ASSESSMENT 2

POONAM PRAFUL SHRISHRIMAL_s8075211

2023-11-08

1. INTRODUCTION

Airbnb is an online marketplace since 2008 that facilitates connections between hosts and guests globally, resulting in extensive data. Analyzing this data is essential for making informed decisions and driving improvement in various areas. This research centers on Airbnb's dataset specific to NYC, offering insights into property specifics, pricing, and additional aspects. The goals involve examining trends, exploring pricing influencers, and constructing predictive models for rental prices and occupancy rates.

```
#Stop / hide warnings
options(warn = -1)


#Import required libraries
library(tidyverse)
library(tidymodels)
library(scales)
library(stacks)
library(ggplot2)
library(dplyr)
library(tidytext)
library(textrecipes)
library(baguette)
library(naniar)
library(corrplot)
library(DT)
library(doParallel)
registerDoParallel()
library(yardstick)


#Import the given dataset
airbnb_dataset=read.csv ("E:/VU SYDNEY/BCO6008 PREDICTIVE ANALYTICS/ASSESSMENT2/train.csv")
```

2. DATA WRANGLING

Data wrangling encompasses managing missing or inconsistent data, converting data types, transformation and resolving other issues to ensure the data is ready for in-depth exploration and modeling.

2.1 Initial Exploration

```
#Lets take a glance at data and its type
glimpse(airbnb_dataset)
```

```
## Rows: 34,226
## Columns: 16
## $ id                            <int> 9901706, 299531, 2461439, 127387, 62931…
## $ name                          <chr> "Cute big one bedroom", "Feel like you …
## $ host_id                       <int> 1904415, 1220404, 12586492, 23276, 2397…
## $ host_name                     <chr> "Natalie", "Tom", "Sausan", "Katharine"…
## $ neighbourhood_group           <chr> "Manhattan", "Brooklyn", "Manhattan", "…
## $ neighbourhood                 <chr> "Upper West Side", "East New York", "Lo…
## $ latitude                      <dbl> 40.77789, 40.66795, 40.72007, 40.66862,…
## $ longitude                     <dbl> -73.97701, -73.89232, -73.98946, -73.99…
## $ room_type                     <chr> "Entire home/apt", "Entire home/apt", "…
## $ price                         <int> 180, 100, 133, 260, 120, 55, 47, 100, 1…
## $ minimum_nights                <int> 1, 1, 14, 30, 3, 7, 3, 3, 1, 3, 2, 3, 2…
## $ number_of_reviews             <int> 0, 119, 177, 3, 22, 98, 32, 9, 11, 169,…
## $ last_review                   <chr> NA, "2019-06-30", "2019-05-03", "2014-0…
## $ reviews_per_month             <dbl> NA, 1.39, 2.82, 0.03, 0.27, 1.75, 0.91,…
## $ calculated_host_listings_count <int> 1, 2, 2, 1, 1, 3, 4, 1, 1, 3, 1, 1, 4, …
## $ availability_365              <int> 0, 289, 221, 316, 189, 312, 258, 0, 0, …
```

The dataset comprises 16 features and a total of 34,226 data points, featuring diverse datatypes such as characters, integers, and floats. Notably, room type and neighborhood group, though fundamentally categorical, are assigned the character datatype, which may not be optimal for their representation.

```
# Check if there are any duplicate rows
any(duplicated(airbnb_dataset))
```

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

No duplicate values!

```
#Checking Null values
apply(airbnb_dataset,2,function(x) sum(is.na(x)))
```

```
##                        id                      name
##                         0                       9
##                   host_id                 host_name
##                         0                      14
##         neighbourhood_group               neighbourhood
##                         0                       0
##                  latitude                 longitude
##                         0                       0
##                 room_type                     price
##                         0                       0
##             minimum_nights          number_of_reviews
```

```
##                    0                    0
##          last_review       reviews_per_month
##                 7008                 7008
## calculated_host_listings_count       availability_365
##                    0                    0
```

The above shows there exists null values in the dataset especially the reviews per month and last review.

2.2 Handling Data Types

Handling data types ensures that your data is in a format suitable for analysis, modeling, and interpretation, ultimately contributing to the accuracy and reliability of your findings.

```
#Convert character to factor
names_to_factor <- c("neighbourhood_group", "room_type")
airbnb_dataset[names_to_factor] <- map(airbnb_dataset[names_to_factor], as.factor)
```

2.3 Handling Missing Values

Handling missing values is crucial for maintaining data quality, ensuring accurate analyses, and making informed decisions. It can be addressed by either removing them, filling in with appropriate values, or using statistical methods to impute. It depends the variable importance and percentage of missing.

```
#Replace the null values in reviews per month by mean i.e., Mean imputation
airbnb_dataset$reviews_per_month[is.na(airbnb_dataset$reviews_per_month)] <-
mean(airbnb_dataset$reviews_per_month, na.rm = TRUE)
summary(airbnb_dataset$reviews_per_month)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.01    0.27    1.21    1.38    1.59   58.50
#Checking NULL Values
apply(airbnb_dataset,2,function(x) sum(is.na(x)))
##                      id                    name
##                       0                       9
##                 host_id               host_name
##                       0                      14
##       neighbourhood_group            neighbourhood
##                       0                       0
##                latitude               longitude
##                       0                       0
##               room_type                   price
##                       0                       0
##           minimum_nights          number_of_reviews
```

```
##                         0                         0
##                last_review          reviews_per_month
##                      7008                         0
## calculated_host_listings_count           availability_365
##                         0                         0
```

```
# Remove null values from specific columns
cols_to_clean <- c("name", "host_name")
airbnb_dataset <- airbnb_dataset[complete.cases(airbnb_dataset[cols_to_clean]), ]
```

```
#Check null values again
apply(airbnb_dataset,2,function(x) sum(is.na(x)))
##                         id                       name
##                         0                          0
##                    host_id                  host_name
##                         0                          0
##          neighbourhood_group              neighbourhood
##                         0                          0
##                   latitude                  longitude
##                         0                          0
##                  room_type                      price
##                         0                          0
##              minimum_nights          number_of_reviews
##                         0                          0
##                last_review          reviews_per_month
##                      6998                         0
## calculated_host_listings_count           availability_365
##                         0                         0
```

The above shows last review still have 6998 values, will keep that column as it is for now.

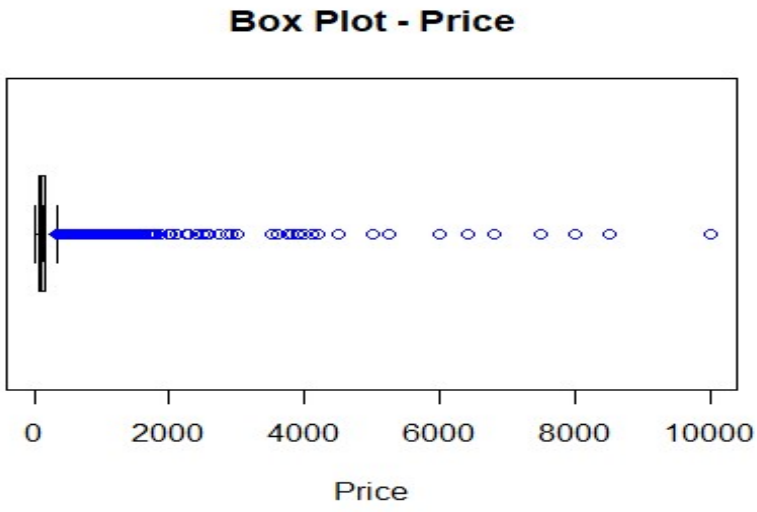
2.4 Identifying and dealing with outliers

Outliers are data points that deviate significantly from the majority of the data in a dataset. Handling outliers carefully is crucial to obtaining accurate and reliable insights from the data.

```
# Check the structure of dataset
str(airbnb_dataset)
## 'data.frame':    34203 obs. of  16 variables:
##  $ id                            : int  9901706 299531 2461439 127387 629315 4607923 12720048 4037379
9396442 763527 ...
##  $ name                          : chr  "Cute big one bedroom" "Feel like you never leave your home" "Pristine
Lower East Side Sanctuary" "Luxe, Spacious 2BR 2BA Nr Trains" ...
##  $ host_id                       : int  1904415 1220404 12586492 23276 2397437 1113080 68787921 5642757
26347594 3604585 ...
##  $ host_name                     : chr  "Natalie" "Tom" "Sausan" "Katharine" ...
```

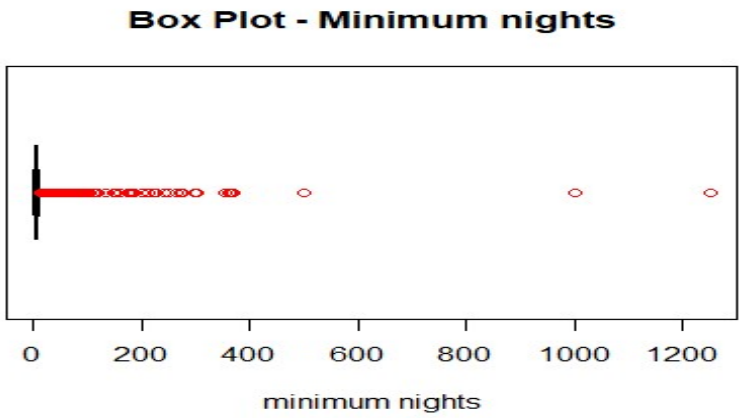
```
##  $ neighbourhood_group          : Factor w/ 5 levels "Bronx","Brooklyn",..: 3 2 3 2 2 2 2 2 3 2 ...
##  $ neighbourhood                : chr  "Upper West Side" "East New York" "Lower East Side" "Gowanus" ...
##  $ latitude                     : num  40.8 40.7 40.7 40.7 40.7 ...
##  $ longitude                    : num  -74 -73.9 -74 -74 -74 ...
##  $ room_type                    : Factor w/ 3 levels "Entire home/apt",..: 1 1 1 1 1 2 2 1 2 2 ...
##  $ price                        : int  180 100 133 260 120 55 47 100 108 70 ...
##  $ minimum_nights               : int  1 1 14 30 3 7 3 3 1 3 ...
##  $ number_of_reviews            : int  0 119 177 3 22 98 32 9 11 169 ...
##  $ last_review                  : chr  NA "2019-06-30" "2019-05-03" "2014-08-04" ...
##  $ reviews_per_month            : num  1.38 1.39 2.82 0.03 0.27 ...
##  $ calculated_host_listings_count: int  1 2 2 1 1 3 4 1 1 3 ...
##  $ availability_365             : int  0 289 221 316 189 312 258 0 0 188 ...

# Extract numeric columns
numeric_columns <- airbnb_dataset[sapply(airbnb_dataset, is.numeric)]



# Calculate summary statistics for numeric columns
summary_stats <- summary(numeric_columns)



# Display the summary statistics
print(summary_stats)
##       id              host_id           latitude      longitude
##  Min.   :   2539   Min.   :    2438   Min.   :40.51   Min.   :-74.24
##  1st Qu.: 9512812   1st Qu.:  7858210   1st Qu.:40.69   1st Qu.:-73.98
##  Median :19763012   Median : 30870512   Median :40.72   Median :-73.96
##  Mean   :19089608   Mean   : 67759668   Mean   :40.73   Mean   :-73.95
##  3rd Qu.:29229497   3rd Qu.:107434423   3rd Qu.:40.76   3rd Qu.:-73.94
##  Max.   :36487245   Max.   :274321313   Max.   :40.91   Max.   :-73.72
##      price       minimum_nights     number_of_reviews reviews_per_month
##  Min.   :    0   Min.   :  1.000   Min.   :  0.0     Min.   : 0.01
##  1st Qu.:   69   1st Qu.:  1.000   1st Qu.:  1.0     1st Qu.: 0.27
##  Median :  105   Median :  3.000   Median :  5.0     Median : 1.21
##  Mean   :  152   Mean   :  7.058   Mean   : 23.2     Mean   : 1.38
##  3rd Qu.:  175   3rd Qu.:  5.000   3rd Qu.: 23.0     3rd Qu.: 1.59
##  Max.   :10000   Max.   :1250.000   Max.   :629.0     Max.   :58.50
##  calculated_host_listings_count availability_365
##  Min.   :  1.00                 Min.   :  0.0
##  1st Qu.:  1.00                 1st Qu.:  0.0
##  Median :  1.00                 Median : 46.0
##  Mean   :  7.18                 Mean   :113.6
##  3rd Qu.:  2.00                 3rd Qu.:230.0
##  Max.   :327.00                 Max.   :365.0

# Box plot for price
boxplot(airbnb_dataset$price,
      main = "Box Plot - Price",
```

```
    xlab = "Price",
    notch = TRUE,       # Add a notch to the box plot for quartiles
    outcol = "blue",    # Color outliers in blue
    horizontal = TRUE   # Create a horizontal box plot

)
```



