Seattle Airbnb Listings HarvardX Capstone Project

Christian McKinnon

8/3/2020

Executive Summary:

It was back in October of 2007 that Joe Gebbia and Brian Chesky decided to throw an air mattress in their living room and set up an "air bed and breakfast" for guests arriving in San Francisco for a major convention. From this lightbulb moment, the founders managed to attract investment and expand their vision to a global empire with a 2020 private valuation of USD $26 billion. Travelers now have the option of staying at short-term vacation rentals or "airbnbs" in many parts of the world, creating a particularly efficient market for hosts who need their listings to be priced competitively. Major cosmopolitan cities like Seattle approaching nearly 10,000 listings will certainly create a demand for accurate price prediction and forecasting.

Our analysis will highlight specific elements of the "Seattle Airbnb Listing" Dataset from 2018 which features 7,576 listings with 18 variables, scraped and maintained by Tom Slee, and available for download on Kaggle.

In this report for the the Capstone course of the HarvardX Professional Certificate in Data Science (PH125.9x), we will begin by examining the unrefined data and perform any necessary tidying of the dataset. We will then determine which features provide most insight into price prediction and remove those that only serve to add noise. Data visualization and exploratory data analysis will then be performed on the tidied dataset to inform our modeling. Several machine learning algorithms will be applied to improve our RMSE (the root mean square error) from our baseline to our final model.

Methods and Analysis:

First we begin with data preparation and loading the data from Kaggle. After the data has been loaded successfully, we will proceed to clean the data by checking it for NAs and other errors such as misspellings or incorrect categorizations or associations.

Once the NAs have been removed and the data tidied, we proceed to our exploratory data analysis where we examine correlations between the price per night and the other features. Visualizations will illuminate these connections and confirm our hypotheses.

Modeling will involve the partitioning of our dataset "airbnb" into training, validation, and test sets. In this report, the "validation" set will be used in conjunction with the training set. This is in contrast to the terminology used in the MovieLens Project, but more in line with contemporary machine learning conventions. We will start with a baseline model based on the median price of the dataset and move on to more advanced algorithms including linear models, elastic net regression, regression trees, random forest models, kNN, and neural nets all tuning with the training set "airbnb_train". Models will be evaluated based on which produces the lowest RMSE.

Mathematically, the RMSE is the standard deviation of the prediction errors (residuals) and is used to measure the difference between observed and predicted values. The advantage of using the RMSE is that its

unit is the same as the unit being measured (in this case the price in USD). The model that produces the lowest RMSE on the validation set will then be run on the test set.

Data Preparation and Required Packages:

We will begin by loading the following libraries: tidyverse, readr, data.table, icesTAF, caret, lubridate, glmnet, scales, stringr, dplyr, ggmap, ggcorrplot, treemapify, rpart, nnet, formatR, rmarkdown, and knitr with the "pacman" package. (If a package below is missing, p_load will automatically download it from CRAN).

```
if (!require(pacman)) install.packages("pacman", repos = "http://cran.us.r-project.org")
library(pacman)
pacman::p_load(tidyverse, readr, data.table, icesTAF, caret, lubridate, ggthemes,
    glmnet, scales, stringr, dplyr, ggmap, ggcorrplot, treemapify, rpart, nnet, formatR,
    rmarkdown, knitr)
```

Data Load and Preparation:

```
# Download the Dataset:
if (!dir.exists("SAirbnb")) mkdir("SAirbnb")
if (!file.exists("./SAirbnb/seattle_01.csv")) download.file("https://raw.githubusercontent.com/christian
    "./SAirbnb/seattle_01.csv")
# Read the Data:
suppressWarnings(airbnb <- read_csv("./SAirbnb/seattle_01.csv"))
# Set the number of significant digits to 4
options(digits = 4)
```

Preliminary Data Exploration and Cleaning:

Check the dimensions of the dataset:

```
dim(airbnb)
```

```
## [1] 7576   18
```

There are 7576 observations and 18 features

```
str(airbnb)
```

```
## tibble [7,576 x 18] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ X1                 : num [1:7576] 0 1 2 3 4 5 6 7 8 9 ...
##  $ room_id            : num [1:7576] 2318 3335 4291 5682 6606 ...
##  $ host_id            : num [1:7576] 2536 4193 35749 8993 14942 ...
##  $ room_type          : chr [1:7576] "Entire home/apt" "Entire home/apt" "Private room" "Entire hom
##  $ address            : chr [1:7576] "Seattle, WA, United States" "Seattle, WA, United States" "Sea
##  $ reviews            : num [1:7576] 21 1 63 462 134 130 401 35 36 76 ...
```

```
##  $ overall_satisfaction: num [1:7576] 5 NA 4.5 5 4.5 4.5 5 5 5 4.5 ...
##  $ accommodates        : num [1:7576] 8 4 2 2 2 2 2 4 3 4 ...
##  $ bedrooms            : num [1:7576] 4 2 1 0 1 1 1 2 2 1 ...
##  $ bathrooms           : num [1:7576] 2.5 1 1 1 1 3 1 1 1 1 ...
##  $ price               : num [1:7576] 250 100 82 49 90 65 78 165 95 115 ...
##  $ last_modified       : POSIXct[1:7576], format: "2018-12-20 03:46:14" "2018-12-20 04:08:45" ...
##  $ latitude            : num [1:7576] 47.6 47.5 47.7 47.5 47.7 ...
##  $ longitude           : num [1:7576] -122 -122 -122 -122 -122 ...
##  $ location            : chr [1:7576] "0101000020E6100000D449B6BA9C925EC0416326512FCE4740" "01010000
##  $ name                : chr [1:7576] "Casa Madrona - Urban Oasis, 1 block from the Park!" "Sweet Sea
##  $ currency            : chr [1:7576] "USD" "USD" "USD" "USD" ...
##  $ rate_type           : chr [1:7576] "nightly" "nightly" "nightly" "nightly" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   X1 = col_double(),
##   ..   room_id = col_double(),
##   ..   host_id = col_double(),
##   ..   room_type = col_character(),
##   ..   address = col_character(),
##   ..   reviews = col_double(),
##   ..   overall_satisfaction = col_double(),
##   ..   accommodates = col_double(),
##   ..   bedrooms = col_double(),
##   ..   bathrooms = col_double(),
##   ..   price = col_double(),
##   ..   last_modified = col_datetime(format = ""),
##   ..   latitude = col_double(),
##   ..   longitude = col_double(),
##   ..   location = col_character(),
##   ..   name = col_character(),
##   ..   currency = col_character(),
##   ..   rate_type = col_character()
##   .. )
```

We note that there 18 features of numeric, character, and POSIXct classes. We also notice that there is already one visible NA value in the "overall_satisfaction" feature.

Data Cleaning:

Coerce airbnb tibble into a data frame for later partitioning purposes:

```
airbnb <- as.data.frame(airbnb)
class(airbnb)
```

```
## [1] "data.frame"
```

Examine whether the data is tidy and if there are indeed any NAs:

```
sum(is.na(airbnb))
```

```
## [1] 1475
```

There are 1475 NAs in the dataset.

Check which features have NAs:

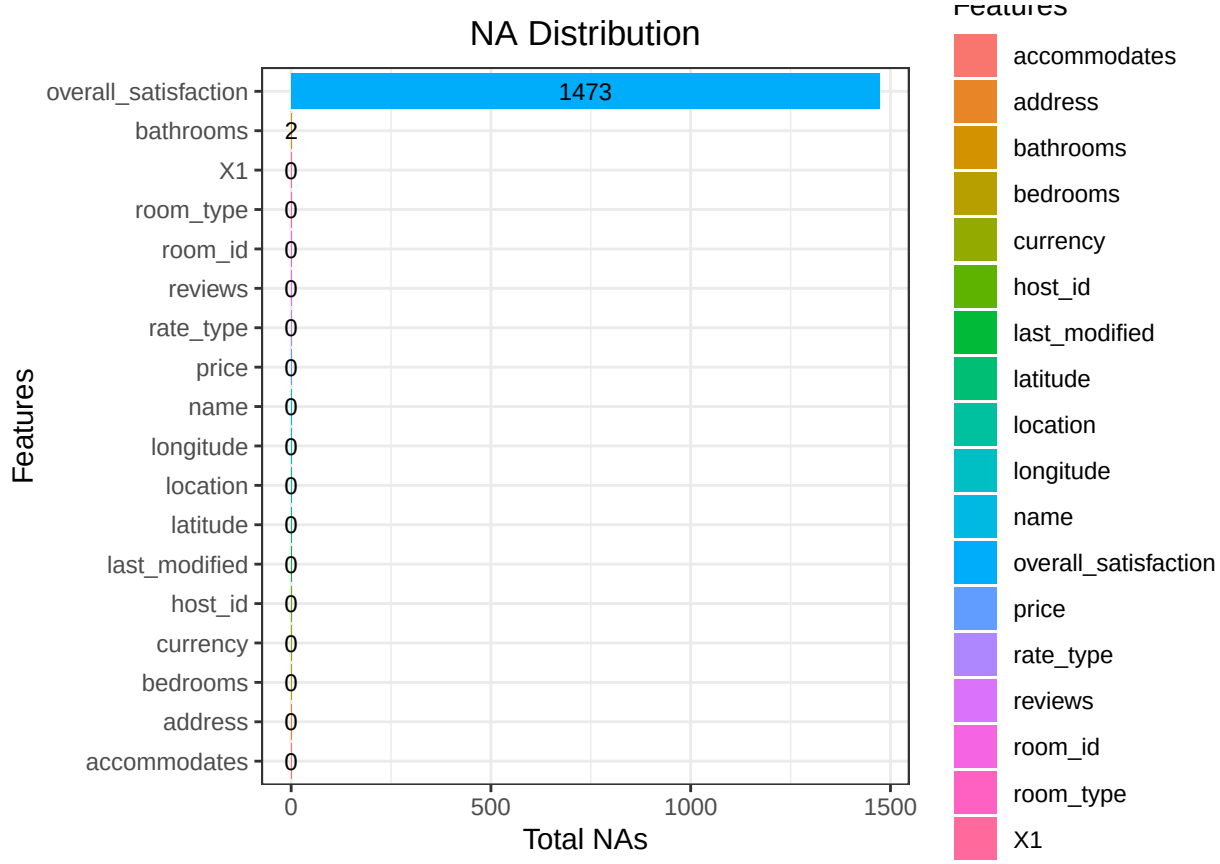
```
colSums(is.na(airbnb))
```

```
##                   X1              room_id              host_id
##                    0                    0                    0
##            room_type              address              reviews
##                    0                    0                    0
## overall_satisfaction         accommodates             bedrooms
##                 1473                    0                    0
##            bathrooms                price        last_modified
##                    2                    0                    0
##             latitude            longitude             location
##                    0                    0                    0
##                 name             currency            rate_type
##                    0                    0                    0
```

We find that "overall_satisfaction" has 1473 NAs while bathrooms has 2.

Create a dataframe na_bar to plot the NAs:

Now let's observe the NA distribution visually:



The visualization confirms there are only two features with missing values and that the all other observations are present.

NA Removal:

After confirming that there are only 2 features with NAs, (overall_satisfaction & bathrooms) we proceed to clean the dataset. As overall_satisfaction has 1473 NAs, assigning a value of 0 will significantly skew the ratings negatively. Therefore we will fill the values with the mean to provide for more accurate predictive pricing in our models.

