Project: Analysis and Prediction of Airbnb Listing Prices

Data Importing

```
#data Exploration
df <-
read.csv('C:\\Users\\vashu\\OneDrive\\Desktop\\Programming\\R\\trial\\Airbnb_Open_
Data.csv')

column_names <- colnames(df)
print(column_names)

df_dimensions <- dim(df)
print(df_dimensions)
```

Output :

```
>print(df_dimensions)
[1] 102599     26

> print(column_names)
 [1] "id"                          "NAME"         "host_id"
 [4] "host_identity_verified"      "host_name"    "neighbourhood_group"
 [7] "neighbourhood"               "lat"          "long"
[10] "country"                     "country_code""instant_bookable"
[13] "cancellation_policy"         "room_type"    "Construction_year"
[16] "price"                       "service_fee" "minimum_nights"
[19] "number_of_reviews"           "last_review" "reviews_per_month"
[22] "review_rate_number" "calculated_host_listings_count" "availability.365"
[25] "house_rules"               "license"
```

Data Cleaning and Transformation:

Storing Revelant Data in new dataframe- Working Df

```
new_df <- df[, !(colnames(df) %in% c("host_name", "NAME" , "country" ,
"country_code",  "review_rate_number"  , "calculated_host_listings_count" ,
"availability.365" , "house_rules"  ,"license" ))]
working_df <- new_df
print(colnames(working_df))
```

Output:

```
working_df <- new_df
> print(colnames(working_df))
 [1] "id"                         "host_id"                "host_identity_verified"
"neighbourhood_group"       "neighbourhood"
 [6] "lat"                        "long"                   "instant_bookable"
"cancellation_policy"         "room_type"
[11] "Construction_year"          "price"                  "service_fee"
"minimum_nights"              "number_of_reviews"
[16] "last_review"                "reviews_per_month"
```

```
#converting char price to int
# Remove the dollar sign and any additional spaces
working_df$price <- gsub("[ $]", "", working_df$price)

working_df$price<- as.integer(working_df$price)
working_df$minimum_nights <- as.integer(working_df$minimum_nights)

working_df$service_fee <- gsub("[ $]", "", working_df$service_fee)

working_df$service_fee <- as.integer(working_df$service_fee)
```

```

# Count missing values in each column
missing_counts <- colSums(is.na(working_df))

# Print the result
print(missing_counts)
```

Output:

```

> # Print the result
> print(missing_counts)
id     host_id   host_identity_verified     neighbourhood_group       neighbourhood
0      0         289                        29                        16
lat                     long         instant_bookable      cancellation_policy          room_type
8                        8           105                    76                           0
Construction_year    price            service_fee          minimum_nights        number_of_reviews
214                   247             273                  409                   183
           last_review        reviews_per_month
                 15893                   15879
```

