

CST4070-Applied Data Analytics

Block 2- Challenge A Submission

Name: Madeeha Abdul Rahim Solanki

Misis no: m00984984

Programme: MSc.Data Science

Date of Submission: 07/01/2024

Submitted to: Dr.Kishnadas Nanath

**Instructions :**

Participants are tasked with completing the following steps:

1. Data Visualisation: Generate visualisations to familiarise yourself with the data, guide the refinement of hypotheses, and aid in the development of a robust feature engineering strategy

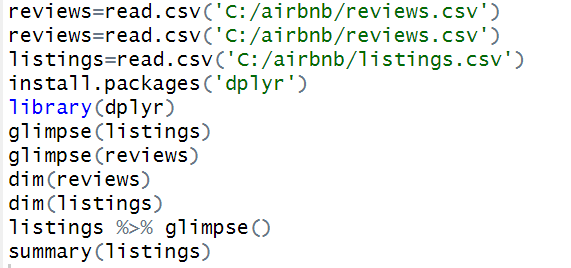
2. Feature Engineering: Construct a feature engineering model aimed at estimating your designed categorical variable (calculated in Task-1).

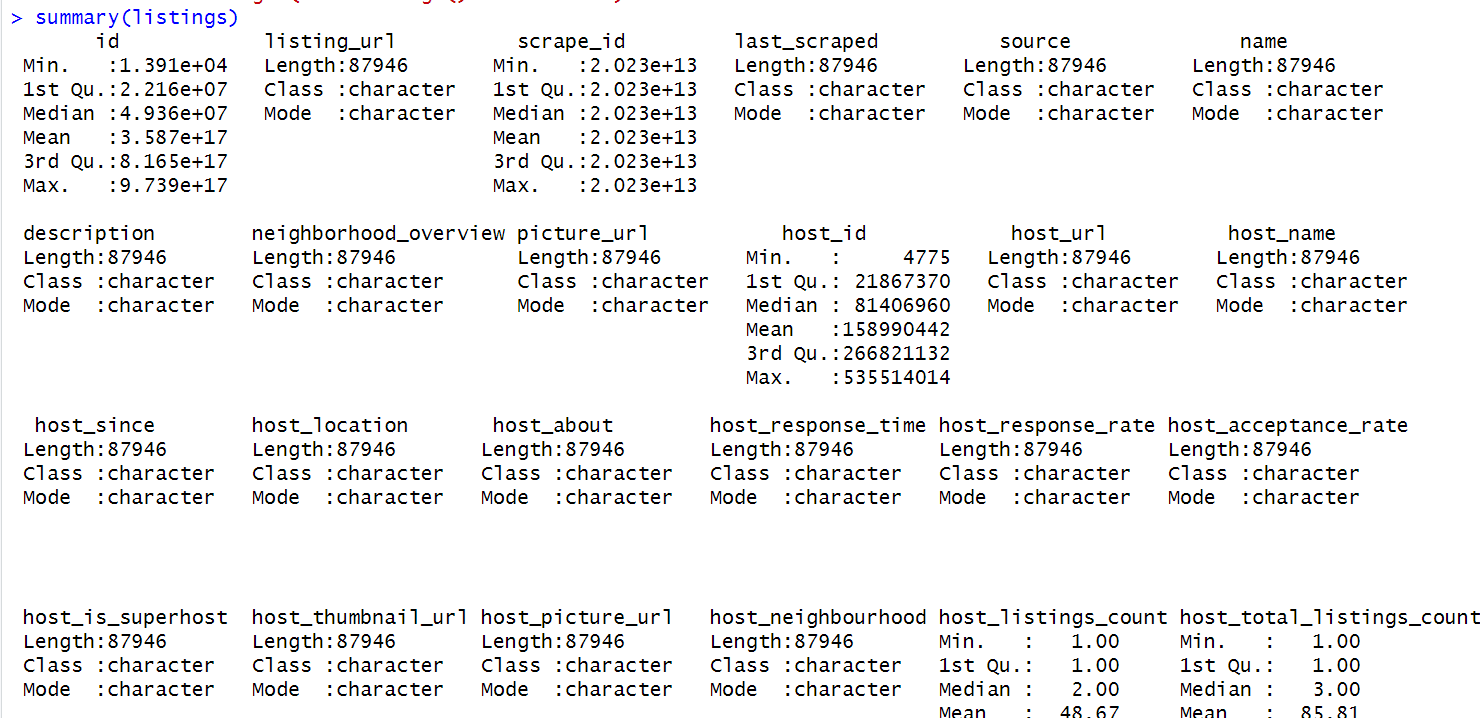
3. Model Building: Develop a predictive model to your categorical variable (designed in Task-1) using the provided features. You have the flexibility to choose any algorithm.

4. Model Evaluation: Assess the performance of your model using appropriate metrics. Ensure the model's generalisability through the use of any suitable data split method, such as train-test or cross-validation.

5. Feature Importance: Examine the importance of features in predicting your target variable.

6. Discussion: Summarise the main findings and emphasise the insights gained during this analysis. Discuss the most crucial results and their significance. Additionally, address any limitations of the proposed method, providing insights into its constraints and potential areas for improvement.

