR Radial Kernel Model



DECISION TREE REGRESSOR

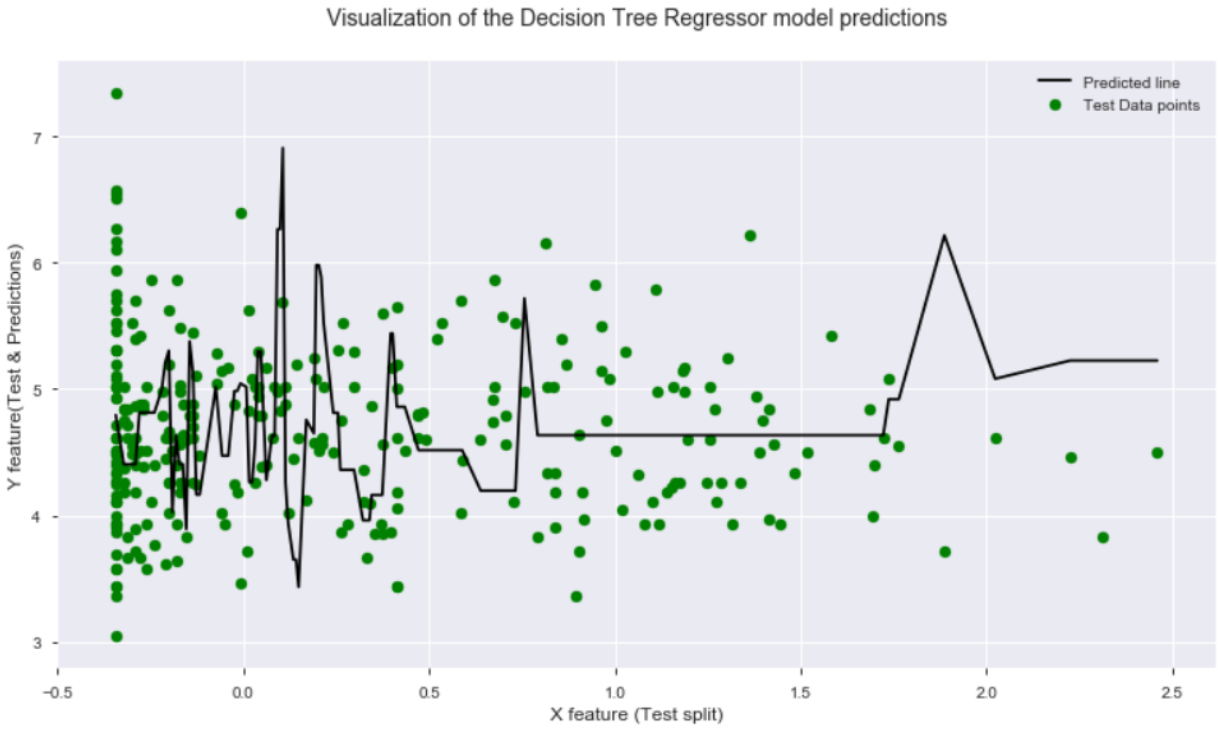
Best Parameters: { 'max_depth' : 8}

Metrics:

- Train Score (accuracy) : 0.5352
- Test Score (accuracy) : 0.5432
- Root Mean Squared Error : 0.4646
- R2 Score : 0.5432
- Mean Absolute Error : 0.3311

```
from sklearn.tree import DecisionTreeRegressor
train_score_list = []
test_score_list = []
cross_score_list=[]
for d in [3,5,8,10,12,15,18,20,22,24]:
    dtree = DecisionTreeRegressor(max_depth=d,random_state=41).
    scores = cross_val_score(dtree, x_train, y_train, cv=5)
    print('\nFor max_depth = ',d)
    cross_score_list.append(scores.mean())
    dtree.fit(x_train,y_train)
    y_pred=dtree.predict(x_test)
    train_score_list.append(dtree.score(x_train,y_train))
    test_score_list.append(dtree.score(x_test, y_test))
    print('\n\t•Training score without Cross validation: ',train_score_list[-1])
    print('\n\t•Cross validation Training score mean:',scores.mean())
    print('\n\t•Testing score without Cross validation: ',test_score_list[-1])
    print(f'\n\t• Root Mean Squared Error:
          {np.sqrt(mean_squared_error(y_test, y_pred))}
          \n\n\t• R2 Score: {r2_score(y_test,y_pred)} \n\n\t•
          Mean Absolute Error: {mean_absolute_error(y_test,y_pred)}')
    print('-------------------------------------------------------------------------
```