Data Cleaning :

```
### Replacing null values in "host_identity_verified" with
Unconfirmed assuming the they are the ones who are also not verified
users
working_df$host_identity_verified <-
ifelse(is.na(working_df$host_identity_verified), "unconfirmed",
working_df$host_identity_verified)



# Count missing values in each column
missing_counts <- colSums(is.na(working_df))

# Print the result
print(missing_counts)


table(working_df$neighbourhood_group)

## replacing manhatan to Manhattan and broklyn to Brooklyn
working_df$neighbourhood_group <-
ifelse((working_df$neighbourhood_group == 'manhatan') , "Manhattan"
,working_df$neighbourhood_group )
working_df$neighbourhood_group <-
ifelse((working_df$neighbourhood_group == 'brookln') , "Brooklyn"
,working_df$neighbourhood_group )
# check after replacing
table(working_df$neighbourhood_group)


table(working_df$neighbourhood)

temp = working_df  %>% group_by(neighbourhood_group , neighbourhood
) %>%

  summarise(total_count =n(),.groups = 'drop')

temp
```

```
### Replacing the null values of "neighbourhood group" with
"Brooklyn" because for most of the null neighbourhood the
"neighbourhood group" is "Brooklyn"


working_df$neighbourhood_group <-
ifelse(is.na(working_df$neighbourhood_group) , "Brooklyn"
,working_df$neighbourhood_group )
table(working_df$neighbourhood)



# Check the number of null values in each column
null_counts <- colSums(is.na(working_df))

# Print the null counts
print(null_counts)


neighborhoods <- c("Greenpoint", "Crown Heights", "East Village",
"West Village", "Elmhurst", "Flatiron District", "Upper West Side")

for (neighborhood in neighborhoods) {
  lat_mode <- mode(working_df$lat[working_df$neighbourhood ==
neighborhood])
  long_mode <- mode(working_df$long[working_df$neighbourhood ==
neighborhood])

  working_df$lat[working_df$neighbourhood == neighborhood &
is.na(working_df$lat)] <- lat_mode
  working_df$long[working_df$neighbourhood == neighborhood &
is.na(working_df$long)] <- long_mode
}



# Check the number of null values in each column
null_counts <- colSums(is.na(working_df))

# Print the null counts
print(null_counts)
```

```
table(working_df$instant_bookable)
### Replacing Null values with instant_bookable is false

working_df$instant_bookable <-
ifelse(is.na(working_df$instant_bookable) , "FALSE"
,working_df$instant_bookable )



# Check the number of null values in each column
null_counts <- colSums(is.na(working_df))

# Print the null counts
print(null_counts)

# Filling "cancellation_policy" null with "Moderate" as for False
"instant_bookable" the max value counts is "Moderate"
working_df$cancellation_policy <-
ifelse(is.na(working_df$cancellation_policy) , "moderate"
,working_df$cancellation_policy )



# Check the number of null values in each column
null_counts <- colSums(is.na(working_df))

# Print the null counts
print(null_counts)

# fill na in neighbourhood
# Replacing Null value wrt "Brooklyn" with "Bedford-Stuyvesant" and
"Manhattan" with "Two Bridges"
working_df$neighbourhood[working_df$neighbourhood_group ==
"Brooklyn"] <-
ifelse(is.na(working_df$neighbourhood[working_df$neighbourhood_group
== "Brooklyn"]), "Bedford-Stuyvesant",
working_df$neighbourhood[working_df$neighbourhood_group ==
"Brooklyn"])
working_df$neighbourhood[working_df$neighbourhood_group ==
"Manhattan"] <-
ifelse(is.na(working_df$neighbourhood[working_df$neighbourhood_group
== "Manhattan"]), "Two Bridges",
```

```
working_df$neighbourhood[working_df$neighbourhood_group ==
"Manhattan"])


# Check the number of null values in each column
null_counts <- colSums(is.na(working_df))

# Print the null counts
print(null_counts)


#finding mean of price AND SERVICE FEE for filling na values


working_df$price <- ifelse(is.na(working_df$price) , 0
,working_df$price )
working_df$service_fee <- ifelse(is.na(working_df$service_fee) , 0
,working_df$service_fee)

price_mean =  mean(working_df$price)
price_mean
service_fee_mean =  mean(working_df$service_fee)
service_fee_mean
working_df$price <- ifelse((working_df$price == 0) , price_mean
,working_df$price )
working_df$service_fee <- ifelse((working_df$service_fee == 0 ) ,
service_fee_mean ,working_df$service_fee)

# filling na for minimum_nights

mode_value <- mode(working_df$minimum_nights)
mode_value
#fill with mode value
working_df$minimum_nights <- ifelse(is.na(working_df$minimum_nights)
, mode_value ,working_df$minimum_nights)





# filling na of  number of review with  median value of all col
```

```
working_df$number_of_reviews <-
ifelse(is.na(working_df$number_of_reviews) , 0
,working_df$number_of_reviews )
median =  median(working_df$number_of_reviews)
median

working_df$number_of_reviews <- ifelse((working_df$number_of_reviews
 ==  0 ) , median ,working_df$number_of_reviews )

# filling last_review to no review
working_df$last_review <- ifelse(is.na(working_df$last_review) , "No
 Review" ,working_df$last_review )

# filling na of construction to no available
working_df$Construction_year <-
ifelse(is.na(working_df$Construction_year) , "Not Available"
,working_df$Construction_year)



# fillin na of review_per_month with mean value

# Calculate the mean of the 'reviews_per_month' column
mean_value <- mean(working_df$reviews_per_month, na.rm = TRUE)
mean_value
# Replace NA values in the 'reviews_per_month' column with the mean
value
working_df$reviews_per_month[is.na(working_df$reviews_per_month)] <-
mean_value


# Check the number of null values in each column
null_counts <- colSums(is.na(working_df))
print(null_counts)
```

