nyc_airbnb

INTRODUCTION

Predicting pricing is one of the more common uses of machine learning. Whether it is predicting the cost of flights in the near future or estimating the current market value of a home, there are many applications for these types of prediction algorithms.

In this report, we will discuss the methods and processes for building an algorithm that will predict the nightly costs of airbnb rentals in New York City. This data set is available on Kaggle and contains airbnb listings in New York City from 2019.

The data set includes information about the name of the listing and host, neighborhood, neighborhood group (borough), latitude, longitude, room type, minimum length of stay, number of reviews, reviews per month, date of last review, lists per host, and availability of the listing over the past 365 days.

The goal of this project is to be able to accurately predict the nightly rate of an airbnb listing in New York City in 2019 given its features. In order to do so, we will explore the data set, wrangle any features to accommodate our use cases, build a variety of models optimized to minimize our loss function (Root Mean Squared Error or RMSE), and lastly discuss our results, limitations, and future improvements.

METHODS/ANALYSIS

The first step in our analysis is to import the data set and examine the data.

```r
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: data.table
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
## The following object is masked from 'package:purrr':
##
##     transpose
```

```r
if(!require(recosystem)) install.packages("recosystem", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: recosystem
```

```r
library(tidyverse)
library(caret)
library(data.table)
library(recosystem)

# This data set was imported from Kaggle. It is available at:
# https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data
airbnb_nyc <- read_csv("./airbnb_nyc.csv")
```

```
## Warning: One or more parsing issues, see `problems()` for details
```

```
## Rows: 38277 Columns: 18
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (5): name, host_name, neighbourhood_group, neighbourhood, room_type
## dbl  (11): id, host_id, latitude, longitude, price, minimum_nights, number_o...
## lgl   (1): license
## date  (1): last_review
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
airbnb_nyc %>% as_tibble()
```

```
## # A tibble: 38,277 x 18
##       id name         host_id host_name  neighbourhood_g~ neighbourhood latitude
##    <dbl> <chr>          <dbl> <chr>      <chr>            <chr>            <dbl>
##  1  2595 Skylit Midt~    2845 Jennifer   Manhattan        Midtown           40.8
##  2  3831 Whole flr w~    4869 LisaRoxan~ Brooklyn         Bedford-Stuy~     40.7
##  3  5121 BlissArtsSp~    7356 Garon      Brooklyn         Bedford-Stuy~     40.7
##  4  5136 Spacious Br~    7378 Rebecca    Brooklyn         Sunset Park       40.7
##  5  5178 Large Furni~    8967 Shunichi   Manhattan        Midtown           40.8
##  6  5203 Cozy Clean ~    7490 MaryEllen  Manhattan        Upper West S~     40.8
##  7  5803 Lovely Room~    9744 Laurie     Brooklyn         South Slope       40.7
##  8  6848 Only 2 stop~   15991 Allen & I~ Brooklyn         Williamsburg      40.7
##  9  6872 Uptown Sanc~   16104 Kae        Manhattan        East Harlem       40.8
## 10  6990 UES Beautif~   16800 Cyn        Manhattan        East Harlem       40.8
## # ... with 38,267 more rows, and 11 more variables: longitude <dbl>,
## #   room_type <chr>, price <dbl>, minimum_nights <dbl>,
## #   number_of_reviews <dbl>, last_review <date>, reviews_per_month <dbl>,
## #   calculated_host_listings_count <dbl>, availability_365 <dbl>,
## #   number_of_reviews_ltm <dbl>, license <lgl>
```

```
glimpse(airbnb_nyc)
```

```
## Rows: 38,277
## Columns: 18
## $ id                             <dbl> 2595, 3831, 5121, 5136, 5178, 5203, 580~
## $ name                           <chr> "Skylit Midtown Castle", "Whole flr w/p~
## $ host_id                        <dbl> 2845, 4869, 7356, 7378, 8967, 7490, 974~
## $ host_name                      <chr> "Jennifer", "LisaRoxanne", "Garon", "Re~
## $ neighbourhood_group            <chr> "Manhattan", "Brooklyn", "Brooklyn", "B~
## $ neighbourhood                  <chr> "Midtown", "Bedford-Stuyvesant", "Bedfo~
## $ latitude                       <dbl> 40.75356, 40.68494, 40.68535, 40.66265,~
## $ longitude                      <dbl> -73.98559, -73.95765, -73.95512, -73.99~
## $ room_type                      <chr> "Entire home/apt", "Entire home/apt", "~
## $ price                          <dbl> 150, 75, 60, 275, 68, 75, 98, 89, 65, 6~
## $ minimum_nights                 <dbl> 30, 1, 30, 5, 2, 2, 4, 30, 30, 30, 27, ~
## $ number_of_reviews              <dbl> 48, 409, 50, 2, 507, 118, 204, 181, 0, ~
## $ last_review                    <date> 2019-11-04, 2021-10-22, 2016-06-05, 20~
## $ reviews_per_month              <dbl> 0.33, 4.86, 0.52, 0.02, 3.68, 0.87, 1.4~
## $ calculated_host_listings_count <dbl> 3, 1, 2, 1, 1, 1, 3, 1, 2, 1, 2, 2, 2, ~
## $ availability_365               <dbl> 338, 194, 365, 123, 192, 0, 322, 179, 3~
## $ number_of_reviews_ltm          <dbl> 0, 32, 0, 1, 33, 0, 23, 1, 0, 1, 0, 42,~
## $ license                        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
```

We see that we have 38,277 rows as well as 18 columns which include listing name, host name, neighborhood, neighborhood group, and room type in addition to many other features.

The most important rule in real estate is "location, location, location" so it is safe to assume that neighborhood or neighborhood group will have some effect on our pricing.

We can also see that there are some "NA" values in our data set. For the "reviews_per_month" column, for our purposes we may assume that they are zero if the current value is "NA." However, there is a chance for some newly introduced inaccuracy if the NA were to be a result of a clerical error.

```r
airbnb_nyc$reviews_per_month[is.na(airbnb_nyc$reviews_per_month)] <- 0
```

We also see some "NA" values for the title field. For this field we can substitute the value to be an empty string. However, there is still the same risk of it being a clerical error.

```r
airbnb_nyc$name[is.na(airbnb_nyc$name)] <- ""
```

Next, we see that the "license" field is "NA" for almost all of the rows. Therefore, we will remove this column since it cannot be used in our models.

```r
airbnb_nyc <- subset(airbnb_nyc, select = -c(license, name, host_name))
```

We can also see that some listings have yet to be rated; therefore, some listings have the last reviewed date as "NA." To combat this, we will substitute an absurd date (Jan 1, 1900) to fill in the missing dates.

```r
airbnb_nyc$last_review[is.na(airbnb_nyc$last_review)] <- as.Date('1900-01-01', format="%Y-%m-%d")
```

After doing so, we can convert the date objects to be a more useful form for us: days since review.

```r
airbnb_nyc <- airbnb_nyc %>%
  mutate(last_review = as.numeric(
    difftime(as.Date('2022-03-31', format="%Y-%m-%d"), airbnb_nyc$last_review, units = "days")
  ))

names(airbnb_nyc)[names(airbnb_nyc) == "last_review"] <- "days_since_review"
head(airbnb_nyc)
```

```
## # A tibble: 6 x 15
##      id host_id neighbourhood_group neighbourhood  latitude longitude room_type
##   <dbl>   <dbl> <chr>               <chr>             <dbl>     <dbl> <chr>
## 1  2595    2845 Manhattan           Midtown            40.8     -74.0 Entire ho~
## 2  3831    4869 Brooklyn            Bedford-Stuyv~     40.7     -74.0 Entire ho~
## 3  5121    7356 Brooklyn            Bedford-Stuyv~     40.7     -74.0 Private r~
## 4  5136    7378 Brooklyn            Sunset Park        40.7     -74.0 Entire ho~
## 5  5178    8967 Manhattan           Midtown            40.8     -74.0 Private r~
## 6  5203    7490 Manhattan           Upper West Si~     40.8     -74.0 Private r~
## # ... with 8 more variables: price <dbl>, minimum_nights <dbl>,
## #   number_of_reviews <dbl>, days_since_review <dbl>, reviews_per_month <dbl>,
## #   calculated_host_listings_count <dbl>, availability_365 <dbl>,
## #   number_of_reviews_ltm <dbl>
```

