Semester Project Write-Up

Team: Charlie's Angels

Taha Nadeem Qureshi

AD 699- Data Mining

Professor Page

December 12, 2023

Step I: Data Preparation & Exploration

I. Missing Values

In handling missing values, our approach was guided by the nature of the data and the impact of missingness on the analysis. For categorical variables, we used mode imputation to fill in missing values, as this method preserves the distribution of the data.

This was particularly appropriate for variables like host_location and host_is_superhost, where a predominant category could represent the missing data without significantly skewing the analysis.

```
#host location

table(neighbourhood_data1$host_location)

# Impute missing values with the mode
mode_host_location <- names(sort(table(neighbourhood_data1$host_location), decreasing = TRUE))[1]
neighbourhood_data1$host_location[is.na(neighbourhood_data1$host_location)] <- mode_host_location
table(neighbourhood_data1$host_location)
```

For numerical variables, especially those with placeholder strings like 'N/A', we first converted these columns to numeric types, accounting for non-numeric characters, and then imputed missing values using the mean. This method was suitable for host_response_rate and host_acceptance_rate, where an average value provided a reasonable estimate for missing entries.

```
table(neighbourhood_data1$host_response_rate)

#Here, response_rate represents a numerical column but there are 207 'N/A' strings. We would have to change it to a numeric value. And taking average

neighbourhood_data1$host_response_rate[neighbourhood_data1$host_response_rate == 'N/A'] <- NA
neighbourhood_data1$host_response_rate <- as.numeric(gsub("[^0-9.]", "", neighbourhood_data1$host_response_rate))
average_response_rate <- mean(neighbourhood_data1$host_response_rate, na.rm = TRUE)
neighbourhood_data1$host_response_rate[is.na(neighbourhood_data1$host_response_rate) | is.na(average_response_rate)] <- average_response_rate

table(neighbourhood_data1$host_acceptance_rate)

#141 N/A strings

neighbourhood_data1$host_acceptance_rate[neighbourhood_data1$host_acceptance_rate == 'N/A'] <- NA
neighbourhood_data1$host_acceptance_rate <- as.numeric(gsub("[^0-9.]", "", neighbourhood_data1$host_acceptance_rate))
average_acceptance_rate <- mean(neighbourhood_data1$host_acceptance_rate, na.rm = TRUE)
neighbourhood_data1$host_acceptance_rate[is.na(neighbourhood_data1$host_acceptance_rate) | is.na(average_acceptance_rate)] <- average_acceptance_rate
```

To go through a few cases, in the process of analyzing missing values and variables, several considerations were addressed. Firstly, within the 'host location' variable, there were 400 missing values,

which were imputed using the mode as a replacement. Moving on to 'host_response_time,' where 207 instances were labeled as N/A, it was determined that N/A here might signify a category where response time is not applicable, and thus, these values were retained.

Regarding the 'host_response_rate' column, 207 occurrences had the value 'N/A' in string format. To enhance analysis, the column was converted to a numerical format. The average of all valid response rates was then computed and imputed for instances where 'N/A' was present. Similarly, in the 'host_acceptance_rate' column, 141 N/A strings were replaced with the average acceptance rate, ensuring a consistent numerical representation for subsequent analysis.

For the 'host_is_superhost' variable, missing values were imputed with the mode under the assumption that the most frequent category provides a reasonable representation for the absent values. These strategies aim to enhance the completeness and reliability of the dataset for comprehensive variable analysis.

For 'bathrooms_text,' where only 4 values were missing, the decision to remove the corresponding rows was made, as this had an insignificant impact on the dataset's representativeness. In the case of 'bedrooms,' given the substantial number of missing values (411), the entire column was removed to mitigate potential distortions in the analysis. For 'Minimum_Nights,' no missing values were present; however, 2 extreme outliers with values of 364 and 365 were removed to enhance the dataset's robustness.

Moving on to 'reviews_per_month,' which had 231 missing values, imputation was performed using the mean to maintain the integrity of the variable. Similarly, for 'review scores,' around 200 missing values prompted imputation with the median to address the absence of scores. These strategic actions aim to optimize the dataset for subsequent analysis while preserving the reliability of the variables under consideration. *(Entire cleaning code reference: clean_up&_exploration[1].R)*

II. Summary Statistics

Let us conduct a more detailed exploration of the variable Price. Price is a crucial variable in our analysis as it directly influences consumer choices and behaviors, providing valuable insights into market trends, affordability, and potential economic factors affecting the Airbnb listings, thereby shaping strategic decision-making for hosts and users. Understanding price dynamics is fundamental for optimizing listing competitiveness and enhancing overall user experience on the platform. Our initial step involves generating a comprehensive five-number summary for the price.

```{r}
summary(df$price)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   20.0    90.0   125.0   153.4   174.0   986.0
```

The output provides a summary of the distribution of the variable "price" in the data.

Min. (Minimum): The smallest value in the "price" variable is 20.0. This is the minimum price observed in the data.

1st Qu. (First Quartile): The value at the first quartile (25th percentile) is 90.0. This means that 25% of the data falls below this value.

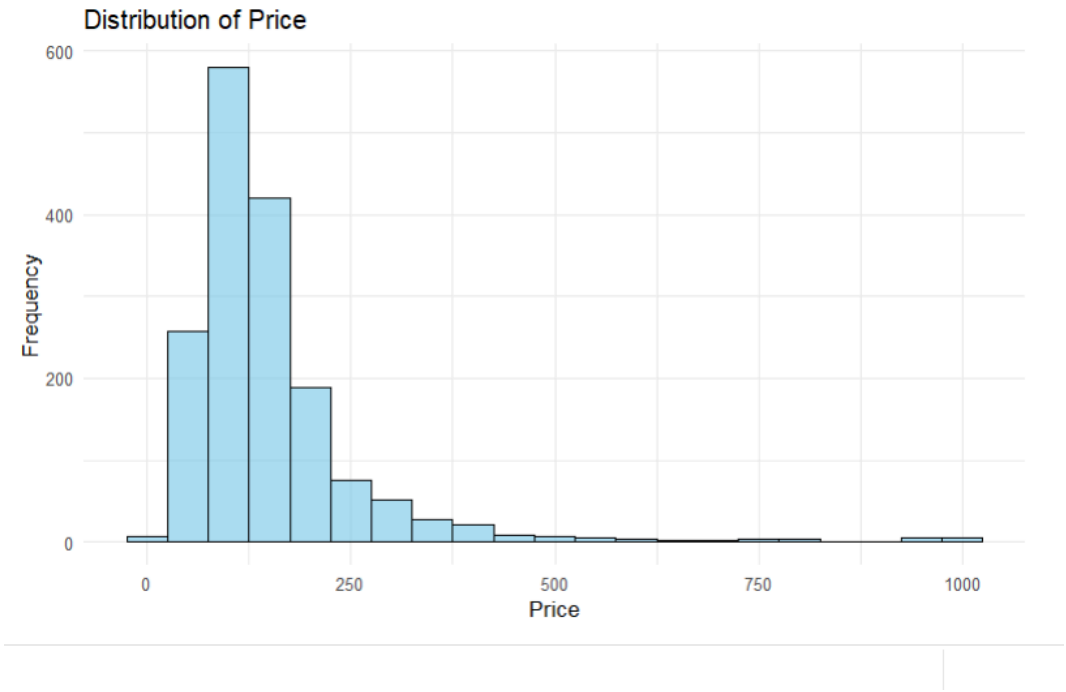
Median: The median (50th percentile) price is 125.0. It represents the middle value of the data when it is ordered.

Mean (Average): The mean (average) price is 153.4. It is calculated by summing up all prices and dividing by the total number of observations.

3rd Qu. (Third Quartile): The value at the third quartile (75th percentile) is 174.0. This means that 75% of the data falls below this value.

Max. (Maximum): The largest value in the "price" variable is 986.0. This is the maximum price observed in the data.

In summary, this output provides a quick overview of the central tendency and spread of the "price" variable, including minimum, quartiles, median, mean, and maximum values.



The histogram of the price variable is right-skewed, which suggests that the majority of the prices are concentrated towards the lower end, with a few higher-priced outliers.

We can further analyze how price varies across some groupings.

```
room_type_summary <- aggregate(price ~ room_type, data = df,
                               FUN = function(x) c(Mean = mean(x), Median = median(x)))

print(room_type_summary)
```

```
room_type_summary
      room_type price.Mean price.Median
Entire home/apt   163.1760     133.5000
     Hotel room   136.1000     118.0000
   Private room   126.8511     100.0000
```

The room_type_summary table summarizes the average and median prices for different room types. It reveals that "Entire home/apt" has the highest mean and median prices among the room types, followed by "Hotel room" and "Private room." This information provides a quick overview of the pricing distribution across room types, highlighting potential patterns or variations in Airbnb listing prices based on the type of accommodation.

```
correlation_matrix <- cor(df[c("price", "beds")])
print("Correlation Matrix:")
print(correlation_matrix)
```

```
[1] "Correlation Matrix:"
          price      beds
price 1.0000000 0.3401174
beds  0.3401174 1.0000000
```

The correlation matrix between "price" and "beds" reveals a positive correlation coefficient of approximately 0.34. This suggests a moderate positive linear relationship between the number of beds in an Airbnb listing and its corresponding price. In other words, as the number of beds increases, there tends to be a modest increase in the listing price.

```
identify_verified <- aggregate(price ~ host_identity_verified, data = df,
                               FUN = function(x) c(Mean = mean(x), Median = median(x)))
identify_verified
```

```
> identify_verified
  host_identity_verified price.Mean price.Median
1                      f   138.1770     100.0000
2                      t   154.4764     126.0000
> 
```

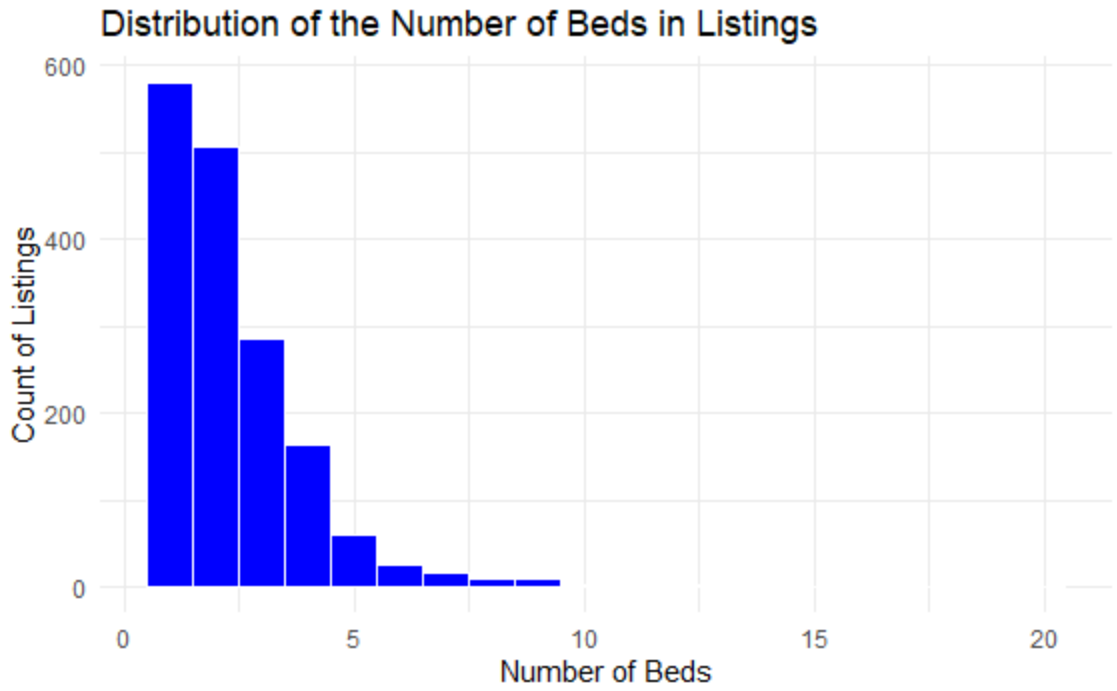
The aggregation based on host identity verification status reveals that listings with verified hosts (host_identity_verified = "t") have a higher average price (Mean = $154.48) compared to listings with unverified hosts (host_identity_verified = "f") with an average price of $138.18. The median price for verified hosts is also higher ($126.00) compared to unverified hosts ($100.00). This suggests a potential association between host identity verification and higher listing prices, emphasizing the role of host verification as a factor influencing pricing patterns in the dataset.

