UNIVERSITÀ
DEGLI STUDI
DI MILANO

LA STATALE

Department od Economics, Management and Quantitative Methods

Università degli Studi di Milano

Data science and economics

Statistical Learning Project

Analysis and prediction models for New York City Airbnb

Report

Author:

Andrea Ierardi

Academic year 2019-2020

Contents

List of Figures

1

Abstract

The aim of the project is to analyse, develop prediction models and define data clusters from the ”New York City Airbnb Open Data” from a Kaggle competition.
In particular, one part is focused on the development of predictive models to forecast house prices using these Supervised Learning technics:

- Linear Regression
- Decision Tree
- Random Forest
- Ranger Random Forest
- Neural Newtworks

For each of these, a comparison between the Mean Square Error and between all R^2 measure of all methods has been made to highlight which have the best performance. Also, for training all the models, a partition of the dataset in three parts has been applied: entire dataset, filtered by neighboorhood group and filtered for neighboorhood group and room type. In this way, it is possible to check the perfomance giving less or more features in input.

The second part is focused on the cluster and data reduction technics using these Unsupervised Learning technics:

- K-means Algorithm
- Clustering for mixed-type data
- Principal Component Analysis

2

Problem Definition and Algorithm

2.1 Two main Goals

2.1.1 Develop predictive models for price

The first objective is the forecast of the prices given some input information. This could be usefull for a lot of scenarios. For example from a AirBnB customer point of view, he/she would like to get the list of houses more in alignment with his/her preference choice; or for a host point of view, where given the position and other information he/she could get information about the possible per day price of his property in New York City.

2.1.2 Define clusters and groups

The second objective is the definition of group or partition between the houses with different characteristics. For a user point of view could be usefull to have information about avaiable houses similar to those booked in the past. This could be also usefull after the booking for a suggestion analysis having the information about last booked houses in NewYork or houses in similar cities around the world.

2.2 Algorithms

2.2.1 Linear Regression

Linear regression is a linear approach to modeling the relationship between a dependent variable and one or more independent variables.

2.2.2 Decison Trees

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making.

2.2.3 Random Forest

Random forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees.

2.2.4 Ranger Random Forest

Ranger is a fast implementation of random forests or recursive partitioning, particularly suited for high dimensional data.

2.2.5 Neural Networks

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns.

2.2.6 K-means

K-means is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean.

2.2.7 Principal Component Analysis

Principal Component Analysis (PCA) is mostly used as a tool in exploratory data analysis and for making predictive models.

3

Experimental Evaluation

3.1 Methodology

3.1.1 Data Inspection

The dataset is part of a Kaggle competition, called the New York City Airbnb Open Data. It contains 48.000 rows per 16 columns. The dataset is structured with these columns:

- **id**
- **name**: name of the listing
- **host_id**
- **host_name**
- **neighbourhood_group**: location
- **neighbourhood**: area
- **latitude**: coordinates
- **longitude**: coordinates
- **room_type**: space type
- **price**: in dollars
- **minimum_nights**: amount of nights minimum
- **number_of_reviews**: number of reviews
- **last_review**: latest review
- **reviews_per_month**: number of reviews per month
- **calculated_host_listings_count**: amount of listing per host
- **availability_365**: number of days when listing is available for booking

It has been select only 5 of these variables: price, latitude, longitude, neighbourhood_group and room_type. The reason is that it is reasonable to select them to predict the prices and to obtain different clusters. The position and the neighbourhood is important since if a property is positioned near the city center will have a higher price with respect to those situated in the outskirts; also the type of room, since an entire apartment will cost more than a single room.

From the Figure 1 is possible to see the distribution of the price. The minimum price is 0 and the maximum is 10000$. It is not possible to rent a house for free, so it is possible to filter the price with a price higher than 15$. It is possible that a luxury house cost a lot per day, but these value can not be consider in the model, instead they are outliers and for this reason it is convenient to filter the price again and take those that have a value lowert than 500$. The reason is that the third quantile has a price of 175$ which is a far from 10000$ (Figure 3). It has also been checked the null and missing value in the dataset and has not been found apart from the reviews_per_month column. It is not a problem, since this feature has been not taken in account to train the models.

```
     price
Min.   :    0.0
1st Qu.:   69.0
Median :  106.0
Mean   :  152.7
3rd Qu.:  175.0
Max.   :10000.0
```

Figure 1: Price summary

From Figure 2, it is possible to see the distribution of all houses in New York City and the price. This picture is not really informative since it can be noticed that the prices for the most part are in the range 0-500$ and only a low number of istances have a price greater than 500$. Deleting the outliers, the Figure 3 is more informative than the one before.

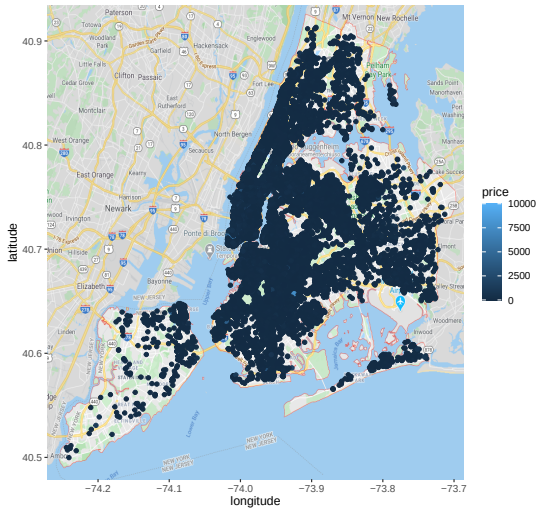


Figure 2: Distribution of all houses in NY colored by prices

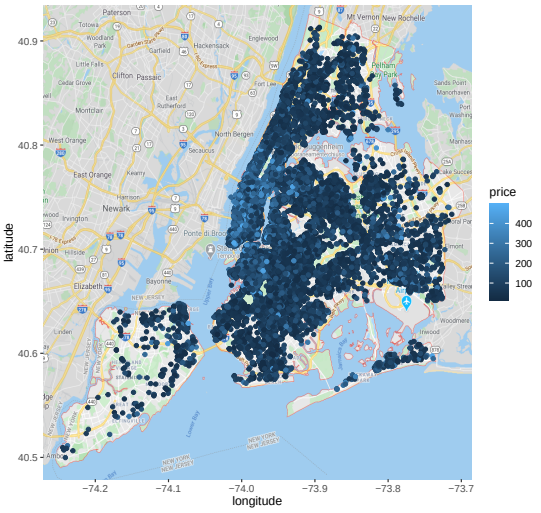


Figure 3: Distribution of all houses in NY of price between 15$ and 500$ per day

3.1.2 Data Cleaning and Pre-processing

The dataset has approximately 48.000 rows for each column, so an important part of the work is related to the pre-processing due to the large amount of data. For this reason, some variable has been rescaled to let the models to learn faster and better and to perform a better prediction. Latitude and longitude have not been rescaled for forecast price model, while for clustering methods to have a consistent distance calculation. Categorical variables have also been rescaled assigning a numerical value to each category, resulting as unordered factors.
The selected categorical variables are: neighbourhood_group and room type.
The selected numerical variables are: latitude, longitude and price.

Considering that every neighbourhood group has his characteristic, also taking in account the room type that could impact largerly the price. It is possible to define of different subset of the original set and try different methods of forecast. This is due to the fact that a customer should choose which of the different neighbourhood and room type is interested in. Models have been run to different scenarios (Figure 4): users interested in all New York City houses and all type of room, users interested only in a single neighbourhood and users interested in a single neighbourhood and a single room type.
Including different scenario could be potentially computationally expensive. In reality, the training proceeded without any problem. [1] Computantional problem may emerge for the case

[1]The models have been trained on a machine with quad-core 3.5 GHz processor, 8 Gb RAM and 4 Gb dedicated GPU.

of hyperparametrisation tuning, even using multiprocessing and multithreading technics. For semplicity, in this project no model tuning have been made.
Also for clustering have been filter every scenario, but not for the case of Hierarchical Clustering because it need to generate a readable dendogram using the mean of all neighbourhood or room type price, Principal Component Analysis since avaiable data are really small for some specific neighbourhood like Staten Island and specific room type.

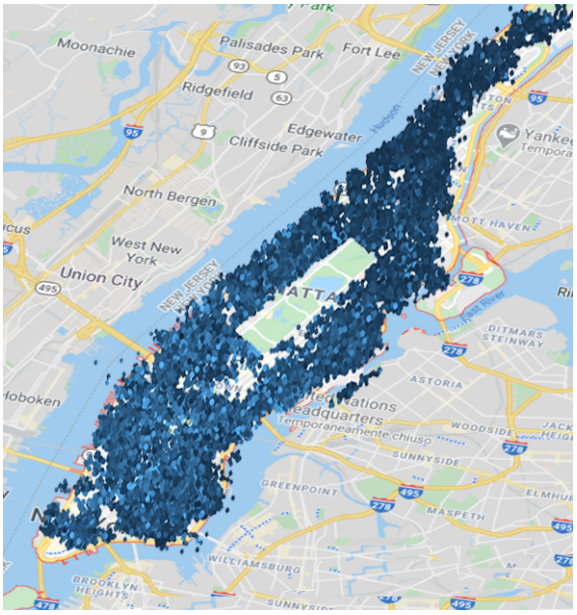


Figure 4: Approximated distribution map of all houses in Manhattan

3.2 Results

Since the number of models trained are very high, the study result will be presented for different macro groups that depend on the different subset of the dataset: entire dataset, filtering by neighborhood and filtering by neighborhood and room type. This is due to the fact that neighborhood in general have similar result (like Manhattan has similar shape of Brooklyn and Staten Island has a similar shape of Bronx).