Convert NAs to the mean value.

```
airbnb$overall_satisfaction[is.na(airbnb$overall_satisfaction)] <- mean(airbnb$overall_satisfaction,
    na.rm = TRUE)
```

The mean is roughly 4.84.

```
mean(airbnb$overall_satisfaction)
```

```
## [1] 4.841
```

Confirm the absence of NAs in this feature:

```
head(airbnb$overall_satisfaction)
```

```
## [1] 5.000 4.841 4.500 5.000 4.500 4.500
```

For the bathrooms feature, there are only 2 NAs and so we set them to zero.

```
airbnb <- airbnb %>% replace_na(list(bathrooms = 0))
```

Now confirm the absence of any NAs in the dataset:

```
sum(is.na(airbnb))
```

```
## [1] 0
```

Feature Exploration and Selection:

```
names(airbnb)
```

```
##  [1] "X1"                   "room_id"             "host_id"
##  [4] "room_type"            "address"             "reviews"
##  [7] "overall_satisfaction" "accommodates"        "bedrooms"
## [10] "bathrooms"            "price"               "last_modified"
## [13] "latitude"             "longitude"           "location"
## [16] "name"                 "currency"            "rate_type"
```

There are 18 features and those less related to price prediction will be dropped to refine our EDA and Modeling Focus:

Feature: "X1"

```
head(airbnb$X1)
```

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

"X1" is simply a numerical list for the dataset.

Features: "room_id" & "host_id"

```
head(airbnb$room_id)
```

```
## [1] 2318 3335 4291 5682 6606 9419
```

```
head(airbnb$host_id)
```

```
## [1]  2536  4193 35749  8993 14942 30559
```

"room_id" & "host_id" are simply numbers arbitrarily assigned to identify rooms and hosts.

Feature: "address"

Check for unique values of the feature "address"

```
airbnb %>% select(address) %>% distinct()
```

```
##                                       address
## 1                   Seattle, WA, United States
## 2                  Kirkland, WA, United States
## 3                  Bellevue, WA, United States
## 4                   Redmond, WA, United States
## 5              Mercer Island, WA, United States
## 6                                 Seattle, WA
## 7                    Renton, WA, United States
## 8           Ballard, Seattle, WA, United States
## 9               West Seattle, WA, United States
## 10                   Medina, WA, United States
## 11 <U+897F><U+96C5><U+56FE>, WA, United States
## 12                Newcastle, WA, United States
## 13                 Seattle , WA, United States
## 14          Ballard Seattle, WA, United States
## 15             Yarrow Point, WA, United States
## 16               Clyde Hill, WA, United States
## 17                  Tukwila, WA, United States
## 18  Seattle, Washington, US, WA, United States
## 19    Capitol Hill, Seattle, WA, United States
## 20                Kirkland , Wa, United States
## 21              Hunts Point, WA, United States
## 22                  Seattle, DC, United States
## 23                      Seattle, United States
## 24                   Vashon, WA, United States
## 25                Kirkland , WA, United States
## 26                  Bothell, WA, United States
## 27              Washington, WA, United States
```

Note that there are 27 values with different formats and 12 repeated instances of "Seattle." Any neighborhood of Seattle, the Chinese language version of "Seattle" and listings with only the State of Washington, will all be converted to Seattle. "WA" and "United States" will also be removed as they are redundant.

Replace the Chinese version of "Seattle" separately using regex:

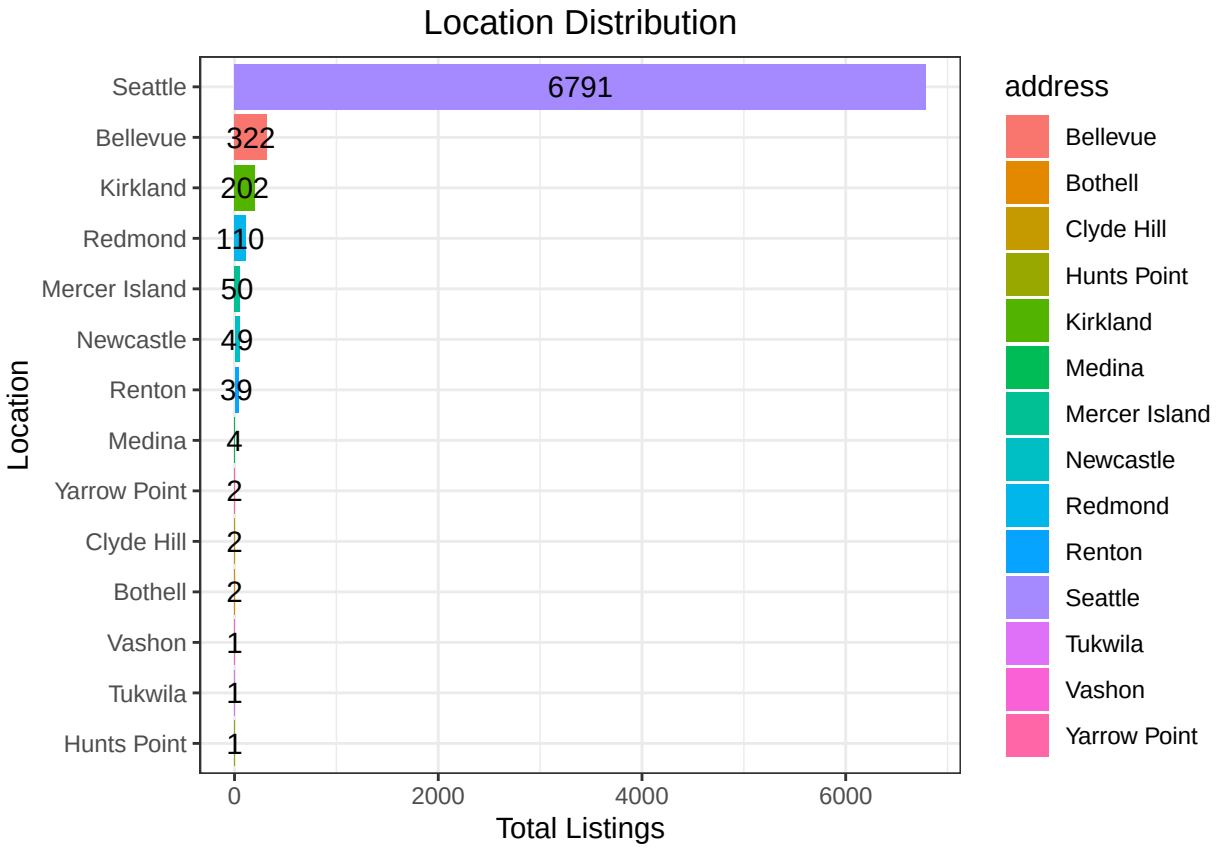
Reassign the column to the feature "address":

```
airbnb$address <- gsub("Seattle, United States", "Seattle", gsub("Seattle United States",
    "Seattle", address_clean2))
```

Now confirm there are only 14 different cities and sort by the greatest numbers of listings:

```
## # A tibble: 14 x 2
##    address       listing_sum
##    <chr>               <int>
##  1 Seattle              6791
##  2 Bellevue              322
##  3 Kirkland              202
##  4 Redmond               110
##  5 Mercer Island          50
##  6 Newcastle              49
##  7 Renton                 39
##  8 Medina                  4
##  9 Bothell                 2
## 10 Clyde Hill              2
## 11 Yarrow Point            2
## 12 Hunts Point             1
## 13 Tukwila                 1
## 14 Vashon                  1
```

###: Let's explore a data visualization to confirm this:



Note: As the remaining locations are all cities, the feature "address" will later be renamed "city."

It is clear the vast majority of listings are in Seattle (6791).

Feature: "last_modified"

The "last_modified" feature refers to the date a listing was updated and nearly all values occur on 2018-12-20, telling us very little about the data, therefore this feature will be removed.

```
head(airbnb$last_modified)
```

```
## [1] "2018-12-20 03:46:14 UTC" "2018-12-20 04:08:45 UTC"
## [3] "2018-12-20 03:04:19 UTC" "2018-12-20 04:11:25 UTC"
## [5] "2018-12-20 03:12:38 UTC" "2018-12-20 04:08:20 UTC"
```

Feature: "location"

The "location" feature will be removed in favor of using "latitude" & "longitude."

```
head(airbnb$location)
```

```
## [1] "0101000020E6100000D449B6BA9C925EC0416326512FCE4740"
## [2] "0101000020E61000006FBBD05CA7915EC04DF564FED1C34740"
## [3] "0101000020E6100000BDAB1E300F945EC0FB93F8DC09D84740"
## [4] "0101000020E6100000FCC7427408975EC009E1D1C611C34740"
## [5] "0101000020E6100000D47D00529B955EC07782FDD7B9D34740"
## [6] "0101000020E6100000145D177E70945EC0522B4CDF6BC64740"
```

Feature: "name"

The "name" feature will be removed as it is a categorical description of each listing.

```
head(airbnb$name)
```

```
## [1] "Casa Madrona - Urban Oasis, 1 block from the Park!"
## [2] "Sweet Seattle Urban Homestead 2 Bdr"
## [3] "Sunrise in Seattle Master Suite"
## [4] "Cozy Studio, min. to downtown -WiFi"
## [5] "Fab, private seattle urban cottage!"
## [6] "Glorious sun room w/ memory foambed"
```

Feature: "currency"

The "currency" feature will be dropped as all rates are in US Dollars.

```
airbnb %>% select(currency) %>% distinct()
```

```
##   currency
## 1      USD
```

Feature: "rate_type"

Check for distinct values of "rate_type":

```
airbnb %>% select(rate_type) %>% distinct()
```

```
##   rate_type
## 1   nightly
```

After confirming only one unique value, "nightly," we determine this feature can be removed.

Create the cleaned dataset:

Remove the above mentioned features and rename the columns:

Reorder the columns:

Confirm the features have been tidied and reordered with only 10 features:

```
names(airbnb)
```

```
##  [1] "price"        "city"         "rating"       "reviews_sum"  "room_type"
##  [6] "bedrooms"     "bathrooms"    "accommodates" "latitude"     "longitude"
```

Check the first few values of the cleaned dataset:

```
head(airbnb)
```

```
##   price    city rating reviews_sum       room_type bedrooms bathrooms
## 1   250 Seattle  5.000          21 Entire home/apt        4       2.5
## 2   100 Seattle  4.841           1 Entire home/apt        2       1.0
## 3    82 Seattle  4.500          63    Private room        1       1.0
## 4    49 Seattle  5.000         462 Entire home/apt        0       1.0
## 5    90 Seattle  4.500         134 Entire home/apt        1       1.0
## 6    65 Seattle  4.500         130    Private room        1       3.0
##   accommodates latitude longitude
## 1            8    47.61    -122.3
## 2            4    47.53    -122.3
## 3            2    47.69    -122.3
## 4            2    47.52    -122.4
## 5            2    47.65    -122.3
## 6            2    47.55    -122.3
```

Explanatory Data Analysis:

Now we will begin analyzing the features of our dataset to inform our price prediction modeling approach.