III. Data Visualization

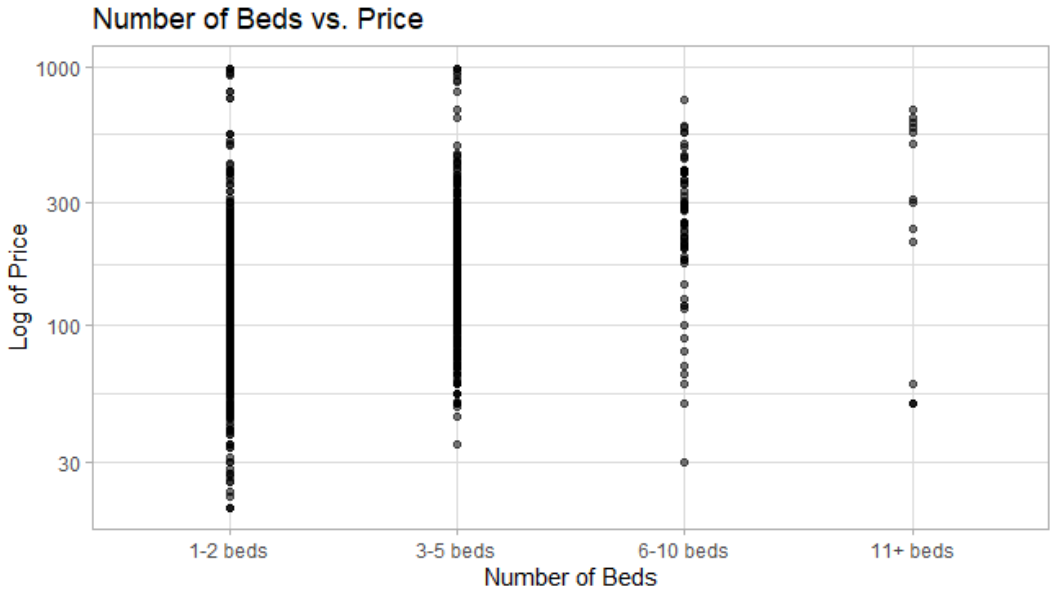
Next, we would like to explore the variable ‘beds’ in greater depth.

5 Visualization for “beds”:

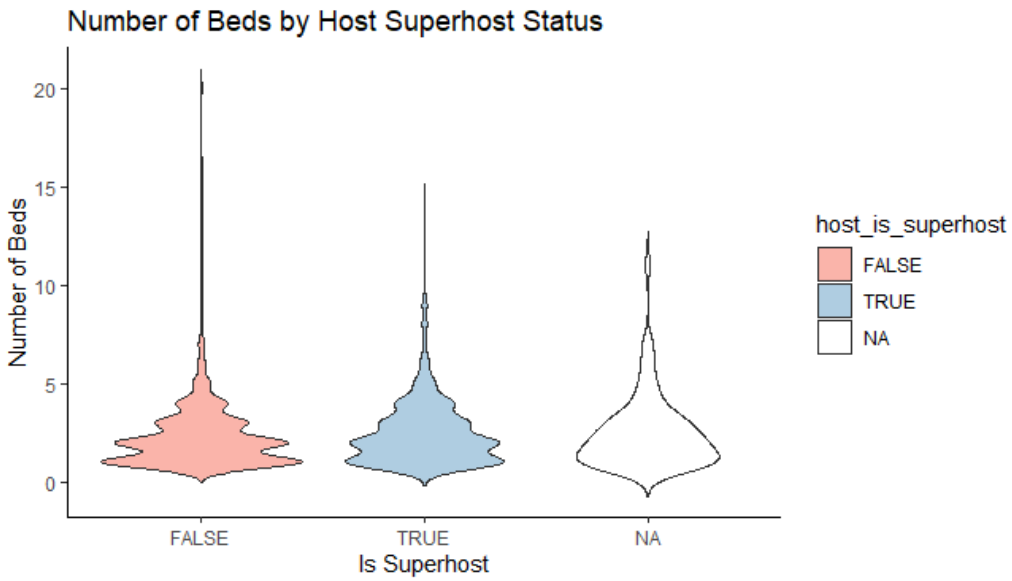
1.


Distribution of the Number of Beds in Listings
Count of Listings
600
400
200
0
0
5
10
15
20
Number of Beds

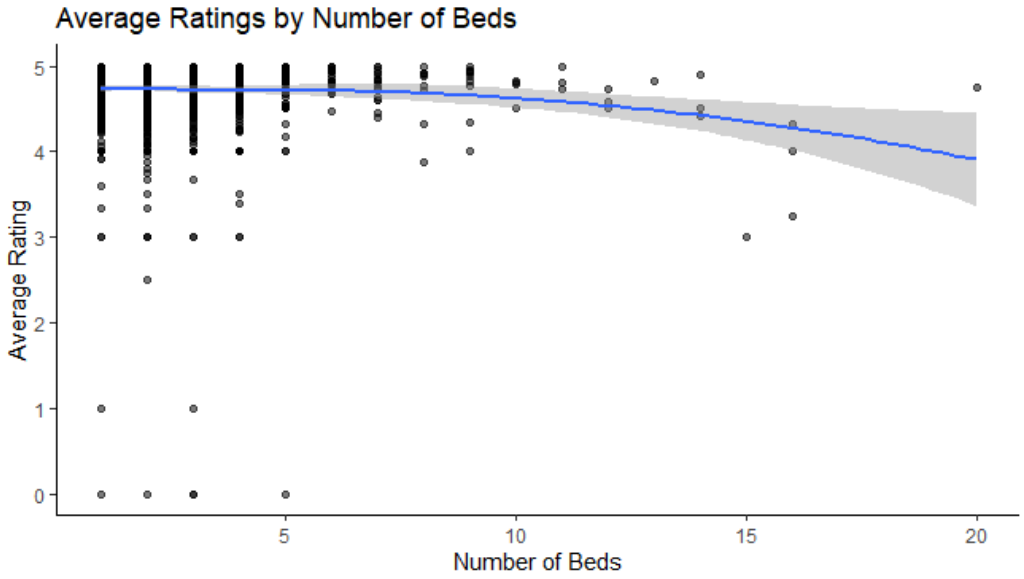
2.

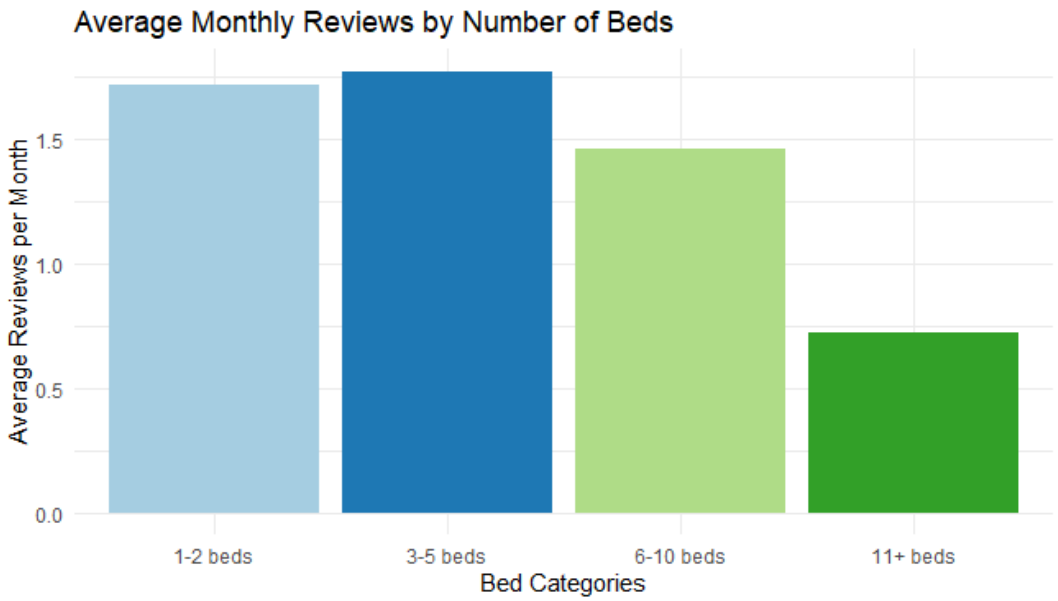

Number of Beds vs. Price
Log of Price
1000
300
100
30
1-2 beds
3-5 beds
6-10 beds
11+ beds
Number of Beds

3.


Number of Beds by Host Superhost Status
Number of Beds
20
15
10
5
0
FALSE
TRUE
NA
Is Superhost
host_is_superhost
FALSE
TRUE
NA

4.


Average Ratings by Number of Beds
Average Rating
5
4
3
2
1
0
5
10
15
20
Number of Beds

5. 

B. In exploring the variable "beds" within an Airbnb dataset, we have generated five visualizations to understand it deeper. Each visualization serves a distinct purpose and provides unique insights into the distribution and impact of the number of beds on Airbnb listings.

The initial histogram shows a high frequency of listings with fewer beds, indicating a market geared predominantly towards solo travelers or small groups. Subsequently, the boxplot correlating bed count to price reveals that while larger accommodations tend to be more expensive, there is a considerable price overlap across categories, suggesting that factors other than bed count also play a significant role in pricing.

In examining host status, the violin plot elucidates that superhosts are not necessarily offering more beds than non-superhosts, which could imply that being a superhost is more about service quality than the size of the accommodation. The trend analysis between bed count and average rating indicates a slight dip in ratings as bed count increases, hinting at potential challenges in maintaining quality or guest satisfaction in larger properties.