The above boxplot shows presence of outliers but it should not be considered as outliers as the luxury properties can have higher rental price.

```
# Box plot for minimum nights
boxplot(airbnb_dataset$minimum_nights,
    main = "Box Plot - Minimum nights",
    xlab = "minimum nights",
    notch = TRUE,       # Add a notch to the box plot for quartiles
    outcol = "red",     # Color outliers in blue
    horizontal = TRUE   # Create a horizontal box plot
)
```



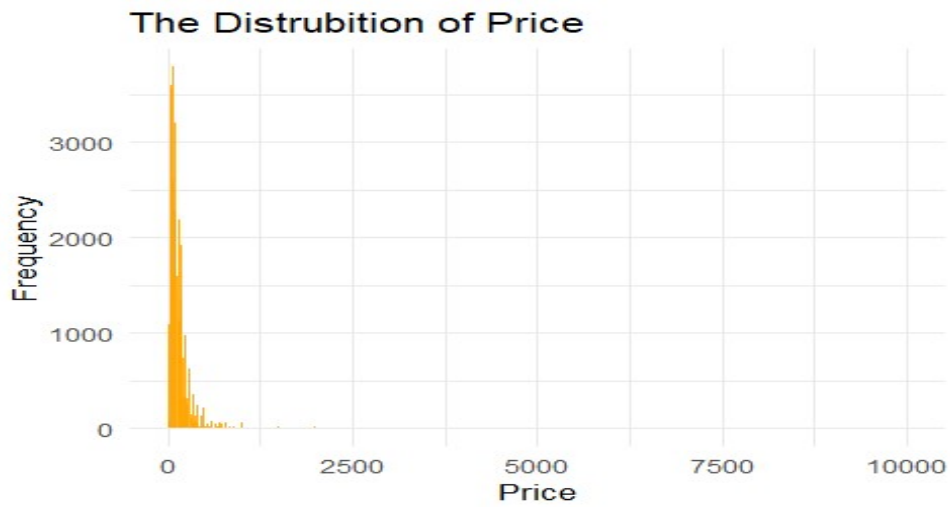
The box plot above shows presence of outliers as minimum nights can't be say 200, 400 , 1200 etc. This needs to be taken care of as below.

```
# Creating a function to remove the outliers from our data
rm_outliers = function(x){

  uc1 = quantile(x, probs = 0.95, na.rm = TRUE)
  lc1 = quantile(x, probs = 0.05, na.rm = TRUE)

  x[x > uc1] = uc1
  x[x < lc1] = lc1

  return(x)

}

# Removing outliers from the variable "minimum_nights"
airbnb_dataset$minimum_nights = as.numeric(sapply(airbnb_dataset["minimum_nights"],
rm_outliers))

summary(airbnb_dataset$minimum_nights)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   3.000   5.978   5.000  30.000

# Rechecking the data
glimpse(airbnb_dataset)
## Rows: 34,203
## Columns: 16
## $ id                            <int> 9901706, 299531, 2461439, 127387, 62931…
## $ name                          <chr> "Cute big one bedroom", "Feel like you …
## $ host_id                       <int> 1904415, 1220404, 12586492, 23276, 2397…
## $ host_name                     <chr> "Natalie", "Tom", "Sausan", "Katharine"…
## $ neighbourhood_group           <fct> Manhattan, Brooklyn, Manhattan, Brookly…
## $ neighbourhood                 <chr> "Upper West Side", "East New York", "Lo…
## $ latitude                      <dbl> 40.77789, 40.66795, 40.72007, 40.66862,…
## $ longitude                     <dbl> -73.97701, -73.89232, -73.98946, -73.99…
## $ room_type                     <fct> Entire home/apt, Entire home/apt, Entir…
## $ price                         <int> 180, 100, 133, 260, 120, 55, 47, 100, 1…
## $ minimum_nights                <dbl> 1, 1, 14, 30, 3, 7, 3, 3, 1, 3, 2, 3, 2…
## $ number_of_reviews             <int> 0, 119, 177, 3, 22, 98, 32, 9, 11, 169,…
## $ last_review                   <chr> NA, "2019-06-30", "2019-05-03", "2014-0…
## $ reviews_per_month             <dbl> 1.380055, 1.390000, 2.820000, 0.030000,…
## $ calculated_host_listings_count <int> 1, 2, 2, 1, 1, 3, 4, 1, 1, 3, 1, 1, 4, …
## $ availability_365              <int> 0, 289, 221, 316, 189, 312, 258, 0, 0, …
```

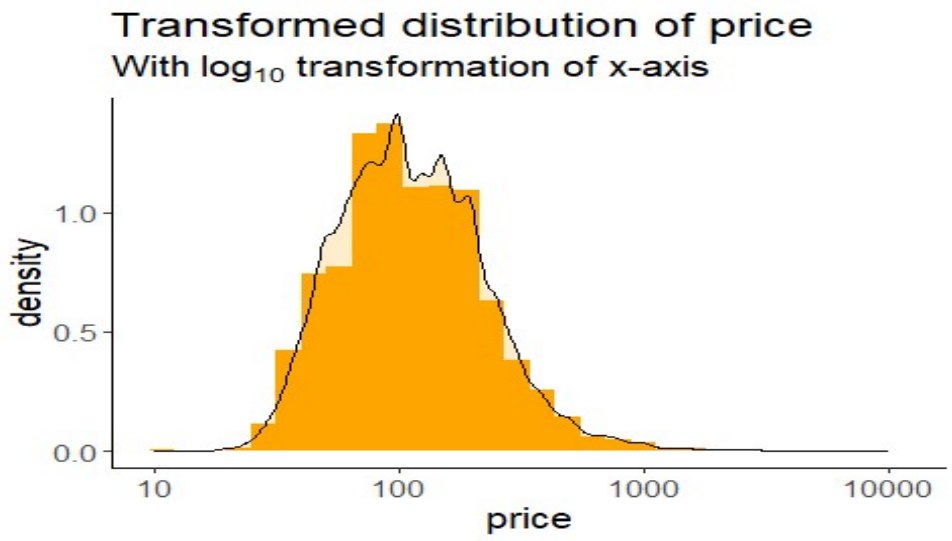
2.5 Data Normalization / Standarization

Data normalization or standardization is necessary to ensure accurate and fair analysis. Both normalization and standardization help in bringing numerical features to a comparable scale, preventing features with larger magnitudes from dominating the analysis.

```
#creating a theme for the plot
cleanup <- theme(panel.grid.major = element_blank(),
                 panel.grid.minor = element_blank(),
                 panel.background = element_blank(),
                 axis.line.x = element_line(color = 'black'),
                 axis.line.y = element_line(color = 'black'),
                 legend.key = element_rect(fill = 'white'),
                 text = element_text(size = 15))

#Plot distribution of price
par(mfrow=c(2,1))
ggplot(airbnb_dataset) +
  cleanup+
  geom_histogram(aes(price),fill = 'orange',alpha = 0.85,binwidth = 15) +
  theme_minimal(base_size = 13) + xlab("Price") + ylab("Frequency") +
  ggtitle("The Distrubition of Price")
```



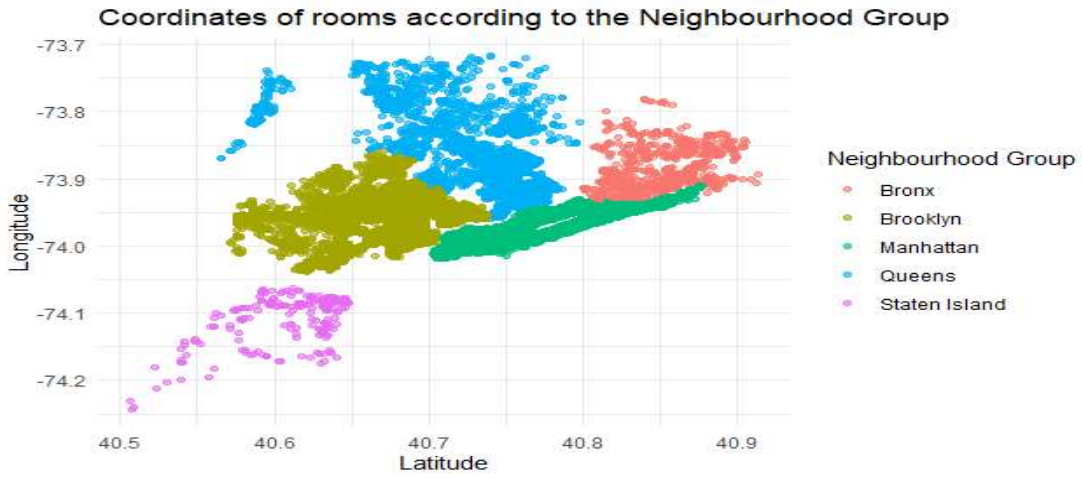

```
#Plot Transformed distribution of Price
ggplot(airbnb_dataset, aes(price)) +
  cleanup+
  geom_histogram(bins = 30, aes(y = ..density..), fill = "orange") +
  geom_density(alpha = 0.2, fill = "orange") +ggtitle("Transformed distribution of price",
  subtitle = expression("With" ~'log'[10] ~ "transformation of x-axis")) + scale_x_log10()
```



