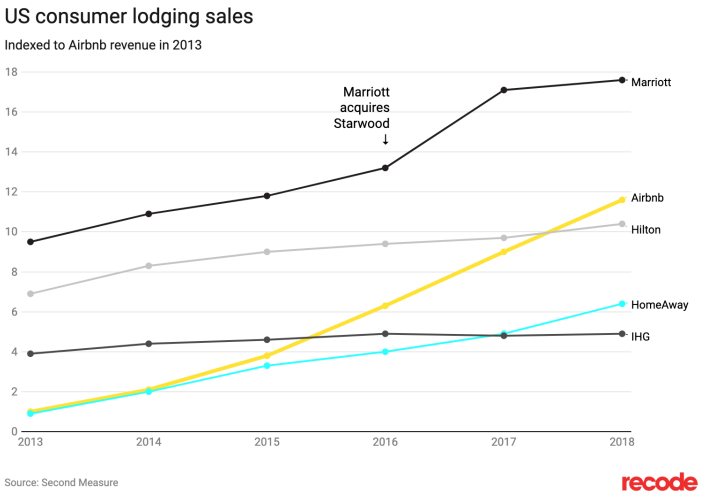
Predicting Host Ratings of Austin Airbnb Listings

Gabrielle Pincomb

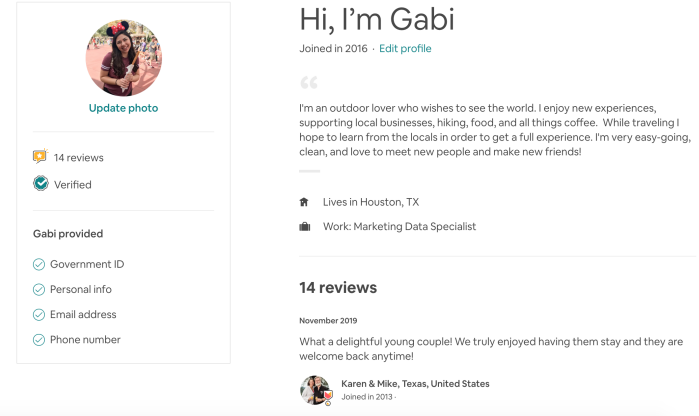
12/02/2019

Project Background

Airbnb is a homesharing platform, founded in 2008. The service allows for accomodations outside of the typical hotel experience. Homeowners, renters, or property owners, get to rent their home, or a space in their home, to travelers seeking housing. This service allows for hosts to be able to make some extra income and allows for the guests to get a stay that is typically cheaper than a hotel room. Airbnb has taken the hotel industry by storm. Consumer spending in Airbnb is growing faster than spending in the hotel industry. Airbnb has changed the way we think of hospitality and lodging, and continues to impact our expectations in accomodations.

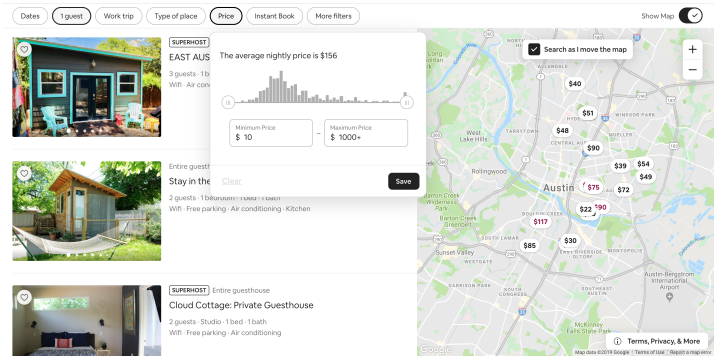


Staying at a complete stranger's house can be a little bit unsettling, but Airbnb has systems in place to help establish trust. Each user (both guest and host) must have a profile and verified identification(s). This allows for guests and hosts to learn more about each other prior to their stay. Additionally, this allows for hosts to decide to allow the guest to stay at their listing or not.



At the completion of the guest's stay, guets and hosts review and rate each other. This helps hosts get future bookings and guests get quicker acceptances.

Being an Airbnb host can be very lucrative. According to smartassest.com, if a host lives in a two-bedroom house/apartment and rents out one room in the two-bedroom unit, the host can expect to earn income equivalent to 80% of the rent. And in some cities, it can be possible to pay the entire rent with about 20 days of bookings per month. Listing a room on Airbnb can be better than having a roomate. But unlike having a roomate, being a host comes with work, and some hosts are better than others. The exceptional hosts of Airbnb are identified on the site as "Superhosts". The Superhosts get perks like more visibility, allowing even greater earning potenital by being able to attract more guests.



Because I am an advocate for Airbnb, and an avid user of the platform (12 stays in 2019), I decided to focus my project on Airbnb. As a user, I recognized my own personal behavior when searching for bookings; there are three things I typically look at: location, cost, and host rating. I noticed a trend in my behavior – most of my stays have been with Superhosts. In fact, that is usually something I filter upon when searching for my listings, and I found myself asking "what does it take to become a Superhost?".

According to Airbnb, the minimum criteria to become a Superhost is to have:

- An overall rating greater than 4.8 based on reviews in the past year
- A minimum of 10 stays in the past year
- A cancellation rate of less than 1%
- A 90% response rate
- An account in good standing

Project Aim

Use machine learning to predict if an Austin Airbnb listing is hosted by a Superhost.

The Dataset

The dataset for this project comes from Insideairbnb.com. Inside Airbnb is an anti-lobby group that utilizes and compiles public information from airbnb.com. It compiles all listings and the respective listings' information, by city, to allow for visibility for how Airbnb is really being used around the world.

```
file <- "data/austin_listings_2019_nov12.csv"
df <- read.csv(file, stringsAsFactors = FALSE)
dim(df)
```

```
## [1] 11250   106
```

The Austin dataset was scraped on November 12, 2019. The original dataset has 11250 observations and 106 variables.

Libaries used for analysis:

```
library(plyr)          # data manipulation
library(tidyverse)     # data cleaning
library(data.table)    # data aggregation
library(lubridate)     # modifying date values
library(corrplot)      # correlation plot
library(ggmap)         # geo-spatial visualization
library(ggplot2)       # visualization
library(gridExtra)     # visualization
library(scales)        # visualization
library(RColorBrewer) # visualization
library(tm)            # text mining
library(caret)         # model training and classification
library(DMwR)          # machine learning
library(pROC)          # model evaluation
```

Data Cleansing

- Variable selection
- Adjusting variable classes
- Handling missing numeric and character values
- Adding additional variables

Variable Selection

The first step in my data cleansing process was to narrow down the amount of variables. From looking at the dataset, I was able to quickly identify columns not necessary for my analysis such as fields containing URLs, scrape details, information regarding additional listings, redundant fields, and calendar information. I narrowed down the columns to the ones below:

```
columnsToDelete <- c("url", "listings", "availability", "\\bhost_id\\b",
                     "_name", "scrape", "minimum", "maximum", "require",
                     "\\bneighbourhood\\b","group","experiences_offered",
                     "street", "city", "state","zipcode", "market","location",
                     "country", "latitude", "longitude","acceptance",
                     "host_neighbourhood", "_price", "security_deposit",
                     "square_feet", "guests_included", "extra_people",
                     "calendar", "license", "travel_ready","cleaning_fee",
                     "first_review", "last_review", "reviews_per_month")

# removing uneccessary columns
df <- df[,colnames(df)[colnames(df) %in% unique(grep(paste(columnsToDelete,collaps
        e="|"),
                        colnames(df), value=TRUE)) == FALSE]]
colnames(df)
```

```
##  [1] "id"                          "name"
##  [3] "summary"                     "space"
##  [5] "description"                 "neighborhood_overview"
##  [7] "notes"                       "transit"
##  [9] "access"                      "interaction"
## [11] "house_rules"                 "host_since"
## [13] "host_about"                  "host_response_time"
## [15] "host_response_rate"          "host_is_superhost"
## [17] "host_verifications"          "host_has_profile_pic"
## [19] "host_identity_verified"      "neighbourhood_cleansed"
## [21] "property_type"               "room_type"
## [23] "accommodates"                "bathrooms"
## [25] "bedrooms"                    "beds"
## [27] "bed_type"                    "amenities"
## [29] "price"                       "number_of_reviews"
## [31] "number_of_reviews_ltm"       "review_scores_rating"
## [33] "review_scores_accuracy"      "review_scores_cleanliness"
## [35] "review_scores_checkin"       "review_scores_communication"
## [37] "review_scores_value"         "instant_bookable"
## [39] "cancellation_policy"
```