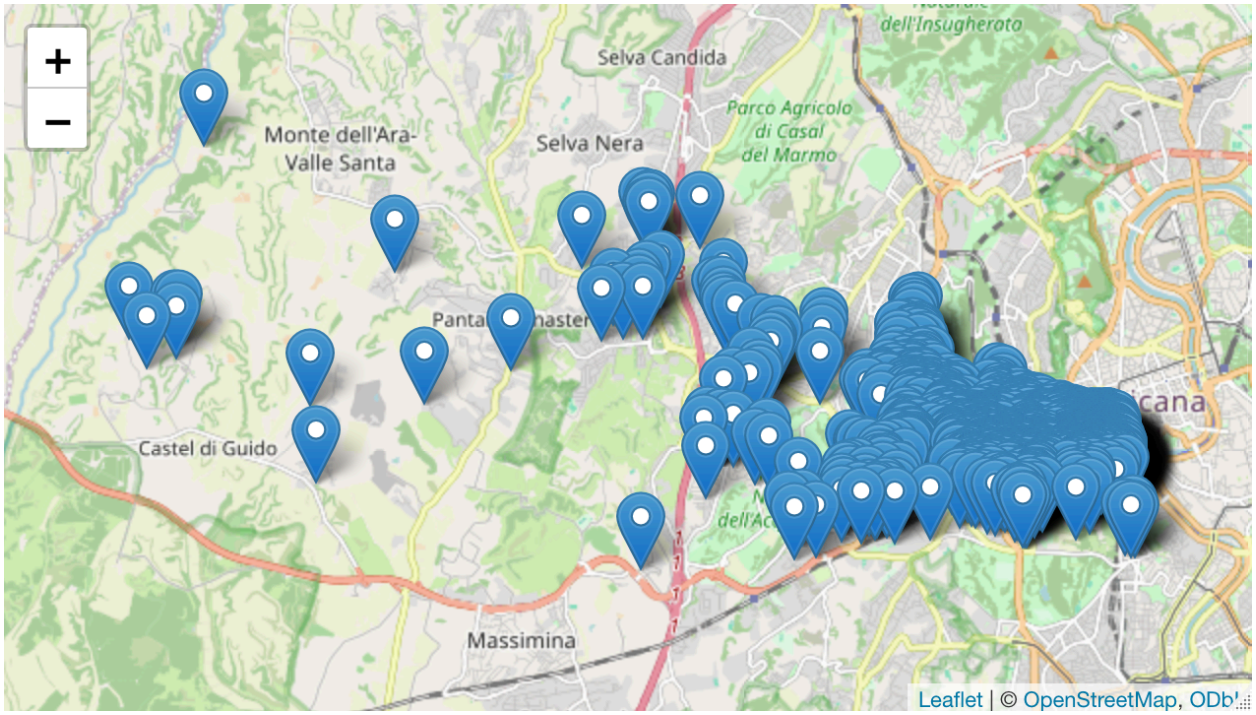
Lastly, the bar chart comparing bed categories with the average monthly reviews suggests that listings with fewer beds are reviewed more frequently, potentially reflecting higher occupancy rates. This data can imply that smaller accommodations are more popular or that they may cater to guests who are more inclined to leave reviews.

IV. Mapping

In our project's initial mapping phase, we used the leaflet package for an interactive property map. However, it lacked clarity in depicting property distribution. To address this, we turned to the tmap and sf packages, enabling the creation of a more detailed map. By converting our dataset into a spatial object (sf), we leveraged tmap's capabilities. The resulting map highlighted concentrated property clusters, particularly around Citta del Vaticano, revealing insights into rental listing density. This shift in mapping underscored the importance of choosing appropriate visualization tools to convey spatial patterns effectively, aligning with our goal of understanding the neighborhood through data visualization.

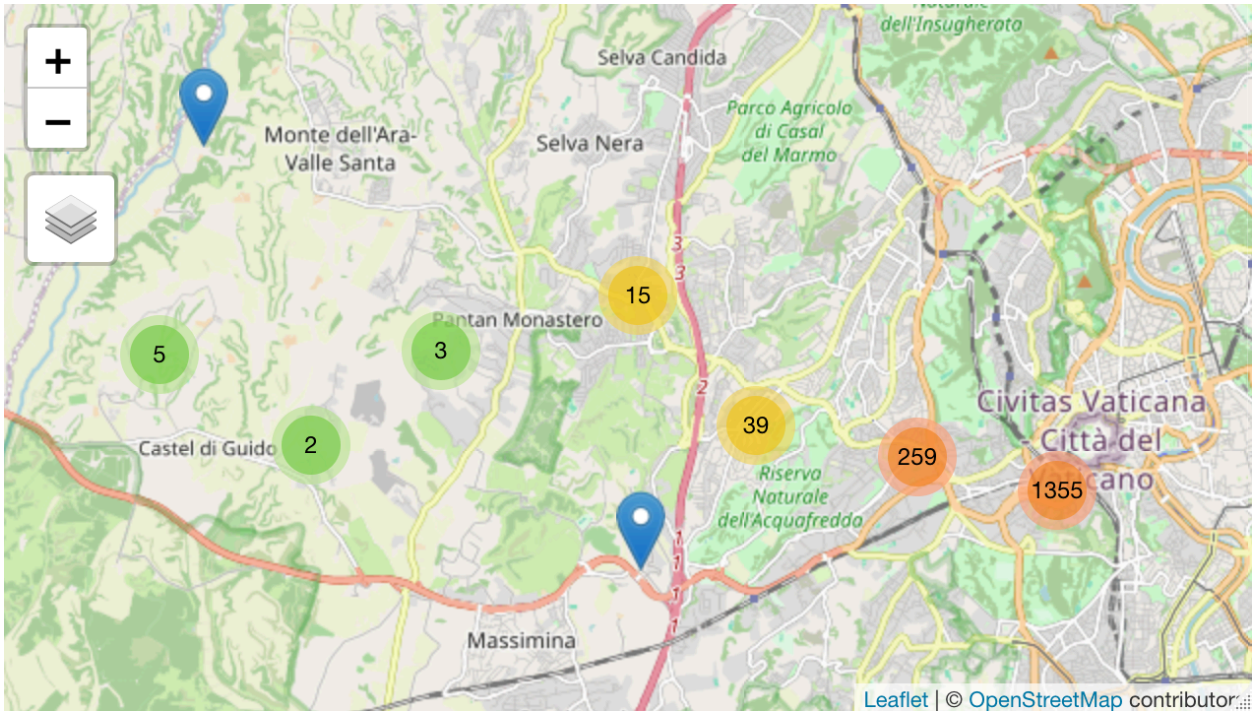
Map 1

```
install.packages("leaflet")
library(leaflet)

# Create a leaflet map
map <- leaflet(neighbourhood_data1) %>%
  addTiles() %>%
  addMarkers(
    ~longitude,
    ~latitude,
    popup = ~paste("Property Type: ", property_type_grouped, "<br>",
                   "Price: $", price, "<br>",
                   "Accommodates: ", accommodates)
  )
print(map)
```



Map 2

```
#### Second mapping
install.packages(c("tmap", "sf"))
library(tmap)
library(sf)

# Convert dataframe to an sf object
neighbourhood_sf <- st_as_sf(neighbourhood_data1, coords = c("longitude", "latitude"))

# Create a tmap object
tm_map <- tm_shape(neighbourhood_sf) +
  tm_basemap(server = "OpenStreetMap") +
  tm_markers(popup.vars = c("property_type_grouped", "price", "accommodates"),
             palette = "Set1")

tmap_leaflet(tm_map)
```



V. Wordcloud

A.

Wordcloud:

The creation of the word cloud from the neighbourhood_overview column in an Airbnb dataset involved a multi-step process, emphasizing both data cleaning and considerations to ensure relevance and clarity. Initially, the dataset likely contained descriptions in various languages, presenting the first challenge: isolating English text to create a coherent and meaningful visualization. To address this, we utilized the cld2 package, which detects and filters out non-English entries, ensuring the word cloud reflected only English descriptions.

Then the step involved converting all text to lowercase to maintain uniformity, stripping white spaces, and removing punctuation and numbers that could clutter the word cloud. Additionally, we removed stopwords—frequent words that carry little unique information about the content of the listings.

However, a preliminary word cloud revealed the presence of common words from other languages, likely Italian, indicating places or common expressions in listings from Italy. To refine the word cloud further, we expanded the stopwords list to include Italian, among other languages, to filter out these common terms and focus on more distinctive words that would be more informative for an English-speaking audience.

The resulting word cloud highlighted key terms that offer insights into the characteristics of the Airbnb listings. Words like "San Pietro", "Vatican", "piazza", "centro", and "minutes" suggest that many listings emphasize proximity to famous landmarks and central locations, indicating a high density of properties near tourist attractions. Descriptors such as "quiet", "safe", "restaurants", and "bars" within the vicinity reflect the neighborhood amenities touted by hosts, which cater to the needs and preferences of potential guests.

And below are the five most valuable aspects we can learn by examining the wordcloud:

Proximity to Attractions: Words like "San Pietro," "Vatican," and "Piazza" suggest listings are close to major tourist spots in Rome.

Transportation: The term "minutes" and the Italian "piedi" hint at listings being well-located for walking and public transport access.

Local Amenities: Frequent mentions of "restaurants" and "bars" indicate vibrant dining and nightlife options nearby.

Quiet Environment: The word "quiet" suggests listings are often described as peaceful, offering a respite from busy areas.

Safety: The appearance of "safe" highlights security as a common feature emphasized in property descriptions.

Step II: Prediction

Regression modeling means not only estimating the coefficients, but also choosing what predictors to include and in what form. First, we filter out variables from the initially cleaned dataset (after missing values treatments))that currently hold no significance in a multiple linear regression model, based on domain knowledge. These include variables such as neighborhood_overview,host_id, host_name, host_since, host_location,latitude,longitude, and a few more.

```{r}

set.seed(194)

train.index <- sample(c(1:nrow(df)), nrow(df)*0.6)
train_df <- df[train.index, ]
valid_df <- df[-train.index, ]

```

When testing a model on new, unseen data, it is crucial to split the data into training and test sets. Train data allows your model to learn, while validation data allows you to determine how well your model learned and performed on new data that it had not previously encountered. Above code splits the data into 60% training and 40% test set.

Before performing MLR, let us check the correlations between numeric variables for our understanding.

```
numeric_df <- train_df[sapply(train_df, is.numeric)]
correlation_matrix <- cor(numeric_df)
correlation_matrix <- as.data.frame(correlation_matrix)
```

The strong correlation between several variables highlights a significant issue of multicollinearity in the regression model. This high correlation can lead to unstable coefficient

estimates and make it hard to figure out the individual impacts of these predictors on the outcome variable.