Looking at the column names, we see that the column with the different boroughs of New York City is named "neighbourhood_group." To make this title more accurate, we will rename it to "borough."

```r
names(airbnb_nyc)[names(airbnb_nyc) == "neighbourhood_group"] <- "borough"
head(airbnb_nyc)
```

```
## # A tibble: 6 x 15
##      id host_id borough   neighbourhood      latitude longitude room_type  price
##   <dbl>   <dbl> <chr>     <chr>                 <dbl>     <dbl> <chr>      <dbl>
```

```
## 1  2595    2845 Manhattan Midtown                40.8     -74.0 Entire ho~   150
## 2  3831    4869 Brooklyn  Bedford-Stuyvesant     40.7     -74.0 Entire ho~    75
## 3  5121    7356 Brooklyn  Bedford-Stuyvesant     40.7     -74.0 Private r~    60
## 4  5136    7378 Brooklyn  Sunset Park            40.7     -74.0 Entire ho~   275
## 5  5178    8967 Manhattan Midtown                40.8     -74.0 Private r~    68
## 6  5203    7490 Manhattan Upper West Side        40.8     -74.0 Private r~    75
## # ... with 7 more variables: minimum_nights <dbl>, number_of_reviews <dbl>,
## #   days_since_review <dbl>, reviews_per_month <dbl>,
## #   calculated_host_listings_count <dbl>, availability_365 <dbl>,
## #   number_of_reviews_ltm <dbl>
```

We also see that there is a column called "neighbourhood." Although this is minor, we should "Americanize" the spelling to "neighborhood."

```
names(airbnb_nyc)[names(airbnb_nyc) == "neighbourhood"] <- "neighborhood"
head(airbnb_nyc)
```

```
## # A tibble: 6 x 15
##      id host_id borough   neighborhood       latitude longitude room_type  price
##   <dbl>   <dbl> <chr>     <chr>                 <dbl>     <dbl> <chr>      <dbl>
## 1  2595    2845 Manhattan Midtown                40.8     -74.0 Entire ho~   150
## 2  3831    4869 Brooklyn  Bedford-Stuyvesant     40.7     -74.0 Entire ho~    75
## 3  5121    7356 Brooklyn  Bedford-Stuyvesant     40.7     -74.0 Private r~    60
## 4  5136    7378 Brooklyn  Sunset Park            40.7     -74.0 Entire ho~   275
## 5  5178    8967 Manhattan Midtown                40.8     -74.0 Private r~    68
## 6  5203    7490 Manhattan Upper West Side        40.8     -74.0 Private r~    75
## # ... with 7 more variables: minimum_nights <dbl>, number_of_reviews <dbl>,
## #   days_since_review <dbl>, reviews_per_month <dbl>,
## #   calculated_host_listings_count <dbl>, availability_365 <dbl>,
## #   number_of_reviews_ltm <dbl>
```

Examining the types of the features, we see that neighborhood, neighborhood group, and room type are represented as "characters" when they are factors. Let's convert those types to factors

```
airbnb_nyc$neighborhood <- as.factor(airbnb_nyc$neighborhood)
airbnb_nyc$borough <- as.factor(airbnb_nyc$borough)
airbnb_nyc$room_type <- as.factor(airbnb_nyc$room_type)
```

We will now examine the data set once more.

```
# Let's explore the features and classes once more
glimpse(airbnb_nyc)
```

```
## Rows: 38,277
## Columns: 15
## $ id                             <dbl> 2595, 3831, 5121, 5136, 5178, 5203, 580~
## $ host_id                        <dbl> 2845, 4869, 7356, 7378, 8967, 7490, 974~
## $ borough                        <fct> Manhattan, Brooklyn, Brooklyn, Brooklyn~
## $ neighborhood                   <fct> "Midtown", "Bedford-Stuyvesant", "Bedfo~
## $ latitude                       <dbl> 40.75356, 40.68494, 40.68535, 40.66265,~
## $ longitude                      <dbl> -73.98559, -73.95765, -73.95512, -73.99~
## $ room_type                      <fct> Entire home/apt, Entire home/apt, Priva~
```

```
## $ price                          <dbl> 150, 75, 60, 275, 68, 75, 98, 89, 65, 6~
## $ minimum_nights                 <dbl> 30, 1, 30, 5, 2, 2, 4, 30, 30, 30, 27, ~
## $ number_of_reviews              <dbl> 48, 409, 50, 2, 507, 118, 204, 181, 0, ~
## $ days_since_review              <dbl> 878, 160, 2125, 235, 143, 1723, 128, 23~
## $ reviews_per_month              <dbl> 0.33, 4.86, 0.52, 0.02, 3.68, 0.87, 1.4~
## $ calculated_host_listings_count <dbl> 3, 1, 2, 1, 1, 1, 3, 1, 2, 1, 2, 2, 2, ~
## $ availability_365               <dbl> 338, 194, 365, 123, 192, 0, 322, 179, 3~
## $ number_of_reviews_ltm          <dbl> 0, 32, 0, 1, 33, 0, 23, 1, 0, 1, 0, 42,~
```

We can see that now all the field types are as expected!

Let's now explore how many unique listings, hosts, neighborhoods, boroughs, and room types there are.

```r
airbnb_nyc %>% summarize(unique_listings = length(unique(id)),
                  unique_hosts = length(unique(host_id)),
                  unique_neighborhoods = length(unique(neighborhood)),
                  unique_boroughs = length(unique(borough)),
                  unique_room_types = length(unique(room_type)))
```

```
## # A tibble: 1 x 5
##   unique_listings unique_hosts unique_neighbor~ unique_boroughs unique_room_typ~
##             <int>        <int>            <int>           <int>            <int>
## 1           38277        25904              222               5                4
```

We see that there are 38,277 unique listings, 25,904 hosts, 222 neighborhoods, 5 neighborhood groups, and 4 room types.

Now let's explore the most expensive airbnb listings.

```r
# Let's now look at the most expensive airbnbs.
airbnb_nyc %>% arrange(-price)
```

```
## # A tibble: 38,277 x 15
##          id   host_id borough   neighborhood  latitude longitude room_type price
##       <dbl>     <dbl> <fct>     <fct>            <dbl>     <dbl> <fct>     <dbl>
##  1 13925864  58480311 Queens    Long Island ~     40.8     -73.9 Entire h~ 10000
##  2 22436899  72390391 Manhattan Upper West S~     40.8     -74.0 Entire h~ 10000
##  3 22985168  71733378 Queens    Astoria           40.8     -73.9 Entire h~ 10000
##  4 31219800 172226912 Manhattan Murray Hill       40.7     -74.0 Shared r~ 10000
##  5 38993493 298338860 Manhattan Midtown           40.7     -74.0 Private ~ 10000
##  6 38993556 298338860 Manhattan Midtown           40.7     -74.0 Private ~ 10000
##  7 38993616 298338860 Manhattan Midtown           40.7     -74.0 Private ~ 10000
##  8 38993679 298338860 Manhattan Midtown           40.8     -74.0 Private ~ 10000
##  9 39100961 220229838 Manhattan Midtown           40.8     -74.0 Private ~ 10000
## 10 39574087 266741420 Manhattan Lower East S~     40.7     -74.0 Private ~ 10000
## # ... with 38,267 more rows, and 7 more variables: minimum_nights <dbl>,
## #   number_of_reviews <dbl>, days_since_review <dbl>, reviews_per_month <dbl>,
## #   calculated_host_listings_count <dbl>, availability_365 <dbl>,
## #   number_of_reviews_ltm <dbl>
```

We see that there are some listings that are $10,000 per night. While these figures would be absurd in most cities, it is totally possible in New York City.