Predicted class = `host_is_superhost`

Variable Classes

I continued cleaning the fields within the dataset to ensure they were the appropriate class.

```
# checking df
glimpse(df)
```

```
## Observations: 11,250
## Variables: 39
## $ id                            <int> 2265, 5245, 5456, 5769, 6413, 6448, 14913…
## $ name                          <chr> "Zen-East in the Heart of Austin (monthly…
## $ summary                       <chr> "Zen East is situated in a vibrant & dive…
## $ space                         <chr> "This colorful and clean 1923 house was c…
## $ description                   <chr> "Zen East is situated in a vibrant & dive…
## $ neighborhood_overview         <chr> "", "", "My neighborhood is ideally locat…
## $ notes                         <chr> "A 2013 Genuine Buddy Scooter 125 may be …
## $ transit                       <chr> "5 min walk to Capitol Metro Rail (train …
## $ access                        <chr> "Several local restaurants, small clubs, …
## $ interaction                   <chr> "Depending on your dates and arrival time…
## $ house_rules                   <chr> "• Check-in time is 4 pm. Check out is 11…
## $ host_since                    <chr> "2008-08-23", "2008-08-23", "2009-02-16",…
## $ host_about                    <chr> "I am a long time resident of Austin. I e…
## $ host_response_time            <chr> "within a few hours", "within a few hours…
## $ host_response_rate            <chr> "100%", "100%", "100%", "100%", "100%", "…
## $ host_is_superhost             <chr> "t", "t", "t", "t", "t", "t", "f", "t", "…
## $ host_verifications            <chr> "['email', 'phone', 'facebook', 'reviews'…
## $ host_has_profile_pic          <chr> "t", "t", "t", "t", "t", "t", "t", "t", "…
## $ host_identity_verified        <chr> "t", "t", "t", "t", "f", "t", "t", "t", "…
## $ neighbourhood_cleansed        <int> 78702, 78702, 78702, 78729, 78704, 78704,…
## $ property_type                 <chr> "House", "House", "Guesthouse", "House", …
## $ room_type                     <chr> "Entire home/apt", "Private room", "Entir…
## $ accommodates                  <int> 4, 2, 3, 2, 2, 3, 6, 5, 4, 4, 2, 6, 4, 6,…
## $ bathrooms                     <dbl> 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1…
## $ bedrooms                      <int> 2, 1, 1, 1, NA, 1, 3, 2, 2, 2, 1, 2, 2, 3…
## $ beds                          <int> 2, 2, 2, 1, 1, 2, 3, 2, 4, 2, 2, 2, 2, 3,…
## $ bed_type                      <chr> "Real Bed", "Real Bed", "Real Bed", "Real…
## $ amenities                     <chr> "{TV,\"Cable TV\",Internet,Wifi,\"Air con…
## $ price                         <chr> "$225.00", "$100.00", "$95.00", "$40.00",…
## $ number_of_reviews             <int> 24, 9, 508, 257, 104, 220, 28, 90, 35, 9,…
## $ number_of_reviews_ltm         <int> 1, 0, 47, 19, 23, 27, 3, 25, 5, 0, 4, 41,…
## $ review_scores_rating          <int> 93, 91, 96, 98, 99, 99, 99, 98, 97, 98, 9…
## $ review_scores_accuracy        <int> 9, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10…
## $ review_scores_cleanliness     <int> 10, 8, 10, 10, 10, 10, 10, 10, 10, 9, 10,…
## $ review_scores_checkin         <int> 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 1…
## $ review_scores_communication   <int> 10, 9, 10, 10, 10, 10, 10, 10, 10, 10, 10…
## $ review_scores_value           <int> 9, 9, 10, 10, 10, 10, 10, 10, 10, 10, 10,…
## $ instant_bookable              <chr> "f", "f", "f", "f", "t", "t", "f", "t", "…
## $ cancellation_policy           <chr> "strict_14_with_grace_period", "strict_14…
```

```
# prior to formatting, setting all blanks to NA
df[df == ""] <- NA

# creating dummy variables
df[df == "t"] <- 1
df[df == "f"] <- 0

# moving class variable to first column in df and creating factor
df <- df %>%
  select(host_is_superhost, everything()) %>%
  filter(!is.na(host_is_superhost)) %>% # removing nas values for class variable
  mutate(host_is_superhost = as.factor(as.character(host_is_superhost)))

# removing special characters in columns
df[c("price", "host_response_rate")] <- sapply(df[c("price", "host_response_rate"
        )],
       function(x) as.numeric(gsub("\\$|%", "", x)))

# class adjustments
df$id <- as.character(df$id)
df$host_since <- as.Date(df$host_since)
df <- df %>%
  dplyr::rename(zipcode = neighbourhood_cleansed) %>%   # renaming zipcode column
  mutate(zipcode = as.character(zipcode))
```

Handling Missing Values

The data is messy and has many missing values that need correction. I noticed that the character variable `host_response_time` has many missing values, so I will be adjusting this variable as well.

```
# looking for NA's in numeric columns
na_count <- colnames(df)[colSums(is.na(df)) > 0]
sapply(df[,na_count], function(x) sum(length(which(is.na(x)))))
```

```
##                     summary                       space
##                         406                        2645
##                 description       neighborhood_overview
##                         180                        3712
##                       notes                     transit
##                        5969                        4232
##                      access                 interaction
##                        4552                        3766
##                 house_rules                  host_about
##                        3623                        3929
##          host_response_rate                   bathrooms
##                        3308                          15
##                    bedrooms                        beds
##                           5                          12
##                       price        review_scores_rating
##                         668                        2322
##      review_scores_accuracy   review_scores_cleanliness
##                        2327                        2327
##       review_scores_checkin review_scores_communication
##                        2328                        2328
##         review_scores_value
##                        2332
```

```
# looking for NA's in host_response_time
sum(grepl("N/A", df$host_response_time))
```

```
## [1] 3308
```

To correct some of the NA's, I replaced the numeric columns with the median value of their column. For the character column, I created dummy values and found the mean based on the `host_response_rate` column.

```
# replacing columns with median values
numericCols <- colnames(select_if(df, is_numeric))
numericCols <- numericCols[-grep("host_since|host_is_superhost", numericCols)]   #
        removing host_since & class variable
df[, numericCols] <- sapply(df[, numericCols],
                            function(x) ifelse(is.numeric(x) & is.na(x),
                                               round(median(x, na.rm = TRUE)), x))

# checking for no NA's in numeric columns
colSums(is.na(df[, numericCols]))
```

```
##          host_response_rate                accommodates
##                           0                           0
##                   bathrooms                    bedrooms
##                           0                           0
##                        beds                       price
##                           0                           0
##           number_of_reviews       number_of_reviews_ltm
##                           0                           0
##        review_scores_rating      review_scores_accuracy
##                           0                           0
##   review_scores_cleanliness       review_scores_checkin
##                           0                           0
## review_scores_communication         review_scores_value
##                           0                           0
```

```
# replacing host_response_time with median values
# first creating a dataframe to match upon
dummy <- as.data.frame(unique(df$host_response_time[df$host_response_time != "N/A"
        ]))
colnames(dummy) <- c("host_response_time.x")
dummy$dummy <- c(2,1,3,4) # setting levels
print(dummy)
```

```
##   host_response_time.x dummy
## 1   within a few hours     2
## 2       within an hour     1
## 3         within a day     3
## 4   a few days or more     4
```

```
# modifying original df
df <- df %>%
  # adding dummy variable
  left_join(dummy, by = c("host_response_time" = "host_response_time.x")) %>%
  # creating groups of host_response_rate using cut
  mutate(host_response_rate_cut = cut(host_response_rate, 10, include.lowest=TRUE
        ))

# check
df %>%
  select(host_response_time, host_response_rate, host_response_rate_cut) %>%
  head(10)
```

```
##    host_response_time host_response_rate host_response_rate_cut
## 1  within a few hours                100               (90,100]
## 2  within a few hours                100               (90,100]
## 3  within a few hours                100               (90,100]
## 4      within an hour                100               (90,100]
## 5      within an hour                100               (90,100]
## 6      within an hour                100               (90,100]
## 7      within an hour                100               (90,100]
## 8      within an hour                 97               (90,100]
## 9  within a few hours                100               (90,100]
## 10     within an hour                100               (90,100]
```

```
# replacing missing value with mean value
df <- df %>%
  group_by(host_response_rate_cut) %>%
  # getting mean values
  summarise(dummy = floor(mean(dummy, na.rm = TRUE))) %>%
  left_join(dummy) %>%
  right_join(df, by = c("host_response_rate_cut")) %>%
  # replacing NAs with mean values
  mutate(host_response_time = ifelse(host_response_time == "N/A",
                                     as.character(host_response_time.x),
                                     host_response_time)) %>%
  select(-host_response_rate_cut, -dummy.x, -dummy.y, -host_response_time.x)

# checking for no NA's in text column
sum(grepl("N/A", df$host_response_time))
```