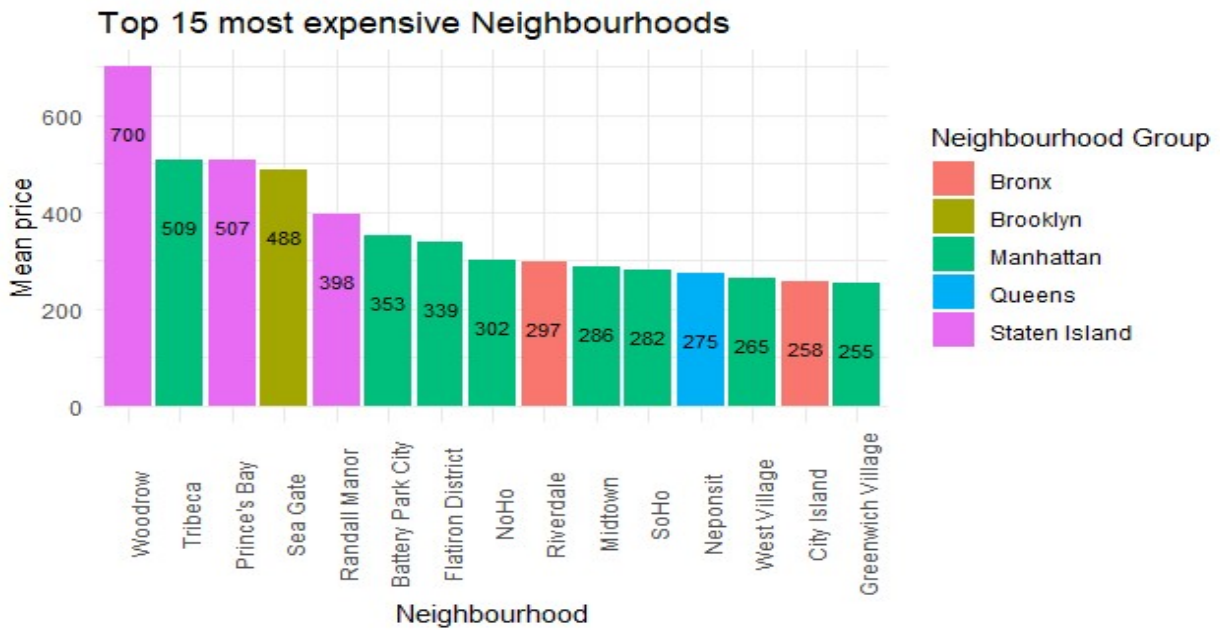
3. EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis employs statistical graphics, charts, and summary metrics to delve into the data, recognize patterns, anomalies, and correlations among variables. EDA is indispensable because it facilitates a profound comprehension of the dataset, yielding valuable insights.

```
#location of airbnb rooms using coordinates longitude and latitude
ggplot(airbnb_dataset, aes(latitude, longitude, color = neighbourhood_group)) +
 geom_point(alpha = 0.6) +
 theme_minimal() +
 labs(title = "Coordinates of Airbnb Rooms According to the Neighbourhood Group",
     subtitle = "2019 NYC Airbnb Data",
     x = "Latitude",
     y = "Longitude",
     color = "Neighbourhood Group")
```

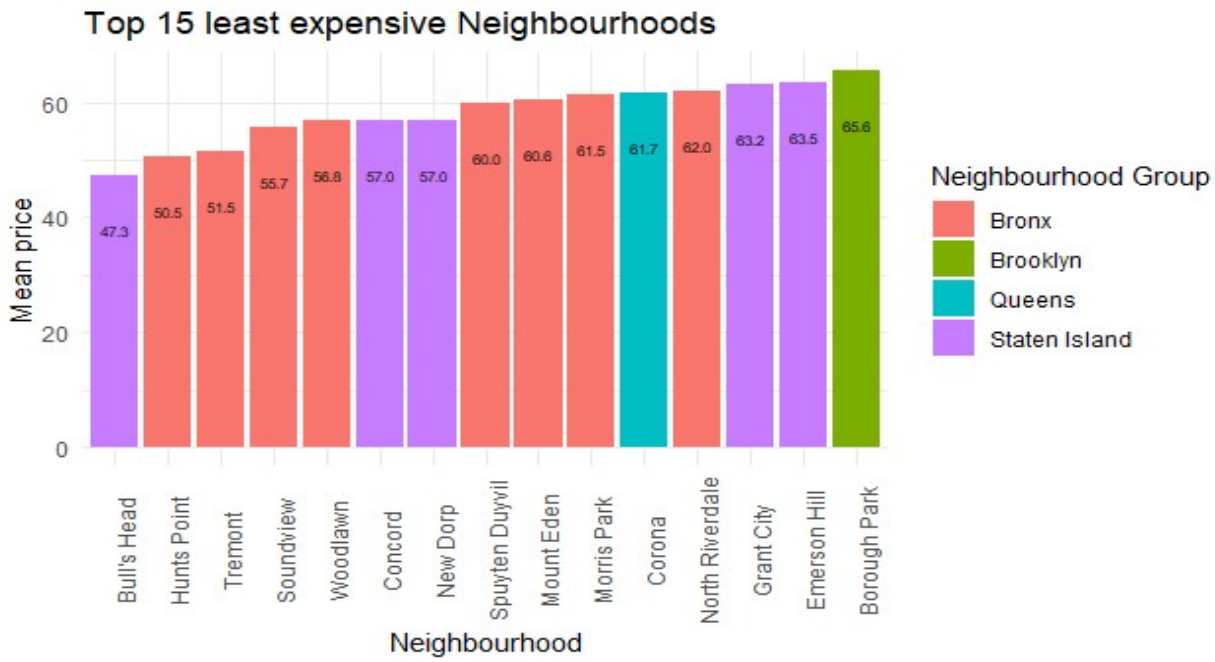


```r
#Plot most expensive neighbourhoods
airbnb_dataset %>%
 group_by(neighbourhood_group,neighbourhood)%>%
 summarise(mean_price = mean(price))%>%
 arrange(desc(mean_price))%>%
 head(15)%>%
 ggplot(., aes(x = reorder(neighbourhood, -mean_price) , y = mean_price, fill = neighbourhood_group))
+
 geom_col() +
 theme_minimal() +
 geom_text(aes(label = format(mean_price,digits=3)), size=3, position = position_dodge(0.9),vjust = 5)
+
 theme(axis.text.x = element_text(angle = 90), legend.position = "right") +
 labs(title = "Top 15 Most Expensive Neighbourhoods",
     subtitle ="2019 NYC Airbnb Data",
     x = "Neighbourhood",
     y = "Mean price",
     fill = "Neighbourhood Group")
## `summarise()` has grouped output by 'neighbourhood_group'. You can override
## using the `.groups` argument.
```

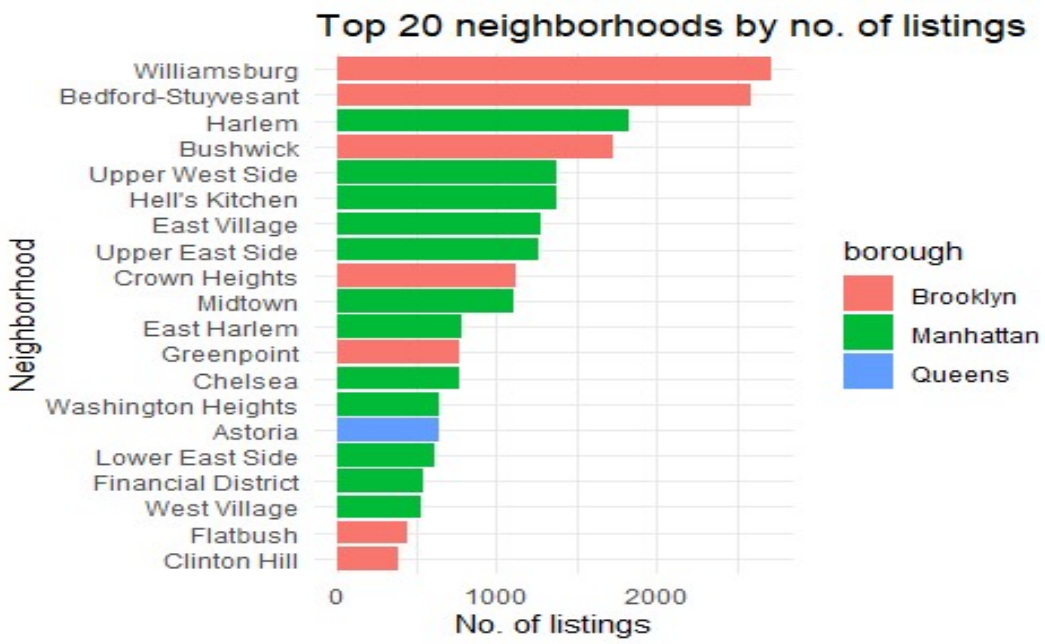



```r
#least expensive neighbourhood
airbnb_dataset %>%
 group_by(neighbourhood_group,neighbourhood)%>%
 summarise(mean_price = mean(price))%>%
 arrange(mean_price) %>%
 head(15)%>%
 ggplot(., aes(x = reorder(neighbourhood, mean_price) , y = mean_price, fill = neighbourhood_group)) +
 geom_col() +
```

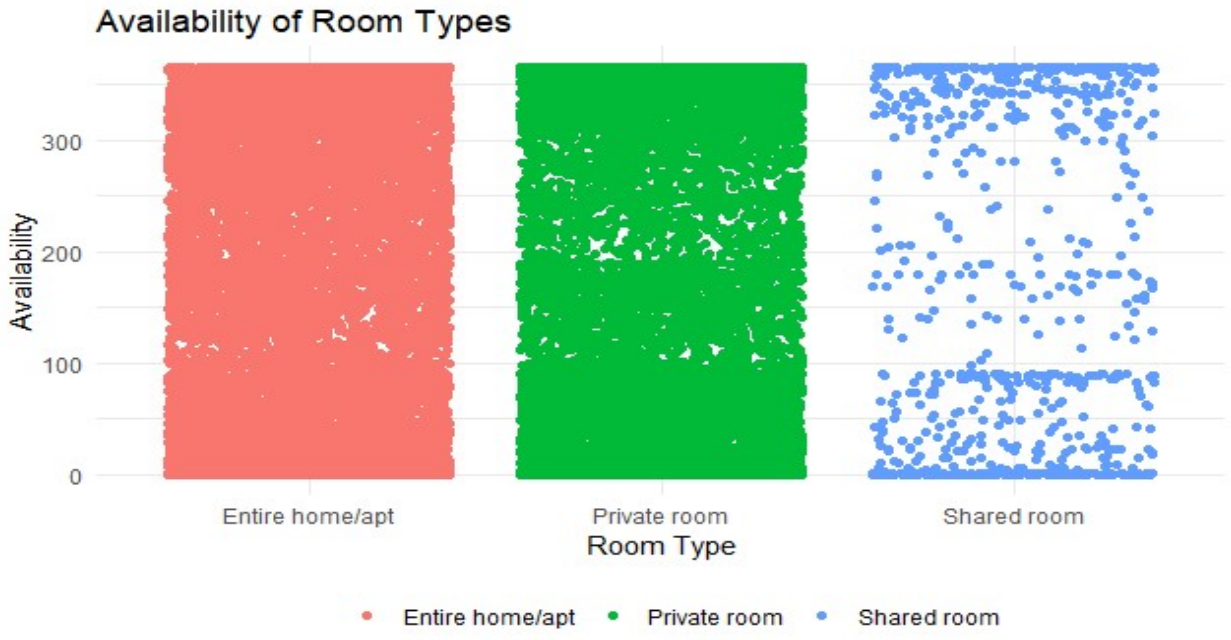
```
 theme_minimal() +
 geom_text(aes(label = format(mean_price,digits=3)), size=2, position = position_dodge(0.9),vjust = 5)
+
 theme(axis.text.x = element_text(angle = 90), legend.position = "right") +
 labs(title = "Top 15 Least Expensive Neighbourhoods",
     subtitle ="2019 NYC Airbnb Data",
     x = "Neighbourhood",
     y = "Mean price",
     fill = "Neighbourhood Group")
## `summarise()` has grouped output by 'neighbourhood_group'. You can override
## using the `.groups` argument.
```