Now let's explore the least expensive listings.

```r
# Now let's look at the least expensive airbnbs.
airbnb_nyc %>% arrange(price)
```

```
## # A tibble: 38,277 x 15
##          id   host_id borough   neighborhood  latitude longitude room_type price
##       <dbl>     <dbl> <fct>     <fct>            <dbl>     <dbl> <fct>     <dbl>
##  1 40560656 273324213 Brooklyn  Williamsburg      40.7     -74.0 Hotel ro~     0
##  2 41740615 268417148 Manhattan Midtown           40.7     -74.0 Hotel ro~     0
##  3 41740622 269311462 Manhattan Upper East S~     40.8     -74.0 Hotel ro~     0
##  4 41792753 197053492 Manhattan Financial Di~     40.7     -74.0 Hotel ro~     0
##  5 42065543 307634016 Manhattan Midtown           40.7     -74.0 Hotel ro~     0
##  6 42065545 310429455 Manhattan Midtown           40.8     -74.0 Hotel ro~     0
##  7 42065547 308721299 Manhattan Hell's Kitch~     40.8     -74.0 Hotel ro~     0
##  8 42065555 309714886 Brooklyn  Williamsburg      40.7     -74.0 Hotel ro~     0
##  9 42065562 307633956 Manhattan Financial Di~     40.7     -74.0 Hotel ro~     0
## 10 42065563 309772430 Bronx     Mott Haven        40.8     -73.9 Hotel ro~     0
## # ... with 38,267 more rows, and 7 more variables: minimum_nights <dbl>,
## #   number_of_reviews <dbl>, days_since_review <dbl>, reviews_per_month <dbl>,
## #   calculated_host_listings_count <dbl>, availability_365 <dbl>,
## #   number_of_reviews_ltm <dbl>
```

We see that some listings have a nightly rate of $0. Although it is possible for a listing to be very cheap, it is improbable at best that a listing is offering a free night's stay. Let's remove these data points from our data set.

```r
airbnb_nyc <- airbnb_nyc %>%
  filter(price > 0) %>%
  arrange(price)
```

Let us now explore the least expensive listings after filtering out the invalid listings.

```r
airbnb_nyc %>% arrange(price)
```

```
## # A tibble: 38,241 x 15
##          id   host_id borough   neighborhood  latitude longitude room_type price
##       <dbl>     <dbl> <fct>     <fct>            <dbl>     <dbl> <fct>     <dbl>
##  1 15932054 103392673 Queens    Astoria           40.8     -73.9 Private ~    10
##  2 16181190 103392673 Queens    Astoria           40.8     -73.9 Private ~    10
##  3 17952277  62685070 Brooklyn  Bushwick          40.7     -73.9 Private ~    10
##  4 26602421 200007278 Manhattan Chelsea           40.8     -74.0 Entire h~    10
##  5 39340742 302174650 Queens    Flushing          40.8     -73.8 Entire h~    10
##  6 47922647  33620899 Queens    Woodside          40.7     -73.9 Entire h~    10
##  7 49409612  24641078 Manhattan Upper East S~     40.8     -74.0 Entire h~    10
##  8 50444487  24641078 Manhattan Upper East S~     40.8     -74.0 Entire h~    10
##  9 51824986 258335548 Manhattan Harlem            40.8     -74.0 Entire h~    10
## 10 21044649 151547086 Bronx     Norwood           40.9     -73.9 Private ~    11
## # ... with 38,231 more rows, and 7 more variables: minimum_nights <dbl>,
## #   number_of_reviews <dbl>, days_since_review <dbl>, reviews_per_month <dbl>,
## #   calculated_host_listings_count <dbl>, availability_365 <dbl>,
## #   number_of_reviews_ltm <dbl>
```

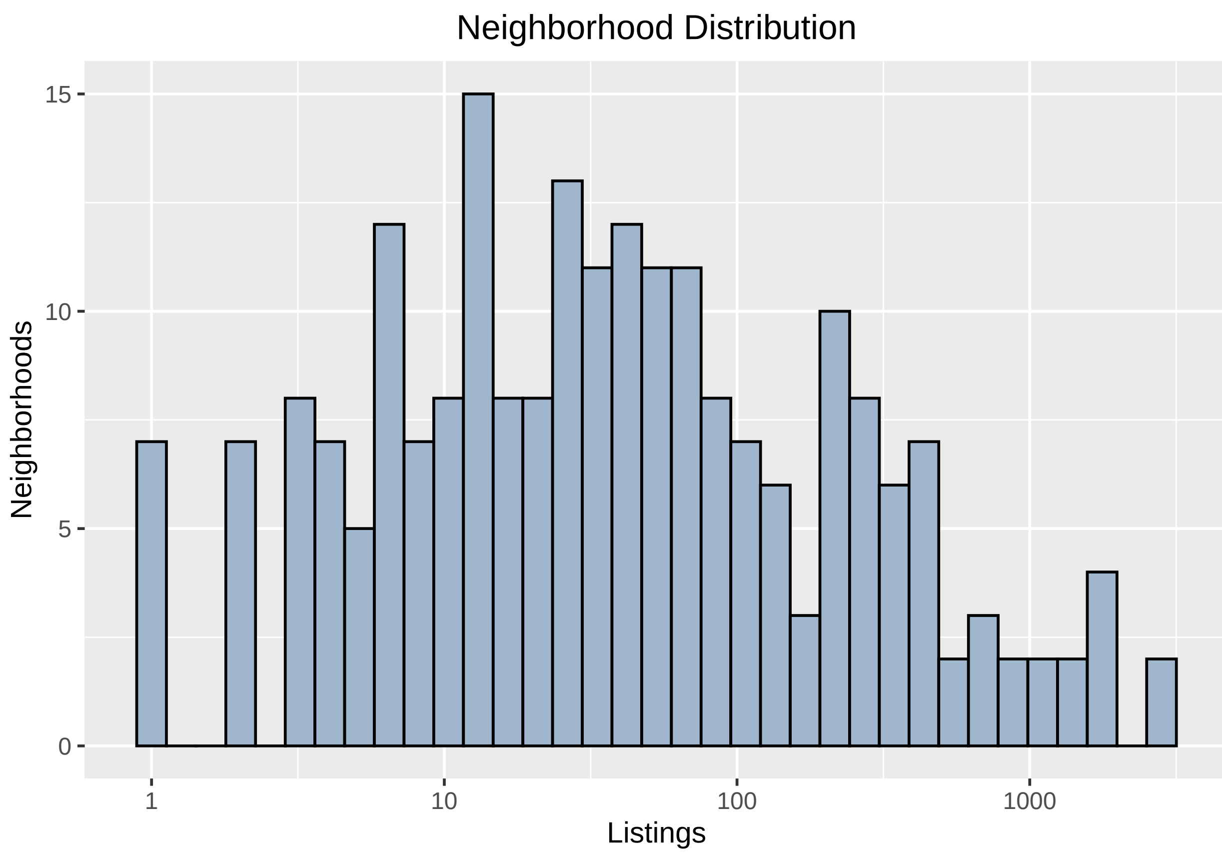
These figures make more sense!

Next, we should examine the distribution of listings across the most important factors. For our use case, the biggest factors will most likely be the neighborhood, availability over the past 365 days, and the room type.

In real estate, the neighborhood holds the greatest impact on price. In pricing short-term rentals, there are vastly different price ranges for private rooms versus hotel rooms. Moreover, depending on how often the rental has been rented in the past year, the price will be "market adjusted" to be more competitive.

Let's examine the distribution of listings across neighborhoods.

```
airbnb_nyc %>%
  group_by(neighborhood) %>%
  summarize(n_listings = n()) %>%
  ggplot(aes(n_listings))+
  geom_histogram(fill = "slategray3", color = "black", bins = 35) +
  scale_x_log10()+
  ggtitle("Neighborhood Distribution")+
  xlab("Listings")+
  ylab("Neighborhoods")+
  theme(plot.title = element_text(hjust = 0.5))
```