Before removing the correlated variables, let us see how the regression model looks with all the features (including correlated ones).

```
model <- lm(log(price) ~ . , data = train_df)
summary(model)
```

Here is a partial snapshot of the summary.

```
bathrooms_text2 baths                          -5.070e-01  2.815e-01  -1.801 0.072058 .
bathrooms_text2 shared baths                    3.928e-01  3.419e-01   1.149 0.250954
bathrooms_text2.5 baths                        -6.066e-01  3.177e-01  -1.909 0.056503 .
bathrooms_text3 baths                          -1.861e-01  2.902e-01  -0.641 0.521565
bathrooms_text3 shared baths                   -4.577e-01  5.319e-01  -0.861 0.389731
bathrooms_text3.5 baths                         2.486e-01  5.229e-01   0.475 0.634577
bathrooms_text4 baths                          -7.908e-02  3.341e-01  -0.237 0.812928
bathrooms_text5 baths                          -3.919e-01  3.365e-01  -1.165 0.244389
bathrooms_text5 shared baths                   -2.068e+00  5.522e-01  -3.744 0.000192 ***
bathrooms_text5.5 baths                         1.967e-01  5.251e-01   0.375 0.708062
bathrooms_text6 baths                          -2.248e+00  5.358e-01  -4.195 2.98e-05 ***
bathrooms_text6 shared baths                   -2.295e+00  5.510e-01  -4.164 3.41e-05 ***
bathrooms_text7 baths                           2.347e-01  4.971e-01   0.472 0.636989
bathrooms_textHalf-bath                        -8.045e-01  5.245e-01  -1.534 0.125425
bathrooms_textShared half-bath                 -8.316e-01  4.144e-01  -2.007 0.045075 *
beds                                           -3.477e-02  1.673e-02  -2.078 0.038003 *
minimum_nights                                 -2.900e-02  2.475e-02  -1.172 0.241559
maximum_nights                                 -1.114e-05  3.480e-05  -0.320 0.748916
minimum_nights_avg_ntm                          2.304e-02  2.464e-02   0.935 0.350112
maximum_nights_avg_ntm                         -2.310e-05  3.418e-05  -0.676 0.499279
has_availability                               -4.340e-01  1.049e-01  -4.139 3.81e-05 ***
availability_30                                 1.367e-02  5.097e-03   2.682 0.007439 **
availability_60                                 2.095e-03  3.358e-03   0.624 0.532792
availability_90                                 3.214e-03  1.754e-03   1.833 0.067125 .
availability_365                                3.631e-04  1.489e-04   2.439 0.014924 *
number_of_reviews                              -5.089e-04  2.478e-04  -2.054 0.040257 *
number_of_reviews_ltm                          -1.439e-04  1.360e-03  -0.106 0.915772
number_of_reviews_l30d                         -2.081e-02  1.128e-02  -1.846 0.065244 .
review_scores_rating                            3.171e-02  4.267e-02   0.743 0.457503
review_scores_accuracy                          1.613e-01  9.491e-02   1.700 0.089474 .
review_scores_cleanliness                      -1.000e-02  8.100e-02  -0.123 0.901773
review_scores_checkin                          -2.668e-02  7.890e-02  -0.338 0.735338
review_scores_communication                    -6.217e-02  1.019e-01  -0.610 0.542092
review_scores_location                          2.219e-01  5.882e-02   3.773 0.000172 ***
review_scores_value                            -9.431e-02  8.151e-02  -1.157 0.247553
instant_bookable                                1.042e-01  3.373e-02   3.089 0.002065 **
calculated_host_listings_count                  6.384e-02  2.859e-02   2.233 0.025797 *
calculated_host_listings_count_entire_homes    -6.250e-02  2.856e-02  -2.188 0.028898 *
calculated_host_listings_count_private_rooms   -4.382e-02  2.949e-02  -1.486 0.137533
reviews_per_month                               1.492e-02  1.336e-02   1.117 0.264366
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4329 on 944 degrees of freedom
Multiple R-squared:  0.4618,    Adjusted R-squared:  0.4265
F-statistic: 13.07 on 62 and 944 DF,  p-value: < 2.2e-16
```

We can see that the multiple R-squared is 0.4618 with a p-value <0.05, but there are a lot of insignificant predictors (p-value>0.05). Also, presence of high correlations between variables can lead to unstable coefficient estimates and make it hard to figure out the individual impacts of these predictors on the outcome variable.

Next, we can check for highly correlated variables.

Highly Correlated Variables:

	review_scores_rating	review_scores_accuracy	review_scores_cleanliness	review_scores_checkin	review_scores_communication	review_scores_location	review_scores_value
review_scores_rating	1	0.606575061	0.585691076	0.494459361	0.557390227	0.396742444	0.598366026
review_scores_accuracy	0.606575061	1	0.760600935	0.708424243	0.769518116	0.530953598	0.789113579
review_scores_cleanliness	0.585691076	0.760600935	1	0.596016783	0.703337795	0.456591476	0.725936375
review_scores_checkin	0.494459361	0.708424243	0.596016783	1	0.776057338	0.509316278	0.662220969
review_scores_communication	0.557390227	0.769518116	0.703337795	0.776057338	1	0.553107094	0.723073371
review_scores_location	0.396742444	0.530953598	0.456591476	0.509316278	0.553107094	1	0.623236332
review_scores_value	0.598366026	0.789113579	0.725936375	0.662220969	0.723073371	0.623236332	1

As shown above from the correlation results, all the review metrics(review_scores_accuracy, review_scores_cleanliness, review_scores_checkin, review_scores_communication, and review_scores_value) show high positive correlations with review_scores_rating. review_scores_accuracy and review_scores_communication have a correlation above 0.7. Similarly, review_scores_cleanliness and review_scores_communication also have a correlation above 0.7. This indicates strong positive relationships. As we want to keep a single variable that represents overall guest satisfaction, review_scores_rating is a suitable choice as it has high correlations with other review scores. It is a single, comprehensive metric that summarizes the overall quality of a listing. So, we simplify our multiple linear regression (MLR) model by retaining only one review score.

	availability_30	availability_60	availability_90	availability_365
availability_30	1	0.842838907	0.701278639	0.282097687
availability_60	0.842838907	1	0.911754388	0.400672482
availability_90	0.701278639	0.911754388	1	0.544553364
availability_365	0.282097687	0.400672482	0.544553364	1

As shown above from the correlation results, availability measures are also highly correlated.To mitigate multicollinearity, we are choosing to keep availability_365 and exclude the other availability variables. Also, we are concerned about long-term patterns and overall availability

throughout the year to determine price, so we thought keeping *availability_365* would be prudent.

'beds' and 'accommodates' have a high positive correlation of 0.86. Choosing "beds" as a variable provides a more accurate representation of the lodging's capacity by explicitly indicating the number of dedicated sleeping spaces, offering clarity on the available bed count. This information is crucial for potential guests seeking details on individual sleeping arrangements, allowing them to assess how many people can be accommodated comfortably in separate beds. So, we proceed to drop 'accommodates'.

	host_listings_count	host_total_listings_count	calculated_host_listings_count	calculated_host_listings_count_entire_homes
host_listings_count	1	0.996854806	0.862194291	0.85924156
host_total_listings_count	0.996854806	1	0.858034493	0.85604383
calculated_host_listings_count	0.862194291	0.858034493	1	0.993248347
calculated_host_listings_count_entire_homes	0.85924156	0.85604383	0.993248347	1

Based on the high correlation coefficients, it would be appropriate to choose either "host_listings_count" or "host_total_listings_count" since they exhibit nearly perfect correlation (0.996854806). We are keeping only 'host_total_listings_count'. Similarly, we are also dropping "calculated_host_listings_count" and "calculated_host_listings_count_entire_homes".

We also drop "minimum_nights_avg_ntm" and "maximum_nights_avg_ntm" due to correlation with "minimum_nights" and "maximum_nights".

	number_of_reviews	number_of_reviews_ltm	number_of_reviews_l30d	reviews_per_month
number_of_reviews	1	0.668601645	0.382259496	0.388724144
number_of_reviews_ltm	0.668601645	1	0.681144541	0.645522093
number_of_reviews_l30d	0.382259496	0.681144541	1	0.648911324

Based on the above correlation results, we are also only keeping the reviews for the last 12 months "number_of_reviews_ltm" and dropping "number_of_reviews", "number_of_reviews_l30d", "reviews_per_month" due to high multicollinearity.

Thus, we removed all mentioned variables.

```
train_df <- subset(train_df, select = -c(review_scores_accuracy, review_scores_cleanliness,review_scores_checkin,
review_scores_communication, review_scores_location, review_scores_value, availability_30,availability_60,
availability_90, accommodates, host_listings_count, calculated_host_listings_count,
calculated_host_listings_count_entire_homes, minimum_nights_avg_ntm, maximum_nights_avg_ntm, number_of_reviews,
number_of_reviews_l30d, reviews_per_month))

valid_df <- subset(valid_df, select = -c(review_scores_accuracy, review_scores_cleanliness,review_scores_checkin,
review_scores_communication, review_scores_location, review_scores_value, availability_30,availability_60,
availability_90, accommodates, host_listings_count, calculated_host_listings_count,
calculated_host_listings_count_entire_homes, minimum_nights_avg_ntm, maximum_nights_avg_ntm, number_of_reviews,
number_of_reviews_l30d, reviews_per_month))
```

Next, we can perform Multiple Linear Regression (MLR) with backward elimination, which is a common approach for model selection. It involves iteratively removing non-significant variables from the model to improve its simplicity and interpretability. We have log-transformed the price to address its right-skewed distribution, so the interpretation of coefficients in the regression output will be based on the log-transformed scale.

```
model <- lm(log(price) ~ . , data = train_df)

model_1 <- step(model, direction = "backward")
summary(model_1)
print(summary(model_1))
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-1.79498 -0.27075  0.00876  0.24009  2.10327

Coefficients:
                                              Estimate Std. Error t value Pr(>|t|)
(Intercept)                                  5.2665933  0.3578884  14.716  < 2e-16 ***
host_identity_verifiedt                      0.1507163  0.0650601   2.317 0.020734 *
room_typeHotel room                         -0.0707140  0.1181726  -0.598 0.549715
room_typePrivate room                       -0.3168582  0.0735292  -4.309 1.80e-05 ***
bathrooms_text0 baths                        0.0788431  0.4459748   0.177 0.859712
bathrooms_text0 shared baths                -0.9489209  0.4459758  -2.128 0.033609 *
bathrooms_text1 bath                        -0.8993893  0.2991314  -3.007 0.002709 **
bathrooms_text1 private bath                -0.7269655  0.2911509  -2.497 0.012694 *
bathrooms_text1 shared bath                 -1.0131384  0.3015677  -3.360 0.000811 ***
bathrooms_text1.5 baths                     -0.6579431  0.3094491  -2.126 0.033740 *
bathrooms_text1.5 shared baths              -1.0808374  0.3566645  -3.030 0.002507 **
bathrooms_text2 baths                       -0.5842525  0.2999519  -1.948 0.051724 .
bathrooms_text2 shared baths                 0.2187651  0.3619769   0.604 0.545744
bathrooms_text2.5 baths                     -0.6529565  0.3394504  -1.924 0.054700 .
bathrooms_text3 baths                       -0.1478475  0.3102109  -0.477 0.633752
bathrooms_text3 shared baths                -0.3980365  0.5578942  -0.713 0.475731
bathrooms_text3.5 baths                      0.4621453  0.5584062   0.828 0.408092
bathrooms_text4 baths                       -0.1093698  0.3572856  -0.306 0.759584
bathrooms_text5 baths                       -0.4030341  0.3562364  -1.131 0.258180
bathrooms_text5 shared baths                -1.9885235  0.5723717  -3.474 0.000535 ***
bathrooms_text5.5 baths                     -0.0954118  0.5626610  -0.170 0.865382
bathrooms_text6 baths                       -2.2488136  0.5735305  -3.921 9.44e-05 ***
bathrooms_text6 shared baths                -2.0834693  0.5750055  -3.623 0.000306 ***
bathrooms_text7 baths                        0.1123378  0.4666547   0.241 0.809815
bathrooms_textHalf-bath                     -0.8135016  0.5654793  -1.439 0.150584
bathrooms_textShared half-bath              -0.8480498  0.4452953  -1.904 0.057144 .
beds                                         0.0372340  0.0125411   2.969 0.003061 **
minimum_nights                              -0.0086070  0.0030610  -2.812 0.005026 **
has_availability                            -0.4283036  0.1102587  -3.885 0.000109 ***
availability_365                             0.0009797  0.0001311   7.472 1.76e-13 ***
number_of_reviews_ltm                       -0.0033300  0.0008064  -4.130 3.95e-05 ***
review_scores_rating                         0.0865730  0.0354458   2.442 0.014767 *
instant_bookable                             0.1376459  0.0315439   4.364 1.42e-05 ***
calculated_host_listings_count_private_rooms  0.0205479  0.0088505   2.322 0.020457 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4719 on 973 degrees of freedom
Multiple R-squared:  0.341,     Adjusted R-squared:  0.3187
F-statistic: 15.26 on 33 and 973 DF,  p-value: < 2.2e-16
```

Above is a screenshot of the model results.