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

Adding Additional Variables

Each listing has a list of amenities that the host provides. My hunch is that a Superhost would have more amenities, so I decided to get a count of amenities each listing provides. The column is messy, so first I cleaned the column and then generated a new column `amenities_count` .

```
# current format
df$amenities[1]
```

```
## [1] "{TV,\"Cable TV\",Internet,Wifi,\"Air conditioning\",Kitchen,\"Free parking
on premises\",\"Paid parking off premises\",Breakfast,\"Pets live on this property
\",Dog(s),\"Free street parking\",Heating,\"Family/kid friendly\",Washer,Dryer,\"S
moke detector\",\"Carbon monoxide detector\",Essentials,Shampoo,Hangers,\"Hair dry
er\",Iron,\"Laptop friendly workspace\",\"Self check-in\",Lockbox,\"Private entran
ce\",\"Hot water\",\"Bed linens\",\"Extra pillows and blankets\",Microwave,\"Coffe
e maker\",Refrigerator,\"Dishes and silverware\",\"Cooking basics\",Oven,Stove,\"G
arden or backyard\"}"
```

```
# re-formating amenities column
df$amenities <- gsub("(^|[[:space:]])([[:alpha:]])", "\\1\\U\\2", df$amenities, p
        erl=TRUE)
df$amenities <- gsub("[^a-zA-Z]+", " ",
                     gsub(",|/", "",
                          gsub(" ", "", gsub("\\-|\\(|\\)", "", df$amenities))))
df$amenities <- trimws(df$amenities)
df$amenities[1]
```

```
## [1] "TV CableTV InternetWifi AirConditioning Kitchen FreeParkingOnPremises Paid
ParkingOffPremises Breakfast PetsLiveOnThisProperty Dogs FreeStreetParking Heating
FamilykidFriendly WasherDryer SmokeDetector CarbonMonoxideDetector EssentialsShamp
ooHangers HairDryer Iron LaptopFriendlyWorkspace SelfCheckin Lockbox PrivateEntran
ce HotWater BedLinens ExtraPillowsAndBlankets Microwave CoffeeMaker Refrigerator D
ishesAndSilverware CookingBasics OvenStove GardenOrBackyard"
```

```
# counting words (amenities) in amenities column
df$amenities_count <- lengths(gregexpr("\\W+", df$amenities)) + 1

# re-formatted and counted
df$amenities_count[1]
```

```
## [1] 33
```

Later, I will do some breif text mining to see what exactly Superhosts are including in their details and amenities that Non-Superhosts are not.

Next, I added a column `years_host` to determine a numeric value of years the host has had listings on Airbnb.

```
# using filename, extracting file scrape year
scrape_year <- str_extract(file, "\\d{4}")
print(scrape_year)
```

```
## [1] "2019"
```

```
# subtracting host_since year from scrape_year to get # years host
df$years_host <- as.numeric(scrape_year) - as.numeric(year(df$host_since))

# check
df %>%
  select(host_since, years_host) %>%
  head(5)
```

```
## # A tibble: 5 x 2
##   host_since years_host
##   <date>          <dbl>
## 1 2008-08-23         11
## 2 2008-08-23         11
## 3 2009-02-16         10
## 4 2009-02-19         10
## 5 2009-04-17         10
```

Exploratory Data Analysis

- Predict class value overview
- Correlations between variables
- Distribution of features by Superhost status
- Geo-spatial data analysis
- Brief text mining of listings

Predicted Class Values

For simplicity during the EDA, I renamed the predicted class variable, `host_is_superhost` , values to “non.superhost” and “superhost”.

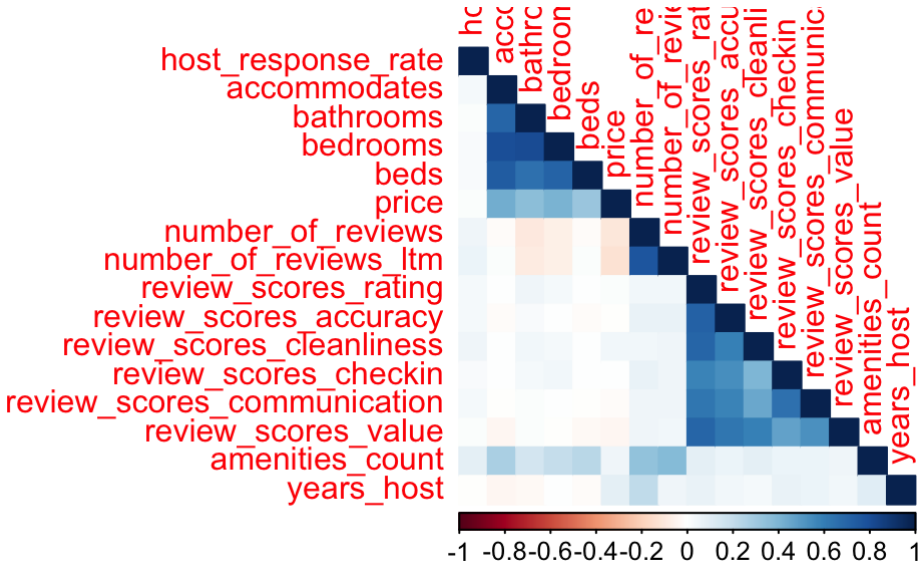
```
# changing factor levels 0 and 1
levels(df$host_is_superhost) <- c("non.superhost", "superhost")
print(levels(df$host_is_superhost))
```

```
## [1] "non.superhost" "superhost"
```

Correlation Variables

To kick off my exploratory data analysis (EDA), first, I checked the correlations between the variables.

```
# getting numeric columns
numericCols <- colnames(select_if(df, is_numeric))
numericCols <- numericCols[-grep("host_since|host_is_superhost", numericCols)]   #
        removing host_since

# correlation plot
corr <- round(cor(df[,numericCols]), 2)
corrplot(corr, method = "color", type = "lower")
```



At a glance, there is a lot of correlation between the review_scores, which is expected. I then checked for all variables that have correlations higher than 0.5 or less than -0.5. This confirmed the majority of variables with high correlations are the review_scores columns. Additionally, the number of guests a listing accomodates and the number of bedrooms and bathrooms also has a high correlation.

```
# checking variables with correlations > 0.5 or < -0.5
setDT(melt(corr))[Var1 != Var2, .SD[which(value > .5 | value < -.5)], keyby=Var1]
        %>%
  arrange(-value) %>%
  mutate(delete = ifelse(lag(Var2) == Var1, TRUE, FALSE)) %>%
  filter(delete == TRUE) %>%
  select(-delete)
```

```
##                            Var1                        Var2 value
## 1                      bedrooms                   bathrooms  0.82
## 2                      bedrooms                accommodates  0.80
## 3         number_of_reviews_ltm           number_of_reviews  0.77
## 4                          beds                accommodates  0.74
## 5        review_scores_accuracy        review_scores_rating  0.72
## 6                          beds                    bedrooms  0.71
## 7                     bathrooms                accommodates  0.70
## 8                          beds                   bathrooms  0.65
## 9  review_scores_communication       review_scores_checkin  0.65
## 10       review_scores_accuracy   review_scores_cleanliness  0.59
## 11   review_scores_cleanliness      review_scores_accuracy  0.59
## 12          review_scores_value   review_scores_cleanliness  0.59
## 13       review_scores_accuracy       review_scores_checkin  0.54
## 14        review_scores_checkin      review_scores_accuracy  0.54
## 15          review_scores_value review_scores_communication  0.53
```

For my analysis, I removed all review_scores variables except for the overall rating (`review_scores_rating`). Additionally, I chose to keep the `accomodates` column and remove `bedrooms` , `bathrooms` , and `beds` . I felt the number of guests a listing can accomodate impacts the number of bathrooms, bedrooms, and beds available.

```
# removing correlated review_scores columns
df <- df[,-grep("review_scores", colnames(df))[-1]]

# removing correlated accomodates, bathrooms, & beds columns
columnsToDelete <- c("bedrooms", "bathrooms", "beds")
df <- df[,colnames(df)[colnames(df) %in%
                         unique(grep(paste(columnsToDelete,collapse="|"),
                        colnames(df), value=TRUE)) == FALSE]]
```

