Boston Airbnb analysis

Rohan Chouthai
April 22, 2018

SUMMARY: Boston is a major educational, business and tourist hub on the East Coast of US which attracts hundreds of thousands of visitors every year.

Airbnb has become an integral part of the travel community and a great source of income for those fortunate enough to own a place that they can rent out. Naturally, there will be a spike in interest among the home owners to consider listing their house on Airbnb. But they'd surely be wondering, how much money is their place likely going to be worth on Airbnb? Similarly, the people who do own a house listed on Airbnb might want to improve the quality of their listing. They'd like to know what are some of the factors which influence the price of a listing? How important a part do the reviews play in a new customers mind and does it have an impact on the price of the listing? All this and more!!

PHASES OF CRISP-DM:

1. BUSINESS UNDERSTANDING: The aim of the project is to predict the Yield of a potential Airbnb listing by developing model(s) to define the impact of various parameters on listing prices in Boston. The project also goes on to fill the void in the analytics world by addressing ways to improve the listing price for those already having their places on Airbnb.

2. DATA UNDERSTANDING:

Dataset: I chose this dataset from Kaggle (source: https://www.kaggle.com/airbnb/boston) [1]. This dataset is originally a part of Airbnb Inside. (source: http://insideairbnb.com/get-the-data.html).

This Dataset consists of 3 individual datasets: Calendar, Listings and Reviews. I have combined the Listings and Reviews datasets at a later point in my project. I'd mostly be working with the Listings dataset.

a.IMPORTING THE DATA:

First let us load all the three datasets into R.

```
Calendar<- read.csv("C:/Users/rohan/Desktop/DMML/Boston AIr BNB/calendar.csv")

Listings<- read.csv("C:/Users/rohan/Desktop/DMML/Boston AIr BNB/listings.csv")

Reviews<-read.csv("C:/Users/rohan/Desktop/DMML/Boston AIr BNB/reviews.csv")
```

Let us check the dimensions of these datasets.

```
dim(Calendar)
```

```
## [1] 1308890       4
```

```
dim(Listings)
```

```
## [1] 3585   95
```

```
dim(Reviews)
```

```
## [1] 68275     6
```

We can see that the Calendar dataset is the largest. It primarily deals with the historical availablity of the Listings. This is not the dataset I would be using for any of my analysis.

The Listings dataset is of primary interest as it has a lot of information about the listings such as the number of rooms, beds, price of the listing etc. I will be using this dataset for my modelling.

The Reviews dataset has reviews for all the listings by multiple users. I will be using this datset for the sentiment analysis.

1. EXPLORATORY DATA ANALYSIS:

In a city as expensive as Boston, there is a lot of curiosity around which areas in Boston are the most expensive. The Listings dataset has a lot of information about the neighbourhood and the price of the listings therein. I will now explore which areas are the most expensive in Boston.

First, let us subset the columns we require for visualizing this.

```
suppressWarnings(library(tidyverse))
```

```
## -- Attaching packages ---------------------- tidyverse 1.2.1 --

## v ggplot2 2.2.1     v purrr   0.2.4
## v tibble  1.4.1     v dplyr   0.7.4
## v tidyr   0.7.2     v stringr 1.2.0
## v readr   1.1.1     v forcats 0.2.0

## -- Conflicts ------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
Daily_Price<- Listings%>% select(host_since,host_location,host_response_time,host_acc
dim(Daily_Price)
```

```
## [1] 3585   18
```

Now, let us explore the most expensive neighbourhoods.

```
suppressWarnings(library(ggplot2))

Daily_Price$price<-as.integer(Daily_Price$price)
```

```
Neighbourhoods<-Daily_Price%>% group_by(neighbourhood_cleansed)%>% summarise(Avg_pric
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.3
```

```
head(Neighbourhoods)
```

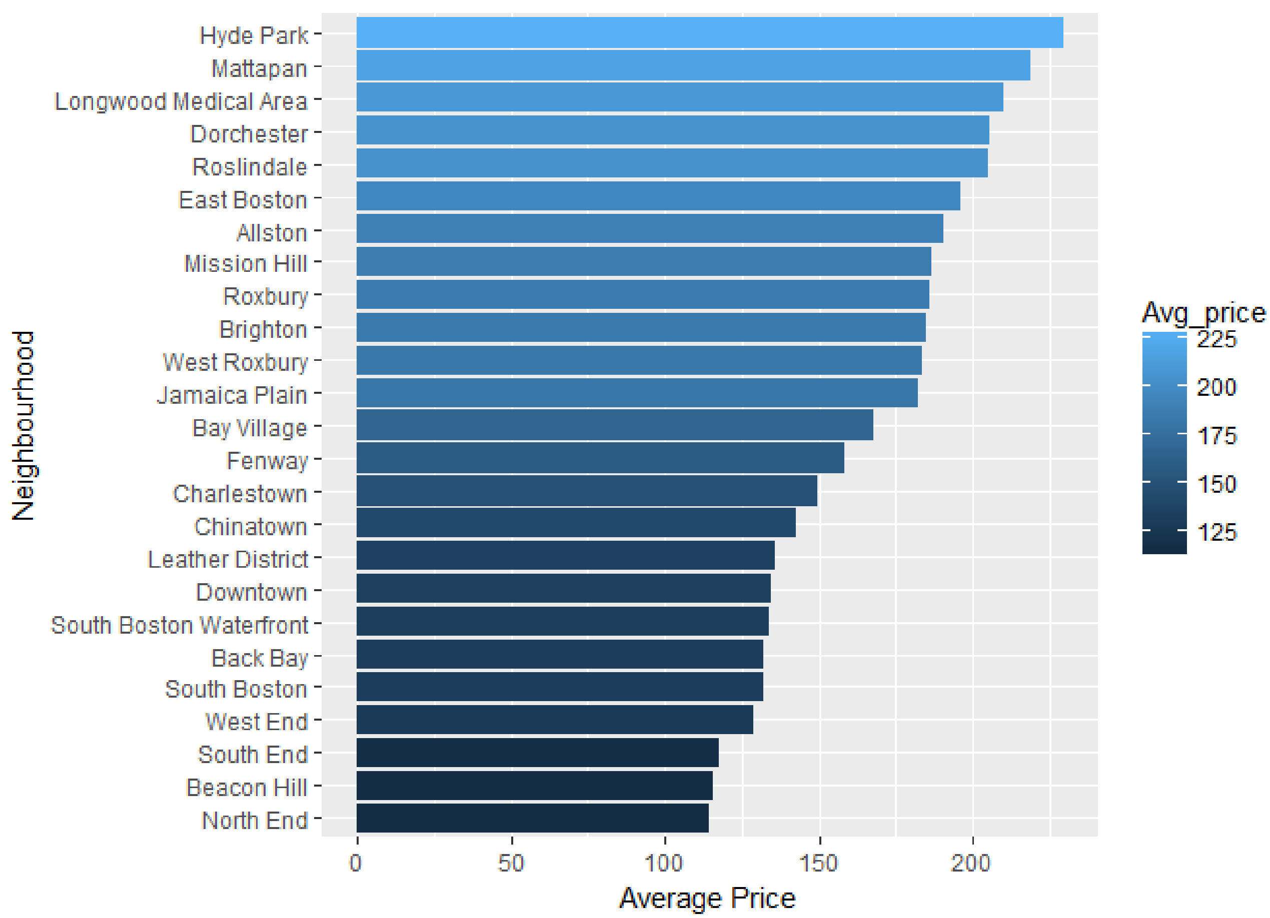
```
## # A tibble: 6 x 2
##   neighbourhood_cleansed Avg_price
##   <fct>                      <dbl>
## 1 Hyde Park                    229
```

```
## 2 Mattapan                  219
## 3 Longwood Medical Area     210
## 4 Dorchester                205
## 5 Roslindale                205
## 6 East Boston               196
```

Looks like Hyde Park is the most expensive neighbourhood in Boston to be renting a bnb in. It costs a whopping $229 per night. Sure we now have the average price per neighbourhood.

Now, let us visualize the most expensive areas.

```
ggplot(Neighbourhoods)+geom_bar(mapping = aes(reorder(neighbourhood_cleansed,Avg_pric
```



We can easily see that North End seems to be the cheaper place to rent out an bnb in. But what do the people who have stayed here got to say about Northend? (Sentiment analysis to follow in the last part)

3.DATA PREPARATION:

This stage will involve addressing the NA values and missing entries for the attributes in the dataset. The comments and reviews on the listings will also need cleansing for analysis. Any other data quality issues will be addressed upon diving deeper into the data exploration.

The Listings dataset has various different kinds of variables with listing URL, host images etc. I will not be working with all the variables for building my model. Therefore, I have dealt with the Data Preparation stage when I have built the models as per the requirement of my models.

PART A: AIRBNB DAILY RATE PREDICTION and HOST-SUPERHOST CLASSIFICATION

1. MODELLING:

Now let us build a regression model to predict the price of the housing per night. We will consider the Listings dataset since it has all the information about the individual listings.

Let us first get some sense of the Listings dataset.

```
dim(Listings)
```

```
## [1] 3585   95
```

```
str(Listings)
```

```
## 'data.frame':    3585 obs. of  95 variables:
##  $ id                              : int  12147973 3075044 6976 1436513 7651065 12
##  $ listing_url                     : Factor w/ 3585 levels "https://www.airbnb.com
##  $ scrape_id                       : num  2.02e+13 2.02e+13 2.02e+13 2.02e+13 2.02
##  $ last_scraped                    : Factor w/ 1 level "2016-09-07": 1 1 1 1 1 1 1
##  $ name                            : Factor w/ 3504 levels " Cozy Spot in Boston's
##  $ summary                         : Factor w/ 3114 levels "","- 1 min walk to T (
##  $ space                           : Factor w/ 2269 levels "","-  Just renovated 2
##  $ description                     : Factor w/ 3423 levels "- 1 min walk to T (sub
##  $ experiences_offered             : Factor w/ 1 level "none": 1 1 1 1 1 1 1 1 1 1
##  $ neighborhood_overview           : Factor w/ 1729 levels "","-  Centrally locate
##  $ notes                           : Factor w/ 1270 levels "","- 5 min walk to Tuf
##  $ transit                         : Factor w/ 1860 levels "","- #1 Bus to Cambrid
##  $ access                          : Factor w/ 1763 levels "","-  Electronic locks
##  $ interaction                     : Factor w/ 1618 levels "","-  My wife and I li
##  $ house_rules                     : Factor w/ 1929 levels "","--No Smoking.   --P
##  $ thumbnail_url                   : Factor w/ 2987 levels "","https://a0.muscache
##  $ medium_url                      : Factor w/ 2987 levels "","https://a0.muscache
```

```
## $ picture_url                  : Factor w/ 3585 levels "https://a0.muscache.co
## $ xl_picture_url               : Factor w/ 2987 levels "","https://a0.muscache
## $ host_id                      : int  31303940 2572247 16701 6031442 15396970
## $ host_url                     : Factor w/ 2181 levels "https://www.airbnb.com
## $ host_name                    : Factor w/ 1334 levels "40 Berkeley",..: 1280
## $ host_since                   : Factor w/ 1281 levels "2008-11-11","2008-12-0
## $ host_location                : Factor w/ 177 levels "","Abington, Massachuse
## $ host_about                   : Factor w/ 1241 levels ""," ","      ã□□æ³¢å£
## $ host_response_time           : Factor w/ 5 levels "a few days or more",..: 2
## $ host_response_rate           : Factor w/ 53 levels "0%","10%","100%",..: 53
## $ host_acceptance_rate         : Factor w/ 73 levels "0%","100%","17%",..: 73
## $ host_is_superhost            : Factor w/ 2 levels "f","t": 1 1 2 1 2 2 1 2 2
## $ host_thumbnail_url           : Factor w/ 2174 levels "https://a0.muscache.co
## $ host_picture_url             : Factor w/ 2174 levels "https://a0.muscache.co
## $ host_neighbourhood           : Factor w/ 54 levels "","Allston-Brighton",..:
## $ host_listings_count          : int  1 1 1 1 1 2 5 2 1 2 ...
## $ host_total_listings_count    : int  1 1 1 1 1 2 5 2 1 2 ...
## $ host_verifications           : Factor w/ 83 levels "['email', 'facebook', 'r
## $ host_has_profile_pic         : Factor w/ 2 levels "f","t": 2 2 2 2 2 2 2 2 2
## $ host_identity_verified       : Factor w/ 2 levels "f","t": 1 2 2 1 2 2 2 2 2
## $ street                       : Factor w/ 1239 levels ", MA 02467, United Sta
## $ neighbourhood                : Factor w/ 31 levels "","Allston-Brighton",..:
## $ neighbourhood_cleansed       : Factor w/ 25 levels "Allston","Back Bay",..:
## $ neighbourhood_group_cleansed : logi  NA NA NA NA NA NA ...
## $ city                         : Factor w/ 39 levels "","æ³¢å£«é¡¿",..: 6 6 6
## $ state                        : Factor w/ 1 level "MA": 1 1 1 1 1 1 1 1 1 1 .
## $ zipcode                      : Factor w/ 44 levels "","02108","02108 02111",
## $ market                       : Factor w/ 5 levels "","Boston","Other (Domest
## $ smart_location               : Factor w/ 39 levels "æ³¢å£«é¡¿, MA",..: 8 8 8
## $ country_code                 : Factor w/ 1 level "US": 1 1 1 1 1 1 1 1 1 1 .
## $ country                      : Factor w/ 1 level "United States": 1 1 1 1 1
## $ latitude                     : num  42.3 42.3 42.3 42.3 42.3 ...
## $ longitude                    : num  -71.1 -71.1 -71.1 -71.1 -71.1 ...
## $ is_location_exact            : Factor w/ 2 levels "f","t": 2 2 2 1 2 2 1 2 2
## $ property_type                : Factor w/ 14 levels "","Apartment",..: 10 2 2
## $ room_type                    : Factor w/ 3 levels "Entire home/apt",..: 1 2
## $ accommodates                 : int  4 2 2 4 2 2 3 2 2 5 ...
## $ bathrooms                    : num  1.5 1 1 1 1.5 1 1 2 1 1 ...
## $ bedrooms                     : int  2 1 1 1 1 1 1 1 1 2 ...
## $ beds                         : int  3 1 1 2 2 1 2 1 2 2 ...
## $ bed_type                     : Factor w/ 5 levels "Airbed","Couch",..: 5 5 5
## $ amenities                    : Factor w/ 3092 levels "{\"24-Hour Check-in\"}
## $ square_feet                  : int  NA NA NA NA NA NA NA NA 12 NA ...
## $ price                        : Factor w/ 324 levels "$1,000.00","$1,235.00",
## $ weekly_price                 : Factor w/ 244 levels "","$1,000.00",..: 1 140
## $ monthly_price                : Factor w/ 289 levels "","$1,000.00",..: 1 1 2
## $ security_deposit             : Factor w/ 55 levels "","$1,000.00",..: 1 53 1
## $ cleaning_fee                 : Factor w/ 80 levels "","$10.00","$100.00",..:
## $ guests_included              : int  1 0 1 2 1 1 1 1 2 4 ...
## $ extra_people                 : Factor w/ 51 levels "$0.00","$10.00",..: 1 1
## $ minimum_nights               : int  2 2 3 1 2 2 1 1 2 4 ...
```

```
##  $ maximum_nights                  : int  1125 15 45 1125 31 1125 1125 1125 1125 1
##  $ calendar_updated                : Factor w/ 38 levels "1 week ago","10 months a
##  $ has_availability                : logi  NA NA NA NA NA NA ...
##  $ availability_30                 : int  0 26 19 6 13 5 22 30 12 20 ...
##  $ availability_60                 : int  0 54 46 16 34 28 39 60 42 50 ...
##  $ availability_90                 : int  0 84 61 26 59 58 69 90 72 80 ...
##  $ availability_365                : int  0 359 319 98 334 58 344 365 347 107 ...
##  $ calendar_last_scraped           : Factor w/ 1 level "2016-09-06": 1 1 1 1 1 1 1
##  $ number_of_reviews               : int  0 36 41 1 29 8 57 67 65 33 ...
##  $ first_review                    : Factor w/ 976 levels "","2009-03-21",..: 1 31
##  $ last_review                     : Factor w/ 405 levels "","2010-10-16",..: 1 38
##  $ review_scores_rating            : int  NA 94 98 100 99 100 90 96 96 94 ...
##  $ review_scores_accuracy          : int  NA 10 10 10 10 10 10 10 10 10 ...
##  $ review_scores_cleanliness       : int  NA 9 9 10 10 10 10 10 10 9 ...
##  $ review_scores_checkin           : int  NA 10 10 10 10 10 10 10 10 10 ...
##  $ review_scores_communication     : int  NA 10 10 10 10 10 10 10 10 10 ...
##  $ review_scores_location          : int  NA 9 9 10 9 9 9 10 9 9 ...
##  $ review_scores_value             : int  NA 9 10 10 10 10 9 10 10 9 ...
##  $ requires_license                : Factor w/ 1 level "f": 1 1 1 1 1 1 1 1 1 1 ..
##  $ license                         : logi  NA NA NA NA NA NA ...
##  $ jurisdiction_names              : logi  NA NA NA NA NA NA ...
##  $ instant_bookable                : Factor w/ 2 levels "f","t": 1 2 1 1 1 1 1 1 1
##  $ cancellation_policy             : Factor w/ 4 levels "flexible","moderate",..:
##  $ require_guest_profile_picture   : Factor w/ 2 levels "f","t": 1 1 2 1 1 1 1 2 1
##  $ require_guest_phone_verification: Factor w/ 2 levels "f","t": 1 1 1 1 1 1 1 2 1
##  $ calculated_host_listings_count  : int  1 1 1 1 1 1 3 2 1 2 ...
##  $ reviews_per_month               : num  NA 1.3 0.47 1 2.25 1.7 4 2.38 5.36 1.01
```

We can see that the dataset has 95 attributes per listing. Many of those attributes we don't need to build our model. Let us, therefore, select the ones which we will be using and put them into a new dataframe.