Break-down of the Regression Equation:

Intercept (Constant): The intercept term is 5.2665933. This represents the predicted log-transformed price when all predictors are zero (or for categorical predictors, for the reference categories).

Coefficients for Predictors: Each coefficient represents the change in the log-transformed price associated with a one-unit change in the corresponding predictor, holding all other predictors

constant. For example, the coefficient 0.1507163 for host_identity_verified suggests that, all else being constant, being a verified host is associated with a 15.07% increase in the log-transformed price.

Significant Predictors:

The significance of each coefficient is indicated by the p-values. For instance, a p-value less than 0.05 is often considered statistically significant. Several predictors are statistically significant, as indicated by the asterisks in the "Estimate" column and the corresponding p-values. Examples of significant predictors include host_identity_verified, room_type (specific categories), bathrooms_text (specific categories), beds, minimum_nights, has_availability, availability_365, number_of_reviews_ltm, review_scores_rating, instant_bookable, and calculated_host_listings_count_private_rooms.

Exponential Transformation: To interpret these coefficients in terms of the original price, we can exponentiate them. For example, exp(0.0372340) for beds would represent the multiplicative change in the price associated with a one-unit increase in the number of beds.

Metrics

The model's performance is assessed using metrics like the residual standard error, multiple R-squared, and adjusted R-squared. The multiple R-squared value indicates the proportion of variance in the log-transformed price explained by the predictors (34.1%).

The overall significance of the model is assessed by the F-statistic and its associated p-value. F-statistic is 15.26. F-Test helps to answer how much more effective is our model, vs. having no model at all. The better the fit, the higher the F-score will be. The F-statistic is used in linear regression to assess the overall significance of a regression model. It quantifies the ratio of the variation explained by the model to the unexplained random variation. The F-statistic of 15.26 in

this case, typically indicates that at least one predictor is significant in explaining the dependent variable, and the model is meaningful. A low p-value associated with the F-statistic suggests that the model is statistically significant in predicting the dependent variable.

```
Residuals:
     Min       1Q   Median       3Q      Max
-1.79498 -0.27075  0.00876  0.24009  2.10327
```

The residuals (differences between predicted and actual log prices) show a summary of the model's predictive accuracy.

Min: This is the minimum value of the residuals. The minimum residual is -1.79498, indicating that there is at least one data point for which the model underpredicts the log-transformed price by approximately 1.79498 units.

1Q (First Quartile): This is the value below which 25% of the residuals fall. It's -0.27075, suggesting that a quarter of the residuals are below -0.27075.

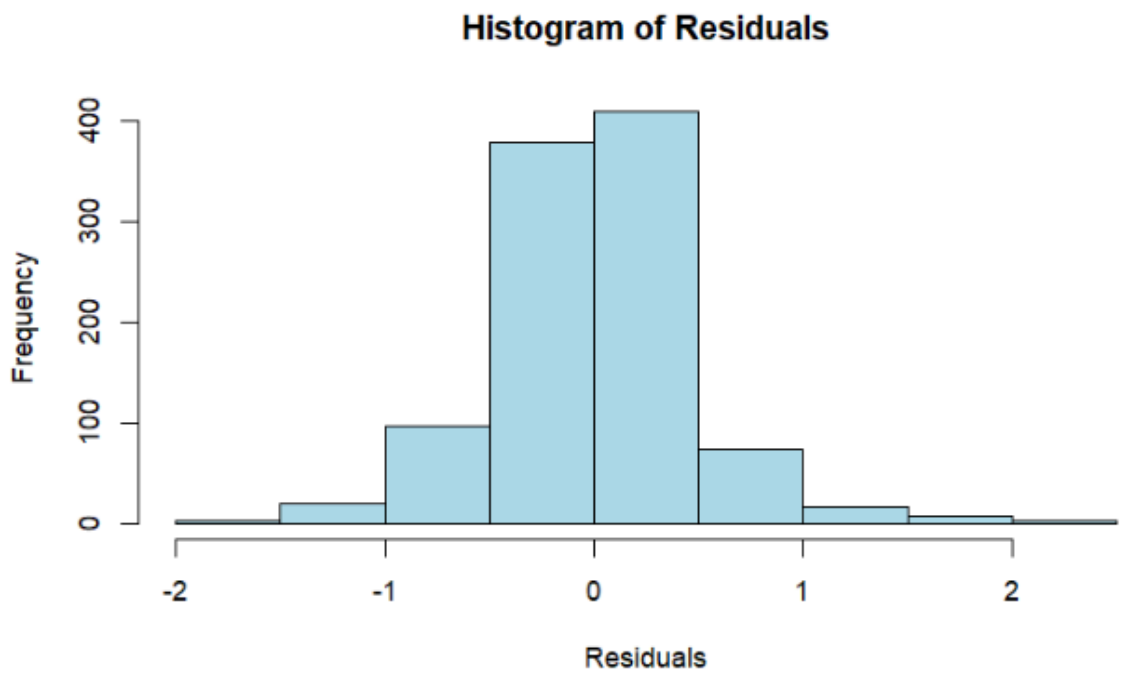
Median: This is the median (or middle) value of the residuals. It is 0.00876, indicating that half of the residuals are below this value, and half are above.

3Q (Third Quartile): This is the value below which 75% of the residuals fall. It's 0.24009, meaning that three-quarters of the residuals are below 0.24009.

Max: This is the maximum value of the residuals. The maximum residual is 2.10327, indicating that there is at least one data point for which the model overpredicts the log-transformed price by approximately 2.10327 units.

We can also plot the histogram of residuals.

```
residuals <- resid(model_1)
hist(residuals, main="Histogram of Residuals", xlab="Residuals", col="lightblue", border="black")
```



Residuals follow a normal distribution, which is a desirable characteristic for a regression model.

Next, let us check the accuracy of the training set.

```
pred = predict (model_1, train_df)
accuracy(pred, train_df$price)
```

```
                ME      RMSE       MAE       MPE      MAPE
Test set 146.8047 187.2087 146.8047 95.62446 95.62446
```

Let us also check the accuracy of the validation set.

```
pred = predict (model_1, valid_df)
accuracy(pred, valid_df$price)
```

```
                ME      RMSE       MAE       MPE      MAPE
Test set 151.5474 194.9045 151.5474 95.72861 95.72861
```

In both the train and validation sets, the RMSE and MAE and all other error measures are comparable but slightly higher for the validation set. The RMSE of the train data is 187.20 while the RMSE of the test data is 194.90. The MAE of the train data is 146.80 while the MAE of the test data is 151.54.

A model that overfits the training data may have limited predictive power for new observations, which could result in poorer accuracy on the validation set. If the accuracy on the validation set is significantly lower than that on the training set, it indicates potential overfitting. Here, there seems to be no issue of overfitting.

Step III: Classification

Part I.

K nearest neighbor

Predictor Choices and Model Development:

In this project, we focused on predicting the availability of wifi in rental properties using the k-nearest neighbors (kNN) algorithm. To begin, we chose predictors that seemed most relevant to the presence of wifi, such as accommodates, host_total_listings_count, price, and beds. These features were normalized to ensure they were on the same scale, enhancing the model's performance.

```r
df$amenities <- factor(df$amenities)


set.seed(1626)
train.index <- sample(c(1:nrow(df)), nrow(df)*0.6)
train.df <- df[train.index, ]
valid.df <- df[-train.index, ]


library(dplyr)
has_wifi <- filter(train.df, amenities =="TRUE")
no_wifi <- filter(train.df, amenities =="FALSE")
```

The grepl() function played a crucial role in our preprocessing steps. It allowed us to create a target variable by scanning through the amenities column to check for the presence of "wifi". This resulted in a new logical column indicating whether each listing included wifi.

```r
t.test(has_wifi$review_scores_rating, no_wifi$review_scores_rating)
```

```
        Welch Two Sample t-test

data:  has_wifi$review_scores_rating and no_wifi$review_scores_rating
t = 1.6602, df = 26.302, p-value = 0.1088
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.06191957  0.58345396
sample estimates:
mean of x mean of y
 4.729286  4.468519
```

We also conducted the Welch Two Sample t-test to compare the means of variables between two groups: has_wifi and no_wifi. We decided to use the alpha value of 0.05. Thus, we did not use the variable review_scores_rating. The p-value for that variable is 0.1088.

Model Tuning and k Value Selection:

The choice of k, the number of neighbors to consider, is critical in kNN modeling. To determine the optimal k value, we utilized cross-validation with a range of k values. This approach helps in identifying a k that balances the bias-variance trade-off effectively.

```
#getting the correct number of clusters

library(caret)
set.seed(1616)


train_control <- trainControl(method = "cv", number = 10)  # 10-fold cross-validation
knn_fit <- train(amenities ~ ., data = train.norm.df, method = "knn", trControl = train_control,
tuneLength = 10)

# Viewing the results
print(knn_fit)
```

```
k-Nearest Neighbors

1007 samples
   6 predictor
   2 classes: 'FALSE', 'TRUE'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 906, 906, 906, 907, 906, 906, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   5  0.9722079  -0.001351351
   7  0.9732079   0.000000000
   9  0.9732079   0.000000000
  11  0.9732079   0.000000000
  13  0.9732079   0.000000000
  15  0.9732079   0.000000000
  17  0.9732079   0.000000000
  19  0.9732079   0.000000000
  21  0.9732079   0.000000000
  23  0.9732079   0.000000000

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 23.
```

Using 10-fold cross-validation provided a robust assessment of the model across different subsets of the data. The final k value was chosen based on the highest accuracy achieved during

this process. Although the cross-validation suggested k = 23 as the optimal value, We decided to proceed with k = 7. This decision was based on a balance between model complexity and the desire to avoid overfitting, considering the size and characteristics of the dataset.

```
#Accuracy was used to select the optimal model using the largest value.
#The final value used for the model was k = 7.

new_df.norm <- new_df.norm[, 1:4]

nn1 <- knn(train = train.norm.df[, 1:4], test = new_df.norm,
          cl = train.norm.df[, 5], k = 7)
row.names(train.df)[attr(nn1, "nn.index")]

nn1
```

```
[1] "642"  "13"   "345"  "849"  "189"  "1007" "218"
[1] TRUE
attr(,"nn.index")
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]  642   13  345  849  189 1007  218
attr(,"nn.dist")
           [,1]       [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
[1,] 0.04973519 0.08289199 0.1714191 0.1807833 0.2417038 0.4217599 0.4285477
Levels: TRUE
```