Output:

```
id     host_id   host_identity_verified      neighbourhood_group       neighbourhood
0      0         289                         29                        16
lat                        long       instant_bookable     cancellation_policy         room_type
8                          8          105                      76                          0
Construction_year    price              service_fee          minimum_nights        number_of_reviews
214                   247               273                  409              183
last_review          reviews_per_month
15893                15879
```

EDA-Exploratory Data Analysis

```
str(working_df)

summary(working_df)
```

Output:

```
 id                  host_id
Min.    :1001254    Min.    :1.236e+08
1st Qu.:15085814    1st Qu.:2.458e+10
Median :29136603    Median :4.912e+10
Mean    :29146235    Mean    :4.925e+10
3rd Qu.:43201198    3rd Qu.:7.400e+10
Max.    :57367417    Max.    :9.876e+10



 host_identity_verified   neighbourhood_group
 Length:102599             Length:102599
Class :character          Class :character
Mode  :character          Mode  :character

neighbourhood         lat
Length:102599       Length:102599
Class :character    Class :character
Mode  :character    Mode  :character


long
Length:102599
Class :character
Mode  :character
Mean    :508.3
3rd Qu.:709.0
Max.    :999.0

instant_bookable
Length:102599
Class :character
Mode  :character
```

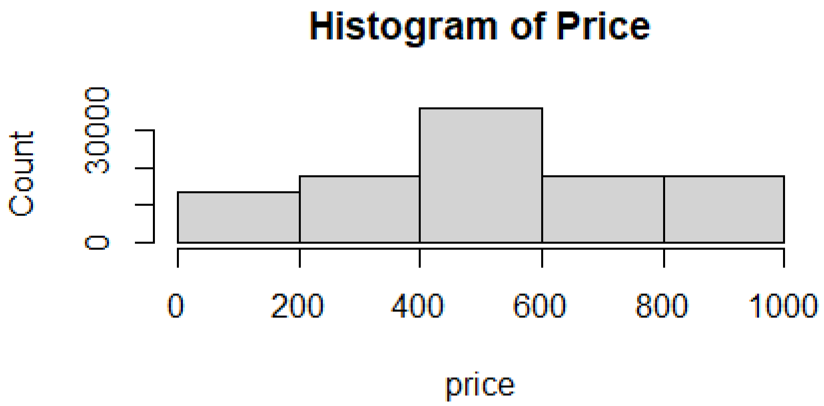
```
Mean    :125.0
3rd Qu.:182.0
Max.    :240.0


cancellation_policy
Length:102599
Class :character
Mode  :character


room_type
 Length:102599
 Class :character
 Mode  :character


Construction_year
Length:102599
Class :character
Mode  :character

price
Min.    : 50.0
1st Qu.:341.0
Median :431.9

service_fee
Min.    : 10.0
1st Qu.: 68.0
Median :124.7

number_of_reviews
Min.    :   1.00
1st Qu.:   4.00
Median :   7.00

last_review
Length:102599
Class :character
Mode  :character

minimum_nights
```

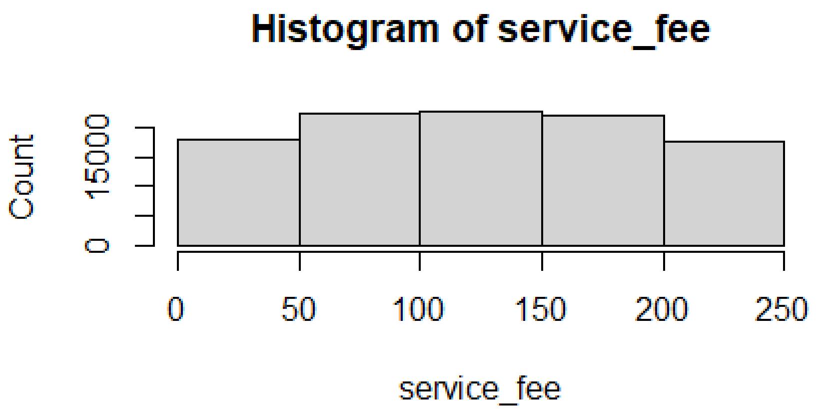
```
Length:102599
Class :character
Mode  :character
Mean   :   28.52
3rd Qu.:   30.00
Max.   :1024.00

 reviews_per_month
 Min.   : 0.010
 1st Qu.: 0.280
 Median : 1.050
 Mean   : 1.374
 3rd Qu.: 1.710
Max.   :90.000
```