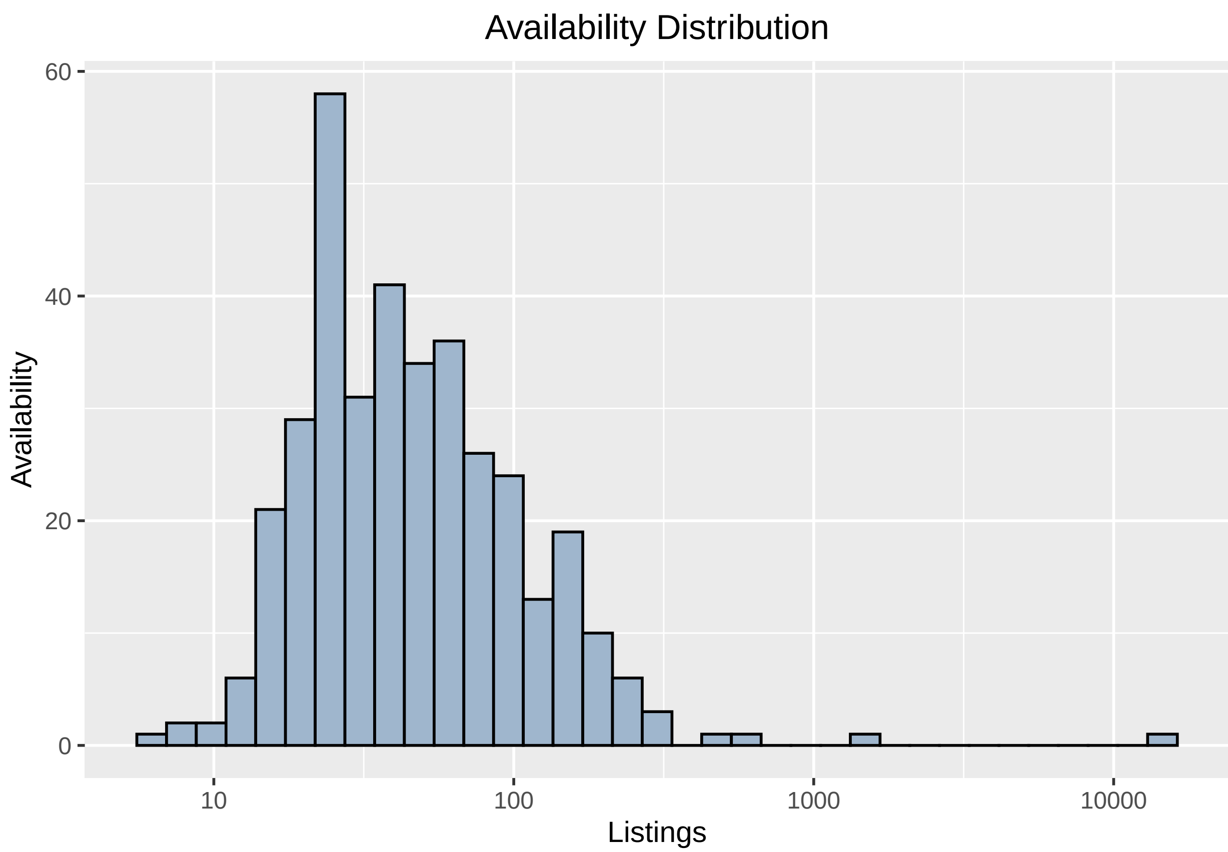


We see an approximately normal distribution.

Next, let's examine the distribution of listings across various availabilities over the past 365 days.

```
airbnb_nyc %>%
  group_by(availability_365) %>%
  summarize(n_listings = n()) %>%
```

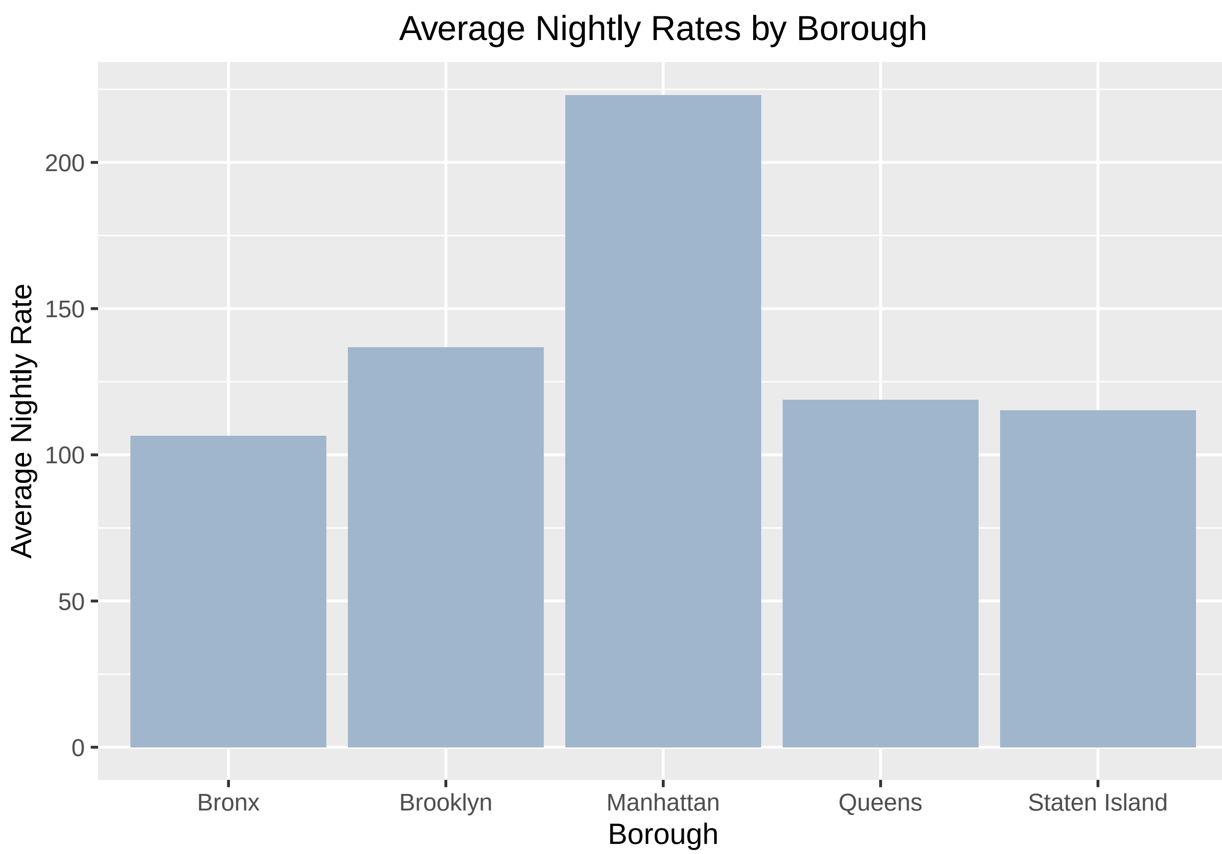
```
  ggplot(aes(n_listings))+
  geom_histogram(fill = "slategray3", color = "black", bins = 35) +
  scale_x_log10()+
  ggtitle("Availability Distribution")+
  xlab("Listings")+
  ylab("Availability")+
  theme(plot.title = element_text(hjust = 0.5))
```



Once again, we see an approximately normal distribution of listings across availabilities!

Next, we will examine the average nightly rate for each of the boroughs.

```
airbnb_nyc %>%
  group_by(borough) %>%
  summarize(avg_nightly_rate = mean(price)) %>%
  ggplot(aes(borough, avg_nightly_rate)) +
  geom_bar(stat="identity", fill = "slategray3") +
  ggtitle("Average Nightly Rates by Borough") +
  xlab("Borough") +
  ylab("Average Nightly Rate") +
  theme(plot.title = element_text(hjust = 0.5))
```



We see that the most expensive borough is Manhattan which makes sense. We see that the least expensive borough is the Bronx which also makes sense.

Now let's examine the most expensive neighborhoods.

```
airbnb_nyc %>%
  group_by(neighborhood) %>%
  summarize(avg_nightly_rate = mean(price), listing_count = n()) %>%
  arrange(-avg_nightly_rate)
```

```
## # A tibble: 222 x 3
##    neighborhood      avg_nightly_rate listing_count
##    <fct>                        <dbl>         <int>
##  1 Jamaica Estates               917.            29
##  2 Fort Wadsworth                800              1
##  3 Tribeca                       482.           141
##  4 Riverdale                     407.             9
##  5 Flatiron District             402.            73
##  6 Briarwood                     394.            31
##  7 Theater District              394.           273
##  8 Midtown                       365.          1626
##  9 SoHo                          304.           250
## 10 Greenwich Village             304.           234
## # ... with 212 more rows
```

We see that Jamaica Estates in Queens is listed as the most expensive neighborhood in New York which doesn't sounds right. This leads us to believe that there are outliers that we need to remove.

As mentioned, the biggest two factors in price will be room type and borough. Knowing how vastly different the price ranges can be for various boroughs and room types, we should clip the outliers for each of the filtered borough and room type subcategories.

To make our code shorter, we will define a function that will handle the outlier identification for us given a specific subcategory.

```
find_outliers <- function(listing_group) {
  lower_percentile <- 0.25
  upper_percentile <- 0.75
  lower_bound <- quantile(listing_group$price, lower_percentile)
  upper_bound <- quantile(listing_group$price, upper_percentile)
  outliers <- listing_group %>%
    filter(price > upper_bound | price < lower_bound)
  outliers
}
```