Correlogram:

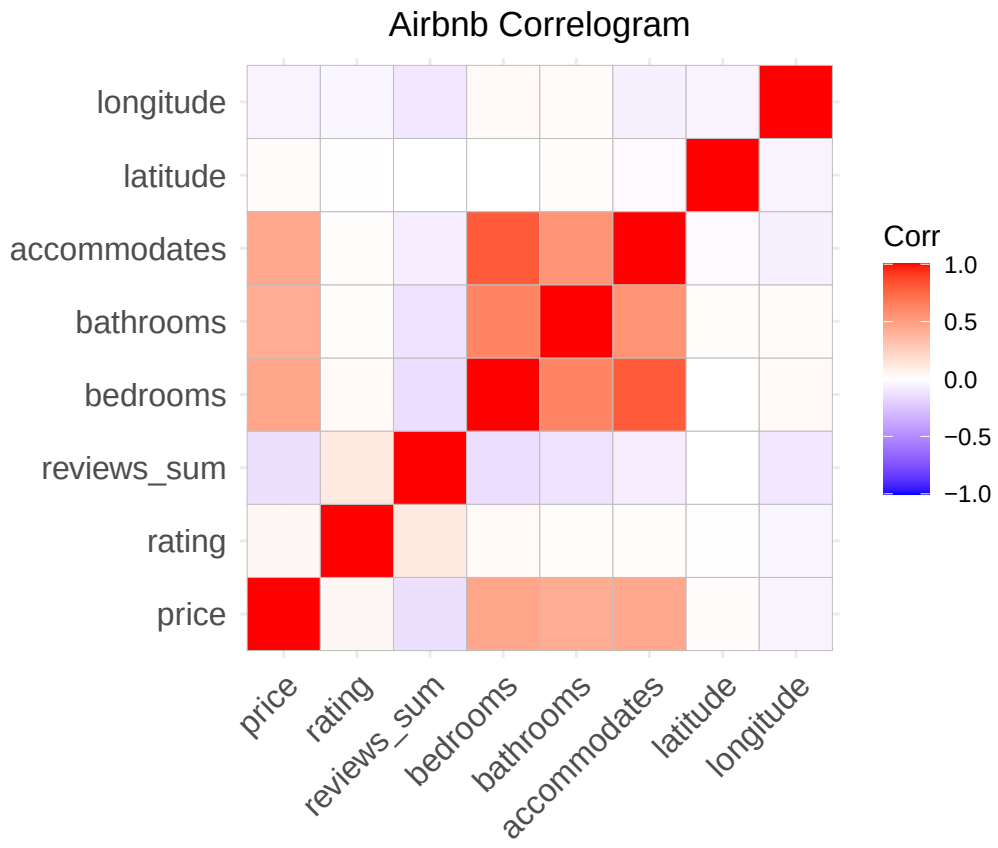
Remove non-numeric features:

```
airbnb_num <- airbnb %>% select(-c(city, room_type))
```

Create the correlation matrix:

```
airbnb_cor <- cor(airbnb_num)
```

Plot the Correlogram:



We discover the price is moderately correlated with the number of people a listing can accommodate as well as the number of bedrooms and bathrooms. Surprisingly, there is little correlation between the location (latitude & longitude) and price. This relationship will be explored with a scatterplot.

Density Plot of Price Distribution below $300:

Next we explore a plot of the price distribution below $300 to determine the most common price ranges.

The plot reveals a significant portion of the prices are below $100/night.

Geographical Scatterplot of Prices in Seattle:

Let's use the ggmap package to load a map of Seattle and visualize which areas are more expensive than others.

Create the map using stamenmap:

Note that the quantiles and price range will influence the pricing scale color:

```
summary(airbnb$price)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      15      65      88     113     125    5900
```

According to the above summary, we know that Q3 of the IQR was $125, so let's determine what percentage of prices are less than or equal to $300.

```
quantile(airbnb$price)
```

```
##   0%  25%  50%  75% 100%
##   15   65   88  125 5900
```

```
sum(airbnb$price <= 300)/length(airbnb$price)
```

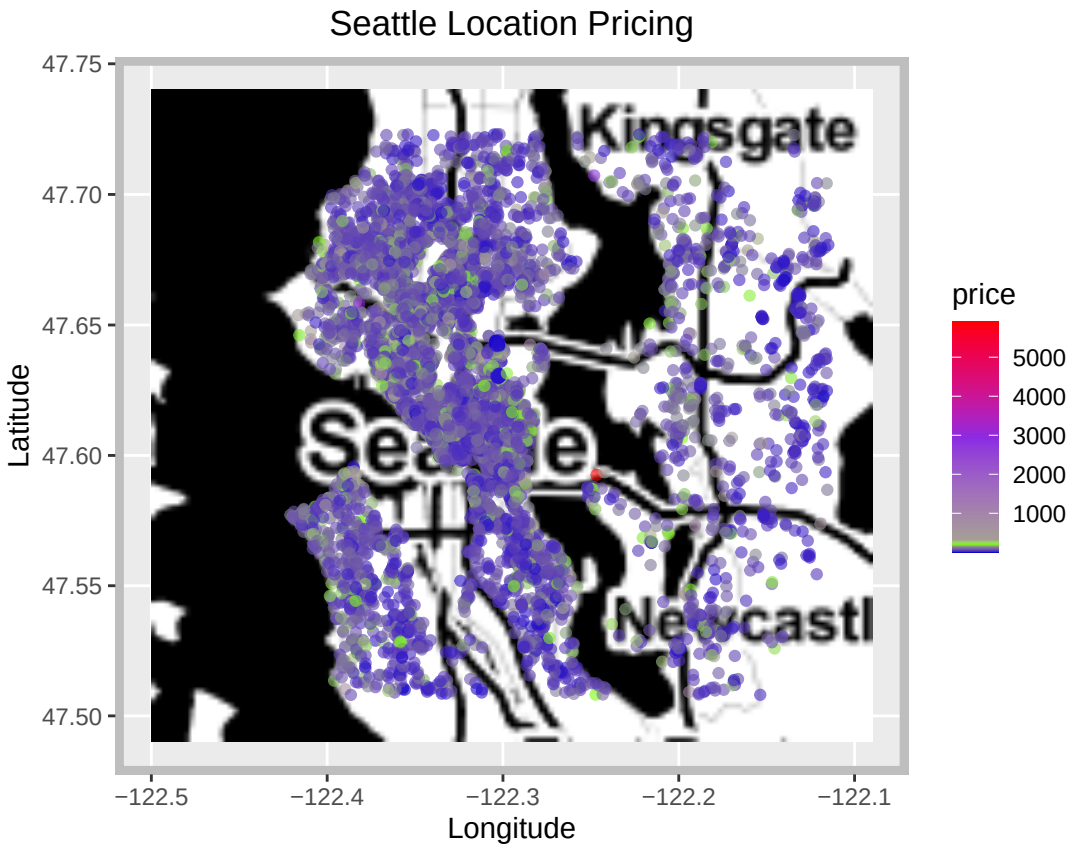
```
## [1] 0.9652
```

~ 96.5% of listings are <= $300, therefore we filter our price to remove outliers.

Filter the dataframe airbnb_map:

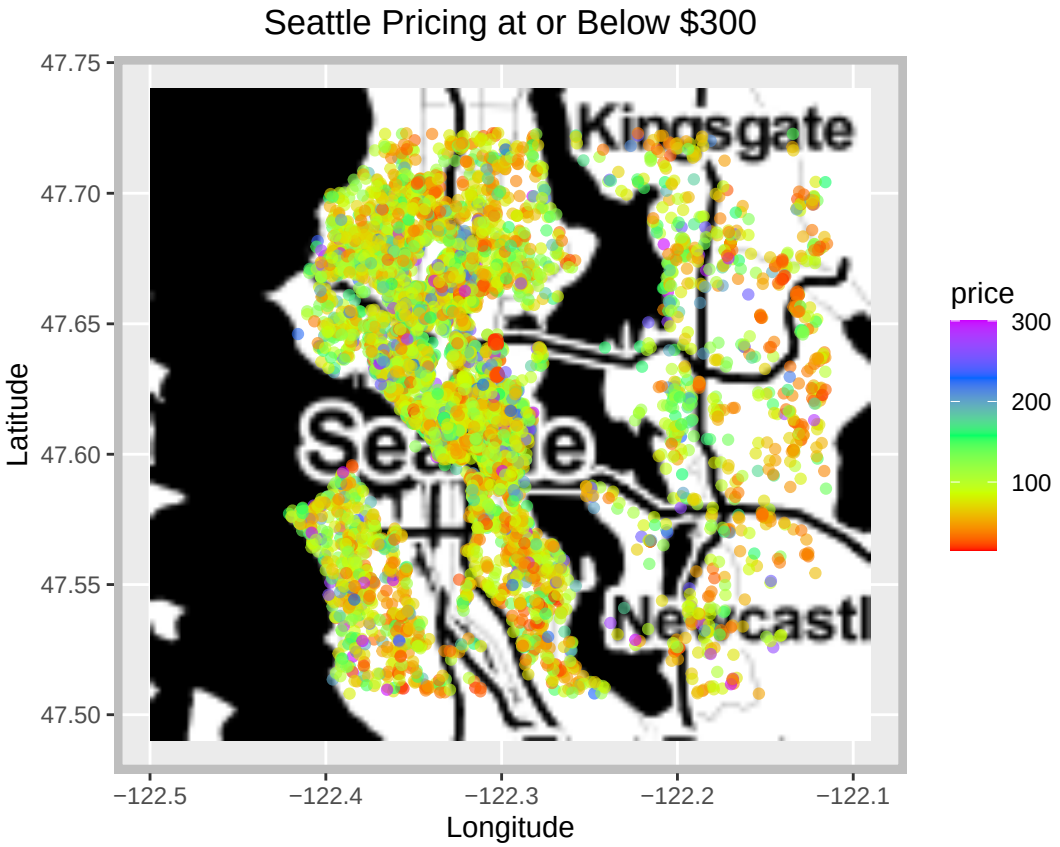
```
airbnb_map <- airbnb %>% filter(price <= 300)
```

Visualize a "Heatmap" of Seattle with all listing prices included:



It does not appear that one area is significantly more expensive than another and that most prices are less than $1000, though outliers may be skewing our data. As such, let's attempt to refine our heatmap by filtering the data.

Plot the Filtered Heat Map of Seattle: (less than or equal to $300)



The visualization suggests the vast majority of listings in Seattle are between USD 50 - 150 / night and are relatively concentrated in Seattle proper (as opposed to Newcastle, Mercer Island, or Bellevue).