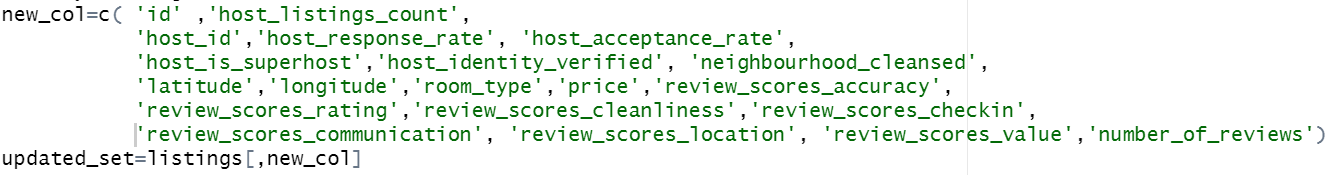


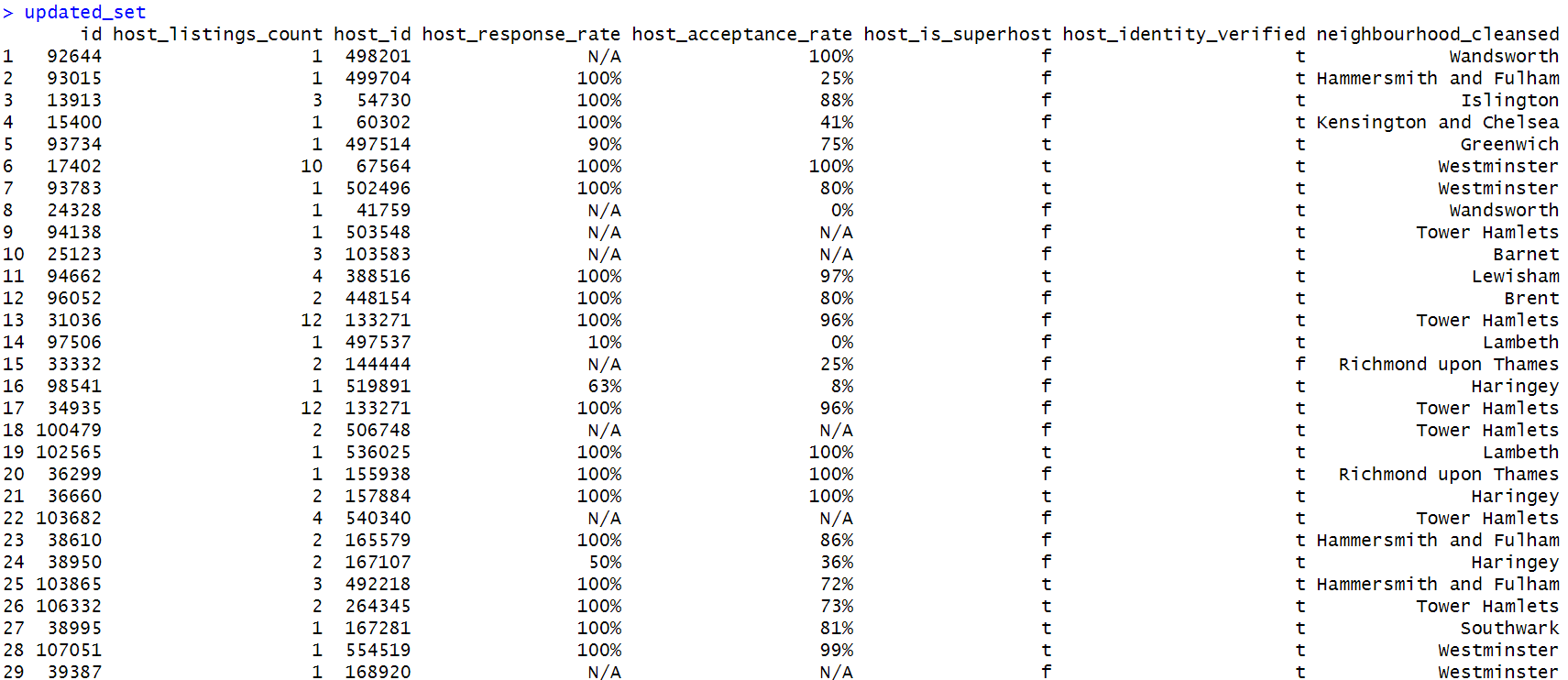


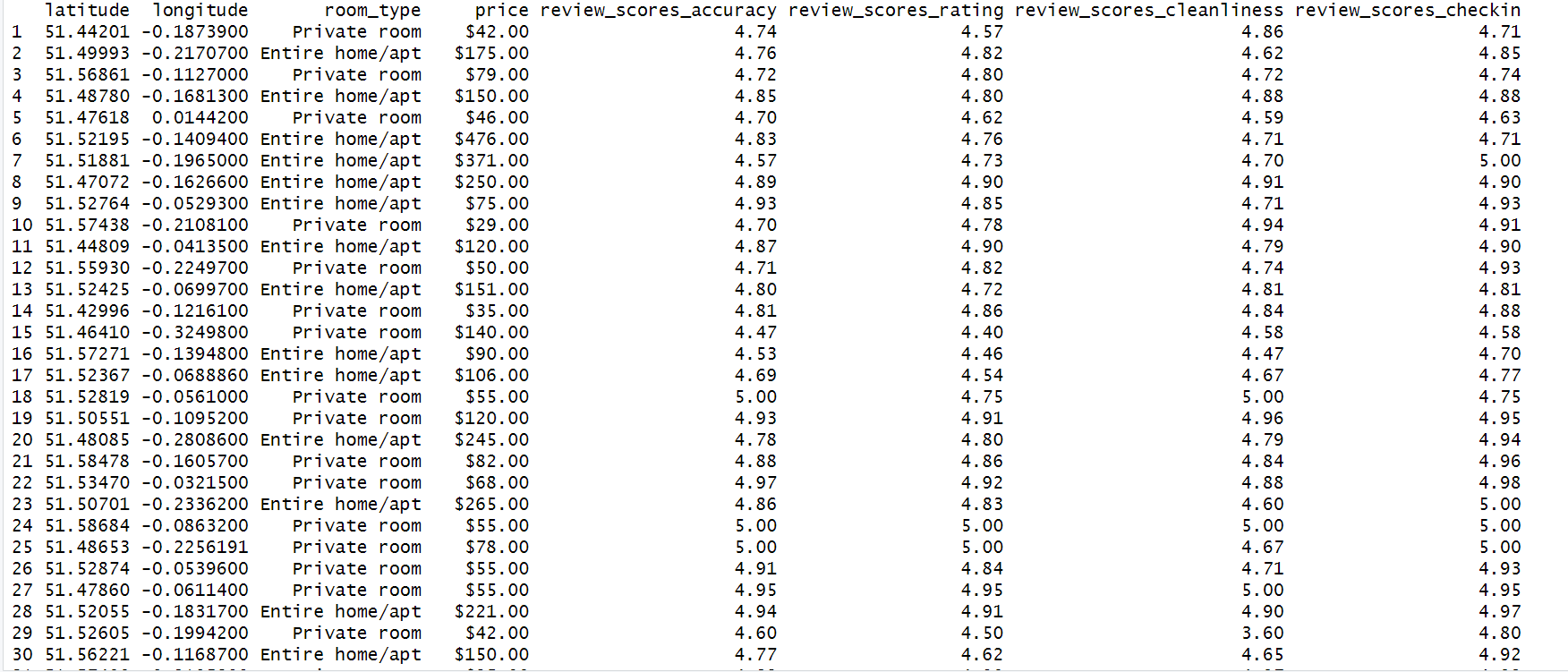
Data Preprocessing:

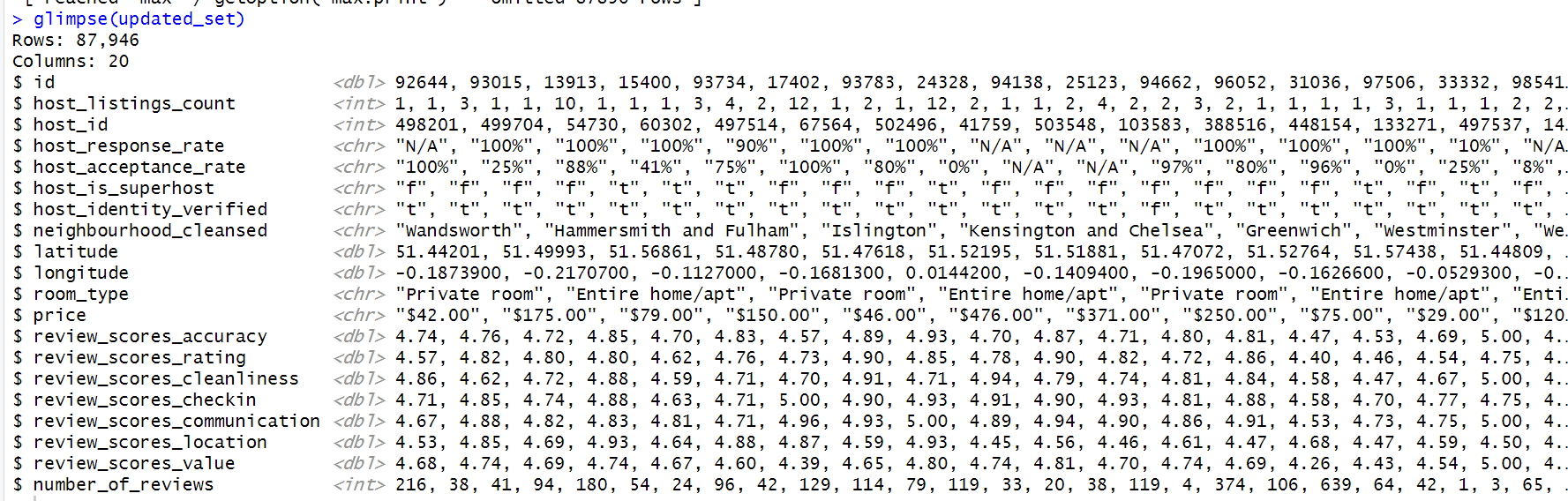
Whenever we do data analysis we make sure that our data is accurate , does not have blank spaces,Null values and any kind of outliers. When we ensure that our feature engineering and model building are likely going to show good results.Moreover, we need to check the data types for each of the columns that we are considering to use. That is a good practice because whenever we do model building and feature engineering we need to ensure that our key features are supportive to the nature of these. That is easier for the system to process and read, and ensures good results.

