A Statistical Analysis of Airbnb Data

Math 533 Statistical Learning

Seth Arreola

2022-12-11

Abstract

Airbnb falls under what is considered a 'sharing-economy', which amounts to a market place where individuals physical assets can be turned into shared services. Many issues have been raised concerning companies who opperate in such economies, Airbnb specifically has a multitude of issue all the way down the the inability of their users to even adequately utilize their worth in Airbnb's market place. In the following analysis, we develope a predictive engine to predict the price per night of a given listing to the benifit of property owners, and help them gauge their listings worth. The study is focused on Airbnb's San Francisco market place, where a model was developed which explains 75% of the variability in these Airbnb listings.

Contents

Introduction/Background

Airbnb Overview

The following paper is a statistical analysis on Airbnb data. Airbnb as a company is based in San Francisco, California, operating an online marketplace focused on short-term homestays and experiences. The company acts as a broker and charges a commission from each booking. The company was founded in 2008 by Brian Chesky, Nathan Blecharczyk, and Joe Gebbia. Specifically this short-term homestays refers to what the name Airbnb stands for "Air Bed and Breakfast", meaning it is a service that lets property owners rent out their spaces to travelers looking for a place to stay. Travelers can rent a space for multiple people to share, a shared space with private rooms, or the entire property for themselves.

Shared Economy

Airbnb is one of a few unique companies that fall under the umbrella of whats referred to as operating in a sharing economy. The "sharing economy", is a concept which is a relatively new economic model "based on the peer-to-peer activity of obtaining, giving, or sharing access to goods and services, coordinated through community-based online services" (Hamari et al., 2016)[1]. The shared economy also has many names; shared market, peer-to-peer economy, access-based consumption, collaborative consumption, and platform economy. Sharing, of course, is a communal concept that has been around and used for thousands of years, however with the recent advances in technology and big-data, such a economic model is far easier to use. Now, individuals can easily turn their physical assets into shared services. In the case of Airbnb, property owners turn their physical properties into a shared bread and breakfast service.

The Issue at Hand and our Goal

From the above circumstances much criticism and arguments against shared economies and, Airbnb specifically, have been raised. Issues range from worker-rights to impact on traditional businesses in this industry, to impact on local housing markets. Lack of regulations ands structure are at at the forefront of many of these issues. In this analysis, we aim to solve one issue that some may argue is consequentially insignificant with respect to others problems, however it demonstrates the inefficiency of Airbnb as it stands currently. Specifically, "research argues that almost all hosts fail to maximize their potential profit due to poorly pricing their listing" (Gibbs et al., 2018)[2].

When property owners are interested in renting out their property or a portion of it, they are left to their own devices for pricing. If one of these property owners looks to Airbnb to direction, they are referred to the "How to set a pricing strategy" forum/page unser Airbnb's Resource Center. There, they are suggested to "Do a little market research", "Consider your location and hospitality", and "Stand out with great pricing". These of course are merely suggestions, Airbnb does however allow property owners to utilize their "Smart Pricing Tool". The Smart Pricing Tool, gives a recommended price per night for a given listing based on the features of said listing. However, the Smart Pricing Tool requires the property owner to set a minimum price that they feel comfortable with, meaning their tool gives a biased estimation of listing pricing. This leads us to the primary goal of the following analysis, we would like to develop a statistical engine that predicts the price of a given Airbnb listing base solely on the features of that listing and if possible identify the features that contribute/impact price the most, to the benefit of these property owners.

The Data

The data use in this analysis to reach our goals is based on Airbnb's San Francisco market place. The data was retrieved via Inside-Airbnb[4], which is a mission driven project that provides data and advocacy about Airbnb's impact on residential communities. Data is gathered and contributed for almost all cities Airbnb operates in across states and countries.

San Fransisco Data

The San Francisco data specifically contains 75 features relating the the listing and owner.

Table 1: Feature names in San Fransisco data set

The Names of our Features in the Data		
First 25 Features	Second 25 Features	Last 25 Features
id	host_has_profile_pic	has_availability
listing_url	host_identity_verified	availability_30
scrape_id	neighbourhood	availability_60
last_scraped	neighbourhood_cleansed	availability_90
source	neighbourhood_group_cleansed	availability_365
name	latitude	calendar_last_scraped
description	longitude	number_of_reviews
neighborhood_overview	property_type	number_of_reviews_ltm
picture_url	room_type	number_of_reviews_l30d
host_id	accommodates	first_review
host_url	bathrooms	last_review
host_name	bathrooms_text	review_scores_rating
host_since	bedrooms	review_scores_accuracy
host_location	beds	review_scores_cleanliness
host_about	amenities	review_scores_checkin
host_response_time	price	review_scores_communication
host_response_rate	minimum_nights	review_scores_location
host_acceptance_rate	maximum_nights	review_scores_value
host_is_superhost	minimum_minimum_nights	license
host_thumbnail_url	maximum_minimum_nights	instant_bookable
host_picture_url	minimum_maximum_nights	calculated_host_listings_count
host_neighbourhood	maximum_maximum_nights	calculated_host_listings_count_entire_homes
host_listings_count	minimum_nights_avg_ntm	calculated_host_listings_count_private_rooms
host_total_listings_count	maximum_nights_avg_ntm	calculated_host_listings_count_shared_rooms
host_verifications	calendar_updated	reviews_per_month

We can see that a number of these features relate to the host themselves, location of the property, availability, and reviews. Of these 75 features, 25 are character or strings, 37 are numeric, 8 are logical or boolian, and 5 are date features. In summary we have these 75 features in the dataset that contains 6,629 San Francisco Airbnb listings.

A note on the utility of the data. Although Airbnb data can be found for almost any location Airbnb operates in, only the current years data is available by Inside-Airbnb. Meaning we are limited in our analytical possibilities in some sense. For example the price per night of a given listing is the current price of that listing the last time that locations data was scraped by Inside-Airbnb. Thus, we are essentially limited to non-time-series analysis. It is possible to obtain previous years data, which would allow for the modeling of seasonal trends and such, however such data must be requested with a fee. Therefore, time-series analysis is left for future studies/analysis.

Cleaning and Wrangling

A number on cleaning and wrangling techniques were employed on this dataset to get it analysis and modeling ready. First and foremost, as we can see in Table 1, there exists many features in the dataset that are not suitable as predictive features for the price per night of listings, for example 'host_id'. Moreover there are a number of features which contained duplicate information. A full list of the features removed, due to the two reasons above, is given in Table 2 found in the appendix.

The next step in the data cleaning process entailed manipulating, and transforming a number of features. Of these operations, a number of which are relatively elementary which we easily performed via the "dplyr" an package in R which contain a set of commands similar to SQL commands, see references [6] for package details. Of these operations, they can be classified into 2 distinct groups; cleaning/transforming of character features in numeric features, and transformation of date features into numeric features.

Cleaning/Transformation of Character Features.

Many of the 25 character/string features in the dataset are only recognized as such because of something like a comma or dollar sign, and really these features are numeric. In order to utilize these features as listing price

per night predictive numeric features, they must be cleaned of these "string" characters so that the statistical software (R) can utilize them. For these cleaning procedures the package "stringr" [7] is also utilized for easy string manipulation. The following lists the transformations performed for string cleaning:

- Removed '%' from 'host_response_rate', then transforming it to a probability scale and set as a numeric feature.
- Removed '%' from 'host_acceptance_rate', then transforming it to a probability scale and set as a numeric feature.
- Removed '$' and ',' from 'price', and set as a numeric feature.

Cleaning/Transformation of Data Features

As noted earlier, we omit time-series analysis for future work due to the limitations of the data. The San Francisco dataset dose however still contain some data features. Specifically, 'calendar_last_scraped', 'host_since', 'last_review_date", and 'first_review_date'. The information in these date features still may contain predictive information on the price of the listing per night, which we would like to utilize. To this point, we transform these variables into the following predictive numeric features:

- 'last_review' and 'calendar_last_scraped' are used to create 'months_since_last_review'.
- 'host_since' and 'calendar_last_scraped' are used to create 'months_being_host'.
- 'host_since' and 'first_review_date' are used to create 'months_till_first_review'.

Cleaning/Transformation of Text Features

After the removal of duplicate information, and the transformation of numeric strings to numeric features, there remained 4 character features that are indeed text features. Specifically, 'description': a description of the listing itself ranging from a single sentence to a paragraph, 'neighborhood_overview': a description of the neighborhood that the listing resides in, 'host_about': a description of the host, and lastly 'amenities': a list of amenities and features that the listing provides for renters. Given the description of each of these features its reasonable to assume that predictive information on the price of a listing exits in these text features. To extract this information, a naive approach is taken, meaning the extraction of the most frequent words in these texts.

Figure 1: Word Cloud of neighbourhood-overview

Figure 1 shows a word cloud of the most frequently used words in 'neighborhood_overview'. However, it becomes more clear which words are most frequent for this feature in Figure 2. In order to filter through these text variables and identify the most frequent words the R-package's "wordcloud"[8] and "tm"[9] where used, for more information on these packages see References. When sorting through words in these text features, stop-words "words in a stop list which are filtered out before or after processing of natural language data because they are insignificant", such as "the, is, and", where filtered out.

Some of the most common words remaining are then selected to become Boolean/logical features. For example, for 'neighborhood_overview' the words:

- restaurants
- parks
- walk
- shops
- quiet

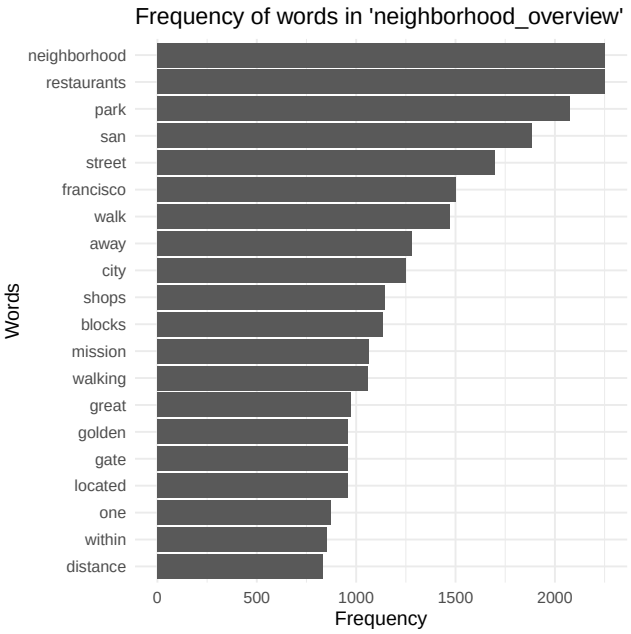

Figure 2: Frequency plot of neighbourhood-overview

are selected, meaning a boolean feature for each of these words is created indicating if the word was mentioned in the neighborhood description for a given listing. The same process is done for the "amenities" feature, where the boolean features:

- alarm-amenity
- dryer-amenity
- wifi-amenity
- smoke-amenity
- washer-amenity

are created, indicating if these specific amenities are in a given listings list of amenities. From the "host_about" a single feature was created indicating if the host has a family, and finally the "description' feature mainly contained information about the listing that can be found in other features in the dataset so no boolean feature where created from it. To view word-clouds and word-frequency plots of"amenities", "host_about", and "description" see the Appendix.

Missing values and Data inputation

This dataset did contain observations which had one or more features with NA's or missing values. There exists 10 features in the dataset which have 900 or more missing values, many of which come from features regarding reviews. See Figure 21 in the Appendix for an informative visualization of the number and patterns of missingness in the data.

For the purpose of imputation, a model based methods was selected. Specifically the algorithm 'MissForest', which is a random forest imputation algorithm for missing data. The algorithm works iteratively in the following manner.

- Initially imputes all missing data using mean/mode of observed observations.
- For each variable with missing values, MissForest fits a random forest on the observed part of the data and predicts the missing part.
- Once each feature with missing values has been imputed the process starts again on the first feature with missing values.
- The process proceeds until a stopping criterion is met, or a maximum number of user-specified iterations is reached.

This algorithm was originally developed by Stekhoven & Buhlmann (2011)[10] and cited as quite robust, see References for more details.

Exploritory Data Analysis

A Geographic Overview

The exploration of our data begins with a general geographic overview of our data.

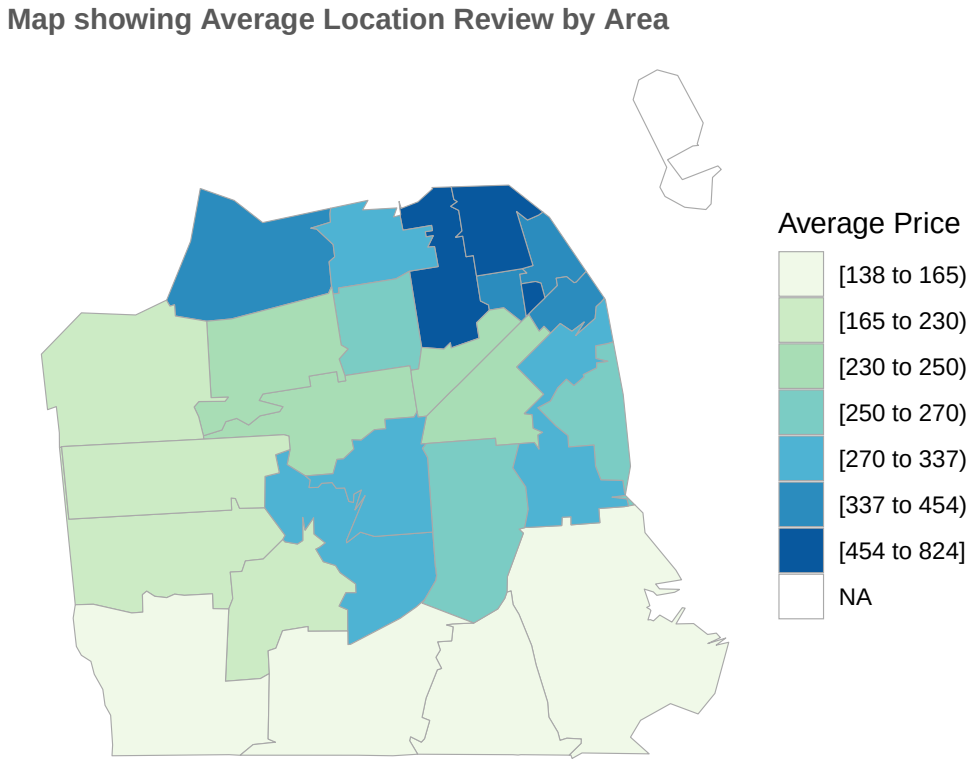

Figure 3: Avg. Price of Listings Per Zipcode

Figure 3 displays the average price of listings (per night) grouped by the listings zip-code. Viewing the choropleth map, we can see immediately that listings in the North-East portion of the map are on average more expensive. Specifically, areas around Fishermans-Wharf, Golden-gate Park, and the Bay Bridge seem to have a higher price per night. Notice, we can visually see that in San Francisco, as you move from the South-Western listings to the North-Eastern listings, the average price seems to increase, with respect to zip-codes. Interestingly, if we color zip-code by the average location reviews instead of average price per night, we will see that the zip-codes which have the highest price per night are not necessarily the ones with have bet best reviews. A plot which colors zip-codes via location review is given in Figure 20 in the Appendix

The map displayed in Figure 3 is a choropleth map, which is a type of statistical thematic map that uses pseudo color, i.e., color corresponding with an aggregate summary of a geographic characteristic within spatial enumeration units. To create the one given in this paper the package choroplethr is used created by Ari Lamstein, for more information see References[11].

Price of Listings per Night

We will now take a closer look at the price of listings per night, the feature of interest that we will build a predictive model for.

Table 2: Summary Statistics of Listing Price

Minimum	Median	Mean	Maximum
24	159	303.6021	25000

Notice in the summary statistics for 'price' the large difference between the median price of the data and the average price of listings in the data (150 - 304), indicating the data is skewed or contains large outliers. Visually inspecting price, we see that this holds true.

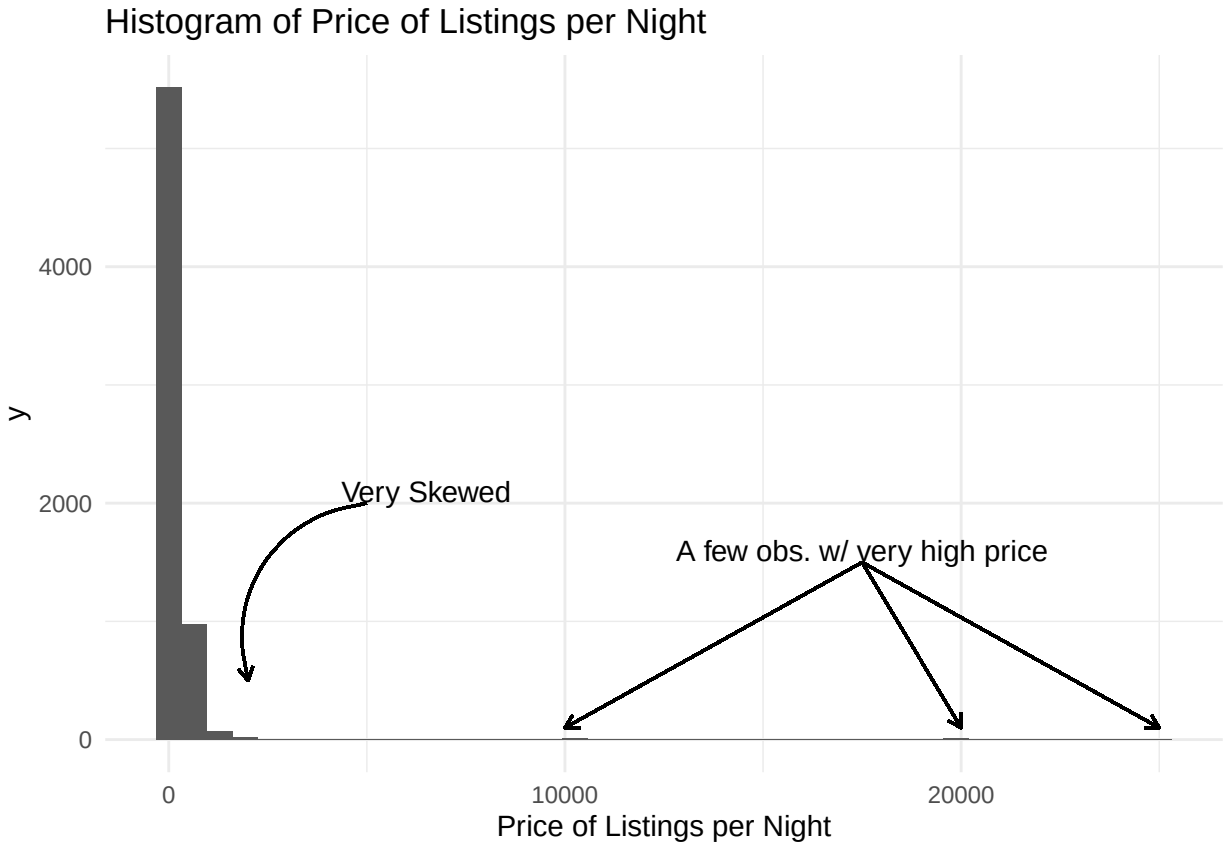

Figure 4: Histogram of Price per Night of Listings

We can see that 'price' is indeed quite skewed with some listings with extremely high prices from Figure 4. Astonishingly, there are three listings in the data which have a 25,000 charge per night. These listings are

- Harbor Court Hotel, Bay View Queen Room
  - 2 guests, 1 bedroom, 1 private bath
  - Description: A historic Army-Navy residence has become one of San Francisco's premier waterfront boutique hotels. The Newly redesigned hotel delivers a sophisticated atmosphere with unobstructed views of the San Francisco Bay. Rooms are both stylish and practical for all visitors to enjoy.
- Harbor Court Hotel, Bay View King Room
  - 2 guests, 1 bedroom, 1 bed, 1 bath
  - Description: A historic Army-Navy residence has become one of San Francisco's premier waterfront boutique hotels. The Newly redesigned hotel delivers a sophisticated atmosphere with unobstructed views of the San Francisco Bay. Rooms are both stylish and practical for all visitors to enjoy.
- Hotel Griffon by the Bay, Queen Room

- 2 guests, 1 bedroom, 1 bed, 1 private bath
- Description: Indulge yourself by choosing a queen sized room in a historic San Francisco boutique hotel directly on San Francisco's Embarcadero. Voted "Best Boutique Hotel" & "Best 24-Hour Getaway" by San Francisco Magazine, you'll enjoy exceptional amenities which include well-appointed guestrooms, & on-site dining at Perry's Embarcadero.