1. PREDICTION MODELS

Problem Statement: I will first build two prediction models to predict the price per day of the Airbnb listing in Boston.

I will be first selecting the features I believe are most likely going to affect the price. I will omit out the other columns for the purpose of this model.

```
suppressWarnings(library(tidyverse))
```

```
Daily_Price<- Listings%>% select(host_since,host_location,host_response_time,host_acc
dim(Daily_Price)
```

```
## [1] 3585   18
```

We have now selected 18 attributes we are going to be building our model on. These still are a lot of parameters to be building the model on.

Let us now see the type of variables we have in our dataframe.

```
str(Daily_Price)
```

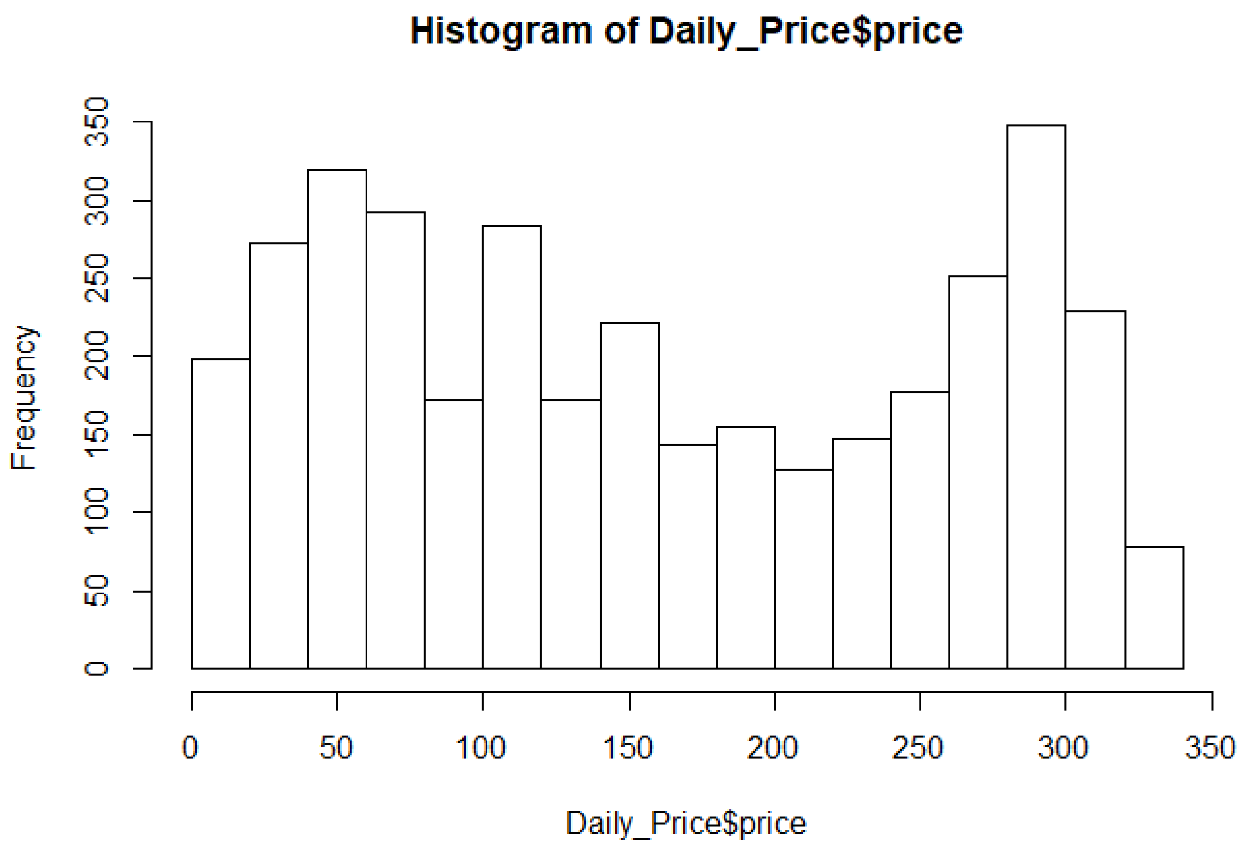
```
## 'data.frame':    3585 obs. of  18 variables:
##  $ host_since            : Factor w/ 1281 levels "2008-11-11","2008-12-03",..: 880
##  $ host_location         : Factor w/ 177 levels "","Abington, Massachusetts, Unite
##  $ host_response_time    : Factor w/ 5 levels "a few days or more",..: 2 5 4 4 5 4
##  $ host_acceptance_rate  : Factor w/ 73 levels "0%","100%","17%",..: 73 2 61 23 2
##  $ host_is_superhost     : Factor w/ 2 levels "f","t": 1 1 2 1 2 2 1 2 2 2 ...
##  $ neighbourhood_cleansed: Factor w/ 25 levels "Allston","Back Bay",..: 19 19 19 1
##  $ is_location_exact     : Factor w/ 2 levels "f","t": 2 2 2 1 2 2 1 2 2 2 ...
##  $ property_type         : Factor w/ 14 levels "","Apartment",..: 10 2 2 10 10 6 2
##  $ room_type             : Factor w/ 3 levels "Entire home/apt",..: 1 2 2 2 2 2 1
##  $ accommodates          : int  4 2 2 4 2 2 3 2 2 5 ...
##  $ bathrooms             : num  1.5 1 1 1 1.5 1 1 2 1 1 ...
##  $ bedrooms              : int  2 1 1 1 1 1 1 1 1 2 ...
##  $ beds                  : int  3 1 1 2 2 1 2 1 2 2 ...
##  $ bed_type              : Factor w/ 5 levels "Airbed","Couch",..: 5 5 5 5 5 5 5 5
##  $ price                 : Factor w/ 324 levels "$1,000.00","$1,235.00",..: 144 27
##  $ security_deposit      : Factor w/ 55 levels "","$1,000.00",..: 1 53 1 7 1 1 1 1
##  $ minimum_nights        : int  2 2 3 1 2 2 1 1 2 4 ...
##  $ maximum_nights        : int  1125 15 45 1125 31 1125 1125 1125 1125 10 ...
```

We see our response variable price is a factor. We need to convert into a numeric value so that we can use it in the linear regression model.

```
Daily_Price$price<-as.numeric(Daily_Price$price)
```

Now let us see if it is normally distributed. A regression model fits better with normally distributed response variable.

```
hist(Daily_Price$price)
```



The figure shows a non normal distribution. We see that most of the listings are either too expensive or quite cheap.

Let us deal with that issue later.

--DATA PREPARATION:

First let us see if there are NA values in our dataset.

```
anyNA(Daily_Price)
```

```
## [1] TRUE
```

```
sum(complete.cases(Daily_Price))/nrow(Daily_Price)
```

```
## [1] 0.9921897
```

There are NA values. However, 99% of the data is complete. That is a good start.

Let us now find out where these NA values are.

```
sapply(Daily_Price,function(x) sum(is.na(x)))
```

```
##            host_since          host_location     host_response_time
##                     0                      0                      0
##   host_acceptance_rate      host_is_superhost neighbourhood_cleansed
##                     0                      0                      0
##      is_location_exact          property_type              room_type
##                     0                      0                      0
##          accommodates              bathrooms               bedrooms
##                     0                     14                     10
##                  beds               bed_type                  price
##                     9                      0                      0
##      security_deposit         minimum_nights         maximum_nights
##                     0                      0                      0
```

We can see that there are NA values in Bathrooms, Bedrooms and Beds. We need to impute these values to get a correlation among these parameters and the daily price.

We will impute the NA values with mode of the respective parameters. We first need to check the class of Bathrooms, Bedrooms and Bed. We need it to be factor since we are going to impute it with mode.

```
class(Daily_Price$bathrooms)
```

```
## [1] "numeric"
```

```
class(Daily_Price$bedrooms)
```

```
## [1] "integer"
```

```
class(Daily_Price$beds)
```

```
## [1] "integer"
```

We see that one of them is a character and others are integers. Let us convert them all into factors.

```
Daily_Price$bathrooms<-as.factor(Daily_Price$bathrooms)
Daily_Price$bedrooms<-as.factor(Daily_Price$bedrooms)
Daily_Price$beds<-as.factor(Daily_Price$beds)
class(Daily_Price$bedrooms)
```

```
## [1] "factor"
```

```
unique(Daily_Price$bedrooms)
```

```
## [1] 2    1    0    3    4    5    <NA>
## Levels: 0 1 2 3 4 5
```

```
mode(Daily_Price$bedrooms)
```

```
## [1] "numeric"
```

Now, let us go ahead and do the imputation.

Let us start of by imputing the Bathrooms parameter.

First, let us write a function or calculating the mode.

```
modeVal<-function(x){
  uniq_x<-unique(x)
  uniq_x[which.max(tabulate(match(x,uniq_x)))]
}
```

Now let us impute the Bathroom with mode.

```
Mode_bathroom<-modeVal(Daily_Price$bathrooms)
```

```
Daily_Price$bathrooms<-ifelse(is.na(Daily_Price$bathrooms),Mode_bathroom,Daily_Price$
```

```
unique(Daily_Price$bathrooms)
```

```
## [1]  4  3  5  1  6  8  7  2 10  9 11 12
```

We can see that we successfully imputed the NA values in the Bathrooms column.

Now, let us repeat the process for Bedrooms and Beds.

```
Mode_bedrooms<-modeVal(Daily_Price$bedrooms)

Daily_Price$bedrooms<-ifelse(is.na(Daily_Price$bedrooms),Mode_bedrooms,Daily_Price$be
unique(Daily_Price$bedrooms)
```

```
## [1] 3 2 1 4 5 6
```

Thus we have imputed the bedrooms as well.

Now, let us impute the beds.

```
Mode_beds<-modeVal(Daily_Price$beds)

Daily_Price$beds<-ifelse(is.na(Daily_Price$beds),Mode_beds,Daily_Price$beds)
unique(Daily_Price$beds)
```

```
## [1]  4  2  3  6  8  5  7 10  9  1 11
```

We have now gotten rid of the NA values in the Beds column as well.

Let us now confirm if there are any NA values anywhere in our dataframe.

```
anyNA(Daily_Price)
```

```
## [1] FALSE
```

There are no more NA values left.

We have to develop a multiple regression model. Let us first convert our data into numeric values.

```
Daily<-as.data.frame(lapply(Daily_Price[,], as.numeric))

str(Daily)
```

```
## 'data.frame':    3585 obs. of  18 variables:
##  $ host_since           : num  880 223 6 388 625 ...
##  $ host_location        : num  31 31 31 31 31 31 95 31 31 31 ...
##  $ host_response_time    : num  2 5 4 4 5 4 5 4 5 5 ...
##  $ host_acceptance_rate  : num  73 2 61 23 2 68 69 2 2 2 ...
##  $ host_is_superhost     : num  1 1 2 1 2 2 1 2 2 2 ...
##  $ neighbourhood_cleansed: num  19 19 19 19 19 19 19 19 19 19 ...
##  $ is_location_exact     : num  2 2 2 1 2 2 1 2 2 2 ...
##  $ property_type         : num  10 2 2 10 10 6 2 10 6 2 ...
##  $ room_type             : num  1 2 2 2 2 2 1 2 2 1 ...
##  $ accommodates          : num  4 2 2 4 2 2 3 2 2 5 ...
##  $ bathrooms             : num  4 3 3 3 4 3 3 5 3 3 ...
##  $ bedrooms              : num  3 2 2 2 2 2 2 2 2 3 ...
##  $ beds                  : num  4 2 2 3 3 2 3 2 3 3 ...
##  $ bed_type              : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ price                 : num  144 276 276 293 299 293 10 293 265 127 ...
##  $ security_deposit      : num  1 53 1 7 1 1 1 1 1 22 ...
##  $ minimum_nights        : num  2 2 3 1 2 2 1 1 2 4 ...
##  $ maximum_nights        : num  1125 15 45 1125 31 ...
```