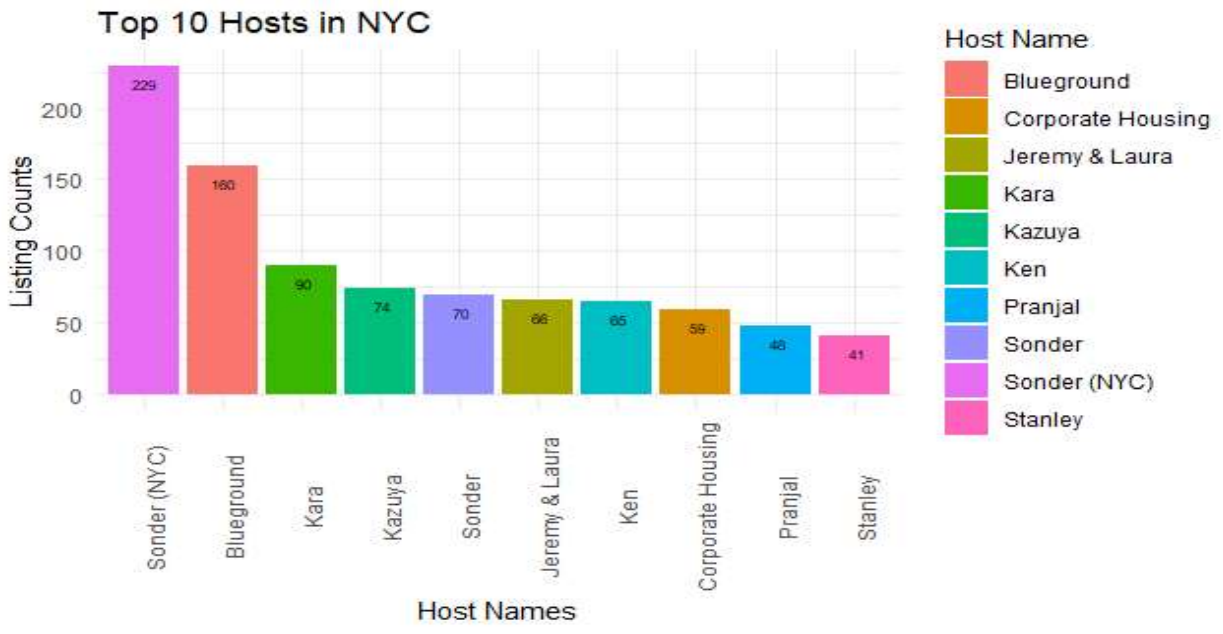

```
#Top20 neighbourhoods by listings
airbnb_dataset %>%
 group_by(neighbourhood) %>%
 dplyr::summarize(num_listings = n(),
         borough = unique(neighbourhood_group)) %>%
 top_n(n = 20, wt = num_listings) %>%
 ggplot(aes(x = fct_reorder(neighbourhood, num_listings),
          y = num_listings, fill = borough)) +
 geom_col() +
 coord_flip() +
 theme(legend.position = "bottom") +
 labs(title = "Top 20 neighborhoods by no. of listings",
     x = "Neighborhood", y = "No. of listings")+theme_minimal()
```




```
airbnb_dataset %>%
 ggplot(., aes(x = room_type, y = availability_365, color = room_type)) +
 geom_jitter() +
 theme_minimal() +
 theme(legend.position="bottom", plot.title = element_text(vjust = 0.5)) +
 labs(title = "Availability of Room Types",
     subtitle = "2019 NYC Airbnb Data",
     x = "Room Type",
     y = "Availability",
     color = " ")
```



```r
# Calculate the number of listings for each host and select the top 10 hosts
top_10_listing_counts = airbnb_dataset %>%
 group_by(host_id) %>%
 summarise(listing_count = n()) %>%
 arrange(desc(listing_count))

# Create a data frame with distinct host IDs and host names
id_name = distinct(airbnb_dataset[, c("host_id", "host_name")])

# Combine the top hosts with their names using a left join
top_10_listing_counts[1:10, ] %>%
 left_join(., id_name, by = "host_id") %>%

# Create a bar chart to visualize the top hosts and their listing counts
 ggplot(., aes(x = reorder(host_name, -listing_count) , y = listing_count, fill = host_name)) +
 geom_col() +
 theme_minimal() +
 geom_text(aes(label = format(listing_count,digits=3)), size=2, position = position_dodge(0.9),vjust = 2)
+
 theme(axis.text.x = element_text(angle = 90), legend.position = "right") +
 labs(title = "Top 10 Hosts in NYC",
     subtitle = "2019 NYC Airbnb Data",
     x = "Host Names",
     y = "Listing Counts",
     fill = "Host Name")
```

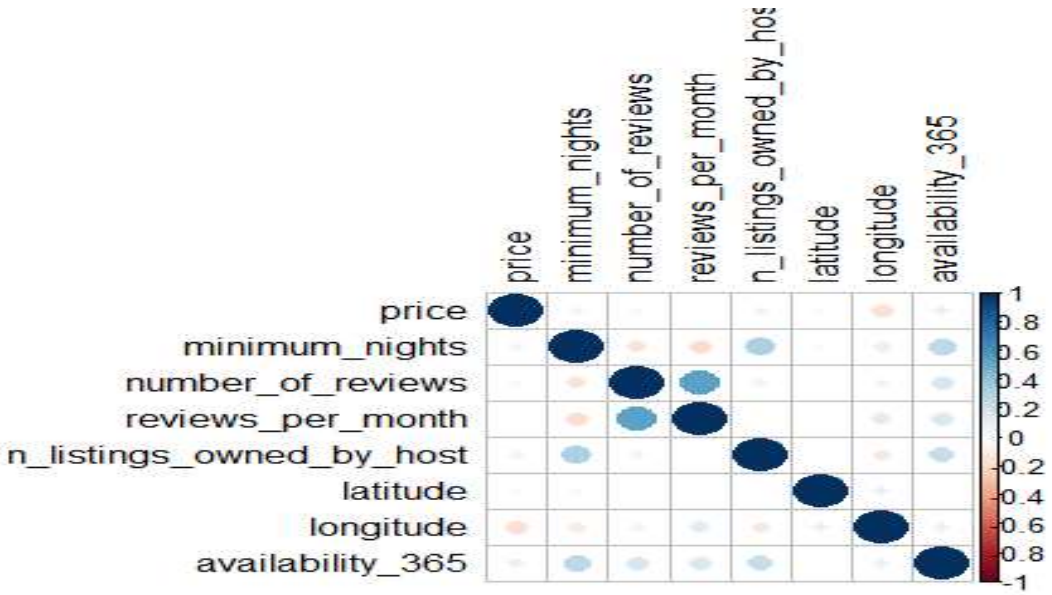


In summary, the EDA study highlights Manhattan as the most expensive neighborhood, contrasting with the least expensive, Bronx. It explores room type preferences, with entire home/apt being the most favored. The examination of number of reviews reveals Bronx and Staten Island have fewer reviews. Visualizations identify Fort Woodrow as the priciest neighborhood and Bull's Head as the most budget-friendly. Manhattan and Staten Island host the highest-priced rooms, while Bronx and Staten Island have the least expensive. Notably, no rooms from Manhattan are among the least expensive

4. CORRELATION ANALYSIS

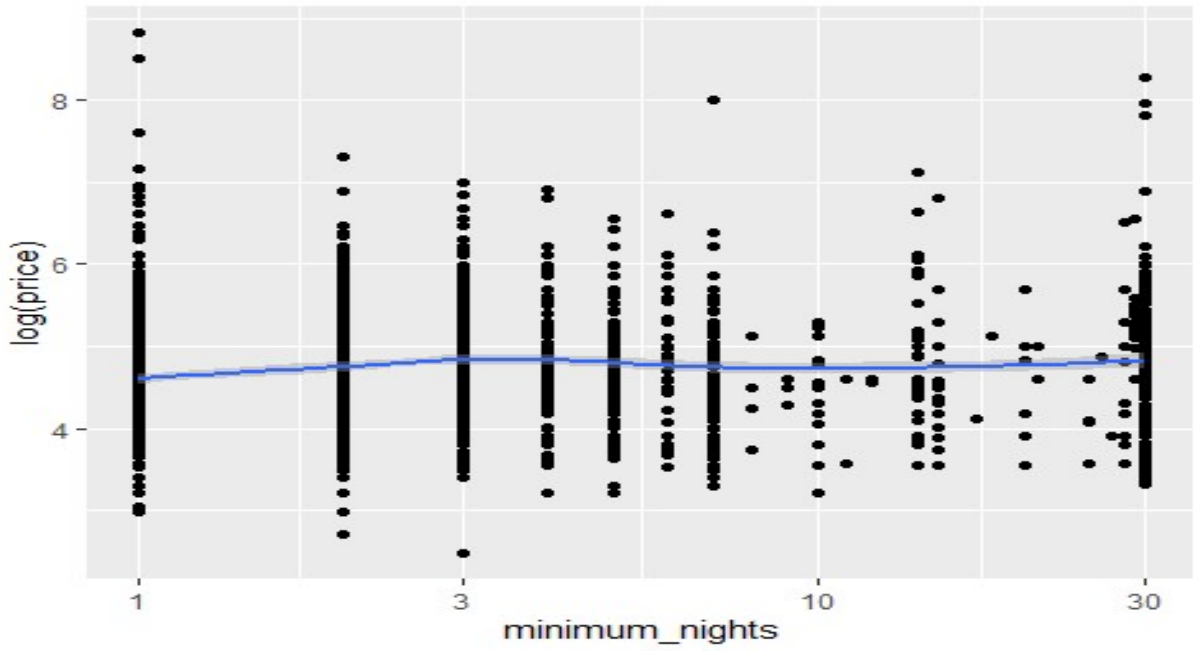
Correlation analysis is a crucial step in data analysis as it helps us to understand the relationship between two or more variables in a dataset. This understanding influence decisions in modeling and offers valuable insights for making well-informed choices.

```
pricing_correlation <- airbnb_dataset %>%
 select(price, minimum_nights, number_of_reviews, reviews_per_month,
calculated_host_listings_count, latitude, longitude, availability_365) %>%
 rename(n_listings_owned_by_host = calculated_host_listings_count)

pricing_corr_calc <- cor(pricing_correlation, use = "complete.obs")
pricing_corr_calc <- round(pricing_corr_calc, 2)
## corrplot
corrplot(pricing_corr_calc, tl.col = "black")
```



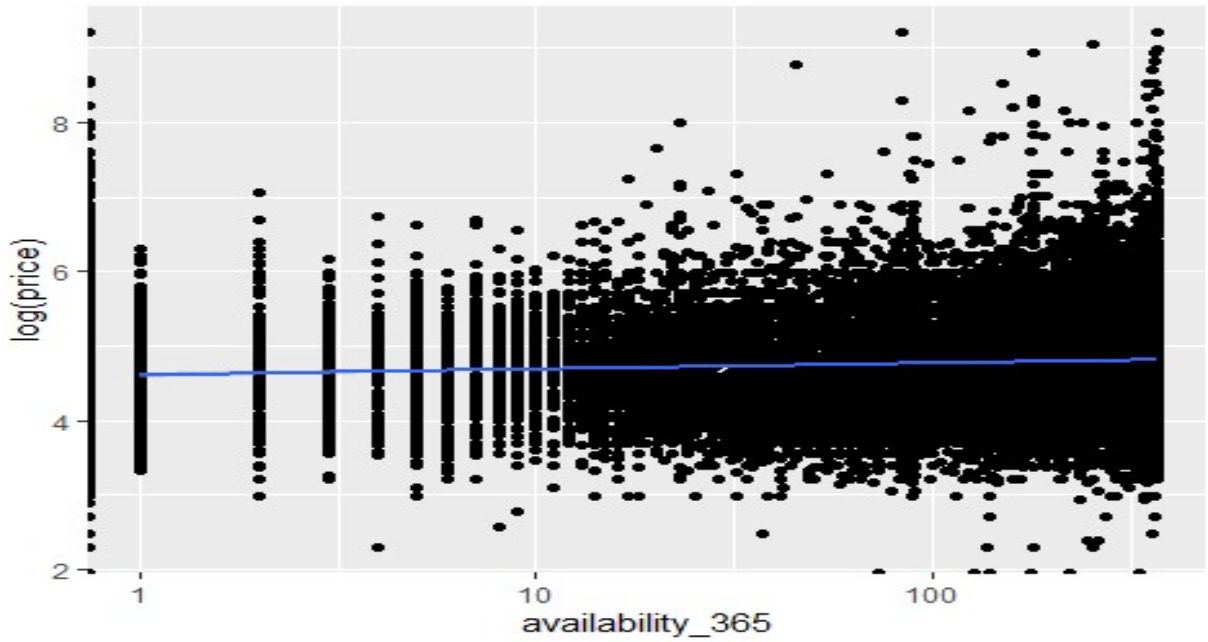

```
# Randomly sample 2000 rows from the 'airbnb_dataset'
sampled_data <- airbnb_dataset %>% sample_n(2000)

# Create a scatterplot
```

```r
scatterplot <- sampled_data %>% ggplot(aes(minimum_nights, log(price)))

# Add a logarithmic scale to the x-axis (minimum_nights)
scatterplot <- scatterplot + scale_x_log10()

# Add individual data points to the plot
scatterplot <- scatterplot + geom_point()

# Add a smoothed curve using the LOESS (Locally Weighted Scatterplot Smoothing) method
scatterplot <- scatterplot + geom_smooth(method = "loess")

# Display the final scatterplot
scatterplot
## `geom_smooth()` using formula = 'y ~ x'
```