It is worth noting that from the date that the data was scrapped, these three listings have not had a reservation. Therefor it is possible that the current price is a typo on the listers part.

In any case, due to the shewness of price a log-transformation is performed on it in order to visualize price in the context of other variables. For example see Figure 5, which contains our newly transformed 'Log-Price', and an example of how variables are far more comparable when viewing on the log-scale.

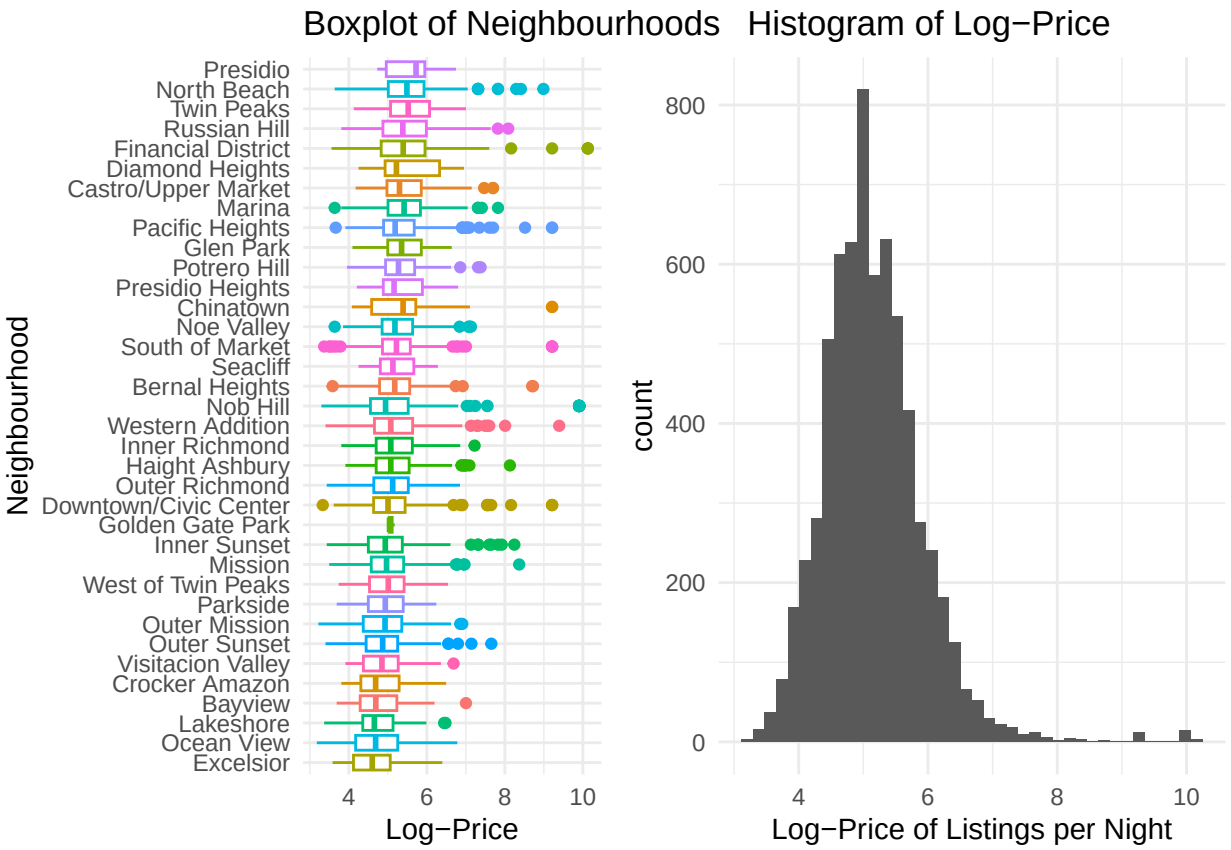

Figure 5: Histogram of Log-Price and Boxplot of Neighbourhoods w.r.t. Log-Price

Numeric Predictor Features

For the prediction of price we have 27 numeric features to work with. Firstly, a quick investigation of colinearity and correlation between these features and our response was performed. See Figure 22 in the Appendix to see a visualization of correlation within the data, expressed in a correlation heat map. From this investigation we first make note of the existence of pockets of colinearity within our predictive features. Specifically, colinearity among

- The different 'review' based features.
- The three features that describe the amount of reviews a listing has on different time scales.
- Beds, Bedrooms, and Accommodates.

Note that it is not our explicit goal to measure the magnitude a given feature has on the price, it is our goal to predict price as accurately as the data will allow for and find the features which impact this prediction the

most. Therefore, while the colinearity within the data does not prohibit us from achieving our goals, it does stop us from measuring and inferring on the exact amount. This is discussed more in-depth in the methods and results sections.

The second thing we make note on, is the amount of correlation between our response and predictive features.

Table 3: Highest Correlation Between Predictors and the Response Vriable

log_price	accommodates	bedrooms	beds
1	0.5576076	0.4959505	0.4346431

Notice from Table 3, we can see the three predictive features with the highest correlation with our price is:

- Accommodates: The number of people that a listing can host.
- Bedrooms: The number of bedrooms a listing contains.
- Beds: The number of individual beds a listing contains.

Lets now take a closer look at these variables with respect to price, beginning with 'accommodates'.

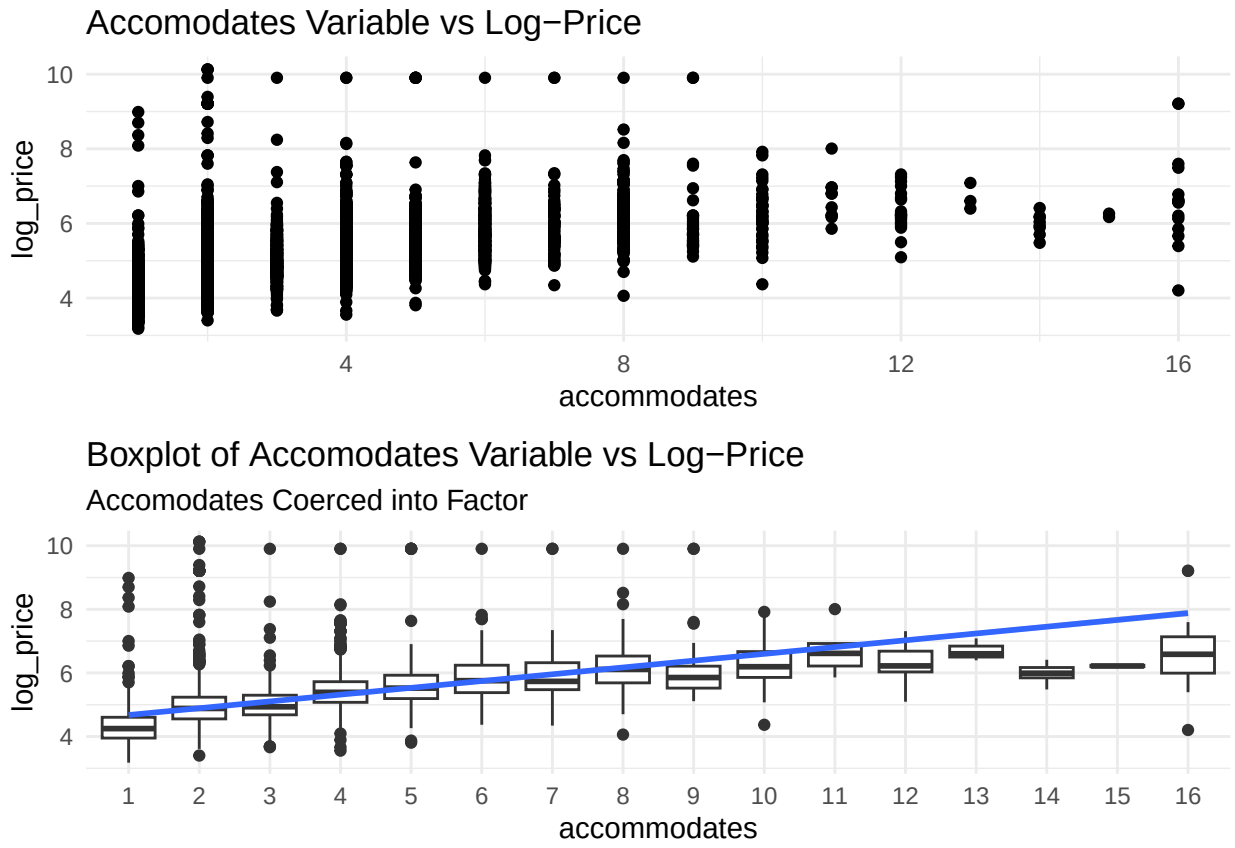

Figure 6: Accommodates vs Log-Price

From Figure 6 we can see two plots both depict the relationship between accommodates and price (on the log scale). The number of people a listing can hold, naturally is already discretized. Therefor coercing the accommodates variable into a categorical variable allows us to visualize this relationship much easier. Once the transformation was performed a simple ordinary regression line is added to the plot to display that as the value of accommodates increases so does the expected log-price and therefore price in general.

The same story holds for for 'beds' and 'bedrooms', as the value of all of these variable increase so does the expected price. This is no surprise given the correlation given above in Table 3, however for a closer look see Figure 23 and 24 in the Appendix.

Catagorical Predictive Features

While exploring the data many of the categorical predictor variables do not seem to have an obvious relationship with the price of listings, at least visually. See the Appendix to see a host of plots attempting to find a relationship between these predictor variables and the response. One categorical variable did have an obvious relationship with the response, 'shared-bathrooms'

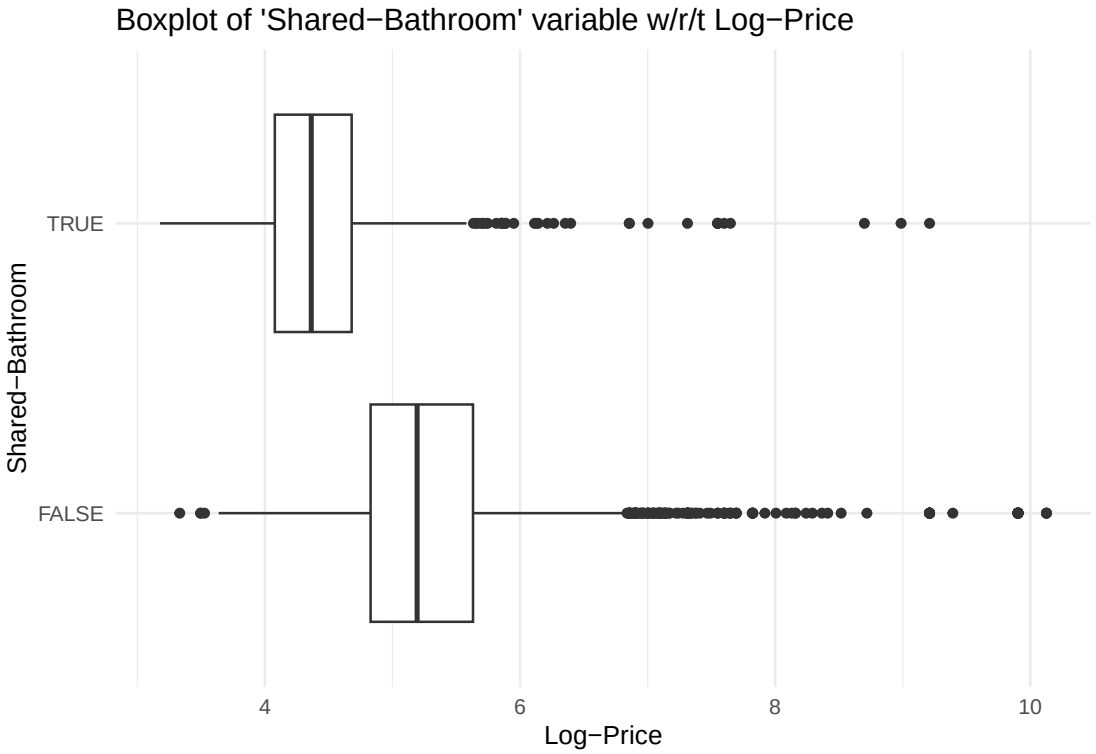

Figure 7: Boxplot of Shared-Bathrooms w.r.t. Log-Price

Which can be seen above in Figure 7. The Shared-Bathrooms variable indicates whether or not a renter would have to share a bathroom with the listings host or that are given private bathrooms.

Extensive Data Exploration

Additional statistical data exploration techniques where performed to reduce the data as a whole for exploration or subset the data into groups for exploration. Firstly, principle component analysis was performed to reduce the dimension of the data for visualization and exploration.

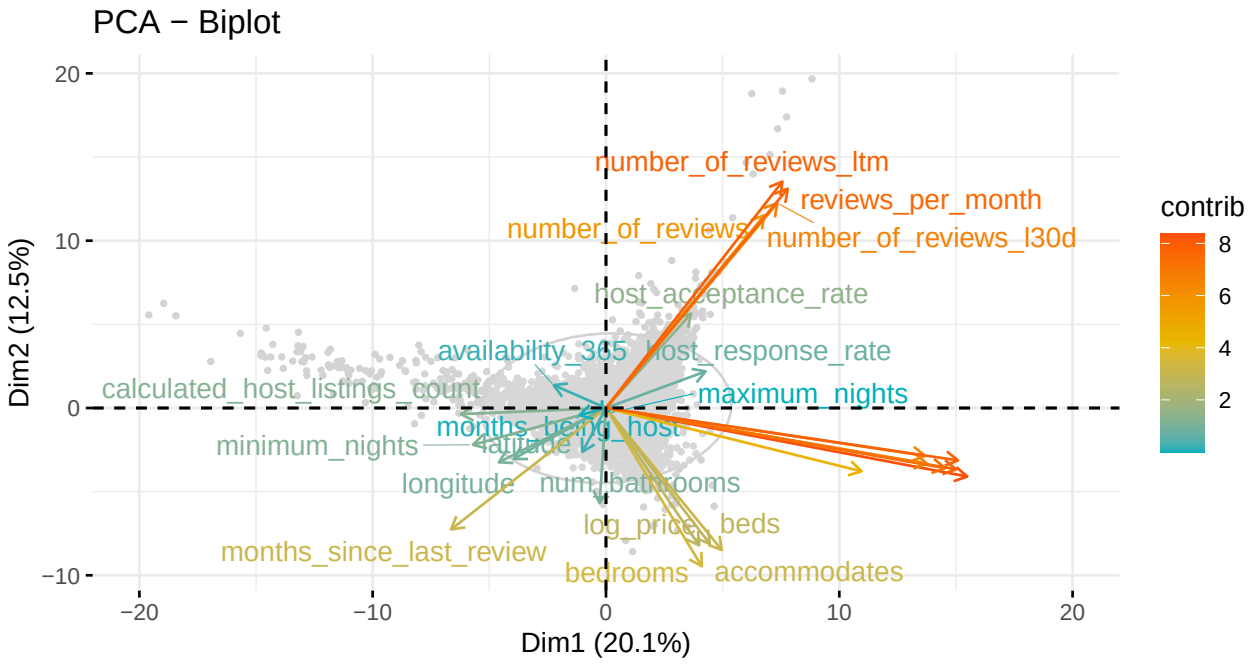

Figure 8: Biplot of PCA

Recall that principle component analysis transforms our predictor space into an orthogonal space, through the use of eigen decomposition. Once this is completed we can visualize the results nicely through a biplot, as seen in Figure 8, which is an overlay of a score plot and a loading plot. Here we can once again see the correlation between our features which is represented as the cosine of the angle between any two features. However, now we are less interested in the relationship between our feature but more so how our individual observations relate to these feature in this two-dimensional space (considering the first two principle components).

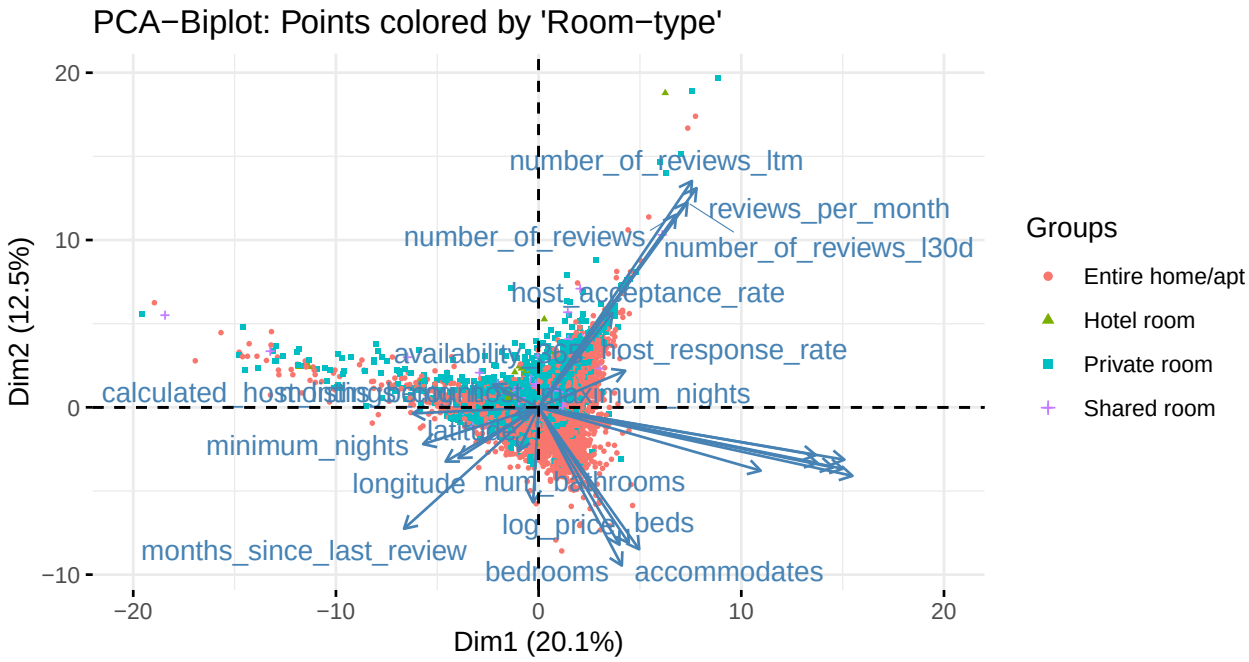

Figure 9: Biplot with observations colored by 'Room-Type'

For example, in Figure 9 each observation is now colored by 'Room-type'. Now viewing the data in this two-dimensional space we can now see some relationship between our categorical variable (room-type) and price. The observations near the log-price loading are overwhelming Airbnb listings for an entire property. To see another example of a categorical variables relationship becoming reveled when visualizing it in this reduced dimensional space see the Appendix for a short discussion on what Airbnb considers a "Super-host".