Next, we will filter by borough and room type and then clip the upper and lower outliers.

```
shared_rooms <- airbnb_nyc %>%
  filter(room_type == "Shared room")

###################################

shared_bronx_rooms <- shared_rooms %>%
  filter(borough == "Bronx")

# Let's first check the dimensions to make sure this is a valid subcategory.
dim(shared_bronx_rooms)
```

```
## [1] 29 15
```

```
outliers <- find_outliers(shared_bronx_rooms)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "p
```

```
###################################

shared_brooklyn_rooms <- shared_rooms %>%
  filter(borough == "Brooklyn")

dim(shared_brooklyn_rooms)
```

```
## [1] 189  15
```

```
outliers <- find_outliers(shared_brooklyn_rooms)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "p
```

```
###################################

shared_manhattan_rooms <- shared_rooms %>%
  filter(borough == "Manhattan")

dim(shared_manhattan_rooms)
```

```
## [1] 244  15
```

```
outliers <- find_outliers(shared_manhattan_rooms)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "p
```

```
###################################

shared_queens_rooms <- shared_rooms %>%
  filter(borough == "Queens")

dim(shared_queens_rooms)
```

```
## [1] 109  15
```

```
outliers <- find_outliers(shared_queens_rooms)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "p
```

```
###################################

shared_si_rooms <- shared_rooms %>%
  filter(borough == "Staten Island")

dim(shared_si_rooms)
```

```
## [1]  1 15
```

```
outliers <- find_outliers(shared_si_rooms)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "p
```

```
###################################
###################################
###################################

private_rooms <- airbnb_nyc %>%
  filter(room_type == "Private room")
```

```
####################################

private_bronx_rooms <- private_rooms %>%
  filter(borough == "Bronx")

dim(private_bronx_rooms)
```

```
## [1] 634  15
```

```
outliers <- find_outliers(private_bronx_rooms)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "pr
```

```
####################################

private_brooklyn_rooms <- private_rooms %>%
  filter(borough == "Brooklyn")

dim(private_brooklyn_rooms)
```

```
## [1] 6989   15
```

```
outliers <- find_outliers(private_brooklyn_rooms)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "pr
```

```
####################################

private_manhattan_rooms <- private_rooms %>%
  filter(borough == "Manhattan")

dim(private_manhattan_rooms)
```

```
## [1] 6158   15
```

```
outliers <- find_outliers(private_manhattan_rooms)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "pr
```

```
####################################

private_queens_rooms <- private_rooms %>%
  filter(borough == "Queens")

dim(private_queens_rooms)
```

```
## [1] 3149   15
```

```r
outliers <- find_outliers(private_queens_rooms)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "p
```

```r
###################################

private_si_rooms <- private_rooms %>%
  filter(borough == "Staten Island")

dim(private_si_rooms)
```

```
## [1] 168  15
```

```r
outliers <- find_outliers(private_si_rooms)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "p
```

```r
###################################
###################################
###################################

entire_place <- airbnb_nyc %>%
  filter(room_type == "Entire home/apt")

###################################

entire_bronx_place <- entire_place %>%
  filter(borough == "Bronx")

dim(entire_bronx_place)
```

```
## [1] 440  15
```

```r
outliers <- find_outliers(entire_bronx_place)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "p
```

```r
###################################

entire_brooklyn_place <- entire_place %>%
  filter(borough == "Brooklyn")

dim(entire_brooklyn_place)
```

```
## [1] 7529   15
```

```r
outliers <- find_outliers(entire_brooklyn_place)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "pr
```

```r
##################################

entire_manhattan_place <- entire_place %>%
  filter(borough == "Manhattan")

dim(entire_manhattan_place)
```

```
## [1] 10188    15
```

```r
outliers <- find_outliers(entire_manhattan_place)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "pr
```

```r
##################################

entire_queens_place <- entire_place %>%
  filter(borough == "Queens")

dim(entire_queens_place)
```

```
## [1] 2056   15
```

```r
outliers <- find_outliers(entire_queens_place)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "pr
```

```r
##################################

entire_si_place <- entire_place %>%
  filter(borough == "Staten Island")

dim(entire_si_place)
```

```
## [1] 184  15
```

```r
outliers <- find_outliers(entire_si_place)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "pr
```

```r
####################################
####################################
####################################

hotel_room <- airbnb_nyc %>%
  filter(room_type == "Hotel room")

####################################

bronx_hotel <- hotel_room %>%
  filter(borough == "Bronx")

dim(bronx_hotel)
```

```
## [1]  0 15
```

```r
# There doesn't look to be any hotels in the Bronx

####################################

brooklyn_hotel <- hotel_room %>%
  filter(borough == "Brooklyn")

dim(brooklyn_hotel)
```

```
## [1]  5 15
```

```r
outliers <- find_outliers(brooklyn_hotel)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "pr
```

```r
####################################

manhattan_hotel <- hotel_room %>%
  filter(borough == "Manhattan")

dim(manhattan_hotel)
```

```
## [1] 160  15
```

```r
outliers <- find_outliers(manhattan_hotel)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "pr
```

```r
####################################

queens_hotel <- hotel_room %>%
  filter(borough == "Queens")

dim(queens_hotel)
```

```
## [1]  9 15
```

```
outliers <- find_outliers(queens_hotel)
airbnb_nyc <- anti_join(airbnb_nyc, outliers)
```

```
## Joining, by = c("id", "host_id", "borough", "neighborhood", "latitude", "longitude", "room_type", "p
```

```
###################################

si_hotel <- hotel_room %>%
  filter(borough == "Staten Island")

dim(si_hotel)
```

```
## [1]  0 15
```

Now that the outliers have been removed, we are now ready to split our data sets into the training, test, and validation sets.

```
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y = airbnb_nyc$price, times = 1, p = 0.1, list = FALSE)
airbnb_train <- airbnb_nyc[-test_index,]
validation <- airbnb_nyc[test_index,]

set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y = airbnb_train$price, times = 1, p = 0.1, list = FALSE)
train_set <- airbnb_train[-test_index,]
test_set <- airbnb_train[test_index,]
```

Next, we need to make sure our test and validation sets contain the same factors that we will be building our models on: neighborhood, room type, and availability over the past 365 days.

```
test_set <- test_set %>%
  semi_join(train_set, by = "neighborhood") %>%
  semi_join(train_set, by = "room_type")%>%
  semi_join(train_set, by = "availability_365")

validation <- validation %>%
  semi_join(train_set, by = "neighborhood") %>%
  semi_join(train_set, by = "room_type") %>%
  semi_join(train_set, by = "availability_365")
```

For our modeling, we will use least squared estimates and incorporate the three key factors that would most likely impact pricing: neighborhood, availability from the past 365 days, and room type. Next, we will build a model using matrix factorization.

The first model we will build is the "mean model" where we will predict all of the test set prices to be the mean of the training set prices. This will be our "benchmark" model.

Next, we will build our first least squared estimates model with the neighborhood effects or bias factored in. Then we will build another model with the neighborhood and availability effects incorporated. Lastly, we will build a model with the neighborhood, availability, and room type effects incorporated.

Then, we incorporate regularization to see if we could tune our models to be even more accurate.

Following this, we will build our matrix factorization model.

Lastly, we will test all of our models once more on our validation set.

To measure the performance of our models, we will use root mean squared error as our loss function. The function for calculating the RMSE is as shown below.

```
RMSE <- function(true_price, predicted_price){
  sqrt(mean((true_price - predicted_price)^2))
}
```

RESULTS

Let's first start off with our mean model. We will predict the mean of the training set prices for all of the listings in the training set.

In validating our models, we will first test on our test set.

```
mu_hat <- mean(train_set$price)
naive_rmse <- RMSE(test_set$price, mu_hat)
```

Let's now visualize the results in a chart.

```
results <- tibble(Model_Type = "Mean Model", RMSE = naive_rmse) %>%
  mutate(RMSE = sprintf("%0.4f", RMSE))
results
```

```
## # A tibble: 1 x 2
##   Model_Type RMSE
##   <chr>      <chr>
## 1 Mean Model 60.1709
```

We see an RMSE of 60.1709. This isn't too bad of an error for a listing that is priced in the thousands but a terrible one for a listing that's in the tens. Let's see if we can get this number down.