Column selection

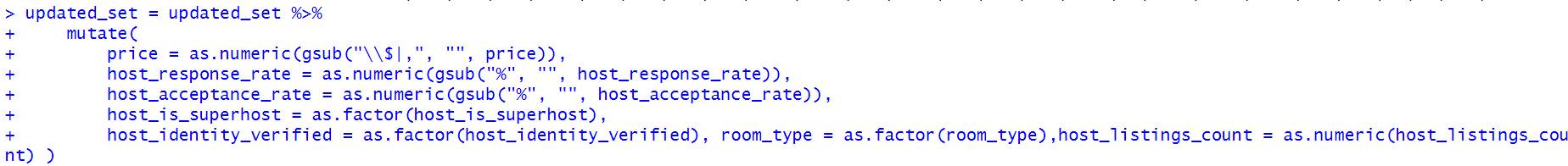


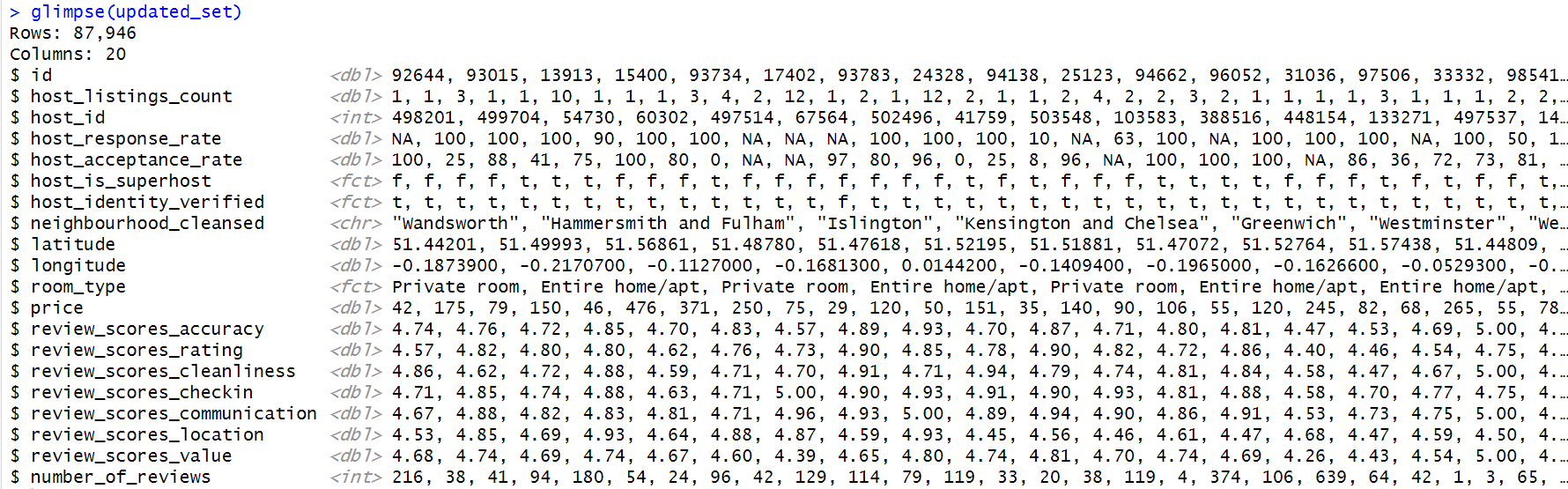






* Changing data types of columns



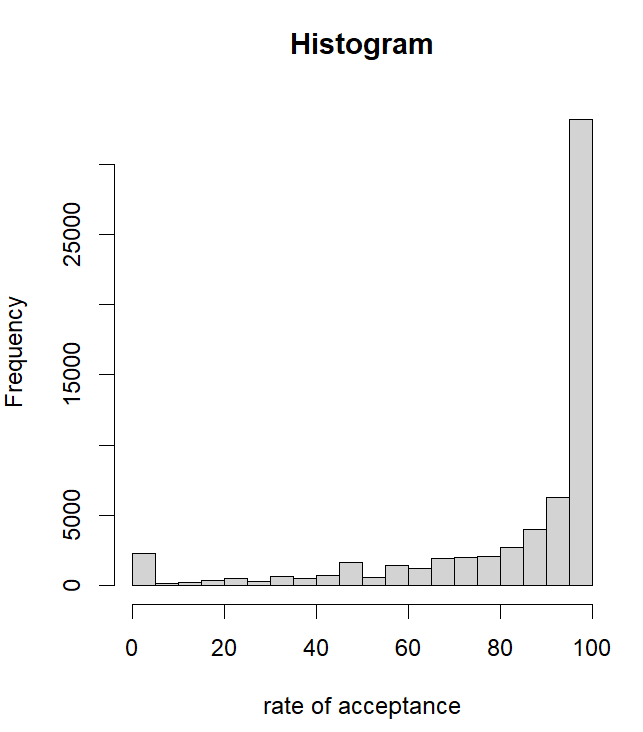
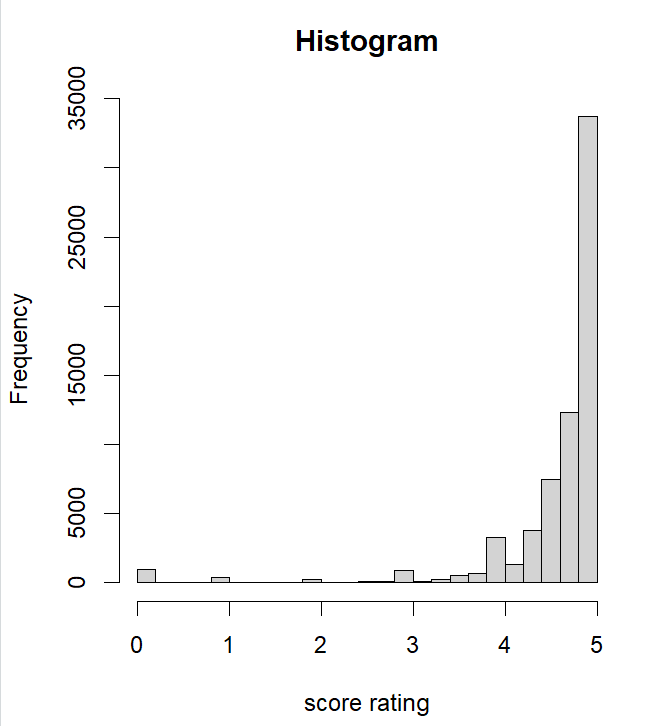
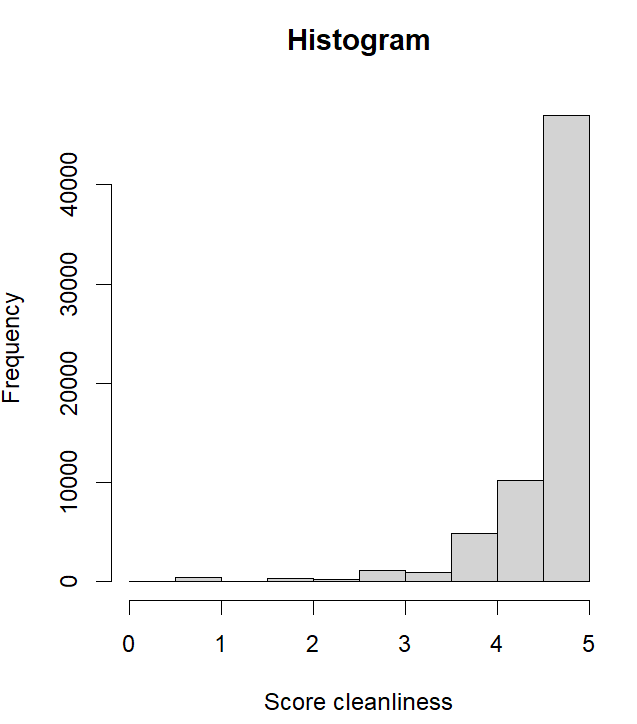


As we can see, there are a lot of NA’s within the dataset . It is crucial to deal with NA values and replace them with median or mean based on how skewed a graph is. The reason we replace the NA’s is because it can lead to biased and inaccurate results.

**Replacing NA’s based on the graph pattern**

I have plotted several histograms to see if the distribution is skewed to the left or right. In case of skewedness I will be replacing the NA’s with the median. If the histogram follows more of a regular distribution it is best to replace the NA’s with mean. In the following I will check and replace the NA’s accordingly.

* The **hist()** function in this case is going to create graphical representation of the distribution in a certain column.
* “main”- this is the title that we want to give for our histogram
* “xlab”- this is the label for out x-axis.

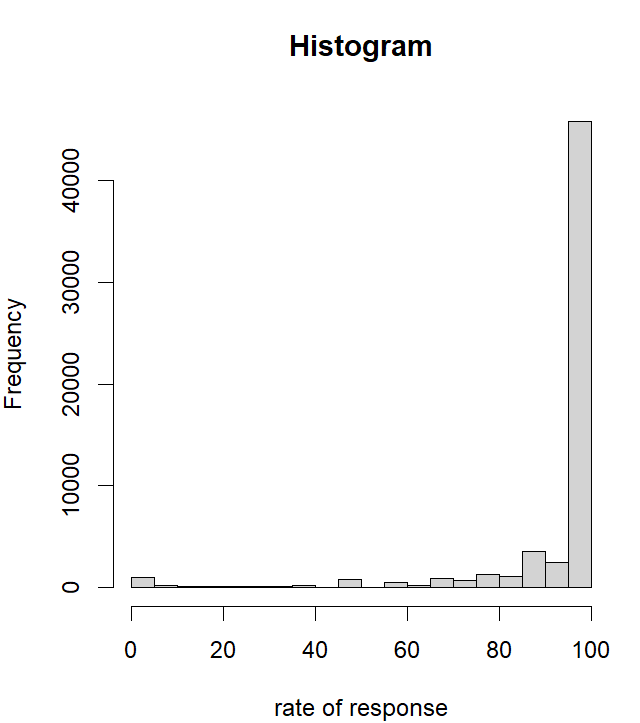


>hist(updated\_set$review\_scores\_cleanliness,main="Histogram",xlab="Score cleanliness")

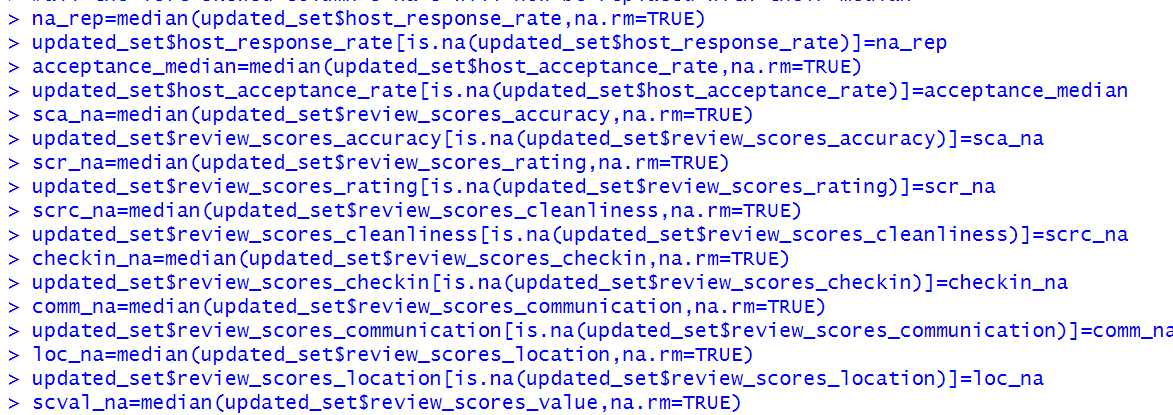
> hist(updated\_set$review\_scores\_rating,main="Histogram",xlab="score rating")

>hist(updated\_set$host\_acceptance\_rate,main="Histogram",xlab="rate of acceptance")

>hist(updated\_set$host\_response\_rate,main="Histogram",xlab="rate of response")



As can be observed above all of the distributions are left-skewed and in this case we will be replacing the NA’s with their medians like the following:

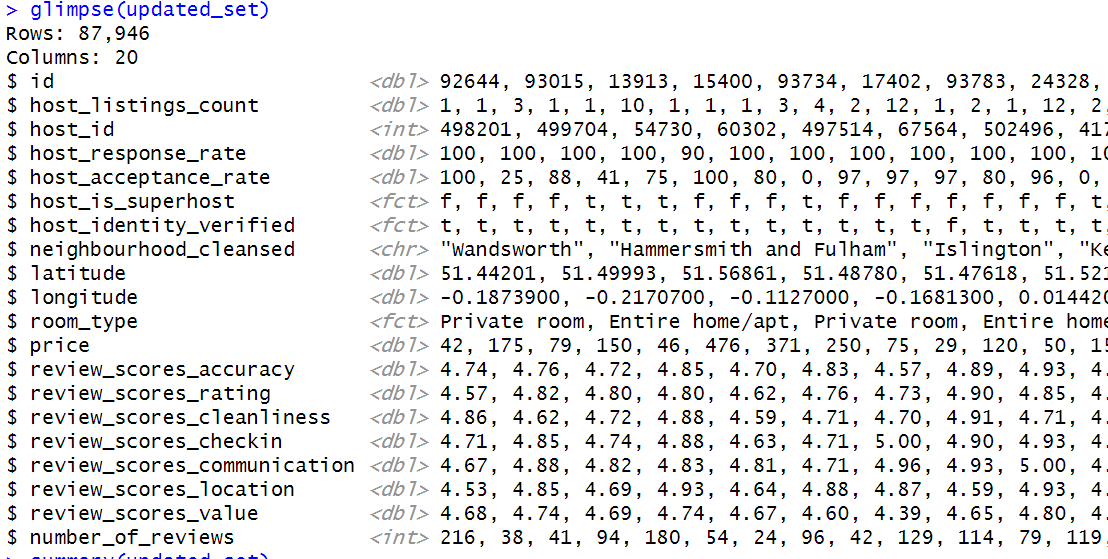


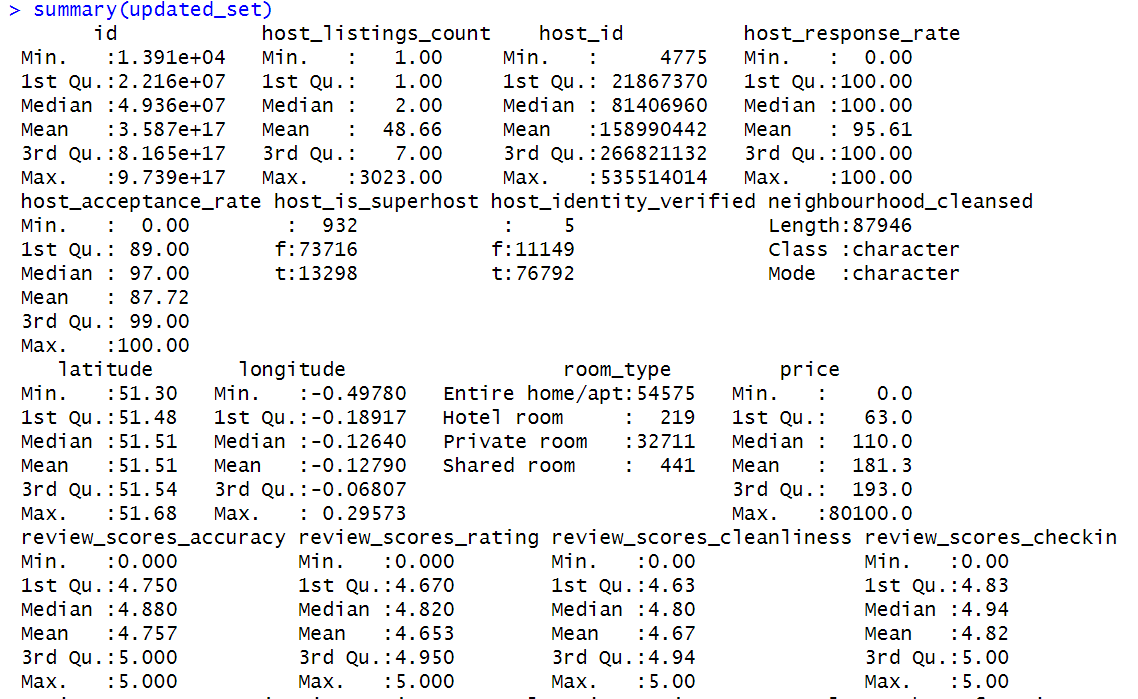
>updated\_set$review\_scores\_value[is.na(updated\_set$review\_scores\_value)]=scval\_na

> lscount\_na=median(updated\_set$host\_listings\_count,na.rm=TRUE)

We are calculating the median for each of the column containing NA’s by using the median() function.the na.rm=TRUE is used to exclude missing values from the calculation.After this step we are checking if the columns are having any kind of NA values by using **is.na** In case they do , those places will be be replaced by the median calculated for that column.

To see if the na’s were replaced or not we will get a glimpse into out dataset “updated\_set” using the function “**glimpse()”**





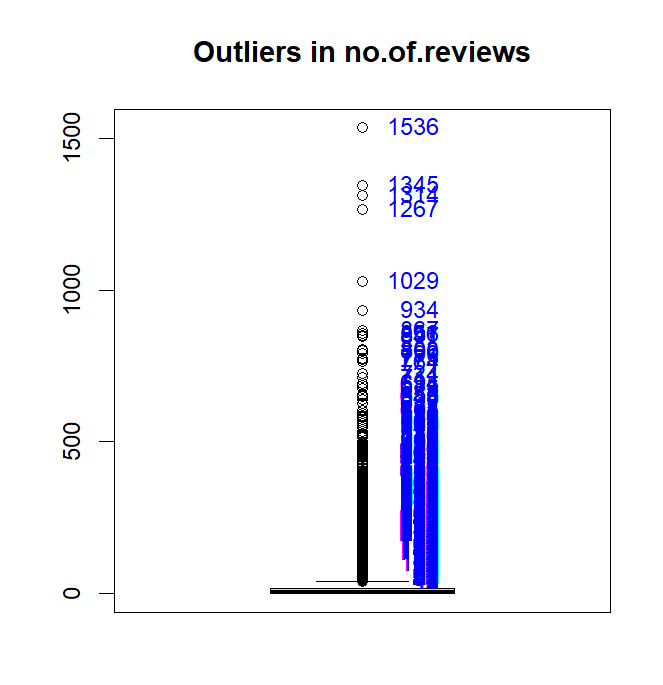
The na’s for all the columns with either left or right skewed were now eliminated and is visible through the summary() function along with the quartiles, mean and the median of the columns.

**Checking for Outliers –**

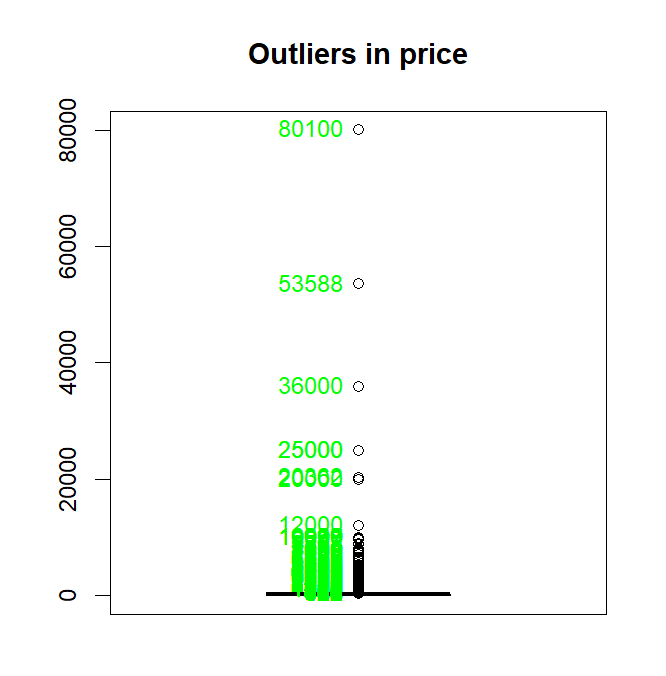
We will plot box plot to see the outliers present in the specific column of the dataset.

>noreviews\_plot=boxplot.stats(updated\_set$number\_of\_reviews)

>boxplot(updated\_set$number\_of\_reviews,main="Outliers in no.of.reviews",outline = TRUE) >text(x=1.2,y=noreviews\_plot$out,labels=round(noreviews\_plot$out,2),pos=2,col="blue")

The following code gives me this boxplot: 

Similarly a box plot for price has also been plotted. After plotting the boxplot we come to know that there are outliers in our columns and we will remove that.



price\_plot=boxplot.stats(updated\_set$price)

> boxplot(updated\_set$price,main="Outliers in price",outline = TRUE)

> text(x=1,y=price\_plot$out,labels=round(price\_plot$out,2),pos=2,col="green")

Now we will check for 0’s in the columns I will be checking for review\_score\_accuracy :

The code ensures that zeros in the review\_scores\_accuracy column are identified, counted and are replaced or removed accordingly.

any(updated\_set$review\_scores\_accuracy == 0)

[1] TRUE

Since this is true I will now be removing the zeros -

zero\_count <- sum(updated\_set$review\_scores\_accuracy == 0)

> print(zero\_count)

[1] 8

* Since I was getting zero values I will be replacing it with the mean values in place of zeros and chcking if after updating any zeros are left or not:

mean\_without\_zeros=mean(updated\_set$review\_scores\_accuracy[updated\_set$review\_scores\_accuracy != 0], na.rm = TRUE)

> #now replace the zeros with the mean

> updated\_set$review\_scores\_accuracy[updated\_set$review\_scores\_accuracy == 0] = mean\_without\_zeros

zero\_count <- sum(updated\_set$review\_scores\_accuracy == 0)

> print(zero\_count)

[1] 0

* Just to be ensure consistency we also checked if there are any other columns that are having the values 0.

#checking if any other column has 0's as values.