===== DA RIVEDERE ===

Subset	Linear.Regression	Decision.Tree	Random.Forest	Ranger.Random.Forest	Neural.Networks
Brooklyn	0.4351730	0.4395665	0.4182327	0.4141652	0.4669738
Manhattan	0.7936143	0.7964149	0.7468374	0.7472513	0.8750774
Queens	0.3721530	0.3699898	0.3513738	0.3515902	0.5491319
Staten Island	0.4776541	0.5173908	0.4873455	0.4956109	1.3730607
Bronx	0.4040408	0.4073877	0.4001876	0.4032073	0.4710128
Brooklyn/Private room	0.1743968	0.1691098	0.1791112	0.1785046	0.2344717
Brooklyn/Entire home/apt	0.7262980	0.7179003	0.7699032	0.7713528	0.7935944
Brooklyn/Shared room	0.2772952	0.3043294	0.2515028	0.2533654	0.2879471
Manhattan/Private room	0.4372064	0.3882155	0.3733743	0.3744351	0.5208271
Manhattan/Entire home/apt	1.0114706	1.0174490	1.0533165	1.0552250	1.1390734
Manhattan/Shared room	0.4904792	0.4992158	0.5619616	0.5622654	0.5155457
Queens/Private room	0.1680487	0.1600965	0.1625676	0.1625325	0.4625950
Queens/Entire home/apt	0.6817353	0.6590635	0.6514473	0.6489757	0.7061921
Queens/Shared room	0.0709897	0.2989787	0.2047612	0.2114044	2.0645615
Staten Island/Private room	0.2640100	0.2339327	0.2291305	0.2252806	0.2557696
Staten Island/Entire home/apt	0.5703345	0.7737648	0.7048053	0.7076976	0.6449374
Staten Island/Shared room	0.3209208	0.5818964	0.3060744	0.5923555	0.5363675
Bronx/Private room	0.1475827	0.1611146	0.1330419	0.1341573	0.4504089
Bronx/Entire home/apt	0.5627967	0.7622148	0.6439600	0.6650687	1.1799774
Bronx/Shared room	0.0690389	0.0489295	0.0608120	0.0536876	0.2355241
All	0.6129722	0.6053303	0.5819694	0.5463735	0.6369116

Figure 5: Summary of the MSE for all the models for each subset of the dataset

3.2.1 Linear Regression Results

Linear Regression on the entire dataset

The results on the entire dataset are acceptable. All variables are significant for the model and the R^2 value is 40% (Figure 7). The Mean Square Error (Figure 5) has a value of 0.6 which is acceptable. All the neighboorhood groups have a positive effect on the price apart for the 4^th one. This could be due to the fact that Staten Island does not have expensive houses compared to the other group.

Also the latitude and longitude have a sligh negative effect on the price. Entire apartment have a slight positive effect while the shared house negative. This could be due to the fact that an entire house will cost more than a shared room, so the price will increase for this type of room.

```
[1] "========== all =========="

Call:
lm(formula = price ~ ., data = trains[[sub]])

Residuals:
    Min      1Q  Median      3Q     Max
-2.1739 -0.4434 -0.1419  0.2269  4.9515

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)             -1.916e+02  1.420e+01 -13.497  < 2e-16 ***
neighbourhood_group2     5.023e-01  1.618e-02  31.041  < 2e-16 ***
neighbourhood_group3     2.216e-01  1.978e-02  11.202  < 2e-16 ***
neighbourhood_group4    -9.535e-01  5.882e-02 -16.211  < 2e-16 ***
neighbourhood_group5     2.687e-01  3.914e-02   6.865 6.81e-12 ***
latitude                -1.726e+00  1.393e-01 -12.385  < 2e-16 ***
longitude               -3.532e+00  1.595e-01 -22.145  < 2e-16 ***
room_type2               9.996e-01  9.622e-03 103.887  < 2e-16 ***
room_type3              -2.379e-01  3.068e-02  -7.754 9.18e-15 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.784 on 28680 degrees of freedom
Multiple R-squared:  0.3914,    Adjusted R-squared:  0.3912
F-statistic:  2306 on 8 and 28680 DF,  p-value: < 2.2e-16
```

Figure 6: Linear Regression output for the entire dataset

Linear Regression for specific Neighboorhood group

Models give different results for the specific neighboorhood group. For the first three (Brooklyn, Manhattan and Queens) R^2 are over approximately 30%. For the MSE, the value have different ranges based on the distribution of prices in the singolar neighboorhood. To be noticed, is the fact that MSE of Brooklyn is significantly lower with respect to Manhattan, given the have a similar number of houses and have almost the same price distribution. The only difference between the two is the fact that Manhattan has more entire apartment type of room. All variables have a significant effect on the price apart for the latitude in Manhattan and longitude for Queens with a 0.1.

For the last two groups (Staten Island and Bronx) all variables does not have significance in the response variable apart for the Entire Apartment dummy which have a positive effect. This is due to the fact that entire properties will cost more than shared rooms.

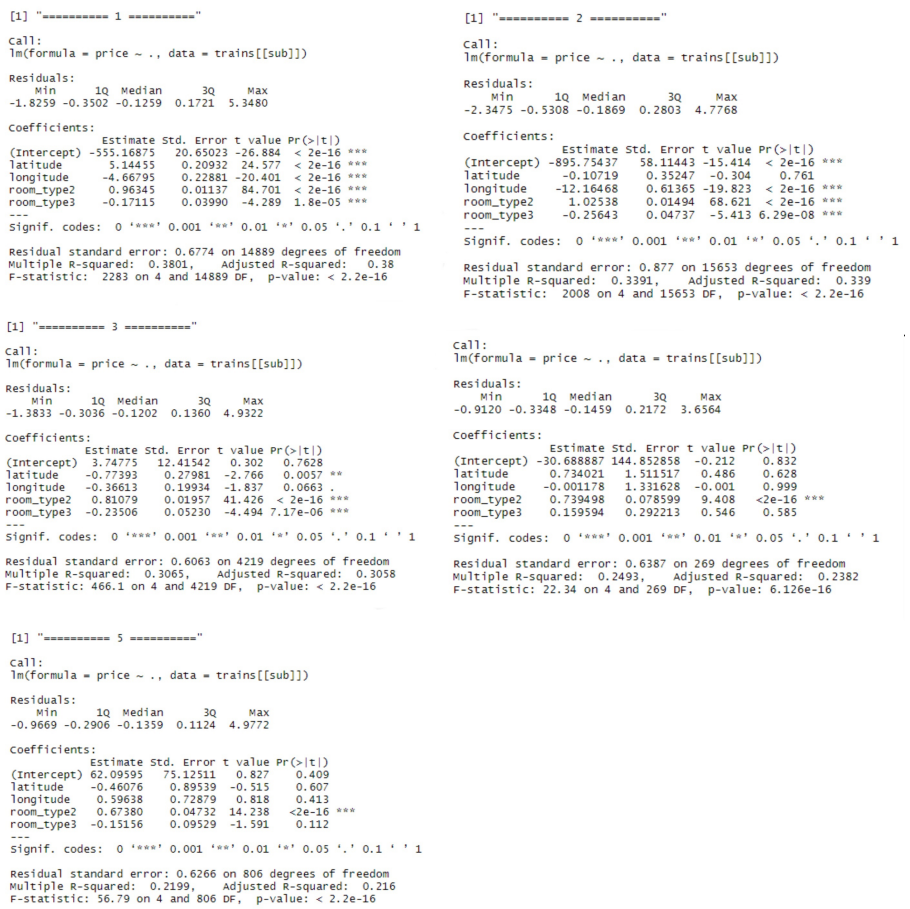


Figure 7: Linear Regression output filtering by Manhattan

Linear Regression for specific Neighboorhood group and room type

Adding a filter also for the room type, models give acceptable results in some cases and not in others. (Figure 8) For the larger subset the model confirm the significant of the variables, apart for latitude in Manhattan such as happened in the model using this neighboorhood data. In Manhattan with Entire apartment model, instead, all variables have a significant effect. For the remaining neighboorhood groups the variables do not result as significant in the model. This could be due to the fact that the subset obtained from this filtering are not large (they have less than 300 rows). Also in all model the R^2 result to be lower than 10% but this depends on the fact to train they have less information.

Also in this model the MSE of Brooklyn is lower in all the type of room type than the Manhattan MSE.