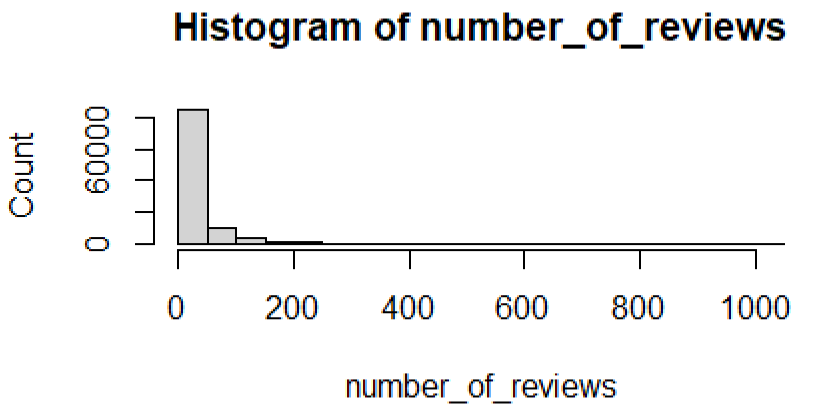
Visualizations

```
# hist graphs
# histogram plot

hist(working_df$price, breaks =4, main = "Histogram of Price", xlab
= "price", ylab = "Count")

hist(working_df$service_fee, breaks =4, main = "Histogram of
service_fee", xlab = "service_fee", ylab = "Count")

hist(working_df$number_of_reviews, breaks =20, main = "Histogram of
number_of_reviews", xlab = "number_of_reviews", ylab = "Count")
```

output:


Histogram of Price
Count
0
30000
0
200
400
600
800
1000
price

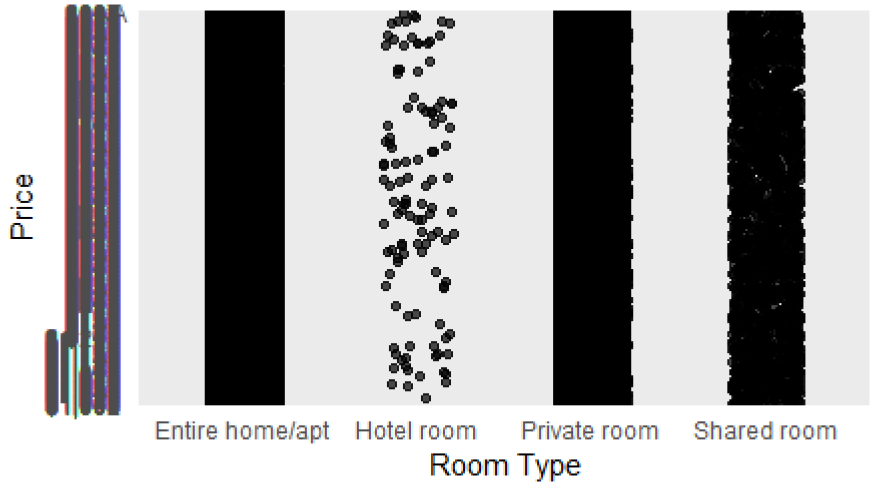

Histogram of service_fee
Count
0
15000
0
50
100
150
200
250
service_fee

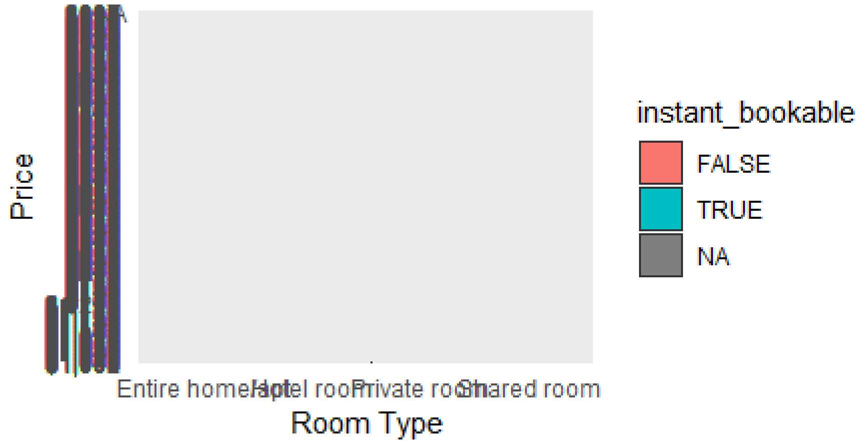

Histogram of number_of_reviews
Count
0
60000
0
200
400
600
800
1000
number_of_reviews