> any\_zero\_values <- any(apply(updated\_set, 2, function(x) any(x == 0)))

> # Print the result

> print(any\_zero\_values)

[1] FALSE

Now that the zeros and null values have been removed I will now be removing the outliers.using a user-defined function.:

I have used Inter-Quartile Range to remove my outliers because it is the least sensitive to extreme values compared to the mean etc.

outlier\_rm = function(data, column\_name, threshold = 1.5) {

column = data[[column\_name]]

quart1 = quantile(column, 0.25)

quart3 = quantile(column, 0.75)

interquart = quart3 - quart1

lw\_bound = quart1 - threshold \* interquart

up\_bound = quart3 + threshold \* interquart

# Identify boundary of outliers

outliers = which(column < lw\_bound | column > up\_bound)

# new data frame without outliers

data\_wout\_outliers = data[-outliers, ]

# Return the data frame without outliers and the indices of outliers

return(list(data\_wout\_outliers, outliers))

} #removing outliers from price

**Renaming the data frame since the name is not feasible and checking the dimensions.**

**I have renamed my dataframe to final\_rm that includes no null values, no zeros, and no outliers.**

>remove = outlier\_rm(updated\_set, "price") # Retrieve the dframe without any outliers

>final\_rm <- remove[[1]]

>dim(final\_rm)

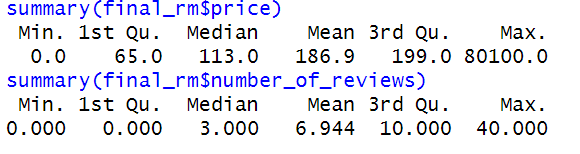
>summary(final\_rm$price)

>summary(final\_rm$number\_of\_reviews)

>remove2 = outlier\_rm(updated\_set, "number\_of\_reviews") # Retrieve the dfra>me without any outliers for number of reviews column

>final\_rm = remove2[[1]]

>summary(final\_rm)



After all the processing I will now do the visualizations that are going to help me to make a firm decision and consider several factors of my hypothesis.

* The price according to each neighbourhood
* The listings in which the host is verified and not verified.
* How many listings are there for a specific room type?
* Listings in which the host is a superhost and in which he/she isn’t.

After generating a summary I came to know that I had missing values in the column host\_is\_superhost of 1% which is 798 rows and hence I substituted the blank values for the mode which is t or f’s mode in the column. This ensures that the regular distribution is not disrupted and it will preserve the majority pattern.

I did it in this way :

# Find the mode of the host\_is\_superhost column

> mode\_value <- as.factor(as.character(names(sort(table(final\_rm$host\_is\_superhost), decreasing = TRUE)[1])))

> # Replace missing and NA values with the mode value

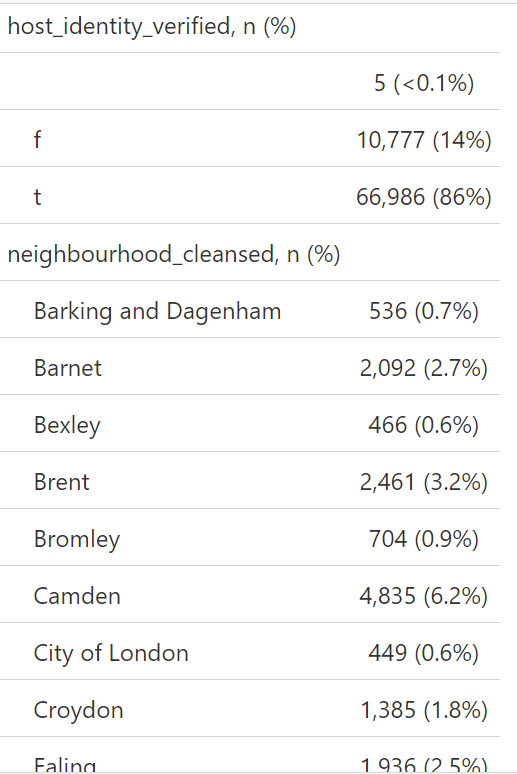
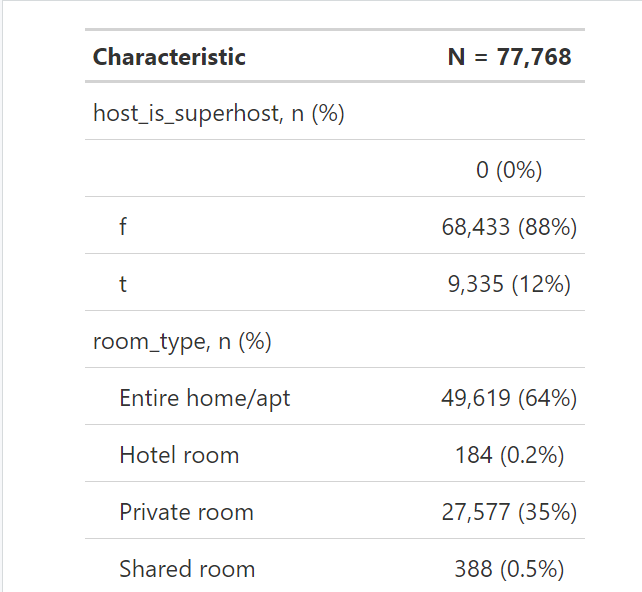
> final\_rm$host\_is\_superhost[is.na(final\_rm$host\_is\_superhost) | final\_rm$host\_is\_superhost == ""] <- mode\_value

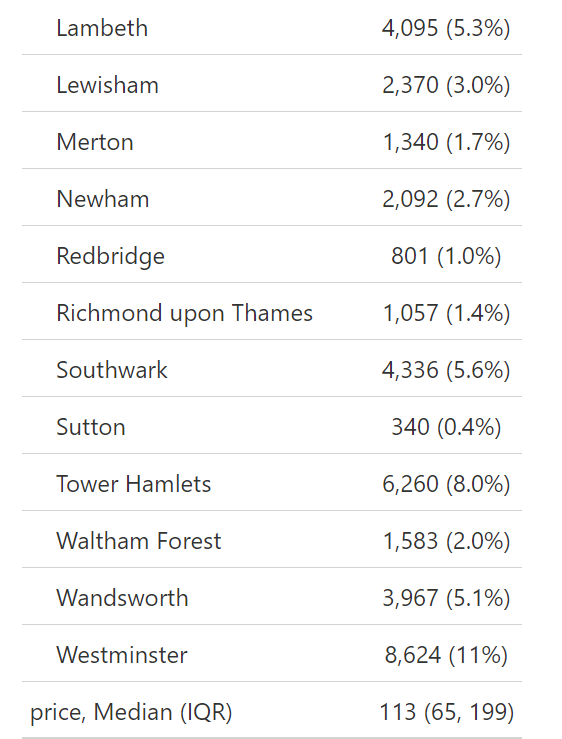
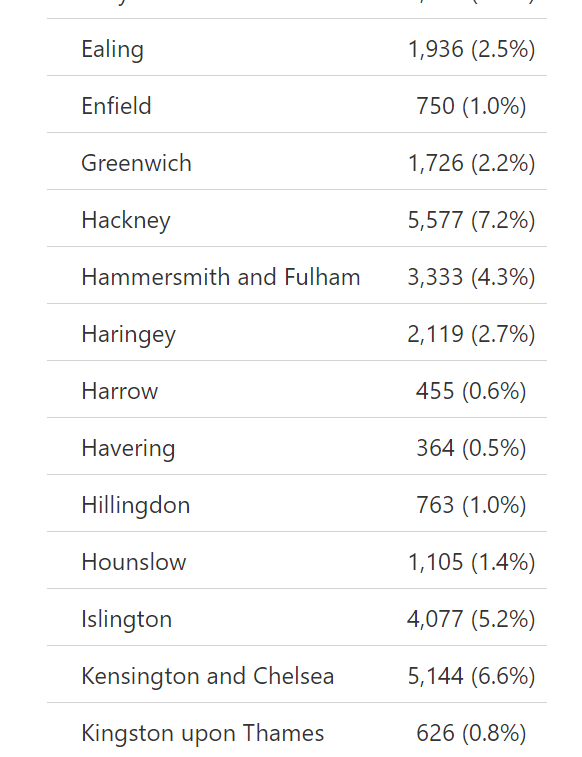
After doing this I generated a summary again and got the blank values to be at 0%.

library(gtsummary)

> y = final\_rm %>% dplyr::select("host\_is\_superhost","room\_type","host\_identity\_verified","neighbourhood\_cleansed","price" )

> y %>% gtsummary::tbl\_summary() %>% add\_stat\_label()





Using the **gtsummary()** we generate a table which gives us more information about our dataset.

**Some of the observations:**

* Entire home/apt make the highest percentage of the room types.
* Westminister makes up the highest percentage of the neighbourhoods.
* Host\_identity\_verified and host\_is\_superhost is also following a unique trend.
* All this information is going to help us to better understand the visualizations.

**Visualizations:**

to better understand the data I have plotted visuals that will help me gain insights into the current data-set.