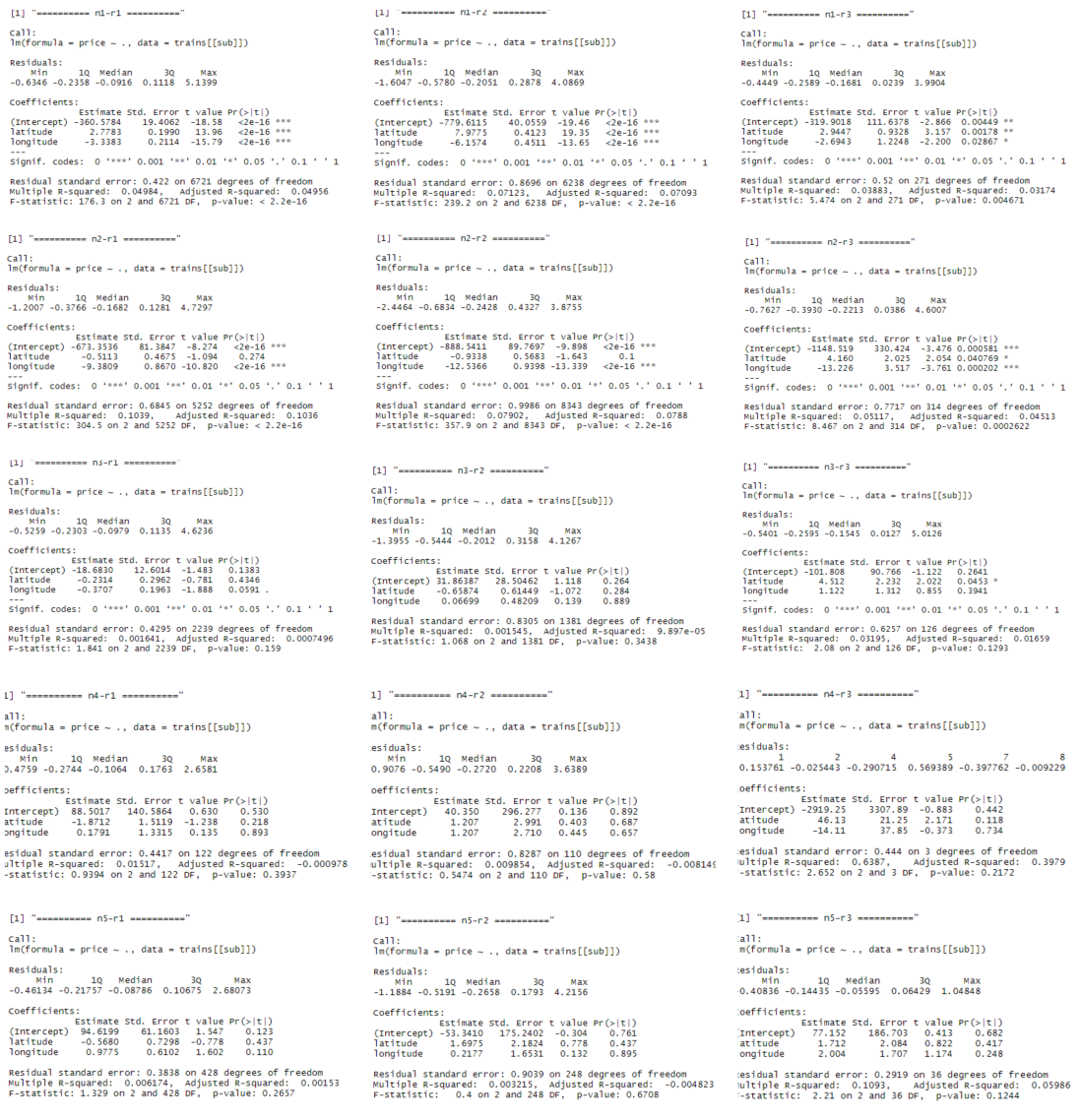


Figure 8: Linear Regression output filtering by Manhattan and Entire home/Apartment

3.2.2 Decison Trees Results

Decison tree without filters

For the entire dataset the MSE has a value of 0.605 (Figure 9) Important to be notice is the fact that the neighboorhood group variable have not been used in the construction of the tree. Moreover, what really influence the price is the type of room (Figure 10).

```
[1] "========== all =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Variables actually used in tree construction:
[1] "room_type" "longitude" "latitude"
Number of terminal nodes:  5
Residual mean deviance:  0.6053 = 17360 / 28680
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.2750 -0.4143 -0.1493  0.0000  0.2142  4.8720
[1] 0.6053303
```

Figure 9: Decision Tree results on the entire dataset

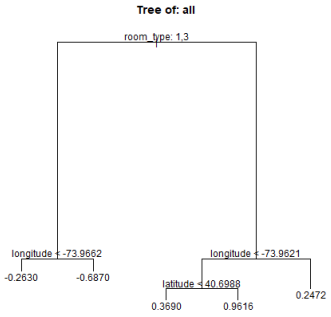


Figure 10: Tree generated for the entire dataset

Decison tree for specific Neighboorhood group

For the specific neighboorhood group decision tree models give output similar MSE results with respect to the Linear Regression. Also in this case, the latitude for Manhattan has not been used in the model to predict the price (Figure 11).

```
[1] "========== 1 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  4
Residual mean deviance:  0.4618 = 6876 / 14890
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.9220 -0.3540 -0.1268  0.0000  0.1630  4.8710
[1] 0.4395665


[1] "========== 2 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Variables actually used in tree construction:
[1] "room_type" "longitude"
Number of terminal nodes:  4
Residual mean deviance:  0.7757 = 12140 / 15650
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.2950 -0.5138 -0.1934  0.0000  0.3072  4.7680
[1] 0.7964149
```

```
[1] "========== 3 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  4
Residual mean deviance:  0.3588 = 1514 / 4220
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.4870 -0.3017 -0.1276  0.0000  0.1564  4.9280
[1] 0.3699898


[1] "========== 4 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  11
Residual mean deviance:  0.2805 = 73.77 / 263
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.60400 -0.25800 -0.08494  0.00000  0.14370  2.82700
[1] 0.5173908


[1] "========== 5 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Variables actually used in tree construction:
[1] "room_type"
Number of terminal nodes:  2
Residual mean deviance:  0.3482 = 281.7 / 809
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.0760 -0.2764 -0.1060  0.0000  0.1212  5.0060
[1] 0.4073877
```

Figure 11: Decision tree results for specific neighboorhood group

Decison tree for specific Neighboorhood group and room type

For neighboorhood group and room price Decision tree construction was not possible for all the subset. For example for shared room for Bronx were not possible to plot the tree since the model had only one node. MSE are similar to Linear Regression but for this type of subset perform slighly worse. This is due to the lack of information of some of these subsets (Figure 12).

```
[1] "========== n1-r1 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  4
Residual mean deviance:  0.1783 = 1198 / 6720
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.75440 -0.22280 -0.07515  0.00000  0.11800  4.94600
[1] 0.1691098


[1] "========== n1-r2 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  5
Residual mean deviance:  0.7349 = 4583 / 6236
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.7400 -0.5756 -0.1893  0.0000  0.2752  4.3100
[1] 0.7179003


[1] "========== n1-r3 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  4
Residual mean deviance:  0.1571 = 42.25 / 269
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.87700 -0.16430 -0.08477  0.00000  0.02883  4.06200
[1] 0.3043294


[1] "========== n2-r1 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  8
Residual mean deviance:  0.3902 = 2048 / 5248
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.5970 -0.3301 -0.1267  0.0000  0.1243  4.8870
[1] 0.3882155


[1] "========== n2-r2 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  3
Residual mean deviance:  0.9637 = 8040 / 8343
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.3220 -0.6882 -0.2203  0.0000  0.4045  3.8560
[1] 1.017449


[1] "========== n2-r3 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  10
Residual mean deviance:  0.4952 = 151.5 / 306
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.6830 -0.2992 -0.1347  0.0000  0.0506  3.5570
[1] 0.4992158


[1] "========== n5-r1 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  5
Residual mean deviance:  0.163 = 69.6 / 427
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.35200 -0.21400 -0.06405  0.00000  0.12680  3.81700
[1] 0.1611146


[1] "========== n5-r2 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  8
Residual mean deviance:  0.6961 = 169.1 / 243
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.2490 -0.4638 -0.1854  0.0000  0.2079  3.7900
[1] 0.7622148


[1] "========== n5-r3 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Variables actually used in tree construction:
[1] "latitude"
Number of terminal nodes:  6
Residual mean deviance:  0.1029 = 3.5 / 34
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.52540 -0.15660 -0.04544  0.00000  0.04203  0.92870
[1] 0.04892947


[1] "========== n3-r1 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  3
Residual mean deviance:  0.1738 = 389.1 / 2239
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.85200 -0.22640 -0.08418  0.00000  0.11440  4.94200
[1] 0.1600965


[1] "========== n3-r2 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  4
Residual mean deviance:  0.6597 = 910.4 / 1380
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.9190 -0.5029 -0.2189  0.0000  0.2876  4.1550
[1] 0.6590635


[1] "========== n3-r3 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  6
Residual mean deviance:  0.2555 = 31.43 / 123
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.81800 -0.15550 -0.07597  0.00000  0.05964  3.40800
[1] 0.2989787


[1] "========== n4-r1 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  12
Residual mean deviance:  0.08816 = 9.962 / 113
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.56420 -0.14540 -0.04604  0.00000  0.12440  1.31700
[1] 0.2339327


[1] "========== n4-r2 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Number of terminal nodes:  6
Residual mean deviance:  0.5117 = 54.76 / 107
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.7900 -0.3757 -0.1485  0.0000  0.3059  2.4700
[1] 0.7737648


[1] "========== n4-r3 =========="

Regression tree:
tree(formula = price ~ ., data = trains[[sub]])
Variables actually used in tree construction:
character(0)
Number of terminal nodes:  1
Residual mean deviance:  0.09365 = 0.3746 / 4
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-0.2931 -0.1908 -0.1795  0.0000  0.3317  0.3317
Not possible to plot tree:  n4-r3[1] 0.5818964
```

Figure 12: Decision tree results for specific neighboorhood group and room type

3.2.3 Random Forest Results

Random Forest on the entire dataset

Random Forest on the entire dataset outputs an explained variance of 42% which is acceptable and also the MSE is lower with respect to the models before.

```
[1] "========== all =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.5754847
                    % Var explained: 42.18
[1] "MSE: 0.578656097325552"
```

Figure 13: Random Forest results on the entire dataset

Random Forest for specific Neighboorhood group

Random Forest on specific neighboorhood outputs overall MSE lower than the last two models. Explained variance are acceptable: for Brooklyn and Manhattan is approximately around 40%, 37% for Queens and about 25% for Staten Island and Bronx.

```
[1] "========== 1 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.4407074
                    % Var explained: 40.45
[1] "MSE: 0.418992779004395"


[1] "========== 2 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.72859
                    % Var explained: 37.38
[1] "MSE: 0.749899501079486"
```

```
[1] "========== 3 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.3474919
                    % Var explained: 34.37
[1] "MSE: 0.352243050175804"


[1] "========== 4 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.3650865
                    % Var explained: 23.27
[1] "MSE: 0.484112832735206"


[1] "========== 5 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.3513488
                    % Var explained: 24.72
[1] "MSE: 0.396692012991184"
```