We will now build our first model incorporating the neighborhood effects and visualize the results.

```
mu <- mean(train_set$price)
bi_avgs <- train_set %>%
  group_by(neighborhood) %>%
  summarize(b_i = mean(price - mu))

bi_predictions <- mu + test_set %>%
  left_join(bi_avgs, by = "neighborhood") %>%
```

```r
  pull(b_i)
bi_rmse <- RMSE(test_set$price, bi_predictions)
# Let's visualize the results in a table
results <- tibble(Model_Type = c(
  "Mean Model",
  "Neighborhood Model"
), RMSE = c(
  naive_rmse,
  bi_rmse
)) %>%
  mutate(RMSE = sprintf("%0.4f", RMSE))
results
```

```
## # A tibble: 2 x 2
##   Model_Type         RMSE
##   <chr>              <chr>
## 1 Mean Model         60.1709
## 2 Neighborhood Model 52.2709
```

We see our RMSE drop to 52.2709! Let's see if we can get that number even lower once we incorporate the availability over the past 365 days.

```r
bu_avgs <- train_set %>%
  left_join(bi_avgs, by = "neighborhood") %>%
  group_by(availability_365) %>%
  summarize(b_u = mean(price - mu - b_i))
bu_predictions <- test_set %>%
  left_join(bi_avgs, by = "neighborhood") %>%
  left_join(bu_avgs, by = "availability_365")  %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
bu_rmse <- RMSE(test_set$price, bu_predictions)

# Let's once again visualize the results in a table
results <- tibble(Model_Type = c(
  "Mean Model",
  "Neighborhood Model",
  "Neighborhood & Availability 365 Model"
), RMSE = c(
  naive_rmse,
  bi_rmse,
  bu_rmse
)) %>%
  mutate(RMSE = sprintf("%0.4f", RMSE))
results
```

```
## # A tibble: 3 x 2
##   Model_Type                            RMSE
##   <chr>                                 <chr>
## 1 Mean Model                            60.1709
## 2 Neighborhood Model                    52.2709
## 3 Neighborhood & Availability 365 Model 52.8127
```

We see our RMSE drop even further to 52.8127. Now let's see how room type affects our predictions.

```
bv_avgs <- train_set %>%
  left_join(bi_avgs, by = "neighborhood") %>%
  left_join(bu_avgs, by = "availability_365") %>%
  group_by(room_type) %>%
  summarize(b_v = mean(price - mu - b_i - b_u))
bv_predictions <- test_set %>%
  left_join(bi_avgs, by = "neighborhood") %>%
  left_join(bu_avgs, by = "availability_365") %>%
  left_join(bv_avgs, by = "room_type") %>%
  mutate(pred = mu + b_i + b_u + b_v) %>%
  pull(pred)
bv_rmse <- RMSE(test_set$price, bv_predictions)

# Let's once again visualize the results in a table
results <- tibble(Model_Type = c(
  "Mean Model",
  "Neighborhood Model",
  "Neighborhood & Availability 365 Model",
  "Neighborhood, Availability 365, & Room Type Model"
), RMSE = c(
  naive_rmse,
  bi_rmse,
  bu_rmse,
  bv_rmse
)) %>%
  mutate(RMSE = sprintf("%0.4f", RMSE))
results
```

```
## # A tibble: 4 x 2
##   Model_Type                                        RMSE
##   <chr>                                             <chr>
## 1 Mean Model                                        60.1709
## 2 Neighborhood Model                                52.2709
## 3 Neighborhood & Availability 365 Model             52.8127
## 4 Neighborhood, Availability 365, & Room Type Model 34.2903
```

We can see a great improvement with our RMSE dropping to 34.2903.

Next we will tune our models to refine our accuracy even further.

First, we will tune our neighborhood and availability effects model. To do so, we need to find the lambda value which minimizes our RMSE.

```
lambdas <- seq(0, 100, 1)
RMSES <- sapply(lambdas, function(l){
  train_mean <- mean(train_set$price)

  bi <- train_set %>%
    group_by(neighborhood) %>%
    summarize(bi = sum(price - train_mean)/(n() + l))

  bu <- train_set %>%
```

```
    left_join(bi, by='neighborhood') %>%
    group_by(availability_365) %>%
    summarize(bu = sum(price - bi - train_mean)/(n() + l))

  predictions <- test_set %>%
    left_join(bi, by = "neighborhood") %>%
    left_join(bu, by = "availability_365") %>%
    mutate(pred = train_mean + bi +  bu) %>%
    .$pred

  return(RMSE(predictions, test_set$price))
})

na_model_lambda <- lambdas[which.min(RMSES)]
na_model_lambda
```

```
## [1] 13
```

We see that a lambda of 13 minimizes our RMSE. Let's now incorporate this lambda into our model and revisit its performance.

```
b_i <- train_set %>%
  group_by(neighborhood) %>%
  summarize(b_i = sum(price - mu)/(n() + na_model_lambda))
b_u <- train_set %>%
  left_join(b_i, by = "neighborhood") %>%
  group_by(availability_365) %>%
  summarize(b_u = sum(price - b_i - mu)/(n() + na_model_lambda))
bu_predictions <- test_set %>%
  left_join(b_i, by = "neighborhood") %>%
  left_join(b_u, by = "availability_365") %>%
  mutate(predictions = mu + b_i + b_u) %>%
  .$predictions

bu_reg_rmse <- RMSE(test_set$price, bu_predictions)

results <- tibble(Model_Type = c(
  "Mean Model",
  "Neighborhood Model",
  "Neighborhood & Availability 365 Model",
  "Neighborhood, Availability 365, & Room Type Model",
  "Neighborhood, Availability 365 Model w/ Reg"
), RMSE = c(
  naive_rmse,
  bi_rmse,
  bu_rmse,
  bv_rmse,
  bu_reg_rmse
)) %>%
  mutate(RMSE = sprintf("%0.4f", RMSE))
results
```

```
## # A tibble: 5 x 2
```

```
##   Model_Type                                        RMSE
##   <chr>                                             <chr>
## 1 Mean Model                                        60.1709
## 2 Neighborhood Model                                52.2709
## 3 Neighborhood & Availability 365 Model             52.8127
## 4 Neighborhood, Availability 365, & Room Type Model 34.2903
## 5 Neighborhood, Availability 365 Model w/ Reg       52.1545
```

We see a slight improvement in performance from our original model to achieve an RMSE of 52.1545.

Let's now tune our neighborhood, availability, and room type effects model. We will once again find the lambda that minimizes our RMSE.

```
lambdas <- seq(0, 100, 1)
RMSES <- sapply(lambdas, function(l){
  train_mean <- mean(train_set$price)

  bi <- train_set %>%
    group_by(neighborhood) %>%
    summarize(bi = sum(price - train_mean)/(n() + l))

  bu <- train_set %>%
    left_join(bi, by='neighborhood') %>%
    group_by(availability_365) %>%
    summarize(bu = sum(price - bi - train_mean)/(n() + l))

  br <- train_set %>%
    left_join(bi, by = "neighborhood") %>%
    left_join(bu, by = "availability_365") %>%
    group_by(room_type) %>%
    summarize(br = sum(price - bi - bu - train_mean)/(n() + l))

  predictions <- test_set %>%
    left_join(bi, by = "neighborhood") %>%
    left_join(bu, by = "availability_365") %>%
    left_join(br, by = "room_type") %>%
    mutate(pred = train_mean + bi +  bu + br) %>%
    .$pred

  return(RMSE(predictions, test_set$price))
})

nar_model_lambda <- lambdas[which.min(RMSES)]
nar_model_lambda
```

```
## [1] 70
```

We see a lambda of 70 minimizes our RMSE. Let's now rebuild our model with the lambda incorporated.

```
b_i <- train_set %>%
  group_by(neighborhood) %>%
  summarize(b_i = sum(price - mu)/(n() + nar_model_lambda))
b_u <- train_set %>%
```

```r
  left_join(b_i, by = "neighborhood") %>%
  group_by(availability_365) %>%
  summarize(b_u = sum(price - b_i - mu)/(n() + nar_model_lambda))
b_r <- train_set %>%
  left_join(b_i, by = "neighborhood") %>%
  left_join(b_u, by = "availability_365") %>%
  group_by(room_type) %>%
  summarize(b_r = sum(price - b_i - b_u - mu)/(n() + nar_model_lambda))

bv_predictions <- test_set %>%
  left_join(b_i, by = "neighborhood") %>%
  left_join(b_u, by = "availability_365") %>%
  left_join(b_r, by = "room_type") %>%
  mutate(predictions = mu + b_i + b_u + b_r) %>%
  .$predictions

bv_reg_rmse <- RMSE(test_set$price, bv_predictions)

# Let's visualize the results once more.
results <- tibble(Model_Type = c(
  "Mean Model",
  "Neighborhood Model",
  "Neighborhood & Availability 365 Model",
  "Neighborhood, Availability 365, & Room Type Model",
  "Neighborhood, Availability 365 Model w/ Reg",
  "Neighborhood, Availability 365, & Room Type w/ Reg"
), RMSE = c(
  naive_rmse,
  bi_rmse,
  bu_rmse,
  bv_rmse,
  bu_reg_rmse,
  bv_reg_rmse
)) %>%
  mutate(RMSE = sprintf("%0.4f", RMSE))
results
```

```
## # A tibble: 6 x 2
##   Model_Type                                         RMSE
##   <chr>                                              <chr>
## 1 Mean Model                                         60.1709
## 2 Neighborhood Model                                 52.2709
## 3 Neighborhood & Availability 365 Model              52.8127
## 4 Neighborhood, Availability 365, & Room Type Model  34.2903
## 5 Neighborhood, Availability 365 Model w/ Reg        52.1545
## 6 Neighborhood, Availability 365, & Room Type w/ Reg 30.9987
```

We see a significant improvement to 30.9987 which is around a 10% improvement compared to the performance of our original model.