Confirm the percentage of listings between USD 50 - 150 per night:

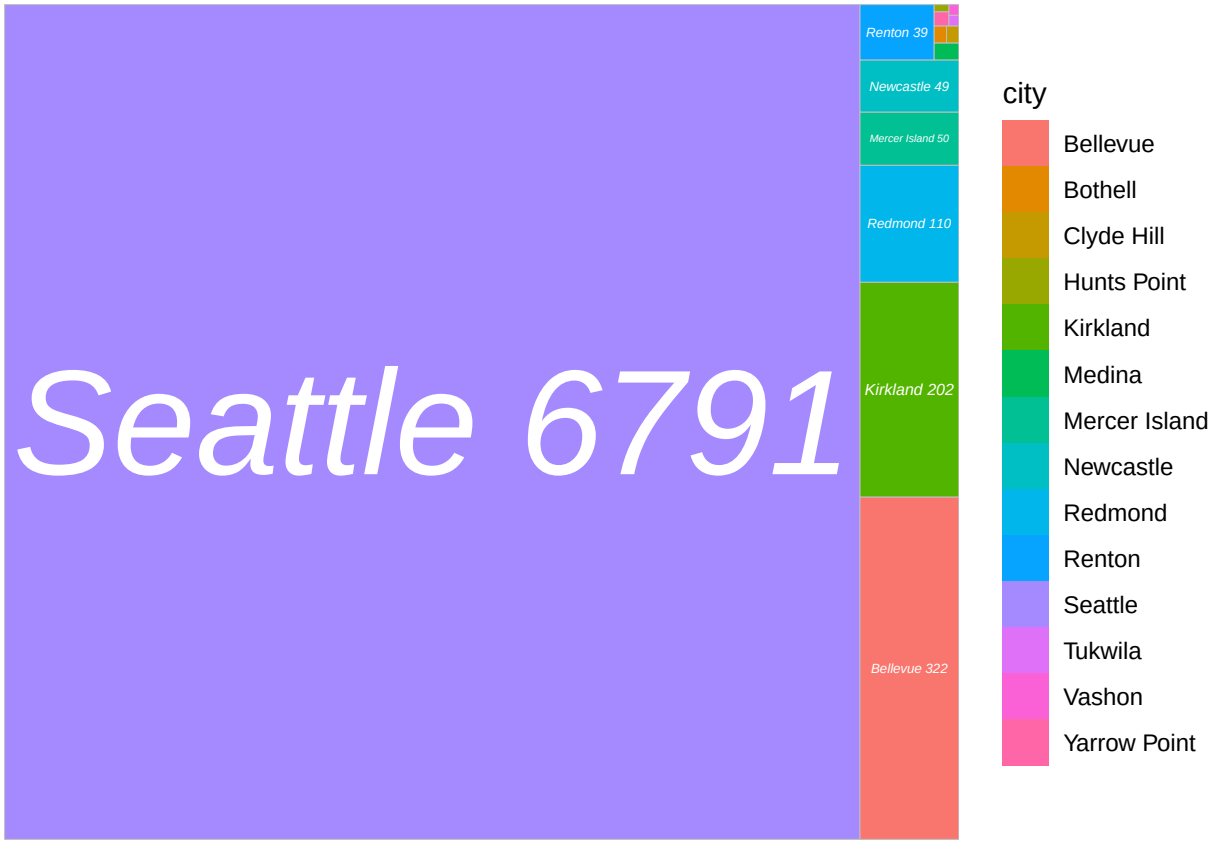
```
sum(airbnb$price >= 50 & airbnb$price <= 150)/length(airbnb$price)
```

```
## [1] 0.6998
```

Nearly 70% of listings range from USD 50 - 150/night.

Treemap:

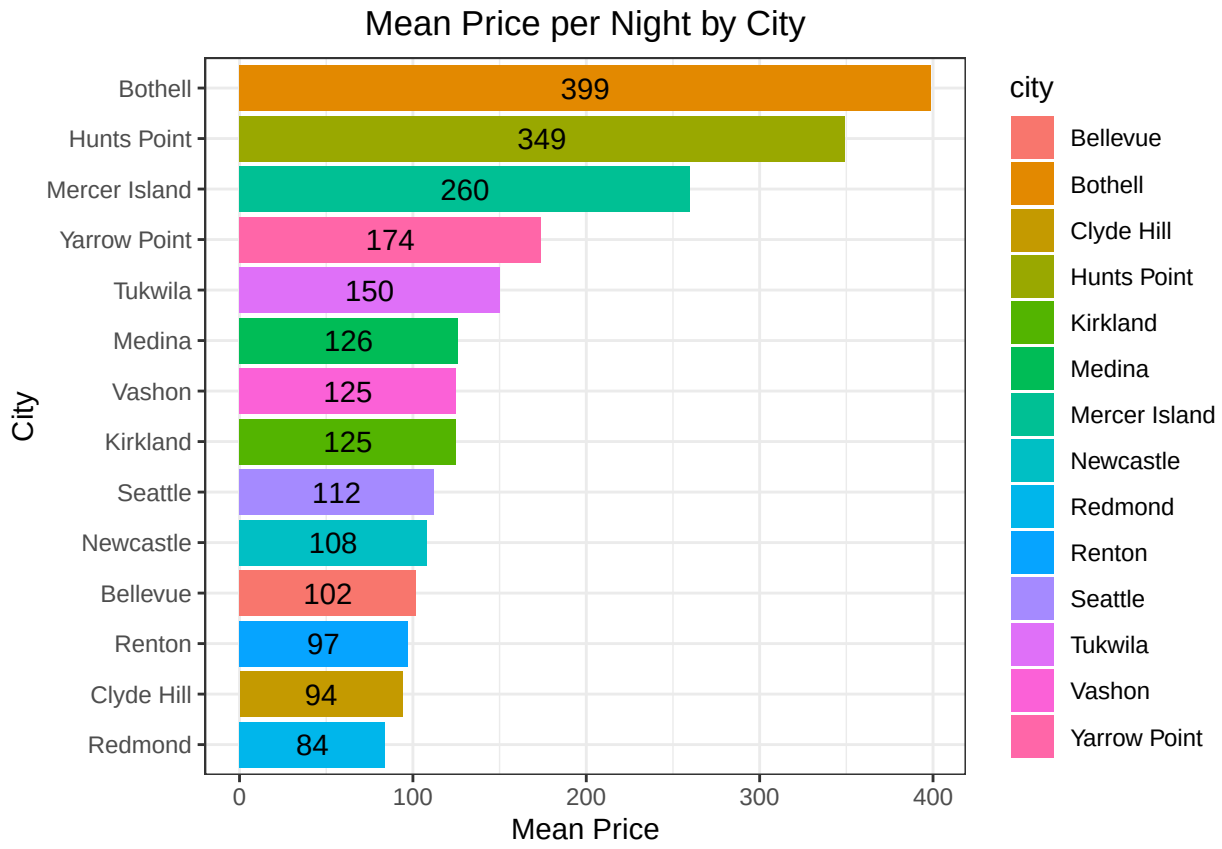
Arrange the cities by listing_sum in a dataframe for the Treemap. Plot the Treemap to visualize the distribution of listings by city:



The Treemap confirms the overwhelming number of listings in Seattle compared to the other cities.

Visualize Price by City:

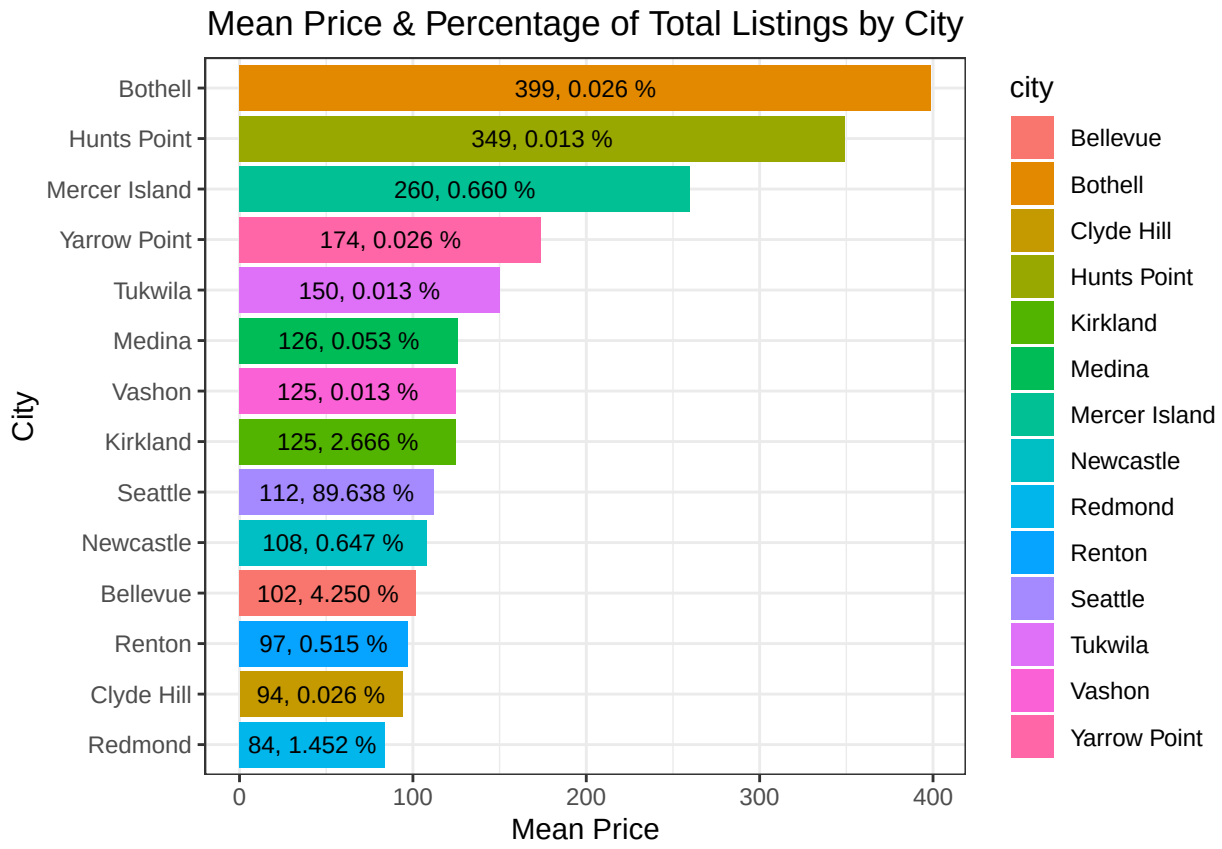
Plot the Visualization:



According to this visualization, it would seem location (latitude & longitude) is highly correlated with price. Let's explore why this was not the case in our correlogram. Let's replot the visualization with the sum of listings included.