Distribution of Features by host_is_superhost Group

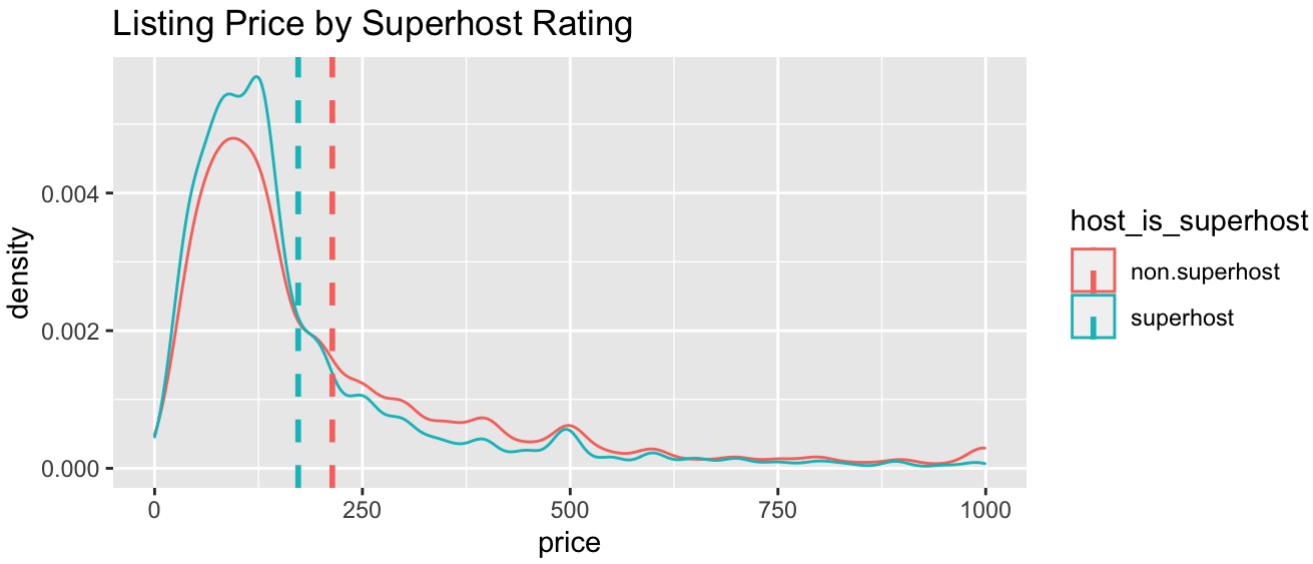
Listing Price

When thinking of staying with an incredible host, you'd assume to pay for quality. I decided to look into this thought by examining the distributions of nightly prices of Superhosts and Non-Superhosts, and the results is opposite of what I expected.

```
# mean for each group
dat <- ddply(df, "host_is_superhost", summarise, mean.price = round(mean(price), 2
        ), median.price = round(median(price), 2))
print(dat)
```

```
##   host_is_superhost mean.price median.price
## 1     non.superhost     213.75          129
## 2         superhost     172.72          125
```

```
# density plots with means
ggplot(df, aes(x = price, colour=host_is_superhost)) +
  geom_density() +
  geom_vline(data = dat, aes(xintercept = mean.price, colour=host_is_superhost),
             linetype="dashed", size=1) +
  ggtitle("Listing Price by Superhost Rating")
```



On average, Non-Superhosts have a higher mean nightly rate than Superhosts (213.75 vs 172.72), but the median of the two are around the same price range. Non-Superhosts have more outliers, a longer right tail, than Superhosts. From my research, this can be the results of the many popular events in Austin such as SXSW and Austin City Limits. Austin residents cash in on the travelers flocking to the city for these events and post their property on Airbnb for an outrageous price. These residents are not dedicated hosts and are probably not seeking a Superhost identification, but rather quick cash.

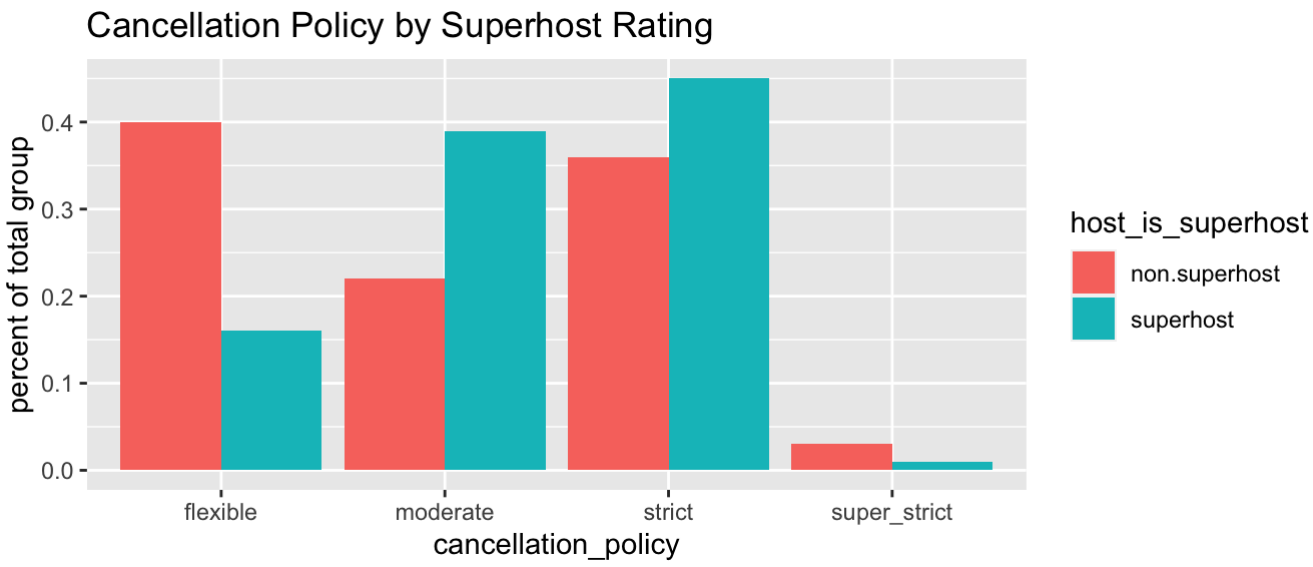
Cancellation Policy

Another interesting trend I found amongst the Superhosts and Non-Superhosts was the cancellation rates. My original thought was that Superhosts would be flexible with cancellations, because if you have to cancel, they are more likely to get another reservation to replace the loss. The data showed me different.

According to Airbnb, the cancellation policies fall into:

- Flexible - Free cancellation until 24 hours before check-in
- Moderate - Free cancellation until 5 days before check-in
- Strict - Free cancellation for 48 hours, as long as the guest cancels at least 14 days before check-in
- Super Strict - Guests can cancel at least 30 or 60 days before check-in and get a 50% refund of the nightly rate and the cleaning fee, but not the service fee

```
# cleaning cancellation policies
df$cancellation_policy[grep("super_strict", df$cancellation_policy)] <- "super_str
        ict"
df$cancellation_policy[grep("^strict", df$cancellation_policy)] <- "strict"
df$cancellation_policy[grep("moderate", df$cancellation_policy)] <- "moderate"

# bar plots with % distribution
df %>%
  group_by(host_is_superhost, cancellation_policy) %>%
  dplyr::count() %>%
  group_by(host_is_superhost) %>%
  mutate(pct_ttl = round(n/sum(n),2)) %>%
  ggplot(aes(x = cancellation_policy, y = pct_ttl, fill = host_is_superhost)) +
  xlab(c("cancellation_policy")) +
  ylab(c("percent of total group")) +
  geom_bar(stat = "identity", position = position_dodge()) +
  ggtitle("Cancellation Policy by Superhost Rating")
```



Superhosts are much less flexible than Non-Superhosts. In fact, a greater proportion of them are more strict. After thinking about this, it makes sense – Superhosts are hosts who are dedicated to Airbnb. From my personal experience with staying with Superhosts, a lot of them use Airbnb as a primary source of income. They are committed to serving you and have the expectation you will be committed to making your stay.

