STAT 444 Final Project : Airbnb Analysis

Jiawei Yu *Wenqi Wu*

July 31, 2019

1 Motivation and Introduction

The goal of this project is to predict the price of Airbnb's listings (in Toronto).
Price prediction is a complicated task, especially with housing prices. Many factors are in play such as the neighbourhood, access to public transit, available amenities, room size, and many more. Traditionally, data with such detailed information are difficult and costly to obtain. Luckily, Airbnb has millions of listings online with a lot of useful information and description, and these data are (almost) open and easy to obtain (i.e. web scraping). As a result, we are able to perform some more in-depth analysis than before.

If you are an Airbnb user, you probably had this experience: you were planinng for a trip, found several places that you liked to stay but can't decide which one to pick. Actually, you don't have to be an Airbnb user to experience this. Many hotel stayers also need to make decisions when they are comparing between different options.
If you've ever been in such shoes, then a price prediction model can help - we can use its predictions to access each listing's value (for example, calculate and compare the ratio of predicted:actual) and choose the one with the highest value. In addition, such a model can also be used to compare airbnb with hotel offers, or even rent prices of different places to see which one offers the most value.

Other than building a predictor, it would also be interesting to find out what factors influence the price the most (well, other than location and size of the place, since we all know they're the most important). In other words, we would like to know what perks people are looking for when they are booking Airbnb. Moreover, owners can use this information to improve their service. For example, if it was found that wifi is really important for most people, then owners may want to install/provide wifi (if not already done) in order to attract more customers.

2 Data and Preprocessing

2.1 General Information

Our data come from this website `http://insideairbnb.com/index.html`, which does regular web scraping to obtain listing data for large cities around the globe.
The data that we use was scraped on May 13, 2019 (i.e. these are what you would see if you visit airbnb.com on that day), and they do not represent the present, since the prices may change depending on the day of travel plans (i.e. Christmas might be more expensive due to higher demand), and new listings are added/deleted all the time.

There are a total of 20303 listings in our data. Some of the more important features (columns) include price, listing type, room type, and location (represented by longitude and latitude). We perform some exploratory analysis on these columns.

2.2 Mising values, Outliers and Preprocessing

It can be argued that prices are very subjectively set by the home owners, and therefore there will be a lot of variations unexplainable by any model. However, since Airbnb is a free market place, one can also argue that the prices are more or less objective enough, since it's driven by the "invisible hands" of supply and demand.

If the price of a place is too high, then it will not attract any customers and the owners will reduce the price or simply remove the listing. On the other hand, if a price is relatively too low, it will always be booked and the owners will want to increase the price in order to earn a higher profit.

With the assumption that prices are objective reflection of values, we should not include certain listings in our analysis because they are not marketable listings. For example, some owners make their places unavailable for anyone to book. These unmarketable listings can be harmful to our analysis, because owners might do weird things (for example, an ordinary condo is priced at $13,000 per night, but is not available at all). To make sure that we only include reasonable listings, we filter out any listing that is entirely unavailable in the next 365 days (these can be considered as "dead" listings). We also remove any listing with zero review, because they are very unlikely to represent market prices (note that a down side of this is that it also removes newly added listings). Then, a manual inspection into the highly priced places (over $1000 a night) was done, and 4 places that were found to be unreasonably (or even shockingly) overpriced were removed. We also removed two listings that are priced at 0.

Finally, we remove rows with important entries missing. The "important" columns are the features we include as candidate for expalantory variables in our model. See section 2.4.

In summary, these are the removals that we've done:

1. Listings unavailable for the next 365 days
2. Listings with no review
3. 4 unreasonably overpriced listings
4. 2 listings with price $0
5. Listings with important entries missing

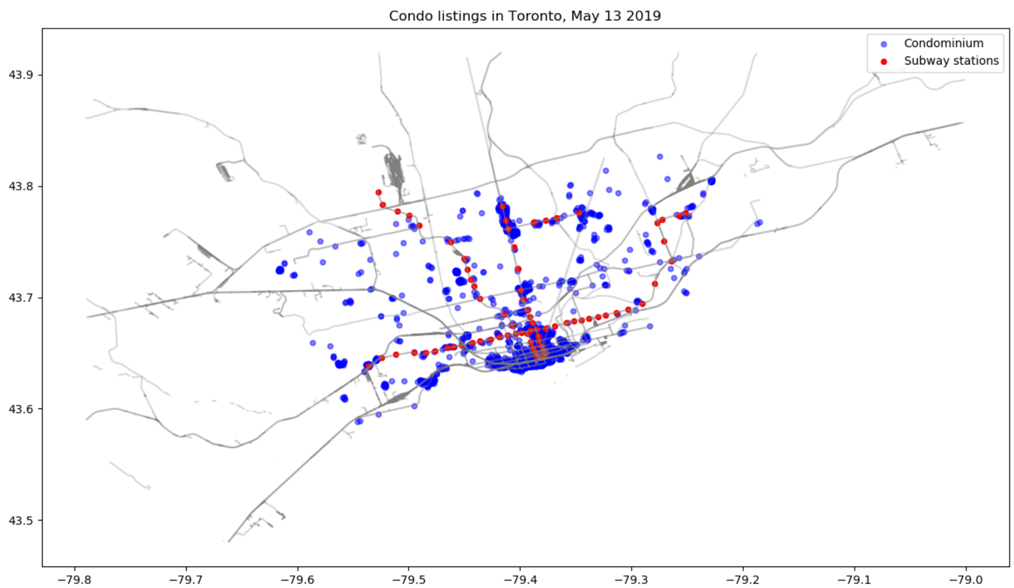
2.3 Features (and more preprocessing)

There are over 100 features (data columns) in the csv downloaded from the link above. However, some of these features are not relevant to our analysis (for example, link to the host's profile picture), and some other information, while being useful, are not suitable for statistical analysis (for example, an English description of the place). We only pick the features that are useful and work-able.

2.3.1 Property Types

When an owner creates a listing, Airbnb gives a list of over 25 property types for the owner to choose from. Some of the common types include houses, apartments, condos, townhouse, etc. A large list of categories is hard for us to work with, so we pick the 7 most common categories, namely houses, apartments, condos, townhouses, lofts, bungalows and guest suites. These categories account for 12118 out of 12554 perprocessed listings. Other categories such as boats are very rare, and are less comparable to the common types (i.e. when people are renting a boat, their consideration will be quite different than when renting an apartment).
See appendix (1) for the full list of categories and their counts.

Besides looking at the numbers for each type, it would also be interesting to look at the geological distributions. Listings for each common type are plotted on Toronto's map. Due to space limitation, we only present the distribution of condos here. The rest are in appendix (2):



Condos are densely distributed in downtown, the lakeshore, along Yonge street, near Don Mills and along the subway stations. As a comparison, apartments and houses have a wider distribution acorss the whole Toronto municipality. See appendix(2) for details.

2.3.2 Amenities

Each listing has an "amenities" column, where each entry is a list of all the amenities and features the place offers. This list tends to be very comprehensive, if not exhaustive. It turns out that there is a total of 194 unique amenities in our dataset. We handpicked a fraction of them that we believe people will take into considerations when they're making decisions. This reduced the number of amenities to 86, with some similar amenities combined (for example, we combined "internet access", "wifi" and "pocket wifi" into one amenity). For each amenity, we will create an indicator variable that takes either 1 (the place has this amenity) or 0. The full list of amenity with their counts are available in appendix(3).