Visualize Mean Price & Sum of Listings by City:

Plot the visualization with Mean Price & Percentage of Total Listings:

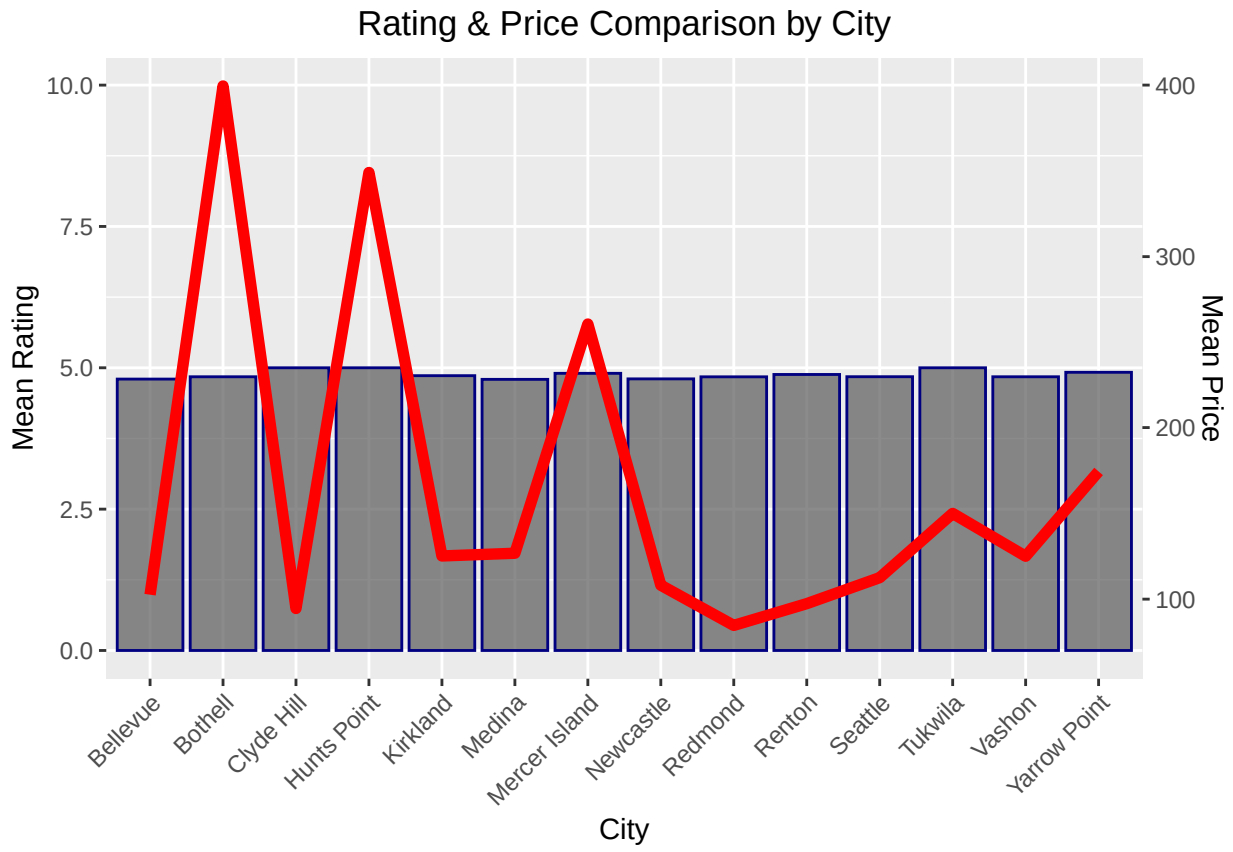


We learn that over 89% of listings are in Seattle with a mean price of $112/night. The top 7 listings with the highest average prices barely equal 1% (~0.8%) of the total listings though their average price is $226.10/night. The lower percentage of total listings in these locations at the higher price range, with almost 90% of listings in Seattle likely explains the lower correlation between price and location in this dataset.

Feature: "rating"

The "rating" feature is based on a numerical rating from 0 to 5 with a mean of 4.841. As this is a relatively high mean rating, we will explore its relationship mean price by city to determine its price prediction potential.

Create the dataframe rating_comp to compare mean price and rating. Set the parameters for the dual-axis plot. Plot the Barplot (Rating) with Overlapping Line (Price in RED):



Check the range of mean ratings by city:

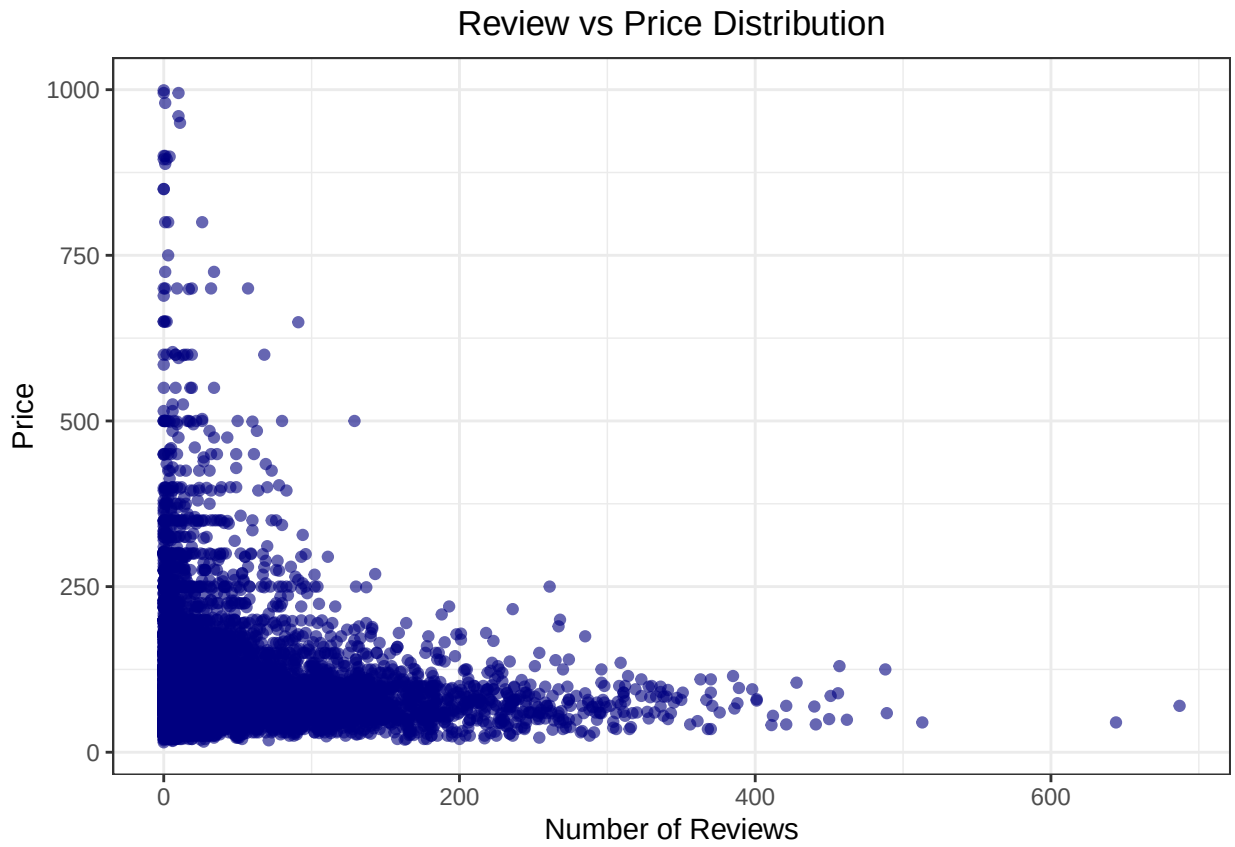
```
range(rating_comp$mean_rating)
```

```
## [1] 4.796 5.000
```

We notice that due to the tight range of mean ratings by city, there is a very low correlation between price and rating. For example, a high price does not guarantee a higher rating as the most expensive city Bothell does not have the highest rating. Furthermore, cities with lower average prices tend to have mean ratings higher than Bothell.

Feature: "reviews_sum"

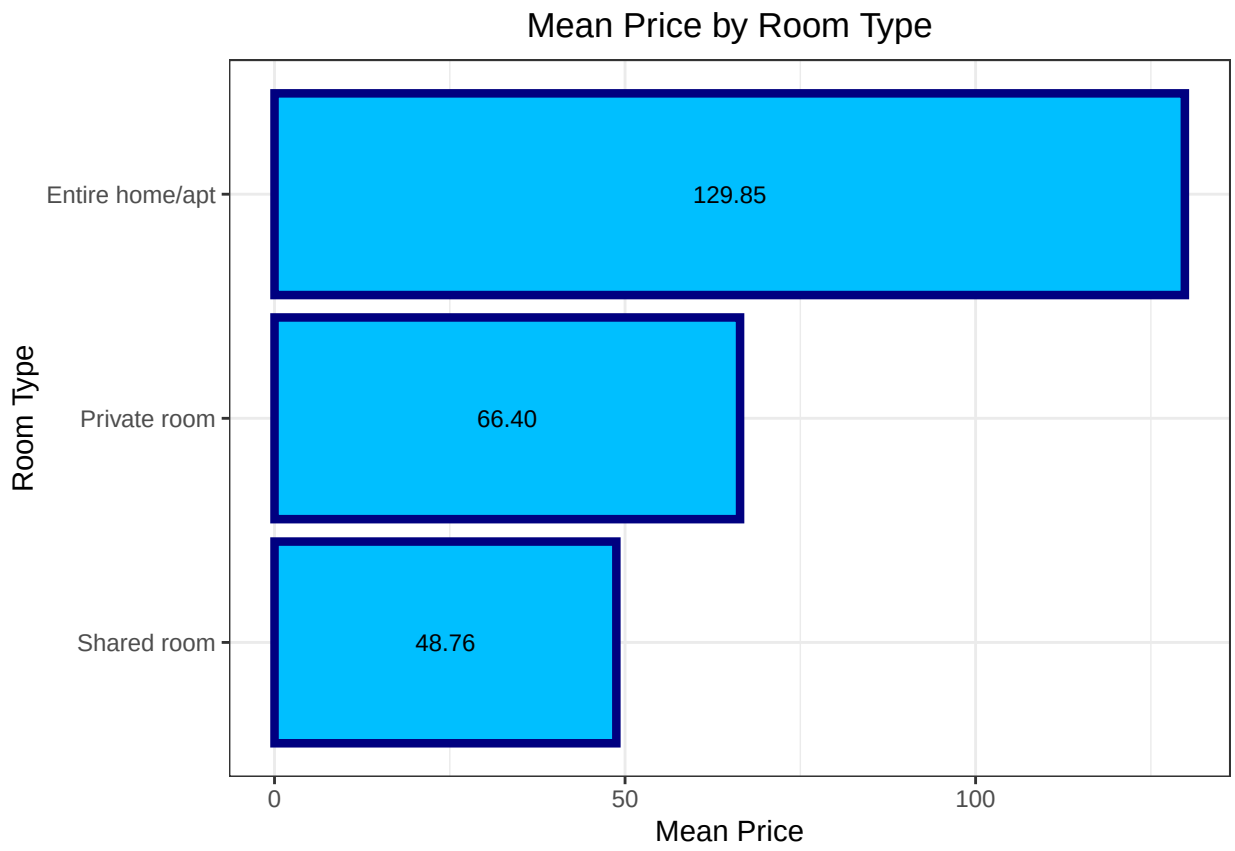
Now let's compare the total number of reviews and different price ranges:



The visualization shows us that there are generally fewer reviews for listings with higher prices (above ~$500). We can infer that this is because there are fewer stays at these higher-priced listings.