Figure 14: Random Forest results for specific neighboorhood group

Random Forest for specific Neighboorhood group and room type

Random Forest for specific neighboorhood and room type does perform poorly since in some cases explained variance is negative. This could be due to the fact of overfitting, but in this case is more probable the problem is underfitting since in some specific case there is a low number of information.

```
[1] "========== n1-r1 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.1885908
                    % Var explained: 2.21
[1] "MSE: 0.1785959929437"


[1] "========== n1-r2 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.8016605
                    % Var explained: 1.5
[1] "MSE: 0.768164722382269"


[1] "========== n1-r3 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.1849337
                    % Var explained: 4.42
[1] "MSE: 0.253230897709792"


[1] "========== n2-r1 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.4015436
                    % Var explained: 23.75
[1] "MSE: 0.373153844299809"


[1] "========== n2-r2 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.9846148
                    % Var explained: 5.71
[1] "MSE: 1.05375292497579"


[1] "========== n2-r3 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.7182231
                    % Var explained: -9.47
[1] "MSE: 0.565461676852938"


[1] "========== n3-r1 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.1710755
                    % Var explained: 5.65
[1] "MSE: 0.162824504191009"


[1] "========== n3-r2 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.7120831
                    % Var explained: -3.14
[1] "MSE: 0.647736155624821"


[1] "========== n3-r3 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.4336741
                    % Var explained: -9.8
[1] "MSE: 0.216479634090822"


[1] "========== n4-r1 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.1289314
                    % Var explained: -4.61
[1] "MSE: 0.229843237622333"


[1] "========== n4-r2 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.7779035
                    % Var explained: -15.23
[1] "MSE: 0.71075187666878"


[1] "========== n4-r3 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.01837979
                    % Var explained: 75.47
[1] "MSE: 0.304203607022542"


[1] "========== n5-r1 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.2107108
                    % Var explained: -15.87
[1] "MSE: 0.134361647516686"


[1] "========== n5-r2 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.902801
                    % Var explained: -11.48
[1] "MSE: 0.661500485887416"


[1] "========== n5-r3 =========="

Call:
 randomForest(formula = price ~ ., data = trains[[sub]])
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.198373
                    % Var explained: -39.67
[1] "MSE: 0.056795238786339"
```

Figure 15: Random Forest results for specific neighboorhood group and room type

3.2.4 Ranger Random Forest Results

Ranger Random Forest is known to be computationally light with respect to the classic Random Forest. In fact, for the tuning part there were not problem in running it for the entire dataset.

Ranger on the entire dataset

Ranger outputs for the entire dataset are consistent. We have a R^2 of 0.47 and OOB error of .

```
[1] "========== all =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      28689
Number of independent variables:  4
Mtry:                             2
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.548326
R squared (OOB):                  0.4568823
[1] "MSE: 0.542787782904249"
```

Figure 16: Ranger Random Forest results on the entire dataset

Ranger for specific Neighboorhood group

```
[1] "========== 1 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      14894
Number of independent variables:  3
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.4348521
R squared (OOB):                  0.415184
[1] "MSE: 0.423576642914145"


[1] "========== 2 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      15657
Number of independent variables:  3
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.7304504
R squared (OOB):                  0.3714795
[1] "MSE: 0.736503144251037"


[1] "========== 3 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      4224
Number of independent variables:  3
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.3458987
R squared (OOB):                  0.3468607
[1] "MSE: 0.351308553169474"
```

```
[1] "========== 4 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      275
Number of independent variables:  3
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.4478808
R squared (OOB):                  0.1898886
[1] "MSE: 0.253040394051809"


[1] "========== 5 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      811
Number of independent variables:  3
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.3959689
R squared (OOB):                  0.2091952
[1] "MSE: 0.270598776462825"
```

Figure 17: Ranger Random Forest results for specific neighboorhood group

Ranger for specific Neighboorhood group and room type

The dataset filtered by neighboorhood and room type gives as result a R^2 of 0.25.

```
[1] "========== n1-r1 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      6724
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.1858609
R squared (OOB):                  0.01750151
[1] "MSE: 0.183555338342419"


[1] "========== n1-r2 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      6241
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.7797858
R squared (OOB):                  0.02281908
[1] "MSE: 0.805082950282569"


[1] "========== n1-r3 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      274
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.2712858
R squared (OOB):                  0.02842518
[1] "MSE: 0.0886830072194648"

[1] "========== n3-r1 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      2242
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.1718005
R squared (OOB):                  0.05295275
[1] "MSE: 0.162340649924091"


[1] "========== n3-r2 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      1384
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.7131906
R squared (OOB):                  -0.03403514
[1] "MSE: 0.6565262152989"


[1] "========== n3-r3 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      129
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.4190039
R squared (OOB):                  -0.0526597
[1] "MSE: 0.198349635747252"
```

```
[1] "========== n2-r1 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      5255
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.4009668
R squared (OOB):                  0.2328408
[1] "MSE: 0.378149521519247"


[1] "========== n2-r2 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      8346
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       1.035405
R squared (OOB):                  0.04343341
[1] "MSE: 0.976143348838547"


[1] "========== n2-r3 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      317
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.5746933
R squared (OOB):                  -0.07013213
[1] "MSE: 0.741990634474665"
```

```
[1] "========== n4-r1 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      125
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.1750849
R squared (OOB):                  0.1016101
[1] "MSE: 0.134701396762548"


[1] "========== n4-r2 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      113
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.5325296
R squared (OOB):                  -0.1367216
[1] "MSE: 1.07190377323593"


[1] "========== n4-r3 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      5
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.1159859
R squared (OOB):                  -0.2385456
[1] "MSE: 0.576764441356849"
```

```
[1] "========== n5-r1 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      431
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.162747
R squared (OOB):                  -0.103441
[1] "MSE: 0.217155374495409"


[1] "========== n5-r2 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      251
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.7250826
R squared (OOB):                  -0.05847549
[1] "MSE: 0.88460642091214"


[1] "========== n5-r3 =========="
Ranger result

Call:
 ranger(price ~ ., data = trains[[sub]], write.forest = TRUE,      classification = F)

Type:                             Regression
Number of trees:                  500
Sample size:                      40
Number of independent variables:  2
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.1958126
R squared (OOB):                  -0.3441889
[1] "MSE: 0.052617328964372 6"
```

Figure 18: Ranger Random Forest results for specific neighboorhood group and room type

3.2.5 Neural Network Results

Neural Network on the entire dataset has been run for 30 epochs. For semplicity, it has been used a single architecture for all the subsets. It has been used three dense layers of respectively 32, 16 and 1 units and Relu activation function.

Neural Networks on the entire dataset

For the entire dataset, the results are acceptable but not in line with random forest and the other regression methods. The model MSE and loss decrease during the epochs and it stabilise both for validation and training at the 15th epoch.

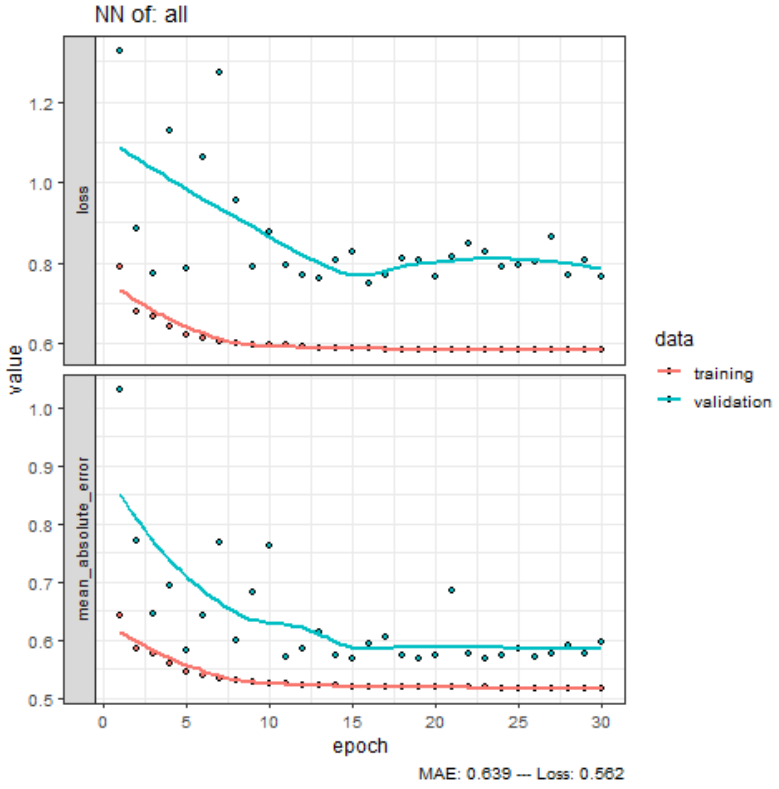


Figure 19: Neural Networks results on the entire dataset

Neural Networks for specific Neighboorhood group

For the specific group the models output different results. The most populous such as Brooklyn and Manhattan and Queens they have a decreasing MSE and loss trend. Only for Manhattan, the validation loss seems to increase approximately after the 18th epochs. For the remaining neighboorhood they have a decreasing training and validation MSE and loss, while the validation remains stable for all the epochs.