The best fitted line by the Decision Tree Model



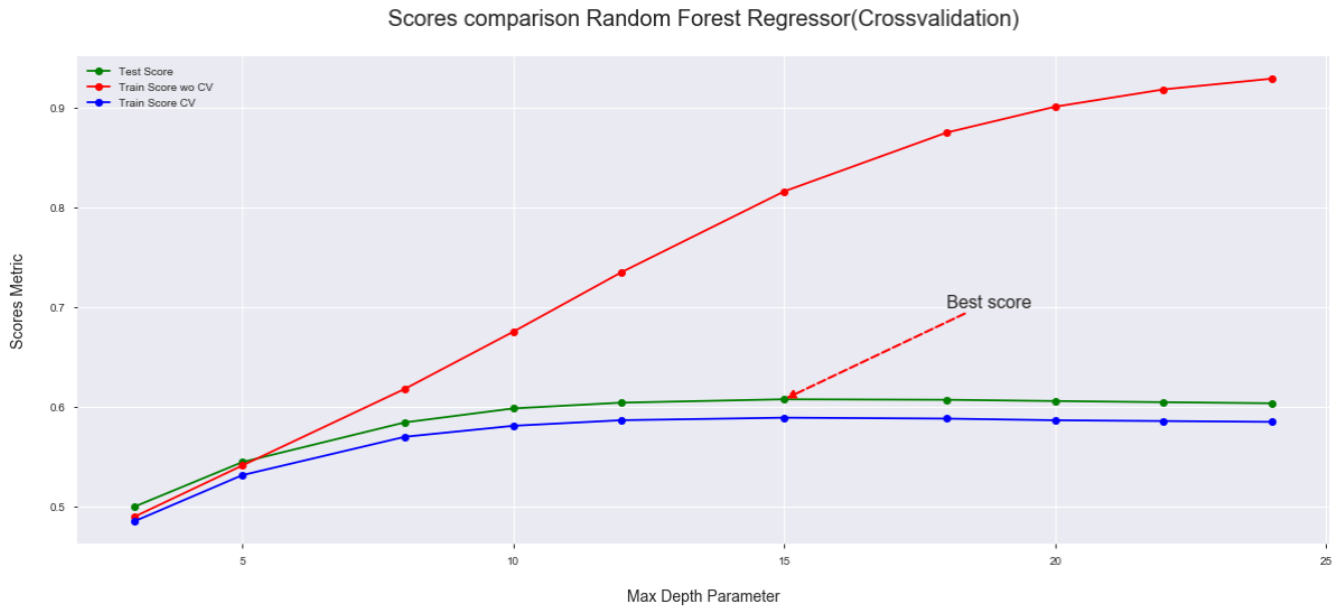
RANDOM FOREST REGRESSOR

Best Parameters: { 'max_depth' : 15, 'n_estimators':100}

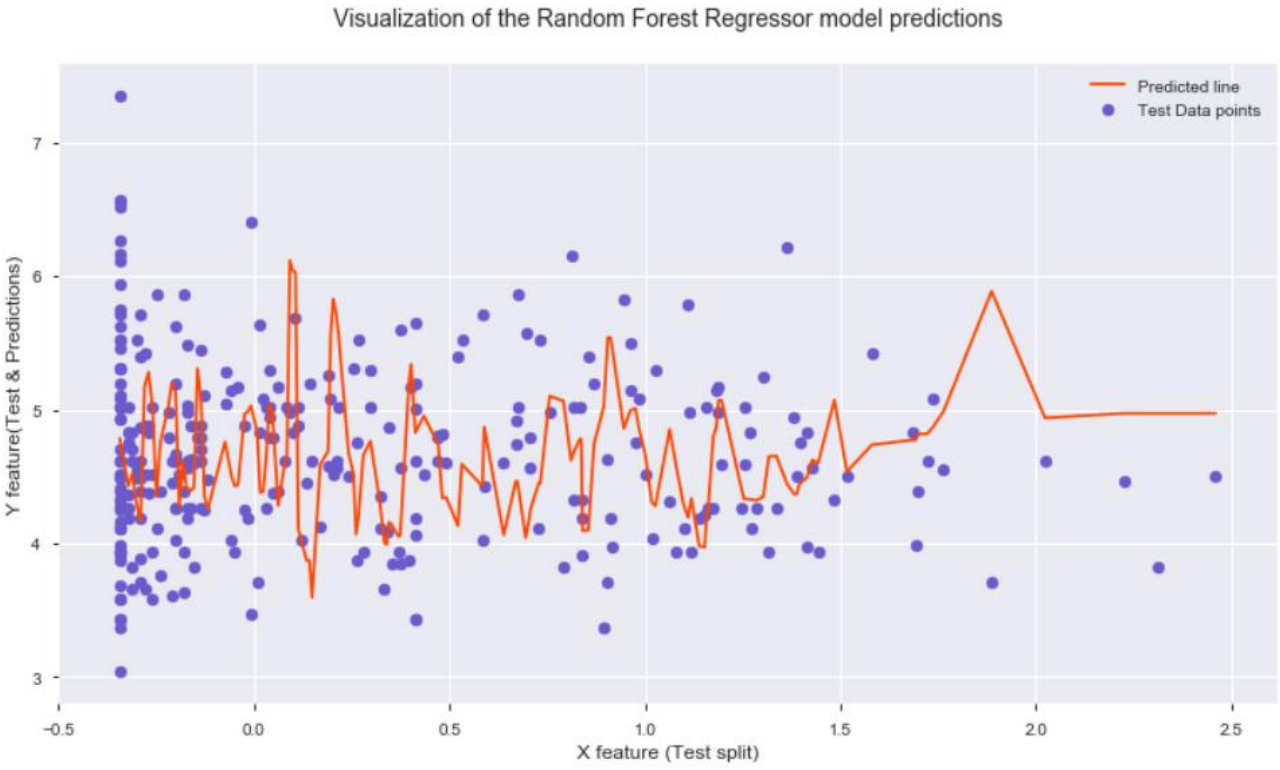
Metrics:

- Train Score (accuracy) : 0.5890
- Test Score (accuracy) : 0.6074
- Root Mean Squared Error : 0.4307
- R2 Score : 0.6074
- Mean Absolute Error : 0.3076

```
from sklearn.ensemble import RandomForestRegressor
train_score_list = []
test_score_list = []
cross_score_list=[]
for n in [100,200,300]:
    for d in [3,5,8,10,12,15,18,20,22,24]:
        rndtree = RandomForestRegressor(n_estimators=n,max_depth=d,
                                        random_state=41,
                                        bootstrap=True,n_jobs=-1)
        scores = cross_val_score(rndtree, x_train, y_train, cv=5)
        print('\nFor max_depth = ',d,' and n_estimators = ',n)
        cross_score_list.append(scores.mean())
        rndtree.fit(x_train,y_train)
        y_pred=rndtree.predict(x_test)
        train_score_list.append(rndtree.score(x_train,y_train))
        test_score_list.append(rndtree.score(x_test, y_test))
        print('\n\t•Training score without Cross validation: ',train_score_list[-1])
        print('\n\t•Cross validation Training score mean:',scores.mean())
        print('\n\t•Testing score without Cross validation: ',test_score_list[-1])
        print(f'\n\t• Root Mean Squared Error:
              {np.sqrt(mean_squared_error(y_test, y_pred))}
              \n\n\t• R2 Score: {r2_score(y_test,y_pred)}
              \n\n\t• Mean Absolute Error: {mean_absolute_error(y_test,y_pred)}')
        print('--------------------------------------------------------------------------
```