Lastly, we will build a model with matrix factorization. To build our model, we will be using the "recosystem" library.

```r
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
train_reco <- with(train_set, data_memory(user_index = neighborhood, item_index = room_type, rating = pr
test_reco <- with(test_set, data_memory(user_index = neighborhood, item_index = room_type, rating = pric
r <- Reco()

para_reco <- r$tune(train_reco, opts = list(dim = c(20, 30),
                                            costp_l2 = c(0.01, 0.1),
                                            costq_l2 = c(0.01, 0.1),
                                            lrate = c(0.01, 0.1),
                                            nthread = 4,
                                            niter = 10))

r$train(train_reco, opts = c(para_reco$min, nthread = 4, niter = 30))
```

```
## iter      tr_rmse          obj
##    0      37.1414   2.2264e+07
##    1      38.4475   2.3830e+07
##    2      34.1038   1.8810e+07
##    3      32.8935   1.7516e+07
##    4      32.2435   1.6844e+07
##    5      31.7052   1.6294e+07
##    6      31.1552   1.5745e+07
##    7      30.8911   1.5484e+07
##    8      30.4718   1.5073e+07
##    9      30.2449   1.4854e+07
##   10      29.8979   1.4519e+07
##   11      29.7188   1.4352e+07
##   12      29.4548   1.4098e+07
##   13      29.1820   1.3847e+07
##   14      29.0579   1.3731e+07
##   15      28.9560   1.3637e+07
##   16      28.7895   1.3484e+07
##   17      28.6631   1.3369e+07
##   18      28.4848   1.3204e+07
##   19      28.4589   1.3183e+07
##   20      28.3380   1.3074e+07
##   21      28.2692   1.3012e+07
##   22      28.1821   1.2934e+07
##   23      28.1069   1.2865e+07
##   24      28.0107   1.2779e+07
##   25      27.9444   1.2721e+07
##   26      27.8777   1.2661e+07
##   27      27.8237   1.2614e+07
##   28      27.8236   1.2613e+07
##   29      27.7705   1.2566e+07
```

```r
results_reco <- r$predict(test_reco, out_memory())
```

```r
factorization_rmse <- RMSE(results_reco, test_set$price)

# Let's visualize the results.

results <- tibble(Model_Type = c(
  "Mean Model",
  "Neighborhood Model",
  "Neighborhood & Availability 365 Model",
  "Neighborhood, Availability 365, & Room Type Model",
  "Neighborhood, Availability 365 Model w/ Reg",
  "Neighborhood, Availability 365, & Room Type w/ Reg",
  "Matrix Factorization"
), RMSE = c(
  naive_rmse,
  bi_rmse,
  bu_rmse,
  bv_rmse,
  bu_reg_rmse,
  bv_reg_rmse,
  factorization_rmse
)) %>%
  mutate(RMSE = sprintf("%0.4f", RMSE))
results
```

```
## # A tibble: 7 x 2
##   Model_Type                                         RMSE
##   <chr>                                              <chr>
## 1 Mean Model                                         60.1709
## 2 Neighborhood Model                                 52.2709
## 3 Neighborhood & Availability 365 Model              52.8127
## 4 Neighborhood, Availability 365, & Room Type Model  34.2903
## 5 Neighborhood, Availability 365 Model w/ Reg        52.1545
## 6 Neighborhood, Availability 365, & Room Type w/ Reg 30.9987
## 7 Matrix Factorization                               28.6495
```

We see a significant improvement in our model to achieve an RMSE under 30!

Let's now retest our models on our validation set to further analyze their efficacy.

As a benchmark, we will start with our mean model

```r
train_mean <- mean(train_set$price)
validation_mean_model <- RMSE(validation$price, train_mean)

results <- tibble(Model_Type = ("Mean Model"),
                  Validation_RMSE = (validation_mean_model)) %>%
  mutate(Validation_RMSE = sprintf("%0.5f", Validation_RMSE))
results
```

```
## # A tibble: 1 x 2
##   Model_Type Validation_RMSE
##   <chr>      <chr>
## 1 Mean Model 61.40009
```

We see an RMSE of 61.40009, not too far off from the value for the test set.

Let's now build our neighborhood effects model.

```
bi <- train_set %>%
  group_by(neighborhood) %>%
  summarize(b_i = mean(price - train_mean))
bi_prediction <- train_mean + validation %>%
  left_join(bi, by = "neighborhood") %>%
  .$b_i
bi_rmse_validation <- RMSE(validation$price, bi_prediction)
results <- tibble(Model_Type = c(
  "Mean Model",
  "Neighborhood Model"
),
Validation_RMSE = c(
  validation_mean_model,
  bi_rmse_validation
)) %>%
  mutate(Validation_RMSE = sprintf("%0.5f", Validation_RMSE))
results
```

```
## # A tibble: 2 x 2
##   Model_Type         Validation_RMSE
##   <chr>              <chr>
## 1 Mean Model         61.40009
## 2 Neighborhood Model 52.68041
```

We see an RMSE of 52.68041 - slightly worse than how our model did on the test set.

Now let's examine our neighborhood and availability effects model.

```
bi <- train_set %>%
  group_by(neighborhood) %>%
  summarize(b_i = mean(price - train_mean))
bu <- train_set %>%
  left_join(bi, by = "neighborhood") %>%
  group_by(availability_365) %>%
  summarize(b_u = mean(price - train_mean - b_i))
bu_prediction <- validation %>%
  left_join(bi, by = "neighborhood") %>%
  left_join(bu, by = "availability_365") %>%
  mutate(predictions = train_mean + b_i + b_u) %>%
  .$predictions
bu_rmse_validation <- RMSE(validation$price, bu_prediction)
results <- tibble(Model_Type = c(
  "Mean Model",
  "Neighborhood Model",
  "Neighborhood & Availability 365 Model"
),
Validation_RMSE = c(
  validation_mean_model,
  bi_rmse_validation,
  bu_rmse_validation
```

```r
)) %>%
  mutate(Validation_RMSE = sprintf("%0.5f", Validation_RMSE))
results
```

```
## # A tibble: 3 x 2
##   Model_Type                             Validation_RMSE
##   <chr>                                  <chr>
## 1 Mean Model                             61.40009
## 2 Neighborhood Model                     52.68041
## 3 Neighborhood & Availability 365 Model 52.53051
```

We achieved an RMSE of 52.53051 - slightly better than the performance on the test set.

Now let's test our neighborhood, availability, & room type model.

```r
bi <- train_set %>%
  group_by(neighborhood) %>%
  summarize(b_i = mean(price - train_mean))
bu <- train_set %>%
  left_join(bi, by = "neighborhood") %>%
  group_by(availability_365) %>%
  summarize(b_u = mean(price - train_mean - b_i))
br <- train_set %>%
  left_join(bi, by = "neighborhood") %>%
  left_join(bu, by = "availability_365") %>%
  group_by(room_type) %>%
  summarize(b_r = mean(price - train_mean - b_i - b_u))
bv_prediction <- validation %>%
  left_join(bi, by = "neighborhood") %>%
  left_join(bu, by = "availability_365") %>%
  left_join(br, by = "room_type") %>%
  mutate(predictions = train_mean + b_i + b_u + b_r) %>%
  .$predictions
bv_val_rmse <- RMSE(validation$price, bv_prediction)
results <- tibble(Model_Type = c(
  "Mean Model",
  "Neighborhood Model",
  "Neighborhood & Availability 365 Model",
  "Neighborhood, Availability 365, & Room Type Model"
),
Validation_RMSE = c(
  validation_mean_model,
  bi_rmse_validation,
  bu_rmse_validation,
  bv_val_rmse
)) %>%
  mutate(Validation_RMSE = sprintf("%0.5f", Validation_RMSE))
results
```

```
## # A tibble: 4 x 2
##   Model_Type                                         Validation_RMSE
##   <chr>                                              <chr>
## 1 Mean Model                                         61.40009
```

```
## 2 Neighborhood Model                                52.68041
## 3 Neighborhood & Availability 365 Model             52.53051
## 4 Neighborhood, Availability 365, & Room Type Model 34.61761
```

We see an RMSE of 34.61761 - slightly worse than the test set's RMSE.