We created a fictional airbnb host to use the model. The new host can accommodate 6 people, has a total listing count of 6, the price for each listing is 300, and they have 3 beds per accommodation . The screenshot below shows their 7 nearest neighbors.

```r
#7 nearest neighbours
values_to_filter <- c("642", "13", "345", "849", "189", "1007", "218")

# Add a new column with row names
train.norm.df$row_id <- rownames(train.norm.df)
valid.norm.df$row_id <- rownames(valid.norm.df)

# Now filter using this new column
filtered_data <- subset(train.norm.df, row_id %in% values_to_filter, select = c("amenities",
"host_name"))


print(filtered_data)
```

Description: df [7 × 2]

	amenities <fctr>	host_name <chr>
13	TRUE	Dani&Pietro
189	TRUE	Somrit
218	TRUE	Maria Luisa
345	TRUE	Cristiano
642	TRUE	Luca
849	TRUE	Valerio
1007	TRUE	Catia

7 rows

Finally, the accuracy of the model was assessed using a separate validation set. This step is crucial for evaluating the model's performance on unseen data, ensuring its reliability and generalizability. The high accuracy (97%) achieved suggests that the model is effective in predicting the presence of wifi in rental listings based on the selected features.

```r
library(caret)

# Assuming 'valid_labels' contains the actual class labels for your validation set

# Calculate accuracy
accuracy <- sum(nn1 == valid.norm.df[, 5]) / length(valid.norm.df[, 5])


print(accuracy)

```

```
[1] 0.9761905
```

Classification, Part II. Naive Bayes

Model Building using Naive Bayes

For building the Naive Bayes model, we first binned the `review_scores_rating` column into four categories using equal frequency binning.

```r
# Binning using cut()
df1$review_scores_rating_bin <- cut(df1$review_scores_rating,
                                    breaks = quantile(df1$review_scores_rating,
                                                      probs = seq(0, 1, length.out = n_bins + 1),
                                                      na.rm = TRUE),
                                    include.lowest = TRUE,
                                    labels = FALSE)
```

This was achieved by calculating quantiles and applying the `cut()` function. After binning, We ensured that this binned variable is a factor, which is necessary for classification tasks.

The values in each bin range from:

review_scores_rating_bin	Min_Rating	Max_Rating
<fctr>	<dbl>	<dbl>
1	0.00	4.67
2	4.68	4.82
3	4.83	4.92
4	4.93	5.00

We the selected a set of predictors - `host_identity_verified`, `property_category`, `beds_grouped`, `price`, `host_response_rate`, and `host_acceptance_rate` - ensuring these are also factors. This choice of predictors reflects a mix of property features and host characteristics.

```{r}
# Converting categorical variables to factors
train_df$host_identity_verified <- factor(train_df$host_identity_verified)
train_df$property_category <- factor(train_df$property_category)
train_df$binned_property_type <- factor(train_df$binned_property_type)
train_df$beds_grouped <- factor(train_df$beds_grouped)

train_df$host_acceptance_rate <- factor(train_df$host_acceptance_rate)
train_df$price <- factor(train_df$price)
train_df$host_response_rate <- factor(train_df$host_response_rate)



# Retrain the model
rating.nb1 <- naiveBayes(review_scores_rating_bin ~ host_identity_verified + property_category +
beds_grouped + price + host_response_rate + host_acceptance_rate, data = train_df)

rating.nb1
```

```
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
        1         2         3         4
0.2790467 0.3207547 0.1698113 0.2303873

Conditional probabilities:
   host_identity_verified
Y            0          1
  1 0.05693950 0.94306050
  2 0.08978328 0.91021672
  3 0.04093567 0.95906433
  4 0.06465517 0.93534483
```

After splitting your data into training and validation sets, we trained a Naive Bayes model using the `naiveBayes` function from the `e1071` package. The model aimed to predict the binned `review_scores_rating` based on the selected predictors.

Predicting for a Fictional Apartment

We created a fictional apartment with specified characteristics (`host_identity_verified = 0`, `property_category = "Private Room"`, etc.) and predicted its review score rating bin using the trained Naive Bayes model. The prediction suggested that this fictional apartment most likely falls into bin 1. The rating range for this bin is 0.00 to 4.67. This bin has the lowest rating score.

Code for Building, Running, and Assessing the Model

The process we followed includes:

1. Binning the `review_scores_rating`.

2. Defining and transforming predictors.

3. Splitting the data into training and validation sets.

4. Training the Naive Bayes model.

5. Predicting the class for both training and validation sets and the new apartment.

6. Assessing the model using confusion matrices.

In this task, we implemented a Naive Bayes classifier to predict the binned rental rating scores of properties. The challenge was to categorize the continuous `review_scores_rating` into discrete bins for classification. We chose four bins of equal frequency, ensuring a balanced representation across the dataset. The predictors for the model were carefully selected to capture diverse aspects like accommodation capacity, host factors, and property type, each converted into categorical factors for suitability in the Naive Bayes framework.

```r
# Converting categorical variables to factors
train_df$host_identity_verified <- factor(train_df$host_identity_verified)
train_df$property_category <- factor(train_df$property_category)
train_df$binned_property_type <- factor(train_df$binned_property_type)
train_df$beds_grouped <- factor(train_df$beds_grouped)

train_df$host_acceptance_rate <- factor(train_df$host_acceptance_rate)
train_df$price <- factor(train_df$price)
train_df$host_response_rate <- factor(train_df$host_response_rate)



# Retrain the model
rating.nb1 <- naiveBayes(review_scores_rating_bin ~ host_identity_verified + property_category +
beds_grouped + price + host_response_rate + host_acceptance_rate, data = train_df)

rating.nb1
```

```
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
        1         2         3         4
0.2790467 0.3207547 0.1698113 0.2303873

Conditional probabilities:
   host_identity_verified
Y            0          1
  1 0.05693950 0.94306050
  2 0.08978328 0.91021672
  3 0.04093567 0.95906433
  4 0.06465517 0.93534483
```

The model's performance was assessed using confusion matrices on both training and validation datasets. While the training data showed a decent level of accuracy, the performance on the validation data was lower, indicating potential overfitting or the need for further model tuning.

We tested different variables to get the best accuracy for the validation set.

```
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3   4
         1 178  39  16  23
         2  38 204  37  37
         3  31  37  89  19
         4  34  43  29 153

Overall Statistics

               Accuracy : 0.6197
                 95% CI : (0.5889, 0.6498)
    No Information Rate : 0.3208
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.4857

 Mcnemar's Test P-Value : 0.1495

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            0.6335   0.6316  0.52047   0.6595
Specificity            0.8926   0.8363  0.89593   0.8632
Pos Pred Value         0.6953   0.6456  0.50568   0.5907
Neg Pred Value         0.8628   0.8278  0.90132   0.8944
Prevalence             0.2790   0.3208  0.16981   0.2304
Detection Rate         0.1768   0.2026  0.08838   0.1519
Detection Prevalence   0.2542   0.3138  0.17478   0.2572
Balanced Accuracy      0.7630   0.7339  0.70820   0.7614
Warning: Type mismatch between training and new data for variable 'price'. Did you use factors with numeric labels for
training, and numeric values for new data?Confusion Matrix and Statistics

          Reference
Prediction  1  2  3  4
         1 67 41 22 37
         2 35 80 29 58
         3 15 13 11 17
         4 40 70 69 68

Overall Statistics

               Accuracy : 0.3363
                 95% CI : (0.3006, 0.3734)
    No Information Rate : 0.3036
    P-Value [Acc > NIR] : 0.03655

                  Kappa : 0.0982

 Mcnemar's Test P-Value : 3.508e-07

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            0.4268   0.3922  0.08397   0.3778
Specificity            0.8058   0.7393  0.91682   0.6362
Pos Pred Value         0.4012   0.3960  0.19643   0.2753
Neg Pred Value         0.8218   0.7362  0.80519   0.7365
Prevalence             0.2336   0.3036  0.19494   0.2679
Detection Rate         0.0997   0.1190  0.01637   0.1012
Detection Prevalence   0.2485   0.3006  0.08333   0.3676
Balanced Accuracy      0.6163   0.5657  0.50040   0.5070
Warning: Type mismatch between training and new data for variable 'host_identity_verified'. Did you use factors with numeric
labels for training, and numeric values for new data?Warning: Type mismatch between training and new data for variable
'beds_grouped'. Did you use factors with numeric labels for training, and numeric values for new data?[1] 1
Levels: 1 2 3 4
```

This discrepancy between training and validation performance highlighted the importance of cross-validation in model assessment. The prediction for a fictional apartment showcased the model's practical application, providing insights into how the model might perform in real-world scenarios. The entire process emphasized the significance of thoughtful feature selection and the challenges in balancing model complexity with predictive accuracy.

Classification, Part III. Classification Tree

In this analysis, we employed the R programming language and several packages, including `caret`, `rpart`, and `rpart.plot`, to build and evaluate a classification tree predicting the instant bookability of units in a dataset.

A. Variable Importance and Tree Construction:

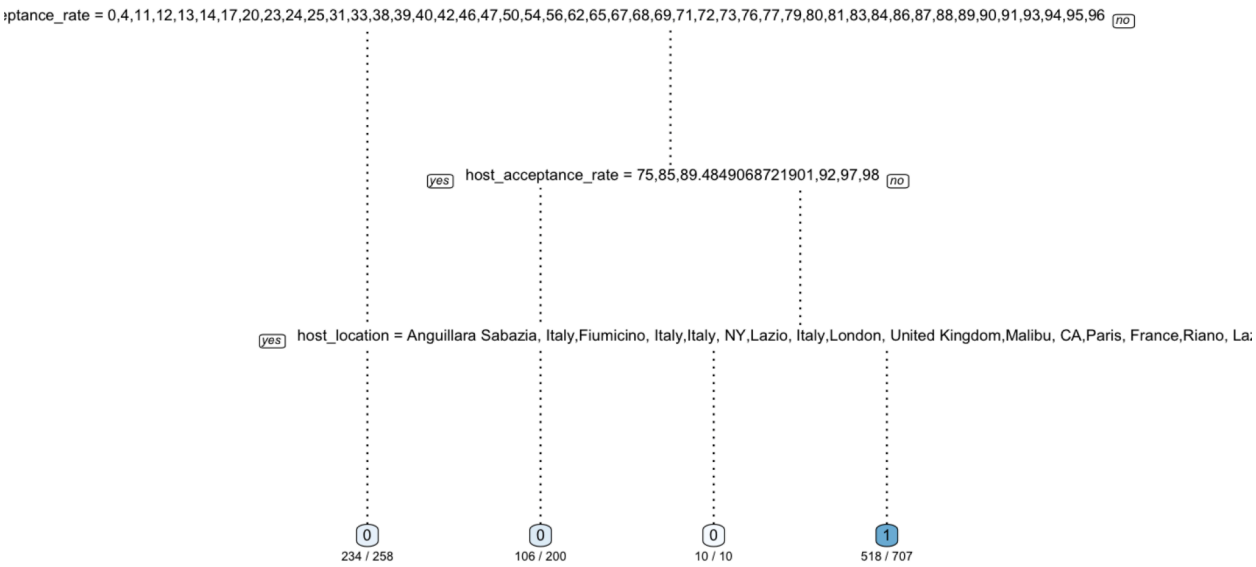
We began by installing and loading the `caret` package to assess the importance of variables for classification. The variable importance scores were obtained and visualized, aiding in the selection of crucial features for the classification tree. Following this, we converted categorical variables to factors and carefully selected important variables for the analysis. The dataset was then split into training and testing sets, with efforts to maintain consistency in factor levels between them.