Feature: "room_type"

There are 3 different room types: Entire home / apartment, Private Room, & Shared Room. Let's explore the relationship between room type and average price:

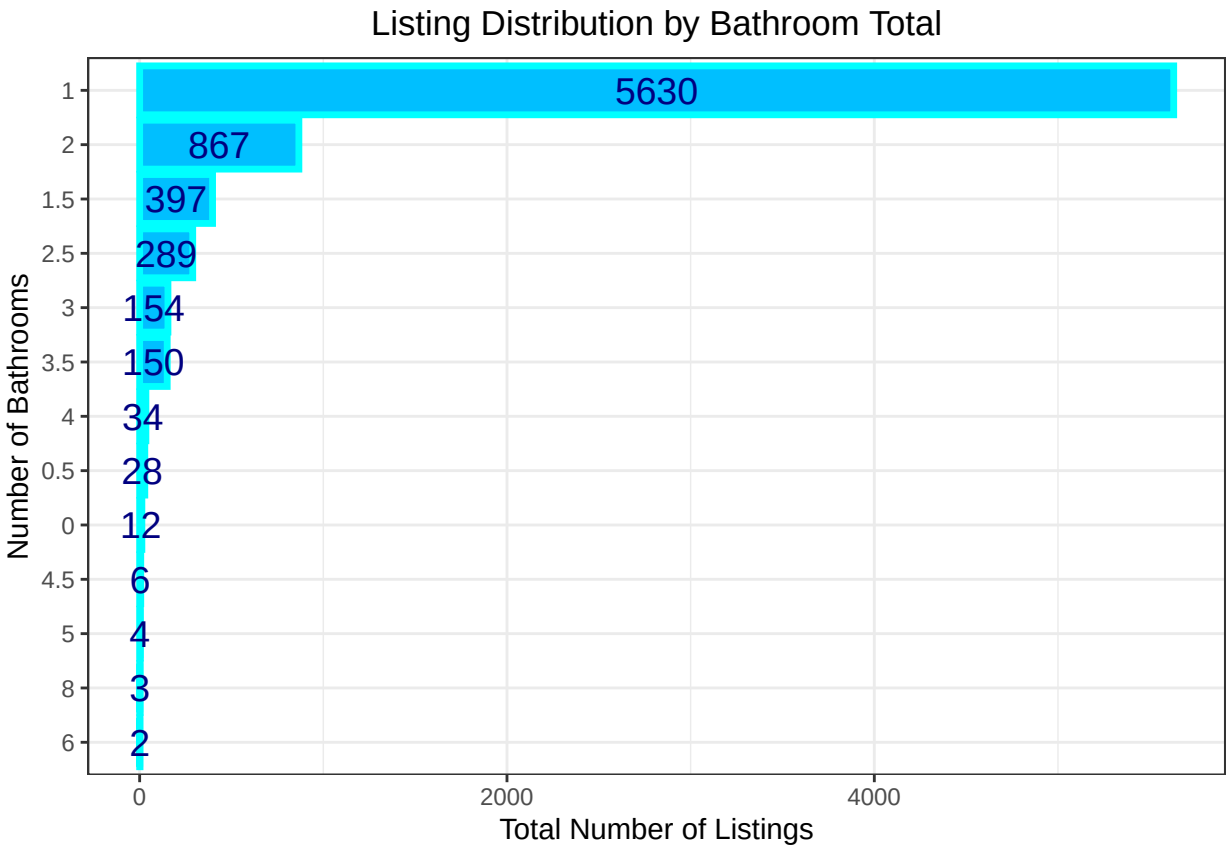


Feature: "bathrooms"

Bathrooms range in quantity from 0 to 8 in increments of 0.5 per listing. Let's explore a visualization that delineates the mean price per listing based on the number of bathrooms available:

We notice that generally, the higher the number of bathrooms per listing, the higher the price. Interestingly, listings with 0 bathrooms actually had a higher average price than listings with 0.5 - 1.5, and 8 bathrooms! It is clear that a listings with 8 bathrooms having a lower mean price than those with one bathroom is suspect and most certainly an outlier.

Now let's visualize a barplot with the total number of listings per bathroom count:

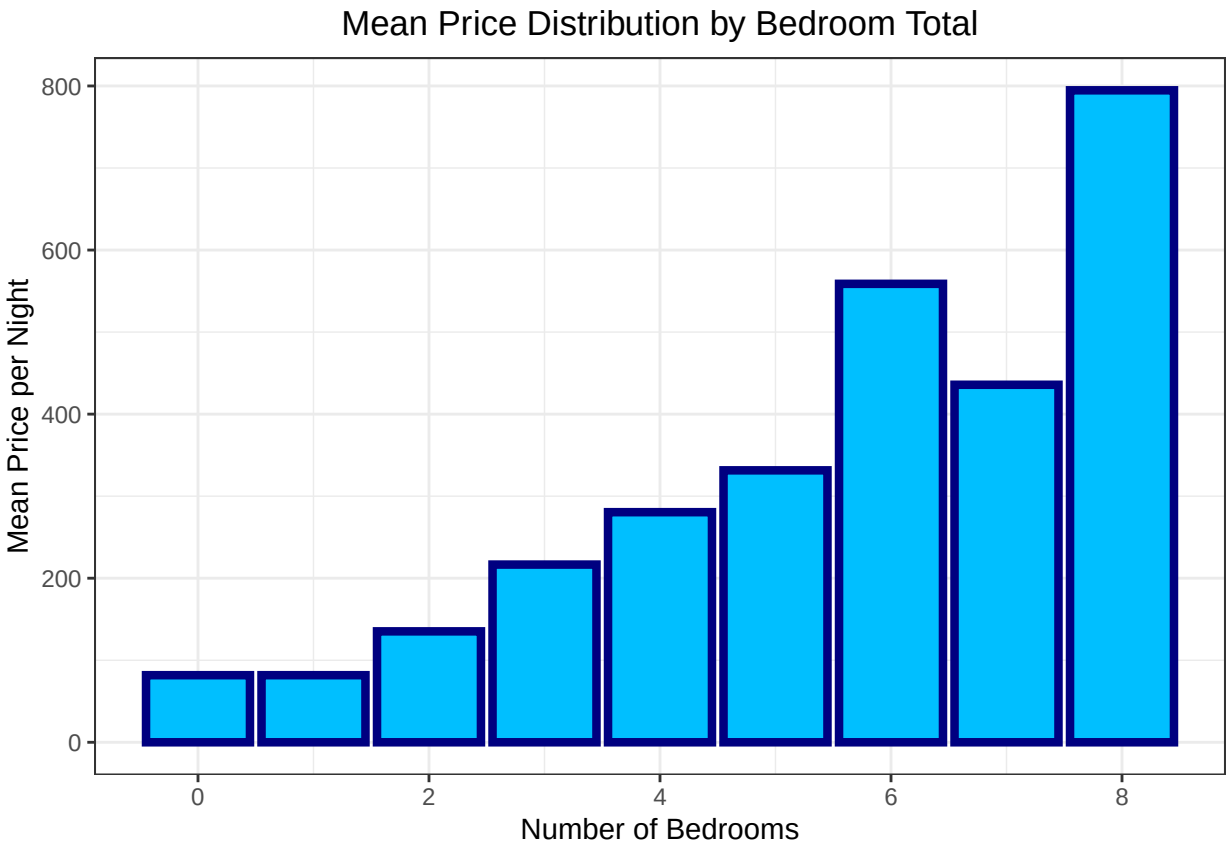


It is evident that the vast majority of listings (~74%) have only 1 bathroom.

Feature: "bedrooms"

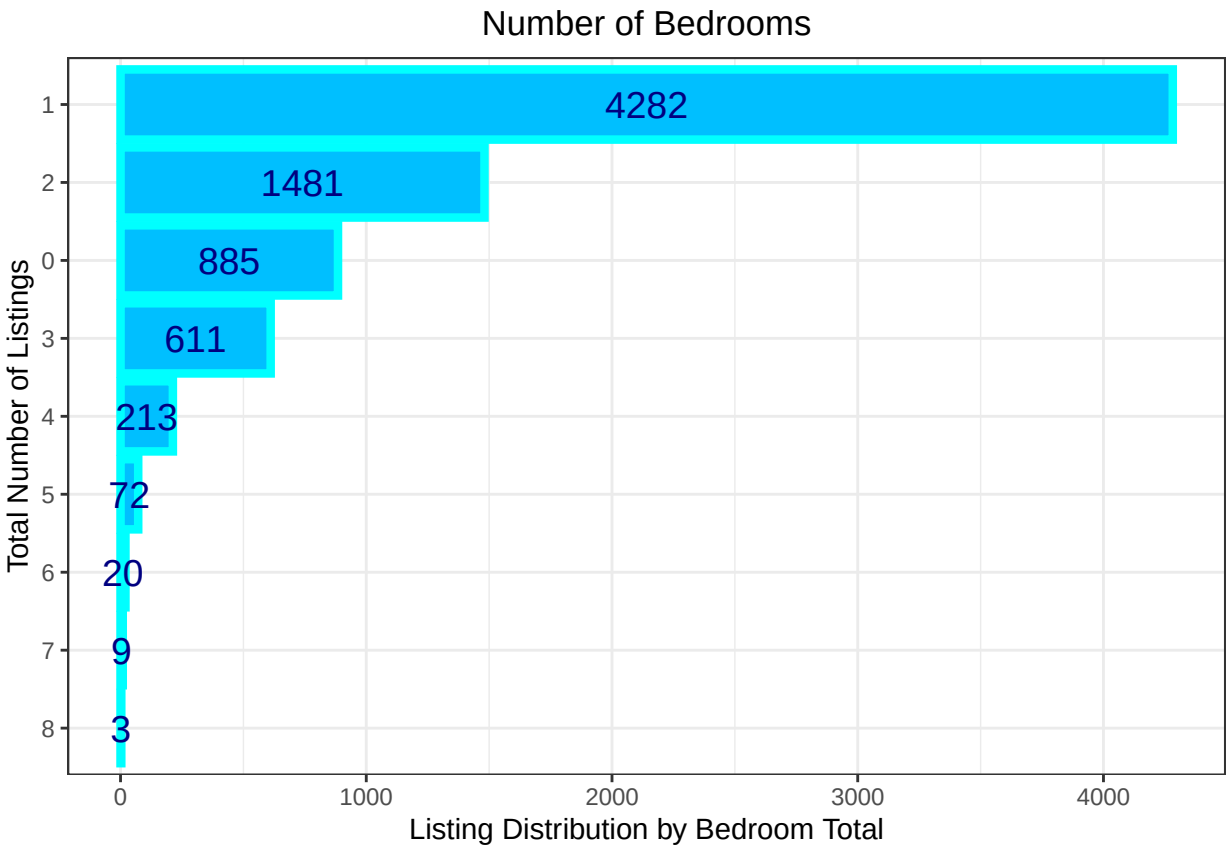
The bedrooms feature details the number of bedrooms available per listing and has been shown in our correlogram to be positively correlated with the price per listing.

Let's explore a visualization that details the mean price per listing versus the number of beds available:



The visualization confirms the moderately positive correlation between price and number of bedrooms. This feature is very likely to contribute to the accuracy of our price prediction models.