Let us now observe the correlation between our dependent variable and the independent variables.

```
cor(Daily[,])
```

```
##                         host_since host_location host_response_time
## host_since              1.000000000   0.134059843       -0.028323365
## host_location           0.134059843   1.000000000       -0.047628745
## host_response_time     -0.028323365  -0.047628745        1.000000000
## host_acceptance_rate   -0.006112269  -0.025784739       -0.379283320
## host_is_superhost      -0.079064385  -0.159469055        0.162217788
## neighbourhood_cleansed -0.034781459   0.007381172       -0.006124583
## is_location_exact      -0.171406863   0.018204490       -0.041132617
## property_type           0.021416858  -0.117041035        0.090747046
## room_type               0.103252186  -0.151011153        0.003833446
## accommodates           -0.105665233   0.087657432        0.096500837
## bathrooms              -0.058208224   0.067502164        0.035640437
## bedrooms               -0.085290675   0.089568751        0.032817308
## beds                   -0.076331857   0.035598783        0.087269440
## bed_type               -0.013536577   0.062301376       -0.003084712
## price                   0.043369989  -0.043812015        0.058460746
## security_deposit       -0.097072380  -0.105459775        0.091120874
```

```
## minimum_nights         -0.028294402   0.017500483       -0.034213070
## maximum_nights         -0.018212350  -0.009857720        0.016256911
##                        host_acceptance_rate host_is_superhost
## host_since                    -0.0061122691      -0.079064385
## host_location                 -0.0257847392      -0.159469055
## host_response_time            -0.3792833196       0.162217788
## host_acceptance_rate           1.0000000000      -0.007190618
## host_is_superhost             -0.0071906183       1.000000000
## neighbourhood_cleansed        -0.0158855921       0.066341834
## is_location_exact             -0.0078270353       0.048861790
## property_type                 -0.0602331383       0.092454112
## room_type                     -0.0263962635       0.054561871
## accommodates                  -0.0114748425       0.016905401
## bathrooms                     -0.0222397679       0.045013293
## bedrooms                      -0.0312437523       0.031710635
## beds                          -0.0260991314       0.018987197
## bed_type                      -0.0274931104       0.014919762
## price                         -0.0618676727       0.056205079
## security_deposit              -0.0009938860       0.063025819
## minimum_nights                 0.0005110294      -0.024150588
## maximum_nights                -0.0192715609      -0.005999207
##                        neighbourhood_cleansed is_location_exact
## host_since                        -0.034781459      -0.171406863
## host_location                      0.007381172       0.018204490
## host_response_time                -0.006124583      -0.041132617
## host_acceptance_rate              -0.015885592      -0.007827035
## host_is_superhost                  0.066341834       0.048861790
## neighbourhood_cleansed             1.000000000       0.007020115
## is_location_exact                  0.007020115       1.000000000
## property_type                      0.051760897       0.026415933
## room_type                         -0.018292400      -0.115251686
## accommodates                       0.049904911       0.063493816
## bathrooms                          0.052354265       0.086099917
## bedrooms                           0.041360229       0.059592341
## beds                               0.050786597       0.041199445
## bed_type                           0.025828964       0.076112703
## price                             -0.069247342      -0.023247652
## security_deposit                   0.046098627      -0.020237476
## minimum_nights                    -0.019224557       0.039806187
## maximum_nights                     0.014546907       0.006755548
##                        property_type    room_type accommodates
## host_since               0.021416858  0.103252186 -0.105665233
## host_location           -0.117041035 -0.151011153  0.087657432
## host_response_time       0.090747046  0.003833446  0.096500837
## host_acceptance_rate    -0.060233138 -0.026396263 -0.011474843
## host_is_superhost        0.092454112  0.054561871  0.016905401
## neighbourhood_cleansed   0.051760897 -0.018292400  0.049904911
## is_location_exact        0.026415933 -0.115251686  0.063493816
## property_type            1.000000000  0.262800641 -0.029132191
## room_type                0.262800641  1.000000000 -0.520912510
## accommodates            -0.029132191 -0.520912510  1.000000000
```

```
## bathrooms              0.227967010 -0.077615722  0.351401359
## bedrooms               0.057589730 -0.270566917  0.724917621
## beds                   0.054677583 -0.355830008  0.824537011
## bed_type               0.008502794 -0.233769264  0.139756953
## price                  0.188583451  0.416530256 -0.129759608
## security_deposit      -0.008853095 -0.157168549  0.211070823
## minimum_nights        -0.023701591 -0.027422516 -0.038236111
## maximum_nights        -0.009409971 -0.013337843 -0.009753806
##                         bathrooms     bedrooms        beds     bed_type
## host_since            -0.058208224 -0.085290675 -0.07633186 -0.013536577
## host_location          0.067502164  0.089568751  0.03559878  0.062301376
## host_response_time     0.035640437  0.032817308  0.08726944 -0.003084712
## host_acceptance_rate  -0.022239768 -0.031243752 -0.02609913 -0.027493110
## host_is_superhost      0.045013293  0.031710635  0.01898720  0.014919762
## neighbourhood_cleansed  0.052354265  0.041360229  0.05078660  0.025828964
## is_location_exact      0.086099917  0.059592341  0.04119945  0.076112703
## property_type          0.227967010  0.057589730  0.05467758  0.008502794
## room_type             -0.077615722 -0.270566917 -0.35583001 -0.233769264
## accommodates           0.351401359  0.724917621  0.82453701  0.139756953

## bathrooms              1.000000000  0.435993647  0.36022155  0.055779595
## bedrooms               0.435993647  1.000000000  0.72506124  0.069284549
## beds                   0.360221545  0.725061240  1.00000000  0.085154525
## bed_type               0.055779595  0.069284549  0.08515453  1.000000000
## price                  0.100035955  0.038157533 -0.04605663 -0.058385407
## security_deposit       0.064470141  0.187037470  0.19255497 -0.047519700
## minimum_nights         0.021936011 -0.004920753 -0.02028719 -0.002072280
## maximum_nights        -0.007413815 -0.005657675 -0.01018974  0.002947237
##                            price security_deposit minimum_nights
## host_since             0.04336999     -0.097072380  -0.0282944019
## host_location         -0.04381201     -0.105459775   0.0175004833
## host_response_time     0.05846075      0.091120874  -0.0342130701
## host_acceptance_rate  -0.06186767     -0.000993886   0.0005110294
## host_is_superhost      0.05620508      0.063025819  -0.0241505881
## neighbourhood_cleansed -0.06924734      0.046098627  -0.0192245574
## is_location_exact     -0.02324765     -0.020237476   0.0398061870
## property_type          0.18858345     -0.008853095  -0.0237015906
## room_type              0.41653026     -0.157168549  -0.0274225161
## accommodates          -0.12975961      0.211070823  -0.0382361105
## bathrooms              0.10003596      0.064470141   0.0219360114
## bedrooms               0.03815753      0.187037470  -0.0049207533
## beds                  -0.04605663      0.192554972  -0.0202871892
## bed_type              -0.05838541     -0.047519700  -0.0020722803
## price                  1.00000000     -0.077336523  -0.0133681983
## security_deposit      -0.07733652      1.000000000   0.0041613693
## minimum_nights        -0.01336820      0.004161369   1.0000000000
## maximum_nights        -0.02131685     -0.010044421  -0.0041008936
##                       maximum_nights
## host_since              -0.018212350
## host_location           -0.009857720
## host_response_time       0.016256911
## host_acceptance_rate    -0.019271561
```

```
## host_is_superhost        -0.005999207
## neighbourhood_cleansed    0.014546907
## is_location_exact         0.006755548
## property_type            -0.009409971
## room_type                -0.013337843
## accommodates             -0.009753806
## bathrooms                -0.007413815
## bedrooms                 -0.005657675
## beds                     -0.010189745
## bed_type                  0.002947237
## price                    -0.021316846
## security_deposit         -0.010044421
## minimum_nights           -0.004100894
## maximum_nights            1.000000000
```

This is a lot of information. We can make a better sense of it by visualizing the information.

Let us visualize the relationships between our dependent variable and the other variables.

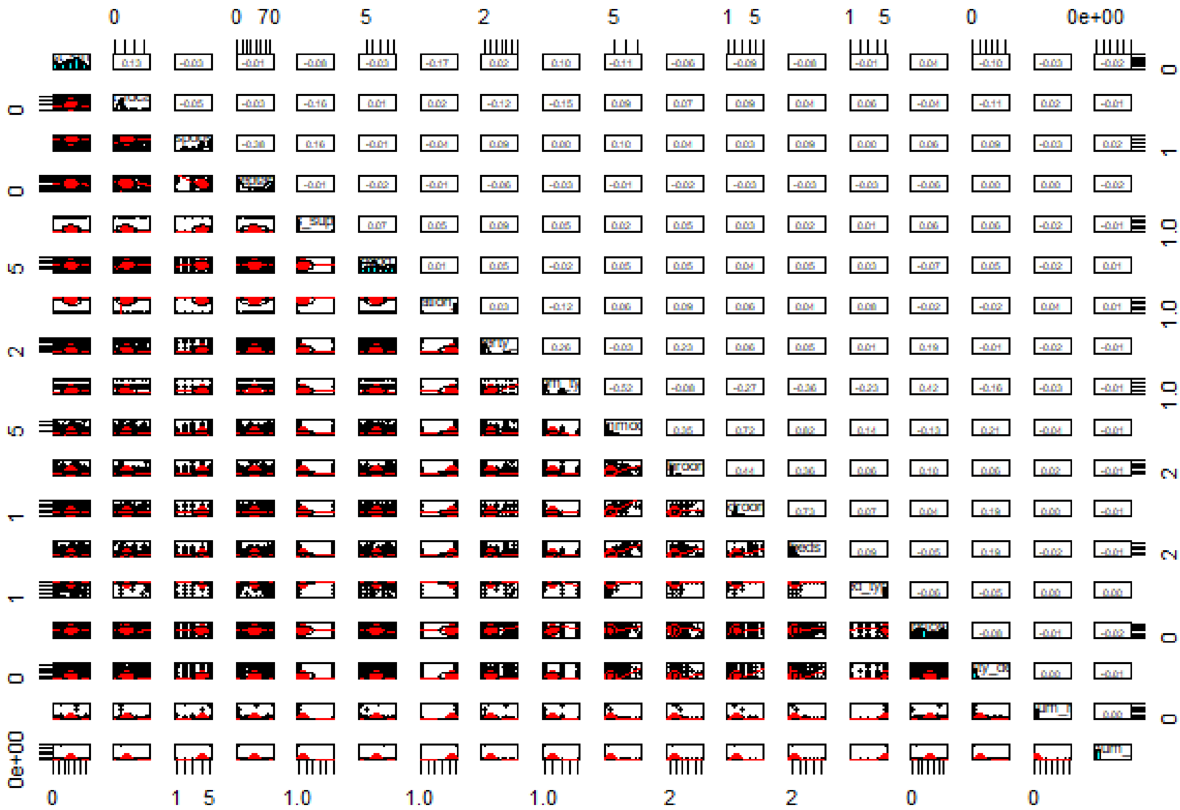
```
suppressWarnings(library(psych))
```

```
##
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
```

```
##
##     %+%, alpha
```

```
pairs.panels(Daily[])
```

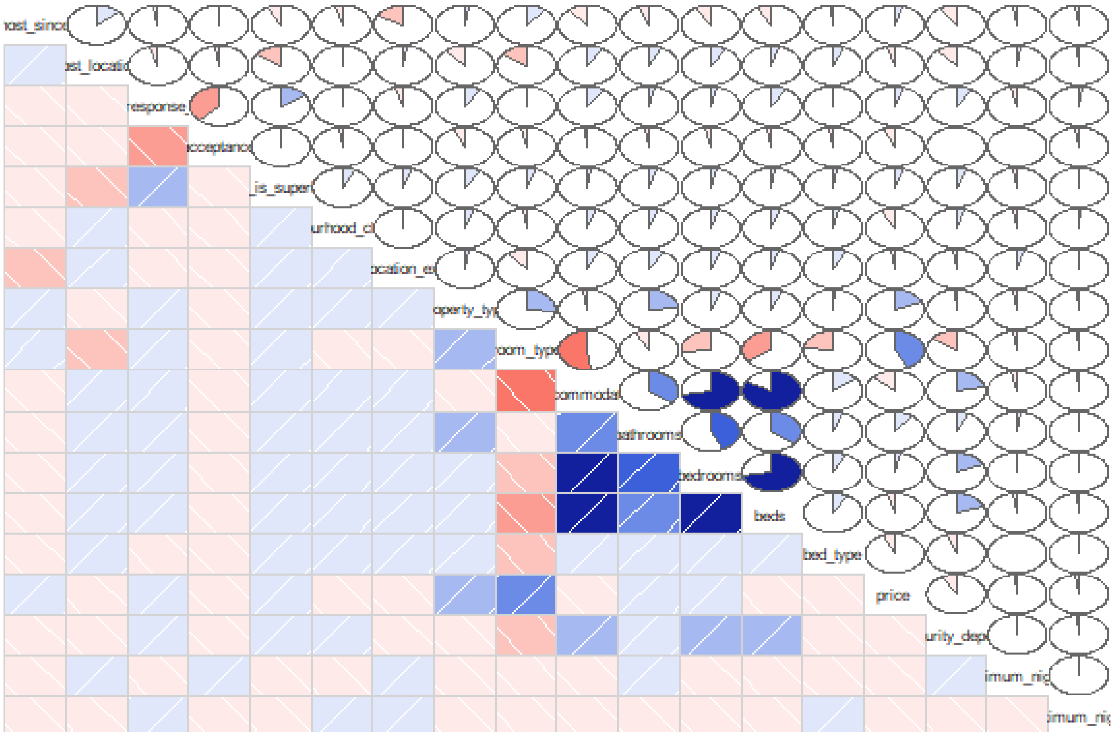


We can see that the plot isn't as clear. Let us use another type of visualization.

```
suppressWarnings(library(corrgram))

Daily_price_coef<-cor(Daily[,])

corrgram(Daily_price_coef,lower.panel = panel.shade,upper.panel = panel.pie)
```



We can see that there is a corellation between the price per night of the listing and the parameters we selected.

We can reduce this collinearity between the features by doing a Principle Componenet analysis. I will then build the models on the principle components which explain the maximum variance.

```
str(Daily)
```

```
## 'data.frame':    3585 obs. of  18 variables:
##  $ host_since            : num  880 223 6 388 625 ...
##  $ host_location         : num  31 31 31 31 31 31 95 31 31 31 ...
##  $ host_response_time    : num  2 5 4 4 5 4 5 4 5 5 ...
##  $ host_acceptance_rate  : num  73 2 61 23 2 68 69 2 2 2 ...
##  $ host_is_superhost     : num  1 1 2 1 2 2 1 2 2 2 ...
##  $ neighbourhood_cleansed: num  19 19 19 19 19 19 19 19 19 19 ...
##  $ is_location_exact     : num  2 2 2 1 2 2 1 2 2 2 ...
##  $ property_type         : num  10 2 2 10 10 6 2 10 6 2 ...
##  $ room_type             : num  1 2 2 2 2 2 1 2 2 1 ...
##  $ accommodates          : num  4 2 2 4 2 2 3 2 2 5 ...
##  $ bathrooms             : num  4 3 3 3 4 3 3 5 3 3 ...
##  $ bedrooms              : num  3 2 2 2 2 2 2 2 2 3 ...
```

```
##  $ beds                 : num  4 2 2 3 3 2 3 2 3 3 ...
##  $ bed_type             : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ price                : num  144 276 276 293 299 293 10 293 265 127 ...
##  $ security_deposit     : num  1 53 1 7 1 1 1 1 1 22 ...
##  $ minimum_nights       : num  2 2 3 1 2 2 1 1 2 4 ...
##  $ maximum_nights       : num  1125 15 45 1125 31 ...
```

We can see that we have 18 numeric variables which we have to build a model on. Since all the variables are numeric and since there obviously is quite a lot of correlation betweeen the variables.

First, let us normalize the data.

```
normalize<-function(x){
  (x-min(x))/(max(x)-min(x))
}

Daily_normalized<-as.data.frame(lapply(Daily, normalize))

summary(Daily_normalized)
```

```
##    host_since     host_location    host_response_time host_acceptance_rate
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000     Min.   :0.00000
##  1st Qu.:0.3055   1st Qu.:0.1705   1st Qu.:0.5000     1st Qu.:0.01389
##  Median :0.5367   Median :0.1705   Median :0.7500     Median :0.54167

##  Mean   :0.5166   Mean   :0.3405   Mean   :0.7391     Mean   :0.48440
##  3rd Qu.:0.7297   3rd Qu.:0.5284   3rd Qu.:1.0000     3rd Qu.:0.91667
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000     Max.   :1.00000
##  host_is_superhost neighbourhood_cleansed is_location_exact
##  Min.   :0.0000    Min.   :0.0000         Min.   :0.0000
##  1st Qu.:0.0000    1st Qu.:0.1667         1st Qu.:1.0000
##  Median :0.0000    Median :0.4167         Median :1.0000
##  Mean   :0.1135    Mean   :0.4411         Mean   :0.8591
##  3rd Qu.:0.0000    3rd Qu.:0.7083         3rd Qu.:1.0000
##  Max.   :1.0000    Max.   :1.0000         Max.   :1.0000
##  property_type       room_type       accommodates       bathroms
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.00000   Min.   :0.0000
##  1st Qu.:0.07692   1st Qu.:0.0000   1st Qu.:0.06667   1st Qu.:0.1818
##  Median :0.07692   Median :0.0000   Median :0.06667   Median :0.1818
##  Mean   :0.22096   Mean   :0.2145   Mean   :0.13609   Mean   :0.2219
##  3rd Qu.:0.38462   3rd Qu.:0.5000   3rd Qu.:0.20000   3rd Qu.:0.1818
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.00000   Max.   :1.0000
##     bedrooms          beds           bed_type          price
##  Min.   :0.000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.200   1st Qu.:0.1000   1st Qu.:1.0000   1st Qu.:0.2136
##  Median :0.200   Median :0.1000   Median :1.0000   Median :0.4427
##  Mean   :0.251   Mean   :0.1606   Mean   :0.9775   Mean   :0.4916
```

```
##  3rd Qu.:0.400   3rd Qu.:0.2000   3rd Qu.:1.0000   3rd Qu.:0.8173  
##  Max.   :1.000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  
##  security_deposit minimum_nights     maximum_nights     
##  Min.   :0.0000   Min.   :0.000000   Min.   :0.0000000  
##  1st Qu.:0.0000   1st Qu.:0.000000   1st Qu.:0.0000036  
##  Median :0.0000   Median :0.003344   Median :0.0000112  
##  Mean   :0.1724   Mean   :0.007262   Mean   :0.0002872  
##  3rd Qu.:0.2222   3rd Qu.:0.006689   3rd Qu.:0.0000112  
##  Max.   :1.0000   Max.   :1.000000   Max.   :1.0000000
```

Let us make training and testing datasets of the model.

```
suppressWarnings(library(caret))
```

```
## Loading required package: lattice

## 
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
## 
##     lift
```

```
roompart<-createDataPartition(Daily_normalized$price,p=0.8,list = FALSE)

Price_train<-Daily_normalized[roompart,]
Price_test<-Daily_normalized[-roompart,]
dim(Price_train)
```

```
## [1] 2870   18
```

```
dim(Price_test)
```

```
## [1] 715  18
```

Now let us do the PCA.[2]

```
pca_dailyprice<-prcomp(Price_train,scale. = T)

names(pca_dailyprice)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

Thus we see that the 5 important features are now formed. Let us see what the output of the mean of variables.