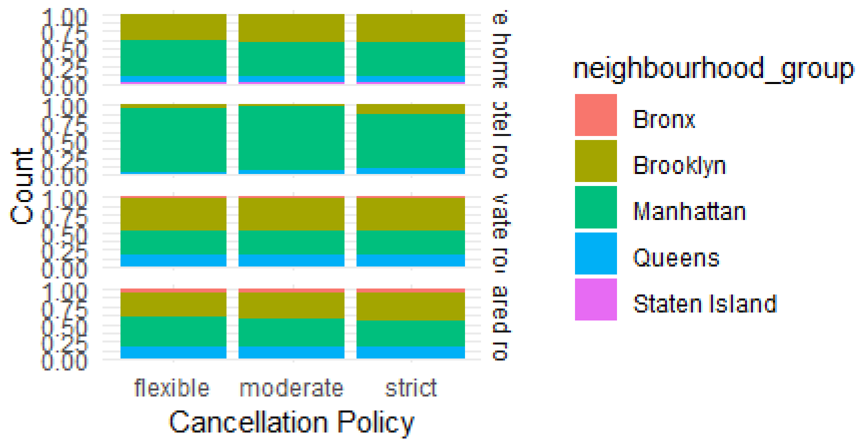
```
# 1.Relations Plots

# Create a scatter plot using ggplot2
ggplot(df, aes(x = price, y = number_of_reviews, color = room_type))
+
  geom_point() +
  labs(x = "Price", y = "Number of Reviews") + theme_minimal()

ggplot(df, aes(x = price, y = number_of_reviews, color = room_type))
+
  geom_line() +
  labs(x = "Price", y = "Number of Reviews") + theme_minimal()

# 2. Distribution Plots

ggplot(data = df, aes(x = review_rate_number, fill =
neighbourhood_group)) +
  geom_histogram(binwidth = 1, position = "identity", alpha = 0.5) +
  labs(x = "Review Rate Number", y = "Count") +
  theme_minimal()


# 3. Categorical Plots

ggplot(data = df, aes(x = room_type, y = price)) +
  geom_point(position = position_jitter(width = 0.2, height = 0),
alpha = 0.7) +
  labs(x = "Room Type", y = "Price") +
  theme_minimal()

ggplot(data = df, aes(x = room_type, y = price, fill =
instant_bookable)) +
  geom_violin() +
  labs(x = "Room Type", y = "Price") +
  theme_minimal()


ggplot(data = working_df, aes(x = cancellation_policy, fill =
neighbourhood_group)) +
  geom_bar(position = "fill") +
  facet_grid(room_type ~ .) +
  labs(x = "Cancellation Policy", y = "Count") +
```