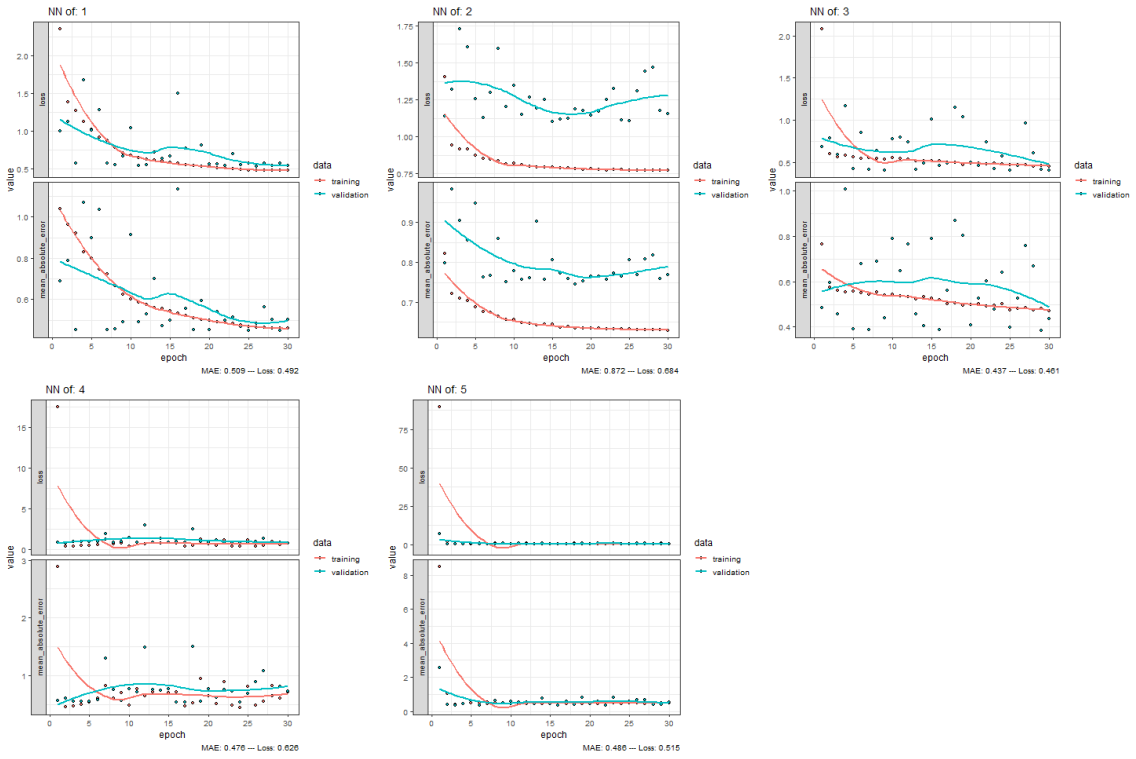


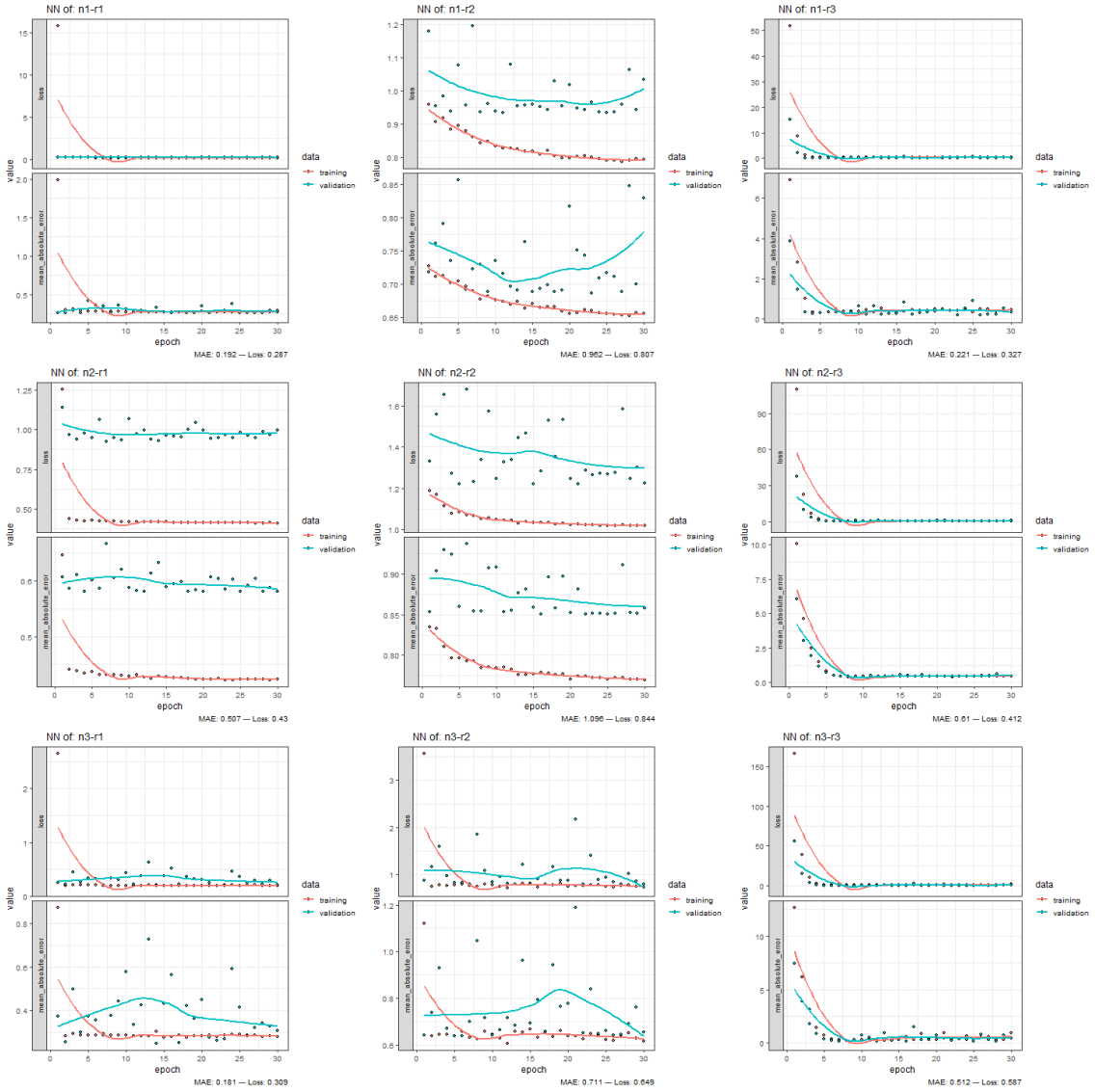
Figure 20: Neurala Networks results for specific neighboorhood group and room type

Neural Networks for specific Neighboorhood group and room type

For the different type of room and neighboorhood in some cases are not satisfactory while in another cases are aceptable. For the type Private room results for Brooklyn and Manhattan and Queens results in a decreasing training MSE and loss and a stable validation MSE and loss. For Brooklyn validation MSE and loss remains nearby 0, for Manhattan instead, are a bit higher compared to the training. For the last two neighboorhood starts with a very high loss and MSE train and validation, while during the epochs the value converg to a lower value near 0.5.

For the entire apartment type, for Brooklyn training mse and loss have a decreasing trend, while validation after 12th epochs increases and get value higher than the starting point. For Manhattan instead we have both train and validation decreasing trend, in particular slightly for the validation. For Queens validation MSE increases till epoch 18th and then decrease untill he reach almost the training MSE at the 30th epochs; while for the loss it continuosly increase and decrease for all the epochs. For Bronx and State Island decreasing training MSE and loss, while a stable validation MSE.

For Shared rooms similar trends of validation and train for the Brooklyng, Manhattan and Queens, starting from a high values. For State Island and Bronx results are decreases very fast but reaching high values of MAE and loss. For Staten Island, train and validation start from a loss value higher than 1000, which is very high considering the dimension of this subsets.


NN of: n1-r1
MAE: 0.192 — Loss: 0.287
NN of: n1-r2
MAE: 0.962 — Loss: 0.807
NN of: n1-r3
MAE: 0.221 — Loss: 0.327
NN of: n2-r1
MAE: 0.507 — Loss: 0.43
NN of: n2-r2
MAE: 1.096 — Loss: 0.844
NN of: n2-r3
MAE: 0.61 — Loss: 0.412
NN of: n3-r1
MAE: 0.181 — Loss: 0.309
NN of: n3-r2
MAE: 0.711 — Loss: 0.649
NN of: n3-r3
MAE: 0.512 — Loss: 0.587
loss
mean_absolute_error
value
epoch
data
training
validation

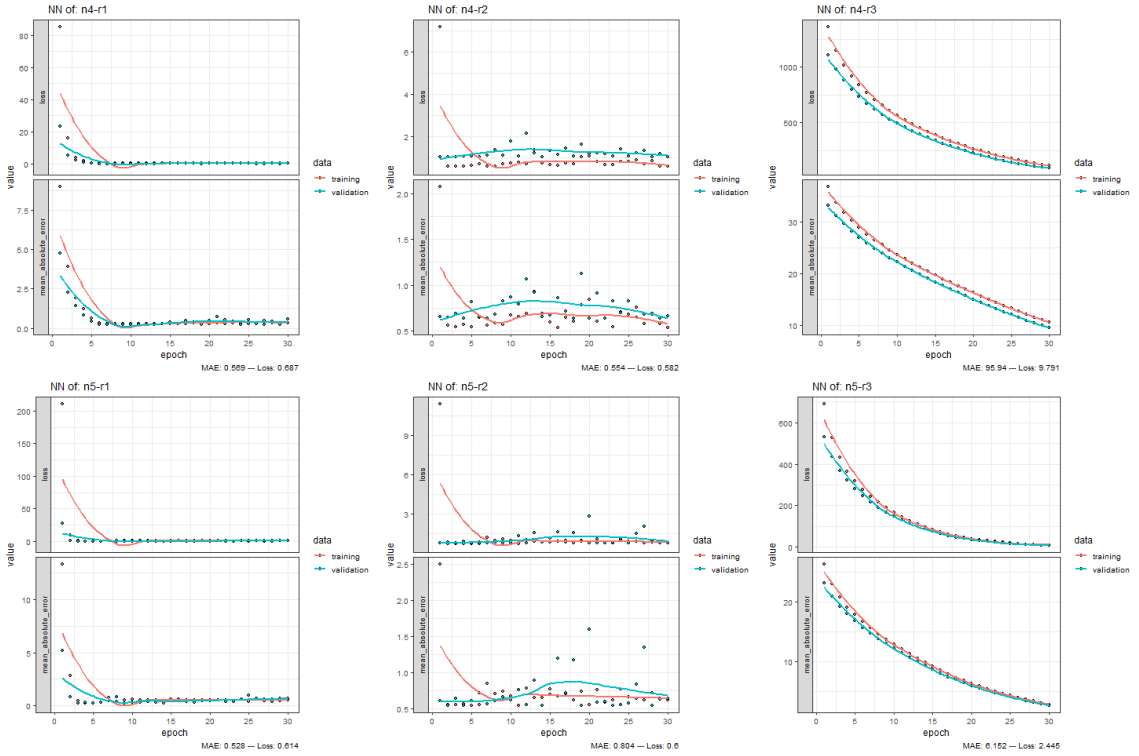


Figure 21: Neural Networks results for specific neighboorhood group and room type

3.2.6 K-means Results

K-means algorithm has been run on all type of subsets as in the models before. Moreover, since the variables are not all numerical, it has been used a specific library called "clustMixType" which computes k-prototypes clustering for mixed-type data. It uses Euclidean distance for numerical variables and simple match coefficent. For semplicity has been computed only setting k = 5 since changing the value for all the different subset will be computationally expensive and also time consuming.

