Introduction

The main dataset we are working with in this project has data on Airbnb listings in Seattle. Through analyzing this data, we are trying to fulfill two main objectives. First, we are trying to determine whether several host and housing quality measurements have an effect on the price of a given listing. Second, we are attempting to determine whether these same measurements have an effect on the review score of the given listing. The first task is a regression based task and the second is a classification based solution. Our tasks have two different datasets, of which the price dataset has 5200 observations and 25 variables, including our response variable *price.* Our review dataset has 4043 observations and 26 variables, including our response variable *review_scores_rating.* Initially, there were 48 NA values in the price dataset and 34 NA values in the review dataset. These NA variables were entirely contained in 2 columns, *price* and *cleaning_fee.* We treated the NA values by converting them to 0, meaning that there was no price or cleaning fee for a given listing.

Supervised Analysis

I. Data Cleaning and Preprocessing

For both the regression and classification task, predicting *price* and classifying *review_scores_rating* in the best way required first looking at the data and evaluating the feature set. Both datasets contained the same feature set with the review dataset containing an extra variable - our response *review_scores_rating* for the classification task.

We began by looking for potentially problematic features in the dataset. In the price dataset, there were two: *amenities* and *property_type.* The *amenities* variable, for each listing, lists the specific amenities that the listing has such as wifi access, TV availability and air conditioning. The *property_type* variable lists the type of the room offered by a given listing. These two features presented due to the number of levels of the factor variables.

Each index in the *amenities* column contained a dictionary containing that specific listing's amenities. Logically, we decided to test if the number of amenities for a given listing would affect its price instead of the amenities themselves. We parsed the amenities column and produced a new feature *num_amenities* that reported the total number of amenities for each listing. We dropped the original amenities column from our analysis, and dropped the room type column as well from our analysis, as the ordering of its levels presented problems in the model creation process. We kept all other variables that were initially in the dataset in some form in our analysis.

Another slightly problematic feature was the latitude and longitude measurements included in the dataset. Each listing in the dataset also included its unique latitude and longitude coordinates. However, initially, we did not have any context for these coordinates and they were an empty measurement without some sense of how good or bad a given location was. To make the latitude and longitude measurements more useful and to improve the feature set, we tabulated the latitude/longitude coordinates of five Seattle places of interest: downtown, the Space Needle, the Gum Wall, Pike Place Market, and the Great Wheel. We wrote a script that calculated, for each listing, the distance between the listing and each of the five important locations in Seattle. This provided context to our task because it established how well-located certain listings were in comparison to others.

Finally, multiple features in the dataset required recoding. For instance, we refactored the *host_is_superhost, host_identity_verified,* and *instant_bookable* variables that were originally true/false variables coded as "t" and "f" into 0/1 variables with 1 coded as true. **## maybe line of code here and below ##** The *host_response_rate* variable was a percentage in the original dataset, and was changed into a proportion. Two variables: *host_response_time* and *cancellation_policy* were refactored into categorical variables with numbers representing levels of factors instead of words. Our response variable *price* and the *cleaning_fee* variable contained the “$” character before their values - this character was removed and the predictors were converted to numeric variables.

II. Regression Model Selection

At this point, after processing and cleaning the data, we had a feature set that we felt good about. To move forward with the regression task, we knew that some variables had linear correlation with our response variable *price,* namely *cleaning_fee* and *beds.* It seemed that fitting a multiple linear regression model was a logical place to start as a baseline technique for prediction.

To begin, we split our price dataset into a training and test set with 75% of data in the price dataset allocated to training and 25% allocated to a validation test set. We fit a multiple linear regression model, **## code here ##** regressing *price* against all the variables in our final, processed training set. The goodness of the fit was evaluated in two ways: against our regression assumptions and with cross-validated mean squared error. The residuals vs fitted values plot of the fitted regression object showed a cluster of points around 0 with high outliers, potential influential points, and an uneven distribution. Our baseline linear model was not optimal for the dataset, and we inferred another technique could produce a better fit. The cross-validated error for the linear model using k-fold cross validation was approximately 9188.028.

With our baseline in hand, we logically assumed that it’s possible the fit could be improved by improving the bias-variance tradeoff we were dealing with. To make the model more generalizable and get a sense for which predictors were important, we fit a ridge and lasso model on our training set. We used **## code here cv.glmnet ##** cross validation techniques to establish the optimal lambda penalty value for both the ridge and lasso regression techniques. Our cross-validated mean squared errors for both ridge and lasso were approximately 8941.927 and 8941.665 respectively, which was an improvement on our baseline model.

We further explored trees and random forests. Given the data had a mix of categorical and numeric predictors, with some outliers in the data, trees and random forests provide a robust measure that does not lose predictive power to high outliers and non-linearity. We fit a normal decision tree, and a pruned version of the tree. **## code here ##** From the plot of the decision tree we chose a cp parameter that minimized relative error. The cross validated mean squared errors both the normal decision tree and pruned tree improved upon ridge/lasso and were approximately 6572.661 and 6999.323 respectively.