```
pca_dailyprice$center
```

```
##            host_since          host_location     host_response_time
##          0.5163390135           0.3436490339           0.7360627178
##   host_acceptance_rate      host_is_superhost neighbourhood_cleansed
##          0.4827187379           0.1094076655           0.4414779326
##      is_location_exact          property_type              room_type
##          0.8585365854           0.2195658001           0.2137630662
##          accommodates              bathrooms               bedrooms
##          0.1354703833           0.2225213811           0.2492682927
##                  beds               bed_type                  price
##          0.1594076655           0.9777874564           0.4907260979
##      security_deposit         minimum_nights         maximum_nights
##          0.1713704994           0.0071574237           0.0003568371
```

Let us also check the SD of the variables.

```
pca_dailyprice$sdev
```

```
##  [1] 1.7728611 1.3341506 1.1764791 1.1435782 1.0856280 1.0354416 1.0015192
##  [8] 0.9986511 0.9600370 0.9171942 0.9034188 0.8817290 0.8607453 0.7859460
## [15] 0.7573708 0.6564160 0.5123807 0.3897577
```

Finally, let us check the principle component loading which is given by the rotation feature.

```
pca_dailyprice$rotation
```

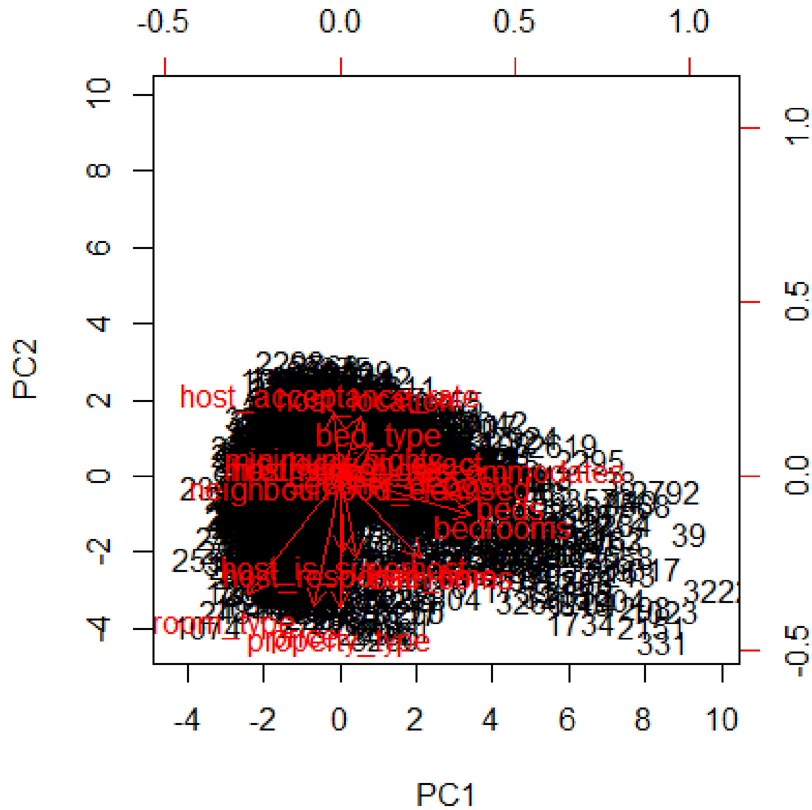
```
##                                  PC1          PC2         PC3          PC4
## host_since             -1.014103e-01  0.01415163 -0.30417098  0.435248690
## host_location           7.425774e-02  0.21407579 -0.29155975  0.405032935
## host_response_time      5.984121e-02 -0.29102040  0.48974266  0.390111701
## host_acceptance_rate   -2.384979e-02  0.22791983 -0.36083032 -0.509797644
## host_is_superhost       1.097356e-02 -0.26830295  0.32287921 -0.254918031
## neighbourhood_cleansed  5.868681e-02 -0.04101195  0.09948239 -0.077077726
## is_location_exact       7.650565e-02  0.03322546  0.06200053 -0.314282054
```

```
## property_type          -9.834314e-05 -0.47356858 -0.15678700 -0.089242029
## room_type              -3.298792e-01 -0.42021945 -0.18953753 -0.048220407
## accommodates            5.177162e-01  0.02158702 -0.01996392  0.031507258
## bathrooms               2.899788e-01 -0.28800922 -0.24659093 -0.057439066
## bedrooms                4.680350e-01 -0.14034233 -0.16892879  0.003905831
## beds                    4.911115e-01 -0.08652961 -0.08397932  0.016828690
## bed_type                1.110724e-01  0.11805420  0.01821047  0.112896335
## price                  -9.618243e-02 -0.46270393 -0.27825583  0.031830058
## security_deposit        1.661554e-01 -0.02272517  0.29747704 -0.160744279
## minimum_nights         -1.562333e-02  0.04677680 -0.02979480 -0.099241188
## maximum_nights         -3.021782e-03  0.01929035  0.09997373  0.020067415
##                                PC5          PC6         PC7          PC8
## host_since             -0.17057419 -0.276344462 -0.04563217  0.073221643
## host_location           0.18945768  0.001791312  0.13755910  0.178939023
## host_response_time      0.02616423  0.081154983 -0.04242929  0.027711315
## host_acceptance_rate   -0.18664487 -0.172102041 -0.05944790 -0.062613209
## host_is_superhost       0.06073483 -0.263641662 -0.15921411  0.071325481
## neighbourhood_cleansed  0.07002084 -0.650394903  0.36133736  0.395026071
## is_location_exact       0.58222423  0.197186682 -0.05441008 -0.055897121
## property_type           0.13476545 -0.201383460  0.03043449  0.045235942
## room_type              -0.08132332  0.027167940  0.05930317 -0.008323308
## accommodates           -0.07601347  0.010601003 -0.03955096 -0.050275464
## bathrooms               0.12626238 -0.039289622  0.09136510  0.048633797
## bedrooms               -0.07347693  0.078222426  0.03244573 -0.031399991
## beds                   -0.11262130  0.026838538 -0.01068508 -0.046395259
## bed_type                0.51712893 -0.254537207 -0.27827780 -0.055831782
## price                   0.04399492  0.231101740 -0.03806973 -0.079431293
## security_deposit       -0.43107893  0.065363816  0.06222613  0.147868240
## minimum_nights          0.15165715  0.424354559  0.39967731  0.630164690
## maximum_nights          0.09564508 -0.060723245  0.74616544 -0.595925910
##                                 PC9        PC10        PC11         PC12
## host_since             -0.313767654  0.02662220 -0.41636024  0.004294754
## host_location           0.280686485 -0.31403710 -0.26207912  0.375598525
## host_response_time     -0.003197123 -0.07068170 -0.08150202 -0.076270070
## host_acceptance_rate   -0.175358721 -0.16108426 -0.08547381  0.059879990
## host_is_superhost      -0.255110929 -0.66017874 -0.26470013  0.066508659
## neighbourhood_cleansed  0.323235935  0.01528628  0.31462551  0.045568225
## is_location_exact       0.314365697  0.08297496 -0.37853098  0.119599538
## property_type          -0.093290408  0.43257501 -0.30532534 -0.187820153
## room_type               0.075069658 -0.01916151  0.10294795  0.069606048
## accommodates           -0.035540877 -0.06326569  0.06169151 -0.105982940
## bathrooms               0.046751912  0.07318094 -0.17691497 -0.034815657
## bedrooms                0.019286774 -0.10057016  0.08946403  0.024578194
## beds                   -0.039725045 -0.04933952  0.08248870 -0.135818090
## bed_type               -0.497688666  0.21983041  0.31538621  0.317643449
## price                  -0.003142283 -0.19449593  0.35498564  0.419554758
## security_deposit       -0.057337685  0.35469556 -0.21763865  0.676774898
## minimum_nights         -0.450144958 -0.04687578  0.03382345 -0.125118987
## maximum_nights         -0.224177882 -0.05672766 -0.06277417  0.080507806
##                                PC13        PC14         PC15          PC16
## host_since             -0.549084472  0.07643306 -0.115295518 -0.0262286650
```

```
## host_location           0.392507064 -0.21605071  0.111302265  0.0774715057
## host_response_time      0.145672479 -0.18208522 -0.646010435  0.0964488835
## host_acceptance_rate    0.188523985 -0.24238007 -0.562762262  0.0514894797
## host_is_superhost       0.013364639  0.06264161  0.272490774 -0.0255791012
## neighbourhood_cleansed -0.199079687 -0.03861043 -0.120832328 -0.0529672078
## is_location_exact      -0.453460746 -0.08721527 -0.147263253  0.0868156507
## property_type           0.279754738 -0.44115272  0.202470371 -0.1964409375
## room_type              -0.043980831 -0.01567096  0.027936389  0.7516352183
## accommodates           -0.077539668 -0.16233485 -0.006894115 -0.0404962621
## bathrooms               0.270318336  0.75110544 -0.224557565 -0.0518212673
## bedrooms               -0.118879161 -0.04171002  0.129641070  0.2621242292
## beds                   -0.143452358 -0.20429511  0.038303555  0.1428759411
## bed_type                0.098869256  0.02560514 -0.011494465  0.2014834530
## price                  -0.191484587 -0.10014556 -0.127781060 -0.4803455756
## security_deposit        0.002834199  0.02908483  0.046421299  0.0397989266
## minimum_nights         -0.041405131 -0.04361635 -0.023888832  0.0213381108
## maximum_nights         -0.009409681 -0.01652666  0.006460469 -0.0007700178
##                               PC17          PC18
## host_since             -0.02238007  0.0172608778
## host_location           0.08022021 -0.0055262415
## host_response_time     -0.09860965 -0.0406843900
## host_acceptance_rate   -0.05988540 -0.0257547520
## host_is_superhost       0.02467476 -0.0016614376
## neighbourhood_cleansed -0.01370962 -0.0033455836
## is_location_exact       0.02449451  0.0107772796
## property_type          -0.06811712  0.0084278415
## room_type               0.18487615  0.2046142175
## accommodates            0.21039705  0.7883571568
## bathrooms               0.10130351  0.0006294514
## bedrooms               -0.76282149 -0.1294606454
## beds                    0.54636761 -0.5624403131
## bed_type                0.01536802 -0.0100903024
## price                   0.03842591 -0.0009543245
## security_deposit        0.05103920  0.0045625857
## minimum_nights          0.01084673  0.0227619632
## maximum_nights          0.01260655  0.0060555624
```

Let us now plot the principal components.

```
biplot(pca_dailyprice,scale = 0)
```



Now, let us calculate the standard deviation of each principal component.

```
SDv<-pca_dailyprice$sdev
```

Let us compute variance.

```
Var<-SDv^2
```

We still need to see the proportion of variance explained.

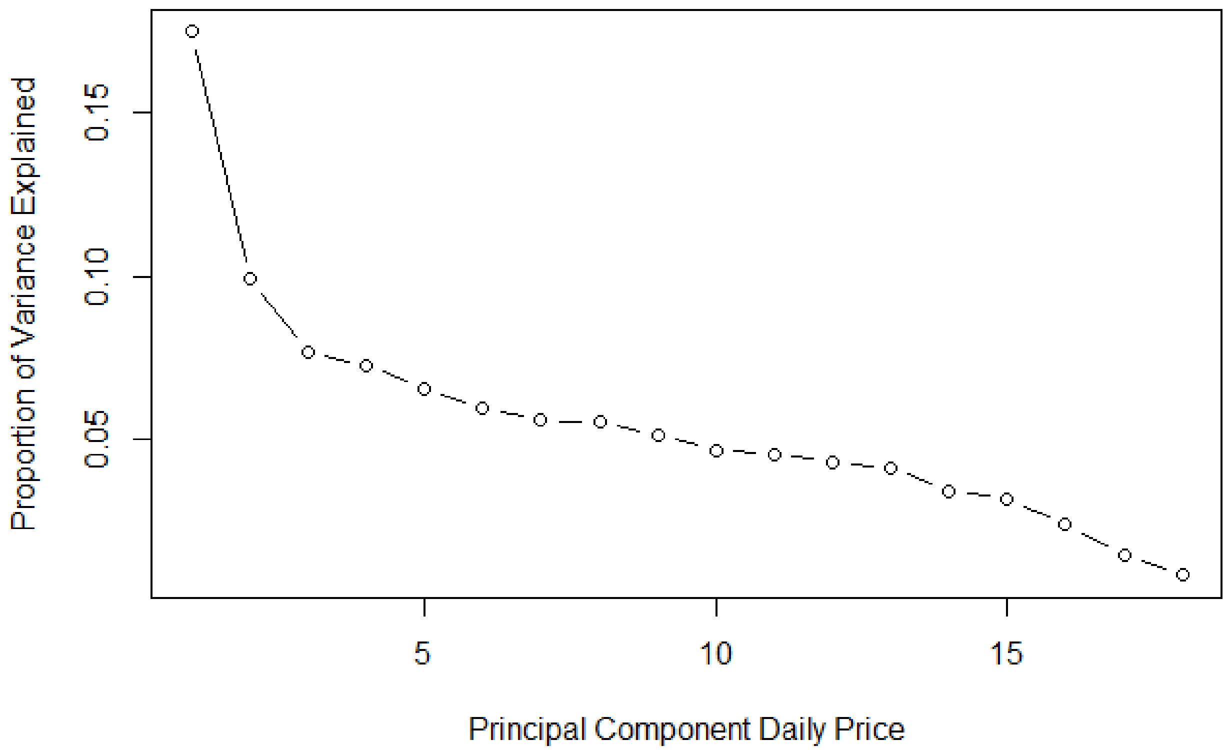
```
price_var<-Var/sum(Var)
price_var[1:10]
```

```
##  [1] 0.17461313 0.09888655 0.07689462 0.07265395 0.06547712 0.05956330
##  [7] 0.05572449 0.05540578 0.05120395 0.04673584
```

We can see that the first principal component explains 17.5% variance. Second component explains 9.8% variance. But we still need to decide how many components to include.

We will build a scree plot to help us out. This plot will give us a decending order of values of variance.

```
plot(price_var,xlab = "Principal Component Daily Price",ylab = "Proportion of Varianc
```



We can see that 13 principal components explain about 95% of the variance in our data. Let us use the first 13 components for modelling.

We will now add the training set with the principal components.

```
data_train<-data.frame(Price_train$price,pca_dailyprice$x)
```

We are interested in getting the first 13 PCAs.

```
data_train<-data_train[,1:14]
```

It is important for us to do the same PCA transformations on both training and testing datasets otherwise they will have unequal variance and hence their vectors will show different directions.

Hence,now let us transform the test data into PCA.

```
data_test<-predict(pca_dailyprice,newdata = Price_test)
data_test<-as.data.frame(data_test)
```

Again, let us select the 13 principal components,

```
data_test<-data_test[,1:13]
```

LINEAR REGRESSION MODEL WITH PCA

Now let us build that regression model.

```
pca_lm<-lm(data_train$Price_train.price~.,data = data_train)
```

Now, let us predict the value.

```
lm_pred_price<-predict(pca_lm,data_test)
```

Let us see what the RMSE of our model is.

Let us write a function to calculate the Root Mean Squared error.

```
RMSE_fun<-function(Pred,TargetVar){
Error<-Pred-TargetVar
SqError<-Error^2
MeanSqerror<-mean(SqError)
RootMSE<-MeanSqerror^0.5
return(RootMSE)
RootMSE
}
```

Let us see the RMSE.

```
RMSE_fun(lm_pred_price,data_test)
```

```
## [1] 1.254943
```

Thus we used PCA to build the linear regression model. Using PCA helped in getting rid of the collinearity which existed in our 'independent'variables. By doing PCA we essentially extracted more information from a lower dimensional plane.

Cross Validation for Linear Regression Model:

We can see that our dataset has some 3585 rows. It is not a huge dataset. In cases like these, one can use the same dataset for training and testing instead of making two separate datasets. However, when the linear regression model is trained and tested on the same data, there is often a risk of the model overfitting. To tackle this problem, we can use a technique called Cross Validation.

We will consider our entire normalized dataframe to do the cross validation.

Let us create 10 folds first.