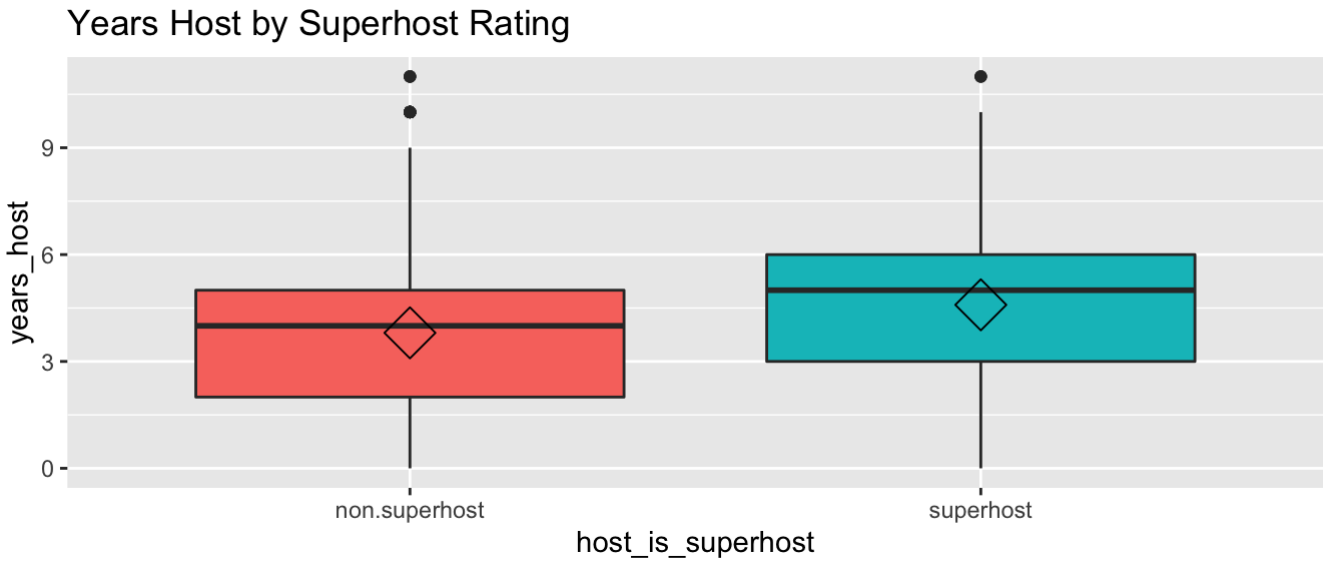
Years Hosting on Airbnb

Another hunch is that Superhosts will have more years of experience hosting on Airbnb. I plotted the `years_host` to see if the data holds true.

```
# mean for each group
dat <- ddply(df, "host_is_superhost", summarise, mean.years = round(mean(years_hos
        t), 1), median.years = round(median(years_host), 1))
print(dat)
```

```
##   host_is_superhost mean.years median.years
## 1     non.superhost        3.8            4
## 2         superhost        4.6            5
```

```
# box plots with means
ggplot(df, aes(x = host_is_superhost, y = years_host, fill = host_is_superhost)) +
  geom_boxplot() +
  guides(fill=FALSE) +
  stat_summary(fun = mean, geom="point", shape=5, size=6) +
  ggtitle("Years Host by Superhost Rating")
```



As suspected, Superhosts in Austin, on average, have been hosts for 4.6 years and Non-Superhosts have been hosts for 3.8 on average.

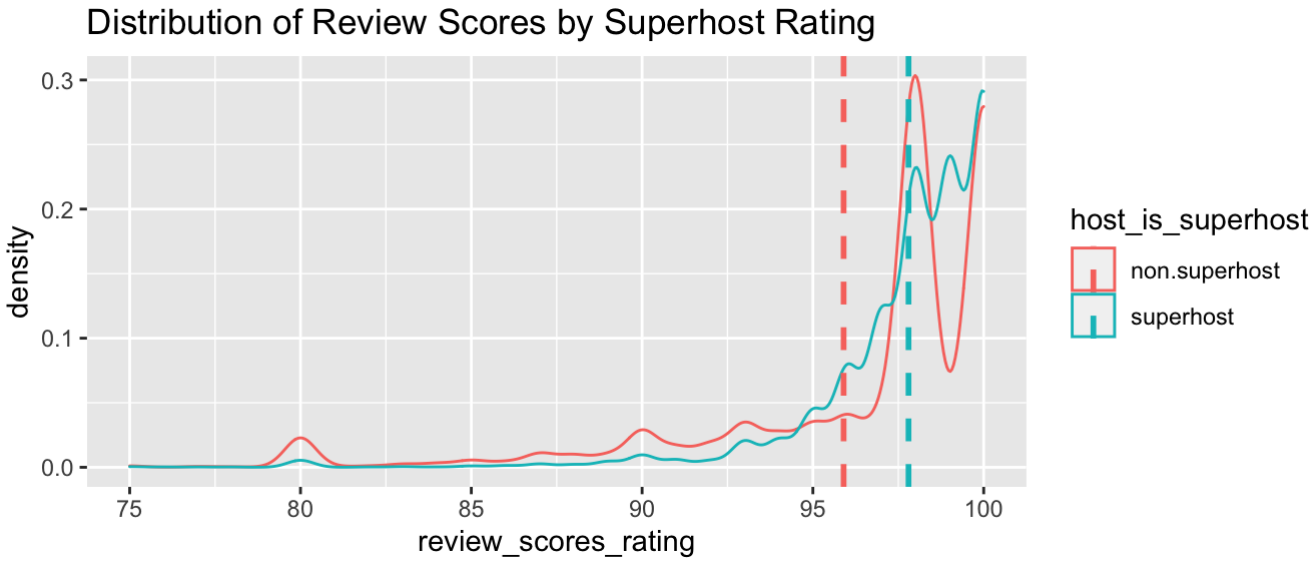
Review Scores

For a host to be a Superhost, there is the expectation that they will have better ratings. I plotted Superhosts and Non-Superhosts density plots to check if this holds up.

```
# mean for each group
dat <- ddply(df, "host_is_superhost", summarise, mean.rating = round(mean(review_s
        cores_rating), 1), median.rating = round(median(review_scores_rating), 1
        ))
print(dat)
```

```
##   host_is_superhost mean.rating median.rating
## 1     non.superhost        95.9            98
## 2         superhost        97.8            98
```

```
# density plots with means
ggplot(df, aes(x = review_scores_rating, colour=host_is_superhost)) +
  geom_density() +
  geom_vline(data = dat, aes(xintercept = mean.rating, colour=host_is_superhost),
             linetype="dashed", size=1) +
  ggtitle("Distribution of Review Scores by Superhost Rating") +
  xlim(75,100)
```



Non-Superhosts have a longer left tail, meaning more negative ratings than Superhosts. Most of the Superhost ratings hover around 95 and higher, peaking at 100. Non-Superhosts peak at around 97. Both Superhosts and Non-Superhosts have a median rating of 98, but Superhosts mean rating score of 97.8 is higher than Non-Superhost's mean rating score of 95.9.

Mapping of Listings

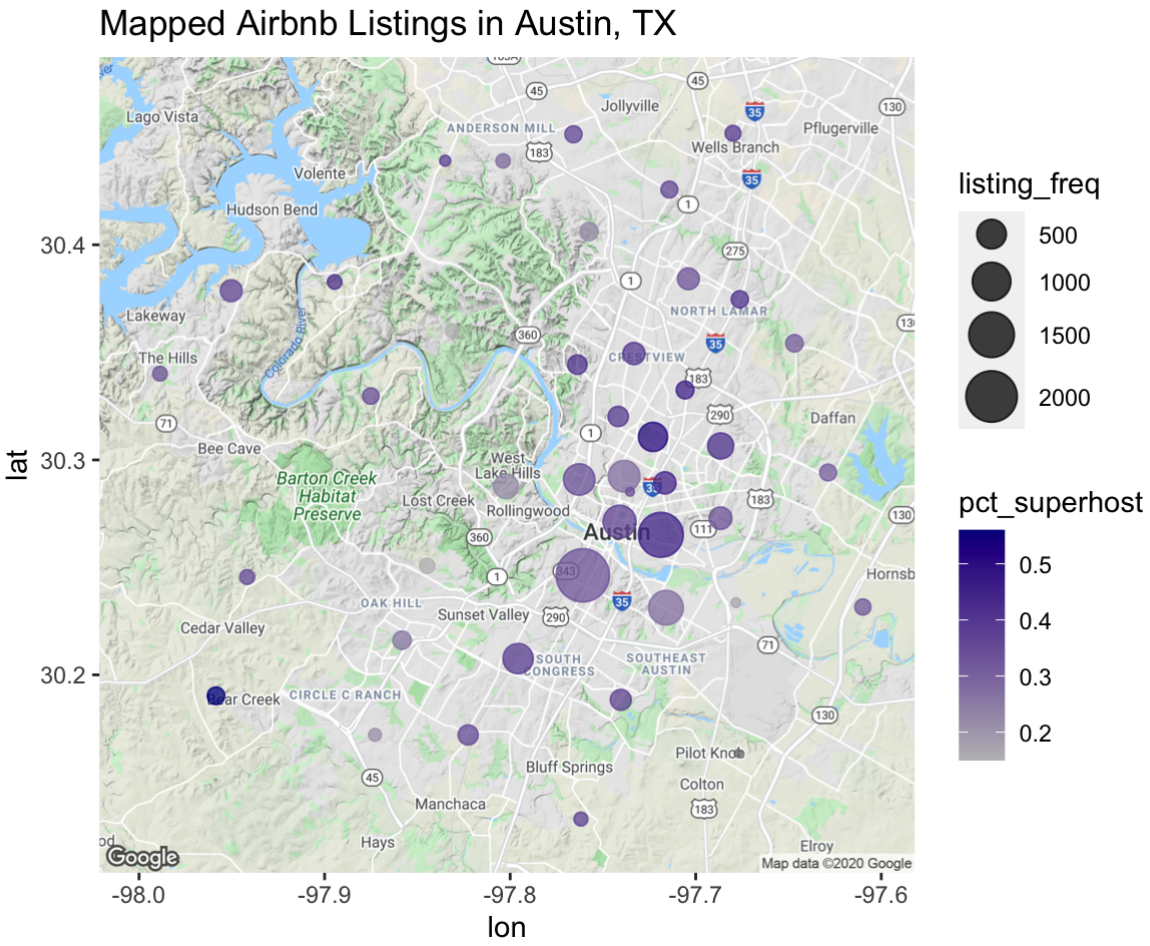
We have gotten a better understanding of what sets a Superhost apart from the rest, but does location have a role in whether a host is likely to be a Superhost or not?