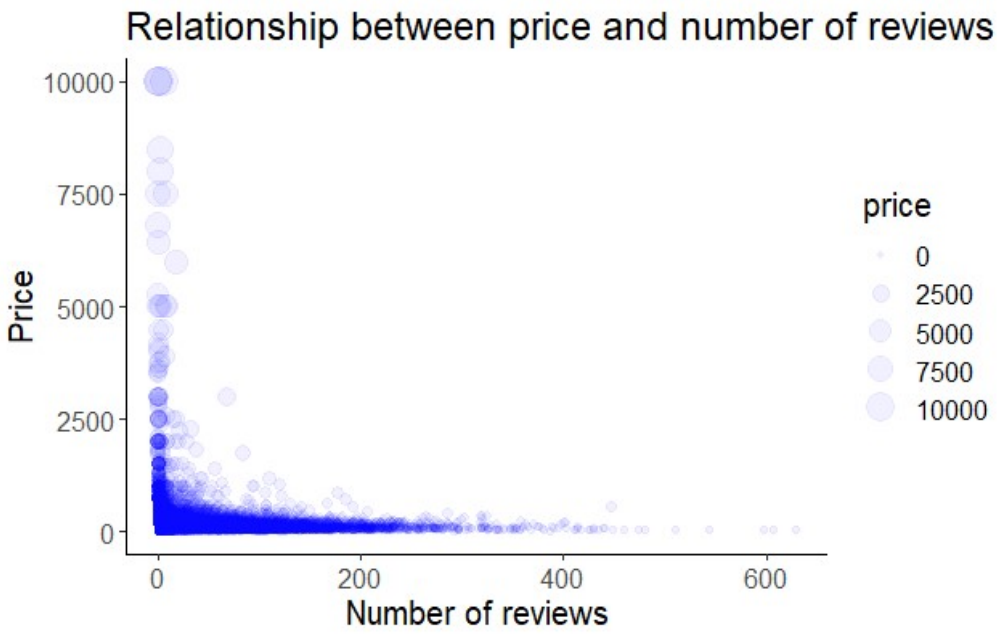

```r
# Create a scatterplot
scatterplot <- airbnb_dataset %>% ggplot(aes(availability_365, log(price)))

# Add a logarithmic scale to the x-axis (availability_365)
scatterplot <- scatterplot + scale_x_log10()

# Add individual data points to the plot
scatterplot <- scatterplot + geom_point()

# Add a linear regression line to visualize the relationship
scatterplot <- scatterplot + geom_smooth(method = "lm")
```

```
# Display the final scatterplot
scatterplot
## `geom_smooth()` using formula = 'y ~ x'
```

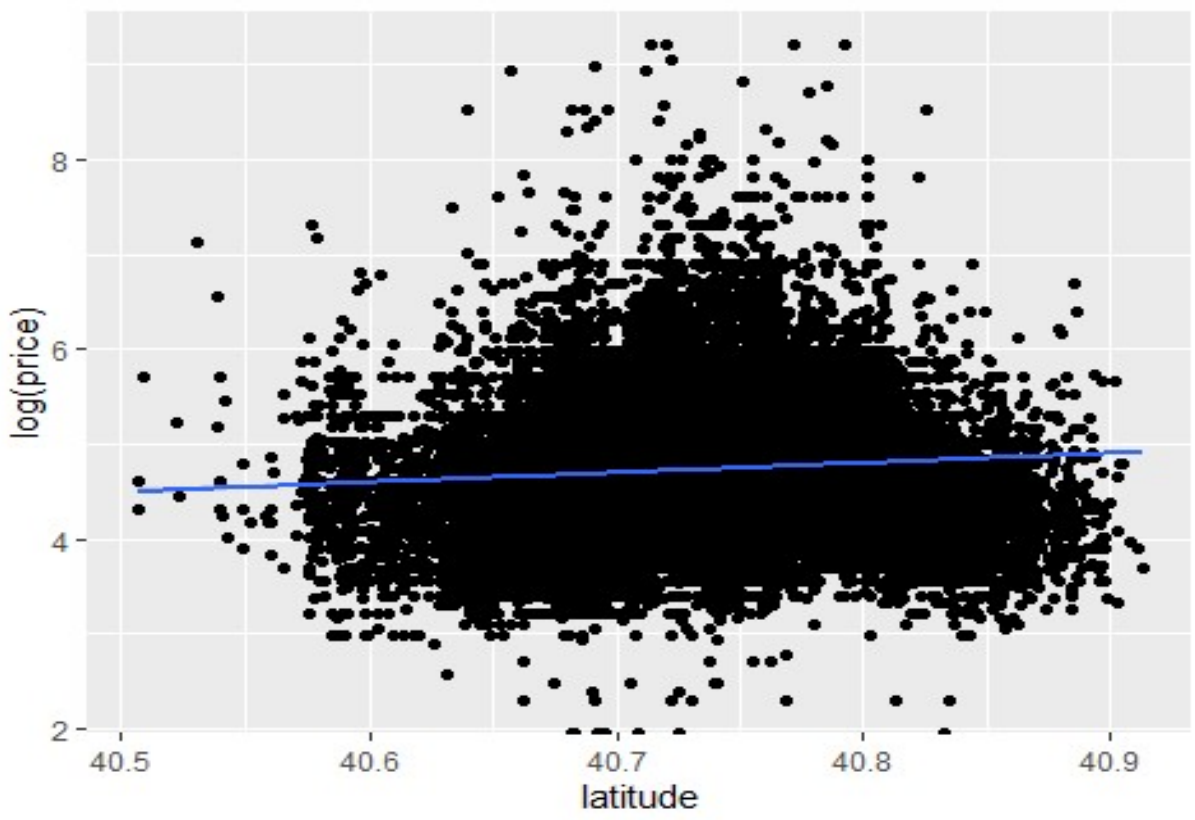



```
# Define a function named 'summarize_prices' that takes a data frame 'tbl' as input
summarize_prices <- function (tbl) {

 # Calculate and summarize statistics for the 'price' variable in the input data frame

 # Use the summarize function to compute summary statistics
 tbl %>%
  summarize(
   # Calculate and exponentiate the geometric mean of 'price' (avg_price)
   avg_price = exp(mean(price)),

   # Calculate and exponentiate the geometric median of 'price' (median_price)
   median_price = exp(median(price)),

   # Count the number of observations in the data frame (n)
   n = n()
  ) %>%

  # Arrange the results in descending order based on the count of observations (n)
  arrange(desc(n))
}
#summarize statistics for the 'price' variable in the 'airbnb_dataset' data frame for each unique host.
airbnb_dataset %>%
```

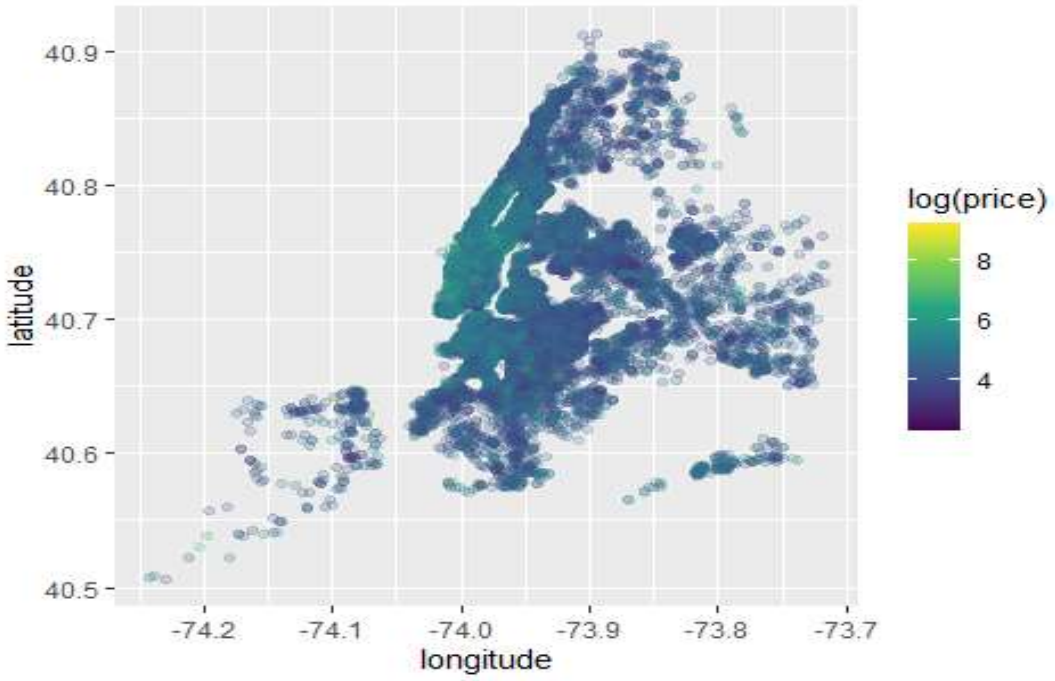
```
 group_by(host_id)%>%
 summarize_prices()
## # A tibble: 27,412 × 4
##      host_id avg_price median_price     n
##        <int>     <dbl>       <dbl> <int>
##  1 219517861  4.23e109     1.05e 99   229
##  2 107434423  3.02e133     2.88e132   160
##  3  30283594  2.81e120     6.26e103    90
##  4 137358866  6.63e 18     6.40e 17    74
##  5  12243051  6.10e 91     5.86e 90    70
##  6  16098958  5.18e 87     1.49e 78    66
##  7  22541573  2.21e 92     1.59e 91    65
##  8  61391963  1.19e 62     4.68e 61    59
##  9 200380610  1.73e127     1.68e109    48
## 10 120762452  2.20e 73     6.76e 73    41
## # ℹ 27,402 more rows


#Plot number of reviews and price to view the relation
ggplot(airbnb_dataset, aes(number_of_reviews, price)) +
 theme(axis.title = element_text(), axis.title.x = element_text()) +
 geom_point(aes(size = price), alpha = 0.05, color = "blue") +
 cleanup+
 xlab("Number of reviews") +
 ylab("Price") +
 ggtitle("Relationship between number of reviews")
```