* The following code is showing the relationship between neighbourhood cleansed and the price color-coded by the “host\_is\_superhost”.

filter(host\_is\_superhost != "") %>%

ggplot() +

aes(x = neighbourhood\_cleansed, y = price, fill = host\_is\_superhost) +

geom\_col(position = "dodge") +

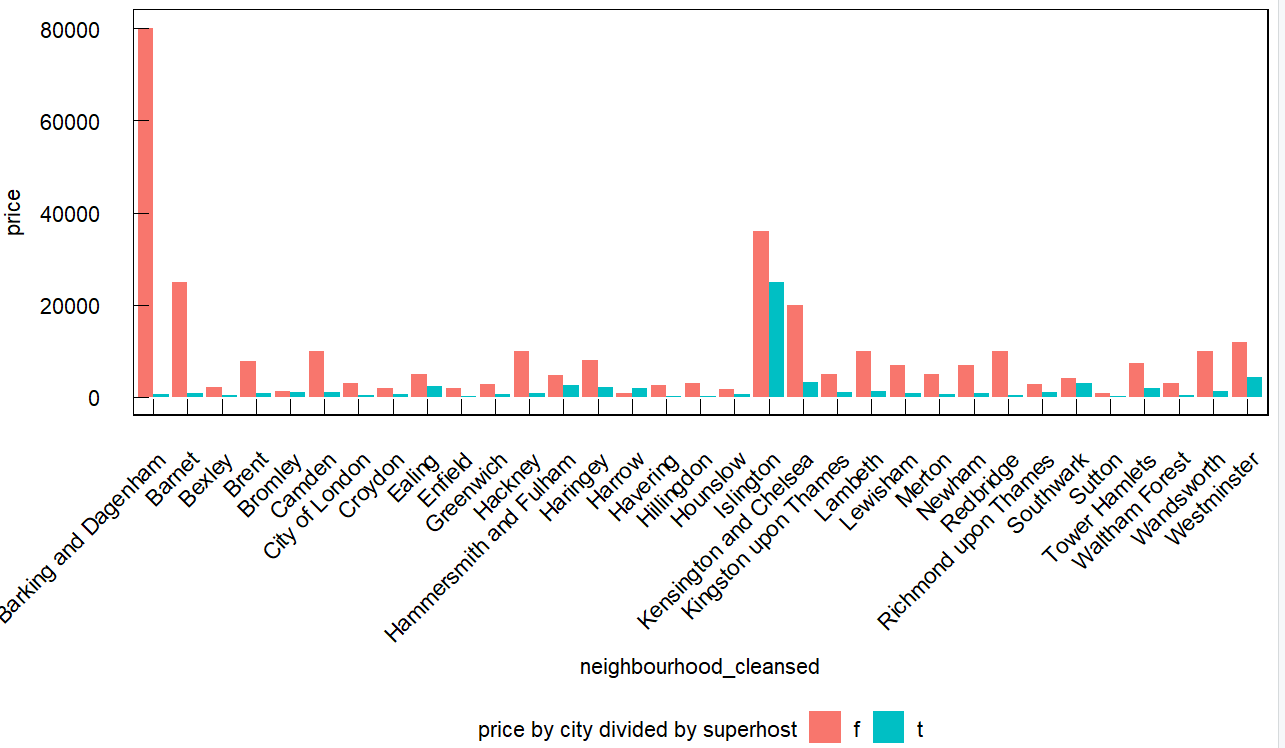
# to add a gradient effect

scale\_color\_gradient() +

+ ggnuplot::theme\_gnuplot()+

+ theme(legend.position = "bottom",axis.text.x = element\_text(angle = 45, hjust = 1))+

+ labs(x = "neighbourhood\_cleansed", y = "price", fill = "price by city divided by superhost")



Using **ggplot()** we are showing a bar plot the count for each of the room types. **geom\_bar()** will add bars to the plot. **geom\_text()** will add text to the bar plot . **stat()** will show the count of each room type.**vjust()** is showing the vertical justification of the labels. **Labs()** is showing the title of the bar plot.

With this information I have further plotted multiple graphs.

**Total listing by room type**

library(ggplot2)

> library(dplyr)

>

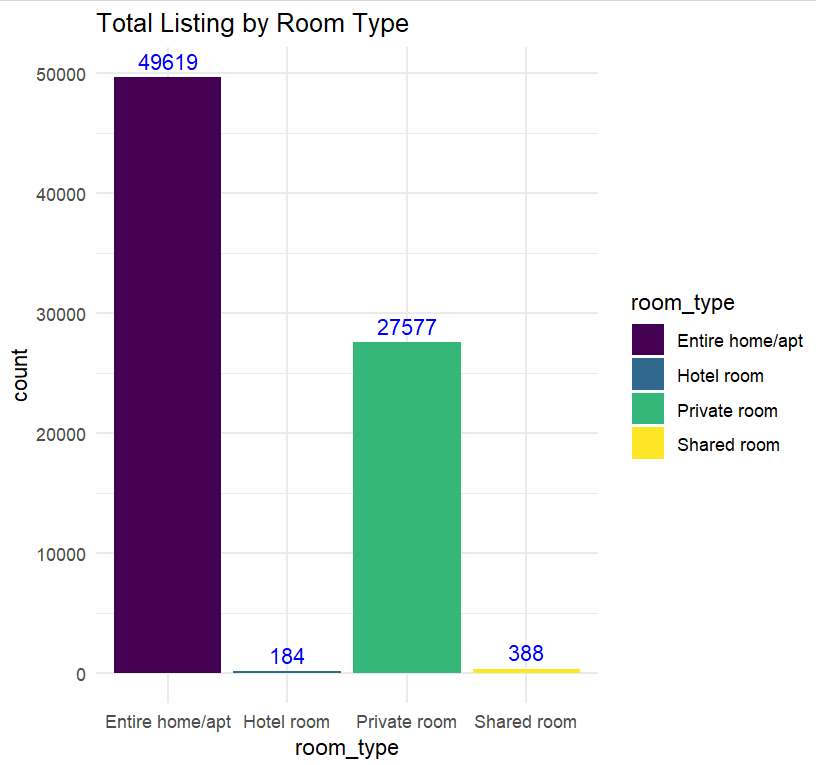
> ggplot(final\_rm, aes(x = room\_type, fill = room\_type)) +

geom\_bar() +

geom\_text(stat = "count", aes(label = stat(count)), color = "blue", vjust = -0.5) +

labs(title = "Total Listing by Room Type")+scale\_fill\_viridis\_d() + # Using Viridis color palette for fill

theme\_minimal()



**Average price by room type**

> ggplot(final\_rm, aes(x = room\_type, y = price, fill = room\_type)) +

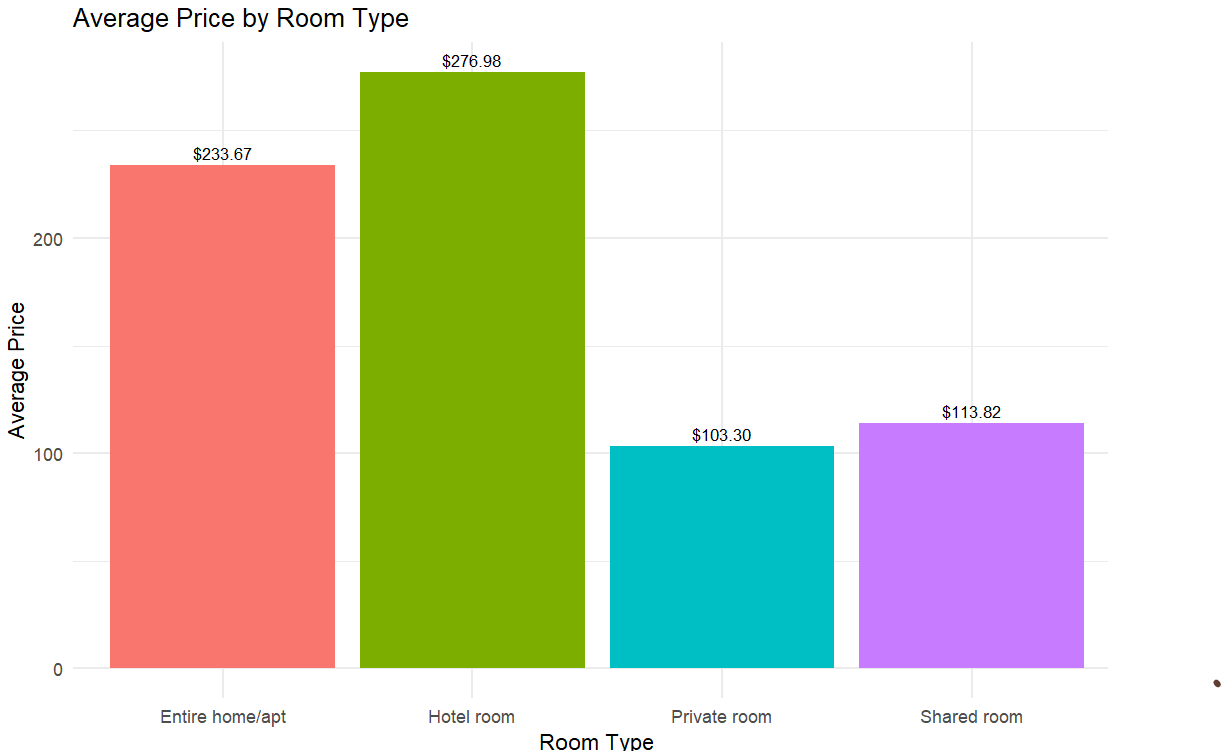
geom\_bar(stat = "summary", fun = "mean", position = "dodge", show.legend = FALSE) +

stat\_summary(fun = "mean", geom = "text", aes(label = sprintf("$%.2f", ..y..)),

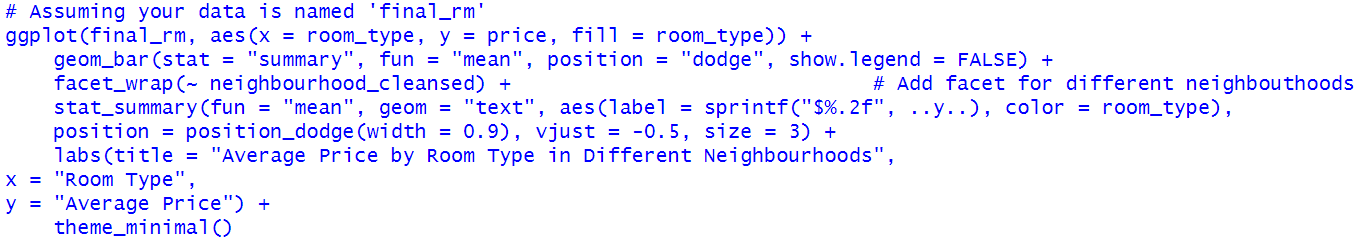
position = position\_dodge(width = 0.9), vjust = -0.5, size = 3) +