The classification tree was constructed using the `rpart` package, with pruning applied to avoid overfitting. The pruned tree was visualized using the `rpart.plot` package, enhancing aesthetics and interpretability.

```r
# Install and load packages
library(rpart)
library(rpart.plot)

# Convert categorical variables to factors
cleaned_df$instant_bookable <- as.factor(cleaned_df$instant_bookable)

# Select only important variables
selected_vars <- c("host_acceptance_rate", "host_location", "host_response_time", "instant_bookable")
cleaned_df_selected <- cleaned_df[selected_vars]

# Split data into training and testing sets
set.seed(123)
split_index <- sample(1:nrow(cleaned_df_selected), 0.7 * nrow(cleaned_df_selected))
train_data <- cleaned_df_selected[split_index, ]
test_data <- cleaned_df_selected[-split_index, ]

# Ensure consistency in factor levels between train and test data
for (var in selected_vars) {
  train_data[[var]] <- as.factor(train_data[[var]])
  test_data[[var]] <- factor(test_data[[var]], levels = levels(train_data[[var]]))
}

# Build the classification tree with pruning
tree_model_pruned <- rpart(instant_bookable ~ ., data = train_data, method = "class", cp = 0.01)

# Get variable importance
var_importance <- varImp(tree_model_pruned)

# Print the variable importance scores
print(var_importance)
```

```r
# Visualize the pruned tree with improved aesthetics
prp(tree_model_pruned,
    extra = 2,          # Adds information about split criterion and variable importance
    under = TRUE,       # Displays branch labels under the arrow line
    varlen = 0,         # Adjusts variable name length
    faclen = 0,         # Adjusts factor level length
    cex = 0.7,          # Adjusts overall text size
    tweak = 1.2,        # Adjusts spacing between branches
    fallen.leaves = TRUE, # Display fallen leaves for terminal nodes
    branch.lty = 3,      # Use dashed lines for branches
    branch.lwd = 2,      # Set line width for branches
    yesno = 2,          # Display "Yes" and "No" instead of 1 and 0
    split.font = 0.5,    # Adjust font size for split labels
    box.palette = "Blues", # Set color palette
    digits = 2           # Number of digits for edge labels
)

# Use the summary function on the pruned tree model
summary(tree_model_pruned)

# Make predictions on the test set
predictions <- predict(tree_model_pruned, newdata = test_data, type = "class")

# Evaluate model performance
confusion_matrix <- table(predictions, test_data$instant_bookable)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Accuracy:", round(accuracy, 2)))
```

```
> print(paste("Accuracy:", round(accuracy, 2)))
[1] "Accuracy: 0.66"
```



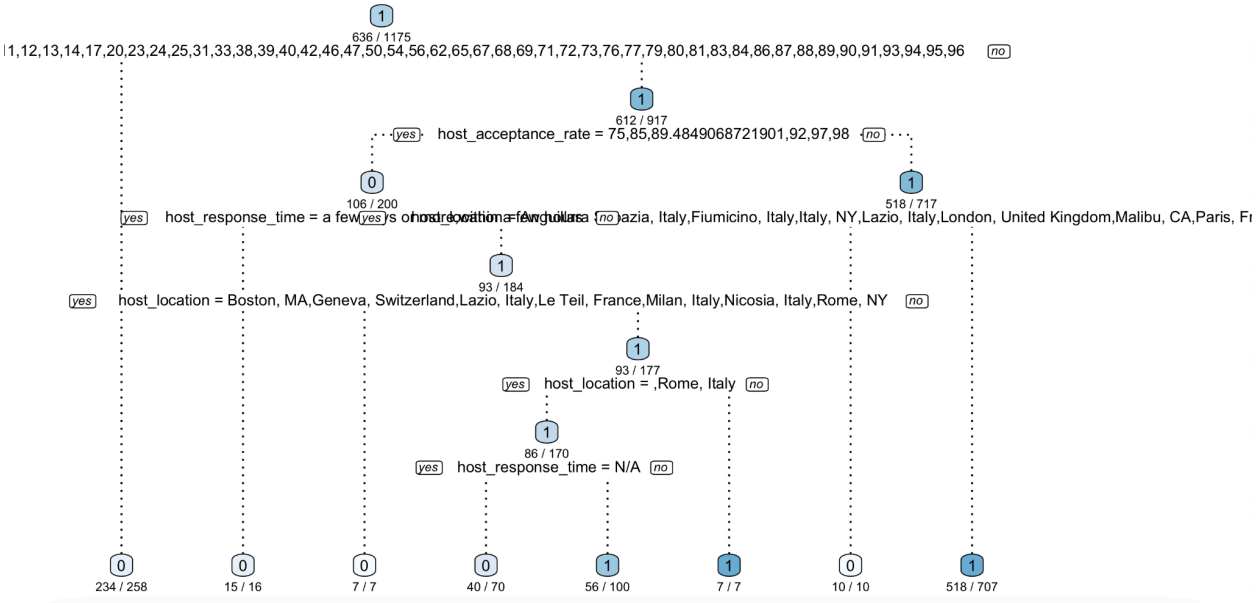
B. Cross-Validation for Tree Tuning:

To determine the ideal size of the tree, we performed cross-validation using the `caret` package. A grid of complexity parameter (cp) values was created, and the cross-validated tuning process identified the optimal cp value. The final tree model was then built using this optimal cp value.

```r
#B. Determine the ideal size of your tree using cross-validation.
# Install and load required packages
install.packages(c("rpart", "rpart.plot", "caret"))
library(rpart)
library(rpart.plot)
library(caret)

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

# Create a grid of complexity parameter (cp) values
cp_grid <- expand.grid(cp = seq(0.001, 0.1, by = 0.001))

# Perform cross-validated tuning of the rpart model
fit_control <- trainControl(method = "cv", number = 5)
cv_results <- train(instant_bookable ~ ., data = train_data, method = "rpart",
                    trControl = fit_control, tuneGrid = cp_grid)

# Extract the best cp value
best_cp <- cv_results$bestTune$cp

# Build the tree with the optimal size
tree_model_optimal <- rpart(instant_bookable ~ ., data = train_data, method = "class", cp = best_cp)
```

C. Visualization of the Optimal Tree:

The optimal tree, obtained through cross-validation, was visualized using the `rpart.plot` package with carefully chosen graphical parameters. This visualization provided a clear representation of the decision-making process within the model.



```r
#C. Part C: Using rpart.plot and your choice of graphical parameters, show your tree model here.
# Visualize the optimal tree using rpart.plot
rpart.plot::rpart.plot(tree_model_optimal, extra = 2, under = TRUE, varlen = 0, faclen = 0, cex = 0.7,
                       tweak = 1.2, fallen.leaves = TRUE, branch.lty = 3, branch.lwd = 2, yesno = 2,
                       split.font = 0.5, box.palette = "Blues", digits = 2)
```

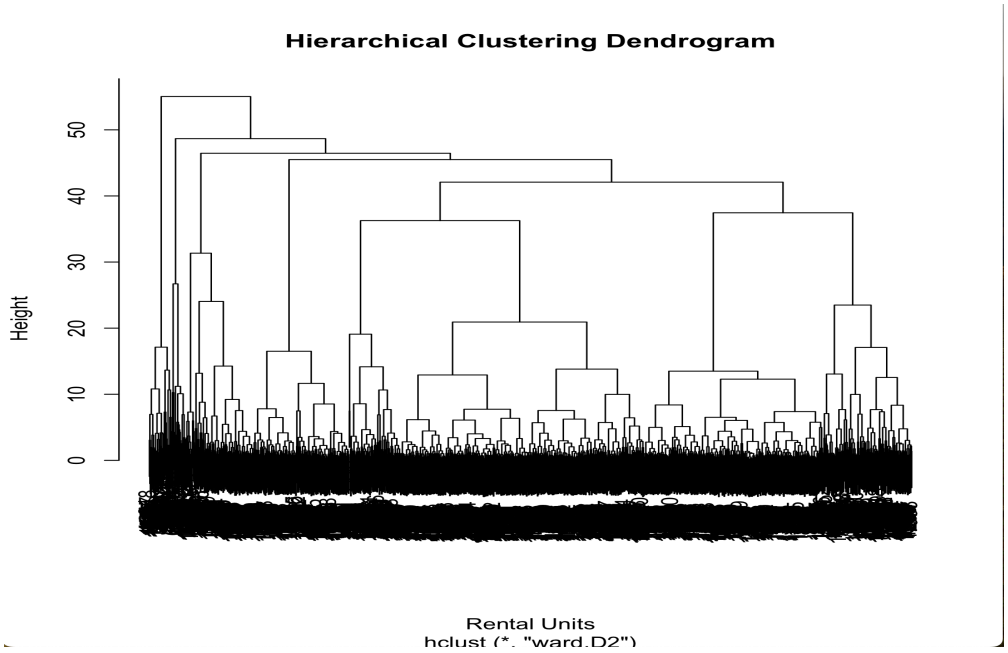
D. Summary and Model Evaluation:

Throughout the exploration, we prioritized interpretability and accuracy. We tracked variable importance, pruned the tree for simplicity, and utilized cross-validation for model tuning. The resultant decision tree exhibited predictive capabilities and enhanced interpretability by highlighting key variables influencing instant bookability predictions.

The iterative refinement process, coupled with consistent attention to variable importance and model interpretability, underscored the balance between model complexity and predictive accuracy. The overall accuracy of the final model was evaluated on the test set, yielding an accuracy of 0.66. This comprehensive approach ensures that the decision tree not only predicts instant bookability effectively but also provides valuable insights into the factors influencing the predictions.

Step IV: Clustering

```r
# clustering analysis


library(ggplot2)
library(dplyr)
numeric_features <- cleaned_df %>% select("accommodates", "price", "minimum_nights","host_response_rate","host_acceptance_rate",
                                          "availability_365","number_of_reviews_ltm","review_scores_value")

# Standardization
scaled_data <- scale(numeric_features)

# Hierarchical Clustering

# Calculating the distance matrix
distance_matrix <- dist(scaled_data)

# Performing hierarchical clustering
hclust_model <- hclust(distance_matrix, method = "ward.D2")

# Cutting the tree to get clusters
num_clusters <- 3
clusters <- cutree(hclust_model, k = num_clusters)

# Interpreting and Evaluate Clusters
cleaned_df$cluster <- clusters

# Visualizations
plot(hclust_model, main = "Hierarchical Clustering Dendrogram", xlab = "Rental Units", sub = NULL)
```



II. Cluster Descriptions:

Cluster 1: Luxurious Retreats

Cluster 1 represents luxurious retreats with a notably high average price. These rental units are characterized by their premium features and unique amenities, attracting guests seeking an upscale and exclusive stay. The presence of more outliers in this cluster suggests a diverse range of high-end properties, and guests in this cluster tend to prefer accommodations with a higher number of beds, making it suitable for larger groups or families looking for a lavish experience.