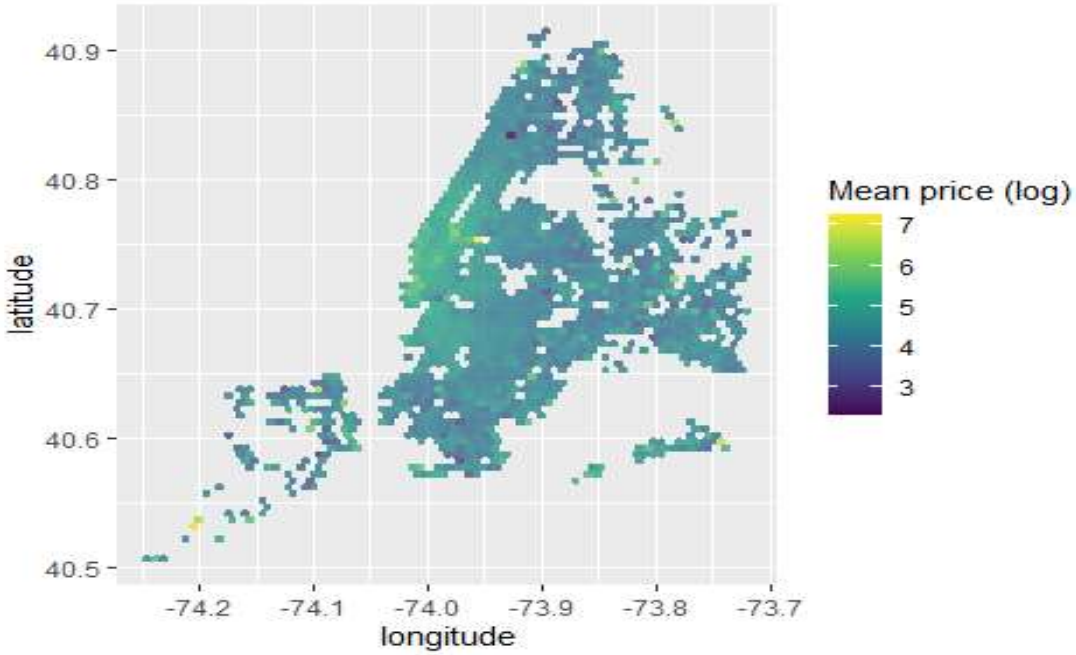
```r
# Create a scatterplot using ggplot2
scatterplot <- airbnb_dataset %>% ggplot(aes(latitude, log(price)))

# Add a logarithmic scale to the x-axis (latitude)
scatterplot <- scatterplot + scale_x_log10()

# Add individual data points to the plot
scatterplot <- scatterplot + geom_point()

# Add a linear regression line to visualize the relationship
scatterplot <- scatterplot + geom_smooth(method = "lm")

# Display the final scatterplot
scatterplot
## `geom_smooth()` using formula = 'y ~ x'
```



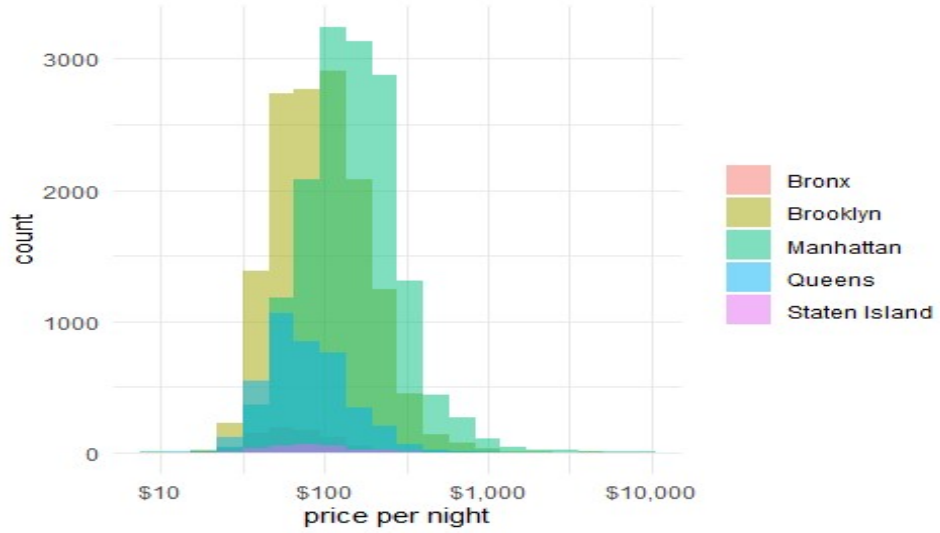

```r
#create a map showing the mean price in each area
airbnb_dataset %>%
 ggplot(aes(longitude, latitude, color = log(price))) +
 geom_point(alpha = 0.2) +
 scale_color_viridis_c()
```




```
#create a map with hex bins showing the mean price in each area.
airbnb_dataset %>%
 ggplot(aes(longitude, latitude, z = log(price))) +
 stat_summary_hex(alpha = 0.8, bins = 70) +
 scale_fill_viridis_c() +
 labs(fill = "Mean price (log)")
```

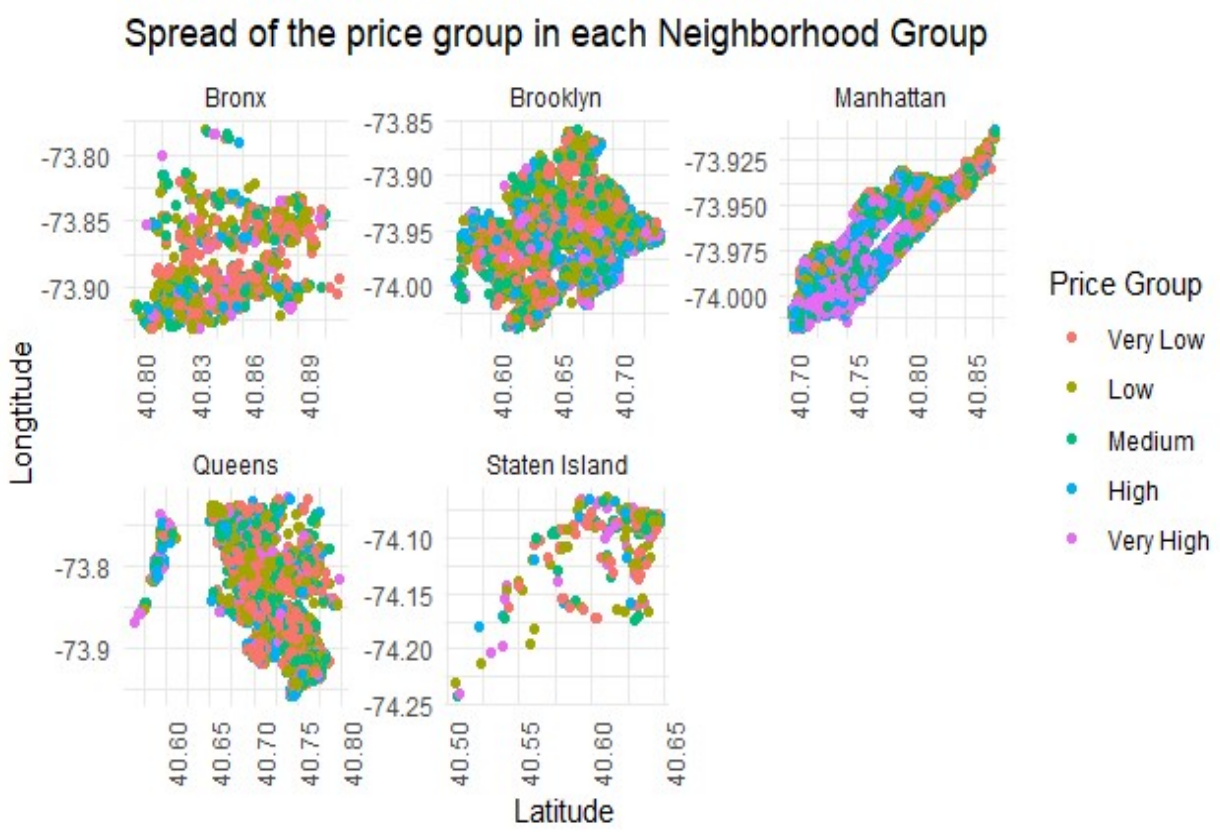


```r
#Plot a histogram to visualize the distribution of prices per night based on the neighborhood group

airbnb_dataset %>%
 ggplot(aes(price, fill = neighbourhood_group)) +
 geom_histogram(position = "identity", alpha = 0.5, bins = 20) +
 scale_x_log10(labels = scales::dollar_format()) +
 labs(fill = NULL, x = "price per night")+theme_minimal()
```

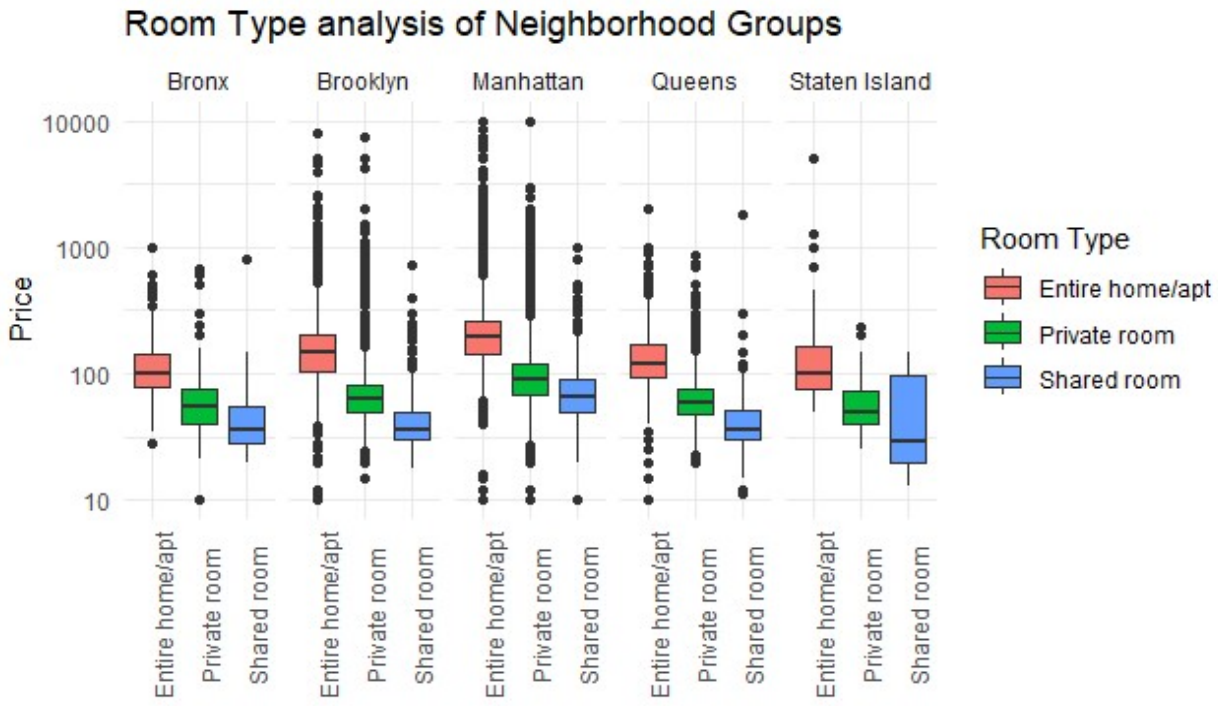



```r
# Calculate quantiles (percentiles) of the 'price' variable
quant = quantile(airbnb_dataset$price, seq(0, 1, 0.2))

# Create a new variable 'price_group' in the dataset based on quantiles
airbnb_price_group = airbnb_dataset %>%
 mutate(price_group = case_when(
   price < quant[2] ~ "Very Low",
   price < quant[3] ~ "Low",
   price < quant[4] ~ "Medium",
   price < quant[5] ~ "High",
   TRUE ~ "Very High"
 )) %>%
# Convert 'price_group' into a factor variable with predefined levels
 mutate(price_group = factor(price_group, levels = c("Very Low", "Low", "Medium", "High", "Very
High")))

# Create a scatterplot to visualize the price groups on a map
airbnb_price_group %>%
 ggplot(., aes(latitude, longitude, color = price_group)) +
 geom_point() +
 theme_minimal() +
 facet_wrap(~neighbourhood_group, scales = "free") +
 labs(title = "Spread of the Price Group In Each Neighborhood Group",
```