```
set.seed(123)
folds<-createFolds(Daily_normalized$price,k=10)
```

Now, we will need to apply a series of similar steps to all the 10 folds. Essentially, every time one fold will act as the test dataset and the remaining 9 folds will be the training datasets.

Since I have done a PCA and predicted the price, I will now use the entire Normalized dataset to do the cross validation. The idea here is to compare the RMSE of the Linear Regression model along with PCA and The Linear Regression Model with cross validation with all the independent variables included.

I will use a function we developed for the Data Management and Processing course. This function will take the formula, data and the number of folds as its input. It will then use the crossv_kfold function from the modelr package to make folds of the data. Then, using mutate, it will join new columns to the dataframe with the folds. These columns will use the map function to apply the linear regression to each element of the vector. Ultimately, it will return the mean RMSE of all the folds as it's output.

```
suppressWarnings(library(modelr))
```

```
##
## Attaching package: 'modelr'

## The following object is masked from 'package:psych':
```

```
##
##     heights
```

```
cross_val<-function(formula,data,folds_num){
  cvd<-crossv_kfold(data,folds_num)
  cvd<-cvd%>%mutate(mod=map(train,~lm(formula,data = .)))%>% mutate(rmse=map2_dbl(mod
  c(cross_val_rmse=mean(cvd$rmse))

}
```

Now, let us use this function to do a 10 fold cross validation of our dataset.

```
cross_val(price~.,Daily_normalized,10)
```

```
## cross_val_rmse
##       5.472549
```

We can see that the RMSE of the model with PCA was lower than the model with cross validation.

RANDOM FOREST MODEL USING PCA

I would like to build a price prediction model using the Random Forest algorithm and compare it with the Linear Regression model.

We already have done the PCA with our training and testing datasets. Let us use those dataframes for our Random Forest Model.

Now let us try building a Random Forest Model and compare the results.

```
suppressWarnings(library(randomForest))
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:psych':
##
```

```
##     outlier

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
Randfor_Price_pca<-randomForest(data_train$Price_train.price~.,data = data_train,impo
```

Now, let us predict the Price using Random Forest.

```
RFPred_pca<-predict(Randfor_Price_pca,data_test)
```

And let us now calculate the RMSE.

```
RMSE_fun(RFPred_pca,data_test)
```

```
## [1] 1.244774
```

We see that the RMSE has decreased using Random Forest. Thus, the model prediction accuracy has increased using Random FOrest algorithm compared to Linear regression.

Cross Validation for Random Forest

I will use the same function as above for cross validation. The only slight modification is that in the function, I will now use the Random Forest formula.

```
suppressWarnings(library(modelr))

cross_val_rf<-function(formula,data,folds_num){
  cvd<-crossv_kfold(data,folds_num)
  cvd<-cvd%>%mutate(mod=map(train,~randomForest(formula,data = .)))%>% mutate(rmse=ma
  c(cross_val_rmse=mean(cvd$rmse))

}
```

Now let us see the RMSE of our Random Forest with 10-fold cross validation.

```
cross_val_rf(price~.,Daily_normalized,10)
```

```
## cross_val_rmse
##       0.247854
```

Thus we can see that the RMSE of the cross validated Random forest model is the lowest with 0.24. This model is likely the best one for our prediction problem.

EVALUATING MODEL PERFORMANCE

We saw 4 models for the prediction problem- Linear regression with PCA, Linear Regression with Cross Validation, Random Forest with PCA and Random Forest with cross validation. OUt of these 4, the rmse for the random forest model with cross validation was the lowest indicating that the model was the best among the lot.

Thus, anyone who stays in Boston and wants to know how much their place is worth if they were to put it up on Airbnb can get to know the daily rate of their place with the help of the above models. This completes the Yeild prediction part of my project.

I am also curious to find out who these 'Superhosts' are. Airbnb says these are highly experienced hosts who ofte have their place rented out. I will build the following classification models for this purpose.

CLASSIFICATION MODELS

Problem statement: I want to classify the host as a superhost or a not. Airbnb classifies the 'more experienced' hosts as superhosts. What I want to find out is who exactly gets classified as superhost. I will build a model to classify the host.

```
Host_info<- Listings%>% select(host_response_rate,host_response_time,host_is_superhos
```

```
str(Host_info)
```

```
## 'data.frame':    3585 obs. of  16 variables:
##  $ host_response_rate       : Factor w/ 53 levels "0%","10%","100%",..: 53 3 3 3 3
##  $ host_response_time       : Factor w/ 5 levels "a few days or more",..: 2 5 4 4
##  $ host_is_superhost        : Factor w/ 2 levels "f","t": 1 1 2 1 2 2 1 2 2 2 ...
```

```
## $ host_neighbourhood       : Factor w/ 54 levels "","Allston-Brighton",..: 41 41
## $ host_total_listings_count: int  1 1 1 1 1 2 5 2 1 2 ...
## $ neighbourhood_cleansed   : Factor w/ 25 levels "Allston","Back Bay",..: 19 19 1
## $ room_type                : Factor w/ 3 levels "Entire home/apt",..: 1 2 2 2 2 2
## $ accommodates             : int  4 2 2 4 2 2 3 2 2 5 ...
## $ bathrooms                : num  1.5 1 1 1 1.5 1 1 2 1 1 ...
## $ bedrooms                 : int  2 1 1 1 1 1 1 1 1 2 ...
## $ price                    : Factor w/ 324 levels "$1,000.00","$1,235.00",..: 144
## $ number_of_reviews        : int  0 36 41 1 29 8 57 67 65 33 ...
## $ review_scores_rating     : int  NA 94 98 100 99 100 90 96 96 94 ...
## $ cancellation_policy      : Factor w/ 4 levels "flexible","moderate",..: 2 2 2 2
## $ instant_bookable         : Factor w/ 2 levels "f","t": 1 2 1 1 1 1 1 1 1 1 ...
## $ reviews_per_month        : num  NA 1.3 0.47 1 2.25 1.7 4 2.38 5.36 1.01 ...
```

We will first see if there are any NA values in our dataframe.

```
sapply(Host_info,function(x) sum(is.na(x)))
```

```
##        host_response_rate        host_response_time
##                         0                         0
##         host_is_superhost        host_neighbourhood
##                         0                         0
## host_total_listings_count    neighbourhood_cleansed
##                         0                         0
##                 room_type              accommodates
##                         0                         0
##                 bathrooms                  bedrooms

##                        14                        10
##                     price         number_of_reviews
##                         0                         0
##      review_scores_rating       cancellation_policy
##                       813                         0
##          instant_bookable         reviews_per_month
##                         0                       756
```

Now let us impute the NAs one by one.

Imputing Bathrooms by the mode.

```
Mode_bathroom<-modeVal(Host_info$bathrooms)
```

```
Host_info$bathrooms<-ifelse(is.na(Host_info$bathrooms),Mode_bathroom,Host_info$bathro
```

```
unique(Host_info$bathrooms)
```

```
##  [1] 1.5 1.0 2.0 0.0 2.5 3.5 3.0 0.5 4.5 4.0 5.0 6.0
```

Imputing Bedrooms by the mode.

```
Mode_bedrooms<-modeVal(Host_info$bedrooms)

Host_info$bedrooms<-ifelse(is.na(Host_info$bedrooms),Mode_bedrooms,Host_info$bedrooms
unique(Host_info$bedrooms)
```

```
## [1] 2 1 0 3 4 5
```

Thus we have imputed the bedrooms as well.

Now, let us impute the beds.

```
Mode_beds<-modeVal(Host_info$bedrooms)

Host_info$bedrooms<-ifelse(is.na(Host_info$bedrooms),Mode_beds,Host_info$bedrooms)
unique(Host_info$bedrooms)
```

```
## [1] 2 1 0 3 4 5
```

We have now gotten rid of the NA values in the Beds column as well.

Now, let us impute the reviews_score_rating.

```
Median_ratings<-median(Host_info$review_scores_rating,na.rm = TRUE)
Host_info$review_scores_rating<-ifelse(is.na(Host_info$review_scores_rating),Median_r

unique(Host_info$review_scores_rating)
```

```
##  [1]  94  98 100  99  90  96  80  97  91  95  88  92  87  93  73  82  20
## [18]  89  81  78  74  60  86  85  75  79  70  83  64  84  40  68  67  48
## [35]  58  62  76  77  71  65  53  47  72  46  50  66  69  55
```

Lastly, let us impute the reviews per month.

```
mean_reviews<-mean(Host_info$reviews_per_month,na.rm = TRUE)
Host_info$reviews_per_month<-ifelse(is.na(Host_info$reviews_per_month),mean_reviews,H

summary(Host_info$reviews_per_month)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.010   0.640   1.910   1.971   2.130  19.150
```

Let us now confirm if there are any NA values anywhere in our dataframe.

```
anyNA(Host_info)
```

```
## [1] FALSE
```

There are no more NA values left.

We have to develop a Support Vector Machine model to predict whether the host is superhost or not. Let us first convert our data into numeric values.

Let us write a function to normalize the data.

```
normalize<-function(x){
  (x-min(x))/(max(x)-min(x))
}
```

To do so, let us first subset all the factor data into our dataframe.

```
suppressWarnings(library(ade4))

Host_nom<-Host_info[c("host_response_rate","host_response_time","host_neighbourhood",
converted_fact<-acm.disjonctif(Host_nom)
converted_fact<-as.data.frame(lapply(converted_fact, normalize))
```

Now, let us store our numeric features in a dataframe.

```
Numeric_feat<-Host_info[c("host_total_listings_count","accommodates","bathrooms","bed

Numeric_feat$price<-as.numeric(Numeric_feat$price)

Numeric_feat<-as.data.frame(lapply(Numeric_feat, normalize))
```

Finally let us convert our decision variable into factor.

```
Dec_var<-as.factor(Host_info$host_is_superhost)
```

Now, let us make a dataframe with all the dummy variables, the numeric variables and our decision variable.

```
Superhost<-cbind(converted_fact,Numeric_feat,Dec_var)
```

Let us now normalize the data.

```
Host_scaled<-as.data.frame(lapply(Superhost[,1:166], normalize))

Superhost_final<-cbind(Host_scaled,Dec_var)
```

Let us now create Training and Testing datasets.

```
set.seed(12345)
suppressWarnings(library(caret))
Hostpart<-createDataPartition(Superhost_final$Dec_var,p=0.8,list = FALSE)

Superhost_train<-Superhost_final[Hostpart,]

Superhost_test<-Superhost_final[-Hostpart,]
```

```
dim(Superhost_train)
```

```
## [1] 2869  167
```

```
dim(Superhost_test)
```

```
## [1] 716 167
```

Let us now train the model on our training dataset.

```
NeuModel<-train(Superhost_train[,-167],Superhost_train$Dec_var,method = "nnet",trCont
```

```
## # weights:  169
## initial  value 2470.724063
## iter  10 value 717.722375
## iter  20 value 607.701077
## iter  30 value 549.746767
## iter  40 value 544.682035
## iter  50 value 541.852993
## iter  60 value 538.911526
## iter  70 value 534.488280
## iter  80 value 533.472955
## iter  90 value 530.191313
## iter 100 value 529.372967
## final  value 529.372967
## stopped after 100 iterations
## # weights:  505
## initial  value 1942.225926
## final  value 913.319399
## converged
## # weights:  841
## initial  value 2817.576426
## final  value 913.319397
## converged
## # weights:  169
## initial  value 1395.185408
## iter  10 value 805.957100
## iter  20 value 751.168540
## iter  30 value 656.379162
## iter  40 value 609.266546
## iter  50 value 586.334946
## iter  60 value 569.890864
## iter  70 value 562.590172
## iter  80 value 559.614174
## iter  90 value 559.123272
## iter 100 value 559.109894
## final  value 559.109894
## stopped after 100 iterations
## # weights:  505
## initial  value 1494.674752
## iter  10 value 743.933459
## iter  20 value 645.299165
```

```
## iter  30 value 594.762259
## iter  40 value 566.337466
## iter  50 value 551.387384
## iter  60 value 537.511716
## iter  70 value 527.766325
## iter  80 value 512.522570
## iter  90 value 499.054877
## iter 100 value 494.994522
## final  value 494.994522
## stopped after 100 iterations
## # weights:  841
## initial  value 1478.348632
## iter  10 value 898.331501
## iter  20 value 736.835029
## iter  30 value 620.962520
## iter  40 value 532.841750
## iter  50 value 478.375306
## iter  60 value 444.113501
## iter  70 value 424.163019
## iter  80 value 410.263212
## iter  90 value 401.927435
## iter 100 value 396.803123
## final  value 396.803123
## stopped after 100 iterations
## # weights:  169
## initial  value 2866.580967
## iter  10 value 790.384038
## iter  20 value 652.027750
## iter  30 value 566.196624
## iter  40 value 543.515348
## iter  50 value 540.470686
## iter  60 value 538.149231
## iter  70 value 536.766659
## iter  80 value 534.216862
## iter  90 value 533.923259
## iter 100 value 533.806800
## final  value 533.806800
## stopped after 100 iterations
## # weights:  505
## initial  value 2365.573959
## iter  10 value 917.272607
## iter  20 value 913.424316
## iter  30 value 913.337333
## iter  30 value 913.337327
## final  value 913.337327
## converged
## # weights:  841
## initial  value 1790.787597
## iter  10 value 733.602834
## iter  20 value 575.293658
## iter  30 value 472.732142
```

```
## iter  40 value 427.913670
## iter  50 value 367.548168
## iter  60 value 325.587423
## iter  70 value 302.404352
## iter  80 value 285.703694
## iter  90 value 276.571893
## iter 100 value 273.269129
## final  value 273.269129
## stopped after 100 iterations
## # weights:  169
## initial  value 2057.288624
## iter  10 value 807.745477
## iter  20 value 689.390149
## iter  30 value 584.491027
## iter  40 value 554.071004
## iter  50 value 551.956016
## iter  60 value 550.494648
## iter  70 value 549.900240
## iter  80 value 549.286617
## iter  90 value 549.268571
## iter 100 value 549.259670
## final  value 549.259670
## stopped after 100 iterations
## # weights:  505
## initial  value 1707.055690
## iter  10 value 768.662044
## iter  20 value 582.395650
## iter  30 value 478.088743
## iter  40 value 426.387318
## iter  50 value 403.549479
## iter  60 value 380.053505
## iter  70 value 357.413880
## iter  80 value 350.344932
## iter  90 value 350.015444
## iter 100 value 350.014011
## final  value 350.014011
## stopped after 100 iterations
## # weights:  841
## initial  value 3105.319366
## iter  10 value 893.642728
## iter  20 value 737.487106
## iter  30 value 666.969955
## iter  40 value 603.487560
## iter  50 value 575.941872
## iter  60 value 571.865247
## iter  70 value 563.409493
## iter  80 value 529.272724
## iter  90 value 504.708266
## iter 100 value 493.365278
## final  value 493.365278
## stopped after 100 iterations
```

```
## # weights:  169
## initial  value 2167.246773
## iter  10 value 758.520991
## iter  20 value 668.272617
## iter  30 value 624.513508
## iter  40 value 599.600141
## iter  50 value 588.375860
## iter  60 value 582.485575
## iter  70 value 581.428253
## iter  80 value 581.365316
## iter  90 value 581.299532
## iter 100 value 581.199975
## final  value 581.199975
## stopped after 100 iterations
## # weights:  505
## initial  value 1273.641851
## iter  10 value 709.071464
## iter  20 value 598.887061
## iter  30 value 548.393171
## iter  40 value 511.727948
## iter  50 value 493.231097
## iter  60 value 485.801049
## iter  70 value 480.795342
## iter  80 value 476.749545
## iter  90 value 474.787162
## iter 100 value 472.564956
## final  value 472.564956
## stopped after 100 iterations
## # weights:  841
## initial  value 1990.327057
## iter  10 value 936.881502
## iter  20 value 770.178827
## iter  30 value 658.409361
## iter  40 value 605.671884
## iter  50 value 555.305964
## iter  60 value 521.721353
## iter  70 value 503.367090
## iter  80 value 492.907570
## iter  90 value 475.093964
## iter 100 value 460.900277
## final  value 460.900277
## stopped after 100 iterations
## # weights:  169
## initial  value 2113.371631
## iter  10 value 751.584588
## iter  20 value 623.456132
## iter  30 value 568.583590
## iter  40 value 557.583820
## iter  50 value 552.399544
## iter  60 value 550.726378
## iter  70 value 549.493567
```