The best fitted line by the Random Forest Model



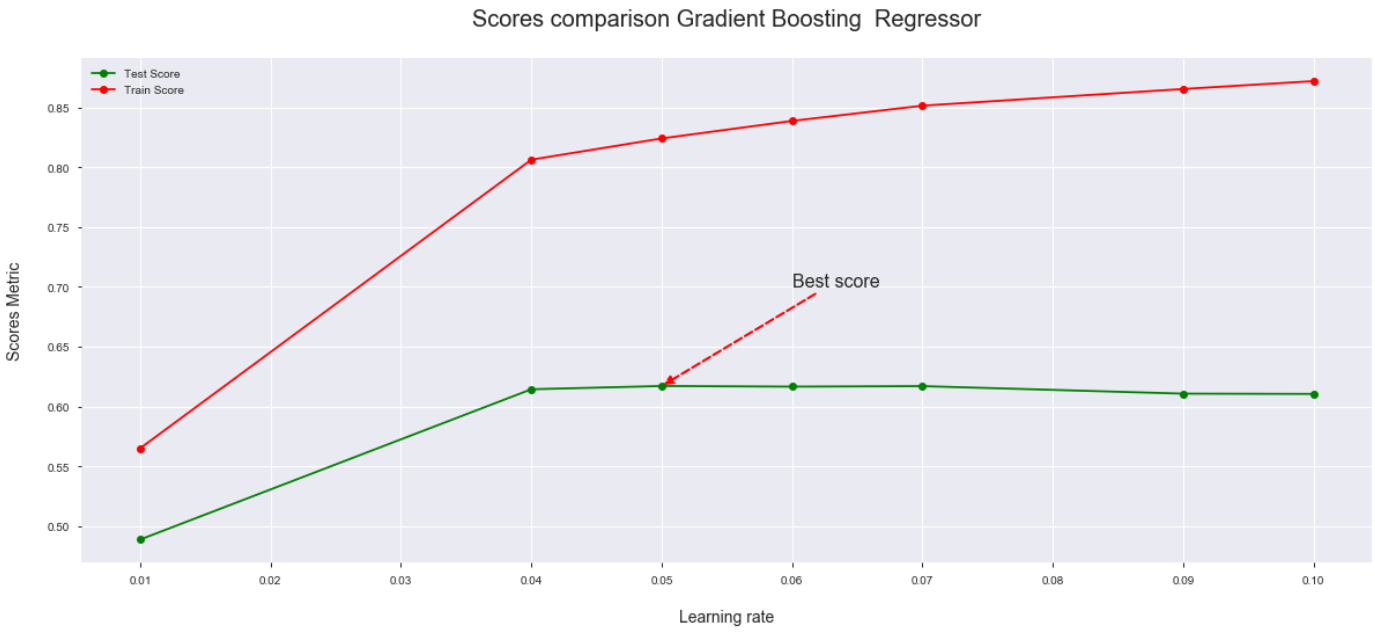
STOCHASTIC GRADIENT BOOSTING REGRESSOR

Best Parameters: { 'learning_rate' : 0.05 }

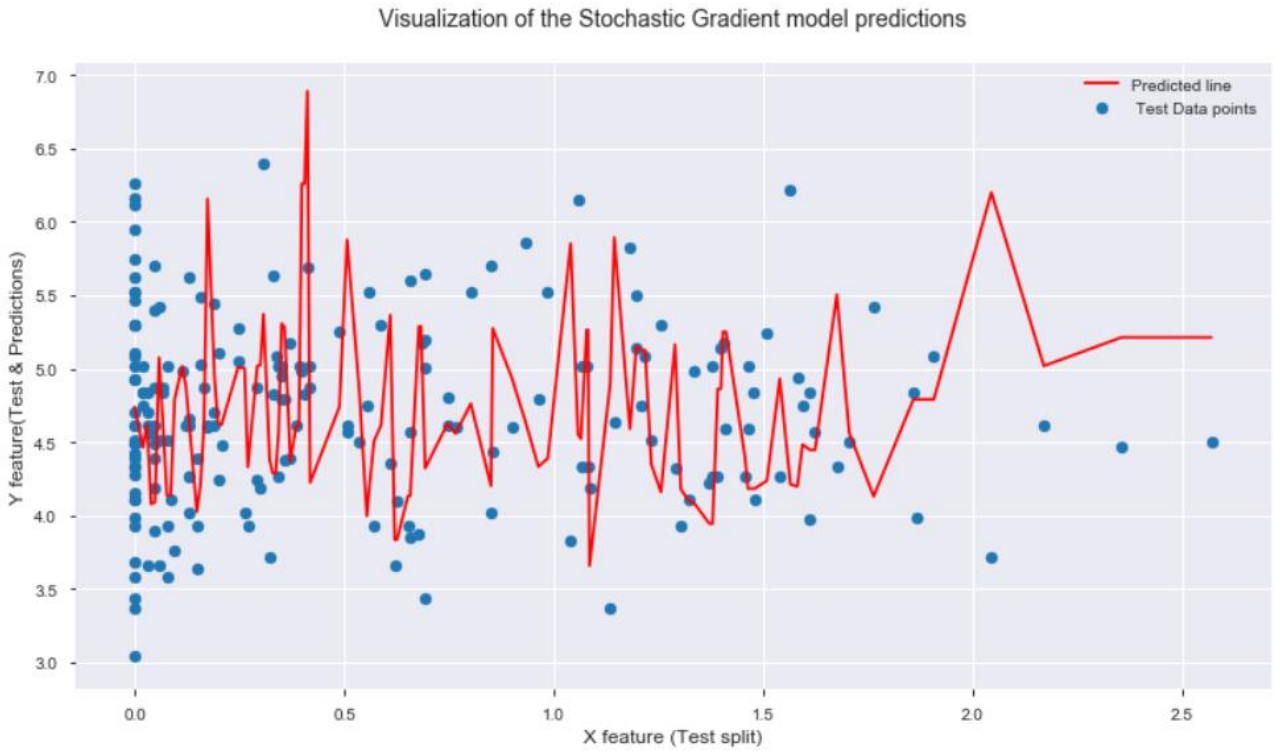
Metrics:

- Train Score (accuracy) : 0.8242
- Test Score (accuracy) : 0.6170
- Root Mean Squared Error : 0.4254
- R2 Score : 0.6170
- Mean Absolute Error : 0.3037

```
from sklearn.ensemble import GradientBoostingRegressor

for l in [0.01,0.04,0.05,0.06,0.07,0.09,0.1]:
    sgb=GradientBoostingRegressor(learning_rate=l,n_estimators=100,
                                  max_depth=15,subsample=0.9,random_state=999
                                  ,max_features='sqrt')
    #scores= cross_val_score(sgb, x_train, y_train, cv=3)
    print('\nFor n_estimator = ',100,' and For maxdepth =',15,'and Learning rate = ',l)
    #print('\nCross validation Testing mean:',scores.mean())
    sgb.fit(x_train,y_train)
    y_pred=sgb.predict(x_test)
    print('\nStochastic Gradient Boosting Metrics:\n\n\t•Training score :'
          ,sgb.score(x_train, y_train))
    print('\n\t•Test score :',sgb.score(x_test, y_test))
    print(f' \n\t• Mean Squared Error: {(mean_squared_error(y_test, y_pred))}
    \n\n\t• Root Mean Squared Error: {np.sqrt(mean_squared_error(y_test, y_pred))}
    \n\n\t• R2 Score: {r2_score(y_test,y_pred)} \n\n\t•
          Mean Absolute Error: {mean_absolute_error(y_test,y_pred)}')
    print('--------------------------------------------------------------------------------')
```