As shown in the cluster map of the entire dataset, clusters define and divides the different neighboorhood. Manhattan is divided in the north and south part that also includes a part of Brooklyn. This should be the case of the most expensive houses in the entire NY city. All cluster can be seen as a partition of price zone in the entire city. As shown in the Figure 23 the distributions of prices and room type differ in the different clusters.

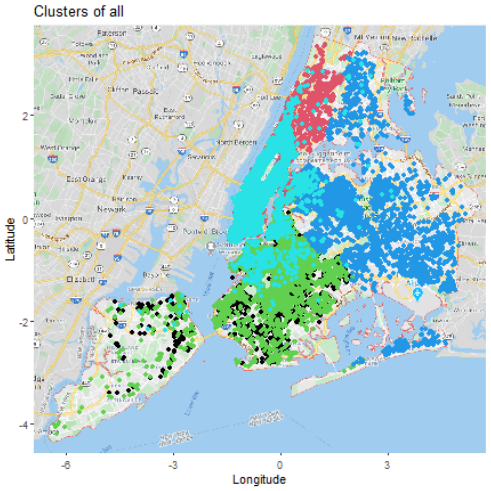


Figure 22: K-means on entire dataset

```
$`1`
  neighbourhood_group    latitude          longitude              room_type         price
 Brooklyn      :9676   Min.   :-4.1808   Min.   :-6.3324   Private room    :1211   Min.   :-1.32482
 Manhattan     :   0   1st Qu.:-1.1986   1st Qu.:-0.6183   Entire home/apt:8758   1st Qu.:-0.37060
 Queens        : 102   Median :-0.8324   Median :-0.1418   Shared room     : 168   Median :-0.02981
 Staten Island: 359   Mean   :-0.9193   Mean   :-0.2816                           Mean   : 0.09813
 Bronx         :   0   3rd Qu.:-0.5292   3rd Qu.: 0.1727                           3rd Qu.: 0.49274
                       Max.   : 0.1840   Max.   : 3.3742                           Max.   : 2.48069

$`2`
  neighbourhood_group    latitude          longitude              room_type         price
 Brooklyn      :9417   Min.   :-2.9821   Min.   :-0.965260   Private room    :11458   Min.   :-1.3248
 Manhattan     :   0   1st Qu.:-0.8932   1st Qu.:-0.005302   Entire home/apt: 1100   1st Qu.:-0.9386
 Queens        :3548   Median :-0.6099   Median : 0.425144   Shared room     :  407   Median :-0.8136
 Staten Island:   0   Mean   :-0.5983   Mean   : 0.786263                           Mean   :-0.7107
 Bronx         :   0   3rd Qu.:-0.2839   3rd Qu.: 1.050479                           3rd Qu.:-0.5864
                       Max.   : 1.0520   Max.   : 5.162639                           Max.   : 3.0487

$`3`
  neighbourhood_group    latitude          longitude              room_type         price
 Brooklyn      :   0   Min.   :-0.04711   Min.   :-1.00744   Private room    :6250   Min.   :-1.3135
 Manhattan     :5666   1st Qu.: 0.88825   1st Qu.:-0.04856   Entire home/apt:1228   1st Qu.:-0.8704
 Queens        :1031   Median : 1.44008   Median : 0.19348   Shared room     : 279   Median :-0.6432
 Staten Island:   0   Mean   : 1.44737   Mean   : 0.29681                           Mean   :-0.5758
 Bronx         :1060   3rd Qu.: 1.84607   3rd Qu.: 0.52681                           3rd Qu.:-0.3706
                       Max.   : 3.36320   Max.   : 3.67901                           Max.   : 2.0149

$`4`
  neighbourhood_group    latitude          longitude              room_type         price
 Brooklyn      : 765   Min.   :-2.7939   Min.   :-3.7011   Private room    : 397   Min.   :1.163
 Manhattan     :3940   1st Qu.:-0.1479   1st Qu.:-0.9388   Entire home/apt:4473   1st Qu.:1.538
 Queens        : 165   Median : 0.2254   Median :-0.6877   Shared room     :  25   Median :1.913
 Staten Island:   7   Mean   : 0.2373   Mean   :-0.5541                           Mean   :2.235
 Bronx         :  18   3rd Qu.: 0.6149   3rd Qu.:-0.2613                           3rd Qu.:2.765
                       Max.   : 2.8482   Max.   : 4.5849                           Max.   :4.185

$`5`
  neighbourhood_group    latitude          longitude              room_type         price
 Brooklyn      :   0   Min.   :-0.4834   Min.   :-1.4459   Private room    :2851   Min.   :-1.3248
 Manhattan     :11271   1st Qu.:-0.0347   1st Qu.:-0.9164   Entire home/apt:8944   1st Qu.:-0.2456
 Queens        :  786   Median : 0.3240   Median :-0.7059   Shared room     : 266   Median : 0.2087
 Staten Island:    0   Mean   : 0.3886   Mean   :-0.5745                           Mean   : 0.1786
 Bronx         :    4   3rd Qu.: 0.7004   3rd Qu.:-0.3107                           3rd Qu.: 0.6063
                       Max.   : 2.2091   Max.   : 1.7230                           Max.   : 1.2879
```

Figure 23: Distribution of each cluster

The maps plot clusters for the single neighboorhood (Figure 24). It is also show how the data are distributied in the maps.

- Brooklyn has 3 main visible clusters;
- Manhattan has 2 clusters for north and south part;
- Queens instead has 4 visible cluster depending on the nearness to the seaside, Manhattan/Brooklyn and inland.
- Staten Island has spars cluster depending on the north east part or the south-west.
- Bronx has 4 clusters visible in which 2 are mixed and the other two depends on the nearness to Manhattan.

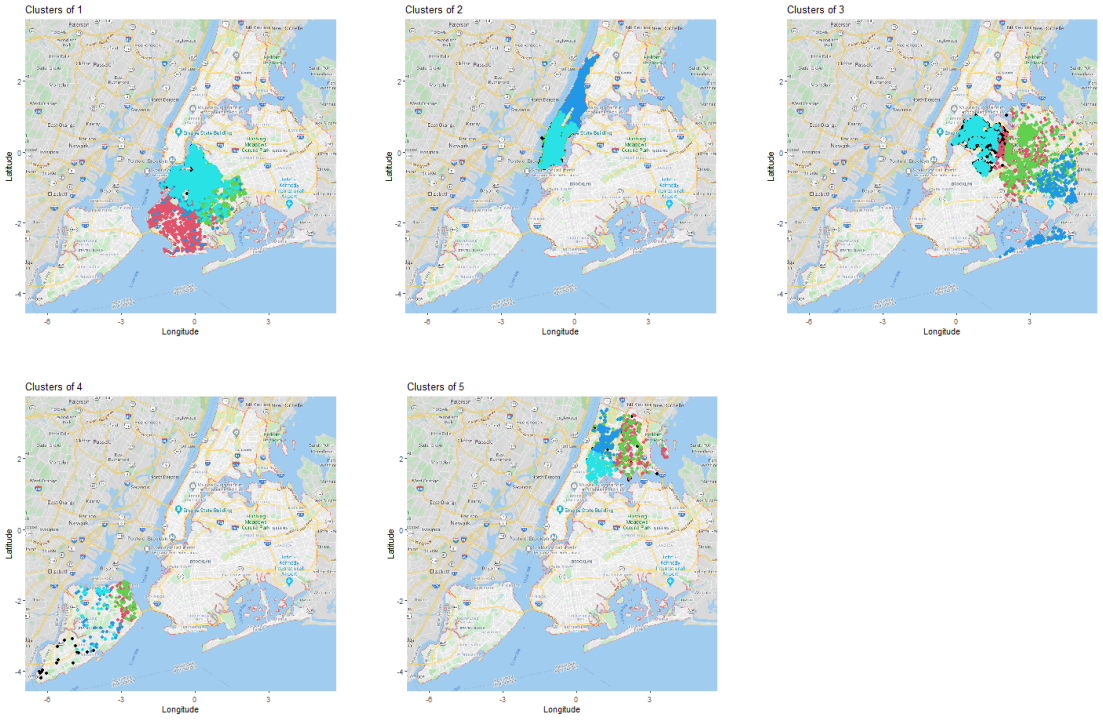
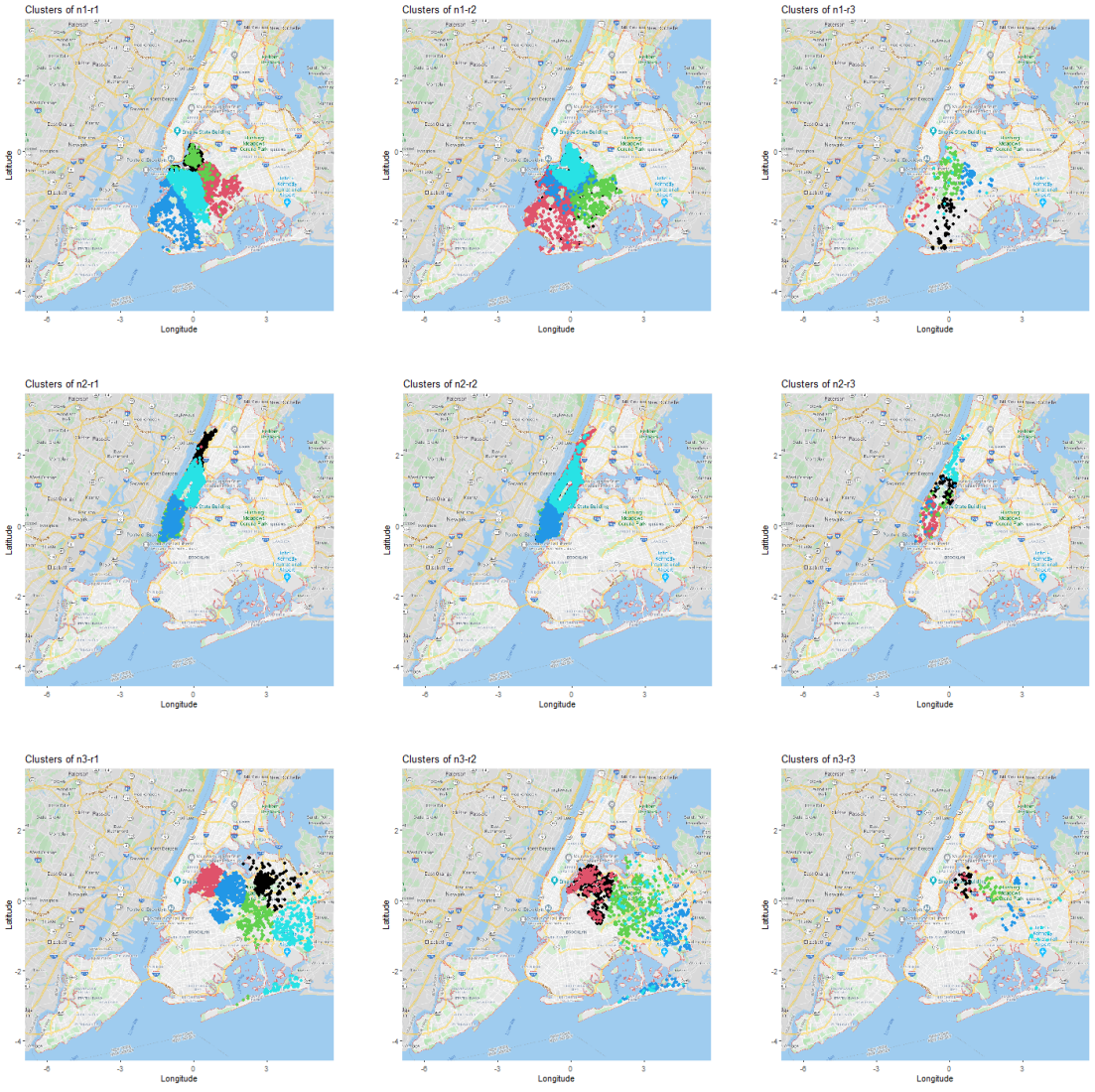