```
## iter  80 value 549.356153
## iter  90 value 549.264974
## iter 100 value 548.910210
## final  value 548.910210
## stopped after 100 iterations
## # weights:  505
## initial  value 1709.348275
## iter  10 value 748.640210
## iter  20 value 585.814690
## iter  30 value 514.437269
## iter  40 value 467.850364
## iter  50 value 443.923983
## iter  60 value 419.214421
## iter  70 value 403.343594
## iter  80 value 390.128411
## iter  90 value 384.624152
## iter 100 value 382.415122
## final  value 382.415122
## stopped after 100 iterations
## # weights:  841
## initial  value 2480.371347
## iter  10 value 803.387214
## iter  20 value 669.327919
## iter  30 value 645.080023
## iter  40 value 553.522759
## iter  50 value 472.781097
## iter  60 value 429.924490
## iter  70 value 407.679988
## iter  80 value 392.315248
## iter  90 value 384.922568
## iter 100 value 379.629512
## final  value 379.629512
## stopped after 100 iterations
## # weights:  169
## initial  value 2402.065197
## iter  10 value 836.153912
## iter  20 value 704.255973
## iter  30 value 627.099898
## iter  40 value 572.768192
## iter  50 value 560.084918
## iter  60 value 548.721878
## iter  70 value 531.279128
## iter  80 value 528.076371
## iter  90 value 527.444692
## final  value 527.444490
## converged
## # weights:  505
## initial  value 3144.136310
## iter  10 value 913.199751
## final  value 913.198923
## converged
```

```
## # weights:  841
## initial  value 2391.090037
## iter  10 value 824.813224
## iter  20 value 686.282946
## iter  30 value 588.483642
## iter  40 value 579.250065
## iter  50 value 562.904728
## iter  60 value 550.135988
## iter  70 value 510.475395
## iter  80 value 482.746699
## iter  90 value 477.813312
## iter 100 value 470.792961
## final  value 470.792961
## stopped after 100 iterations
## # weights:  169
## initial  value 1681.215158
## iter  10 value 737.000484
## iter  20 value 647.289162
## iter  30 value 616.724910
## iter  40 value 594.669633
## iter  50 value 583.685139
## iter  60 value 579.461323
## iter  70 value 578.915778
## iter  80 value 578.612112
## iter  90 value 577.744473
## iter 100 value 577.723205
## final  value 577.723205
## stopped after 100 iterations
## # weights:  505
## initial  value 3066.601791
## iter  10 value 849.080588
## iter  20 value 699.160686
## iter  30 value 628.915001
## iter  40 value 596.709296
## iter  50 value 567.267615
## iter  60 value 549.873698
## iter  70 value 526.660178
## iter  80 value 516.669569
## iter  90 value 510.732568
## iter 100 value 506.588736
## final  value 506.588736
## stopped after 100 iterations
## # weights:  841
## initial  value 2072.591932
## iter  10 value 825.603141
## iter  20 value 677.492923
## iter  30 value 608.409624
## iter  40 value 574.774578
## iter  50 value 540.441159
## iter  60 value 514.944029
## iter  70 value 484.367840
```

```
## iter  80 value 460.924650
## iter  90 value 442.718413
## iter 100 value 433.258774
## final  value 433.258774
## stopped after 100 iterations
## # weights:  169
## initial  value 2136.164529
## iter  10 value 772.904831
## iter  20 value 716.112129
## iter  30 value 652.116411
## iter  40 value 591.374478
## iter  50 value 557.572066
## iter  60 value 546.895163
## iter  70 value 542.262906
## iter  80 value 540.816692
## iter  90 value 538.506762
## iter 100 value 538.041916
## final  value 538.041916
## stopped after 100 iterations
## # weights:  505
## initial  value 1794.630978
## iter  10 value 915.059399
## iter  20 value 842.351015
## iter  30 value 631.276544
## iter  40 value 531.419039
## iter  50 value 474.277998
## iter  60 value 429.719057
## iter  70 value 399.842531
## iter  80 value 387.987863
## iter  90 value 374.479073
## iter 100 value 365.633466
## final  value 365.633466
## stopped after 100 iterations
## # weights:  841
## initial  value 3095.658512
## iter  10 value 702.123708
## iter  20 value 536.443943
## iter  30 value 464.972395
## iter  40 value 384.491990
## iter  50 value 323.943463
## iter  60 value 283.768921
## iter  70 value 261.817427
## iter  80 value 254.899894
## iter  90 value 247.829702
## iter 100 value 244.784585
## final  value 244.784585
## stopped after 100 iterations
## # weights:  169
## initial  value 2190.846566
## iter  10 value 726.744488
## iter  20 value 611.954886
```

```
## iter  30 value 563.909655
## iter  40 value 557.658679
## iter  50 value 554.684989
## iter  60 value 550.108470
## iter  70 value 548.986929
## iter  80 value 546.698434
## iter  90 value 545.916329
## iter 100 value 544.926360
## final  value 544.926360
## stopped after 100 iterations
## # weights:  505
## initial  value 2370.846754
## iter  10 value 913.184035
## iter  20 value 806.842162
## iter  30 value 709.701703
## iter  40 value 634.632713
## iter  50 value 587.608622
## iter  60 value 564.540848
## iter  70 value 560.282725
## iter  80 value 546.498798
## iter  90 value 536.489093
## iter 100 value 535.299833
## final  value 535.299833
## stopped after 100 iterations
## # weights:  841
## initial  value 1621.910278
## iter  10 value 726.460296
## iter  20 value 605.620647
## iter  30 value 493.397258
## iter  40 value 424.883157
## iter  50 value 397.244396
## iter  60 value 370.578283
## iter  70 value 340.484793
## iter  80 value 313.657164
## iter  90 value 296.748494
## iter 100 value 288.310251
## final  value 288.310251
## stopped after 100 iterations
## # weights:  169
## initial  value 1634.582887
## iter  10 value 754.429432
## iter  20 value 661.972583
## iter  30 value 625.602258
## iter  40 value 602.541066
## iter  50 value 587.301979
## iter  60 value 577.996425
## iter  70 value 575.057745
## iter  80 value 574.034971
## iter  90 value 573.629063
## iter 100 value 573.559075
## final  value 573.559075
```

```
## stopped after 100 iterations
## # weights:  505
## initial  value 1589.754990
## iter  10 value 765.449985
## iter  20 value 676.887374
## iter  30 value 655.252784
## iter  40 value 631.141360
## iter  50 value 571.353833
## iter  60 value 534.865639
## iter  70 value 518.093478
## iter  80 value 505.261563
## iter  90 value 494.364427
## iter 100 value 482.710878
## final  value 482.710878
## stopped after 100 iterations
## # weights:  841
## initial  value 1802.242017
## iter  10 value 725.846760
## iter  20 value 642.742676
## iter  30 value 602.322076
## iter  40 value 572.082946
## iter  50 value 555.120532
## iter  60 value 541.642948
## iter  70 value 532.606946
## iter  80 value 529.269696
## iter  90 value 521.440218
## iter 100 value 497.957782
## final  value 497.957782
## stopped after 100 iterations
## # weights:  169
## initial  value 1695.332127
## iter  10 value 741.962906
## iter  20 value 614.159766
## iter  30 value 575.541518
## iter  40 value 560.218458
## iter  50 value 555.893188
## iter  60 value 552.743800
## iter  70 value 547.571454
## iter  80 value 546.952564
## iter  90 value 546.612729
## iter 100 value 545.706055
## final  value 545.706055
## stopped after 100 iterations
## # weights:  505
## initial  value 2179.762581
## iter  10 value 913.471048
## iter  20 value 820.715297
## iter  30 value 654.392207
## iter  40 value 564.053068
## iter  50 value 546.588612
## iter  60 value 542.937385
```

```
## iter  70 value 540.716343
## iter  80 value 538.589452
## iter  90 value 538.269086
## iter 100 value 538.189465
## final  value 538.189465
## stopped after 100 iterations
## # weights:  841
## initial  value 3251.684210
## iter  10 value 732.824160
## iter  20 value 586.882604
## iter  30 value 465.551986
## iter  40 value 385.872909
## iter  50 value 338.465392
## iter  60 value 307.505947
## iter  70 value 286.267578
## iter  80 value 275.335354
## iter  90 value 268.598795
## iter 100 value 262.621054
## final  value 262.621054
## stopped after 100 iterations
## # weights:  169
## initial  value 1969.421876
## iter  10 value 738.441764
## iter  20 value 616.014861
## iter  30 value 567.139505
## iter  40 value 560.363682
## iter  50 value 559.029713
## iter  60 value 557.568638
## iter  70 value 551.781233
## iter  80 value 540.392782
## iter  90 value 536.717739
## iter 100 value 536.680984
## final  value 536.680984
## stopped after 100 iterations
## # weights:  505
## initial  value 2155.051653
## iter  10 value 796.766317
## iter  20 value 683.122936
## iter  30 value 620.029406
## iter  40 value 617.916591
## iter  50 value 614.965703
## final  value 614.943526
## converged
## # weights:  841
## initial  value 1155.554649
## iter  10 value 697.354270
## iter  20 value 539.218887
## iter  30 value 437.435213
## iter  40 value 383.743482
## iter  50 value 361.021935
## iter  60 value 345.696827
```

```
## iter  70 value 332.094644
## iter  80 value 306.710627
## iter  90 value 295.958092
## iter 100 value 294.737604
## final  value 294.737604
## stopped after 100 iterations
## # weights:  169
## initial  value 1919.786934
## iter  10 value 823.034115
## iter  20 value 764.506955
## iter  30 value 669.686515
## iter  40 value 628.605430
## iter  50 value 610.726914
## iter  60 value 601.168282
## iter  70 value 588.943737
## iter  80 value 586.586167
## iter  90 value 586.244539
## iter 100 value 586.198081
## final  value 586.198081
## stopped after 100 iterations
## # weights:  505
## initial  value 1428.360527
## iter  10 value 711.857231
## iter  20 value 640.481376
## iter  30 value 605.941588
## iter  40 value 581.109955
## iter  50 value 569.601739
## iter  60 value 554.224249
## iter  70 value 530.460025
## iter  80 value 512.897911
## iter  90 value 503.254435
## iter 100 value 492.528446
## final  value 492.528446
## stopped after 100 iterations
## # weights:  841
## initial  value 2332.048691
## iter  10 value 929.153224
## iter  20 value 723.473622
## iter  30 value 646.955495
## iter  40 value 621.085047
## iter  50 value 578.837828
## iter  60 value 536.359711
## iter  70 value 497.645329
## iter  80 value 473.956993
## iter  90 value 461.371552
## iter 100 value 450.769144
## final  value 450.769144
## stopped after 100 iterations
## # weights:  169
## initial  value 1840.153075
## iter  10 value 904.226236
```

```
## iter  20 value 709.329256
## iter  30 value 616.576586
## iter  40 value 591.107003
## iter  50 value 584.972161
## iter  60 value 582.497308
## iter  70 value 580.555304
## iter  80 value 580.161590
## iter  90 value 579.891242
## iter 100 value 579.545247
## final  value 579.545247
## stopped after 100 iterations
## # weights:  505
## initial  value 1913.920714
## iter  10 value 747.585344
## iter  20 value 615.147381
## iter  30 value 567.383242
## iter  40 value 561.775462
## iter  50 value 560.642559
## iter  60 value 559.579774
## iter  70 value 557.869898
## iter  80 value 557.388587
## iter  90 value 556.521273
## iter 100 value 555.878406
## final  value 555.878406
## stopped after 100 iterations
## # weights:  841
## initial  value 1619.709524
## iter  10 value 750.792377
## iter  20 value 590.361296
## iter  30 value 482.423453
## iter  40 value 423.653084
## iter  50 value 395.345466
## iter  60 value 372.837380
## iter  70 value 366.951600
## iter  80 value 362.126722
## iter  90 value 357.527790
## iter 100 value 354.421085
## final  value 354.421085
## stopped after 100 iterations
## # weights:  169
## initial  value 2186.079006
## iter  10 value 731.937584
## iter  20 value 615.469372
## iter  30 value 568.822273
## iter  40 value 558.923327
## iter  50 value 557.022253
## iter  60 value 550.494702
## iter  70 value 547.071598
## iter  80 value 546.054138
## iter  90 value 545.783934
## iter 100 value 545.466919
```

```
## final  value 545.466919
## stopped after 100 iterations
## # weights:  505
## initial  value 1685.339128
## iter  10 value 714.164385
## iter  20 value 572.317971
## iter  30 value 518.907932
## iter  40 value 482.059607
## iter  50 value 456.097830
## iter  60 value 442.963813
## iter  70 value 418.729838
## iter  80 value 393.888060
## iter  90 value 374.664276
## iter 100 value 369.754322
## final  value 369.754322
## stopped after 100 iterations
## # weights:  841
## initial  value 2231.257741
## iter  10 value 915.491374
## iter  20 value 915.462489
## iter  30 value 856.880458
## iter  40 value 699.226176
## iter  50 value 605.998342
## iter  60 value 584.738053
## iter  70 value 579.219358
## iter  80 value 576.270410
## iter  90 value 567.918678
## iter 100 value 554.454237
## final  value 554.454237
## stopped after 100 iterations
## # weights:  169
## initial  value 1625.475136
## iter  10 value 759.171618
## iter  20 value 675.048529
## iter  30 value 630.527763
## iter  40 value 604.617747
## iter  50 value 585.521503
## iter  60 value 578.066628
## iter  70 value 574.818100
## iter  80 value 574.685544
## iter  90 value 574.530295
## iter 100 value 574.330199
## final  value 574.330199
## stopped after 100 iterations
## # weights:  505
## initial  value 2860.783405
## iter  10 value 832.472052
## iter  20 value 701.991928
## iter  30 value 621.698496
## iter  40 value 579.425053
## iter  50 value 553.816597
```

```
## iter  60 value 527.924992
## iter  70 value 500.903409
## iter  80 value 478.957097
## iter  90 value 467.172753
## iter 100 value 460.169608
## final  value 460.169608
## stopped after 100 iterations
## # weights:  841
## initial  value 1729.726869
## iter  10 value 743.540910
## iter  20 value 626.060971
## iter  30 value 565.894788
## iter  40 value 544.535772
## iter  50 value 522.651626
## iter  60 value 492.629581
## iter  70 value 462.468736
## iter  80 value 442.451467
## iter  90 value 431.269938
## iter 100 value 424.834255
## final  value 424.834255
## stopped after 100 iterations
## # weights:  169
## initial  value 1968.534885
## iter  10 value 700.989929
## iter  20 value 597.125906
## iter  30 value 551.146585
## iter  40 value 542.054809
## iter  50 value 541.034477
## iter  60 value 539.565884
## iter  70 value 537.719271
## iter  80 value 537.698557
## iter  90 value 537.693745
## iter 100 value 537.687346
## final  value 537.687346
## stopped after 100 iterations
## # weights:  505
## initial  value 2838.439437
## iter  10 value 687.882052
## iter  20 value 565.102055
## iter  30 value 503.866229
## iter  40 value 463.704103
## iter  50 value 431.915812
## iter  60 value 409.949064
## iter  70 value 405.647083
## iter  80 value 403.586962
## iter  90 value 401.848435
## iter 100 value 399.139846
## final  value 399.139846
## stopped after 100 iterations
## # weights:  841
## initial  value 1557.074936
```

```
## iter  10 value 713.684360
## iter  20 value 557.945529
## iter  30 value 480.230167
## iter  40 value 436.823264
## iter  50 value 407.912798
## iter  60 value 388.484725
## iter  70 value 373.300736
## iter  80 value 366.441782
## iter  90 value 364.134605
## iter 100 value 362.941402
## final  value 362.941402
## stopped after 100 iterations
## # weights:  169
## initial  value 2757.502939
## iter  10 value 853.957045
## iter  20 value 682.441071
## iter  30 value 601.418029
## iter  40 value 581.459671
## iter  50 value 569.103117
## iter  60 value 554.271301
## iter  70 value 549.089887
## iter  80 value 543.344208
## iter  90 value 531.457567
## iter 100 value 530.223483
## final  value 530.223483
## stopped after 100 iterations
## # weights:  505
## initial  value 1654.369905
## iter  10 value 880.696795
## iter  20 value 728.752855
## iter  30 value 661.955520
## iter  40 value 627.167436
## iter  50 value 588.193039
## iter  60 value 574.576452
## iter  70 value 561.976279
## iter  80 value 555.108478
## iter  90 value 551.110894
## iter 100 value 535.386850
## final  value 535.386850
## stopped after 100 iterations
## # weights:  841
## initial  value 1396.511976
## iter  10 value 729.191682
## iter  20 value 542.630479
## iter  30 value 438.072308
## iter  40 value 395.079823
## iter  50 value 370.679699
## iter  60 value 332.794913
## iter  70 value 315.583122
## iter  80 value 313.056880
## iter  90 value 312.488304
```