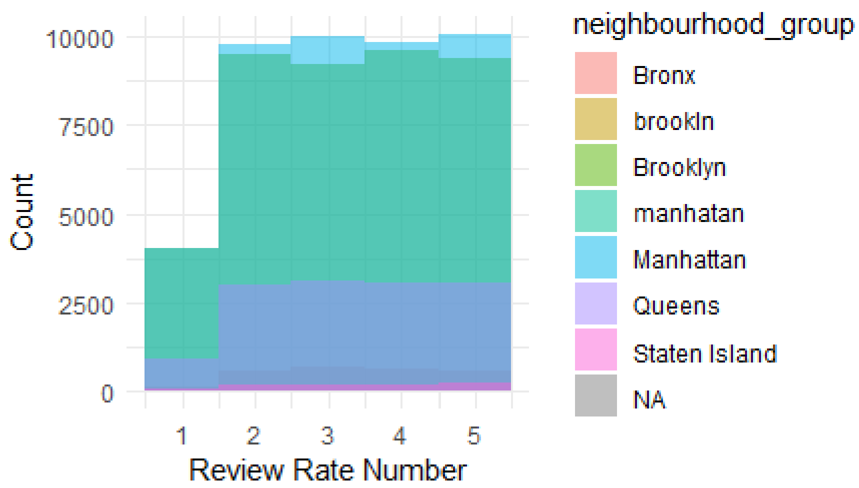
```
theme_minimal()
```

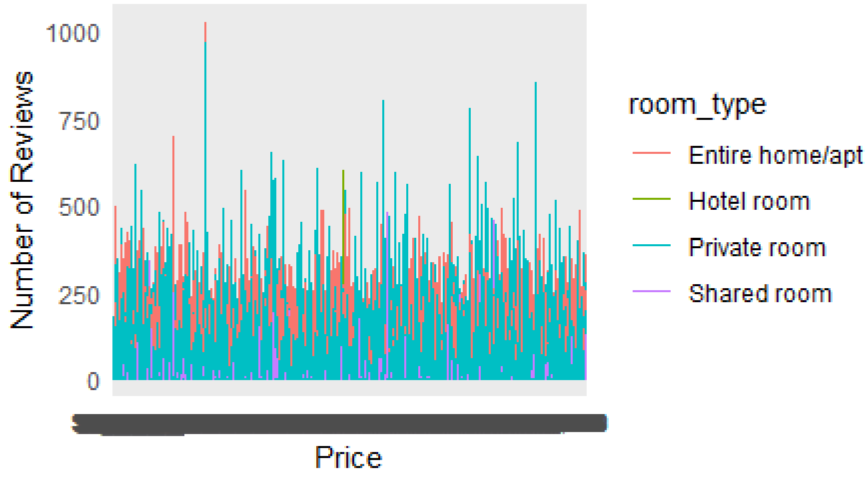
Output:











Feature Engineering:

Calculate Price Per Night:

```
# Calculate price per night

working_df$price<- as.integer(working_df$price)
working_df$minimum_nights <- as.integer(working_df$minimum_nights)

working_df$price_per_night <- working_df$price /
working_df$minimum_nights
working_df$price_per_night
```

Sample Output:

```
   [1]    96.6000000     4.7333333   206.6666667    12.2666667    20.4000000
192.3333333     1.5777778     9.5777778   215.5000000
```

Finding Distance from a popular Landmark

```
# Function to calculate distance using Haversine formula
haversine_distance <- function(lat1, lon1, lat2, lon2) {
  # Convert degrees to radians
  lat1_rad <- lat1 * pi / 180
  lon1_rad <- lon1 * pi / 180
  lat2_rad <- lat2 * pi / 180
  lon2_rad <- lon2 * pi / 180

  # Radius of the Earth in kilometers
  radius <- 6371

  # Haversine formula
  dlat <- lat2_rad - lat1_rad
  dlon <- lon2_rad - lon1_rad
  a <- sin(dlat/2)^2 + cos(lat1_rad) * cos(lat2_rad) * sin(dlon/2)^2
  c <- 2 * atan2(sqrt(a), sqrt(1-a))
  distance <- radius * c

  return(distance)
}

# Assuming your dataset is named 'airbnb' and latitude and longitude
columns are 'lat' and 'long'
landmark_lat <- 143
  landmark_lon <- 233

  # Calculate the distance from the landmark for each location
  working_df$distance_from_landmark <-
haversine_distance(as.numeric(working_df$lat),as.numeric(working_df$
long), landmark_lat, landmark_lon)
```

Output:

```
   [1] 9831.759 9823.287 9816.295 9827.857 9817.307 9823.325 9827.504
9827.504 9822.385 9818.276 9827.082 9818.062 9822.939 ……….
```

Modeling :

Split Dataset:

```
  # Split the data into a training set and a testing set
  train_indices <- createDataPartition(working_df$price, p = 0.7,
list = FALSE)
  training_set <- working_df[train_indices, ]
  testing_set <- working_df[-train_indices, ]
  # 70 percent for training and 30 for testing
```