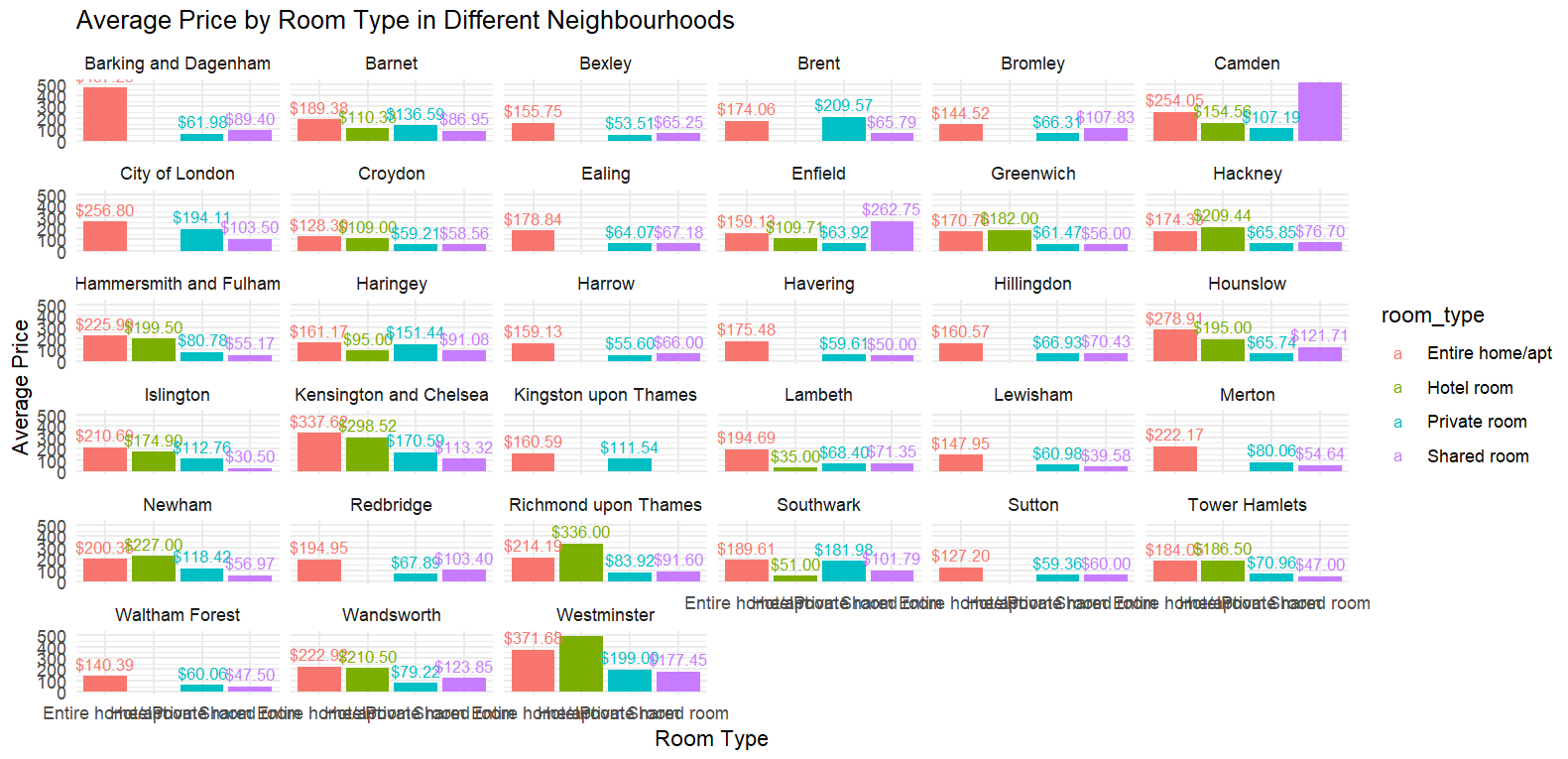
labs(title = "Average Price by Room Type", x = "Room Type",y = "Average Price") +

theme\_minimal()



**Average price by room types in different neighbourhoods.**





The hotel room in sutton, Waltham forest, redbridge don’t have a lot of price this could be due the the non existence of them or very few. on the other hand entire home in barking and Dagenham,city of London,Kensington and Chelsea and westminister are having a higher price.

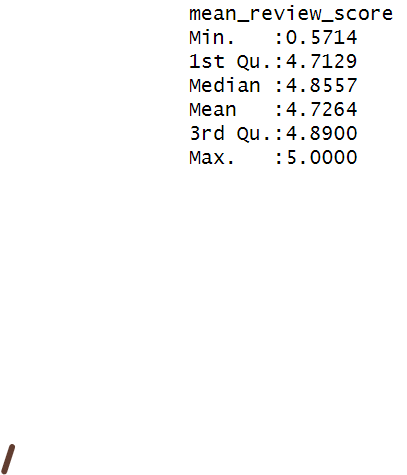
After doing the visualizations I now need to work on my feature engineering, for doing that I will be firstly summarizing all of my review column as one so ii have the average review score with me.

For that :

> # make a new column mean review score and assign to it the mean of all of the different considerations of review score.

> final\_rm$mean\_review\_score = rowMeans(final\_rm[, c("review\_scores\_accuracy", "review\_scores\_rating",

+ "review\_scores\_cleanliness", "review\_scores\_checkin", "review\_scores\_communication", "review\_scores\_location", "review\_scores\_value")], na.rm = TRUE)



all the columns of review have now summed up as mean for the mean\_review\_score this step will help when you will assign weights for feature engineering.

I will also change the data type of room type since it is an important feature to predict revenue:

> #for target variable generation i want to consider the room type so i have to convert it to factor datatype.

> final\_rm$room\_type = factor(final\_rm$room\_type, levels = c("Entire home/apt", "Hotel room", "Private room", "Shared room"))

However, we will still convert it to numeric datatype when we are assigning the weights to out target variable.

* In the following step I have generated a variable called weight which will hold the weight of the variables I want to consider when making my target variable revenue.

> weight = c(host\_acceptance\_rate = 0.15, host\_identity\_verified=0.15,price = 0.2, room\_type = 0.2, mean\_review\_score = 0.15,number\_of\_reviews=0.15 )

# I am making a new variable listing score and assigning to it the weight of the variable I took into consideration while generating these variables

> final\_rm$listing\_score <- with(final\_rm,

host\_acceptance\_rate \* weight['host\_acceptance\_rate'] +

as.numeric(host\_identity\_verified) \* weight['host\_identity\_verified'] +

price \* weight['price'] +

as.numeric(room\_type) \* weight['room\_type'] +

mean\_review\_score \* weight['mean\_review\_score'] +

number\_of\_reviews \* weight['number\_of\_reviews']

)

> library(ggplot2)

> ggplot(final\_rm, aes(x = room\_type, y = revenue, fill = room\_type)) +

After creating the variable I need to categorize if the variable listing score is good or bad for that I have checked the distribution of my listing score and since it is normal I have calculated the mean and have set that as my threshold value like below:

mean\_th=mean(final\_rm$listing\_score,na.rm=TRUE)

> mean\_th

[1] 52.99715

After performing this step I will now create a factor variable “classification” and assign the threshold to it to classify if it is good or bad.

final\_rm$classification = ifelse(final\_rm$listing\_score >= mean\_th, 'Good', 'Bad')

> head(final\_rm$classification)

[1] "Bad" "Good" "Bad" "Bad" "Bad" "Bad"

After doing this I will be implementing my model for logistic regression and decision tree.

> set.seed(60)

> indices\_samp = sample(1:nrow(final\_rm), 0.7 \* nrow(final\_rm)) #split 70% for train and 30% for test

> train\_data = final\_rm[indices\_samp, ]

> test\_data = final\_rm[-indices\_samp, ]

Explanation:

The set.seed() ensures that everytime the code is run it will produce the same results in short it ensures reproducibility.

* Indices\_samp = sample(1:nrow(final\_rm), 0.7 \* nrow(final\_rm)) #split 70% for train and 30% for test

this line generates several indices starting from 1 to the number of rows in the dataset.