The best fitted line by the Gradient Boosting Model



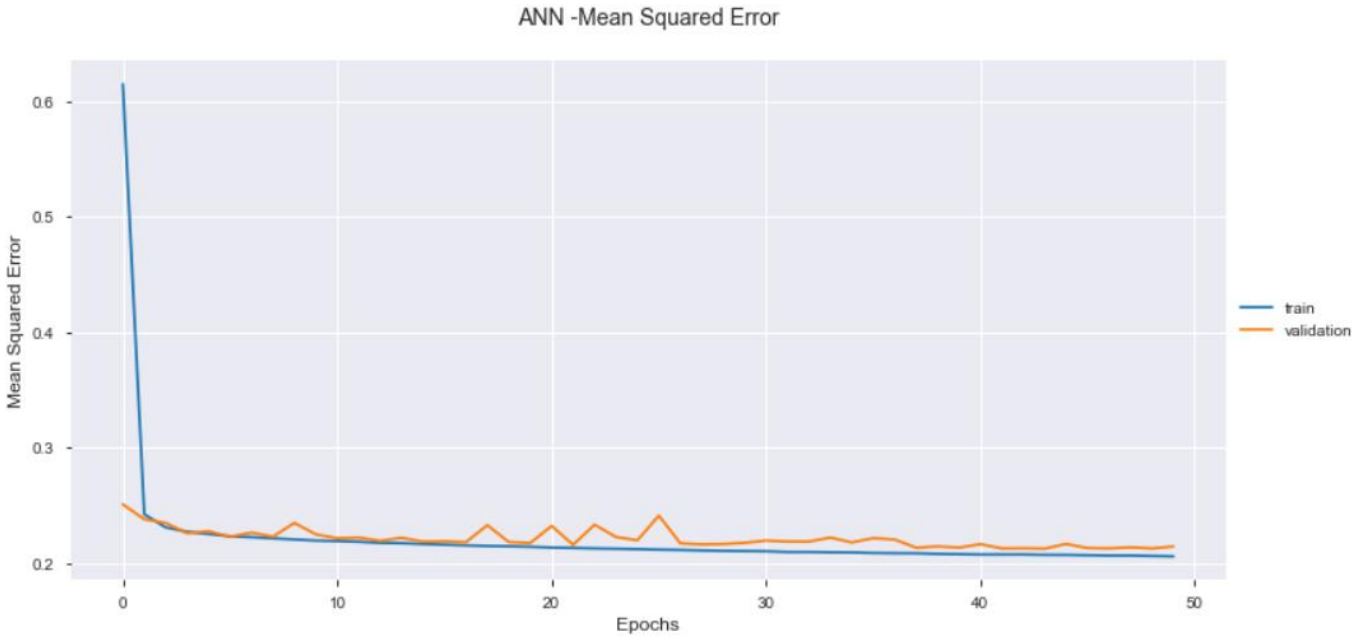
ARTIFICIAL NEURAL NETWORK

KERAS

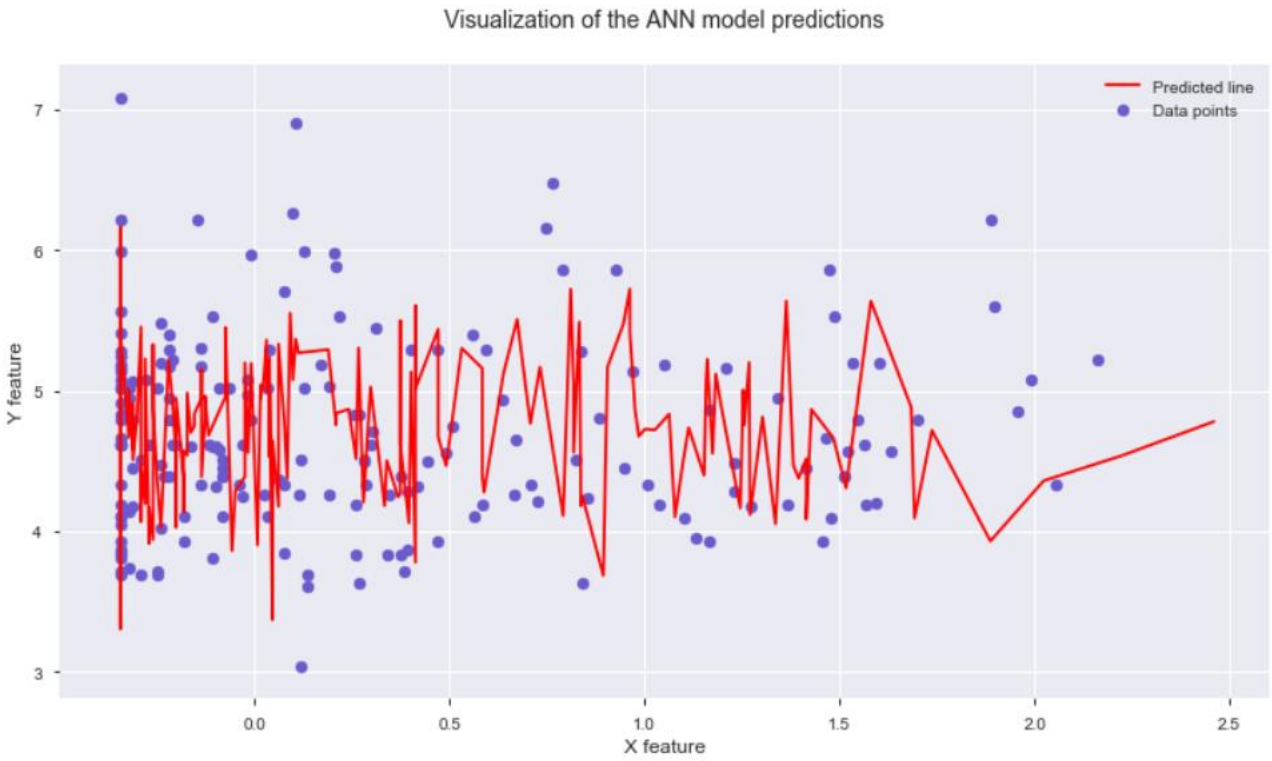
Metrics:

- Test Score (accuracy) : 0.5821
- Root Mean Squared Error : 0.4443
- R2 Score : 0.5821
- Mean Absolute Error : 0.3252

```
# Import `Sequential` from `keras.models`
import keras
from keras.models import Sequential
# Import `Dense` from `keras.layers`
from keras.layers import Dense
# Initialize the constructor
ann_model = Sequential()
# Add one input layer and first hidden layer
ann_model.add(Dense(50, activation='sigmoid', kernel_initializer='normal'
                    ,input_dim=236))
# Add second hidden layer
ann_model.add(Dense(40, activation='sigmoid',kernel_initializer='normal'))
ann_model.add(Dense(20, activation='sigmoid',kernel_initializer='normal'))
# Add an output layer
ann_model.add(Dense(1, activation="linear"))
ann_model.compile(loss='mean_squared_error',optimizer='adadelta',
                  metrics=['mse'])
ann_result=ann_model.fit(x_train_sc, y_train,epochs=50, batch_size=20,
                         verbose=1,validation_split=0.15)
```



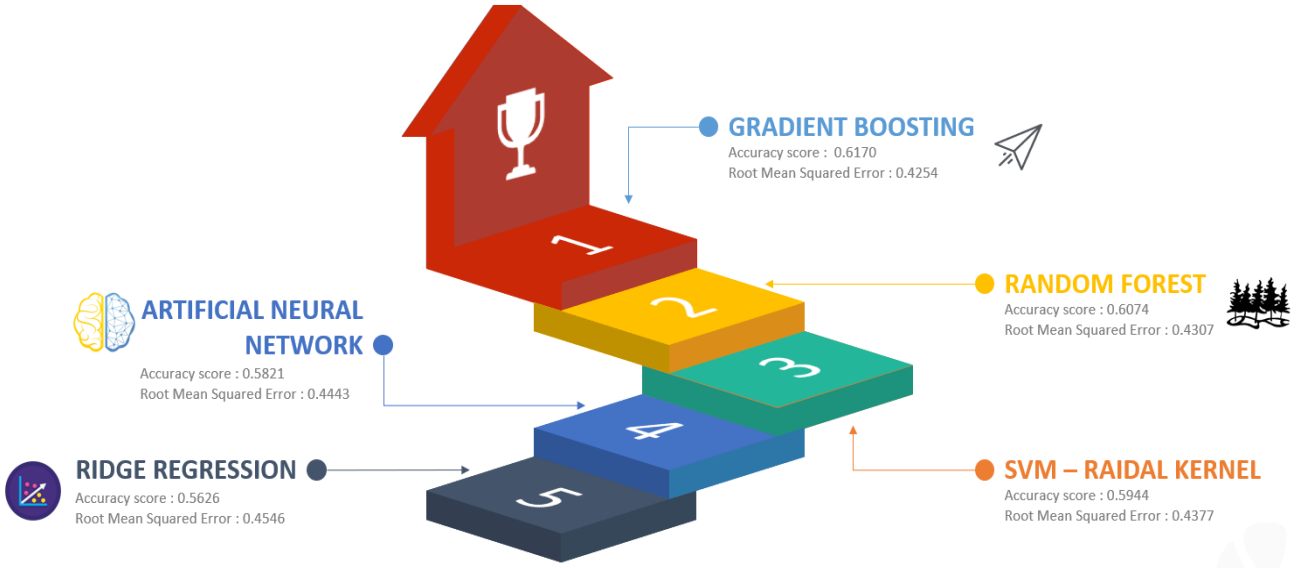
The best fitted line by the ANN Model



SCORES COMPARISON

Models	Train Accuracy	Test Accuracy	Root Mean Squared Error	R2 Score	Mean Absolute Error
KNN Regressor	0.5409	0.5626	0.6742	0.5626	0.3285
Linear Regression	0.554	-3.2041	11093.2266	-3.2041	1801289.535
Ridge Regression	0.5474	0.5626	0.4546	0.5626	0.3289
LASSO Regression	0.547	0.5622	0.4549	0.5622	0.3292
Polynomial Regression	0.6296	-4.94222E+11	483303.7485	-4.942E+11	21123.2228
Linear SVR	0.5381	0.5534	0.4593	0.5534	0.3243
SVR Radial Kernel	0.6105	0.5944	0.4377	0.5944	0.3068
Decision Tree	0.5352	0.5432	0.4646	0.5432	0.3311
Random Forest Regressor	0.589	0.6074	0.4307	0.6074	0.3076
Stochastic Gradient Boosting	0.8242	0.6170	0.4254	0.6170	0.3037
Artifical Neural Network	0.6032	0.5821	0.4443	0.5821	0.3252