Now let's explore listing quantity by number of bedrooms:

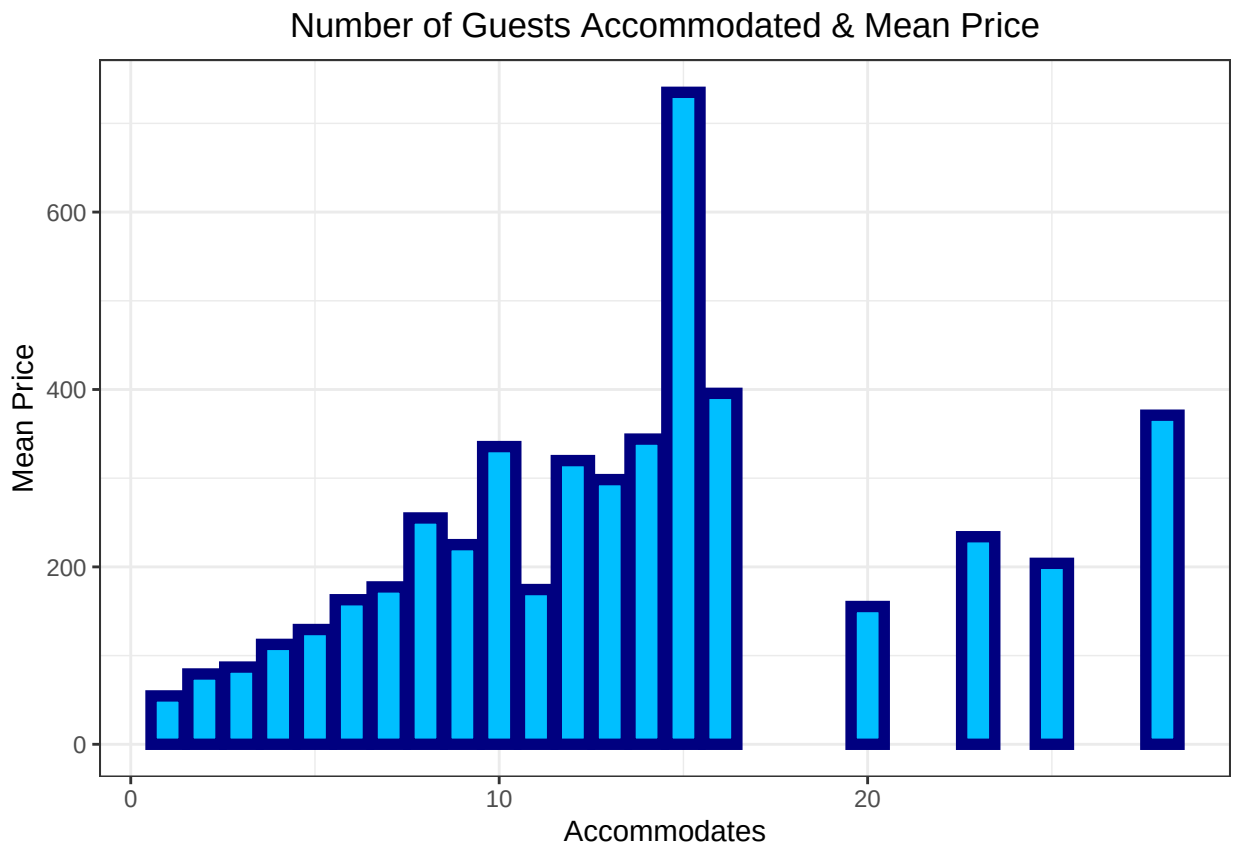


1 bedroom listings lead the distribution with 4,282 listings (56.5%). As a result of the above findings, there is a high probability of a guest renting a 1 bedroom, 1 bath unit.

Feature: "accommodates"

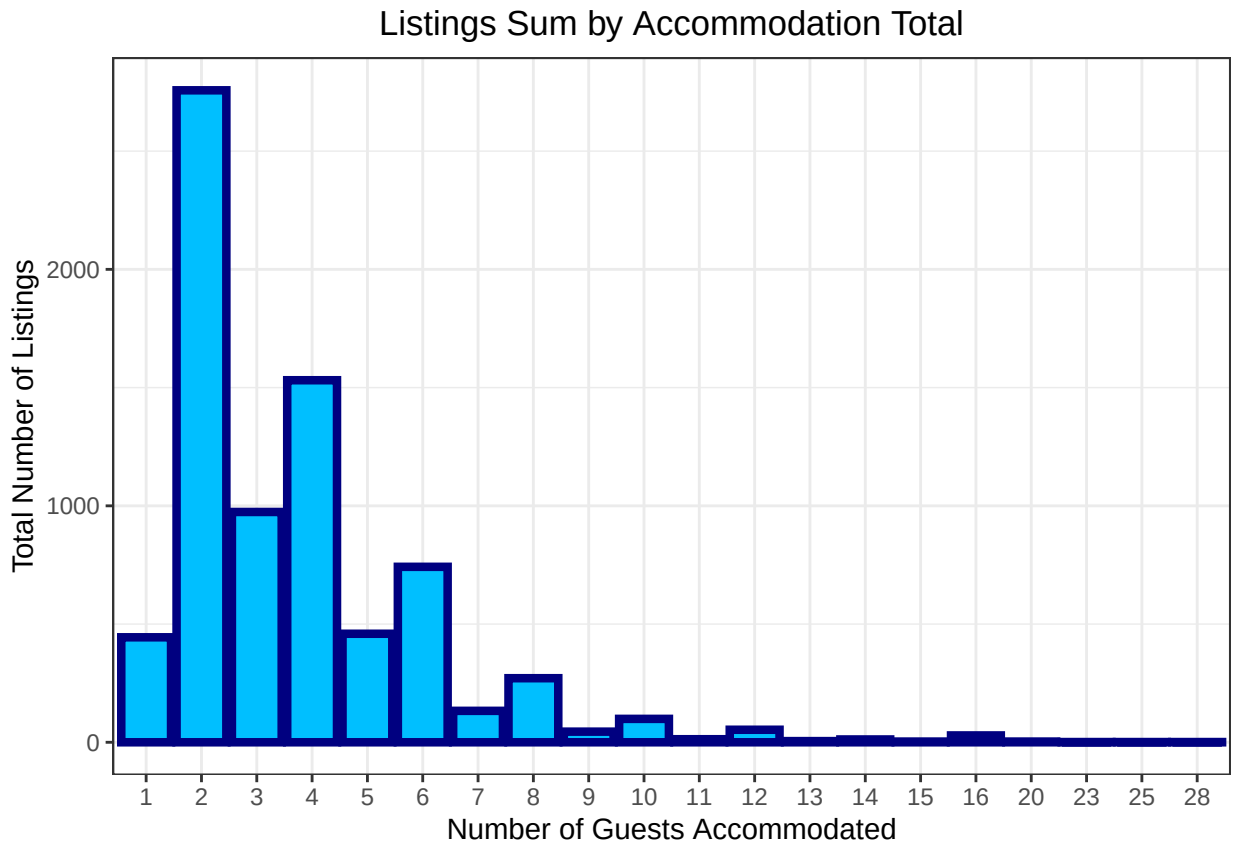
Generally, we might predict that the mean price of a listing per night would increase if it could provide space for more guests.

Let's explore this idea with a visualization:



The visualization confirms there is a general increase in average price as the number of guests able to be accommodated increases.

Finally, let's check the number of people most units can accommodate:



The majority of listings can accommodate 4 or fewer guests with 2 guests being the most popular.

EDA Conclusion:

The correlogram starts us off by highlighting important relationships between price, bedrooms, bathrooms, and accommodates which are later confirmed by visualizations and quantitative analysis. Additionally, we learn location (using latitude and longitude), rating, and reviews sum all have certain potential to increase the accuracy of our predictive models if included in our formula. "Room_type", an important feature, could be vectorized into a quantitative scale, though for the purposes of the following regression models, will be revisited in our conclusion.

Modeling:

Partition Airbnb Combined & Test Sets for the Final Model:

`airbnb_combined` will be used to fit the Final Model as it is equal to the sum of the training and validation sets and equal to 90% of the total data. `airbnb_test`, comprised of the remaining 10% of the data, will be used as the test set for the final model.

```
# Set the seed for reproducibility:
set.seed(123, sample.kind = "Rounding")
test_index <- createDataPartition(y = airbnb$price, times = 1, p = 0.1, list = F)
airbnb_combined <- airbnb[-test_index, ]
airbnb_test <- airbnb[test_index, ]
```

```r
# Remove test_index:
rm(test_index)
```

Split Training and Validation Sets from airbnb_combined to train our models: `airbnb_train` will constitute 80% and validation the remaining 20% of `airbnb_combined`.

```r
set.seed(123, sample.kind = "Rounding")
test_index <- createDataPartition(y = airbnb_combined$price, times = 1, p = 0.2,
    list = F)
airbnb_train <- airbnb_combined[-test_index, ]
validation <- airbnb_combined[test_index, ]

# Remove test_index once again:
rm(test_index)
```

The Loss Function / RMSE is defined as follows:

Results will be based on calculating the Root Mean Square Error on the test set. The RMSE weighs large errors more heavily and tends to be the preferred measurement of error in regression models when large errors are undesirable.

Mathematically, it is defined as:

$$\sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

```r
RMSE <- function(true_ratings, predicted_ratings) {
    sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Baseline Model: Median Model

This model will use the median price of the training set to calculate a "baseline" RMSE as a benchmark. The formula can be defined as follows with $y_{u,i}$ = predicted rating, M = the median of the observed data, $\epsilon_{u,i}$ = independent errors centered at 0.

$$y_{u,i} = M + \epsilon_{u,i}$$

```r
airbnb_train_median <- median(airbnb_train$price)

# Table the Results
MM_RMSE <- RMSE(validation$price, airbnb_train_median)
results_table <- tibble(Model_Type = "Baseline Median", RMSE = MM_RMSE) %>% mutate(RMSE = sprintf("%0.2f
    RMSE))
knitr::kable(results_table)
```

Model_Type	RMSE
Baseline Median	99.05

The Baseline Model achieves an RMSE of 99.05.

Vectorize the optimal formula that will be used for most Models:

The formula has been determined by the above EDA as well as experimentation on the training models. Additionally, the formula will reduce lines of code.

```
airbnb_form <- price ~ rating + reviews_sum + bedrooms + bathrooms + accommodates +
    latitude + longitude
```

Linear Model:

With linear regression, we attempt to predict a (dependent) y variable, in this case "price", with the(independent) input x variables: rating + reviews_sum + bedrooms + bathrooms + accommodates + latitude + longitude in order to build a statistically significant model with a p-value less than .05.

```
lm_airbnb <- lm(airbnb_form, data = airbnb_train)

# Create the prediction
lm_preds <- predict(lm_airbnb, validation)

# Table the Results
LM_RMSE <- RMSE(validation$price, lm_preds)
results_table <- tibble(Model_Type = c("Baseline Median", "Linear"), RMSE = c(MM_RMSE,
    LM_RMSE)) %>% mutate(RMSE = sprintf("%0.2f", RMSE))

knitr::kable(results_table)
```

Model_Type	RMSE
Baseline Median	99.05
Linear	73.80

The Linear Model vastly improves upon the Baseline Model with an RMSE of 73.8.

Elastic Net Regression with glmnet:

This model introduces regularization on a generalized linear model using penalized maximum likelihood and tuning optimal alpha and lambda parameters unlike Lasso and Ridge Regression Models. It combines advantages of both models by penalizing by the L1 & L2-norms. Mathematically, the estimates can be defined as follows:

$$\hat{\beta} \equiv \underset{\beta}{\operatorname{argmin}}(\|y - X\beta\|^2 + \lambda_2\|\beta\|^2 + \lambda_1\|\beta\|_1)$$

```
# Set the seed for reproducibility:
set.seed(123, sample.kind = "Rounding")
train_enr <- train(airbnb_form, data = airbnb_train, method = "glmnet", preProcess = c("center",
    "scale"), tuneLength = 10, trace = F)
```

```
# Confirm the optimal alpha and lambda parameters
train_enr$bestTune
```

```
##   alpha lambda
## 7   0.1  21.24
```

```
# alpha = 0.1, and lambda = 21.24

# Create the Prediction:
elastic_preds <- predict(train_enr, validation)

# Table the Results
ENR_RMSE <- RMSE(validation$price, elastic_preds)
results_table <- tibble(Model_Type = c("Baseline Median", "Linear", "Elastic Net Regression"),
    RMSE = c(MM_RMSE, LM_RMSE, ENR_RMSE)) %>% mutate(RMSE = sprintf("%0.2f", RMSE))
knitr::kable(results_table)
```

Model_Type	RMSE
Baseline Median	99.05
Linear	73.80
Elastic Net Regression	74.06

The Elastic Net with Regression underperforms the Linear Model with an RMSE of 74.06.

Regression Tree Model with rpart:

Regression trees or regression decision trees partition data recursively into smaller, more simplistic partitions where specific models can be applied using conditional statements on the predictors. It is generally not as robust as models with bootstrap aggregation or Random Forest Models.