The final method of EDA performed on this dataset was clustering. That is the application of a clustering algorithm to identify an informative sub-populations within the dataset. As we will see although the results obtained are not as conclusive as we would like, the results do open the door for additional analysis in future work. Since the data contains a mixture of data-types, numeric and categorical, we apply an extension of K-means, K-Prototypes, which allows for the distance of both data types to be measured. Clusters are assigned using a modified measure of distance

$$d(x_i, y_i) = \sum_{m=1}^{q} (x_i^m - y_i^m)^2 + \lambda \sum_{m=q+1}^{p} \delta(x_i^m, y_i^m).$$

Where m is an index over all variables in the dataset where the first q variables are numeric and the remaining $p - q$ variables are categorical. Note that $\delta(a, b) = 0$ for $a = b$ and $\delta(a, b) = 1$ for a $a \neq b$, and $d()$ corresponds to weighted sum of Euclidean distance between two points in the metric space and simple matching distance for categorical variables (i.e. the count of mismatches). The trade off between both terms can be controlled by the parameter λ which has to be specified in advance as well as the number of clusters k. For more information see References [12-14].

Thus, when applying this method the crux is, like many other clustering algorithms, to determine the number of clusters suitable for the data. The number of clusters was searched for via the "elbow-method", and analysis of clusters within sum of squares. However, during this process an intriguing discovery was made.

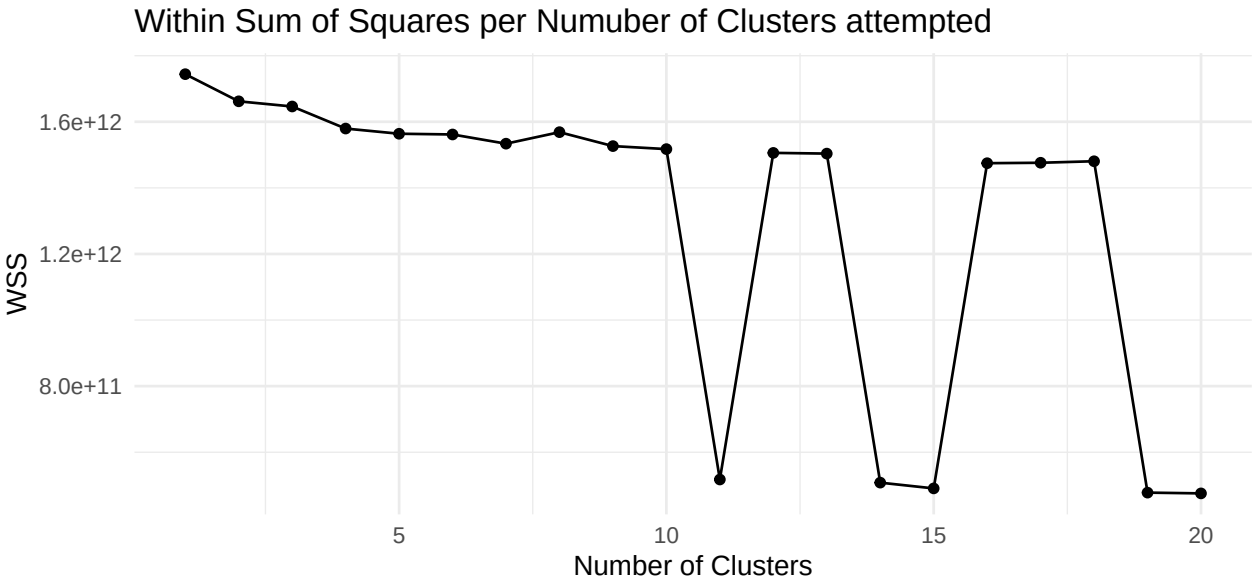

Figure 10: Within Cluster Sum of Squares per Number of Clusters

From Figure 10 we can see some very interesting behavior from the data, there does not seem to be an obvious "elbow", but moreover the within cluster sum of squares begins to oscillate as the number of clusters increase. Current research could not be found to explain this behavior but future work may still be fruitful.

Methods

Recall the the primary goal of this analysis is to develop a predictive engine for prediction of a given listings price per night. Therefor as we have seen when exploring that data, we do have some colinearity within the data, but we do not need to worry as this will only hamper our inferential ability. Thus to build and identify the most accurate model we can, we begin with five statistical model frameworks.

- Multiple Regression
- Lasso Regression
- XGBoost
- Neural Network

Model Tuning

Each of the listed models, with the exception of multiple regression, requires the selection of hyper-parameters. Therefor for each of the models their tuning procedure is listed in the following:

Lasso

- 10-fold cross-validation grid search (CV over a grid of possible hyper-parameter combinations)
- The grid used contains 20 different possible values of lassos λ penalty

Random Forest

- 10-fold cross-validation grid search (CV over a grid of possible hyper-parameter combinations)
- The grid used is an 80 x 2 grid of possible combinations of
  - mtry: Number of variables randomly sampled as candidates at each split.
  - min_n: The minimum number of observations that must exist in a node in order for a split to be attempted.

XGBoost

- 10-fold cross-validation grid search (CV over a grid of possible hyper-parameter combinations)
- The grid used is a 40 x 6 grid of possible combinations of
  - tree_depth: The depth of a decision tree is the length of the longest path from a root to a leaf
  - learn_rate: the shrinkage applied to reduce quick overfitting
  - mtry: Number of variables randomly sampled as candidates at each split.
  - min_n: The minimum number of observations that must exist in a node in order for a split to be attempted.
  - loss_reduction: Minimum amount of reduction to the loss function
  - sample_size: Proportion Observations Sampled

Neural Network

- Two neural networks were trained
  - One hidden layer with 60 units
  - Three hidden layers with 70, 40, and 10 hidden units respectively

Model Selection

Once hyper-parameters were selected for each model that needed tuning. 10-fold cross-validation was performed again to choose between models. Specifically so that each model is trained and tested on the same folds/slits of the data to ensure equal and comparable results.

Table 4: Average RMSE across folds of CV prodecure

Random Forest RMSE	XGBoost RMSE	Neural Net RMSE	Lasso RMSE	OLS Regression RMSE
0.3860604	0.3768912	0.5198146	0.5720703	0.7196674

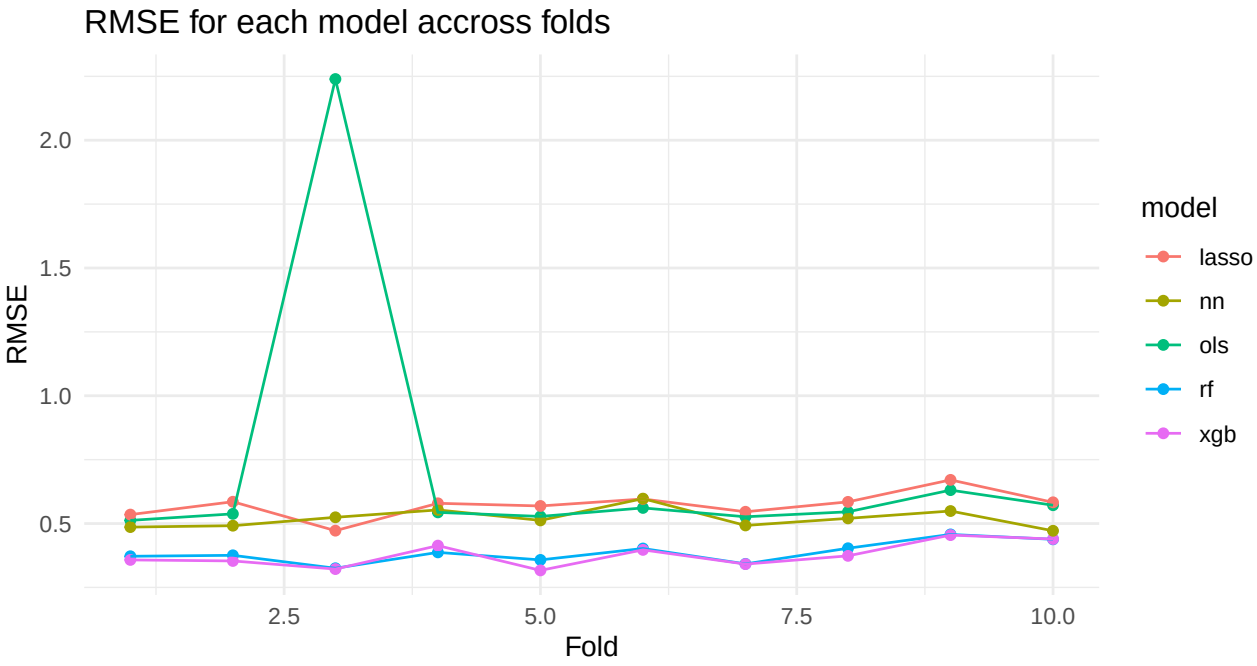

Figure 11: Root Mean Squared Error per Fold of Cross-Validation

We can see the cross-validations results from Table 4 and Figure 11 are both suggesting that tree/forest based approached performed the best. It should be noted the neural network used in the CV procedure for model selection is the NN with three hidden layers mentioned above, note this model was trained with 900 epoch's. During the CV procedure this amounted to $10 * 900 = 9000$ total epochs ran. It is reasonable to assume that an increase in training epoch would improve accuracy, however due to computational limitation of the machine this model was trained on we move forward with the tree-based models. In summary, the two tree based methods perform the best, the neural network and lasso models are comparable, and the multiple regression model performs the worst.

Depending on the repetition of the 10-fold cross-validation, the best model alternated between the two tree-based methods, Random Forest and XGBoost. Meaning to say, the best model depended on the split our our folds. Therefor we proceed with the Random Forest as our final model for computational purposes.

Once the final model is selected, the Random Forest model, it is validated on a hold out test set. Meaning the objective measure of fit (RMSE) is calculated on a hold-out set of the data that was untouched during model tuning and model selection.

Table 5: Random Forest RMSE on Validation Test Set

RMSE
0.37

We can see in Table 5 the results from this validation. The root mean squared error computed here is very similar to the value obtained in the cross-validation process during model selection, indicating we did not over-optimize on a subset of the data, and we proceed with this as our final predictive model.

Results

The results of the predictive engine developed under the methods section, for the purpose of predicting the price per night of a given listing are reported here. The average RMSE of the model is 3.7 (when price is on the log-scale), which was computed during the cross-validation process. However a more intuitive metric of the model's performance is R^2, which the model computes as a 'pseudo R-squared': 1-MSE/Var(y), where MSE is computed on out-of-bag observations.

Table 6: Random Forest R-Saured

R-Squared
0.7496

From Table 6 we can see that only about 75% of the variation in the price of Airbnb listings is explained by the model. Note that this very close to the results obtained in another study by Pouya Kalehbasti, Liubov Nikolenko, and Hoormazd Rezaei, on a very similar analysis but on New York Airbnb, see References[15].

Moreover, we can examine the importance of the features in the model to address our secondary goal (identify the features that contribute/impact price the most). Feature importance can be measured for a given variable for tree-based models by measuring the total decrease in node impurities from splitting on the variable, averaged over all trees. For regression, it is measured by residual sum of squares.

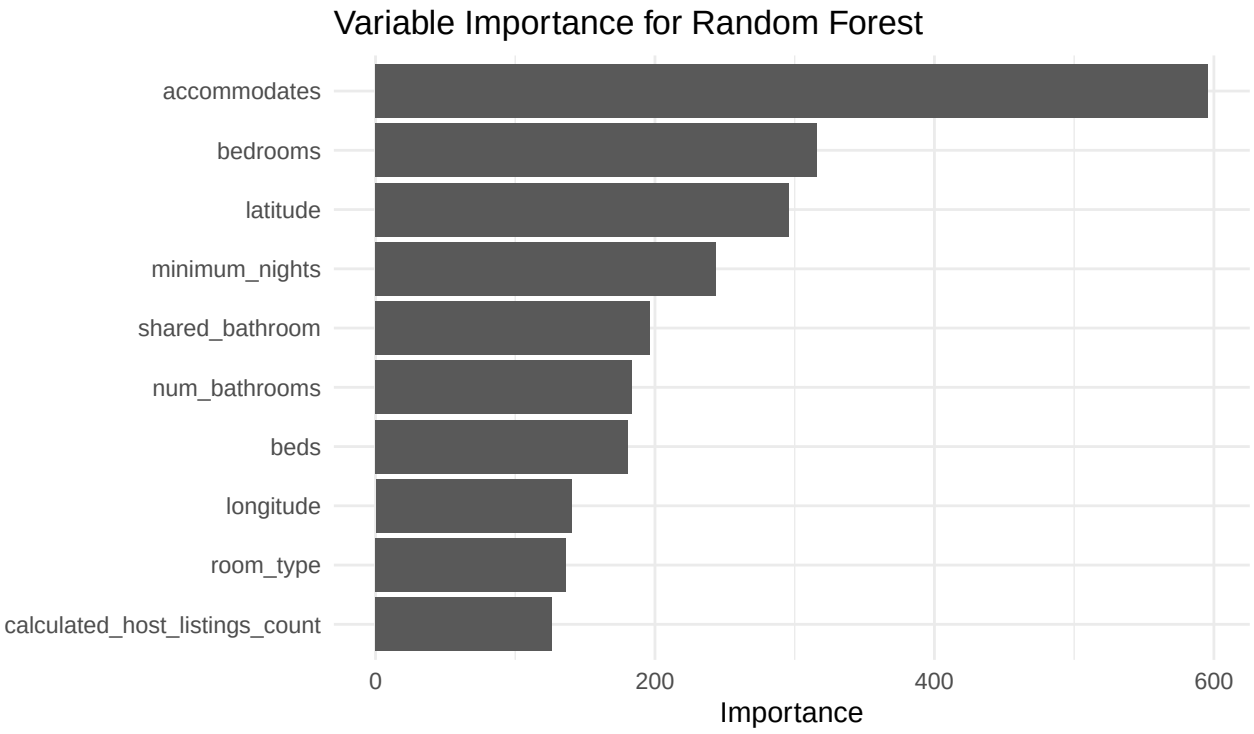

Figure 12: Feature Importance for our Finalizd Random FOrest Model

From Figure 12 we can see the three most important features for the developed model are 'accommodates', 'bedrooms', and 'latitude'. These shouldn't be a surprise to us as we saw from our exploratory data analysis:

- As we move to the North-East region of the map provided in Figure 3, the price on average increases.
- Accommodates and bedrooms both have a positive linear relationship between themselves and the price of listings per night.

To demonstrate the importance that 'accommodates' has on the model for the prediction of the price, we present a quick "case-study".

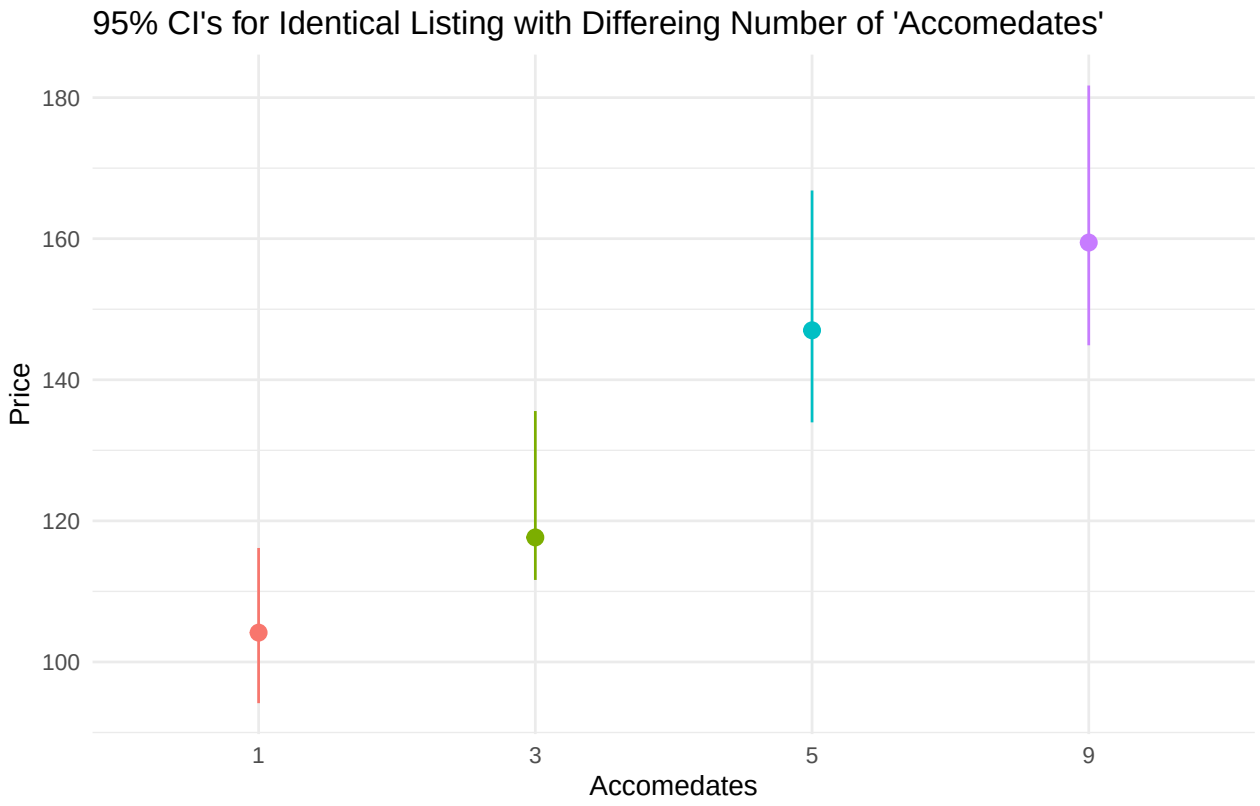

Figure 13: Case-Study on the impact that accommodates has on the predicted price on a specific listing

In Figure 13 we investigate the impact that the number of people a listing can accommodate has on a specific listing. The listing we look at is randomly selected from the dataset, which belongs to the Marina neighborhood in San Francisco. For this procedure the model was trained on all the data and then it predicted the price of this listing four times. However each time the model was told this listing accommodates a different number of people, 1, 3, 5, and 9. For each of these predictions a 95% confidence interval is also presented, which was computed via a boot-strapping methodology. We can see that there is about a 60 dollar difference in the predicted price, given by the model, when the number of people the listing holds changes from 1 to 9 people. Moreover the model suggests that the results are significantly different.