```
## iter 100 value 312.238336
## final  value 312.238336
## stopped after 100 iterations
## # weights:  169
## initial  value 2581.158768
## iter  10 value 757.743751
## iter  20 value 659.539661
## iter  30 value 616.242833
## iter  40 value 588.423080
## iter  50 value 575.545952
## iter  60 value 567.602923
## iter  70 value 565.621642
## iter  80 value 565.466214
## iter  90 value 565.367120
## iter 100 value 565.364341
## final  value 565.364341
## stopped after 100 iterations
## # weights:  505
## initial  value 1718.470071
## iter  10 value 758.442687
## iter  20 value 636.239798
## iter  30 value 593.243142
## iter  40 value 557.734000
## iter  50 value 532.046755
## iter  60 value 503.416301
## iter  70 value 488.617626
## iter  80 value 479.893397
## iter  90 value 473.318237
## iter 100 value 469.662463
## final  value 469.662463
## stopped after 100 iterations
## # weights:  841
## initial  value 2368.431217
## iter  10 value 903.851600
## iter  20 value 731.909460
## iter  30 value 627.007921
## iter  40 value 556.656741
## iter  50 value 519.072928
## iter  60 value 498.624699
## iter  70 value 483.535620
## iter  80 value 467.654136
## iter  90 value 457.226448
## iter 100 value 448.442873
## final  value 448.442873
## stopped after 100 iterations
## # weights:  169
## initial  value 1382.441543
## iter  10 value 779.347103
## iter  20 value 731.891860
## iter  30 value 705.388989
## iter  40 value 699.437964
```

```
## iter  50 value 687.364878
## iter  60 value 676.754393
## iter  70 value 662.902681
## iter  80 value 641.444835
## iter  90 value 624.646155
## iter 100 value 621.190853
## final  value 621.190853
## stopped after 100 iterations
## # weights:  505
## initial  value 2589.588453
## iter  10 value 718.717557
## iter  20 value 577.558105
## iter  30 value 530.105228
## iter  40 value 509.221006
## iter  50 value 464.290810
## iter  60 value 431.205273
## iter  70 value 412.182018
## iter  80 value 400.990057
## iter  90 value 394.703154
## iter 100 value 388.315347
## final  value 388.315347
## stopped after 100 iterations
## # weights:  841
## initial  value 1977.544885
## iter  10 value 805.301250
## iter  20 value 619.025277
## iter  30 value 502.319852
## iter  40 value 460.490900
## iter  50 value 438.555249
## iter  60 value 423.532559
## iter  70 value 412.777603
## iter  80 value 408.144218
## iter  90 value 391.416917
## iter 100 value 375.122103
## final  value 375.122103
## stopped after 100 iterations
## # weights:  169
## initial  value 1927.806021
## iter  10 value 741.737448
## iter  20 value 612.011493
## iter  30 value 578.798021
## iter  40 value 572.626386
## iter  50 value 565.884730
## iter  60 value 547.324412
## iter  70 value 538.984614
## iter  80 value 538.657849
## iter  90 value 538.646795
## final  value 538.646782
## converged
## # weights:  505
## initial  value 1810.375943
```

```
## iter  10 value 820.957207
## iter  20 value 669.356346
## iter  30 value 591.436215
## iter  40 value 564.847712
## iter  50 value 559.185560
## iter  60 value 549.668866
## iter  70 value 530.722862
## iter  80 value 529.540005
## iter  90 value 529.534524
## iter 100 value 528.877657
## final  value 528.877657
## stopped after 100 iterations
## # weights:  841
## initial  value 1663.674115
## iter  10 value 740.050813
## iter  20 value 604.969870
## iter  30 value 563.291433
## iter  40 value 553.563061
## iter  50 value 552.573568
## iter  60 value 551.145172
## iter  70 value 548.563837
## iter  80 value 548.180791
## iter  90 value 548.153953
## iter 100 value 548.124043
## final  value 548.124043
## stopped after 100 iterations
## # weights:  169
## initial  value 2361.334031
## iter  10 value 733.127759
## iter  20 value 653.468618
## iter  30 value 626.535639
## iter  40 value 606.863749
## iter  50 value 587.959273
## iter  60 value 582.461584
## iter  70 value 581.443348
## iter  80 value 581.376410
## iter  90 value 581.374186
## iter 100 value 581.369379
## final  value 581.369379
## stopped after 100 iterations
## # weights:  505
## initial  value 2303.144083
## iter  10 value 820.480068
## iter  20 value 688.194133
## iter  30 value 626.914890
## iter  40 value 589.333046
## iter  50 value 544.993707
## iter  60 value 519.331116
## iter  70 value 508.320335
## iter  80 value 500.709706
## iter  90 value 495.874001
```

```
## iter 100 value 493.424477
## final  value 493.424477
## stopped after 100 iterations
## # weights:  841
## initial  value 1611.497003
## iter  10 value 912.366834
## iter  20 value 744.211579
## iter  30 value 644.435955
## iter  40 value 582.609591
## iter  50 value 528.125663
## iter  60 value 488.398386
## iter  70 value 455.796136
## iter  80 value 438.447932
## iter  90 value 425.188238
## iter 100 value 415.574002
## final  value 415.574002
## stopped after 100 iterations
## # weights:  169
## initial  value 2013.331760
## iter  10 value 696.696436
## iter  20 value 585.486296
## iter  30 value 555.403390
## iter  40 value 552.290299
## iter  50 value 548.366224
## iter  60 value 545.874624
## iter  70 value 545.092871
## iter  80 value 544.837506
## iter  90 value 544.224813
## iter 100 value 543.593862
## final  value 543.593862
## stopped after 100 iterations
## # weights:  505
## initial  value 1266.727199
## iter  10 value 758.168533
## iter  20 value 580.883103
## iter  30 value 501.462747
## iter  40 value 457.247378
## iter  50 value 422.756153
## iter  60 value 404.099496
## iter  70 value 388.214204
## iter  80 value 386.963961
## iter  90 value 385.627945
## iter 100 value 382.636112
## final  value 382.636112
## stopped after 100 iterations
## # weights:  841
## initial  value 3498.488447
## iter  10 value 688.611173
## iter  20 value 506.709328
## iter  30 value 406.554602
## iter  40 value 350.283303
```

```
## iter  50 value 321.538867
## iter  60 value 300.846018
## iter  70 value 285.416853
## iter  80 value 278.260828
## iter  90 value 275.308433
## iter 100 value 273.588525
## final  value 273.588525
## stopped after 100 iterations
## # weights:  169
## initial  value 1381.260034
## iter  10 value 681.019508
## iter  20 value 569.139623
## iter  30 value 552.272219
## iter  40 value 548.011013
## iter  50 value 545.877066
## iter  60 value 543.558364
## iter  70 value 542.583774
## iter  80 value 542.007593
## iter  90 value 541.434346
## iter 100 value 541.047618
## final  value 541.047618
## stopped after 100 iterations
## # weights:  505
## initial  value 1915.140639
## iter  10 value 682.785915
## iter  20 value 568.048754
## iter  30 value 546.048075
## iter  40 value 513.627085
## iter  50 value 473.090325
## iter  60 value 451.900211
## iter  70 value 435.955836
## iter  80 value 423.642886
## iter  90 value 419.029455
## iter 100 value 417.018420
## final  value 417.018420
## stopped after 100 iterations
## # weights:  841
## initial  value 1755.448888
## iter  10 value 755.670897
## iter  20 value 556.601389
## iter  30 value 470.944084
## iter  40 value 435.478981
## iter  50 value 377.089773
## iter  60 value 319.567724
## iter  70 value 288.043704
## iter  80 value 274.071095
## iter  90 value 255.239524
## iter 100 value 246.703330
## final  value 246.703330
## stopped after 100 iterations
## # weights:  169
```

```
## initial  value 1247.183705
## iter  10 value 766.366380
## iter  20 value 661.370995
## iter  30 value 622.038362
## iter  40 value 596.592081
## iter  50 value 580.402492
## iter  60 value 573.346066
## iter  70 value 572.055050
## iter  80 value 570.512450
## iter  90 value 570.210572
## iter 100 value 570.207682
## final  value 570.207682
## stopped after 100 iterations
## # weights:  505
## initial  value 3635.692822
## iter  10 value 939.996702
## iter  20 value 792.738750
## iter  30 value 663.593630
## iter  40 value 597.515568
## iter  50 value 536.416602
## iter  60 value 505.511994
## iter  70 value 490.305771
## iter  80 value 481.329427
## iter  90 value 472.825230
## iter 100 value 467.304392
## final  value 467.304392
## stopped after 100 iterations
## # weights:  841
## initial  value 2106.456567
## iter  10 value 797.793786
## iter  20 value 660.605012
## iter  30 value 575.042147
## iter  40 value 520.228429
## iter  50 value 483.828139
## iter  60 value 461.794240
## iter  70 value 443.724111
## iter  80 value 429.425664
## iter  90 value 420.463415
## iter 100 value 416.413287
## final  value 416.413287
## stopped after 100 iterations
## # weights:  169
## initial  value 1684.167127
## iter  10 value 755.293316
## iter  20 value 656.448273
## iter  30 value 577.241250
## iter  40 value 553.079972
## iter  50 value 547.613326
## iter  60 value 543.393505
## iter  70 value 542.274633
## iter  80 value 540.677226
```

```
## iter  90 value 540.547537
## iter 100 value 540.521851
## final  value 540.521851
## stopped after 100 iterations
## # weights:  505
## initial  value 1347.709377
## iter  10 value 914.777199
## iter  20 value 765.486702
## iter  30 value 627.160656
## iter  40 value 576.137152
## iter  50 value 563.700159
## iter  60 value 561.905324
## iter  70 value 559.427530
## iter  80 value 554.688990
## iter  90 value 550.163548
## iter 100 value 548.099719
## final  value 548.099719
## stopped after 100 iterations
## # weights:  841
## initial  value 1000.065688
## iter  10 value 753.741829
## iter  20 value 637.021134
## iter  30 value 533.441197
## iter  40 value 472.766093
## iter  50 value 428.111349
## iter  60 value 395.925373
## iter  70 value 380.244590
## iter  80 value 369.248043
## iter  90 value 351.319487
## iter 100 value 346.758815
## final  value 346.758815
## stopped after 100 iterations
## # weights:  169
## initial  value 2182.906151
## iter  10 value 747.202793
## iter  20 value 653.540971
## iter  30 value 601.391670
## iter  40 value 572.237353
## iter  50 value 563.556320
## iter  60 value 558.337113
## iter  70 value 546.527898
## iter  80 value 538.067751
## iter  90 value 537.652178
## iter 100 value 537.643307
## final  value 537.643307
## stopped after 100 iterations
## # weights:  505
## initial  value 1330.403191
## iter  10 value 710.641919
## iter  20 value 587.340891
## iter  30 value 535.025715
```

```
## iter  40 value 500.838770
## iter  50 value 472.505246
## iter  60 value 455.330431
## iter  70 value 439.905422
## iter  80 value 430.425155
## iter  90 value 415.728544
## iter 100 value 400.706328
## final  value 400.706328
## stopped after 100 iterations
## # weights:  841
## initial  value 3326.861416
## iter  10 value 908.944827
## iter  20 value 779.478813
## iter  30 value 642.720785
## iter  40 value 580.647810
## iter  50 value 570.520606
## iter  60 value 562.614509
## iter  70 value 552.193161
## iter  80 value 550.595971
## iter  90 value 550.043294
## iter 100 value 549.681895
## final  value 549.681895
## stopped after 100 iterations
## # weights:  169
## initial  value 1349.717596
## iter  10 value 763.848449
## iter  20 value 665.612532
## iter  30 value 617.007304
## iter  40 value 598.437608
## iter  50 value 586.270649
## iter  60 value 579.381265
## iter  70 value 577.481524
## iter  80 value 577.294747
## iter  90 value 576.618656
## iter 100 value 576.574545
## final  value 576.574545
## stopped after 100 iterations
## # weights:  505
## initial  value 1713.230067
## iter  10 value 802.579691
## iter  20 value 660.615333
## iter  30 value 572.925590
## iter  40 value 526.177382
## iter  50 value 498.807646
## iter  60 value 488.513083
## iter  70 value 482.616320
## iter  80 value 477.005175
## iter  90 value 473.898246
## iter 100 value 472.912136
## final  value 472.912136
## stopped after 100 iterations
```

```
## # weights:  841
## initial  value 1990.980696
## iter  10 value 771.992498
## iter  20 value 658.234801
## iter  30 value 595.674071
## iter  40 value 562.051690
## iter  50 value 538.372013
## iter  60 value 514.425161
## iter  70 value 481.280762
## iter  80 value 454.893932
## iter  90 value 437.962877
## iter 100 value 430.377891
## final  value 430.377891
## stopped after 100 iterations
## # weights:  169
## initial  value 1226.223842
## iter  10 value 782.338696
## iter  20 value 669.252038
## iter  30 value 614.686965
## iter  40 value 574.721852
## iter  50 value 556.397031
## iter  60 value 547.911184
## iter  70 value 543.685498
## iter  80 value 543.159711
## iter  90 value 542.755350
## iter 100 value 542.572242
## final  value 542.572242
## stopped after 100 iterations
## # weights:  505
## initial  value 2112.116266
## iter  10 value 712.314066
## iter  20 value 593.195948
## iter  30 value 517.678469
## iter  40 value 481.718120
## iter  50 value 463.419520
## iter  60 value 449.100757
## iter  70 value 443.414962
## iter  80 value 442.420976
## iter  90 value 440.704875
## iter 100 value 439.827931
## final  value 439.827931
## stopped after 100 iterations
## # weights:  841
## initial  value 2640.502689
## iter  10 value 913.866295
## iter  20 value 913.510674
## final  value 913.433852
## converged
## # weights:  841
## initial  value 1688.117353
## iter  10 value 822.981764
```

```
## iter  20 value 689.926612
## iter  30 value 626.096756
## iter  40 value 578.956854
## iter  50 value 544.550780
## iter  60 value 524.830737
## iter  70 value 513.198319
## iter  80 value 506.243016
## iter  90 value 502.050121
## iter 100 value 498.547301
## final  value 498.547301
## stopped after 100 iterations
```

Let us see what the model parameters are.

```
NeuModel
```

```
## Neural Network
##
## 2869 samples
##  166 predictor
##    2 classes: 'f', 't'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2582, 2582, 2581, 2581, 2583, 2583, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##   1     0e+00  0.8755678  0.2345375
##   1     1e-04  0.8780408  0.3387030
##   1     1e-01  0.8902056  0.2887397

##   3     0e+00  0.8762646  0.1500213
##   3     1e-04  0.8839229  0.2979025
##   3     1e-01  0.8905468  0.3762540
##   5     0e+00  0.8800803  0.3808216
##   5     1e-04  0.8773099  0.3158091
##   5     1e-01  0.8936937  0.4117839
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 5 and decay = 0.1.
```

Now, let us test our model on the testing dataset.

```
Pred_neural<-predict(NeuModel,Superhost_test)
```

Let us now see how well the model performed.

```
confusionMatrix(Pred_neural,Superhost_test$Dec_var)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   f   t
##          f 602  47
##          t  33  34
##
##                Accuracy : 0.8883
##                  95% CI : (0.8629, 0.9104)
##     No Information Rate : 0.8869
##     P-Value [Acc > NIR] : 0.4825
##
##                   Kappa : 0.3978
##  Mcnemar's Test P-Value : 0.1461
##
##             Sensitivity : 0.9480
##             Specificity : 0.4198
##          Pos Pred Value : 0.9276
##          Neg Pred Value : 0.5075
##              Prevalence : 0.8869
##          Detection Rate : 0.8408
##    Detection Prevalence : 0.9064
##       Balanced Accuracy : 0.6839
##
##        'Positive' Class : f
##
```

Looks like the model performed very well indeed with a 89.94% accuracy.

Let us see how does a Random Forest perform for this data.

```
Randfor_host<-randomForest(Dec_var~.,data = Superhost_train,importance=TRUE)
```

Let us predict the values of our testing dataset.