> train\_data = final\_rm[sample\_indices, ]- here the train data set is separated from the test dataset and it is retaining all the rows of the original subset. Similarly for the test data I have made sure to include

> log\_model = glm(classification ~ host\_acceptance\_rate + host\_identity\_verified + price +

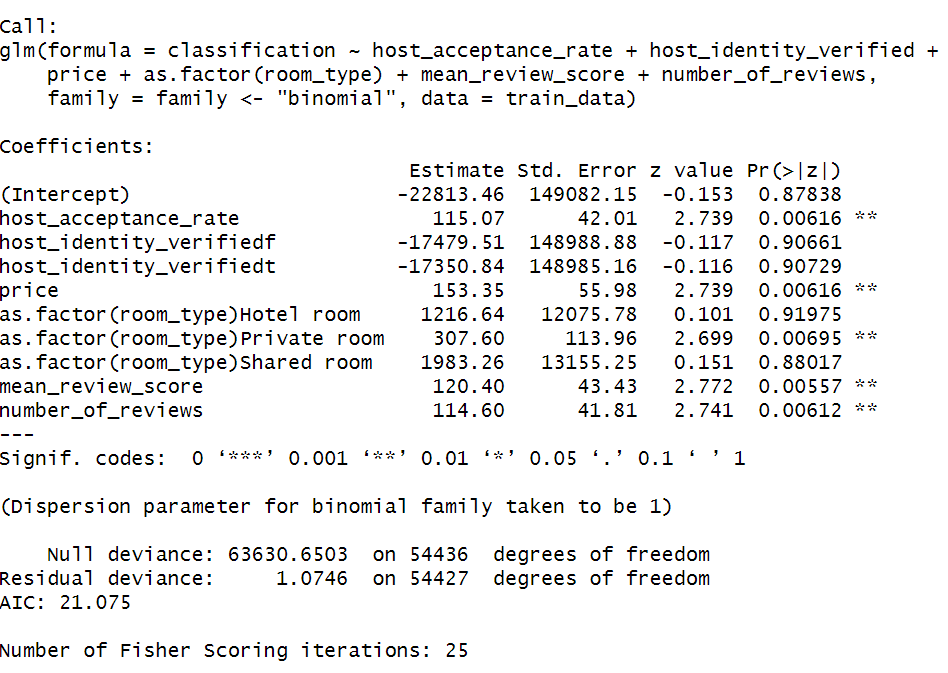
+ as.factor(room\_type) + mean\_review\_score + number\_of\_reviews,

+ family <- "binomial", data = train\_data)

The log model is specifying a formula where classification is the response variable and the rest of the columns are the predictor variables which means that each of these variables will be taken into consideration when predicting the classification.

After building the model we look at the summary and this is what the summary is giving us.

> summary(log\_model)



* We got an intercept of -22813.46 which states that the log-odds of the response variable are in the ‘good’ category even when the predictor variable is 0.
* host\_acceptance\_rate – it has a coefficience of 115.07 which shows that there is a one-unit increase in it and is showing a high significant p-value and this states that it is statistically significant in predicting the response variable.

host\_identity\_verifiedf: it is showing coefficience of -17479.51 (not statistically significant).

host\_identity\_verifiedt: it is showing a significance of -17350.84 (not statistically significant)

price: 153.35 coefficience is significant as this shows a unit increase and it has a p values of 0.006 which is highly significant in predicting the response variable

as.factor(room\_type)Hotel room: coefficience of 1216.64 (this is not statistically significant)

as.factor(room\_type)Private room: this is showing a co-efficience of 307which is statistically significant and it has a p value of 0.006 which is highly significant.

as.factor(room\_type)Shared room: 1983.26 (this is not statistically significant)

mean\_review\_score: 120.40 coefficience and p value less the 0.01 shows that it is highly significant.

number\_of\_reviews: 114.60 (p-value < 0.01)

Model Evaluation:

Accuracy: 99.98% this is a good accuracy I have also tried doing the same with train data and it is almost the same which shows minimal overfitting.

> predicted\_prob <- predict(log\_model, newdata = test\_data, type = "response")

> predicted\_classify <- ifelse(predicted\_prob >= 0.5, '1', '0')

> confusion\_matrix = table(test\_data$classification, predicted\_classify)

> log\_accuracy = sum(diag(confusion\_matrix)) / sum(confusion\_matrix)

> log\_accuracy

[1] 0.9998286

> confusion\_matrix

predicted\_classify

0 1

Bad 17027 2

Good 2 6300

As I have performed logistic regression for the test data the accuracy is quite high this is giving signs of overfitting however upon testing the model on train data the accuracy sums up to 1 which is indicaticating that it the difference is not very significant and hence I have come to the conclusion that overfitting is at its minimal.

Here is my model testing on my train data and as can be observed although the accuracy is higher it is not very significant.

> predicted\_probte <- predict(log\_model, newdata = train\_data, type = "response")

> predicted\_classifyte <- ifelse(predicted\_probte >= 0.5, '1', '0')

>

> confusion\_matrixte = table(train\_data$classification, predicted\_classifyte)

> log\_accuracyte = sum(diag(confusion\_matrixte)) / sum(confusion\_matrixte)

> log\_accuracyte

[1] 1

> confusion\_matrixte

predicted\_classifyte

0 1

Bad 39674 0

Good 0 14763

DECISION TREE:

To further evaluate my dataset I have made decision trees :

> # i will install the rpart and the rpartplot packages.

> install.packages("rpart")

> install.packages("rpart.plot")

library(rpart)

> dectree\_model <- rpart(classification ~ host\_acceptance\_rate + host\_identity\_verified + price +

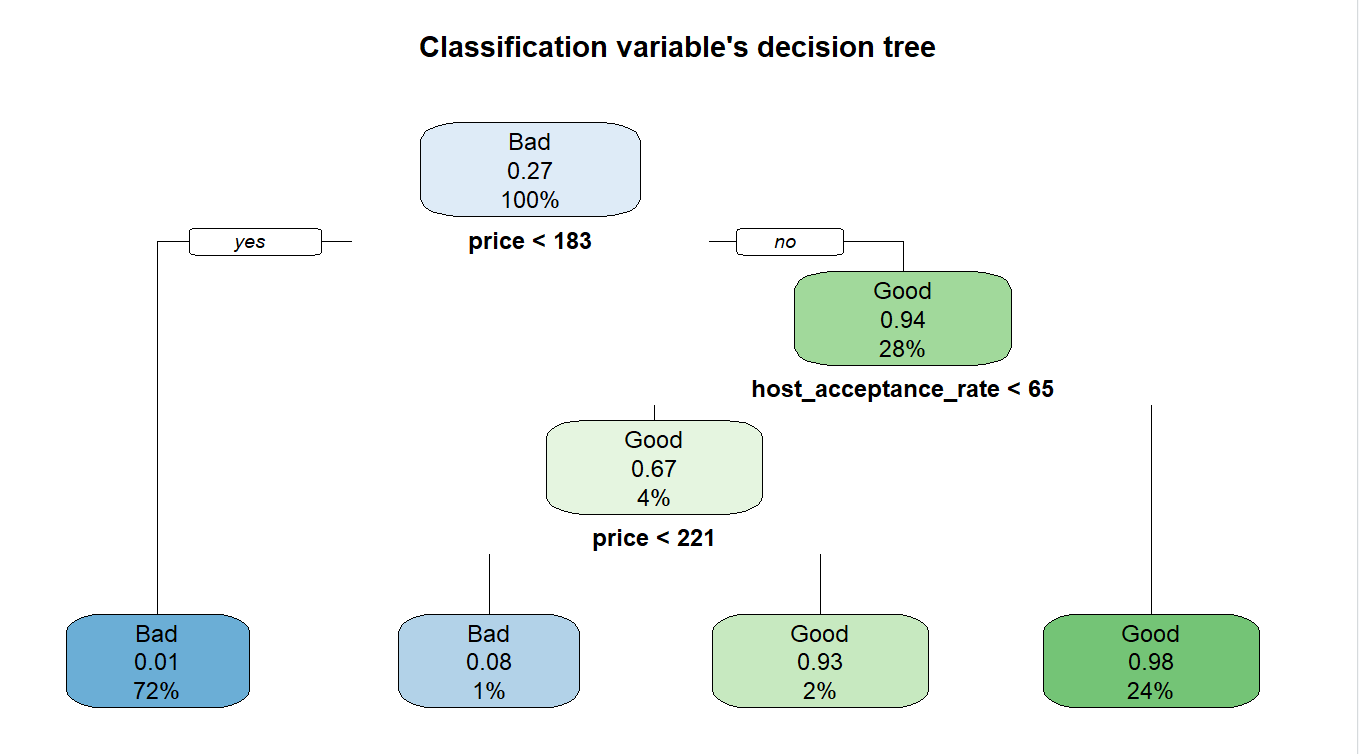
as.factor(room\_type) + mean\_review\_score + number\_of\_reviews, data = train\_data, method = "class")

> #load the library rpartplot to plot a decision tree for the model

#load the library rpartplot to plot a decision tree for the model as it helps us to view the decision tree in a diagrammatic view.

> library(rpart.plot)

> rpart.plot(dectree\_model, main = "Classification variable's decision tree")



The following shows that if the price is less than 183 it will be classified as bad and if greater it will be classified as good.and if the host acceptance rate is less then 65 which is 4 percent it will still see if price is relevant or not

> #now we will make predictions for our test data

> tree\_pred = predict(dectree\_model, newdata = test\_data, type = "class")

> #i will now evaluate the decision tree

> #model

> dectree\_confusion = table(test\_data$classification, tree\_pred)

> tree\_accuracy <- sum(diag(dectree\_confusion)) / sum(dectree\_confusion)

> print(dectree\_confusion)

tree\_pred

Bad Good

Bad 16876 153

Good 202 6100

> print(paste("Accuracy of the dt model:", tree\_accuracy))

[1] "Accuracy of the dt model: 0.984784192704985"