```
# Set the seed for reproducibility:
set.seed(123, sample.kind = "Rounding")
train_rpart <- train(airbnb_form, method = "rpart", data = airbnb_train, tuneGrid = data.frame(cp = seq
    0.05, len = 25)), preProcess = c("center", "scale"))

# Check the bestTune to find the final complexity parameter used for the model:
train_rpart$bestTune
```

```
##        cp
## 4 0.00625
```

```
# The final cp is 0.00625.

# Create the Prediction:
rt_preds <- predict(train_rpart, validation)

# Table the Results:
RT_RMSE <- RMSE(validation$price, rt_preds)
```

```
results_table <- tibble(Model_Type = c("Baseline Median", "Linear", "Elastic Net Regression",
    "Regression Tree"), RMSE = c(MM_RMSE, LM_RMSE, ENR_RMSE, RT_RMSE)) %>% mutate(RMSE = sprintf("%0.2f
    RMSE))
knitr::kable(results_table)
```

Model_Type	RMSE
Baseline Median	99.05
Linear	73.80
Elastic Net Regression	74.06
Regression Tree	76.03

The Regression Tree model underperforms the both the Linear Model and the Elastic Net Regression Model with an RMSE of 76.03.

Random Forest Model:

Random Forests are similar to the Regression Tree model, though they average several random subsets of decision trees in order improve predictive performance.

```
# Set the tuneGrid parameters:
rf_tune <- expand.grid(.mtry = c(1:3))

# Set the seed for reproducibility:
set.seed(123, sample.kind = "Rounding")
train_rf <- train(airbnb_form, data = airbnb_train, method = "rf", ntree = 150, tuneGrid = rf_tune,
    nSamp = 1000, preProcess = c("center", "scale"))

# Check the bestTune:
train_rf$bestTune
```

```
##   mtry
## 1    1
```

```
# The bestTune is a mtry of 1

# Create the Prediction:
rf_preds <- predict(train_rf, validation)

# Table the Results
RF_RMSE <- RMSE(validation$price, rf_preds)
results_table <- tibble(Model_Type = c("Baseline Median", "Linear", "Elastic Net Regression",
    "Regression Tree", "Random Forest"), RMSE = c(MM_RMSE, LM_RMSE, ENR_RMSE, RT_RMSE,
    RF_RMSE)) %>% mutate(RMSE = sprintf("%0.2f", RMSE))
knitr::kable(results_table)
```

Model_Type	RMSE
Baseline Median	99.05
Linear	73.80
Elastic Net Regression	74.06

Model_Type	RMSE
Regression Tree	76.03
Random Forest	66.57

The Random Forest significantly improves upon the all previous Models with an RMSE of 66.57.

"Bagging Tree" – Bootstrap Aggregating Model:

Bagging, similar to Random Forest Models, uses aggregation to create more robust predictions. It is an ensemble method that improves stability over classic regression tree models while attempting to minimize overfitting.

```
# Set the seed for reproducibility:
set.seed(123, sample.kind = "Rounding")
train_bag <- train(airbnb_form, data = airbnb_train, method = "treebag", importance = T,
    tuneLength = 10, preProcess = c("center", "scale"))

# Create the Prediction
bag_preds <- predict(train_bag, validation)

# Table the Results
BAG_RMSE <- RMSE(validation$price, bag_preds)
results_table <- tibble(Model_Type = c("Baseline Median", "Linear", "Elastic Net Regression",
    "Regression Tree", "Random Forest", "BAG"), RMSE = c(MM_RMSE, LM_RMSE, ENR_RMSE,
    RT_RMSE, RF_RMSE, BAG_RMSE)) %>% mutate(RMSE = sprintf("%0.2f", RMSE))
knitr::kable(results_table)
```

Model_Type	RMSE
Baseline Median	99.05
Linear	73.80
Elastic Net Regression	74.06
Regression Tree	76.03
Random Forest	66.57
BAG	68.98

The BAG Model underperforms the Random Forest Model with an RMSE of 68.98.

kNN Regression Model:

k Nearest Neighbors uses the parameter k to tune a weighted average of "neighbors" that is calculated by an algorithm that measures Euclidean or Mahalanobis distance. It is often used in pattern recognition and a popular modeling technique for both classification and regression prediction.

```
# Set the seed for reproducibility:
set.seed(123, sample.kind = "Rounding")
train_knn <- train(airbnb_form, method = "knn", data = airbnb_train, tuneLength = 5,
    preProcess = c("center", "scale"))
```

```
# Find best value for k:
train_knn$bestTune
```

```
##    k
## 5 13
```

```
# k = 13 is the final value used for the model.

# Create the Prediction
knn_preds <- predict(train_knn, validation)

# Table the Results
kNN_RMSE <- RMSE(validation$price, knn_preds)
results_table <- tibble(Model_Type = c("Baseline Median", "Linear", "Elastic Net Regression",
    "Regression Tree", "Random Forest", "BAG", "kNN"), RMSE = c(MM_RMSE, LM_RMSE,
    ENR_RMSE, RT_RMSE, RF_RMSE, BAG_RMSE, kNN_RMSE)) %>% mutate(RMSE = sprintf("%0.2f",
    RMSE))
knitr::kable(results_table)
```

Model_Type	RMSE
Baseline Median	99.05
Linear	73.80
Elastic Net Regression	74.06
Regression Tree	76.03
Random Forest	66.57
BAG	68.98
kNN	69.94

The kNN Model underperforms both the RF and BAG CV Models with an RMSE of 69.94.

Neural Net Model:

ANNs or artificial neural networks are algorithms modeled on the human brain, that generally utilize one to three "nodes" to cluster raw inputs and predict real-world data. It is known as a "universal approximator" due to its ability to accurately determine correlations and event probabilities. Mathematically, ANNs assign weights to inputs that reduce noise and amplify predictive power, which are then passed through Net input and activation functions before being released as the output.

```
# Create the tuneGrid parameters:
NN_grid <- expand.grid(size = c(1, 5, 20), decay = c(0, 0.01, 0.1))

# Set the seed for reproducibility:
set.seed(123, sample.kind = "Rounding")
train_NN <- train(airbnb_form, data = airbnb_train, method = "nnet", linout = T,
    trace = F, tuneGrid = NN_grid, preProc = c("center", "scale"))

# Check bestTune:
train_NN$bestTune
```

```
##   size decay
```

```
## 5    5  0.01
```

```
# The optimal size is 5 and decay = 0.01

# Create the Prediction:
NN_preds <- predict(train_NN, validation)

# Table the Results:
NN_RMSE <- RMSE(validation$price, NN_preds)
results_table <- tibble(Model_Type = c("Baseline Median", "Linear", "Elastic Net Regression",
    "Regression Tree", "Random Forest", "BAG", "kNN", "Neural Net"), RMSE = c(MM_RMSE,
    LM_RMSE, ENR_RMSE, RT_RMSE, RF_RMSE, BAG_RMSE, kNN_RMSE, NN_RMSE)) %>% mutate(RMSE = sprintf("%0.2f
    RMSE))
knitr::kable(results_table)
```

Model_Type	RMSE
Baseline Median	99.05
Linear	73.80
Elastic Net Regression	74.06
Regression Tree	76.03
Random Forest	66.57
BAG	68.98
kNN	69.94
Neural Net	71.41

The Neural Net Model underperforms all models except for the Baseline Linear & Regression Tree Models with an RMSE of 71.41.

Final Model:

Random Forest Model (with test set):

```
# Set the mtry to 1 as determined from the previous tuning on airbnb_train:
tune_grid_rf <- expand.grid(mtry = 1)

# Set the seed for reproducibility:
set.seed(123, sample.kind = "Rounding")
train_rf_final <- train(airbnb_form, data = airbnb_combined, method = "rf", ntree = 150,
    tuneGrid = tune_grid_rf, nSamp = 1000, preProcess = c("center", "scale"))

# Create the Prediction:
rf_preds_final <- predict(train_rf_final, airbnb_test)

# Table the Results
RFF_RMSE <- RMSE(airbnb_test$price, rf_preds_final)
results_table <- tibble(Model_Type = c("Baseline Median", "Linear", "Elastic Net Regression",
    "Regression Tree", "Random Forest", "BAG", "kNN", "Neural Net", "Random Forest Final (Test Set)"),
    RMSE = c(MM_RMSE, LM_RMSE, ENR_RMSE, RT_RMSE, RF_RMSE, BAG_RMSE, kNN_RMSE, NN_RMSE,
        RFF_RMSE)) %>% mutate(RMSE = sprintf("%0.2f", RMSE))
```

Results:

Model_Type	RMSE
Baseline Median	99.05
Linear	73.80
Elastic Net Regression	74.06
Regression Tree	76.03
Random Forest	66.57
BAG	68.98
kNN	69.94
Neural Net	71.41
Random Forest Final (Test Set)	71.43

As expected, the Random Forest Final Model vastly improved upon the Baseline Model with a final RMSE of 71.43 on the test set – a 27.88% increase in predictive performance.

The Final Model underperformed the RMSE it achieved on the validation set (66.27) as anticipated, though it still outperformed most models except

for the BAG, kNN, and Neural Net Models and their RMSEs based on the validation set. (All Models were tested on the test set though only the best performing Model (Random Forest) was selected for the final report).

In general, ensemble machine learning algorithms that aim to decrease variance were most successful (Bagging Tree and RF). Parallel ensemble methods

like Random Forests have proven to be robust price prediction algorithms and therefore it comes as no surprise it was the top performer.

Conclusion:

Accurate price prediction and regression analysis continues to be a popular application of machine learning, though it is known to be riddled with challenges and seemingly endless input variables to contend with. This report presented visual and quantitative explorations that informed and augmented our predictive pricing models with some interesting results.

The visualizations, for example, those concerned with the number of guests, bedrooms, and bathrooms a listing accommodates, could inform Airbnb hosts of the potential changes they need to make in order to increase their cash flow. Additionally, the predictive pricing models can further delineate to what extent guests are willing to pay for specific listing configurations (e.g. location, number of bedrooms) with quantitative specificity.

The limitations of this report include the lack of feature engineering as certain variables could have been formed into new metrics to potentially increase predictive power. Furthermore, the presence of NAs without complete information available on the web to fill them with always leads to some semblance of inaccuracy. In terms of the modeling, the use of repeated cross-validation may have led to lower RMSEs as the dataset itself is not exceptionally large.

Future reports involving price prediction in the vacation-rental industry are likely to become more accurate with increasingly transparent data on sites like Inside Airbnb which feature detailed information about listings all over the globe. Additionally, recent advances in artificial neural networks as well as regression tree analysis promise to make price forecasting a more precise science.

Citations:

Bradley Boehmke, "Regression Trees"
https://uc-r.github.io/regression_trees

Elastic net regularization https://en.wikipedia.org/wiki/Elastic_net_regularization

Irizarry, Rafael A., "Introduction to Data Science: Data Analysis and Prediction Algorithms in R" https://rafalab.github.io/dsbook/

Pathmind, "A Beginner's Guide to Neural Networks and Deep Learning" https://pathmind.com/wiki/neural-network