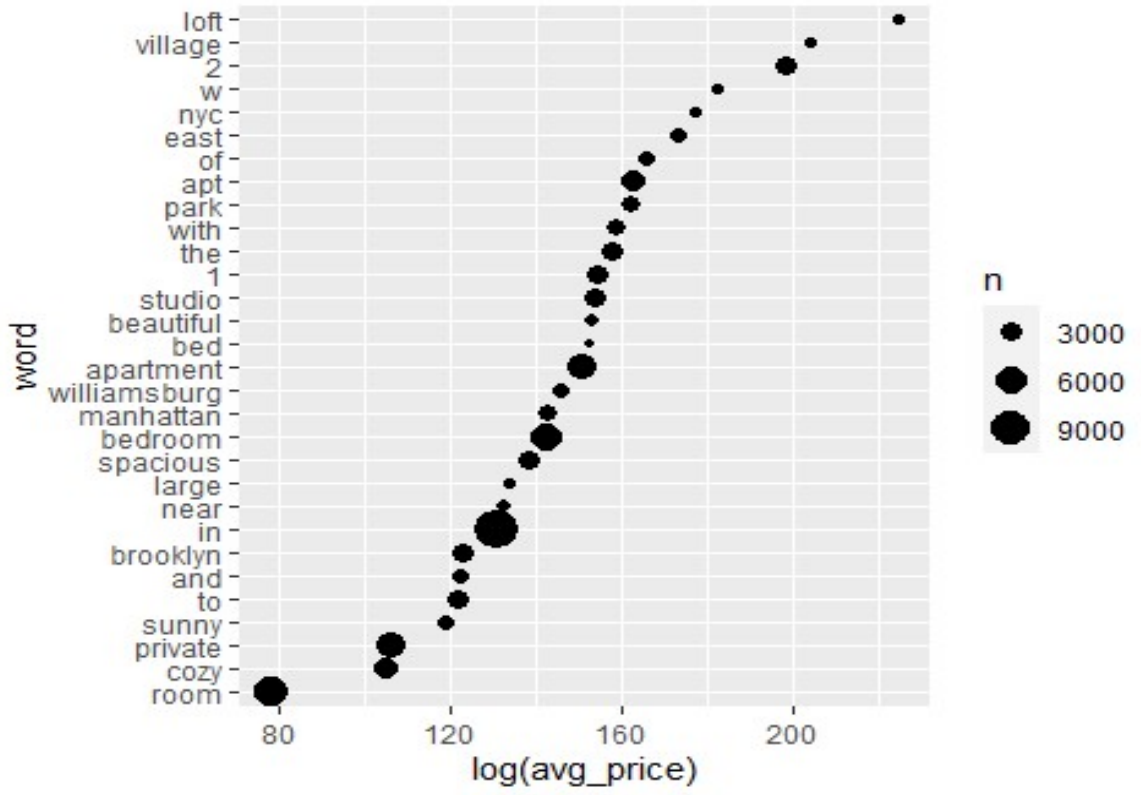
```
    subtitle = "2019 NYC Airbnb Data",
    x = "Latitude",
    y = "Longtitude",
    color = "Price Group")
```




```
# Create a series of box plots to analyze the distribution of prices for different room types in various neighborhood groups

ggplot(airbnb_dataset, aes(x = room_type, y = price, fill = room_type)) + scale_y_log10() +
  geom_boxplot() +
  theme_minimal() +
  labs (x="", y= "Price") +
  facet_wrap(~neighbourhood_group) +
  facet_grid(.~ neighbourhood_group) +
  theme(axis.text.x = element_text(angle = 90), legend.position = "right") +
  labs(title = "Room Type Analysis of Neighborhood Groups",
       subtitle = "2019 NYC Airbnb Data",
       fill = "Room Type")
```




```
# Tokenize the 'name' column of 'airbnb_dataset' into individual words
# and create a new column named 'word' to store these words
tokenized_data <- airbnb_dataset %>% unnest_tokens(word, name)

# Group the data by the 'word' column
grouped_data <- tokenized_data %>% group_by(word)

# Calculate and summarize statistics for each word, potentially related to average prices
summarized_data <- summarize_prices(grouped_data)

# Select the top 30 words with the highest counts
top_30_words <- head(summarized_data, 30)

# Reorder the 'word' factor based on the logarithm of average prices
top_30_words <- top_30_words %>%
  mutate(word = fct_reorder(word, log(avg_price)))

# Create a scatterplot to visualize the relationship between the logarithm of average prices,
# word frequencies, and the size of points representing word counts
word_frequency_plot <- ggplot(top_30_words, aes(log(avg_price), word, size = n)) + geom_point()

word_frequency_plot
```



The correlation study shows a slight positive correlation between price and minimum nights, while indicates a very slight or flat correlation availability 365 . A plot highlights a negative correlation between expensive objects and number of reviews. Geographically, Manhattan exhibits concentrated high prices, contrasting with more evenly spread prices in Bronx, Brooklyn, Queens, and Staten Island. Overall, price shows a clear connection to geography. Also, it reveals a relation between words and price, indicating a word's impact on pricing.

5. FEATURE SELECTION

This thorough correlation and exploratory data analysis (EDA) indicate a strong association between price and geographic factors, prompting the inclusion of geography-related features such as room type, neighborhood, longitude, and latitude. The recognition of this connection suggests that these variables significantly contribute to the pricing dynamics and are essential for a comprehensive model.

Additionally, while variables like minimum nights and number of reviews exhibit a relatively subtle relationship with price, their inclusion is deemed valuable. Despite the modest correlation, these variables might contribute meaningfully to the overall predictive power of the model, enhancing its accuracy and capturing nuanced patterns in the data. Furthermore, the analysis underscores the impact of linguistic aspects in variable names on price, suggesting a

unique dimension for model development. In summary, the chosen variables are justified based on their demonstrated correlations with price.

6. MODELING

6.1 Data Preparation

```
# Set a seed for reproducibility (optional)
# It ensures consistency in the randomization processes for cross-validation
set.seed(123)

# Perform an initial data split into training and testing sets
nyc_split <- airbnb_dataset %>%
  mutate(price = log(price + 1)) %>%
  initial_split(strata = price)

# Extract the training set from the split
nyc_train <- training(nyc_split)

# Extract the testing set from the split
nyc_test <- testing(nyc_split)

# Set a new seed for reproducibility
set.seed(234)

# Create cross-validation folds for model assessment
nyc_folds <- vfold_cv(nyc_train, v = 5, strata = price)

# Display information about the cross-validation folds
nyc_folds
## #  5-fold cross-validation using stratification
## # A tibble: 5 × 2
##   splits            id
##   <list>            <chr>
## 1 <split [20519/5132]> Fold1
## 2 <split [20519/5132]> Fold2
## 3 <split [20521/5130]> Fold3
## 4 <split [20522/5129]> Fold4
## 5 <split [20523/5128]> Fold5
```

The code snippet "nyc_folds <- vfold_cv(nyc_train, v = 5, strata = price)" creates a set of five cross-validation folds to assess the performance of machine learning models using the training data. Cross-validation is a crucial technique for gauging a model's ability to generalize to new data, preventing overfitting, optimizing hyperparameters, and ensuring reliable model assessment. The 'strata = price' inclusion in the code ensures that each fold maintains a similar distribution of the 'price' variable, particularly beneficial when dealing with datasets having

uneven target variable distributions. These cross-validation folds enhance the accuracy of performance estimation by subjecting the model to testing on multiple training data subsets.

6.2 Modeling Approach

6.2.1 Bagged Decision Tree Model

Bagging, short for Bootstrap Aggregating, involves training multiple decision tree models on different subsets of the training data created through bootstrapping (sampling with replacement).

6.2.1.1 Model Building

```
#creating a data pre-processing recipe using 'tidymodels' framework

nyc_rec <-
 recipe(price ~ latitude + longitude + neighbourhood + room_type +
   minimum_nights + number_of_reviews + availability_365 + name,
 data = nyc_train
 ) %>%
 step_novel(neighbourhood) %>%
 step_other(neighbourhood, threshold = 0.01) %>%
 step_tokenize(name) %>%
 step_stopwords(name) %>%
 step_tokenfilter(name, max_tokens = 30) %>%
 step_tf(name)

nyc_rec
##
## ── Recipe ──────────────────────────────────────────────────────────────────────
##
## ── Inputs
## Number of variables by role
## outcome:   1
## predictor: 8
##
## ── Operations
## • Novel factor level assignment for: neighbourhood
## • Collapsing factor levels for: neighbourhood
## • Tokenization for: name
## • Stop word removal for: name
## • Text filtering for: name
## • Term frequency with: name
```

The data preprocessing steps are crucial for machine learning, specifically in predicting the 'price' variable. The 'step_novel' addresses novel levels in 'neighbourhood,' ensuring the model

gracefully handles new values. 'step_other' groups infrequent 'neighbourhood' levels, reducing dimensionality to enhance model efficiency and prevent overfitting. Tokenizing and removing stopwords in the 'name' variable with 'step_tokenize' and 'step_stopwords' aid in processing text data, reducing noise, and improving model performance. Additionally, 'step_tokenfilter' limits tokens in 'name' to manage dimensionality, while 'step_tf' applies term frequency transformation, fundamental for natural language processing. In summary, these preprocessing steps address data quality, prepare text data, and reduce dimensionality, collectively enhancing machine learning model performance and interpretability.

```
# Set up the bagged decision tree model specification
bag_spec <-
  bag_tree(min_n = 10) %>%
  set_engine("rpart", times = 25) %>%
  set_mode("regression")

# Create a workflow that combines the model specification with data pre-processing
bag_wf <-
  workflow() %>%
  add_recipe(nyc_rec) %>%
  add_model(bag_spec)

# Set a seed for reproducibility
set.seed(123)

# Fit the bagged decision tree model to the training data
bag_fit <- fit(bag_wf, data = nyc_train)

# Display information about the fitted bagged model
bag_fit
## ══ Workflow [trained] ══════════════════════════════════════════════════════
## Preprocessor: Recipe
## Model: bag_tree()
##
## ── Preprocessor ────────────────────────────────────────────────────────────
## 6 Recipe Steps
##
## • step_novel()
## • step_other()
## • step_tokenize()
## • step_stopwords()
## • step_tokenfilter()
## • step_tf()
##
## ── Model ───────────────────────────────────────────────────────────────────
## Bagged CART (regression with 25 members)
```

```
##
##
Variable importance scores include
##
## # A tibble: 37 × 4
##   term           value std.error  used
##   <chr>          <dbl>     <dbl> <int>
## 1 room_type       4841.    15.7     25
## 2 longitude       3012.    16.8     25
## 3 neighbourhood   2534.    13.4     25
## 4 tf_name_room    2071.     5.39    25
## 5 latitude        1796.    10.8     25
## 6 minimum_nights  1533.    10.2     25
## 7 availability_365 1107.   13.0     25
## 8 tf_name_private  997.     7.77    25
## 9 number_of_reviews 914.    10.2     25
## 10 tf_name_studio  182.     2.52    25
## # ℹ 27 more rows
```

Feature Importance

```
## # A tibble: 37 × 4
##   term           value std.error  used
##   <chr>          <dbl>     <dbl> <int>
## 1 room_type       4841.    15.7     25
## 2 longitude       3012.    16.8     25
## 3 neighbourhood   2534.    13.4     25
## 4 tf_name_room    2071.     5.39    25
## 5 latitude        1796.    10.8     25
## 6 minimum_nights  1533.    10.2     25
## 7 availability_365 1107.   13.0     25
## 8 tf_name_private  997.     7.77    25
## 9 number_of_reviews 914.    10.2     25
## 10 tf_name_studio  182.     2.52    25
## # ℹ 27 more rows
```

6.2.1.1 Model Evaluation (Custom Centric)

```
# Register the 'doParallel' package for parallel processing
doParallel::registerDoParallel(cores=4)

# Set a seed for reproducibility
set.seed(123)

# Perform model resampling on the bagged decision tree model
```

```
bag_rs <- fit_resamples(bag_wf, nyc_folds)

# Collect and summarize evaluation metrics for each resample
collect_metrics(bag_rs)
## # A tibble: 2 × 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   0.438     5 0.00656 Preprocessor1_Model1
## 2 rsq     standard   0.600     5 0.00842 Preprocessor1_Model1
# Apply the above fitted model to the test data
test_rs <- augment(bag_fit, nyc_test)
#compare the true price values to the predicted price values from the above bagged decision tree model

test_rs %>%
 ggplot(aes(exp(price), exp(.pred), color = neighbourhood_group)) +
 geom_abline(slope = 1, lty = 2, color = "black", alpha = 1) +
 geom_point(alpha = 0.2) +
 scale_x_log10(labels = scales::dollar_format()) +
 scale_y_log10(labels = scales::dollar_format()) +
 labs(color = NULL, x = "True price", y = "Predicted price") +theme_light()
```


```
#Calculate the Root Mean Square Error (RMSE) for the 'price' variable
test_rs %>%
 rmse(price, .pred)
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.432
```

6.2.2 XGBoost Model

XGBoost, or eXtreme Gradient Boosting, is an ensemble learning method that combines the power of gradient boosting with regularization techniques.