With the help of Google Map's API and the Zipcode library, I was able to plot Superhosts by zipcode on a map. The size of the circles `listing_freq` is the amount of listings that live within that zipcode. The color `pct_superhost` is the relative percent of listings that are ran by Superhosts in that zipcode.

```
# loading zipcode library
library(zipcode)
data(zipcode)

# frequencies of listings and superhosts by zipcodes
zip_df <- left_join(df[,c("host_is_superhost", "zipcode")], zipcode, by = c("zipc
        ode" = "zip"))
listing_freq <- ddply(zip_df, .(zipcode), "nrow")
names(listing_freq)[2] <- "listing_freq"

superhost_freq <- ddply(zip_df, .(zipcode, host_is_superhost), "nrow") %>%
  filter(host_is_superhost == "superhost") %>%
  select(-host_is_superhost)
names(superhost_freq)[2] <- "superhost_freq"

# creating zipcode df
zip_df <- zip_df %>%
  select(-host_is_superhost) %>%
  distinct() %>%
  left_join(listing_freq) %>%
  left_join(superhost_freq) %>%
  arrange(-listing_freq) %>%
  mutate(pct_superhost = round(superhost_freq/listing_freq,2)) # calculating % of
        superhosts per total listings in zipcode
print(head(zip_df))
```

```
## # A tibble: 6 x 8
##   zipcode city  state latitude longitude listing_freq superhost_freq
##   <chr>   <chr> <chr>    <dbl>     <dbl>        <int>          <int>
## 1 78704   Aust… TX        30.2     -97.8         2179            714
## 2 78702   Aust… TX        30.3     -97.7         1449            647
## 3 78741   Aust… TX        30.2     -97.7          759            207
## 4 78701   Aust… TX        30.3     -97.7          669            249
## 5 78705   Aust… TX        30.3     -97.7          599            163
## 6 78703   Aust… TX        30.3     -97.8          586            200
## # … with 1 more variable: pct_superhost <dbl>
```

```
# plotting map of Austin, TX
location <- geocode("West Lake Hills, Austin, TX")
ggmap(get_map(location, zoom = 11), ylab = "Latitude", xlab = "Longitude") +
  geom_point(aes(x=longitude, y=latitude, size=listing_freq, color=pct_superhost),
             alpha = .75, data=zip_df) +
  scale_size_continuous(range = c(1,9)) +
  scale_color_gradient(high = "dark blue", low = "grey") +
  ggtitle("Mapped Airbnb Listings in Austin, TX")
```



The zipcode with the highest amount of listings is 78704, which is the Bouldin/South Lamar neighbhoorhood. According to austinrelocationguide.com, this zipcode region is the epitome of Austin living, so it makes sense why so many property listings are in this high-demand area. Although 78704 has the highest amount of listings, it does not have the highest percentage of Superhosts. The top zipcode with the highest percentage of Superhosts is 78737 with 56% of listings being Superhosts. This area is a rural area outside of Austin.

Brief Text Mining

Each host has the opportunity to provide details of their listing showcased in the free-form columns. The listing details are a good opportunity for text mining. Although I will not be going into a deep text-mining analysis, I thought it would be interesting to explore these fields briefly.

```
# getting text columns
txtCols <- colnames(select_if(df, is_character))
txtCols <- txtCols[c(2:12,21)]
print(txtCols)
```

```
##  [1] "name"                "summary"               "space"
##  [4] "description"         "neighborhood_overview" "notes"
##  [7] "transit"             "access"                "interaction"
## [10] "house_rules"         "host_about"            "amenities"
```

Next, I wrote a function for counting the frequencies of words in each of the free-form text columns. I then wrote a function for calculating the delta in word frequency counts between Superhost and Non-Superhosts. Additionally, I added in a function for plotting the deltas.

```
# function for counting words in text columns
wordcounts <- function(df, txt_column){
  # removing NAs
  tmp <- df %>%
    select(txt_column) %>%
    filter(!is.na(txt_column))
  # creating corpus
  corp <- SimpleCorpus(VectorSource(tmp))
  corp <- tm_map(corp, content_transformer(tolower))
  # cleaning words
  corp <- tm_map(corp, content_transformer(function(x) gsub("'ll|'ve", "", x)))
  corp <- tm_map(corp, content_transformer(function(x)
        gsub("minutes|\\bmi(ns|n)\\b", "minute", x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("guests", "guest",
         x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("bedrooms", "bedroom",
        x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("beds", "bed", x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("mattresses", "mattres
        s", x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("bathrooms", "bathroo
        m", x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("^rooms", "room",
        x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("homes", "home", x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("neighborhoods", "neig
        hborhood", x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("austins", "austin",
         x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("\\bbar\\b", "bars",
         x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("\\bstore\\b", "store
        s", x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("\\bshop\\b", "shops"
        , x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("\\btrail\\b", "trail
        s", x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("miles", "mile", x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("walking", "walk",
         x)))
  corp <- tm_map(corp, content_transformer(function(x) gsub("nearby", "near",
        x)))
  corp <- tm_map(corp, removePunctuation)
  corp <- tm_map(corp, removeWords, c(stopwords("english"),
                                      "just","away","can","also","within",
                                      "will","like","need","use","lots",
                                      "many","get","well","feel","make","find",
                                      "one","two","three","four","five"))
  dtm <- DocumentTermMatrix(corp)
  # getting sums of word frequencies
  sums <- as.data.frame(colSums(as.matrix(dtm)))
  sums <- rownames_to_column(sums)
```

```
  colnames(sums) <- c("term", "count")
  sums <- sums %>%
    arrange(desc(count)) %>%
    mutate(pct = count / sum(count))
  invisible(sums)
}

# function for calculating delta between superhosts and non-superhosts
ratings_counts <- function(df, txt_column){
  # splitting to 2 separate df based on class
  tmp_1 <- df %>%
    filter(host_is_superhost == "superhost")
  tmp_0 <- df %>%
    filter(host_is_superhost == "non.superhost")
  # applying wordcounts function
  tmp_1 <- wordcounts(tmp_1, txt_column)
  tmp_0 <- wordcounts(tmp_0, txt_column)
  # joining df to find differences in word frequencies
  tmp <- full_join(tmp_1, tmp_0, by = c("term")) %>%
    mutate(delta = pct.x - pct.y) %>%
    filter(pct.x > .002 | pct.y > .002) %>%
    arrange(-delta)
  colnames(tmp) <- c("term", "freq_superhosts", "proportion_superhosts",
                     "freq_nonsuperhosts", "proportion_nonsuperhosts", "delta")
  invisible(tmp)
}

# fuction for plotting delta between superhosts and non-superhosts
delta_plot <- function(df, name){
  ggplot(df, aes(proportion_superhosts, proportion_nonsuperhosts)) +
  geom_point(alpha = .1, size = 2.5, width = 0.25, height = 0.25) +
  geom_text(aes(label = term), check_overlap = TRUE, vjust = 1) +
  ylab("Non-Superhosts") +
  xlab("Superhosts") +
  scale_x_continuous(labels = percent_format()) +
  scale_y_continuous(labels = percent_format()) +
  geom_abline(color = "red") +
  ggtitle(name)
}
```