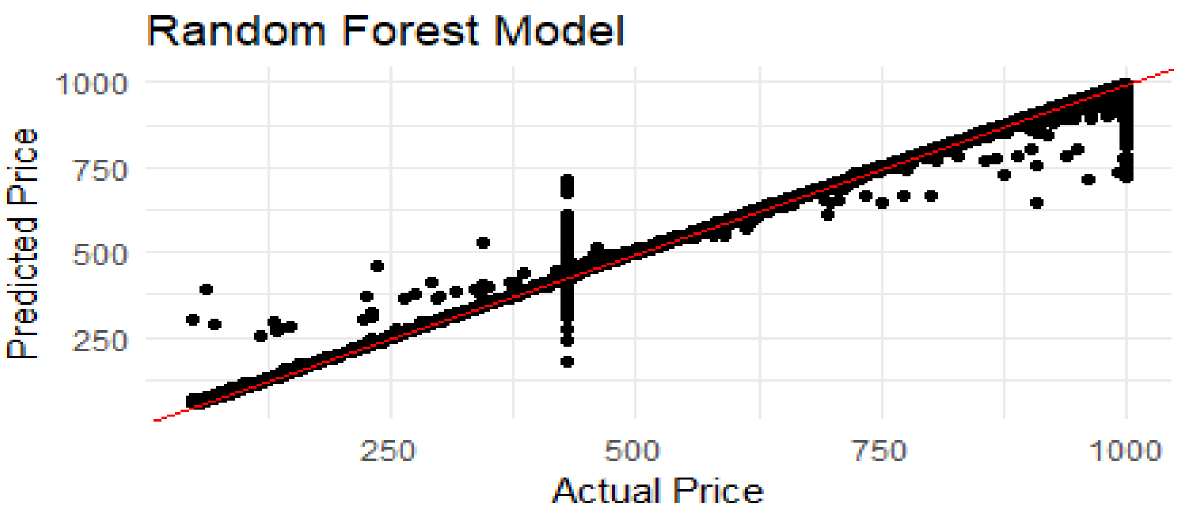
Random Forest Regression Model

```
 # random forest
  # Train the Random Forest model with handling missing values
  rf_model <- randomForest(price ~ ., data = training_set, na.action
= na.exclude)
  rf_model
  # Make predictions on the testing set
  rf_predictions <- predict(rf_model, newdata = testing_set)

  # Subset the testing set to align with the predictions
  aligned_testing_set <- testing_set[!is.na(rf_predictions), ]

  # Subset the predictions to align with the testing set
  aligned_predictions <- rf_predictions[!is.na(rf_predictions)]
  rf_rmse <- caret::RMSE(aligned_predictions,
aligned_testing_set$price)

  # Print the RMSE
  print(paste("Random Forest RMSE:", rf_rmse))
```

Output:

```
Call:
 randomForest(formula = price ~ ., data = training_set, na.action =
na.exclude)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 6

          Mean of squared residuals: 171.772
                    % Var explained: 99.73

"Random Forest RMSE: 10.4225547187656"
```

Visualization Random Forest Regression model



Model Evaluation:

```
  # Calculate R-squared
  rf_r_squared <- caret::R2(aligned_predictions,
aligned_testing_set$price)

  # Calculate Mean Absolute Error (MAE)
  rf_mae <- caret::MAE(aligned_predictions,
aligned_testing_set$price)

  # Calculate Mean Percentage Error (MAPE)
  rf_mape <- mean(abs((aligned_predictions -
aligned_testing_set$price) / aligned_testing_set$price)) * 100

  # Print the evaluation metrics
  print(paste("Random Forest R-squared:", rf_r_squared))
  print(paste("Random Forest MAE:", rf_mae))
  print(paste("Random Forest MAPE:", rf_mape))
```

Output:

```
print(paste("Random Forest R-squared:", rf_r_squared))
[1] "Random Forest R-squared: 0.997171891519946"
>   print(paste("Random Forest MAE:", rf_mae))
[1] "Random Forest MAE: 2.71069366528757"
>   print(paste("Random Forest MAPE:", rf_mape))
[1] "Random Forest MAPE: 0.724678384638143"
```

Conclusion

The goal of the project is to analyze and predict Airbnb listing prices. The initial steps involve data exploration and cleaning, including handling missing values and transforming data types. Exploratory data analysis is conducted to understand the relationships between variables and identify patterns. Feature engineering techniques are applied to create new variables, such as price per night and distance from a popular landmark. The ultimate objective is to develop a predictive model that can accurately estimate Airbnb listing prices.