Now let's see how our regularized neighborhood and availability model does.

```r
b_i <- train_set %>%
  group_by(neighborhood) %>%
  summarize(b_i = sum(price - mu)/(n() + na_model_lambda))
b_u <- train_set %>%
  left_join(b_i, by = "neighborhood") %>%
  group_by(availability_365) %>%
  summarize(b_u = sum(price - b_i - mu)/(n() + na_model_lambda))
bu_val_predictions <- validation %>%
  left_join(b_i, by = "neighborhood") %>%
  left_join(b_u, by = "availability_365") %>%
  mutate(predictions = mu + b_i + b_u) %>%
  .$predictions

bu_val_reg_rmse <- RMSE(validation$price, bu_val_predictions)

results <- tibble(Model_Type = c(
  "Mean Model",
  "Neighborhood Model",
  "Neighborhood & Availability 365 Model",
  "Neighborhood, Availability 365, & Room Type Model",
  "Neighborhood & Availability 365 Model w/ Reg"
),
Validation_RMSE = c(
  validation_mean_model,
  bi_rmse_validation,
  bu_rmse_validation,
  bv_val_rmse,
  bu_val_reg_rmse
)) %>%
  mutate(Validation_RMSE = sprintf("%0.5f", Validation_RMSE))
results
```

```
## # A tibble: 5 x 2
##   Model_Type                                        Validation_RMSE
##   <chr>                                             <chr>
## 1 Mean Model                                        61.40009
## 2 Neighborhood Model                                52.68041
## 3 Neighborhood & Availability 365 Model             52.53051
## 4 Neighborhood, Availability 365, & Room Type Model 34.61761
## 5 Neighborhood & Availability 365 Model w/ Reg      52.37774
```

We achieve an RMSE of 52.37774 - slight worse than our test set RMSE.

Now let's see how our regularized neighborhood, availability, and room type model performs.

```r
b_i <- train_set %>%
  group_by(neighborhood) %>%
  summarize(b_i = sum(price - mu)/(n() + nar_model_lambda))
b_u <- train_set %>%
  left_join(b_i, by = "neighborhood") %>%
  group_by(availability_365) %>%
  summarize(b_u = sum(price - b_i - mu)/(n() + nar_model_lambda))
b_v <- train_set %>%
  left_join(b_i, by = "neighborhood") %>%
  left_join(b_u, by = "availability_365") %>%
  group_by(room_type) %>%
  summarize(b_v = sum(price - b_i - b_u - mu)/(n() + nar_model_lambda))

bv_val_predictions <- validation %>%
  left_join(b_i, by = "neighborhood") %>%
  left_join(b_u, by = "availability_365") %>%
  left_join(b_v, by = "room_type") %>%
  mutate(predictions = mu + b_i + b_u + b_v) %>%
  .$predictions

bv_val_reg_rmse <- RMSE(validation$price, bv_val_predictions)

results <- tibble(Model_Type = c(
  "Mean Model",
  "Neighborhood Model",
  "Neighborhood & Availability 365 Model",
  "Neighborhood, Availability 365, & Room Type Model",
  "Neighborhood, Availability 365 Model w/ Reg",
  "Neighborhood, Availability 365, & Room Type w/ Reg"
), Validation_RMSE = c(
  validation_mean_model,
  bi_rmse_validation,
  bu_rmse_validation,
  bv_val_rmse,
  bu_val_reg_rmse,
  bv_val_reg_rmse
)) %>%
  mutate(Validation_RMSE = sprintf("%0.4f", Validation_RMSE))
results
```

```
## # A tibble: 6 x 2
##   Model_Type                                         Validation_RMSE
##   <chr>                                              <chr>
## 1 Mean Model                                         61.4001
## 2 Neighborhood Model                                 52.6804
## 3 Neighborhood & Availability 365 Model              52.5305
## 4 Neighborhood, Availability 365, & Room Type Model  34.6176
## 5 Neighborhood, Availability 365 Model w/ Reg        52.3777
## 6 Neighborhood, Availability 365, & Room Type w/ Reg 31.8404
```

We achieved an RMSE of 31.8404 - slightly worse than on the test set.

Lastly, let's see how our matrix factorization model does on our validation set.

```r
val_reco <- with(validation, data_memory(
  user_index = neighborhood,
  item_index = room_type,
  rating = price))
results_reco <- r$predict(val_reco, out_memory())

factorization_val_rmse <- RMSE(results_reco, validation$price)

results <- tibble(Model_Type = c(
  "Mean Model",
  "Neighborhood Model",
  "Neighborhood & Availability 365 Model",
  "Neighborhood, Availability 365, & Room Type Model",
  "Neighborhood, Availability 365 Model w/ Reg",
  "Neighborhood, Availability 365, & Room Type w/ Reg",
  "Matrix Factorization"
), Validation_RMSE = c(
  validation_mean_model,
  bi_rmse_validation,
  bu_rmse_validation,
  bv_val_rmse,
  bu_val_reg_rmse,
  bv_val_reg_rmse,
  factorization_val_rmse
)) %>%
  mutate(Validation_RMSE = sprintf("%0.4f", Validation_RMSE))
results
```

```
## # A tibble: 7 x 2
##   Model_Type                                         Validation_RMSE
##   <chr>                                              <chr>
## 1 Mean Model                                         61.4001
## 2 Neighborhood Model                                 52.6804
## 3 Neighborhood & Availability 365 Model              52.5305
## 4 Neighborhood, Availability 365, & Room Type Model  34.6176
## 5 Neighborhood, Availability 365 Model w/ Reg        52.3777
## 6 Neighborhood, Availability 365, & Room Type w/ Reg 31.8404
## 7 Matrix Factorization                               28.9615
```

We were still able to achieve an RMSE under 30!

Now let's compare the RMSEs of the models on the test and validation sets.

```r
results <- tibble(Model_Type = c(
  "Mean Model",
  "Neighborhood Model",
  "Neighborhood & Availability 365 Model",
  "Neighborhood, Availability 365, & Room Type Model",
  "Neighborhood, Availability 365 Model w/ Reg",
  "Neighborhood, Availability 365, & Room Type w/ Reg",
  "Matrix Factorization"
),
RMSE = c(
```

```
  naive_rmse,
  bi_rmse,
  bu_rmse,
  bv_rmse,
  bu_reg_rmse,
  bv_reg_rmse,
  factorization_rmse
),
Validation_RMSE = c(
  validation_mean_model,
  bi_rmse_validation,
  bu_rmse_validation,
  bv_val_rmse,
  bu_val_reg_rmse,
  bv_val_reg_rmse,
  factorization_val_rmse
)) %>%
  mutate(Validation_RMSE = sprintf("%0.5f", Validation_RMSE))
results
```

```
## # A tibble: 7 x 3
##   Model_Type                                          RMSE Validation_RMSE
##   <chr>                                              <dbl> <chr>
## 1 Mean Model                                          60.2 61.40009
## 2 Neighborhood Model                                  52.3 52.68041
## 3 Neighborhood & Availability 365 Model               52.8 52.53051
## 4 Neighborhood, Availability 365, & Room Type Model   34.3 34.61761
## 5 Neighborhood, Availability 365 Model w/ Reg         52.2 52.37774
## 6 Neighborhood, Availability 365, & Room Type w/ Reg  31.0 31.84040
## 7 Matrix Factorization                                28.6 28.96151
```

```
results %>% knitr::kable()
```

Model_Type	RMSE	Validation_RMSE
Mean Model	60.17093	61.40009
Neighborhood Model	52.27090	52.68041
Neighborhood & Availability 365 Model	52.81267	52.53051
Neighborhood, Availability 365, & Room Type Model	34.29026	34.61761
Neighborhood, Availability 365 Model w/ Reg	52.15447	52.37774
Neighborhood, Availability 365, & Room Type w/ Reg	30.99868	31.84040
Matrix Factorization	28.64953	28.96151

We see our models performed similarly on both the test and validation sets. The matrix factorization models gave us the lowest RMSE when tested on the validation set.

Considering the wide range of prices available in our listings, it is remarkable that we are able to achieve an RMSE under 30.

CONCLUSION

In determining the price of an airbnb, the most significant factors are its location, room type, and availability. The importance of location goes without saying. For room type, the price range for a shared room versus for a hotel room are vastly different. As for availability, the more vacant a rental is, the likelier the landlord is to reduce the price. The more popular the rental is, the price is likely to be higher.

Price prediction systems are a large subset of real world machine learning uses. The particular tool we've built above could be helpful in suggesting a price to the author of a new listing, indicating to a potential renter that the price is a good deal, or suggesting to a landlord to alter their prices based on the current market.

Some limitations are with the representation of different neighborhoods in our data set. Some neighborhoods are simplier more popular for short-term rentals over others. Therefore, for less popular neighborhoods it may be difficult to accurately predict a listing's price.

One way to improve the accuracy of our algorithm would be via ensembling with other models. Another method could be via incorporating other factors into our models such as minimum nights or number of reviews.