The inclusion of amenities in our model is mainly for the purpose of exploring relative importance of these "variables", rather than for predictive purposes. That is, we're interested in what people care about the most(or what most people care about). This is because different people have different needs when they travel. For example, free parking is highly desired by those who drive, but is a useless feature for those who don't.

2.3.3 Location

You've probably heard this from real estate agents: "location, location, location". They're describing the top 3 most important aspects of real estate. We can never properly valuate a listing without referring to its location.
Our data includes a column "review_scores_location" which is the average review scores on how convenient the location is, for each listing. Unfortunately, as we all know Canadians are very kind, most listings (96%) have received 9 or higher scores on location (out of 10).

Luckily, we have access to the latitude-longitude of all the places. For privacy reasons, some of these numbers are not exact, but they are still precise enough so we know which neighborhood it is in.
We make use of the lat-lon information by calculating the distance to the nearest TTC subway station. Note that the distince calculated is the length of the straight line between two points, not the actual walking distance. Since most of Toronto's roads are grids, we expect the walking distance to be within $\sqrt{2}d_e$ most of the time, where d_e is the Euclidean distance. We're talking about a factor of roughly 1.414, a more or less negligible difference.

We could have done a similar thing with bus stops, but we chose not to, because Toronto has a rather high density of bus stops (also because obtaining the GPS lcoations of all the bus stops is very hard). According to Toronto Transit Commission, in 2017 there were 8640 bus stops in Toronto. If we assume that these bus stops are placed equi-spaced throughout Toronto (which has an area of 630 KM^2), then that's equivalent to a maximum distance of 190 meters between any location to its nearest bus stop. The points is, Toronto has a very high density of bus stops.

Measurements of proximity to subway stations is great, but still does not properly reflect the important difference between locations. For example, a condo near a subway station in Downtown costs more than a condo near a subway station in the rural area. To account for this effect, we want to "normalize" the prices. To do this, we divide each price by the relative real estate price of its neighbourhood. Suppose there are N neighborhoods in Toronto (N = 142 in reality), then

$$P_{ni}^r = \frac{P_{ni}}{F_n}$$

Where

1. P_{ni}^r is the normalized price of the i-th listing in the n-th neighborhood
2. P_{ni} is the price of i-th listing in the n-th neighborhood
3. F_n is the scaling factor for the n-th neighborhood. $F_n = \frac{C_n}{C_m}$ where C_n is the average price of 3-bedroom houses in the n-th neighborhood, and $C_m = \frac{1}{N}\sum_N C_n$

We obtain the average price of 3-bedroom houses from `https://www.zolo.ca/toronto-real-estate/neighbourhoods`, and use the average sold price during the Apr 20 - June 15 period.

By calculting the "normalized price", we hope to account for the effects of different regions in order to reduce the unexplained variations in our response.

2.4 Final set of features used

We use normalized_price as the response variable. Below are all the explanatory variables that we will include in our models:

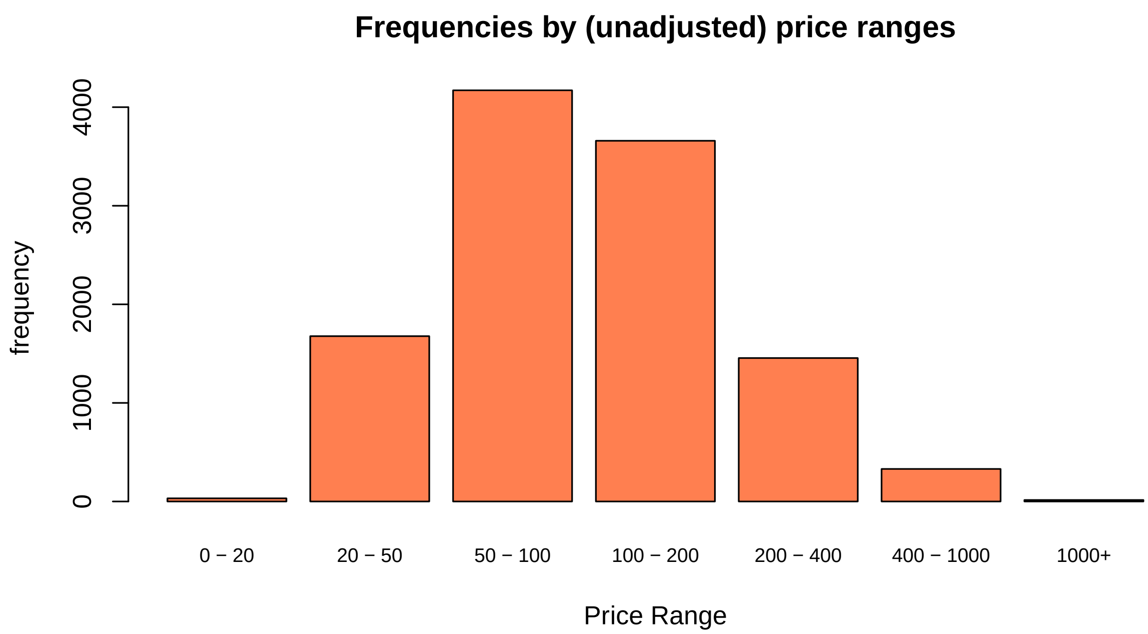
name	description
response time	On average, how long does it take for the host to respond to inquiries
response rate	On average, what percentage of inquiries do the host respond to
host is superhost	If a host is a "super host" or not (as verified by Airbnb)
subway distance	distance (in meters) to the nearest subway station
property type	Either house, condo, apartment, guest suite or hotel
room type	Either entire home, private home or shared room
accomodates	Number of guests the place can accomodate
bathrooms	Number of bathrooms
bedrooms	Number of bedrooms
beds	Number of beds
minimum nights	Minimum number of nights a guest can book
number of reviews	Total number of reviews given to this place
review scores rating	Average overall rating
review scores accuracy	Average rating on accuracy of the host's descriptions
review scores cleanliness	Average rating on cleanliness
review scores checkin	Average rating on the checkin procedure
review scores communication	Average rating on communication with the host
review scores location	Average rating on the location
instant bookable	whether the place can be directly booked without the need to be approved by host
amenities	86 indicator variables. See appendix (3)

Note that when we build models, the categorical features with more than 1 levels will be transformed into (l-1) indicator variables, where l = number of levels.