Summary/Discussion

In conclusion we were able to model and predict the price of a given Airbnb listing solely using the information of the listing itself provided on Airbnb's website. As noted in the result's, only about 75% of the variation in the price of these Airbnb listings is explained by the model. This, although not extremely accurate, still provides the beneficiaries of this analysis (property owners) a good indication of what their property is worth without accounting fo any of the individual biases of the property owner. The reason the model is not particularly accurate could be a multitude of reasons:

- Property owners inject a significant amount of personal bias when pricing their listing, resulting in a latent variable for the individual property owners which is difficult to account for.
- There simply might be a considerable amount of noise in the data. In other words the signal-to-noise ratio is quite low due to the simple fact that these features do not particularly explain listing price well.

In any case, as we arrived at similar metrics as other analysis arrived at for different city's/regions, and thus are able to provide property owners with an "idea of listing price". Moreover, we saw that the number of people a given listing can accommodate seems to be the most important indicator of listing price. However, recall that we saw colinearity between the number of people that can be accommodated, the bedrooms, and beds, thus in general we should say the size of the listing in total is the most important contributor to the price, and should be considered by property owners when building/posting their listing.

A number of questions and ideas also arouse from this analysis, that can be investigated and addressed more in depth in future work.

- It would be interesting and possibly beneficial to prediction performance to perform a text analysis on "neighborhood overview" and "amenities" and use the results as predictive features for the price.
- As seem in Figure 21 in the Appendix, we so see some relationship's between the the missing observations between our features. Utilizing this pattern/structure of missingness in data imputation could be beneficial.
- Further investigate San Francisco Airbnb data by analyzing the subsets of listings identified during clustering, and moreover investigate the oscillating behavior observed.
- Extend the scope of research beyond San Francisco. It would be interesting to see if the behavior observed is specific to the city level or extends to regions, for example at the state level.
- Further tuning of models may always improve prediction accuracy.

References

1. Hamari, J., Sj€oklint, M., Ukkonen, A., 2016. The sharing economy: why people participate in collaborative consumption. J. Assoc. Inf. Sci. Technol. 67, 2047e2059. https://doi.org/10.1002/asi.23552.
2. Gibbs, C., Guttentag, D., Gretzel, U., Yao, L., & Morton, J. (2018). Use of dynamic pricing strategies by Airbnb hosts. International Journal of Contemporary Hospitality Management, 30(1), 2–20. https://doi.org/10.1108/IJCHM-09-2016-0540
3. McNeil, Brandon, "Price Prediction in the Sharing Economy: A Case Study with Airbnb data" (2020). Honors Theses and Capstones. 504. https://scholars.unh.edu/honors/504
4. Airbnb: Resource Center "How to set a pricing strategy" https://www.airbnb.com/resources/hosting-homes/a/how-to-set-a-pricing-strategy-15
5. Inside-Airbnb: http://insideairbnb.com/get-the-data
6. dplyr: "The grammer of data manipulation" https://dplyr.tidyverse.org/
7. Stringr: https://stringr.tidyverse.org/
8. wordcloud: https://CRAN.R-project.org/package=wordcloud
9. https://CRAN.R-project.org/package=tm
10. Daniel J. Stekhoven, Peter Bühlmann, MissForest—non-parametric missing value imputation for mixed-type data, Bioinformatics, Volume 28, Issue 1, 1 January 2012, Pages 112–118, https://doi.org/10.1093/bioinformatics/btr597
11. Choroplethr: https://arilamstein.com/packages/
12. Amir and L. Dey. A k-mean clustering algorithm for mixed numeric and categorical data. Data and Knowledge Engineering, 63:88–96, 2007. doi: 10.1016/j.datak.2007.03.016.
13. Szepannek, G. (2018): clustMixType: User-Friendly Clustering of Mixed-Type Data in R, The R Journal 10/2, 200-208, 10.32614/RJ-2018-048.
14. Z.Huang (1998): Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Variables, Data Mining and Knowledge Discovery 2, 283-304
15. Pouya Kalehbasti, Liubov Nikolenko, and Hoormazd Rezaei (2019): "Airbnb Price Prediction Using Machine Learning and Sentiment Analysis" International Cross-Domain Conference for Machine Learning and Knowledge Extraction, pp. 173-184. Springer, Cham, 2021.
16. C. Michael Hall, Girish Prayag, Alexander Safonov, Tim Coles, Stefan Gössling & Sara Naderi Koupaei (2022) Airbnb and the sharing economy, Current Issues in Tourism, 25:19, 3057-3067, DOI: 10.1080/13683500.2022.2122418
17. Adeoluwa Akande, Pedro Cabral, Sven Casteleyn (2020) "Understanding the sharing economy and its implication on sustainability in smart cities" Journal of Cleaner Production, Volume 277, 20 December 2020, 124077

Appendix

Tables:

Table 7: Features Removed During Data Cleaning

First 10 Removed	Second 10 Removed	Third 10 Removed	Last 3 Removed
id	host_verifications	host_total_listings_count	availability_30
listing_url	neighbourhood_group_cleansed	calculated_host_listings_count_entire_homes	availability_60
scrape_id	bathrooms	calculated_host_listings_count_private_rooms	availability_90
source	calendar_updated	calculated_host_listings_count_shared_rooms	
picture_url	license	minimum_minimum_nights	
host_id	last_scraped	maximum_minimum_nights	
host_url	neighbourhood	minimum_maximum_nights	
host_location	calendar_last_scraped	maximum_maximum_nights	
host_thumbnail_url	host_neighbourhood	minimum_nights_avg_ntm	
host_picture_url	host_listings_count	maximum_nights_avg_ntm	

Figures:

Figure 14: Word-Cloud of 'description'

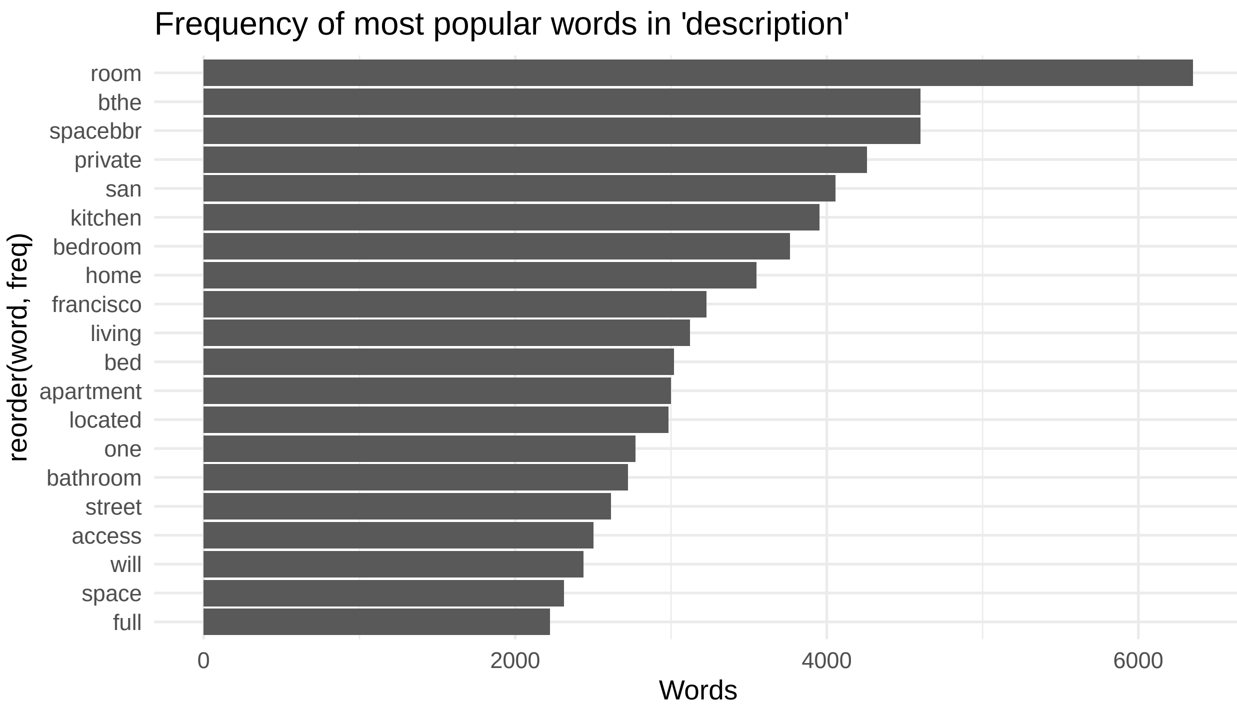

Figure 15: Frequency plot of description words

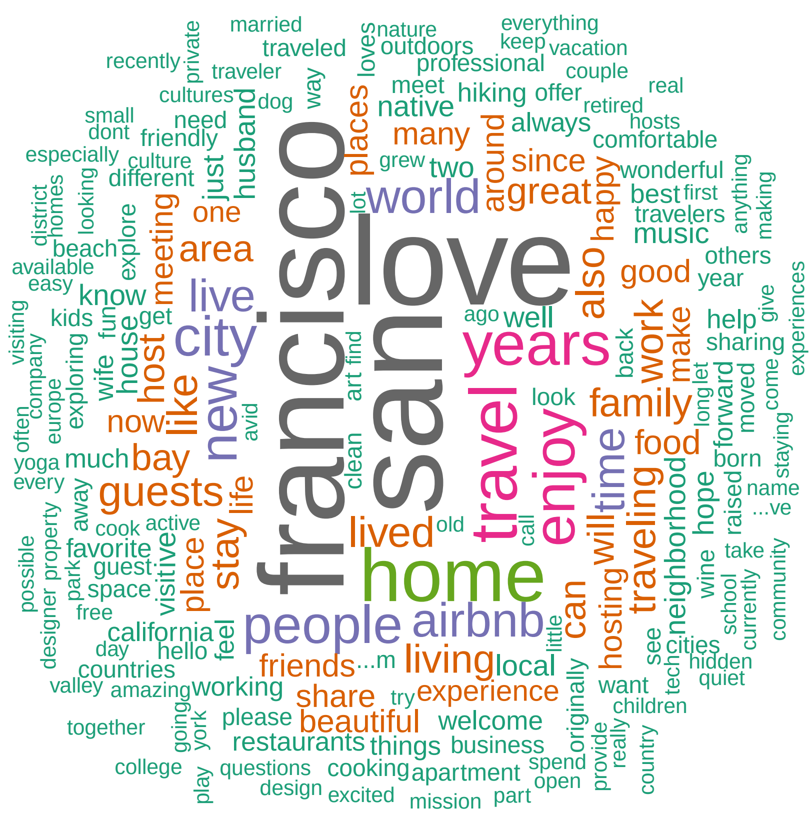

Figure 16: Word-Cloud of 'about-host'

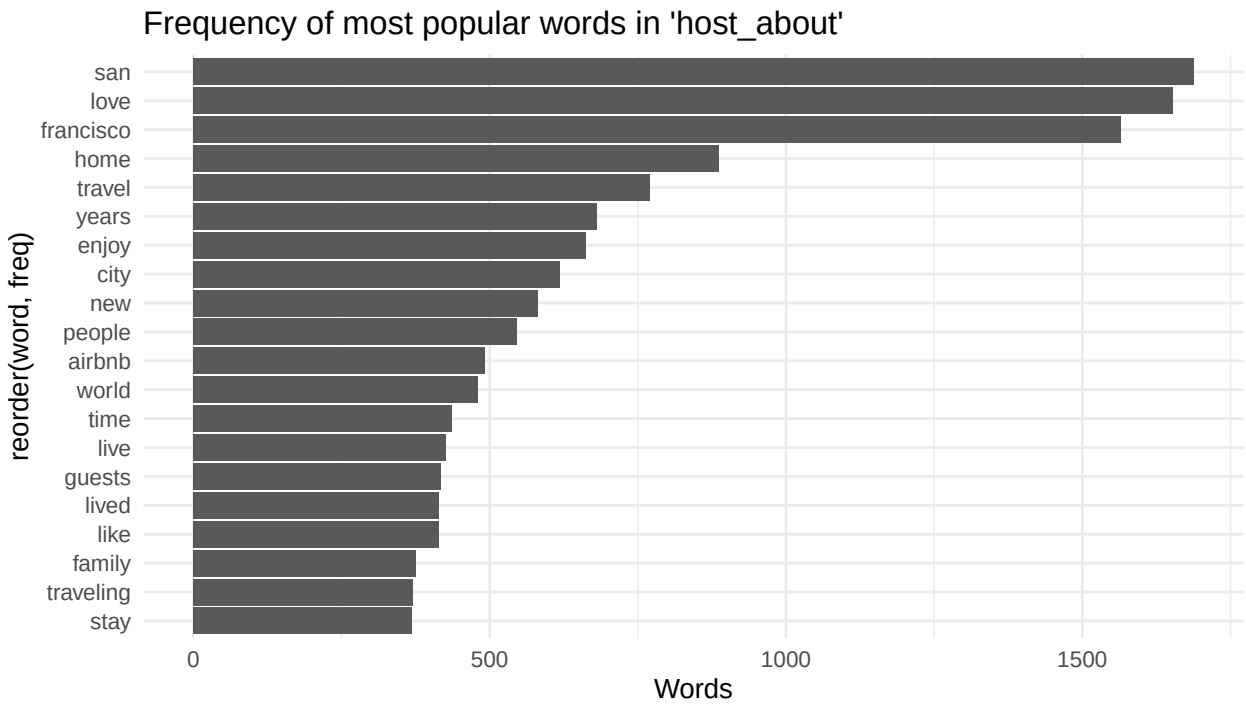

Figure 17: Frequency plot of about-hostwords

Figure 18: Word-Cloud of 'amenities'