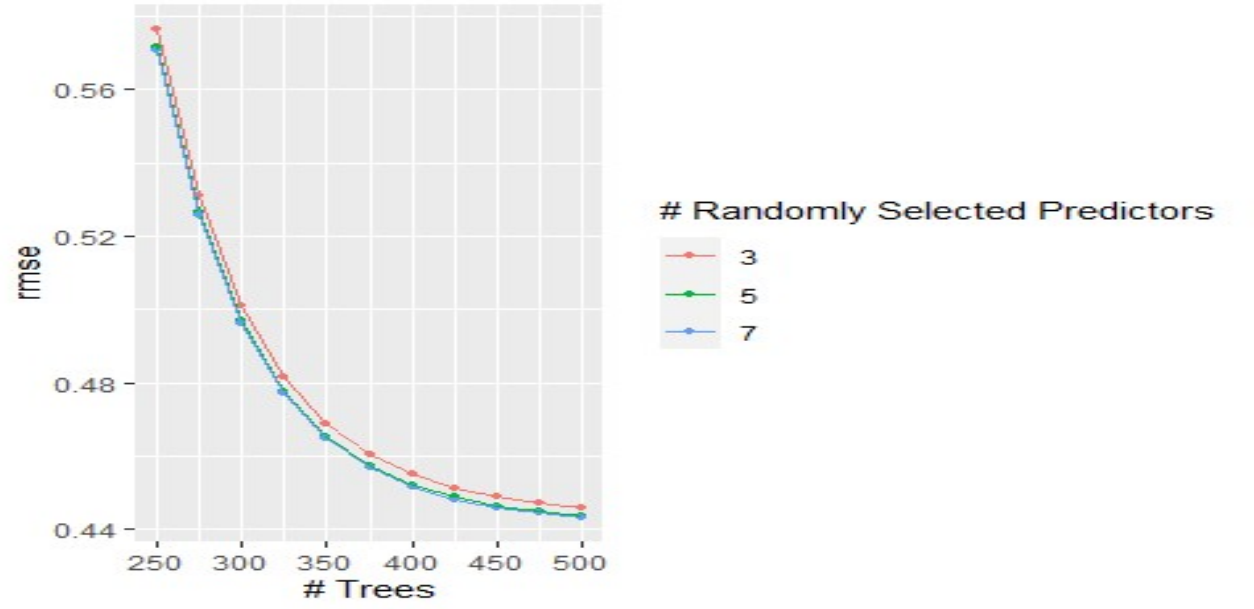
6.2.2.1 Model Building

```
#set the metric

mset <-metric_set(rmse)

grid_control <- control_grid(save_pred = TRUE,
                     save_workflow=TRUE,
                     extract = extract_model)

set.seed(2021)
prep_juice <- function(d) juice (prep(d))

xg_rec <- recipe (price ~ minimum_nights + room_type + number_of_reviews + longitude +latitude +
neighbourhood_group +reviews_per_month + calculated_host_listings_count +availability_365 +
last_review, data = nyc_train) %>%
 #step_log(all_numeric_predictors(), offset=1) %>%
 step_mutate(last_review = coalesce(as.integer(Sys.Date() - as.Date(last_review)), 0))%>%
 step_dummy(all_nominal_predictors())

xg_mod <- boost_tree ("regression",
                mtry = tune(),
                trees=tune(),
                learn_rate=0.01) %>%
 set_engine("xgboost")

xg_wf<- workflow() %>%
 add_recipe(xg_rec)%>%
 add_model(xg_mod)

xg_tune <- xg_wf%>%
 tune_grid(nyc_folds,
        metrics=mset,
        control=grid_control,
        grid=crossing(mtry=c(3,5,7), trees=seq(250, 500, 25)),
        workers = 1)
```

The step_log function, when activated, logarithmically transforms numeric predictors—common for log-scale relationships. The step_mutate function computes days between the current date

and last_review, replacing missing values with 0. This is crucial as it converts a date-related variable into a numeric feature, capturing the recency of the last review. Lastly, step_dummy generates dummy variables for nominal predictors, essential for machine learning algorithms like XGBoost, which demand numerical input from categorical data.

```
autoplot(xg_tune)
```



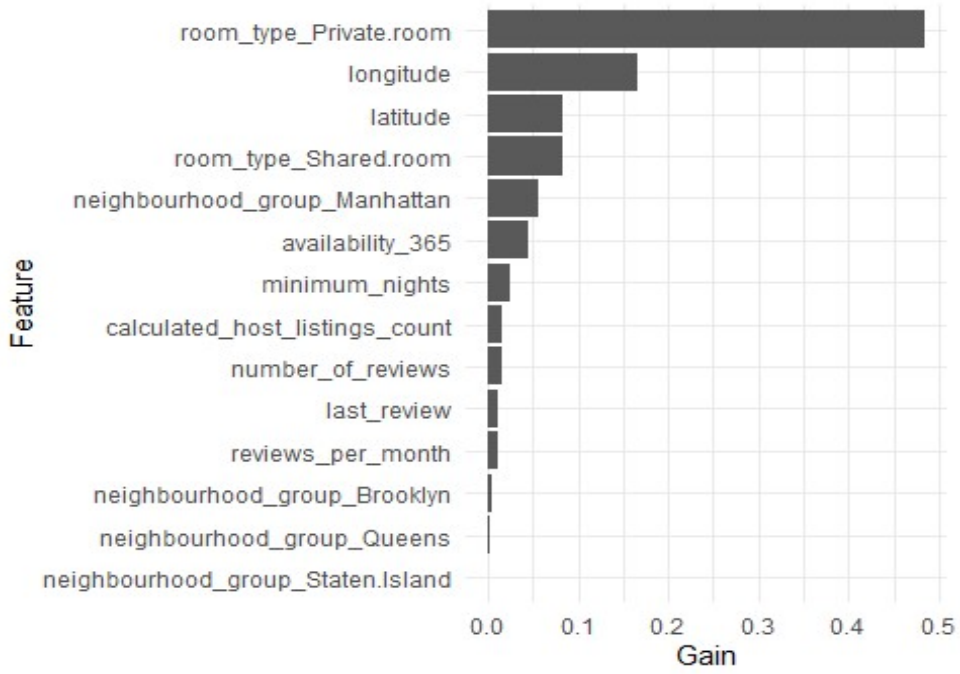

```
xg_tune %>%
  collect_metrics()%>%
  arrange(mean)
## # A tibble: 33 × 8
##    mtry trees .metric .estimator  mean     n std_err .config
##   <dbl> <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     7   500 rmse    standard   0.443     5 0.00544 Preprocessor1_Model33
## 2     5   500 rmse    standard   0.444     5 0.00535 Preprocessor1_Model22
## 3     7   475 rmse    standard   0.444     5 0.00544 Preprocessor1_Model32
## 4     5   475 rmse    standard   0.445     5 0.00533 Preprocessor1_Model21
## 5     3   500 rmse    standard   0.446     5 0.00532 Preprocessor1_Model11
## 6     7   450 rmse    standard   0.446     5 0.00543 Preprocessor1_Model31
## 7     5   450 rmse    standard   0.446     5 0.00534 Preprocessor1_Model20
## 8     3   475 rmse    standard   0.447     5 0.00530 Preprocessor1_Model10
## 9     7   425 rmse    standard   0.448     5 0.00542 Preprocessor1_Model30
## 10    5   425 rmse    standard   0.449     5 0.00532 Preprocessor1_Model19
## # ℹ 23 more rows
```

6.2.2.2 Model Evaluation

```
xg_fit <- xg_wf %>% finalize_workflow(select_best(xg_tune))%>%
  fit(nyc_train)
```

```
xg_fit %>%
 augment(nyc_test)%>%
 rmse(price,.pred)
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.445
doParallel::registerDoParallel(cores=4)

set.seed(123)
xg_rs <- fit_resamples(xg_fit, nyc_folds)
collect_metrics(xg_rs)
## # A tibble: 2 × 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   0.443     5 0.00541 Preprocessor1_Model1
## 2 rsq     standard   0.593     5 0.00675 Preprocessor1_Model1
importances <- xgboost::xgb.importance(model=xg_fit$fit$fit$fit)

importances %>%
 mutate(Feature = fct_reorder(Feature, Gain))%>%
 ggplot(aes(Gain, Feature))+
 geom_col()+theme_minimal()
```

Feature importance plot



6.3 Comparison of Models

```
# Collect and summarize evaluation metrics for the bagged decision tree model
bag_metrics <- collect_metrics(bag_rs, metrics = c("mae", "mse", "rmse", "rsquared"))

# Collect and summarize evaluation metrics for the xgboost model
xg_metrics <- collect_metrics(xg_rs, metrics = c("mae", "mse", "rmse", "rsquared"))

# Display metrics for each model
print("Bagged Decision Tree Metrics:")
```

```
## [1] "Bagged Decision Tree Metrics:"
```

```
print(bag_metrics)
```

```
## # A tibble: 2 × 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   0.438     5 0.00656 Preprocessor1_Model1
## 2 rsq     standard   0.600     5 0.00842 Preprocessor1_Model1
```

```
print("XGBoost Metrics:")
```

```
## [1] "XGBoost Metrics:"
```

```
print(xg_metrics)
```

```
## # A tibble: 2 × 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   0.443     5 0.00541 Preprocessor1_Model1
## 2 rsq     standard   0.593     5 0.00675 Preprocessor1_Model1
```

In comparing the Bagged Decision Tree and XGBoost models, we examined two key metrics: Root Mean Squared Error (RMSE) and R-squared (R^2). The Bagged Decision Tree model demonstrated slightly better performance, with a lower RMSE indicating more accurate predictions and a higher R-squared suggesting a better fit to the data. Also, from feature importance we can say that the room type is the most contributing feature followed by geographical feature including latitude, longitude and neighbourhood.

However, it's important to note that the differences in performance, though present, are not substantial. Additional considerations, such as the interpretability of the models, their computational efficiency, and specific requirements of our use case, should guide our final model selection. While the metrics provide valuable insights into predictive accuracy and model fit, a comprehensive evaluation takes into account a range of factors for a well-informed decision.

7. Conclusion

In conclusion, the study thoroughly examined Airbnb's NYC dataset revealing important trends. Expanding on this, the study crafted predictive models, utilizing XGBoost and Bagged Decision Tree methods, to improve rental price forecasts. This not only deepens the understanding of NYC's Airbnb dynamics but also provides stakeholders with potent tools for anticipating and optimizing pricing strategies. The amalgamation of detailed analysis and advanced models places our research at the forefront of informed decision-making in the ever-evolving landscape of Airbnb hosting and accommodations.