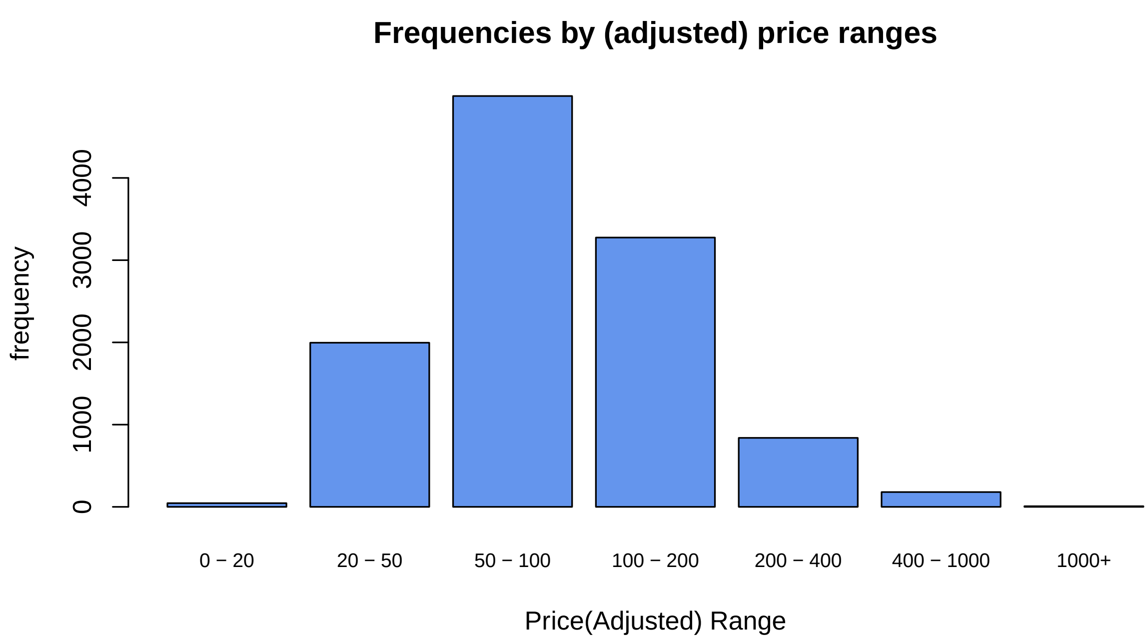
2.5 Distributions After Preprocessing

After all data preprocessing is done, we are left with 11339 data points. It would be interesting and useful to check/compare the distributions of prices before and after preprocessing.

2.5.1 Price

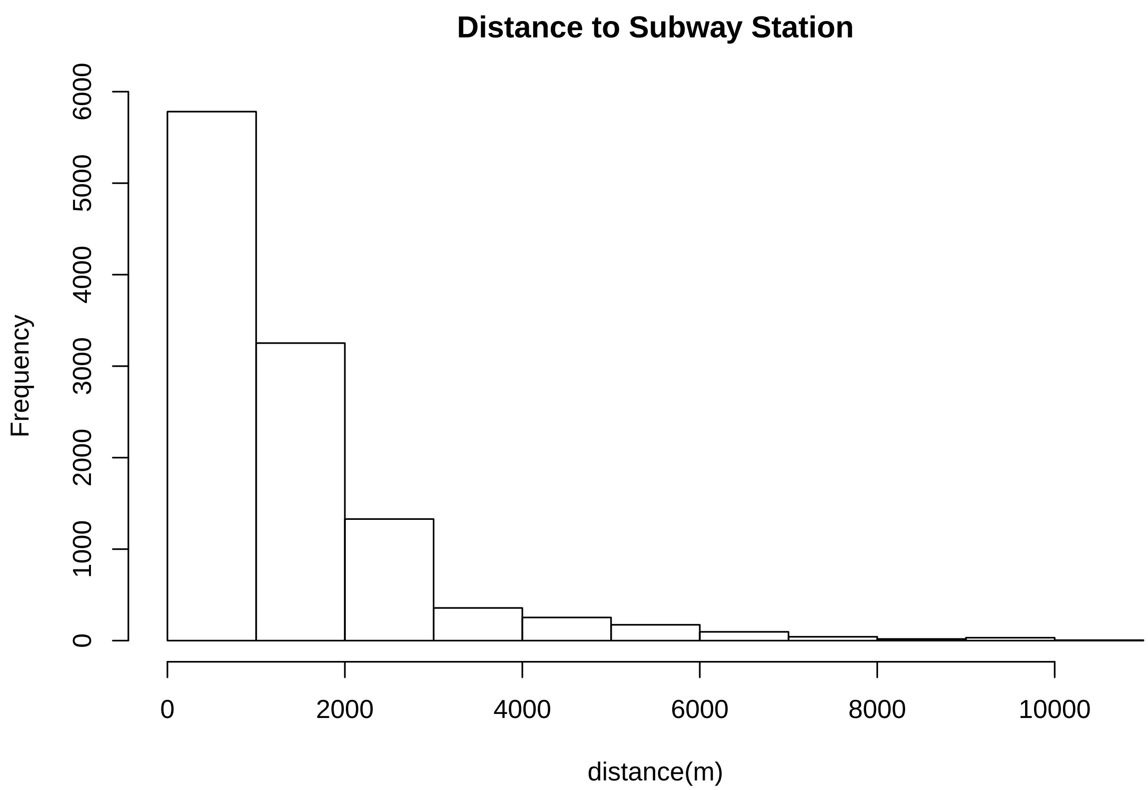


Most listings are in the [50,200) range, with a good amount of listings between \$20 and \$50, and between \$200 and \$400. There are some listings in the (400,1000) range, and places cheaper than \$20 or higher than \$1000 are rare to find.



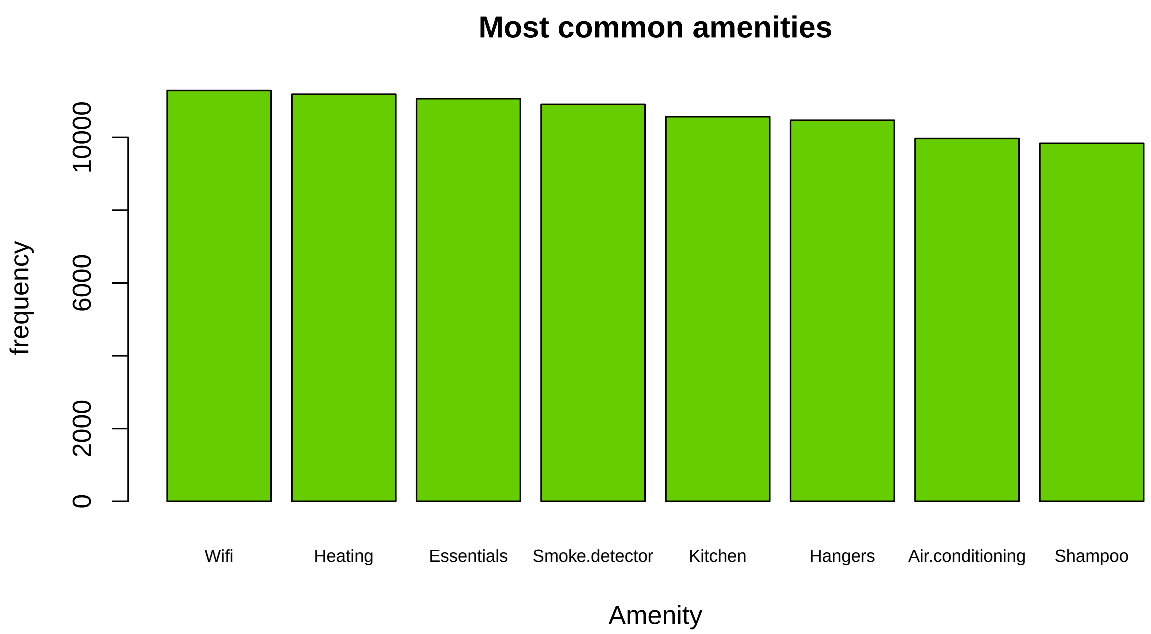
After adjusting for regional effect, the distribution changes considerably. The price range is now more concentrated in the [50, 100) band than before, and we're seeing fewer listings over 100 dollars.