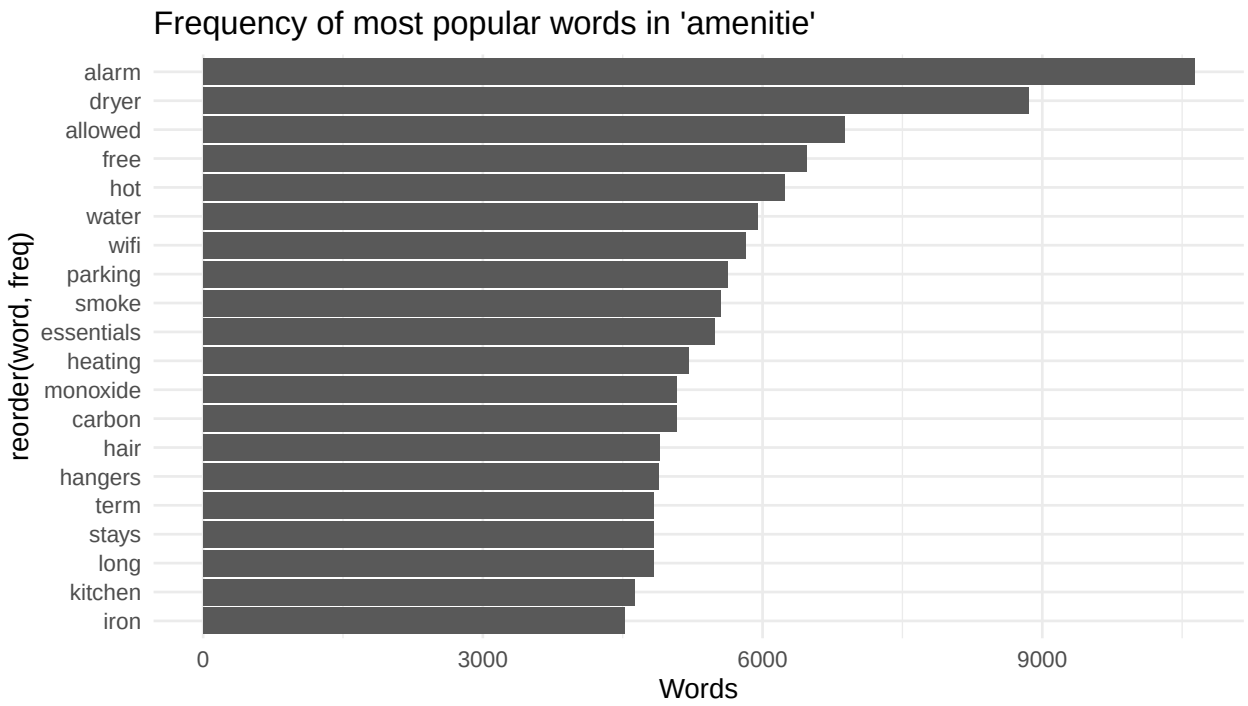

Figure 19: Frequency plot of amenities words

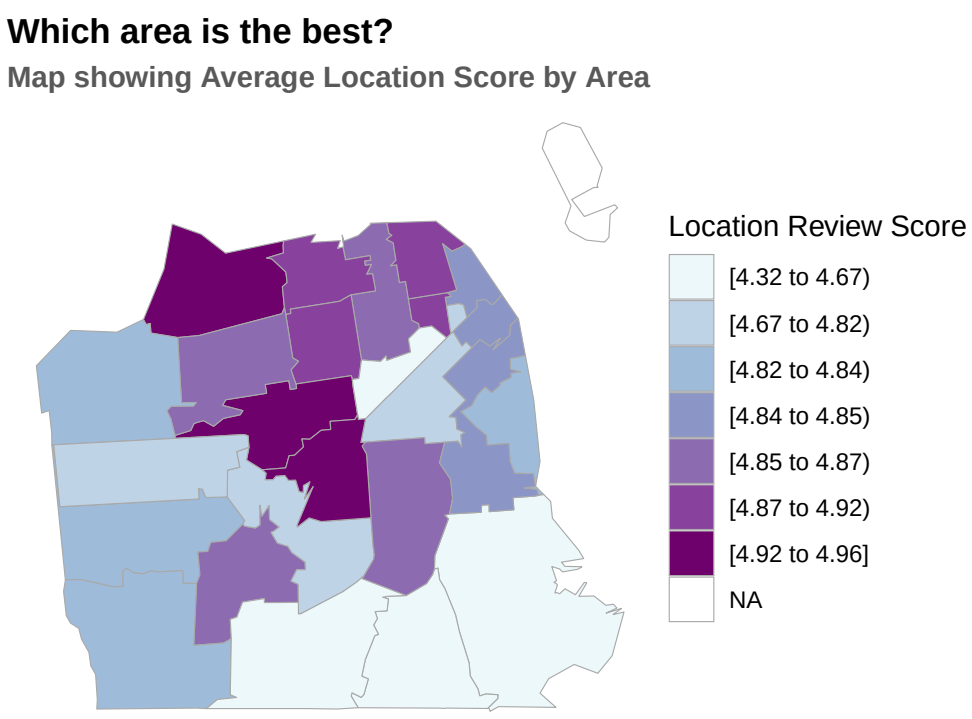

Figure 20: Avg. Location Review Score Per Zipcode

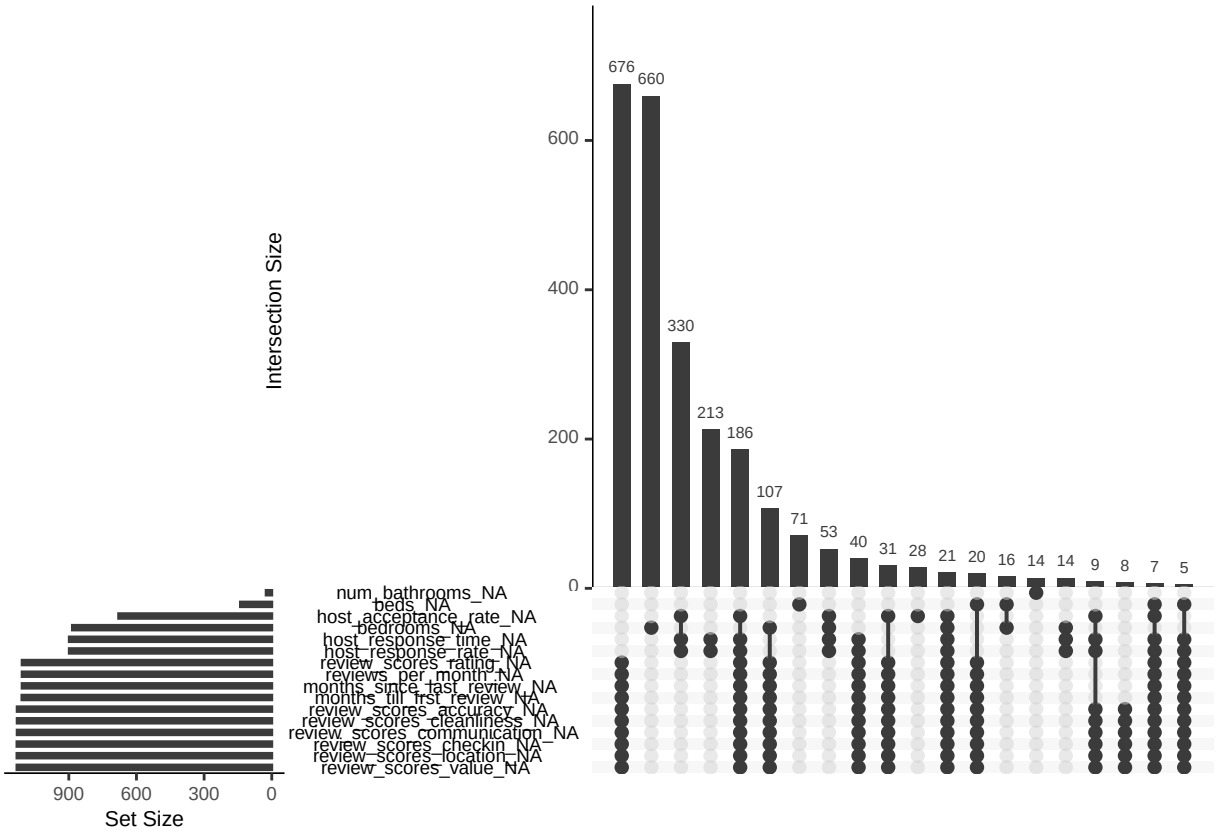

Figure 21: Missing Values and Missing data Pattern

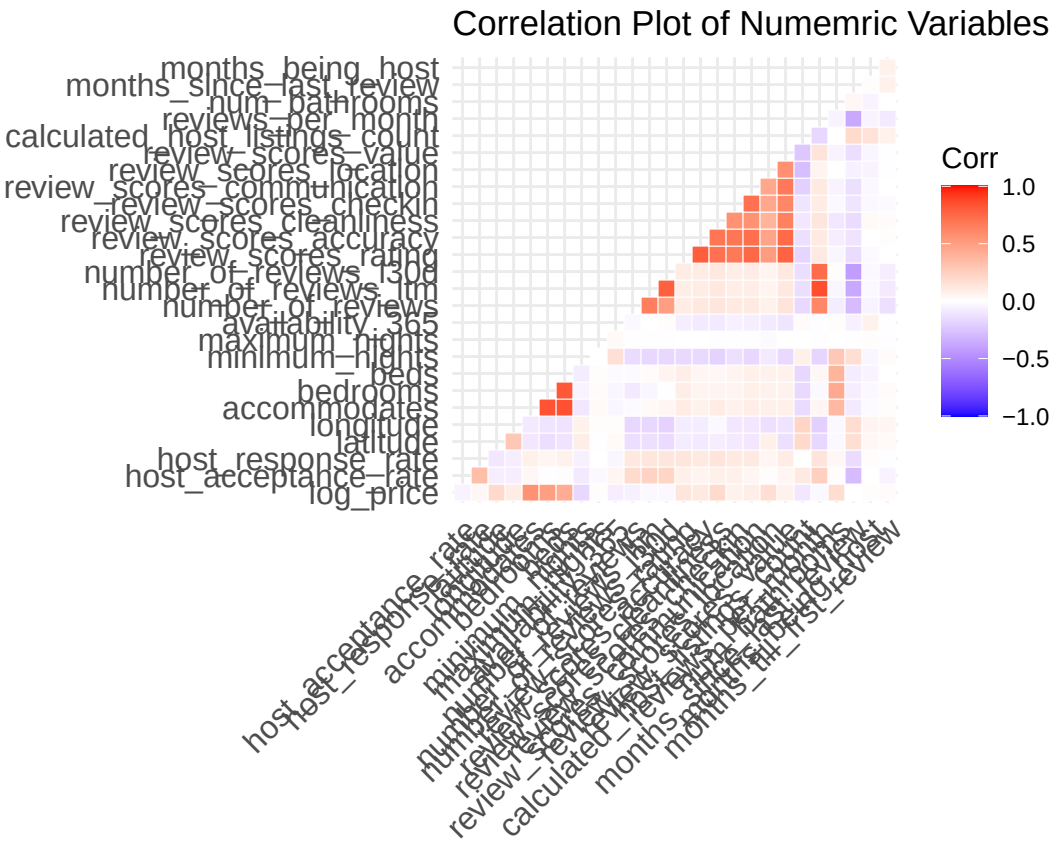

Figure 22: Correlation Heat-Map

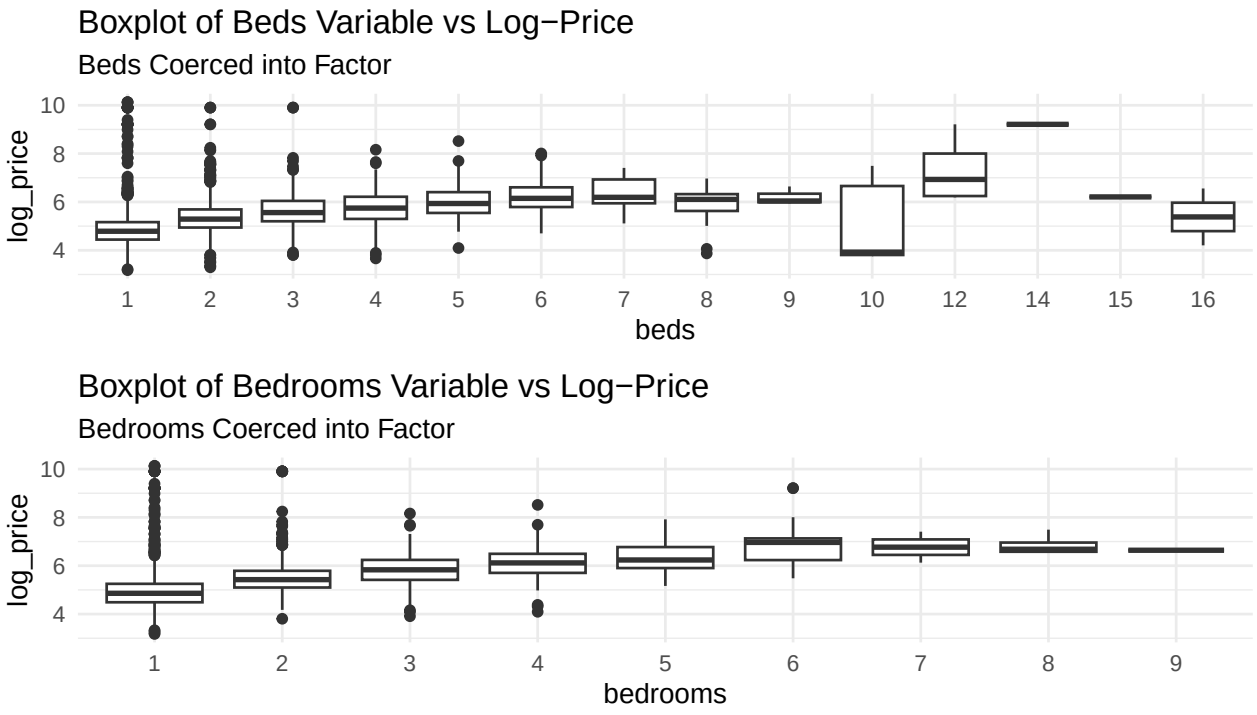

Figure 23: Beds and Bedrooms vs Log-Price

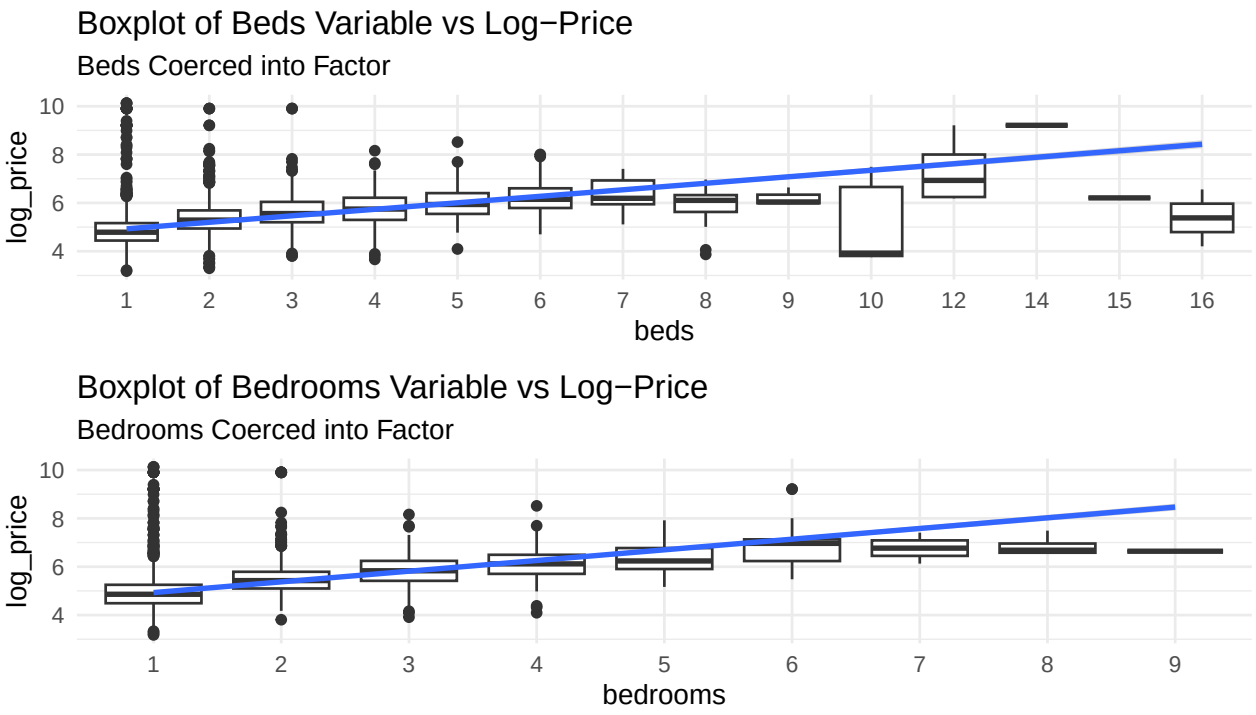

Figure 24: Beds and Bedrooms vs Log-Price

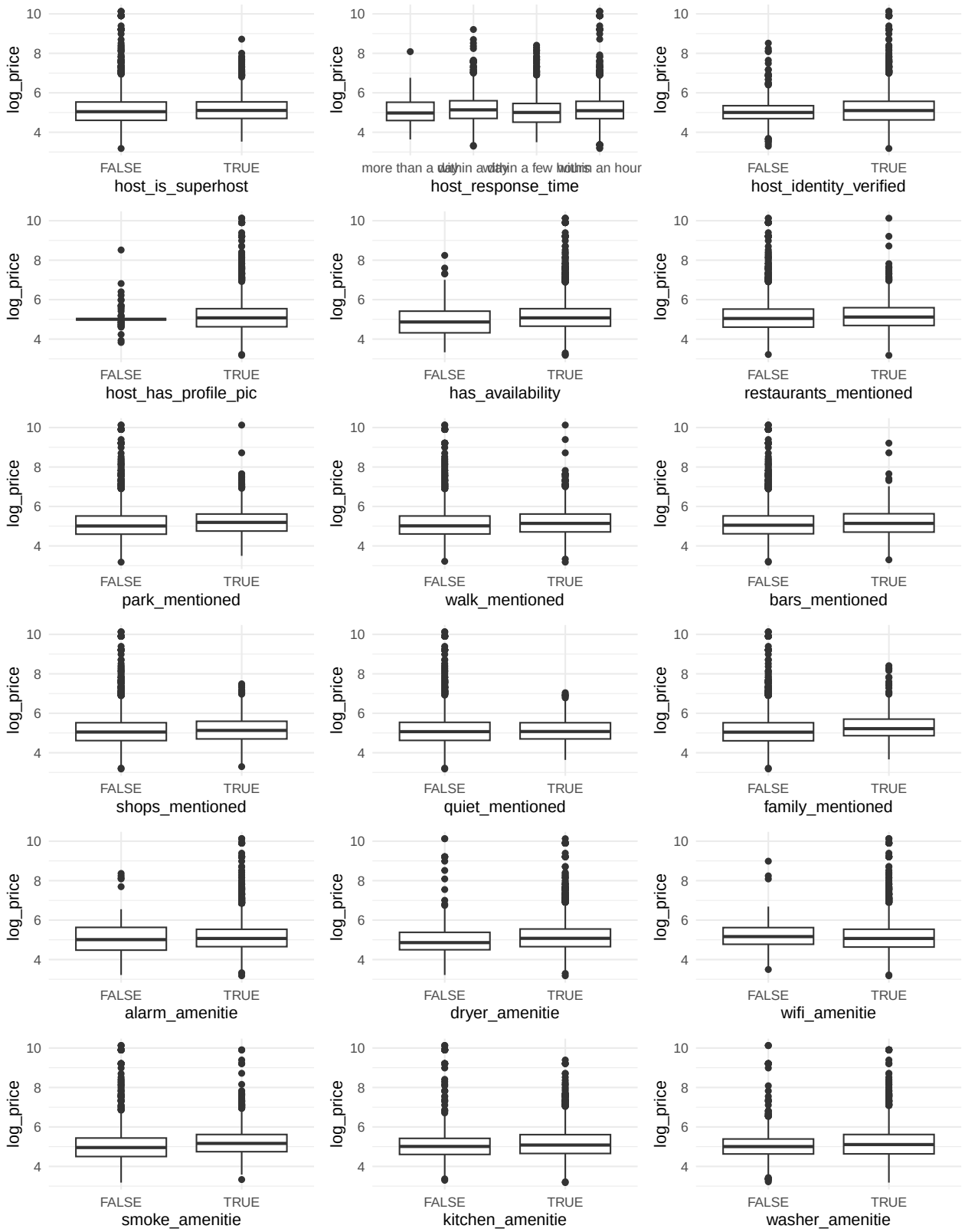

Figure 25: Catagorical Variables vs Log-Price

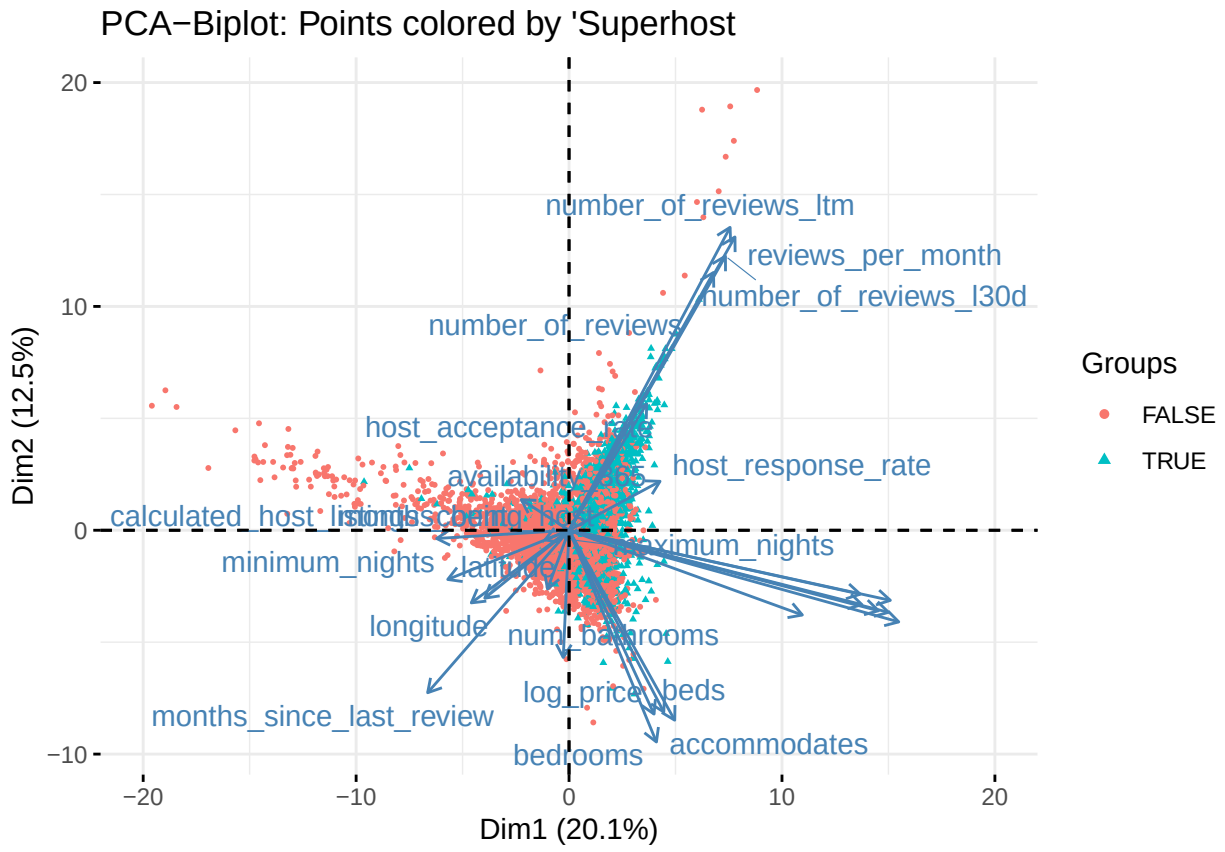

Figure 26: Biplot, where points are colored by 'Super-host': A host recomended by Airbnb