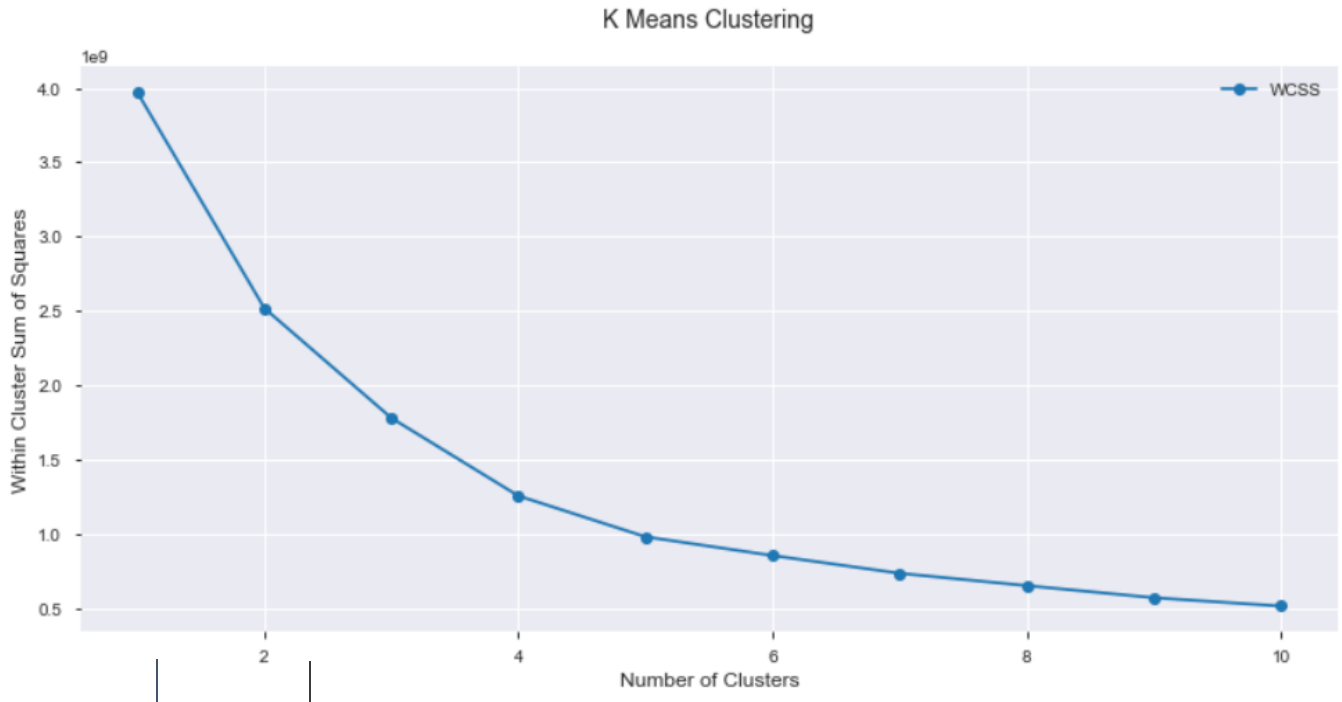
RANKING OF MODELS

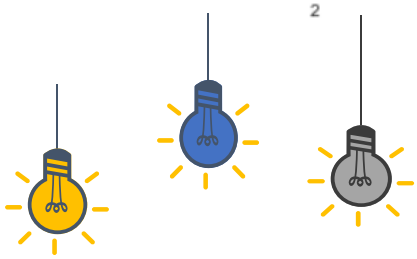


K MEANS CLUSTERING

```
In [11]:
from sklearn.cluster import KMeans
wcss=[]   #within cluster sum of squares
for k in range(1,11):
    kmc=KMeans(n_clusters=k,random_state=42)
    kmc.fit(kdf)
    wcss.append(kmc.inertia_)
```