Cluster 2: Urban Comfort

Cluster 2 encompasses urban comfort, offering rental units with a relatively high average price. These properties provide a comfortable and well-equipped urban living experience, attracting guests looking for a convenient yet quality stay. While still featuring high prices, the presence of fewer outliers indicates a more consistent pricing pattern. Guests in this cluster may appreciate multiple beds, though the emphasis is slightly less pronounced compared to Cluster 1, appealing to those seeking a mix of comfort and sophistication.

Cluster 3: Affordable Stays

Cluster 3 represents affordable stays, offering rental units with a less expensive price range. These accommodations cater to budget-conscious travelers seeking cost-effective options without compromising on basic amenities. With fewer outliers, this cluster suggests a more straightforward pricing structure. Guests in this cluster may find the standard number of beds suitable, and the emphasis is on affordability, making it an attractive choice for those prioritizing budget-friendly options.

PROCESS OF VARIABLE SELECTION AND MODEL BUILDING:

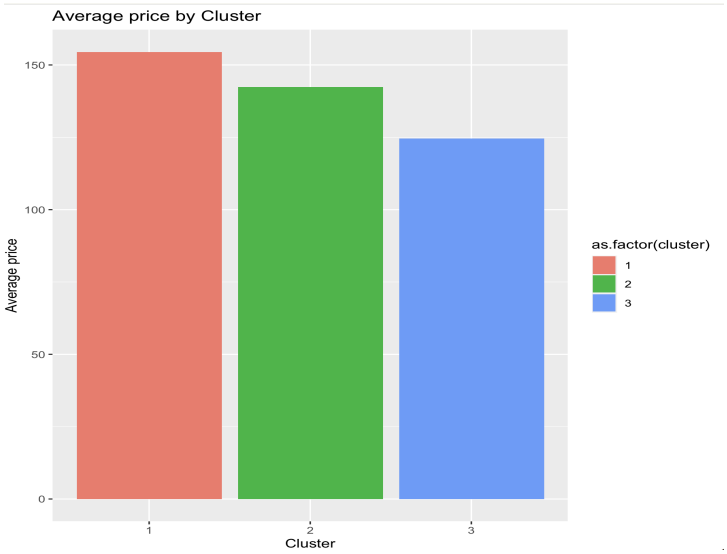
In selecting variables, we focused on key aspects of accommodation, pricing, host responsiveness, and guest reviews. These variables provide a comprehensive view of the rental units' characteristics and the overall hosting experience. We applied the hierarchical clustering model, using Ward's linkage method, to uncover natural groupings within the dataset. We determined the choice of three clusters by visually inspecting the dendrogram and aiming for a balance between granularity and interpretability.

This process allowed for the creation of distinct clusters that capture different aspects of the rental market within our neighborhood, providing valuable insights for property owners, hosts, and potential guests. The interpretation is based on a combination of accommodation capacity, pricing, host interaction metrics, and guest feedback, offering a nuanced understanding of the diverse offerings in the area.

III. Visualizations that describe your clustering model

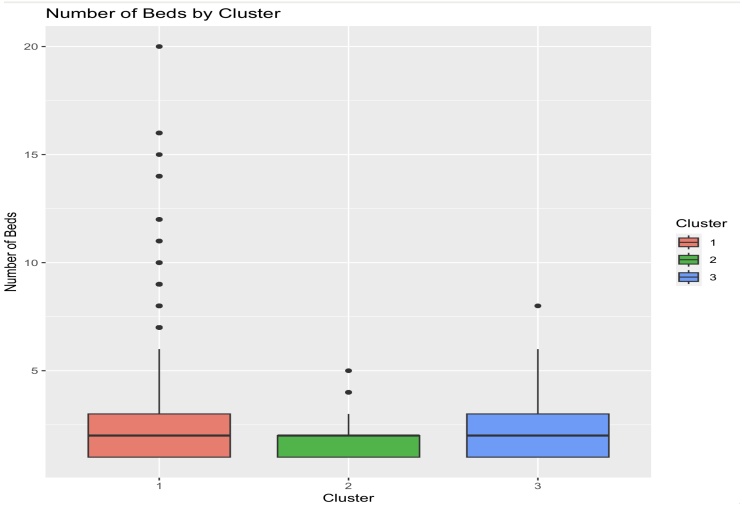
```r
#Part-iii
library(ggplot2)
library(dplyr)

# Barplot of Average Guest Review Scores by Cluster
cleaned_df %>%
  group_by(cluster) %>%
  summarise(avg_price = mean(price)) %>%
  ggplot(aes(x = as.factor(cluster), y = avg_price, fill = as.factor(cluster))) +
  geom_bar(stat = "identity") +
  labs(title = "Average price by Cluster", x = "Cluster", y = "Average price")

# Boxplot of Number of Beds by Cluster
ggplot(cleaned_df, aes(x = as.factor(cluster), y = beds, fill = as.factor(cluster))) +
  geom_boxplot() +
  labs(title = "Number of Beds by Cluster", x = "Cluster", y = "Number of Beds") +
  scale_fill_discrete(name = "Cluster")

#Violin Plot of Review Scores

ggplot(cleaned_df, aes(x = as.factor(cluster), y = review_scores_value, fill = as.factor(cluster))) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.1, fill = "white", color = "black") +
  labs(title = "Violin Plot of Review Scores by Cluster", x = "Cluster", y = "Review Scores") +
  scale_fill_discrete(name = "Cluster") +
  facet_wrap(~ as.factor(cluster), scales = "free_y", ncol = 1) +
  theme_minimal()
```

Bar Plot of Average Guest Review Scores by Cluster:



Description: The comparison graph of average prices among the three clusters reveals that cluster one has the highest average price, indicating that the houses in this cluster are more expensive compared to the other clusters. In contrast, the third cluster exhibits the lowest average price when compared to both cluster one and cluster two.

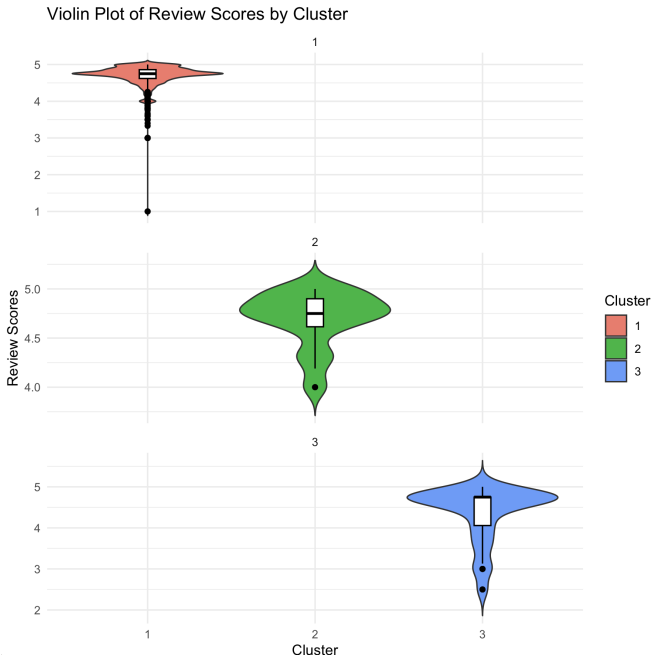
Boxplot of Number of Beds by Cluster:



Description: Boxplot of Number of Beds by Cluster:

The boxplot reveals that the first cluster exhibits a higher number of outliers compared to the other clusters. The presence of more outliers suggests a greater variability or spread in the number of beds within this cluster. Interestingly, the similarity in the box characteristics between the first and last clusters indicates a commonality or consistency in the distribution of the number of beds.The observation that the first cluster has more outliers could imply a preference among individuals in this cluster for rental units with a higher number of beds. The extended range of bed counts beyond the typical values seen in the other clusters suggests that people in the first cluster might prioritize or prefer accommodations with a greater capacity, possibly indicating a trend toward larger or more spacious living arrangements.

Violin Graph of Review scores by clusters:



The violin plots for the three clusters reveal distinctive patterns in their review scores. Cluster 2 exhibits a wide distribution of values, ranging from 4.5 to 5, which represents the highest scores

among the three clusters. Notably, no poor ratings are observable within Cluster 2. Cluster 1, although having high ratings, displays the smallest density among the clusters. It encompasses several outliers, contributing to a range spanning from 1 through 5. In contrast, Cluster 3 demonstrates the highest density within the range of 4 to 5, while its overall range extends from 2 to 5. These observations provide a nuanced understanding of the review score distributions within each cluster, highlighting the distinct characteristics of each group.

Step V: Conclusions

Throughout this assignment, we navigated a comprehensive process of data preprocessing, exploration, and analysis focused on Airbnb listing data. The journey began with cleaning the data, addressing missing values, and transforming variables into suitable formats. Our exploration phase unearthed valuable patterns, trends, and relationships within the dataset. Crucial decisions were made regarding variable selection, transformation, and managing multicollinearity, laying the foundation for creating multiple linear regression models. An important step involved log-transforming the price variable to better align with normality assumptions.

The assignment was challenging as it exposed us to real world data and the messy nature of the scraped dataset. We had to work together to understand the dataset and what each variable might represent. The data had many missing values and some of the columns had large amounts of text data (such as the amenities column). Once we cleaned the dataset and came to some sort of a conclusion about what each column might represent, we each worked on different parts of the assignment. Since our abilities compliment one another, this assignment was a great learning opportunity, underscoring the importance of teamwork and adaptability.

Our iterative approach to building and refining regression models through backward elimination enabled us to identify significant predictors and evaluate their impact on listing prices. By analyzing residuals, assessing multicollinearity, and considering various metrics, we gauged the models' performance and reliability. The insights derived from this data mining effort hold significance for a diverse range of stakeholders. Hosts and property managers can leverage these findings to optimize their listings, set competitive prices, and enhance overall guest satisfaction. Prospective Airbnb users can benefit by gaining a deeper understanding of the factors influencing pricing, empowering them to make informed choices based on preferences and budget constraints.

In this project, we also applied k-nearest neighbors (kNN) and Naive Bayes algorithms to predict key aspects of Airbnb listings. Using kNN, we successfully predicted the availability of wifi in rentals. For the Naive Bayes model, we categorized the continuous review_scores_rating into discrete bins for classification and selected predictors that encompassed both property features and host characteristics. The practical application of these models was demonstrated through the prediction of rental ratings for a fictional apartment, emphasizing the importance of thoughtful feature selection and the challenges of model complexity. Our exploration also extended to understanding the instantly bookable feature, where we employed a classification tree. Key steps included assessing variable importance, careful variable selection, and constructing a pruned tree. Cross-validation determined the optimal tree size, visualized using rpart.plot. The final model achieved 66% accuracy on the test set, striking a balance between interpretability and predictive performance. Our exploration of Airbnb data also delved into cluster analysis, revealing intriguing insights.

From a broader perspective, these data-driven insights can inform strategic decisions for Airbnb as a platform, contributing to the refinement of algorithms, user interfaces, and policy formulations. Policymakers and researchers may find value in the patterns uncovered, offering insights into the dynamics of short-term rental markets and potentially informing regulations or urban planning initiatives. Overall, our comprehensive analysis contributes not only to the understanding of Airbnb pricing dynamics but also to the broader field of data mining and its practical applications in the real-world context of the sharing economy.