Code:

```
############################
### Math 533 Final
### Cleaning and imputation
### Seth Arreola
############################

# Packages
library(tidyverse)
library(lubridate)

# The data
listings_df <- read_csv("listings.csv")      # 18 features
listings2_df <- read_csv("listings 2.csv")   # 75 features
reviews_df <- read_csv("reviews.csv")        # 2 features
reviews2_df <- read_csv("reviews 2.csv")     # 6 features


## ---------------------- Cleaning Data ------------------------- ##
# The primary dataset that will be used is listings2_df
df <- listings2_df
glimpse(df)
dim(df)
```

```r
# we have some variables that will not be used and thus we drop
df <- df %>%
  select(-c(id, listing_url, scrape_id, source, picture_url, host_id, host_url,
            host_location,host_thumbnail_url, host_picture_url, host_verifications,
            neighbourhood_group_cleansed, bathrooms, calendar_updated, license,
            last_scraped, neighbourhood, calendar_last_scraped,
            host_neighbourhood, host_listings_count, host_total_listings_count,
            calculated_host_listings_count_entire_homes,
            calculated_host_listings_count_private_rooms, calculated_host_listings_count_shared_rooms,
            minimum_minimum_nights, maximum_minimum_nights,
            minimum_maximum_nights, maximum_maximum_nights, minimum_nights_avg_ntm,
            maximum_nights_avg_ntm, availability_30, availability_60, availability_90))

glimpse(df)

# A lot of the features are strings, this needs to be changed
# Price
df <- df %>%
  mutate(price = str_remove(price, "[$]")) %>%
  mutate(price = str_remove(price, "[,]")) %>%
  mutate(price = as.numeric(price))

glimpse(df)

# Response and acceptance rate
df <- df %>%
  mutate(host_response_rate = str_remove(host_response_rate, "[%]")) %>%
  mutate(host_response_rate = as.numeric(host_response_rate)) %>%
  mutate(host_response_rate = host_response_rate*0.01) %>%
  mutate(host_acceptance_rate = str_remove(host_acceptance_rate, "[%]")) %>%
  mutate(host_acceptance_rate = as.numeric(host_acceptance_rate)) %>%
  mutate(host_acceptance_rate = host_acceptance_rate*0.01)

glimpse(df)

# host_neighborhood and neighborhood_cleansed
unique(df$neighbourhood_cleansed)

df <- df %>%
  mutate(neighbourhood_cleansed = as.factor(neighbourhood_cleansed))

glimpse(df)

# Property type and room_type
unique(df$property_type)
unique(df$room_type)

df <- df %>%
  mutate(boat_property = str_detect(property_type, "Boat")) %>%
  mutate(castel_property = str_detect(property_type, "Castle")) %>%
  mutate(cabin_property = str_detect(property_type, "cabin")) %>%
  mutate(villa_property = str_detect(property_type, "villa")) %>%
  select(-property_type) %>%
```

```
  mutate(room_type = as.factor(room_type))

glimpse(df)

# Bathrooms
unique(df$bathrooms_text)
pat <- "(\\d)+"

df <- df %>%
  mutate(shared_bathroom = str_detect(bathrooms_text, "shared")) %>%
  mutate(num_bathrooms = as.numeric(str_extract(bathrooms_text, pat))) %>%
  select(-bathrooms_text)

glimpse(df)

# Host response time
unique(as.factor(df$host_response_time))

df <- df %>%
  mutate(host_response_time = case_when(
    host_response_time == "within an hour" ~ "within an hour",
    host_response_time == "within a day" ~ "within a day",
    host_response_time == "within a few hours" ~ "within a few hours",
    host_response_time == "a few days or more" ~ "more than a day")) %>%
  mutate(host_response_time = as.factor(host_response_time))

# OK we have a couple left but they are more complex

# date columns
df %>%
  select_if(function(x) inherits(x, 'Date'))

scraped_date <- unique(listings2_df$calendar_last_scraped)
df <- df %>%
  mutate(months_since_last_review = interval(last_review,scraped_date) %>%
           as.period() %>%
           month()) %>%
  mutate(months_being_host = interval(host_since,scraped_date) %>%
           as.period() %>%
           month()) %>%
  mutate(months_till_frst_review = interval(host_since, first_review) %>%
           as.period() %>%
           month()) %>%
  select(-c(host_since, last_review, first_review))

glimpse(df)


# Remaining character features
# str columns
df %>%
  select(name, description, neighborhood_overview, host_name, host_about, amenities)
```

```r
# name
unique(df$name)

library(wordcloud)
library(wordcloud2)
library(tm)

text <- unique(df$name)
docs <- Corpus(VectorSource(text))
docs <- docs %>%
  tm_map(removeNumbers) %>%
  tm_map(removePunctuation) %>%
  tm_map(stripWhitespace)
docs <- tm_map(docs, content_transformer(tolower))
docs <- tm_map(docs, removeWords, stopwords("english"))
dtm <- TermDocumentMatrix(docs)
matrix <- as.matrix(dtm)
words <- sort(rowSums(matrix),decreasing=TRUE)
df_words <- data.frame(word = names(words),freq=words)

set.seed(1234) # for reproducibility
wordcloud(words = df_words$word, freq = df_words$freq, min.freq = 1,
          max.words=200, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))

df <- df %>%
  select(-name)

glimpse(df)

# description
text <- unique(df$description)
docs <- Corpus(VectorSource(text))
docs <- docs %>%
  tm_map(removeNumbers) %>%
  tm_map(removePunctuation) %>%
  tm_map(stripWhitespace)
docs <- tm_map(docs, content_transformer(tolower))
docs <- tm_map(docs, removeWords, stopwords("english"))
dtm <- TermDocumentMatrix(docs)
matrix <- as.matrix(dtm)
words <- sort(rowSums(matrix),decreasing=TRUE)
df_words <- data.frame(word = names(words),freq=words)

set.seed(1234) # for reproducibility
wordcloud(words = df_words$word, freq = df_words$freq, min.freq = 1,
          max.words=200, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))

df <- df %>%
  select(-description)

glimpse(df)
```

```
# neighborhood_overview
text <- unique(df$neighborhood_overview)
docs <- Corpus(VectorSource(text))
docs <- docs %>%
  tm_map(removeNumbers) %>%
  tm_map(removePunctuation) %>%
  tm_map(stripWhitespace)
docs <- tm_map(docs, content_transformer(tolower))
docs <- tm_map(docs, removeWords, stopwords("english"))
dtm <- TermDocumentMatrix(docs)
matrix <- as.matrix(dtm)
words <- sort(rowSums(matrix),decreasing=TRUE)
df_words <- data.frame(word = names(words),freq=words)

set.seed(1234) # for reproducibility
wordcloud(words = df_words$word, freq = df_words$freq, min.freq = 1,
          max.words=200, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))

df_words %>%
  as_tibble() %>%
  slice(-c(4,5,6,8)) %>%
  slice(1:10)

df <- df %>%
  mutate(restaurants_mentioned =
           map_lgl(neighborhood_overview,
                   ~(str_detect(.x, "restaurants")|| str_detect(.x, "Restaurants")))) %>%
  mutate(park_mentioned =
           map_lgl(neighborhood_overview,
                   ~(str_detect(.x, "park")|| str_detect(.x, "Park")))) %>%
  mutate(walk_mentioned =
           map_lgl(neighborhood_overview,
                   ~(str_detect(.x, "walk")|| str_detect(.x, "Walk")))) %>%
  mutate(shops_mentioned =
           map_lgl(neighborhood_overview,
                   ~(str_detect(.x, "shops")|| str_detect(.x, "Shops")))) %>%
  mutate(bars_mentioned =
           map_lgl(neighborhood_overview,
                   ~(str_detect(.x, "bars")|| str_detect(.x, "Bars")))) %>%
  mutate(quiet_mentioned =
           map_lgl(neighborhood_overview,
                   ~(str_detect(.x, "quiet")|| str_detect(.x, "Quiet")))) %>%
  select(-neighborhood_overview)

glimpse(df)

# host about
text <- unique(df$host_about)
docs <- Corpus(VectorSource(text))
docs <- docs %>%
  tm_map(removeNumbers) %>%
  tm_map(removePunctuation) %>%
```

```
  tm_map(stripWhitespace)
docs <- tm_map(docs, content_transformer(tolower))
docs <- tm_map(docs, removeWords, stopwords("english"))
dtm <- TermDocumentMatrix(docs)
matrix <- as.matrix(dtm)
words <- sort(rowSums(matrix),decreasing=TRUE)
df_words <- data.frame(word = names(words),freq=words)

set.seed(1234) # for reproducibility
wordcloud(words = df_words$word, freq = df_words$freq, min.freq = 1,
          max.words=200, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))

df <- df %>%
  mutate(family_mentioned =
           map_lgl(host_about,
                   ~(str_detect(.x, "family")|| str_detect(.x, "Family")))) %>%
  select(-host_about)

glimpse(df)

# amenities
text <- unique(df$amenities)
docs <- Corpus(VectorSource(text))
docs <- docs %>%
  tm_map(removeNumbers) %>%
  tm_map(removePunctuation) %>%
  tm_map(stripWhitespace)
docs <- tm_map(docs, content_transformer(tolower))
docs <- tm_map(docs, removeWords, stopwords("english"))
dtm <- TermDocumentMatrix(docs)
matrix <- as.matrix(dtm)
words <- sort(rowSums(matrix),decreasing=TRUE)
df_words <- data.frame(word = names(words),freq=words)

set.seed(1234) # for reproducibility
wordcloud(words = df_words$word, freq = df_words$freq, min.freq = 1,
          max.words=200, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))

df_words %>%
  as_tibble() %>%
  slice(-c(3,4,5,6,10,12,13,14, 15,17,18)) %>%
  slice(-c(10,13)) %>%
  slice(1:20)

df <- df %>%
  mutate(alarm_amenitie =
           map_lgl(amenities, ~(str_detect(.x, "alarm")|| str_detect(.x, "Alarm")))) %>%
  mutate(dryer_amenitie =
           map_lgl(amenities, ~(str_detect(.x, "dryer")|| str_detect(.x, "Dryer")))) %>%
  mutate(wifi_amenitie =
           map_lgl(amenities, ~(str_detect(.x, "wifi")|| str_detect(.x, "Wifi")))) %>%
```

```
  mutate(smoke_amenitie =
           map_lgl(amenities, ~(str_detect(.x, "smoke")|| str_detect(.x, "Smoke")))) %>%
  mutate(kitchen_amenitie =
           map_lgl(amenities, ~(str_detect(.x, "kitchen")|| str_detect(.x, "Kitchen")))) %>%
  mutate(smoke_amenitie =
           map_lgl(amenities, ~(str_detect(.x, "coffee")|| str_detect(.x, "Coffee")))) %>%
  mutate(washer_amenitie =
           map_lgl(amenities, ~(str_detect(.x, "washer")|| str_detect(.x, "Washer")))) %>%
  select(-amenities)

glimpse(df)

# host name
df <- df %>% select(-host_name)

glimpse(df)

df <- df %>%
  mutate(host_is_superhost = as.factor(host_is_superhost)) %>%
  mutate(host_has_profile_pic = as.factor(host_has_profile_pic)) %>%
  mutate(host_identity_verified = as.factor(host_identity_verified)) %>%
  mutate(neighbourhood_cleansed = as.factor(neighbourhood_cleansed)) %>%
  mutate(has_availability = as.factor(has_availability)) %>%
  mutate(instant_bookable = as.factor(instant_bookable)) %>%
  mutate(boat_property = as.factor(boat_property)) %>%
  mutate(castel_property = as.factor(castel_property)) %>%
  mutate(cabin_property = as.factor(cabin_property)) %>%
  mutate(villa_property = as.factor(villa_property)) %>%
  mutate(shared_bathroom = as.factor(shared_bathroom)) %>%
  mutate(restaurants_mentioned = as.factor(restaurants_mentioned)) %>%
  mutate(park_mentioned = as.factor(park_mentioned)) %>%
  mutate(walk_mentioned = as.factor(walk_mentioned)) %>%
  mutate(shops_mentioned = as.factor(shops_mentioned)) %>%
  mutate(bars_mentioned = as.factor(bars_mentioned)) %>%
  mutate(quiet_mentioned = as.factor(quiet_mentioned)) %>%
  mutate(family_mentioned = as.factor(family_mentioned)) %>%
  mutate(alarm_amenitie = as.factor(alarm_amenitie)) %>%
  mutate(dryer_amenitie = as.factor(dryer_amenitie)) %>%
  mutate(wifi_amenitie = as.factor(wifi_amenitie)) %>%
  mutate(smoke_amenitie = as.factor(smoke_amenitie)) %>%
  mutate(kitchen_amenitie = as.factor(kitchen_amenitie)) %>%
  mutate(washer_amenitie = as.factor(washer_amenitie))

glimpse(df)

write_csv(df,"data_before_imp.csv")

### ------------------------ Missing value analysis ------------------------ ###
sapply(df, function(y) sum(length(which(is.na(y)))))

# we did create some missing values when looking at neighbourhood_overview
library(forcats)
df <- df %>%
```

```
  mutate(park_mentioned = fct_explicit_na(park_mentioned, "FALSE")) %>%
  mutate(restaurants_mentioned =  fct_explicit_na(restaurants_mentioned, "FALSE")) %>%
  mutate(walk_mentioned =  fct_explicit_na(walk_mentioned, "FALSE")) %>%
  mutate(shops_mentioned =  fct_explicit_na(shops_mentioned, "FALSE")) %>%
  mutate(bars_mentioned =  fct_explicit_na(bars_mentioned, "FALSE")) %>%
  mutate(quiet_mentioned =  fct_explicit_na(quiet_mentioned, "FALSE")) %>%
  mutate(family_mentioned =  fct_explicit_na(family_mentioned, "FALSE"))

glimpse(df)

library(visdat)
library(naniar)
vis_miss(df)
gg_miss_upset(df)
gg_miss_upset(df, nsets = 16, nintersects = 20)


library(missForest)
imputed_data <- missForest(data.frame(df),maxiter = 10, ntree = 100, verbose = TRUE)

df2 <- as_tibble(imputed_data$ximp)
glimpse(df2)
sapply(df2, function(y) sum(length(which(is.na(y)))))

glimpse(df2)

write_csv(df2,"listings_cleaned.csv")
```

```
############################
### Math 533 Final
### EDA
### Seth Arreola
############################

# Packages
library(tidyverse)

# The data
# listings_df <- read_csv("listings.csv")     # 18 features
listings2_df <- read_csv("listings 2.csv")  # 75 features
# reviews_df <- read_csv("reviews.csv")       # 2 features
# reviews2_df <- read_csv("reviews 2.csv")    # 6 features
listings_cleaned <- read_csv("listings_cleaned.csv")


df <- listings_cleaned

# make sure logicals are changed to factors
df <- df %>%
  mutate(room_type = as.factor(room_type)) %>%
  mutate(host_response_time = as.factor(host_response_time)) %>%
  mutate(host_is_superhost = as.factor(host_is_superhost)) %>%
  mutate(host_has_profile_pic = as.factor(host_has_profile_pic)) %>%
  mutate(host_identity_verified = as.factor(host_identity_verified)) %>%
```

```
  mutate(neighbourhood_cleansed = as.factor(neighbourhood_cleansed)) %>%
  mutate(has_availability = as.factor(has_availability)) %>%
  mutate(instant_bookable = as.factor(instant_bookable)) %>%
  mutate(boat_property = as.factor(boat_property)) %>%
  mutate(castel_property = as.factor(castel_property)) %>%
  mutate(cabin_property = as.factor(cabin_property)) %>%
  mutate(villa_property = as.factor(villa_property)) %>%
  mutate(shared_bathroom = as.factor(shared_bathroom)) %>%
  mutate(restaurants_mentioned = as.factor(restaurants_mentioned)) %>%
  mutate(park_mentioned = as.factor(park_mentioned)) %>%
  mutate(walk_mentioned = as.factor(walk_mentioned)) %>%
  mutate(shops_mentioned = as.factor(shops_mentioned)) %>%
  mutate(bars_mentioned = as.factor(bars_mentioned)) %>%
  mutate(quiet_mentioned = as.factor(quiet_mentioned)) %>%
  mutate(family_mentioned = as.factor(family_mentioned)) %>%
  mutate(alarm_amenitie = as.factor(alarm_amenitie)) %>%
  mutate(dryer_amenitie = as.factor(dryer_amenitie)) %>%
  mutate(wifi_amenitie = as.factor(wifi_amenitie)) %>%
  mutate(smoke_amenitie = as.factor(smoke_amenitie)) %>%
  mutate(kitchen_amenitie = as.factor(kitchen_amenitie)) %>%
  mutate(washer_amenitie = as.factor(washer_amenitie))

#### ---------------------------------------- Map plots! ----------------------------------------
# library(choroplethr)
# library(choroplethrMaps)
library(choroplethrZip)

zip_codes <- tibble(neighbourhood_cleansed = unique(df$neighbourhood_cleansed),
                    zip = c(94115, 94110, 94112, 94123, 94121, 94117, 94131, 94103,
                            94107, 94114, 94109, 94109, 94104, 94124, 94112, 94108,
                            94131, 94110, 94109, 94133, 94131, 94118, 94102, 94122,
                            94116, 94103, 94122, 94127, 94132, 94118, 94112, 94121,
                            94112, 94129, 94121, 94134))

df <- df %>%
  mutate(zipcode = case_when(
    neighbourhood_cleansed == unlist(zip_codes[1,1]) ~ zip_codes[1,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[2,1]) ~ zip_codes[2,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[3,1]) ~ zip_codes[3,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[4,1]) ~ zip_codes[4,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[5,1]) ~ zip_codes[5,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[6,1]) ~ zip_codes[6,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[7,1]) ~ zip_codes[7,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[8,1]) ~ zip_codes[8,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[9,1]) ~ zip_codes[9,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[10,1]) ~ zip_codes[10,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[11,1]) ~ zip_codes[11,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[12,1]) ~ zip_codes[12,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[13,1]) ~ zip_codes[13,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[14,1]) ~ zip_codes[14,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[15,1]) ~ zip_codes[15,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[16,1]) ~ zip_codes[16,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[17,1]) ~ zip_codes[17,2] %>% unlist(),
```

```
    neighbourhood_cleansed == unlist(zip_codes[18,1]) ~ zip_codes[18,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[19,1]) ~ zip_codes[19,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[20,1]) ~ zip_codes[20,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[21,1]) ~ zip_codes[21,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[22,1]) ~ zip_codes[22,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[23,1]) ~ zip_codes[23,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[24,1]) ~ zip_codes[24,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[25,1]) ~ zip_codes[25,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[26,1]) ~ zip_codes[26,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[27,1]) ~ zip_codes[27,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[28,1]) ~ zip_codes[28,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[29,1]) ~ zip_codes[29,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[30,1]) ~ zip_codes[30,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[31,1]) ~ zip_codes[31,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[32,1]) ~ zip_codes[32,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[33,1]) ~ zip_codes[33,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[34,1]) ~ zip_codes[34,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[35,1]) ~ zip_codes[35,2] %>% unlist(),