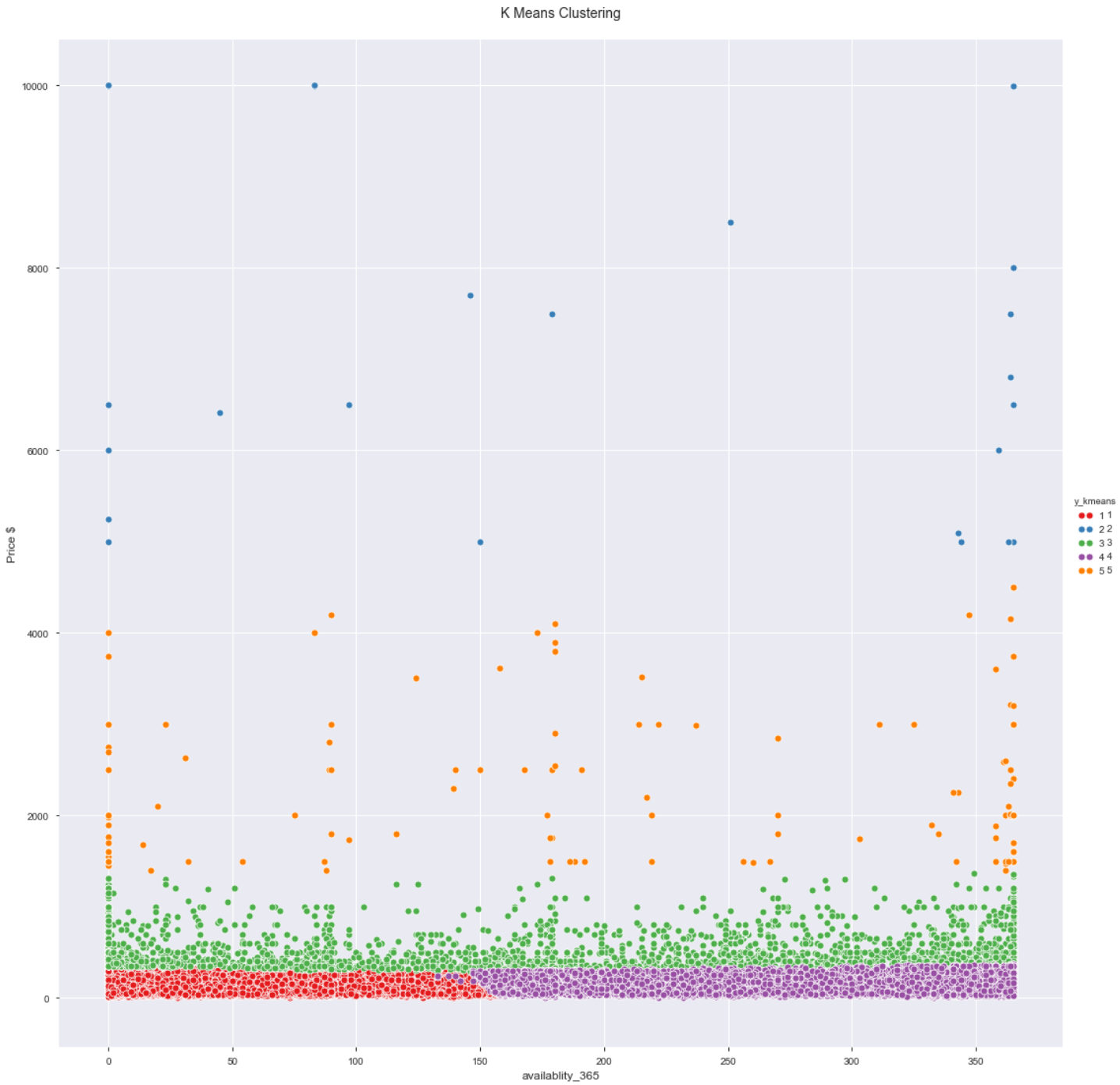
Optimal number of clusters: 5





In the plot we can see that the:

- **Cluster 1** is formed where we have availability less than 150 days and price ranges 0 to 400 dollars
- **Cluster 2** is having all the availability and price ranges between 5000 to 10000 dollars
- **Cluster 3** is having all the availability and price ranges 400 to 1400 dollars
- **Cluster 4** is having availability more than 150 days and price ranges 0 to 400 dollars
- **Cluster 5** is having all the availability and price ranges 1400 to 5000 dollars



Let's see the properties of each cluster based on each Features by using groupby and describe function

In [59]:

```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
display(tot.groupby(['y_kmeans']).describe().transpose())
```

		1	2	3	4	5
latitude	count	30832	26	2771	15118	148
	mean	40.72902245	40.72987308	40.73615508	40.7274414	40.73252568
	std	0.053798454	0.044031122	0.03888035	0.05831602	0.047306644
	min	40.50943	40.63952	40.53076	40.49979	40.57645
	25%	40.69067	40.699695	40.715795	40.6866325	40.71692
	50%	40.721515	40.72195	40.73795	40.723285	40.736115
	75%	40.7635425	40.768205	40.761125	40.76313	40.76051
	max	40.90734	40.82511	40.89338	40.91306	40.88671
longitude	count	30832	26	2771	15118	148
	mean	-73.95300702	-73.97362808	-73.97452523	-73.94610103	-73.97528196
	std	0.041392961	0.041383372	0.034175156	0.055045767	0.037202553
	min	-74.24442	-74.0973	-74.20295	-74.24285	-74.02884
	25%	-73.98151	-73.9951625	-73.99467	-73.98289	-73.9983975
	50%	-73.95511	-73.981075	-73.98137	-73.952315	-73.98275
	75%	-73.93803	-73.9538525	-73.962275	-73.92568	-73.96408
	max	-73.71795	-73.88128	-73.73874	-73.71299	-73.77069
price	count	30832	26	2771	15118	148
	mean	115.9383108	7087.269231	515.6062793	128.8053314	2245.777027
	std	64.73356263	1916.54125	198.0064228	75.04495713	787.800916
	min	0	5000	295	0	1395
	25%	65	5137.5	378.5	68	1599
	50%	100	6500	450	110	2000
	75%	150	8375	599	177	2606.5
	max	345	10000	1369	370	4500

(Half the table only pasted)