Once my functions were established, I decided to look at 4 free-form text columns `name`, `description`, `neighborhood_overview`, and `amenities` to determine the deltas between word frequencies for Superhosts and Non-Superhosts.

```
# finding deltas between most frequent words used in superhosts versus non-superho
        sts listings
name <- ratings_counts(df, "name")
description <- ratings_counts(df, "description")
neighborhood_overview <- ratings_counts(df, "neighborhood_overview")
amenities <- ratings_counts(df, "amenities")

# plotting deltas
p_name <- delta_plot(name, "Listing Name")
p_description <- delta_plot(description, "Description of Listing")
p_neighborhood <- delta_plot(neighborhood_overview, "Neighborhood Overview")
p_amenities <- delta_plot(amenities, "Listing Amenities")

grid.arrange(p_name, p_description, p_neighborhood, p_amenities, ncol = 2)
```

How to Read the Plot

Words near the line are used at about the same frequency between both Superhosts and Non-Superhosts. The further away from the line, the more likely the word is to be used by one group versus the other. The above words are words used in both Superhosts and Non-Superhosts listings.

Listing Name

Both Superhosts and Non-Superhosts are likely to use the terms “condo”, “room”, “downtown”, and “austin” in their listing name, with “austin” being the highest frequency. Superhosts are more likely to use words that showcase the lifestyle of their listing, such as “modern” and “retreat”. Non-Superhosts commonly use “sxsw” and “acl”" in their listing name. This validates my theory that Non-Superhosts are more likely to throw up a listing for big events like South By Southwest (SXSW) and Austin City Limits (ACL).

Description of Listing

Words such as “downtown”, “minute”, “walk”, “apartment”, and “sxsw” are more frequently mentioned in Non-Superhosts’ listings, meanwhile, words such as “private”, “home”, “guest”, and “coffee”, are more frequently mentioned in Superhosts’ listings. Based on these top words, it appears that Non-Superhosts have listings focused on a more party-like experience and appears to have heavy focus on being downtown and walkable. Superhosts focus more on a peaceful getaway, luxuries, and amenities.

Neighborhood Overview

Both Superhosts and Non-Superhosts frequently mention “bars” and “restaurants”, but “restaurants” is mentioned slightly more for Non-Superhosts. Non-Superhosts once again describe “downtown” and other popular areas such as “south” (South Congress) and “barton” (Barton Springs). Superhosts mention “historic” and “east” (East Austin) frequently.

Amenities

Amenities provided by Superhosts and Non-Superhosts are very different. Some amenities Superhosts typically provide, but Non-Superhosts do not, are a coffee maker, extra pillows and blankets, kitchen items/access, and luggage dropoff. Both provide first aid kits, locks on the bedroom doors, and irons at about the same rate.

Predicting if Listing is Hosted by a Superhost

The machine learning model I chose to use to predict `host_is_superhost` is a random forest model. I chose this model because it is a good feature selector, can handle binary, categorical, and numerical values, is robust to outliers, and has a low bias relative to other models.

Prior to beginning the machine learning portion, I trimmed down the dataframe to include only the necessary columns.

```
columnsToDelete <- c(txtCols, "id", "host_verifications", "host_since")
dat <- df[,colnames(df)[colnames(df) %in% unique(grep(paste(columnsToDelete,colla
        pse="|"),
                        colnames(df), value=TRUE)) == FALSE]]
print(colnames(dat))
```

```
##  [1] "host_is_superhost"     "host_response_time"    "host_response_rate"
##  [4] "host_has_profile_pic"  "zipcode"               "property_type"
##  [7] "room_type"             "accommodates"          "bed_type"
## [10] "price"                 "number_of_reviews"     "number_of_reviews_ltm"
## [13] "review_scores_rating"  "instant_bookable"      "cancellation_policy"
## [16] "years_host"
```

The dataframe used for the machine learning portion now has 15 variables.

Class Balance

Prior to beginning to class balance check, I renamed the `host_is_superhost` column to `class` and converted the class variable into a factor. Then I split the data frame into training and testing sets with a 80/20 split.

```
# renaming host_is_superhost to class
colnames(dat)[1] <- "class"

# converting character columns to factor
dat[,colnames(select_if(dat, is_character))] <- lapply(dat[,colnames(select_if(da
        t, is_character))], as.factor)

# reordering factors
dat$class <- factor(dat$class, levels(dat$class)[c(2,1)])
levels(dat$class)
```

```
## [1] "superhost"     "non.superhost"
```

```
# partitioning datasets into training and testing sets on a 80/20 split
set.seed(123)
intrain <- createDataPartition(y = dat$class, p = 0.8)[[1]]
training <- dat[intrain,]
testing <- dat[-intrain,]

# viewing class balances in each set
table(training$class)
```

```
##
##     superhost non.superhost
##          3220          5778
```

```
table(testing$class)
```

```
##
##     superhost non.superhost
##           805          1444
```

Sampling Methods

Although a random forest can handle class imbalances, I used down-sampling and up-sampling techniques to see if these techniques improved my model in any way.

Down-sampling randomly samples a data set on the majority class and reduces the number of observations so that all classes have the same frequency as the minority class.

Up-Sampling samples with replacement on the minority class to make the class distributions equal. An advantage of the up-sampling technique is there is no data loss involved.

```
# creating down and up sampling training sets
training_down <- downSample(x = dplyr::select(training,-class), training$class, yn
        ame="class")
training_up <- upSample(x = dplyr::select(training,-class), training$class, yname=
        "class")

# viewing class balances in each set
table(training_down$class)
```

```
##
##     superhost non.superhost
##          3220          3220
```

```
table(training_up$class)
```

```
##
##     superhost non.superhost
##          5778          5778
```

Both sets look balanced!

Training Random Forest Models

Next, I created 3 random forest models for each training set using the area under the curve as the metric of measure and ntree = 200.

```
# setting up function that allows extraction of five metrics instead of just accur
        acy
fiveStats <- function(...)c(twoClassSummary(...), defaultSummary(...))

# model parameters
gridRF <- expand.grid(mtry = seq(from = 1, by = 1, to = ncol(training)-1))
ctrl.crossRF <- trainControl(method = "cv", number = 5,
                             classProbs = TRUE, summaryFunction = fiveStats,
                             savePredictions = TRUE)

# rf models
rf_original <- train(class ~ ., data = training, method = "rf", metric = "ROC",
                     preProc = c("center", "scale"),
                     ntree = 200, tuneGrid = gridRF,
                     trControl = ctrl.crossRF)
rf_down <- train(class ~ ., data = training_down, method = "rf", metric = "ROC",
                     preProc = c("center", "scale"),
                     ntree = 200, tuneGrid = gridRF,
                     trControl = ctrl.crossRF)
rf_up <- train(class ~ ., data = training_up, method = "rf", metric = "ROC",
                     preProc = c("center", "scale"),
                     ntree = 200, tuneGrid = gridRF,
                     trControl = ctrl.crossRF)
```

Performance of Random Forest Models

For ease of analysis, I created a list of all the random forest models and utilized `map` to write functions for the list.

```
# creating list of all models ran
model_list <- list(original = rf_original,
                   down = rf_down,
                   up = rf_up)
```

Confusion Matrices of Training and Testing Data

```
# model performance
model_list %>%
  map(function(x) x$finalModel)
```

```
## $original
##
## Call:
##  randomForest(x = x, y = y, ntree = 200, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 15
##
##         OOB estimate of  error rate: 13.86%
## Confusion matrix:
##               superhost non.superhost class.error
## superhost          2416           804  0.24968944
## non.superhost       443          5335  0.07667013
##
## $down
##
## Call:
##  randomForest(x = x, y = y, ntree = 200, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 15
##
##         OOB estimate of  error rate: 15.71%
## Confusion matrix:
##               superhost non.superhost class.error
## superhost          2666           554   0.1720497
## non.superhost       458          2762   0.1422360
##
## $up
##
## Call:
##  randomForest(x = x, y = y, ntree = 200, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 15
##
##         OOB estimate of  error rate: 8.02%
## Confusion matrix:
##               superhost non.superhost class.error
## superhost          5436           342  0.05919003
## non.superhost       585          5193  0.10124611
```