    neighbourhood_cleansed == unlist(zip_codes[36,1]) ~ zip_codes[36,2] %>% unlist()))

zipReviews <- df %>%
  group_by(zipcode) %>%
  summarise(avg_loc_review = mean(review_scores_location))

temp <- tibble(zipcode = c(94158, 94105, 94111),
       avg_loc_review = c(mean(4.84,4.82,4.324077), mean(4.843967,4.678884,4.818172),
                          mean(4.843967,4.678884,4.818172,4.876755,4.874494)))

zipReviews <- bind_rows(zipReviews, temp)

colnames(zipReviews) <- c("region","value")
zipReviews$region <- as.character(zipReviews$region)

zip_choropleth(zipReviews,county_zoom = 6075) +
  ggtitle("Which area is the best?", subtitle = "Map showing Average Location Score by Area") +
  #geom_point(data = df2, aes(longitude, latitude, group = zipcode)) +
  theme(plot.title = element_text(face = "bold")) +
  theme(plot.subtitle = element_text(face = "bold", color = "grey35")) +
  theme(plot.caption = element_text(color = "grey68")) +
  scale_color_gradient(low="#d3cbcb", high="#852eaa") +
  scale_fill_brewer("Location Review Score",palette=3)



zipPrices <- df %>%
  group_by(zipcode) %>%
  summarise(avg_price = mean(price))

temp <- tibble(zipcode = c(94158, 94105, 94111),
               avg_price = c(mean(269.9007,241.9355, 144.0568),
                             mean(269.9007, 241.9355, 823.5375),
                             mean(416.3704, 454.0284)))
```

```r
zipPrices <- bind_rows(zipPrices, temp)

colnames(zipPrices) <- c("region","value")
zipPrices$region <- as.character(zipPrices$region)


zip_choropleth(zipPrices, county_zoom = 6075) +
  ggtitle("Which area is expensive?", subtitle = "Map showing Average Price by Area") +
  theme(plot.title = element_text(face = "bold")) +
  theme(plot.subtitle = element_text(face = "bold", color = "grey35")) +
  theme(plot.caption = element_text(color = "grey68")) +
  scale_color_gradient(low="#d3cbcb", high="#852eaa") +
  scale_fill_brewer("Average Price",palette=4)


### --------------------------- The response: Price ---------------------- ###

# we should spice this plot up
df %>%
  ggplot(aes(price)) +
  geom_histogram(bins = 40) +
  labs(title = "Histogram of Price of Listings per Night",
       x = "Price of Listings per Night") +
  theme_minimal()

# which observations have a price > 10000 for one night
df %>%
  filter(price > 10000) %>%
  glimpse()

# 19 observations

# how about price > 20000
df %>%
  filter(price > 20000) %>%
  glimpse()
# three observations

# out of curiosity, what listings are these?
listings2_df %>%
  mutate(price = str_remove(price, "[$]")) %>%
  mutate(price = str_remove(price, "[,]")) %>%
  mutate(price = as.numeric(price)) %>%
  filter(price > 20000) %>%
  select(price, id, listing_url)

# for visualizations.. lets do a log-transformation
# this helps for skewed variables
# to see hoe price relates to the other features lets look on a log scale
df <- df %>%
  mutate(log_price = log(price))

df %>%
```

```
  ggplot(aes(log_price)) +
  geom_histogram()

df %>%
  summarise(min(log_price), max(log_price))
df %>%
  mutate(index = seq(1,dim(df)[1])) %>%
  filter(log_price == min(log_price)) %>%
  select(index)
# 4658 4758 5227

df <- df %>%
  slice(-c(4658,4758,5227))

# we should spice this plot up
df %>%
  ggplot(aes(log_price)) +
  geom_histogram()

### ---------------------------- Correlations analysis -------------------------- ###
library(ggcorrplot)
glimpse(df)

df_numeric <- df %>%
  select(log_price, host_acceptance_rate, host_response_rate,
         latitude, longitude, accommodates,bedrooms,
         beds, minimum_nights, maximum_nights, availability_365,
         number_of_reviews, number_of_reviews_ltm, number_of_reviews_l30d,
         review_scores_rating, review_scores_accuracy,
         review_scores_cleanliness, review_scores_checkin,
         review_scores_checkin, review_scores_communication,
         review_scores_location, review_scores_value,
         calculated_host_listings_count, reviews_per_month,
         num_bathrooms, months_since_last_review, months_being_host,
         months_till_frst_review)

corr <- cor(df_numeric)
p_mat <- cor_pmat(df_numeric)
ggcorrplot(corr, type = "lower", outline.color = "white", p.mat = p_mat)

# lets take a look at the variables that are most correlated with log_price
# all of the following plots need to be spiced up

# log price vs accommodates
df %>%
  ggplot(aes(accommodates)) +
  geom_histogram()

df %>%
  ggplot(aes(accommodates, log_price)) +
  geom_point() +
  geom_smooth(method = "lm")
```

```r
df %>%
  mutate(accommodates = round(accommodates)) %>%
  mutate(accommodates = as.factor(accommodates)) %>%
  ggplot(aes(accommodates, log_price)) +
  geom_boxplot()

# log price vs beds
df %>%
  ggplot(aes(beds)) +
  geom_histogram()

df %>%
  ggplot(aes(beds, log_price)) +
  geom_point() +
  geom_smooth(method = "lm")

df %>%
  mutate(beds = round(beds)) %>%
  mutate(beds = as.factor(beds)) %>%
  ggplot(aes(beds, log_price)) +
  geom_boxplot()

df %>%
  filter(beds > 9)
# only 55

# log price vs bedrooms
df %>%
  ggplot(aes(bedrooms)) +
  geom_histogram()

df %>%
  ggplot(aes(bedrooms, log_price)) +
  geom_point() +
  geom_smooth(method = "lm")

df %>%
  mutate(bedrooms = round(bedrooms)) %>%
  mutate(bedrooms = as.factor(bedrooms)) %>%
  ggplot(aes(bedrooms, log_price)) +
  geom_boxplot()

# log price vs minimum_number_of nigts
df %>%
  ggplot(aes(minimum_nights)) +
  geom_histogram()

df %>%
  ggplot(aes(minimum_nights, log_price)) +
  geom_point() +
  geom_smooth(method = "lm")

df %>%
```

```
  mutate(minimum_nights = round(minimum_nights)) %>%
  mutate(minimum_nights = as.factor(minimum_nights)) %>%
  mutate(minimum_nights = fct_collapse(minimum_nights,
                                       '1' = '1',
                                       '2' = '2',
                                       '3' = '3',
                                       '4' = '4',
                                       '5' = '5',
                                       '6' = '6',
                                       '7' = '7',
                                       '8' = '8',
                                       '10-30' = c('10','12','13','14','15','20','21','27','28','30'),
                                       '31-50' = c('31','32','35','45','50'),
                                       '51-100' = c('60','62','75','80','85','90'),
                                       '120-365' = c('120','150','179','180','183','190','192',
                                                     '200','214','240','280','300','356','360',
                                                     '356','360','364','365'),
                                       '>365' = c('366','500','1000','1125'))) %>%
  ggplot(aes(minimum_nights, log_price)) +
  geom_boxplot()

# how many observations are min_nights 10 or above
df %>%
  mutate(minimum_nights = round(minimum_nights)) %>%
  mutate(minimum_nights = as.factor(minimum_nights)) %>%
  mutate(minimum_nights = fct_collapse(minimum_nights,
                                       '1' = '1',
                                       '2' = '2',
                                       '3' = '3',
                                       '4' = '4',
                                       '5' = '5',
                                       '6' = '6',
                                       '7' = '7',
                                       '8' = '8',
                                       '10-30' = c('10','12','13','14','15','20','21','27','28','30'),
                                       '31-50' = c('31','32','35','45','50'),
                                       '51-100' = c('60','62','75','80','85','90'),
                                       '120-365' = c('120','150','179','180','183','190','192',
                                                     '200','214','240','280','300','356','360',
                                                     '356','360','364','365'),
                                       '>365' = c('366','500','1000','1125'))) %>%
  count(minimum_nights)

### ------------------------------------ PCA ---------------------------------------- ###
# Perform PCA on the predictor space
pca <- df_numeric %>%
  prcomp(scale = TRUE)

tibble(Variance  = (pca$sdev)^2, PC = seq(1:27)) %>%
  ggplot(aes(PC, Variance)) +
  geom_point() +
  geom_line() +
  scale_x_continuous(breaks = scales::pretty_breaks(n = 27)) +
```

```
  labs(title = "Variance of Principal Componets")

library(factoextra) # package that contains biplot

# Biplot
fviz_pca(
  pca,
  axes = c(1, 2),              # choose two PC's to plot
  geom = c("point"),           # plot points, text labels, or both
  pointsize = 0.7,             # point size for states
  col.ind = "lightgrey",       # point and label color
  col.var = "contrib",         # use variable contributions to PC as legend
  repel = TRUE,
  gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),  # color for variable contribution
  addEllipses = TRUE) +
  xlim(-20,20)

fviz_pca(
  pca,
  axes = c(1, 3),              # choose two PC's to plot
  geom = c("point"),           # plot points, text labels, or both
  pointsize = 0.7,             # point size for states
  col.ind = "lightgrey",       # point and label color
  col.var = "contrib",         # use variable contributions to PC as legend
  repel = TRUE,
  gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),  # color for variable contribution
  addEllipses = TRUE) +
  xlim(-20,20)


fviz_pca(
  pca,
  axes = c(1, 2),              # choose two PC's to plot
  geom = c("point"),           # plot points, text labels, or both
  pointsize = 0.7,             # point size for states
  repel = TRUE,
  #col.ind = "lightgrey",       # point and label color
  #col.var = "contrib",         # use variable contributions to PC as legend
  #gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),  # color for variable contribution
  habillage = df$host_is_superhost) +
  xlim(-20,20)

fviz_pca(
  pca,
  axes = c(1, 2),              # choose two PC's to plot
  geom = c("point"),           # plot points, text labels, or both
  pointsize = 0.7,             # point size for states
  col.ind = df$log_price,      # point and label color
  col.var = "contrib",         # use variable contributions to PC as legend
  gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),  # color for variable contribution
  addEllipses = TRUE) +
  xlim(-20,20)
```

```
fviz_pca_biplot(pca, lable = "none") +
  geom_point(aes(colour = factor(df$host_is_superhost))) +
  guides(shape = guide_legend(title = "shape"),
         colour = guide_legend(title = "color"))

library(pca3d)
pca3d(pca, components = 1:3, group = as.factor(df$host_is_superhost), biplot = TRUE)
### ------------------------------ price vs categorical variables -------------------- ###
glimpse(df)


df %>%
  ggplot(aes(host_is_superhost, log_price)) +
  geom_boxplot()

df %>%
  ggplot(aes(host_response_time, log_price)) +
  geom_boxplot()


df %>%
  ggplot(aes(reorder(neighbourhood_cleansed, log_price), log_price)) +
  geom_boxplot() +
  coord_flip()

df %>%
  filter(maximum_nights < 2000) %>%
  ggplot(aes(maximum_nights, log_price)) +
  geom_point()

df %>%
  ggplot(aes(host_identity_verified, log_price)) +
  geom_boxplot()

df %>%
  ggplot(aes(host_has_profile_pic, log_price)) +
  geom_boxplot()

df %>%
  ggplot(aes(has_availability, log_price)) +
  geom_boxplot()

# this one might be interesting
df %>%
  ggplot(aes(shared_bathroom, log_price)) +
  geom_boxplot()

df %>%
  ggplot(aes(restaurants_mentioned, log_price)) +
  geom_boxplot()
df %>%
  ggplot(aes(park_mentioned, log_price)) +
  geom_boxplot()
```

```r
df %>%
  ggplot(aes(walk_mentioned, log_price)) +
  geom_boxplot()

df %>%
  ggplot(aes(bars_mentioned, log_price)) +
  geom_boxplot()

df %>%
  ggplot(aes(shops_mentioned, log_price)) +
  geom_boxplot()

df %>%
  ggplot(aes(quiet_mentioned, log_price)) +
  geom_boxplot()

df %>%
  ggplot(aes(family_mentioned, log_price)) +
  geom_boxplot()

df %>%
  ggplot(aes(alarm_amenitie, log_price)) +
  geom_boxplot()
df %>%
  ggplot(aes(dryer_amenitie, log_price)) +
  geom_boxplot()

df %>%
  ggplot(aes(wifi_amenitie, log_price)) +
  geom_boxplot()

df %>%
  ggplot(aes(smoke_amenitie, log_price)) +
  geom_boxplot()

df %>%
  ggplot(aes(kitchen_amenitie, log_price)) +
  geom_boxplot()

df %>%
  ggplot(aes(washer_amenitie, log_price)) +
  geom_boxplot()



### ---------------------- clustering ----------------------------- ###
library(clustMixType)
df <- df %>%
  mutate(zipcode = as.factor(zipcode))
set.seed(123)
# Compute and plot wss for k = 2 to k = 15.
set.seed(3)
k.max <- 20
```

```
wss <- sapply(1:k.max,
              function(k){kproto(df, k)$tot.withinss})
wss
plot(1:k.max, wss,
     type="b", pch = 19, frame = FALSE,
     xlab="Number of clusters K",
     ylab="Total within-clusters sum of squares")


df_clusters <- kproto(df, 12)


df_clusters$centers

summary(df_clusters)

clust_fit_df <- factor(df_clusters$cluster, order =  TRUE, levels = c(1:12))
fit <- data.frame(df, clust_fit_df)
result_df <- df_clusters$centers
Member <- df_clusters$size
result <- data.frame(Member, result_df)
result

ggplot(fit, aes(beds, log_price, color = clust_fit_df)) +
  geom_point()

df %>%
  mutate(beds = round(beds)) %>%
  mutate(beds = as.factor(beds)) %>%
  ggplot(aes(beds, log_price, color = clust_fit_df)) +
  geom_boxplot()

df %>%
  mutate(accommodates = round(accommodates)) %>%
  mutate(accommodates = as.factor(accommodates)) %>%
  ggplot(aes(accommodates, log_price, color = clust_fit_df)) +
  geom_boxplot()


fit %>%
  mutate(bedrooms = round(bedrooms)) %>%
  mutate(bedrooms = as.factor(bedrooms)) %>%
  ggplot(aes(bedrooms, log_price, color = clust_fit_df)) +
  geom_boxplot()


ggplot(fit, aes(host_is_superhost, log_price, color = clust_fit_df)) +
  geom_boxplot()

fit %>%
  ggplot(aes(reorder(neighbourhood_cleansed, log_price), log_price, color = clust_fit_df)) +
  geom_boxplot() +
  coord_flip()
```

```
fit %>%
  ggplot(aes(review_scores_location, log_price, color = clust_fit_df)) +
  geom_point()

fit %>%
  ggplot(aes(host_response_time, log_price, color = clust_fit_df)) +
  geom_boxplot()

fit %>%
  ggplot(aes(log_price, fill = clust_fit_df)) +
  geom_histogram()

####
df_pca <- as.data.frame(pca$x)
k.max <- 15
wss <- sapply(1:k.max,
              function(k){kmeans(df_pca, k)$tot.withinss})
wss
plot(1:k.max, wss,
     type="b", pch = 19, frame = FALSE,
     xlab="Number of clusters K",
     ylab="Total within-clusters sum of squares")


df_clusters2 <- kmeans(df_pca, 4)

df_clusters2$centers

summary(df_clusters2)

clust_fit_df <- factor(df_clusters2$cluster, order =  TRUE, levels = c(1:4))
pca_clust_fit <- data.frame(df_pca, clust_fit_df)

fviz_cluster(df_clusters2, df_pca)


km_clust <- kmeans(df_numeric,3)
fviz_cluster(km_clust, df_numeric)
```

```
############################
### Math 533 Final
### Modeling
### Seth Arreola
############################

library(tidyverse)
library(randomForest)
library(glmnet)
library(tidymodels)
library(vip)
library(keras)
library(reticulate)
library(tensorflow)
```

```r
listings_cleaned <- read_csv("listings_cleaned.csv")
df_all <- listings_cleaned

# make sure logical's are changed to factors
df_all <- df_all %>%
  mutate(room_type = as.factor(room_type)) %>%
  mutate(host_response_time = as.factor(host_response_time)) %>%
  mutate(host_is_superhost = as.factor(host_is_superhost)) %>%
  mutate(host_has_profile_pic = as.factor(host_has_profile_pic)) %>%
  mutate(host_identity_verified = as.factor(host_identity_verified)) %>%
  mutate(neighbourhood_cleansed = as.factor(neighbourhood_cleansed)) %>%
  mutate(has_availability = as.factor(has_availability)) %>%
  mutate(instant_bookable = as.factor(instant_bookable)) %>%
  mutate(boat_property = as.factor(boat_property)) %>%
  mutate(castel_property = as.factor(castel_property)) %>%
  mutate(cabin_property = as.factor(cabin_property)) %>%
  mutate(villa_property = as.factor(villa_property)) %>%
  mutate(shared_bathroom = as.factor(shared_bathroom)) %>%
  mutate(restaurants_mentioned = as.factor(restaurants_mentioned)) %>%
  mutate(park_mentioned = as.factor(park_mentioned)) %>%
  mutate(walk_mentioned = as.factor(walk_mentioned)) %>%
  mutate(shops_mentioned = as.factor(shops_mentioned)) %>%
  mutate(bars_mentioned = as.factor(bars_mentioned)) %>%
  mutate(quiet_mentioned = as.factor(quiet_mentioned)) %>%
  mutate(family_mentioned = as.factor(family_mentioned)) %>%
  mutate(alarm_amenitie = as.factor(alarm_amenitie)) %>%
  mutate(dryer_amenitie = as.factor(dryer_amenitie)) %>%
  mutate(wifi_amenitie = as.factor(wifi_amenitie)) %>%
  mutate(smoke_amenitie = as.factor(smoke_amenitie)) %>%
  mutate(kitchen_amenitie = as.factor(kitchen_amenitie)) %>%
  mutate(washer_amenitie = as.factor(washer_amenitie))

df_all <- df_all %>%
  mutate(log_price = log(price)) %>%
  select(-price) %>%
  slice(-c(4658,4758,5227))

df_split <- initial_split(df_all, prop = 6/7)
df <- training(df_split)