Figure 24: K-means for specific neighboorhood group

Hierarchical Cluster Analysis

For this type of method dataset is different with respect to those of the other methods. The reason is that obtaining a dendogram for all the possible houses is not informative, since the output of the Hierarchical Cluster will be a dendogram in which each label is the correspondent id of the column. For this reason the dataset has been aggregate using the mean of the prices depending on the case.

As shown in Figure 26 the dendogram is generated from the dataset that has been aggregate for mean of the neighboorhood and also for the type of room obtaining the mean of the prices for each case. From Figure 27 is possible to see how giving a certain interval of 0.6 the colored branches are those that could be consider in the same cluster for the HC algorithm.
From Figure 27 is possible to see the dendogram for the case of the mean for each neighboorhood without considering the single case for the room type.



3.2.7 Principal Component Analysis Results

PCA mixed type

Factor Analysis of Mixed Data (FAMD)

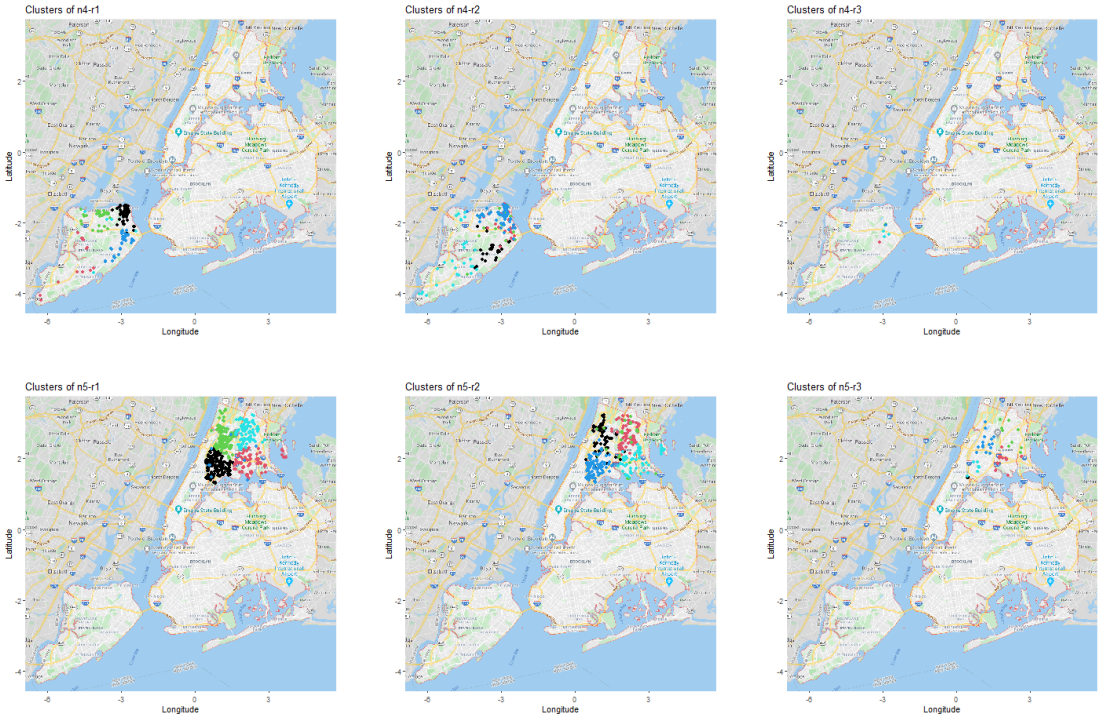


Figure 25: K-means for specific neighboorhood group and room type

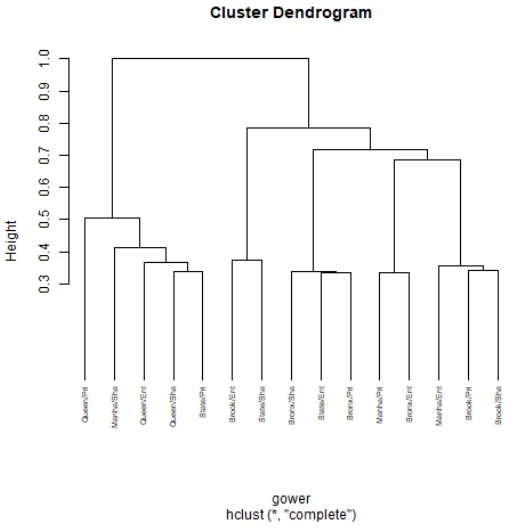


Figure 26: Hierarchical Clustering

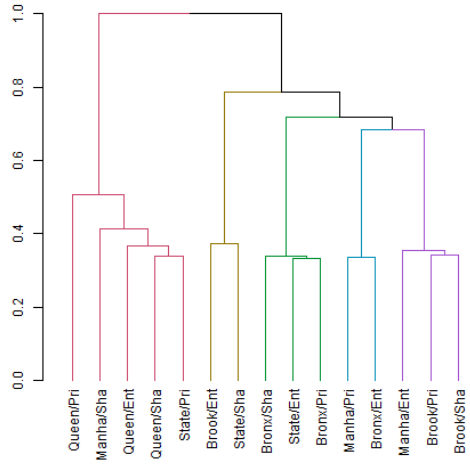


Figure 27: Hierarchical Clustering colored

3.3 Discussion

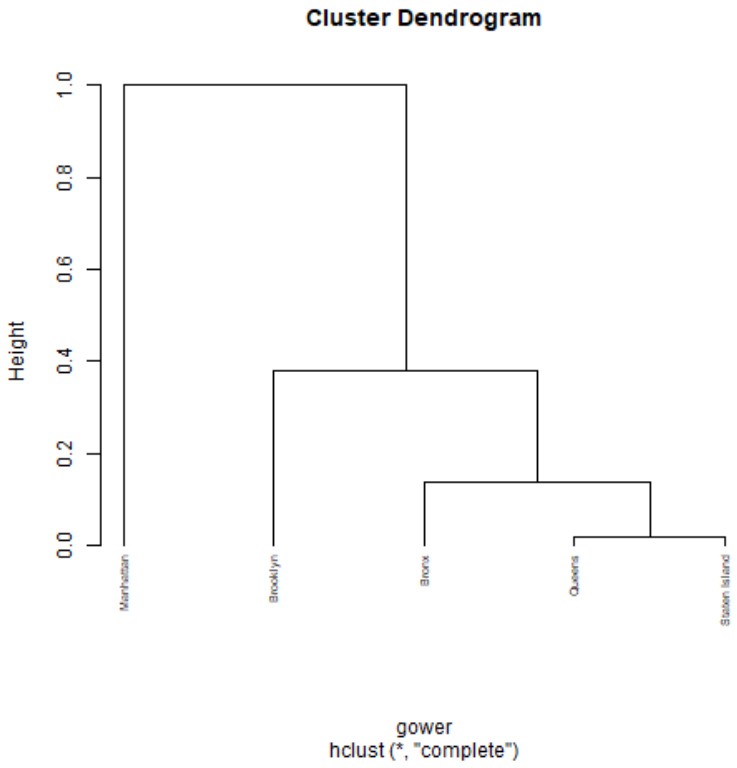


Figure 28: Hierarchical Clustering

	Eigenvalue	Proportion	Cumulative
dim 1	2.1303801	23.670890	23.67089
dim 2	1.8046238	20.051375	43.72227
dim 3	1.2456603	13.840670	57.56294
dim 4	1.0028761	11.143067	68.70600
dim 5	0.9971611	11.079568	79.78557
dim 6	0.9570021	10.633357	90.41893
dim 7	0.4201255	4.668061	95.08699
dim 8	0.2652652	2.947391	98.03438
dim 9	0.1769058	1.965620	100.00000