Random forests are an algorithmic improvement on decision trees, due to their greedy search on a subset of features for the best split to make rather than evaluating all features in a decision tree. In addition, random forests retain the strengths of decision trees in working with numeric and categorical predictors and as an ensemble learner, they generally produce low bias, moderate variance models. The algorithm averages over multiple predictions to better predict the response.

We fit a random forest with 500 trees as a baseline on our training set. **## code here ##** Random forests had the lowest cross-validated mean squared error around 4005.778. The predictions that the random forest produced were the closest to the price of the listings in our training set. There are two parameters of interest in the random forest algorithm: the number of trees to grow and the number of variables randomly sampled as candidates for each split. We attempted to tune the random forest to find the best possible number of variables to sample for each split to minimize error, using tuneRF. **## code here ##** Tuning the number of trees to grow for a random forest isn’t necessary after a certain amount of trees, as the forests stabilize at around 200 trees.[1] Our tuned random forest used the default number of variables to consider for splits. We made our predictions on the test set using the tuned random forest.

After trying various algorithms, we concluded that random forests produced the lowest prediction error without extreme bias or variance, and we chose the random forest model as our final regression model. Regarding the relationship between the predictors in the model and the predictions themselves, an advantage of random forests is that they have a very interpretable variable importance measure that compares the prediction strength of each variable. **## Variable importance plot here ##** The most powerful predictors for price in terms of prediction strength were *cleaning_fee* and *host_listings_count.* Interestingly, the variables we added for location mapping were among the best predictors in the feature set. Our model selection process led us to choose our random forest model as the model that best explains the test data.

III. Classification Model Selection

Similar preprocessing was done for the price and review datasets, hence the model selection process was similar for both the regression and classification tasks - albeit with different algorithmic techniques. We split our review dataset into 75% training and 25% test as done in the regression task. The *amenities* and *property_type* variables were dropped from the overall dataset as in the previous task. The same location mapping and amenities cleaning added to the regression data feature set was included in the classification data feature set.To get a baseline accuracy, it seemed logical to begin with a naive classifier.

We chose a Naive Bayesian classifier as a baseline to learn more about our ability to classify for our response variable *review_scores_rating.* **## code here ##** Our cross-validated misclassification error was 17% as a baseline. Given that our response variable *review_scores_rating* was split into two classes - 0 and 1 - we thought logistic regression would serve as a good starting point.

A logistic regression model was fit on the training set. ##**code here ##** The cross-validated misclassification error was 12.8%. The variables that contribute most to the fit were *cancellation_policy, num_amenities, neighborhood_group, host_identity_verified* and *host_is_superhost.* The cleaning of the amenities variable done in preprocessing, on the evidence of the logistic fit, seemed to improve the feature set. However, there seemed to be possible improvements in misclassification accuracy, hence we tried three other classification techniques: support vector machines, boosting, and tree-based random forests.

Support vector machines were deployed in three different ways: utilizing a linear kernel, polynomial kernel and a radial kernel. The models were trained on the training set with the same set of predictors used for the naive and logistic models. **## code here ##** Without tuning, the three methods all produced cross-validated misclassification error of approximately 12%, improving on the logistic regression fit. Tuning the models for the parameters cost and gamma took an long time to run due to the large feature set - and the best models marginally improved misclassification error.

Boosting, like random forests, is an ensemble learner that produces low bias, moderate variance models. We theorized that adaptive boosting would be an algorithmic improvement on all three of the classification techniques used previously due to its additive nature. Utilizing gradient descent to minimize error over a fixed number of trees led us to believe boosting would increase classification accuracy. We fit an adaptive boost model with 100 iterations, using the “Freund” Adaboost algorithm. **## code here ##** Cross-validated misclassification error dropped to sub 12% - approximately 11.7%.

Finally, we chose to try random forests for many of the same reasons that we denoted for the regression task and for its flexibility. We fit a random forest with 500 trees on the training

[1] Hastie, Tibshirani. *Elements of Statistical Learning. 587.*

data. **## code here ##** Initially, the cross-validated misclassification error hovered around where the boosting algorithm was performing, but after tuning the number of variables randomly sampled as candidates for each split, random forests produced the lowest cross-validated misclassification rate of 11.35%. The boosting algorithm and the random forest algorithm produced similar results, but since the random forest classifier was easily interpretable i.e. it was clear to see which variables that were important for improving accuracy, we chose our random forest model as our final model. In a similar fashion to the regression task, we modeled the relationship between the predictors and their relative importance to the fit. **## plot here ## Say something about predictors/predictions here.**

Analysis of Results

1. Listings we did well or poorly on and possibly why
2. Outliers - Data cleaning can really improve accuracy
3. More time -> include better features that could possibly do as well as distance
4. Logical correlation exploration -> higher price more desirable, more amenities, better location etc -> time of the year?

The performance, on both the regression and classification task, of our predictor set was better than baseline models on average. With more time, there are clear next steps. The data was skewed and contained high outliers in multiple categories. More time dedicated to filtering the data and making it as usable as possible