val_df <- testing(df_split)

### ---------------------------- Tuning Models ---------------------------- ###
################################################################################

#### --- Lasso Regression:
##########################

# Lambda values to tune over
lam <- seq(0,2,length = 50)

# Option 1)
predictors <- model.matrix(log_price~., df)[,-1]
cv.out = cv.glmnet(predictors, df$log_price, alpha = 1, lambda = lam)
```

```r
glmnets_best_lam <- cv.out$lambda.min
glmnets_best_lam

# Option 2) write up the cross validation ourselves
tune_lasso <-  function(df,lam,f){
  ## Input description:
  # df is a dataframe w/ standardized predictors
  # lam is a vector of testable values for lambda
  # f is the number of folds in k-fold CV

  ## Create k-folds (k=f)
  n <- dim(df)[1]
  n.adj <- floor(n/f)*f # we cant get an even f folds
  folds <- sample(1:n, n.adj, replace = FALSE) %>%
    matrix(nrow = f)
  mse_results <- tibble(.rows = length(lam))

  ## Iterate CV through each fold
  for(k in 1:f){
    # Test/Train for this fold
    train <- df[-folds[k,],]
    test <- df[folds[k,],]

    # Reformat for glmnet()
    train.x <- model.matrix(log_price  ~., data = train)[,-1]
    train.y <- train$log_price
    test.x <- model.matrix(log_price  ~., data = test)[,-1]
    test.y <- test$log_price

    # For this fold, MSE is computed
    mse_results[,k] <- lam %>%
      as_tibble() %>%
      rename(lam = value) %>%
      group_by(lam) %>%
      mutate(model = map(lam, ~ glmnet(train.x, train.y, lambda = .x, alpha = 1))) %>%
      mutate(pred = map(model, ~ predict(.x, newx = test.x, s = .x$lambda))) %>%
      mutate(mse = map(pred, ~ mean((test.y - .x)^2))) %>%
      select(lam, mse) %>%
      unnest(mse) %>%
      ungroup(lam) %>%
      select(mse) %>%
      as_vector()
  }

  # Summarize CV
  cv_results <- mse_results %>%
    mutate(mean_mse = rowMeans(across())) %>%
    mutate(lambda = lam) %>%
    select(lambda, mean_mse)

  return(cv_results) # Return CV results
}
```

```r
# Run our function
cv_lasso <- tune_lasso(df,lam,10)

# Save best CV values
best_lasso <- cv_lasso %>%
  filter(mean_mse == min(mean_mse)) %>%
  as.data.frame()

cv_lasso %>%
  ggplot() +
  geom_point(aes(lambda, sqrt(mean_mse))) +
  geom_line(aes(lambda, sqrt(mean_mse))) +
  geom_vline(aes(xintercept = best_lasso[1,1]), lty = "dashed", color = "red") +
  xlim(0,0.4) +
  labs(title = "Lasso 10-Fold Cross Validated MSE for different Lambda's",
       x = "Lambda",
       y = "RMSE")

cv_lasso %>%
  filter(mean_mse == min(mean_mse)) %>%
  mutate(rmse = sqrt(mean_mse))



#####  ----- Random forest:
############################

## tidy models
folds <- vfold_cv(df, v = 10)

rf_spec <- rand_forest(
  mtry = tune(), min_n = tune()) %>%
  set_engine("ranger") %>%
  set_mode("regression")

tree_rec <- recipe(log_price ~ ., data = df)

# Ranfom forest workflow
rf_Wf <- workflow() %>%
  add_recipe(tree_rec) %>%
  add_model(rf_spec)

rf_grid <-
  grid_latin_hypercube(
    min_n(),
    mtry(range = c(4, 20)),
    size = 80)

rf_tune_res <-
  rf_Wf %>%
  tune_grid(
    resamples = folds, grid = rf_grid,
    metrics = metric_set(rmse, mae, rsq)
```

```
  )

rf_tune_res %>%
  collect_metrics() %>%
  filter(.metric == "rmse")

rf_rmse_res <- rf_tune_res %>%
  collect_metrics() %>%
  filter(.metric == "rmse")

write_csv(rf_rmse_res, "rf_tume_results.csv")

select_best(rf_tune_res, "rmse")

rf_best_rmse <- select_best(rf_tune_res, "rmse")

rf_tune_res %>%
  collect_metrics() %>%
  filter(mtry == best_rmse$mtry,
         min_n == best_rmse$min_n)

final_rf <- finalize_model(
  rf_spec,
  best_rsq
)

final_rf

final_rf %>%
  set_engine("ranger",importance = "impurity") %>%
  fit(log_price ~ .,
      data = df
  ) %>%
  vip(geom = "point")


# Mis-Classification function
misclass_rf <- function(p, train, test){
  mtry_val <- p[1]
  minsplit_val <- p[2]
  cp_val <- p[3]

  mcr <- train %>%
    randomForest(diagnosis ~.,
                 data = .,
                 ntree = 500,
                 mtry = mtry_val,
                 control = rpart.control(minsplit=minsplit_val,cp=cp_val)) %>%
    predict(newdata = test) %>%
    as_tibble() %>%
    mutate(truth  = test$diagnosis) %>%
    summarise(MCR = mean(value != truth)) %>%
    unlist()
```

```
  return(mcr)
}

f = 10
# Create train/testing folds
n <- dim(df)[1]
n.adj <- floor(n/f)*f # we cant get an even f folds
folds <- sample(1:n, n.adj, replace = FALSE) %>% matrix(nrow = f)
# Initialize an empty matrix to save misclassification rate
# Rage of values fr each parameter
minsplit_values <- c(20, 45, 80)
cp_values <- c(0.005, 0.01, 0.04)
mtry_values <- c(4, 7.5, 12, 20)
# all the unique combinations of our parameters
parameters <- expand_grid(minsplit_values, cp_values)

mis_class <- matrix(0,dim(parameters)[1],10)

# Cross-Validation for tuning bagging
for(k in 1:10){
  # For each of the folds we will train all possible models on the training data
  # and we will compute the mis-classification rate on the test data
  train <- df[-folds[k,],]
  test <- df[folds[k,],]

  # fit each of the models and calc MCR
  for(i in 1:dim(parameters)[1]){
    p <- c(55,
           parameters$minsplit_values[i],
           parameters$cp_values[i])
    mis_class[i,k] <- misclass_rf(p,train,test)
  }
  print(c("Fold completed:", k))
}
# end of temp code




#### ----- XGBoost:
###################

xgb_spec <- boost_tree(
  trees = 1000,
  tree_depth = tune(),
  min_n = tune(),
  loss_reduction = tune(),
  sample_size = tune(),
  mtry = tune(),
  learn_rate = tune()) %>%
  set_engine("xgboost", objective = "reg:squarederror") %>%
  set_mode("regression")
```

```
xgb_grid <- grid_latin_hypercube(
  tree_depth(),
  min_n(),
  loss_reduction(),
  sample_size = sample_prop(),
  finalize(mtry(), df),
  learn_rate(),
  size = 40)

xgb_rec <- recipe(log_price ~ ., data = as.data.frame(df)) %>%
  step_dummy(all_nominal())

xgb_wf <- workflow() %>%
  add_recipe(xgb_rec) %>%
  add_model(xgb_spec)

xgb_tune_res <- xgb_wf %>%
  tune_grid(resamples = folds, grid = xgb_grid,
            metrics = metric_set(rmse, mae))


xgb_tune_res %>%
  collect_metrics() %>%
  filter(.metric == "rmse")

xgb_rmse_res <- xgb_tune_res %>%
  collect_metrics() %>%
  filter(.metric == "rmse")

write_csv(xgb_rmse_res, "xgboost_tume_results.csv")

select_best(xgb_tune_res, "rmse")

xgb_best_mse <- select_best(xgb_tune_res, "rmse")

xgb_tune_res %>%
  collect_metrics() %>%
  filter(mtry == best_rsq$mtry,
         min_n == best_rsq$min_n)

final_rf <- finalize_model(
  rf_spec,
  best_rsq
)

final_rf

final_rf %>%
  set_engine("ranger",importance = "impurity") %>%
  fit(log_price ~ .,
      data = df
  ) %>%
  vip(geom = "point")
```

```
### ------ Neural Network:
#########################

X <- model.matrix(log_price ~ . - 1, data = df) # (-1 omits the intercept)
Y <- df$log_price

modelnn <- keras_model_sequential() %>%
  layer_dense(units = 60, activation = "relu", input_shape = ncol(X)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 40, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = "relu") %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 1)

modelnn <- keras_model_sequential() %>%
  layer_dense(units = 60, activation = "relu", input_shape = ncol(X)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 1)

modelnn %>%
  compile(loss = "mse",
          optimizer = optimizer_rmsprop(),
          metrics = list("mse"))

testid <- sample(1:nrow(df), 1000,replace = FALSE)
history <- modelnn %>%
  fit(X[-testid,], Y[-testid],
      epochs = 100, batch_size = 800,
      validation_data = list(X[testid,], Y[testid]))

history <- modelnn %>%
  fit(X[-testid,], Y[-testid],
      epochs = 100, batch_size = 800)

plot(history)

npred <- predict(modelnn, X[testid, ])
sqrt(mean((Y[testid] - npred)^2))

#####
# Function to perform k-fold CV
nn_CV <-  function(df,x,y,f, eph, bts, nn_mod){
  ## Create k-folds (k=f)
  n <- dim(df)[1]
  n.adj <- floor(n/f)*f # we cant get an even f folds
  folds <- sample(1:n, n.adj, replace = FALSE) %>%
    matrix(nrow = f)
  rmse <- numeric(length = f)

  ## Iterate CV through each fold
```

```r
  for(k in 1:f){
    # Test/Train for this fold
    x_train <- x[-folds[k,],]
    x_test <- x[folds[k,],]
    y_train <- y[-folds[k,]]
    y_test <- y[folds[k,]]

    # Model is defined outside of function

    # Train model;
    history <- nn_mod %>%
      fit(x_train, y_train,
          epochs = eph, batch_size = bts,
          validation_data = list(x_test, y_test))

    # Compute rmse
    y_pred <- predict(nn_mod, x_test)
    rmse[k] <- sqrt(mean((y_test - y_pred)^2))

  }

  # Summarize CV
  cv_results <- mean(rmse)

  return(cv_results) # Return CV results
}

# Prep data for all mods
x <- model.matrix(log_price ~ . - 1, data = df) # (-1 omits the intercept)
y <- df$log_price

# Lets make three different neral network models
nn_mod1 <- keras_model_sequential() %>%
  layer_dense(units = 60, activation = "relu", input_shape = ncol(x)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 1)

nn_mod1 %>%
  compile(loss = "mse",
          optimizer = optimizer_rmsprop(),
          metrics = list("mse"))


f = 10
eph = 300
bts = 40
nn1_results <- nn_CV(df,x,y,f, eph, bts, nn_mod1)
nn1_results

nn_mod3 <- keras_model_sequential() %>%
  layer_dense(units = 70, activation = "relu", input_shape = ncol(x)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 40, activation = "relu") %>%
```

```
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 10, activation = "relu") %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 1)

nn_mod3 %>%
  compile(loss = "mse",
          optimizer = optimizer_rmsprop(),
          metrics = list("mse"))

f = 10
eph = 300
bts = 40
nn3_results <- nn_CV(df,x,y,f, eph, bts, nn_mod3)
nn3_results

### All tuning results:
########################

# Read in results from rf and xgb if new session
rf_tune_res <- read_csv("rf_tume_results.csv")
xgb_tune_res <- read_csv("xgboost_tume_results.csv")

# best rf
rf_tune_res %>%
  filter(mean == min(mean))
# best xgb
xgb_tune_res %>%
  filter(mean == min(mean))
# best lasso
cv_lasso %>%
  filter(mean_mse == min(mean_mse)) %>%
  mutate(rmse = sqrt(mean_mse))


### -------------------------- Model Selection ----------------------###
################################################################################

### Cross-validation between models:
CV_between_mods <-  function(df,f){
  ## Input description:
  # df is a dataframe
  # f is the number of folds in k-fold CV

  ## Create k-folds (k=f)
  n <- dim(df)[1]
  n.adj <- floor(n/f)*f # we cant get an even f folds
  folds <- sample(1:n, n.adj, replace = FALSE) %>%
    matrix(nrow = f)
  rmse_results <- tibble(ols_rmse = rep(0,10),
                         lasso_rmse = rep(0,10),
                         rf_rmse = rep(0,10),
                         xgb_rmse = rep(0,10),
```

```
                        nn_rmse = rep(0,10))

## Iterate CV through each fold
for(k in 1:f){

  # Test/Train for this fold
  train <- df[-folds[k,],]
  test <- df[folds[k,],]
  # Reformat for lasso and NN
  train.x <- model.matrix(log_price ~ . - 1, data = train)
  train.y <- train$log_price
  test.x <- model.matrix(log_price ~ . - 1, data = test)
  test.y <- test$log_price

  ### RMSE for OLS
  rmse_results[k,1] <- glmnet(train.x, train.y, lambda = 0, alpha = 1) %>%
    predict(newx = test.x, s = 0) %>%
    as_tibble() %>% rename(pred = s1) %>%
    summarise(rmse = sqrt(mean((test.y - pred)^2)))

  ### RMSE for Lasso
  rmse_results[k,2] <- glmnet(train.x, train.y, lambda = 0.0408 , alpha = 1) %>%
    predict(newx = test.x, s = 0.0480) %>%
    as_tibble() %>% rename(pred = s1) %>%
    summarise(rmse = sqrt(mean((test.y - pred)^2)))

  ### RMSE for Random Forest
  rmse_results[k,3] <- rand_forest(mode = "regression",
                          mtry = 15, min_n = 3) %>%
    set_engine("ranger") %>%
    fit(log_price ~ ., data = train) %>%
    predict(new_data = test) %>%
    summarise(rmse = sqrt(mean((test.y - .pred)^2)))

  ### RMSE for XGBoost
  rmse_results[k,4] <- boost_tree(mode = "regression",
    trees = 1000,
    tree_depth = 13,
    min_n = 14,
    loss_reduction = 0.133,
    sample_size = 0.652,
    mtry = 42,
    learn_rate = 0.0600) %>%
    set_engine("xgboost", objective = "reg:squarederror") %>%
    fit(log_price ~ ., data = train) %>%
    predict(new_data = test) %>%
    summarise(rmse = sqrt(mean((test.y - .pred)^2)))

  ### RMSE for Neural Nets
  nn_mod <- keras_model_sequential() %>%
    layer_dense(units = 70, activation = "relu", input_shape = ncol(train.x)) %>%
    layer_dropout(rate = 0.4) %>%
    layer_dense(units = 50, activation = "relu") %>%
```

```
      layer_dropout(rate = 0.4) %>%
      layer_dense(units = 10, activation = "relu") %>%
      layer_dropout(rate = 0.4) %>%
      layer_dense(units = 1) %>%
      compile(loss = "mse",
              optimizer = optimizer_rmsprop(),
              metrics = list("mse"))
    # callbacks <- list(callback_early_stopping(monitor = "val_loss",
    #                                           min_delta = 0.001,
    #                                           patience = 10,
    #                                           mode = "auto",
    #                                           verbose = 1))
    nn_history <- nn_mod %>%
      fit(train.x, train.y,
          epochs = 900, batch_size = 40,
          validation_data = list(test.x, test.y))

    y_pred <- predict(nn_mod, test.x)
    rmse_results[k,5] <- sqrt(mean((test.y - y_pred)^2))
  }
  return(rmse_results)
}

f = 10
CV_between_results <- CV_between_mods(df,f)
CV_between_results

write_csv(CV_between_results, "CV_between_results.csv")

CV_between_results <- read_csv("CV_between_results.csv")
CV_between_results %>%
  summarise(mean(ols_rmse),
            mean(lasso_rmse),
            mean(rf_rmse),
            mean(xgb_rmse),
            mean(nn_rmse))

# Plotting
tibble(rmse = c(CV_between_results$ols_rmse, CV_between_results$lasso_rmse,
                CV_between_results$rf_rmse, CV_between_results$xgb_rmse,
                CV_between_results$nn_rmse),
       model = c(rep("ols", 10), rep("lasso", 10),
                 rep("rf", 10), rep("xgb", 10),
                 rep("nn", 10)),
       fold  = c(seq(1:10), seq(1:10),
                 seq(1:10), seq(1:10),
                 seq(1:10))) %>%
  ggplot(aes(fold, rmse, color = model)) +
  geom_point() +
  geom_line() +
  labs(title = "RMSE for each model accross folds") +
  theme_minimal()
```

```r
CV_between_results %>%
  summarise(mean(rf_rmse),
            mean(xgb_rmse))

# Testing random forest on validation set
rand_forest(mode = "regression",
            mtry = 15, min_n = 3) %>%
  set_engine("ranger") %>%
  fit(log_price ~ ., data = df) %>%
  predict(new_data = val_df) %>%
  summarise(rmse = sqrt(mean((val_df$log_price - .pred)^2)))

# 0.37 nice

# train on all data
rand_forest(mode = "regression",
            mtry = 15, min_n = 3) %>%
  set_engine("ranger") %>%
  fit(log_price ~ ., data = df_all)


##### ------------------- Case study ------------------- ######
glimpse(df)

x_new <- df %>%
  slice(300)
x_accom1 <- x_new %>%
  mutate(accommodates = 1)
x_accom3 <- x_new %>%
  mutate(accommodates = 3)
x_accom5 <- x_new %>%
  mutate(accommodates = 5)
x_accom9 <- x_new %>%
  mutate(accommodates = 9)

boots <- bootstraps(df_all, times = 1000)

# frist_resample <- boots$splits[[1]]
# as.data.frame(frist_resample)
#
# fit_rf_on_bootstraps <- function(splits){
#   rand_forest(mode = "regression",
#               mtry = 15, min_n = 3) %>%
#     set_engine("ranger") %>%
#     fit(log_price ~ ., data = as.data.frame(splits))
# }
#
# boot_models <- boots %>%
#   mutate(model = map(splits, ~ fit_rf_on_bootstraps(.x)))

accom1 <- rep(0,1000)
accom3 <- rep(0,1000)
accom5 <- rep(0,1000)
```

```
accom9 <- rep(0,1000)
B <- 1000
for(b in 1:B){
  BS_data <- as.data.frame(boots$splits[[b]])
  BS_mod <- rand_forest(mode = "regression",
                        mtry = 15, min_n = 3) %>%
    set_engine("ranger") %>%
    fit(log_price ~ ., data = BS_data)

  accom1[b] <- predict(BS_mod, new_data = x_accom1) %>%
    exp()

  accom3[b] <- predict(BS_mod, new_data = x_accom3) %>%
    exp()

  accom5[b] <- predict(BS_mod, new_data = x_accom5) %>%
    exp()

  accom9[b] <- predict(BS_mod, new_data = x_accom9) %>%
    exp()
  print(c("Iterations completed:",b))
}


BS_results <- tibble(accom1,
                     accom3,
                     accom5,
                     accom9)

BS_results <- BS_results %>%
  mutate(accom1 = unlist(accom1),
         accom3 = unlist(accom3),
         accom5 = unlist(accom5),
         accom9 = unlist(accom9))


write_csv(BS_results, "BS_results.csv")

BS_results %>%
  mutate(accom1 = unlist(accom1),
         accom3 = unlist(accom3),
         accom5 = unlist(accom5),
         accom9 = unlist(accom9)) %>%
  summarise(accom1 = quantile(accom1, c(0.025, 0.5, 0.975)),
            accom3 = quantile(accom3, c(0.025, 0.5, 0.975)),
            accom5 = quantile(accom5, c(0.025, 0.5, 0.975)),
            accom9 = quantile(accom9, c(0.025, 0.5, 0.975))) %>%
  t() %>%
  as_tibble() %>%
  rename(lower = V1, Median = V2, upper = V3) %>%
  mutate(Accomedates = as.factor(c(1,3,5,9))) %>%
  ggplot() +
  geom_pointrange(aes(Accomedates, y = Median, ymin = lower, ymax = upper)) +
```

```
  labs(title = "Bootstrapped 95% Confidence intervals for identical listing with differeing
       'Accomedates",
       y = "Log-Price") +
  theme_minimal()
```