When evaluating the 3 models, `rf_original` , `rf_down` , and `rf_up` had a mtry of 15. The class error is different between `rf_original` and the sampled models, the up-sample technique `rf_up` had the best adjustment.

```
# confusion matrices
model_list %>%
  map(function(x) confusionMatrix(predict(x, testing), testing$class)[2])
```

```
## $original
## $original$table
##                Reference
## Prediction      superhost non.superhost
##   superhost           605           102
##   non.superhost       200          1342
##
##
## $down
## $down$table
##                Reference
## Prediction      superhost non.superhost
##   superhost           655           190
##   non.superhost       150          1254
##
##
## $up
## $up$table
##                Reference
## Prediction      superhost non.superhost
##   superhost           632           136
##   non.superhost       173          1308
```

```
model_list %>%
  map(function(x) confusionMatrix(predict(x, testing), testing$class)$overall[c(1
        )])
```

```
## $original
##  Accuracy
## 0.8661627
##
## $down
##  Accuracy
## 0.8488217
##
## $up
##  Accuracy
## 0.8643842
```

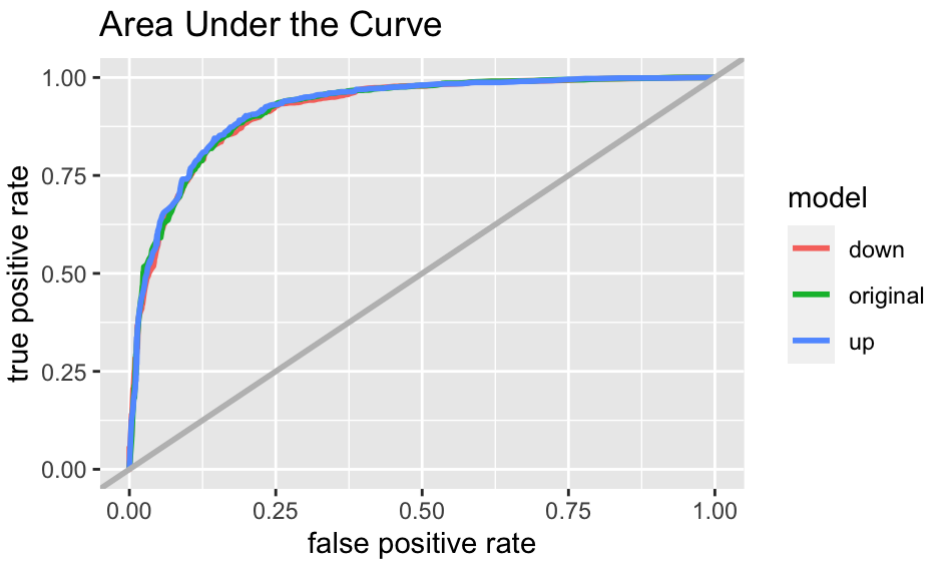
Surprsingly, the highest accuracy on the testing sets is the `rf_original` model – the `rf_up` model accuracy is slightly lower. The lowest accuracy is the `rf_down` model. Next, I looked at area under the curve (AUC) to help determine the best model.

Area Under the Curve

```
# area under the curve
model_list_roc <- model_list %>%
  map(function(x) roc(testing$class,
                      predict(x, testing, type = "prob")[, "superhost"]))
model_list_roc %>%
  map(auc)
```

```
## $original
## Area under the curve: 0.9199
##
## $down
## Area under the curve: 0.9177
##
## $up
## Area under the curve: 0.9216
```

```
# function to get roc results for each model
results_list_roc <- list(NA)
num_mod <- 1

for(the_roc in model_list_roc){
  results_list_roc[[num_mod]] <- data_frame(tpr = the_roc$sensitivities,
              fpr = 1 - the_roc$specificities,
              model = names(model_list)[num_mod])
  num_mod <- num_mod + 1
}

results_df_roc <- bind_rows(results_list_roc)

# plotting roc curve for each model
ggplot(aes(x = fpr,  y = tpr, group = model), data = results_df_roc) +
  geom_line(aes(color = model), size = 1) +
  geom_abline(intercept = 0, slope = 1, color = "gray", size = 1) +
  ggtitle("Area Under the Curve") +
  xlab("false positive rate") +
  ylab("true positive rate")
```

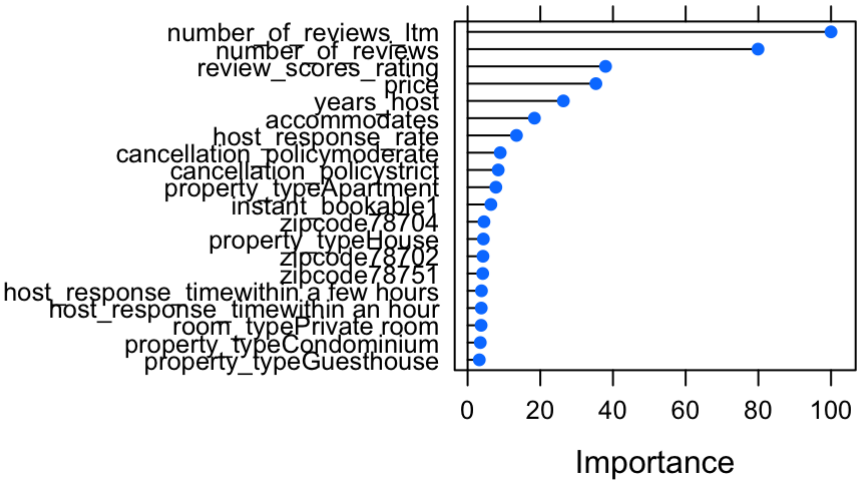


Based on AUC, `rf_up` appears to be the best model. It has a larger AUC than the other models. The lowest AUC is `rf_down` due to the higher false positive rate.

Model Selection and Further Interpretations

Based off of the class balances, accuracy, and AUC, I was able to determine `rf_up` as the optimal model. Given this model selection, I decided to use this model and determine what features are most important for predicting a Superhost.

```
# evaluating variable performance
imp <- as.data.frame(varImp(rf_up)$importance)
imp <- rownames_to_column(imp) %>%
  dplyr::rename(variable = rowname) %>%
  arrange(-Overall)

plot(varImp(rf_up), top = 20)
```



The business model of Airbnb, guests and hosts mutually reviewing each other, is vital to a host being a Superhost. Having many reviews, not just the last 12 months, but over years, is the greatest indicator of being a Superhost. Longevity is important. As shown in the EDA, having a strict or moderate cancellation policy is also indicative of being a Superhost.

Conclusion

How does the EDA and machine learning model hold up to Airbnb's standards? I added interpretations from my EDA to each bullet point. To review, according to Airbnb, the minimum criteria to become a Superhost is to have:

- An overall rating greater than 4.8 based on reviews in the past year
- A minimum of 10 stays in the past year
- A cancellation rate of less than 1%
- A 90% response rate
- An account in good standing

An overall rating greater than 4.8 based on reviews in the past year In the EDA, we saw that Superhosts have, on average, higher review scores than Non-Superhosts. Additionally, review scores, along with number of reviews (lifetime and last 12 months) was a significant variable in the machine learning model.

A minimum of 10 stays in the past year Although we don't know the count of stays in the past year, we can assume the count of stays based on the number of reviews. From the model, we were shown that number of reviews is actually the most vital variable to predicting if a listing is hosted by a Superhost in Austin.

A cancellation rate of less than 1% The data does not collect the amount of cancellations that the host contributes, but going back to the EDA, we saw that Superhosts typically had a more strict cancellation policy for their guests, which is also emphasized in the model. From this, we can assume that the host takes their role as a host seriously, and holds their guests to the same standards they hold themselves, i.e., being committed.

A 90% response rate Superhosts, on average, have a higher response rate than Non-Superhosts. As shown in the model, responding within a few hours is an important factor to being a Superhost.

An account in good standing This is a vague statement, but given a host having high reviews, many bookings, low cancellations, and a high response rate, it would make sense for these accounts to be "in good standing".

Other key takeaways Specific to Austin, the listing zipcode can be a good indicator of if the host will be a Superhost. Majority of the Superhosts in the Austin area are located in 78704 (South Austin). South Austin has access to some sought-out Austin destinations such as Zilker Park, Lady Bird Lake, and South Lamar.

Future Work

I think it would be interesting to re-run this model but on different cities, such as New York, and see if it holds true. I would like to test models other than the random forest and see how they predict, perhaps I can get a better accuracy. Additionally, I think there is much to be done with Natural Language Processing (NLP) for the guest reviews.