```
RandomForest_pred_host<-predict(Randfor_host,Superhost_test)
```

Now, let us calculate the accuracy of the Random Forest model.

```
confusionMatrix(RandomForest_pred_host,Superhost_test$Dec_var)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   f   t
##          f 629  54
##          t   6  27
##                                           
##                Accuracy : 0.9162          
##                  95% CI : (0.8934, 0.9354)
##     No Information Rate : 0.8869          
##     P-Value [Acc > NIR] : 0.006157        
##                                           
##                   Kappa : 0.4368          
##  Mcnemar's Test P-Value : 1.298e-09       
##                                           
##             Sensitivity : 0.9906          
##             Specificity : 0.3333          
##          Pos Pred Value : 0.9209          
##          Neg Pred Value : 0.8182          
##              Prevalence : 0.8869          
##          Detection Rate : 0.8785          
##    Detection Prevalence : 0.9539          
##       Balanced Accuracy : 0.6619          
##                                           
##        'Positive' Class : f               
## 
```

MODEL ENSEMBLE:

Models are often ensembled, that is, combined together to make a more robust model out of it which has a better accuracy.[3]

Let us start off by making a dataframe of all the predictions of our randomforest and neural net models. I will also include the decision variable in this dataframe.

```
mod_list<-data.frame(Pred_neural,RandomForest_pred_host,Host=Superhost_test$Dec_var,s
```

Now, let us train the model on the newly created dataframe. I will use the decision variable that I have already crated in that dataframe.

```r
model_stack<-train(Host~.,data = mod_list,method="knn")

stack_pred<-predict(model_stack,Superhost_test[,1:166])

confusionMatrix(stack_pred,Superhost_test[,167])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   f   t
##          f 629  54
##          t   6  27
##
##                Accuracy : 0.9162
##                  95% CI : (0.8934, 0.9354)
##     No Information Rate : 0.8869
##     P-Value [Acc > NIR] : 0.006157
##
##                   Kappa : 0.4368
##  Mcnemar's Test P-Value : 1.298e-09
##
##             Sensitivity : 0.9906
##             Specificity : 0.3333
##          Pos Pred Value : 0.9209
##          Neg Pred Value : 0.8182
##              Prevalence : 0.8869
##          Detection Rate : 0.8785
##    Detection Prevalence : 0.9539
##       Balanced Accuracy : 0.6619
##
##        'Positive' Class : f
##
```

EVALUATING MODEL PERFORMANCE

We saw that the model performed quite well with a 91.89% accuracy. It is evident that Random Forest is better suited to our problem of classification since it has higher accuracy. The accuracy improved slightly when we used the model ensemble to solve the classification problem.

On the whole, it turns out that this classifier is more accurate in screening off the hosts which arent superhosts. The correct classification of hosts as being superhosts is less accurate.

PART B: SENTIMENT ANALYSIS

We saw in the Exploratory Data Analysis the most expensive neighbourhoods in Boston. But what did people who stayed there have to say about the neighbourhood? I want to explore the general sentiment of the neighbourhood and compare it with the average price paid for the listing in that neighbourhood.

I will use the Reviews dataset for this purpose. Then, after tokenizing the reviews per listing, I will join the price, neighbourhood and few other important columns from the Listings dataset. And then, I will proceed to analyze the sentiments and plot them for the neighbourhood. [4][5]

Let us load the data in the Tidytext format.

```
suppressWarnings(library(tidyr))
suppressWarnings(library(tidytext))
```

We need our comments to be of character type. So I will first convert it into character and then unnest tokens by words.

```
Reviews$comments<-as.character(Reviews$comments)

Reviews_words<-Reviews%>% select(listing_id,comments)%>%unnest_tokens(word,comments)
```

Now, let us remove all the stop words from our dataframe.

```
Reviews_words<-Reviews_words%>% anti_join(stop_words,by="word")
```

Positive and Negative sentiment per review:

I now wish to see the overall sentiment for each neighbourhood. I will use the "Bing" sentiments for assigning a total positive and negative score to each listing, Basically, each word is matched with the Bing sentiments as falling in either positive or negative sentiment and then the number of positive and negative sentiments are counted. Ultimately, my mutating a new column called Sentiment which is the difference between the positive and negative word score for each listing, we get the overall sentiment. Lastly, I have grouped the listings by area.

```
Sentiment_reviews<-Reviews_words%>% inner_join(get_sentiments("bing"),by="word")%>% c

Sentiment_reviews<-as.tibble(Sentiment_reviews)
```

```
# Making sure I remove the NAs.
Sentiment_reviews$negative<-ifelse(Sentiment_reviews$negative %in% NA,0,Sentiment_rev
Sentiment_reviews$positive<-ifelse(Sentiment_reviews$positive %in% NA,0,Sentiment_rev

Sentiment_reviews$sentiment<-Sentiment_reviews$positive-Sentiment_reviews$negative

Sentiment_reviews_top10<-Sentiment_reviews%>% arrange(desc(Sentiment_reviews$sentimen
```

```
## Selecting by sentiment
```

```
str(Sentiment_reviews)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    2749 obs. of  4 variables:
##  $ listing_id: int  3353 5506 6695 6976 8792 9273 9765 9824 9855 9857 ...
##  $ negative  : num  32 7 34 10 4 10 7 25 0 11 ...
##  $ positive  : num  150 167 233 232 106 67 33 81 15 69 ...
##  $ sentiment : num  118 160 199 222 102 57 26 56 15 58 ...
```

```
colnames(Sentiment_reviews)<-c("id","negative","positive","sentiment") # amking sure
```

Let us determine how the sentiment is related to the average rating of the listing. I will join the Listings dataframe to the Sentiment_reviews dataframe so that we get information about all the listings we are analyzing sentiments for.

```
Sentiment_analysis<-Sentiment_reviews%>% left_join(Listings,by="id")

head(Sentiment_analysis)
```

```
## # A tibble: 6 x 98
##      id negative positive sentiment listing_url     scrape_id last_scraped
##   <int>    <dbl>    <dbl>     <dbl> <fct>               <dbl> <fct>
## 1  3353    32.0      150       118  https://www.ai~   2.02e13 2016-09-07
## 2  5506     7.00     167       160  https://www.ai~   2.02e13 2016-09-07
## 3  6695    34.0      233       199  https://www.ai~   2.02e13 2016-09-07
## 4  6976    10.0      232       222  https://www.ai~   2.02e13 2016-09-07
## 5  8792     4.00     106       102  https://www.ai~   2.02e13 2016-09-07
## 6  9273    10.0       67.0      57.0 https://www.ai~   2.02e13 2016-09-07
## # ... with 91 more variables: name <fct>, summary <fct>, space <fct>,
```

```
## #   description <fct>, experiences_offered <fct>, neighborhood_overview
## #   <fct>, notes <fct>, transit <fct>, access <fct>, interaction <fct>,
## #   house_rules <fct>, thumbnail_url <fct>, medium_url <fct>, picture_url
## #   <fct>, xl_picture_url <fct>, host_id <int>, host_url <fct>, host_name
## #   <fct>, host_since <fct>, host_location <fct>, host_about <fct>,
## #   host_response_time <fct>, host_response_rate <fct>,
## #   host_acceptance_rate <fct>, host_is_superhost <fct>,
## #   host_thumbnail_url <fct>, host_picture_url <fct>, host_neighbourhood
## #   <fct>, host_listings_count <int>, host_total_listings_count <int>,
## #   host_verifications <fct>, host_has_profile_pic <fct>,
## #   host_identity_verified <fct>, street <fct>, neighbourhood <fct>,
## #   neighbourhood_cleansed <fct>, neighbourhood_group_cleansed <lgl>, city
## #   <fct>, state <fct>, zipcode <fct>, market <fct>, smart_location <fct>,
## #   country_code <fct>, country <fct>, latitude <dbl>, longitude <dbl>,
## #   is_location_exact <fct>, property_type <fct>, room_type <fct>,
## #   accommodates <int>, bathrooms <dbl>, bedrooms <int>, beds <int>,
## #   bed_type <fct>, amenities <fct>, square_feet <int>, price <fct>,
## #   weekly_price <fct>, monthly_price <fct>, security_deposit <fct>,
## #   cleaning_fee <fct>, guests_included <int>, extra_people <fct>,
## #   minimum_nights <int>, maximum_nights <int>, calendar_updated <fct>,
## #   has_availability <lgl>, availability_30 <int>, availability_60 <int>,
## #   availability_90 <int>, availability_365 <int>, calendar_last_scraped
## #   <fct>, number_of_reviews <int>, first_review <fct>, last_review <fct>,
## #   review_scores_rating <int>, review_scores_accuracy <int>,
## #   review_scores_cleanliness <int>, review_scores_checkin <int>,
## #   review_scores_communication <int>, review_scores_location <int>,
## #   review_scores_value <int>, requires_license <fct>, license <lgl>,
## #   jurisdiction_names <lgl>, instant_bookable <fct>, cancellation_policy
## #   <fct>, require_guest_profile_picture <fct>,
## #   require_guest_phone_verification <fct>, calculated_host_listings_count
## #   <int>, reviews_per_month <dbl>
```

Now, let us select the neighbourhood and the price along with the sentiments.

```
Sa<-Sentiment_analysis%>% group_by(neighbourhood_cleansed)%>% summarise(Sentiment=mea

head(Sa)
```

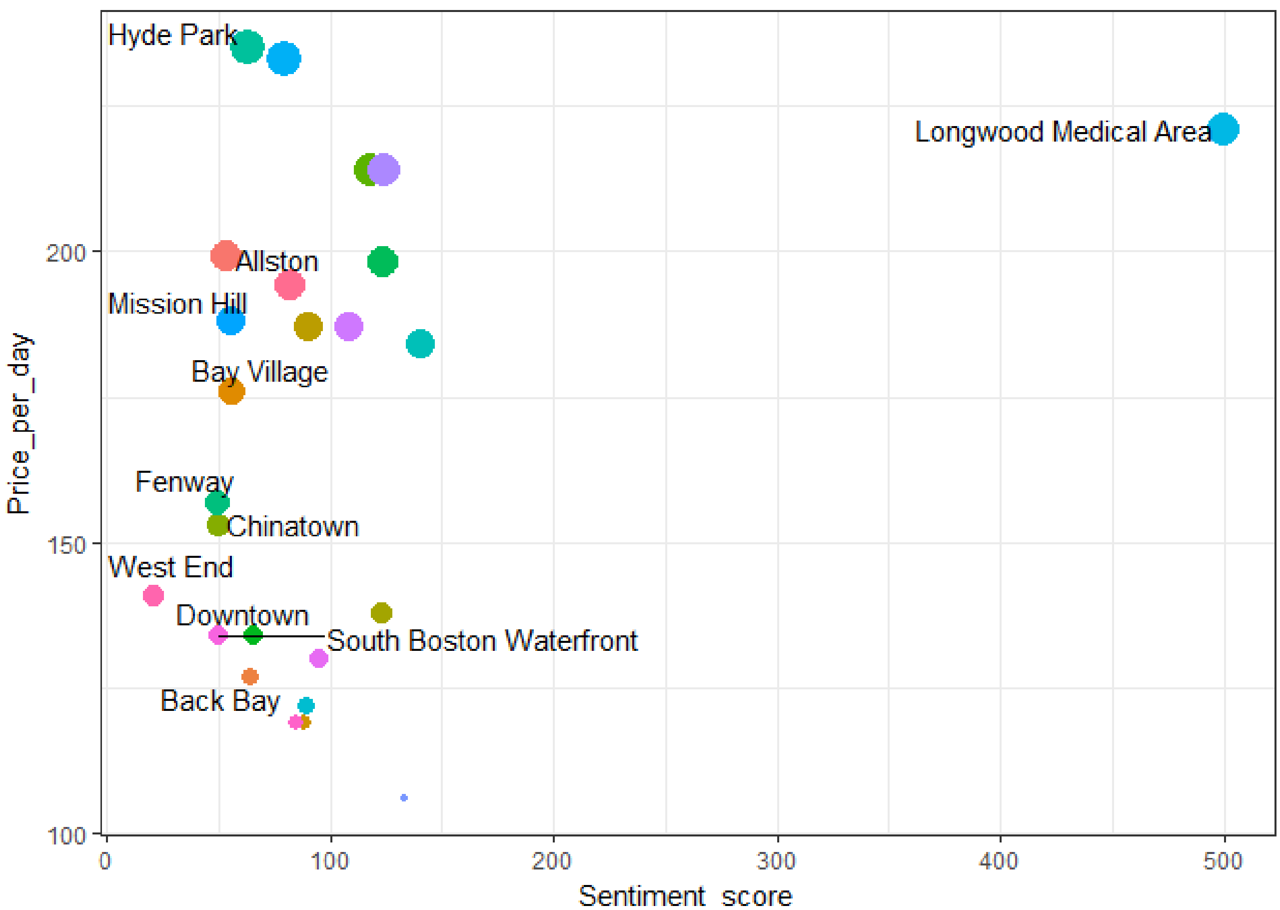
```
## # A tibble: 6 x 3
##   neighbourhood_cleansed Sentiment Price
##   <fct>                      <dbl> <dbl>
## 1 Allston                     54.7   200
## 2 Back Bay                    64.9   128
## 3 Bay Village                 56.5   176
## 4 Beacon Hill                 88.9   119
## 5 Brighton                    91.0   187
## 6 Charlestown                123     139
```

Now, let us plot the sentiment vs price.

```
suppressWarnings(library(ggrepel))

ggplot(data = Sa,mapping = aes(x=as.integer(Sa$Sentiment),y=as.integer(Sa$Price))+ge
```

```
## Warning: Ignoring unknown aesthetics: hjust
```



Thus, we can see that even if Hyde park neighbourhood is one of the most expensive, it's sentiment score isnt all that high. Longwood medical area, on the other hand has a high sentiment score as well as a high daily price.

Thus, I conclude the analysis of the Boston Airbnb.

MODEL DEPLOYMENT:

I will be publishing a RPubs document of this project as a part of the deployment stage.

Thus, we saw that the project followed all the stage of CRISP-DM and analyzed the Boston Airbnb.

References: [1]https://www.kaggle.com/airbnb/boston)

[2]https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/

[3]https://www.analyticsvidhya.com/blog/2017/02/introduction-to-ensembling-along-with-implementation-in-r/

[4]https://piazza-resources.s3.amazonaws.com/jc12zfeh6dh657/jezzh7djdamxs/13TextModels.pdf?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=ASIAJYEHEYWU6YRFEBEQ%2F20180420%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20180420T213215Z&X-Amz-Expires=10800&X-Amz-SignedHeaders=host&X-Amz-Security-Token=FQoDYXdzEPX%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaDMLqsaofHQuVUS47WCK3A0jfik1ksFk6%2B1X7T2cHYN9p%2B5pS2dOdDO0XJvKFrNPLJjhqOrCBI%2FIFwFr%2FPFFJT87PWtH0qgyc9zueG%2FDH20SAIrptO%2B0i34YJOC2RDzU0M%2F7i%2BVE1F6m8J25f9dG7d61CbE4f6gR6GvAM9qLs5QSzHGqZ7AgNhymoqLIOXz1DuVN150uR0edxJFHH3tQjvXZ%2BLLGrSx2w%2B2S6wfxmHLgqsq0FQO8naw0R4w7Fl4HiHvqXXkH%2B9j4bUE5bn1XfZMHOUAT3tJkxZ1s29nnMUOO8fleBKUwNcKNR2kvmYVHZ8Q8QNWcpQW6Ybp6CZdU2QBzfxnuVBBws2U%2BuH5mDYDb3kM1zFiW2%2BwUI4%2BKmelCWvCENRxbNz34un%2FoBtOFS3UncE8PEM9bcPc%2B3iq1QZoF6kct5ObiEYgn6jkoeB%2FwuH2hXZlwzg00gICF6%2BbzFO0v0I6lXpW6Ujz%2FwKuz2axkiW%2Bp8yuImNkByGLRxSOOGmj05YxW1WFl8uEObbFoxStXY8MGgnOpcowTRWktP%2BI1AjSy6NCnaGf8ABQ%2FL0fFFaSPyGAC7t8Ll2pyPzU13zZLEmGlePM4ospPp1gU%3D&X-Amz-Signature=f7828472afd7fde5eb832671f4a8fd924a97c78e16b354a239d08a34ca03f522

[5]http://varianceexplained.org/r/yelp-sentiment/

[6]Machine Learning with R (Second Edition) by Brett Lantz.