Figure 29: PCA on mixed type of data

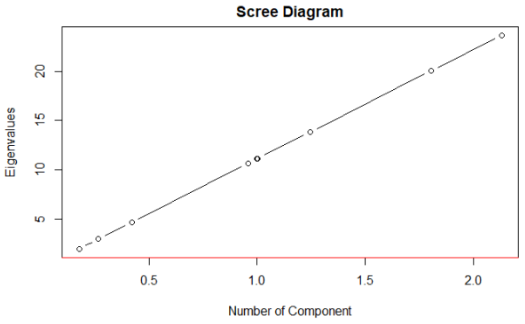


Figure 30: Scree plot

3.3.1 Linear Regression Discussion

Linear regression model gives interesting results for the non-filtered dataset and also for the filtered by neighbourhood group. All variable results to rejected by Null hyphotesis, so the model depends on all the selected variables. Latitude and longitude are correlated with the target, room type is strongly positive correlated with the price and neighbourhood group does not seem to have a great contribution in the prediction of the price.

	eigenvalue	variance.percent	cumulative.variance.percent
Dim.1	2.1303801	23.67089	23.67089
Dim.2	1.8046238	20.05138	43.72227
Dim.3	1.2456603	13.84067	57.56294
Dim.4	1.0028761	11.14307	68.70600
Dim.5	0.9971611	11.07957	79.78557

Figure 31: ...

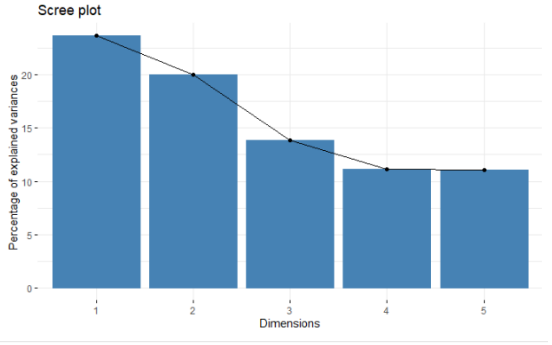


Figure 32: ....

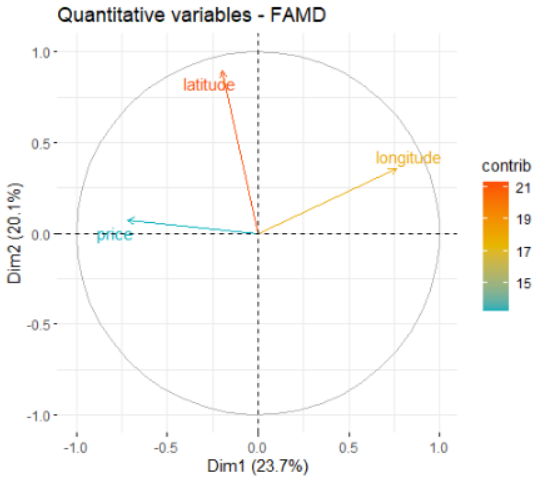


Figure 33: ...

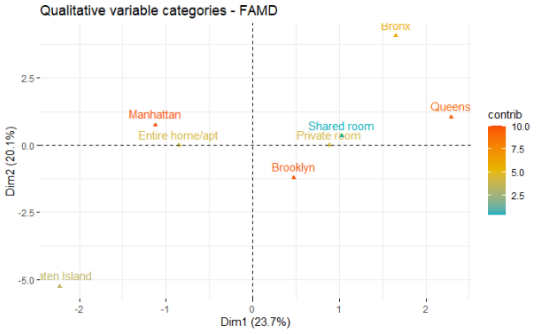


Figure 34: ....

	Dim.1	Dim.2	Dim.3	Dim.4	Dim.5
latitude	0.03892236	0.799743946	0.04903898	0.0016811055	0.001825849
longitude	0.58030735	0.126293692	0.16517205	0.0001203636	0.000168545
price	0.51607212	0.005313173	0.23572672	0.0005079180	0.000184276
neighbourhood_group	0.63900709	0.871714919	0.43427934	0.5273304673	0.480693958
room_type	0.35607121	0.001558062	0.36144319	0.4732362202	0.514288495

	Dim.1	Dim.2	Dim.3	Dim.4	Dim.5
latitude	0.00151495	6.395904e-01	0.002404821	2.826116e-06	3.333727e-06
longitude	0.33675663	1.595010e-02	0.027281805	1.448739e-08	2.840749e-08
price	0.26633043	2.822981e-05	0.055567086	2.579807e-07	3.395780e-08
neighbourhood_group	0.10208251	1.899717e-01	0.047149637	6.951936e-02	5.776667e-02
room_type	0.06339335	1.213779e-06	0.065320588	1.119763e-01	1.322463e-01

	Dim.1	Dim.2	Dim.3	Dim.4	Dim.5
latitude	1.827015	44.31638047	3.936786	0.16762844	0.18310478
longitude	27.239615	6.99833910	13.259799	0.01200184	0.01690250
price	24.224415	0.29441998	18.923837	0.05064614	0.01848011
neighbourhood_group	29.994980	48.30452324	34.863386	52.58181750	48.20624738
room_type	16.713975	0.08633722	29.016193	47.18790608	51.57526523

Figure 35: ...

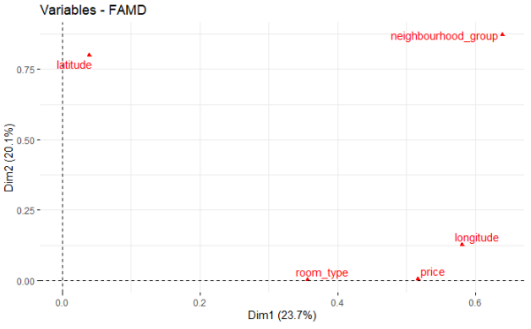


Figure 36: ....

3.3.2 Decision Tree Discussion

The performance with respect to the other models are not the best, but acceptable. The predicton results are not also very precised for the filtered neighboorhood and room type. Also, the plots of the predicted value are not so consistent since the values are divided in category

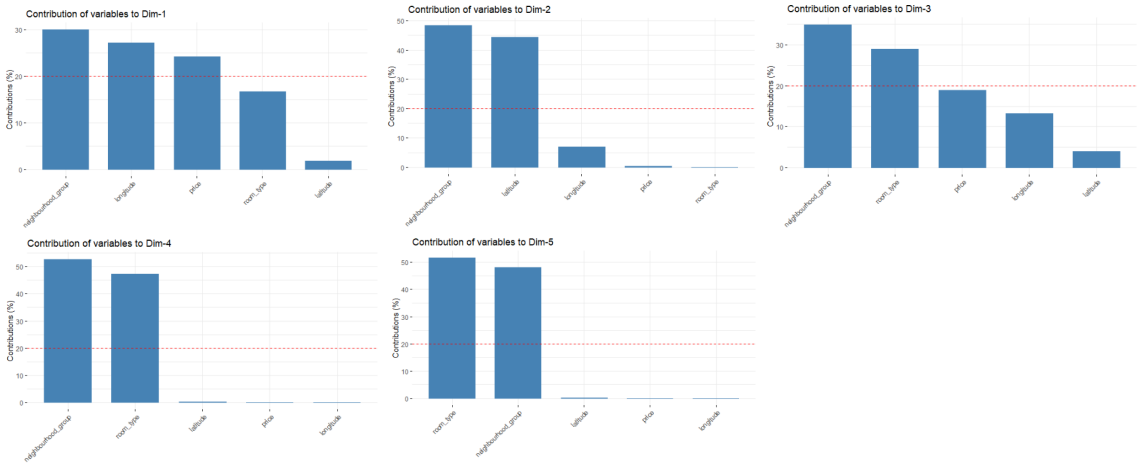


Figure 37: ....

which correspond to the leaves that are not so strong with respect to the other model predictions.

3.3.3 Random Forest Discussion

Random forest outputs consistent results and performs better than linear regression and decision tree. Parameters tuning does not give big improvement in performance and also are computationally expensive.

3.3.4 Ranger Random Forest Discussion

Results of Ranger are the best with respect to the previous models. Also the tuning part was fast and computationally cheaper than the classic Random Forest model but does not give great improvements in performance.

3.3.5 Neural Network Discussion

3.3.6 K-Means Discussion

3.3.7 Principal Component Analysis Discussion

4

Conclusion

From the result the method with the most higher accuracy is the Random Forest method... while the worst are ....
Moreover, Random Forest method is also the worst in term of computation time for the tuning part since it takes for a configuration with 4 core, more or less 1 hour to tune the parameters.

4.1 Linear Regression

4.2 Decision Tree

Decision tree are one of the most used model in the Machine Learning world since are very familiar to human users and can be easily plotted.

4.3 Random Forest

Random Forest is an ensemble method which use a combination of decision tree to get the prediction.

4.3.1 Ranger Random Forest

Ranger Random Forest is a computationally light model which results are very close the classical Random Forest.

5

Appendix

The code is in the file ”project-code.Rmd” and is attached to this file.