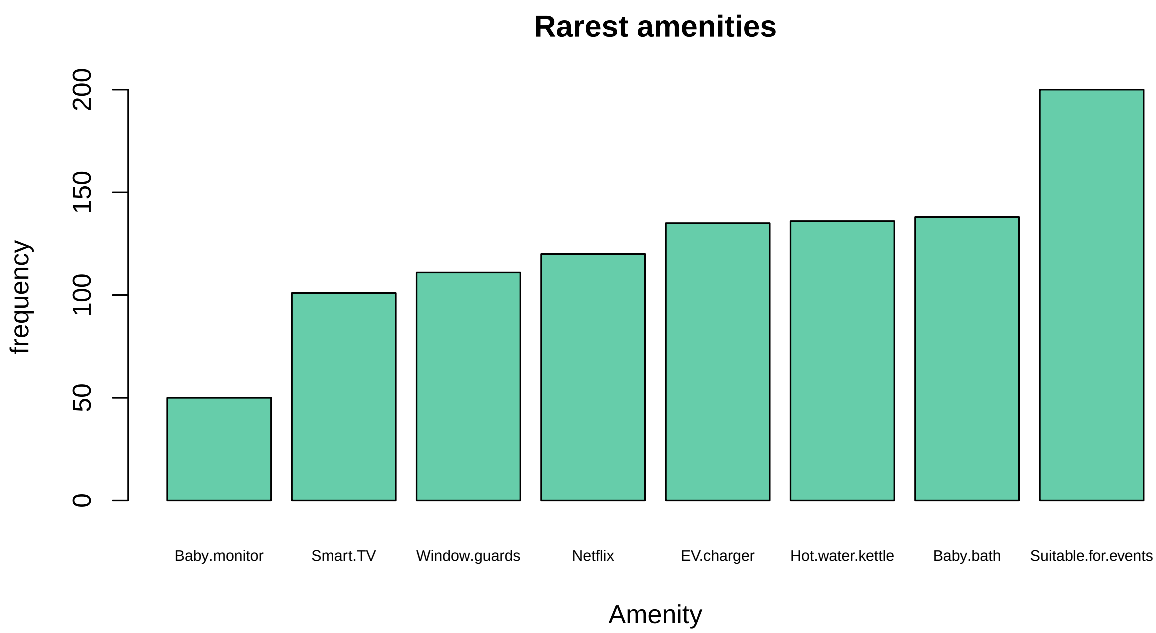
2.5.2 Distance to Subway Stations



About half of the listings are within 1KM of (Euclidean) distance within a subway station, and most places are within 2 KM of stations.

2.5.3 Most Common Amenities





The most common amenities are no surprise.
As for the least commonly offered amenities, it's kind of surprising that hot water kettles are a rare find.

We are now ready to move to the function estimations.

3 Base line: Linear Regression

For this section and all subsequent sections, we use the fisrt 10000 data points as training data, and remaining 1339 data points as testing.
We consider an additive linear regression model, and use this as the base line of comparisons for other models.

3.1 Including amenities

Below are outputs from the main effects model. Coefficients are not shown due to space limit.
Residual standard error: 62.96 on 9886 degrees of freedom
Multiple R-squared: 0.5255, Adjusted R-squared: 0.5201
F-statistic: 96.89 on 113 and 9886 DF, p-value: $< 2.2e-16$

The list of amenities is huge yet most of them is statistically insignificant in the main effects model. Furthermore, many of these amenities appear to have negative effects (negative coefficients) on the price. Some of the possible reasons include:

1. People don't care about certain amenities
2. Some amenities (such as wifi) are included in all but a few listings
3. There is collinearity between different amenities, and also between amenities and other explanatory variables (such as property type being house will be related to the place having a BBQ)
4. We are not considering any interaction

The sMSE on the training and MSPE on the test sets are, respectively:

```
## [1] 3918.314 4266.040
```

Perhaps we can benefit from a much sparser model that excludes all the amenities.

3.2 Not including amenities

```
##
## Call:
## lm(formula = normalized_price ~ ., data = train.sparse)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -616.52  -28.06   -5.05   19.41 1243.59
##
## Coefficients:
##                                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     -37.629561  14.658836  -2.567  0.01027 *
## response_timewithin a day       -21.362600   8.437236  -2.532  0.01136 *
## response_timewithin a few hours -23.849588   9.092209  -2.623  0.00873 **
## response_timewithin an hour     -26.532613   9.160463  -2.896  0.00378 **
## response_rate                    18.590954   8.971921   2.072  0.03828 *
## host_is_superhost                -1.676694   1.457343  -1.151  0.24996
## property_typeBungalow            -2.116001   3.720130  -0.569  0.56951
## property_typeCondominium          4.844030   1.784695   2.714  0.00665 **
## property_typeGuest suite         -3.767857   3.008722  -1.252  0.21049
## property_typeHouse                6.235315   1.934381   3.223  0.00127 **
## property_typeLoft                25.218507   4.426662   5.697 1.25e-08 ***
## property_typeTownhouse            5.230946   3.208538   1.630  0.10307
```

```
## room_typePrivate room           -38.084639   1.851378 -20.571  < 2e-16 ***
## room_typeShared room            -31.012741   5.821732  -5.327 1.02e-07 ***
## accommodates                     13.515169   0.670064  20.170  < 2e-16 ***
## bathrooms                        38.969669   1.580134  24.662  < 2e-16 ***
## bedrooms                         18.321473   1.342496  13.647  < 2e-16 ***
## beds                             -3.028356   1.163990  -2.602  0.00929 **
## minimum_nights                   -0.082686   0.038862  -2.128  0.03339 *
## number_of_reviews                -0.123889   0.012184 -10.168  < 2e-16 ***
## review_scores_rating              0.411577   0.185862   2.214  0.02682 *
## review_scores_accuracy           -2.232747   1.513976  -1.475  0.14031
## review_scores_cleanliness         4.381797   1.113935   3.934 8.42e-05 ***
## review_scores_checkin            -2.201815   1.492154  -1.476  0.14008
## review_scores_communication       0.498803   1.633180   0.305  0.76005
## review_scores_location            0.878069   1.280401   0.686  0.49287
## instant_bookable                 -3.008598   1.388394  -2.167  0.03026 *
## subway_distance                   0.003826   0.000493   7.759 9.37e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 64.06 on 9972 degrees of freedom
## Multiple R-squared:  0.5045, Adjusted R-squared:  0.5031
## F-statistic:   376 on 27 and 9972 DF,  p-value: < 2.2e-16
```

The sMSE on the training set and MSPE on the test set are, respectively:

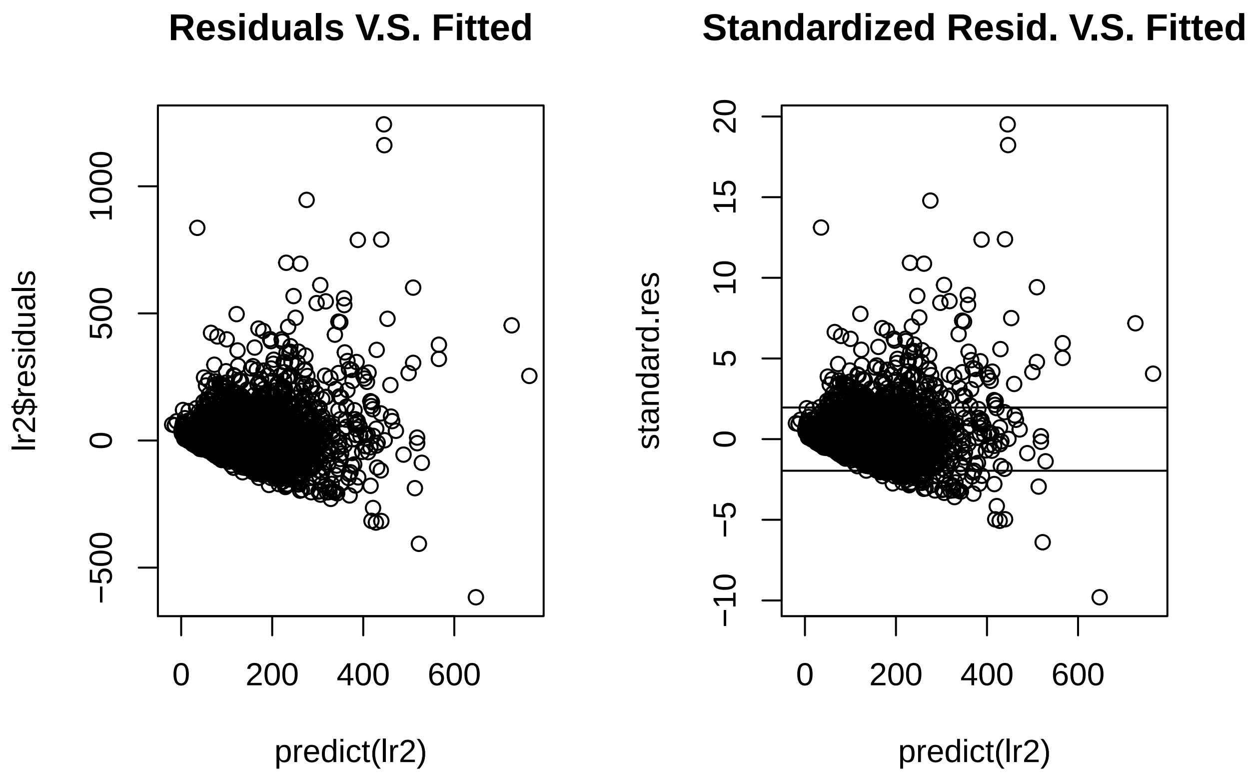
```
## [1] 4091.965 4573.585
```

Most of these coefficients make sense, with a few exceptions:

1. Number of reviews has a negative coefficient.
2. Instant bookable has a negative coefficient.
3. Review scores on checkin has a negative coefficient.
4. Distance to subway has a positive coefficient (although the coefficient is quite small).

The nested model has $RSS = 39183135$ compared to 40919654 for the full model. Even though we reject the null hypothesis that none of the amenities are important, the improvement in R-squared is pretty small (0.5045 to 0.5255), and for sparsity reasons we might prefer the small model.

We take a brief look at the residuals:



The large residuals are mostly large houses (suitable for events of large groups). Houses with many bedrooms tend to be much more expensive than houses with 3 or 4 bedrooms, therefore our model with only linear terms are unable to capture this relationship. Also these places tend to have a higher price due to higher cleaning cost.

4 Smoothing Splines

We are not going to perform smoothing slines on all variables because

1. It's not possible to capture all interaction effects.
2. It's not possible to visualize in a space with over 100 dimensions
3. Many variables are categorical instead of continuous

We are going to perform smoothing splines with only continuous variables.
consider a scenario where a group of people is looking to book an entire house for 1 day. We pick `bathrooms`, `bedrooms`, `number_of_reviews` and `subway_distance` as the explanatories and test for interaction effects between `bathrooms` and `bedrooms`. Some other continuous variables such as accomodates and beds are highly correlated with number of bedrooms so we do not include them. We also exclude review scores because they have few unique values (we are not able to perform smoothing splines with variables that has few unique values).

We split the training data into 5 folds and perform cross validation. For each fold we build a smoothing splines with `bathroom` and `bedrooms` having degree 1 (these variables have very few unique values), and for `number_of_reviews` and `subway_distance` we use cubic regression spline. We construct two models, one with and one without an interaction between `bathrooms` and `bedrooms`, then see whether we obtain smaller CV error with it.

We obtain the following CV errors for each fold, using interaction:

```
## [1] 23411.188  3080.415  3914.741  3481.360  3938.860
```

And the following CV errors without interaction:

```
## [1] 7677.158 3292.818 4087.663 3488.324 4316.117
```

The mean CV errors for two models are, respectively

```
## [1] 7565.313 4572.416
```

The mean CV error for no interaction is lower (4572 v.s. 7565), so we adopt the smoothing spline with no interaction, and re-train the model on the whole (filtered) training set. Model output is below:

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## normalized_price ~ s(bathrooms, k = 1) + s(bedrooms, k = 1) +
##     s(number_of_reviews, bs = "cr") + s(subway_distance, bs = "cr")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  126.462      2.149   58.84   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                       edf Ref.df      F  p-value
## s(bathrooms)         1.989  2.000 59.427  < 2e-16 ***
## s(bedrooms)          1.000  1.000 26.298 3.52e-07 ***
## s(number_of_reviews) 7.699  8.467  3.834 0.000132 ***
## s(subway_distance)   2.079  2.437  7.943 0.000132 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.306   Deviance explained = 31.5%
## GCV = 4563.2  Scale est. = 4498.7    n = 974
```

Obtain the MSPE on the test set (filtered by entire condos with minimum 1 night):

```
## [1] 10166.77
```

Compare this with the MSPE of a linear regression model using the same 4 explanatory variables and the same filtered training set:

```
## [1] 10500.22
```

The smoothing spline performs slightly better than the linear regression model.

5 Random Forests

5.1 Prediction - without amenities

Models in this subsection are selected based on pedictive power. We will have a different subsection for variable importance.

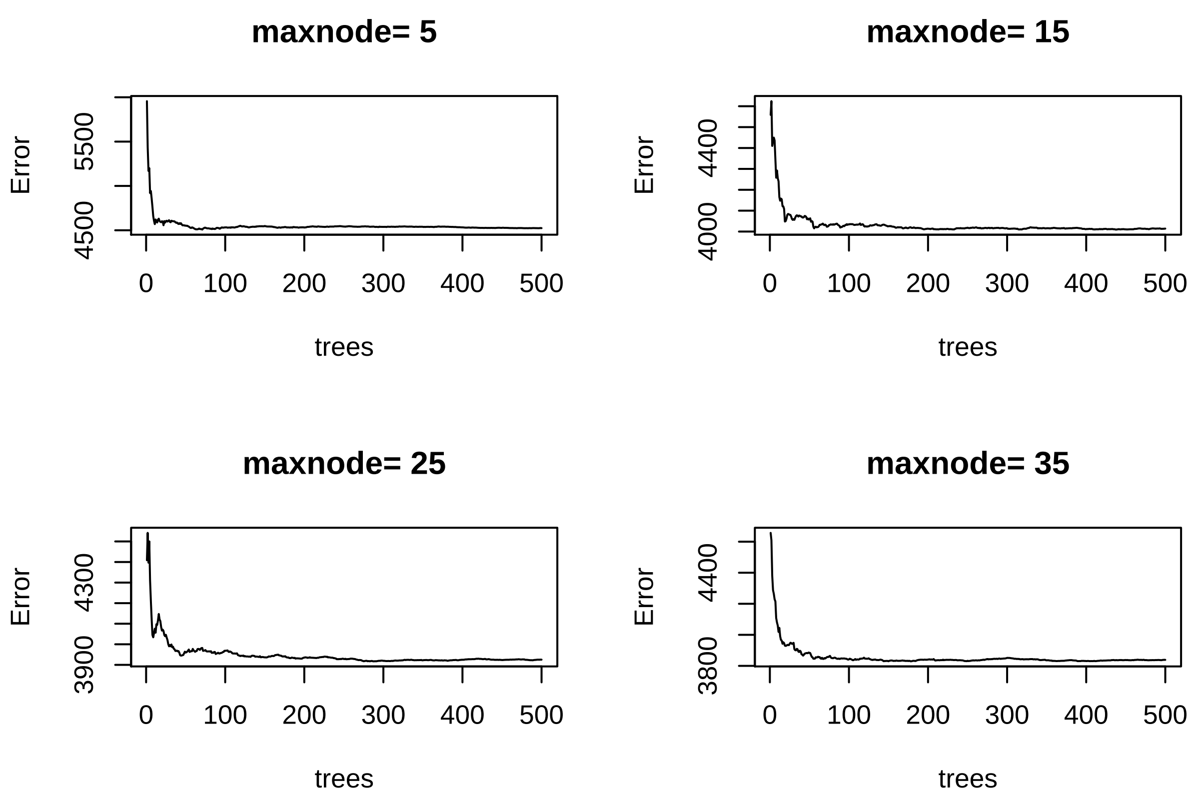
We first construct random forests without using amenities.
There are multiple tuning parameters, including number of trees, size of subset of explanatories to use at each node, and maximum number of nodes. We will fix the number of trees at 500 since in random forests, more trees usually does not harm the performance, and we will show later that 500 trees is enough.
We first select the maximum number of nodes, using a two-step approach:

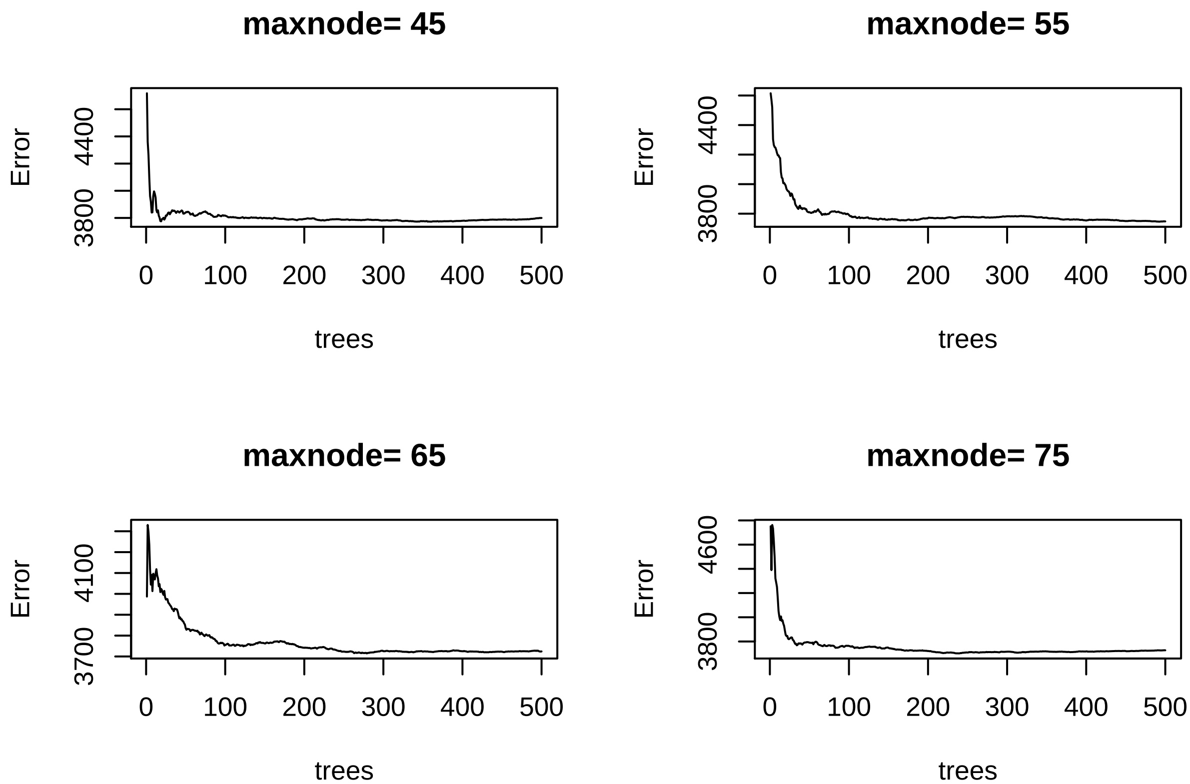
1. Experiment with a wide range of values to find the upper and lower bound on the optimal choice

2. Try within the bounds to find a more precise optimal value

At each step we use the average OOB errors as our criterion.

We start with the wider range of values in (5, 15, 25, 35, 45, 55, 65, 75). The size of variates to use at each node (m) is fixed at p/3 where p=total number of explanatories. We obtain the graphs of OOB error v.s. number of trees below:





As we can see from these graphs of OBB error v.s. number of trees, for any value of maximum nodes, we reach the minimum OOB error well before 500 trees, indicating that 1000 trees are enough.
To pick upper and lower bounds on the optimal value, we take a look at the mean OOB errors for each random forest:

```
##        5       15       25       35       45       55       65       75
## 4523.890 4014.214 3925.217 3838.739 3799.817 3747.670 3724.186 3727.100
```

Max node = 65 produces the minimum OOB error, so the upper bound is 74 and the lower bound is 56. We proceed to step 2, experimenting with max nodes taking values in (56, 59, 62, 65, 68, 71, 74), and evaluate the models similar to above. Doing so obtains the following OOB errors:

```
##       56       59       62       65       68       71       74
## 3753.969 3762.242 3722.117 3745.610 3744.386 3729.745 3711.941
```

These numbers are very close to each other and the differences are very likely to be results of randomness. We pick the final value to be 74 since it produces the minimum OOB errors.

We then experiment with size of variates to use at each node (m). We using values in (1, p/5, p/4, p/3, p/2, p) where p = total number of explanatories, holding max nodes = 74 and ntrees = 500 fixed. As before, OOB errors is used as the criterion.

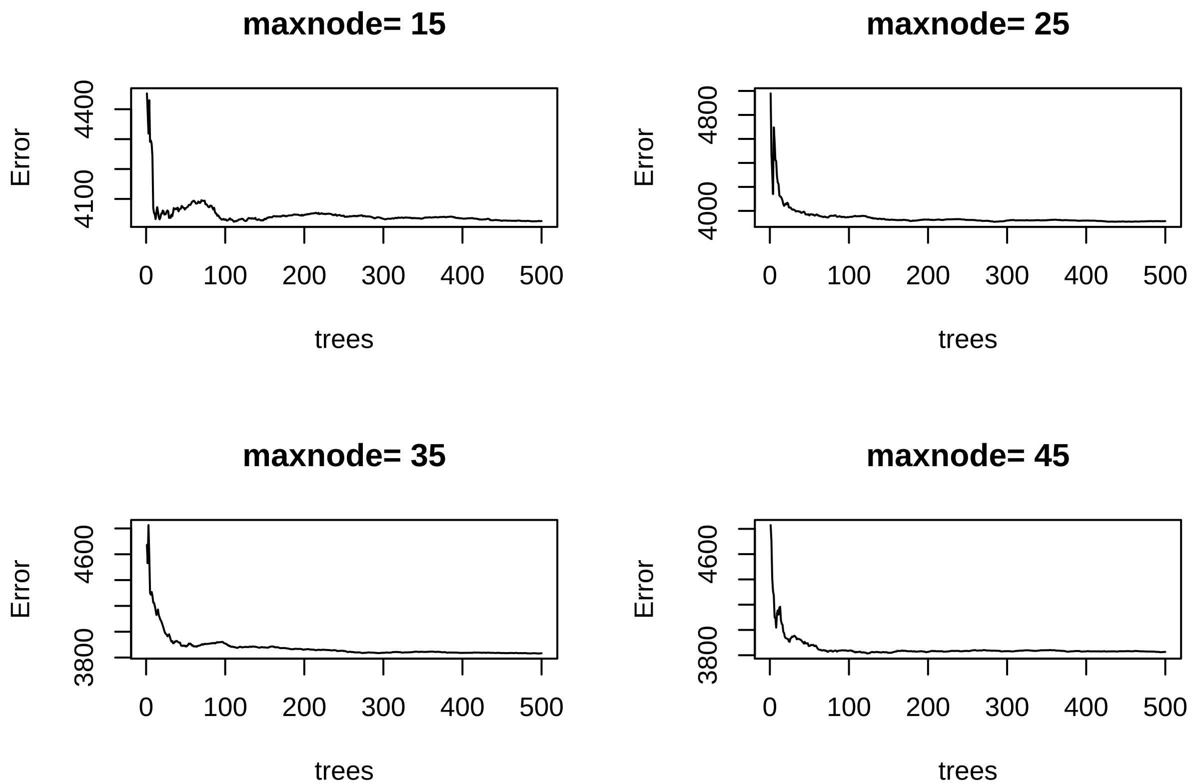
```
##        1      p/5      p/4      p/3      p/2        p
## 4859.230 3795.100 3754.610 3741.076 3716.515 3776.382
```

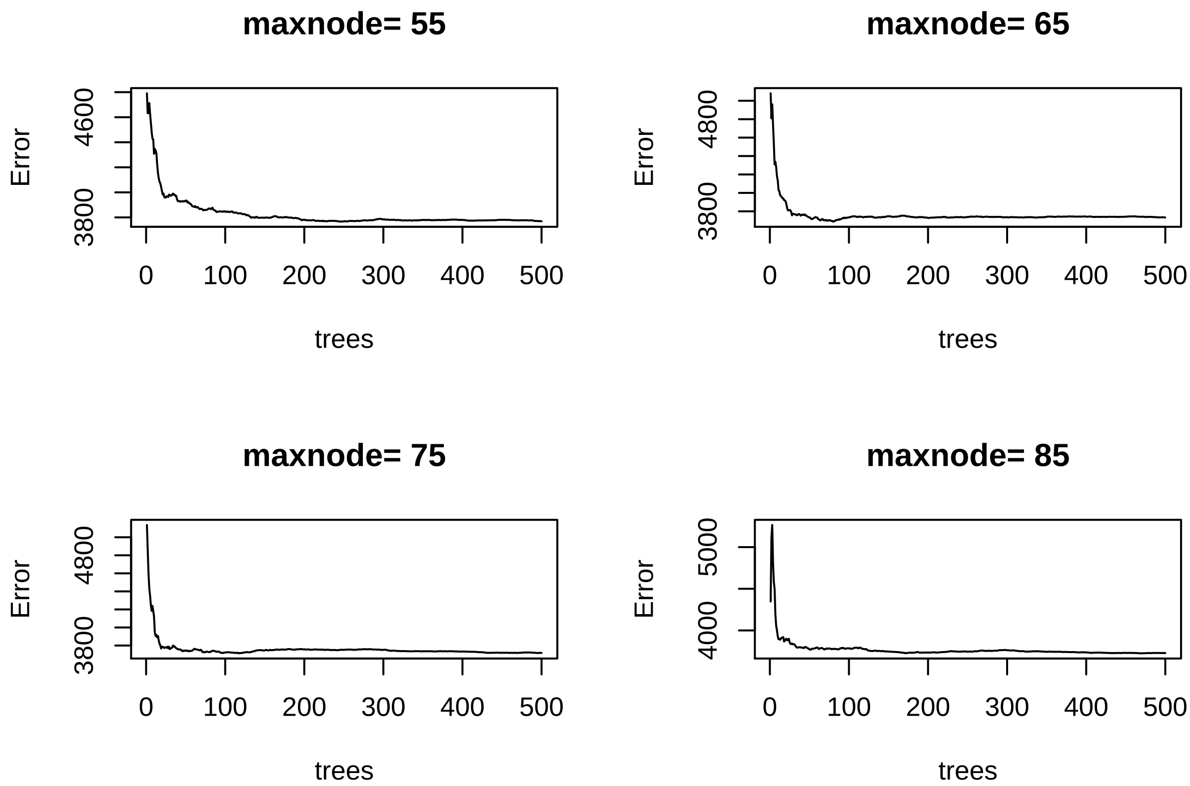
Using m=p/2 gives us the smallest OOB error. We have found our hyperparameters to be ntree=500, maxnode=74 and m=p/2. The performance on the test set as measured by MSPE is:

```
## [1] 4520.163
```

5.2 Prediction: With Amenities

We repeat the above process, now including amenities. Number of trees is still 500, and we first experiment values of maxnodes in (15, 25, 35, 45, 55, 65, 75, 85) holding m fixed at p/3 (p is now much larger). The following OOB error vs ntree graphs are obtained:





500 trees are sufficient based on the plots presented above.
The OOB errors:

```
##       15       25       35       45       55       65       75       85
## 4026.213 3914.741 3833.445 3826.525 3769.063 3732.613 3717.780 3728.060
```

Now try values between 66 and 84 (in increments of 3) to obtain the following OOB errors:

```
##       66       69       72       75       78       81       84
## 3744.371 3740.977 3723.982 3736.500 3721.810 3730.220 3711.009
```

Optimal value is 84. We use this and ntrees=500 to find the optimal m.
The OOB errors for different values of m:

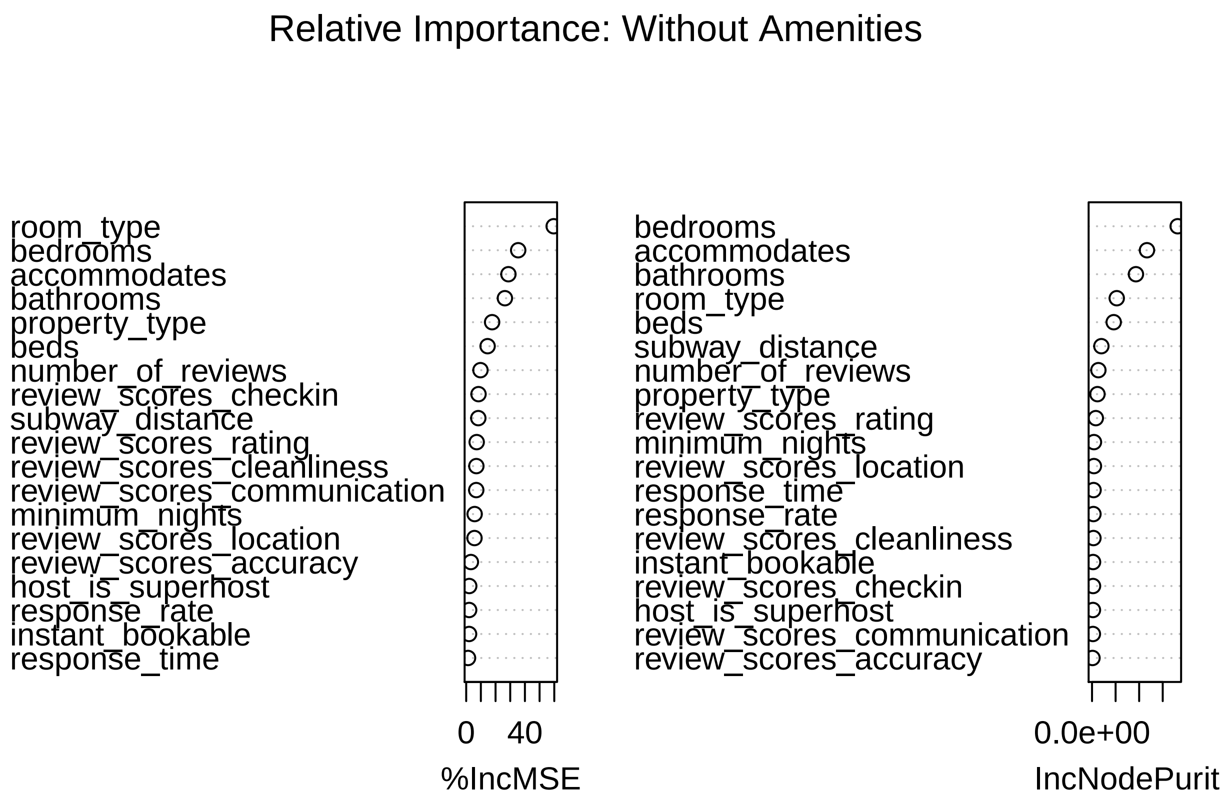
```
##        1      p/5      p/4      p/3      p/2        p
## 6614.469 3784.191 3738.205 3715.883 3725.688 3807.708
```

We choose m = p/3. Using these hyperparameters, the MSPE on the test set is:

```
## [1] 4098.838
```

5.3 Variable Importance - Without Amenities

To evaluate the relative importance of variables, we build a random forest using the optimal hyperparameters we previously found (for no amenities model). The whole dataset (train+test) is used. Below is the relative importance graph.



Variable importance as ranked by potential increase in MSE:

```
##                               %IncMSE IncNodePurity
## room_type                   59.592950    5243730.64
## bedrooms                    35.422486   18172810.21
## accommodates                28.764553   11653890.61
## bathrooms                   26.366711    9314540.49
## property_type               17.668470    1149197.50
## beds                        14.593127    4600891.17
## number_of_reviews            9.731027    1344833.99
## review_scores_checkin        8.405300     206517.65
## subway_distance              8.258284    1977183.21
## review_scores_rating         7.159920     798890.95
## review_scores_cleanliness    6.932933     295106.63
## review_scores_communication  6.848932     177015.37
## minimum_nights               5.771194     404638.37
## review_scores_location       5.662641     400836.97
## review_scores_accuracy       3.208235      93392.75
## host_is_superhost            2.080432     191453.33
## response_rate                2.020099     299769.23
## instant_bookable             1.974163     213430.15
## response_time                1.269722     323344.69
```

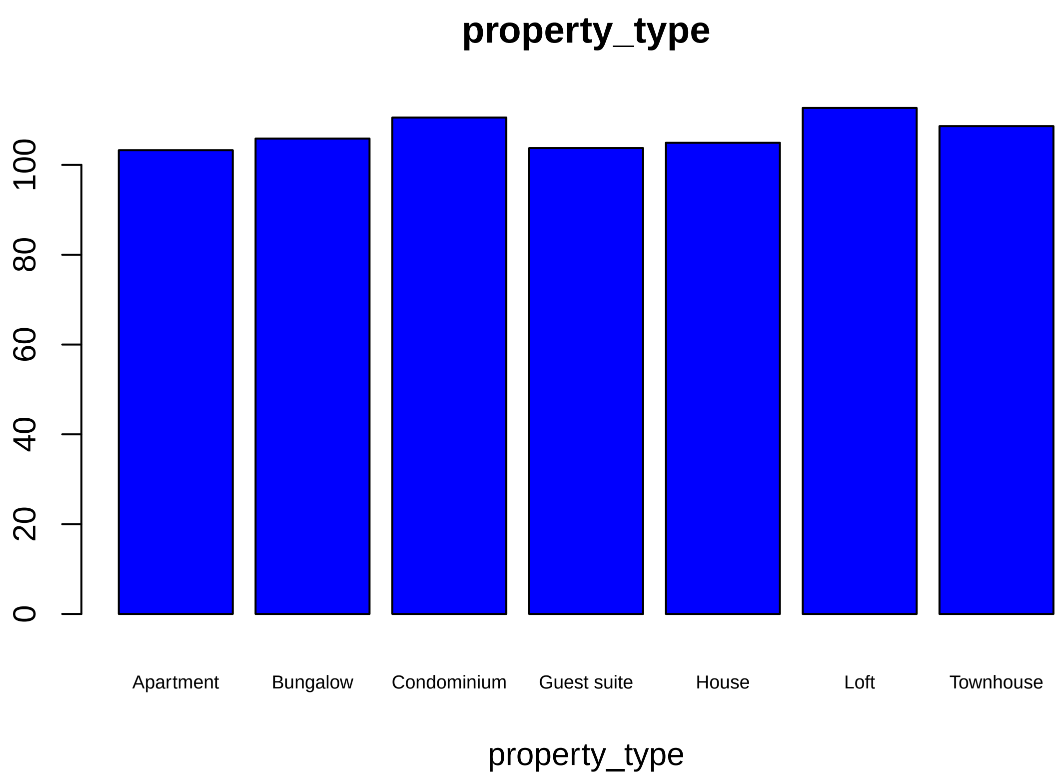
As we can see, some variables are more important in terms of potential increase in MSE than in terms of reduction in RSS. Variables like room type have few unique values, therefore their contribution to RSS is not as large compared to other numeric variables. However, there may be many interaction effects involving room type, so removing it has a huge impact on prediction accuracy. For this reason, we use increase in MSE as our main criterion for relative importance.

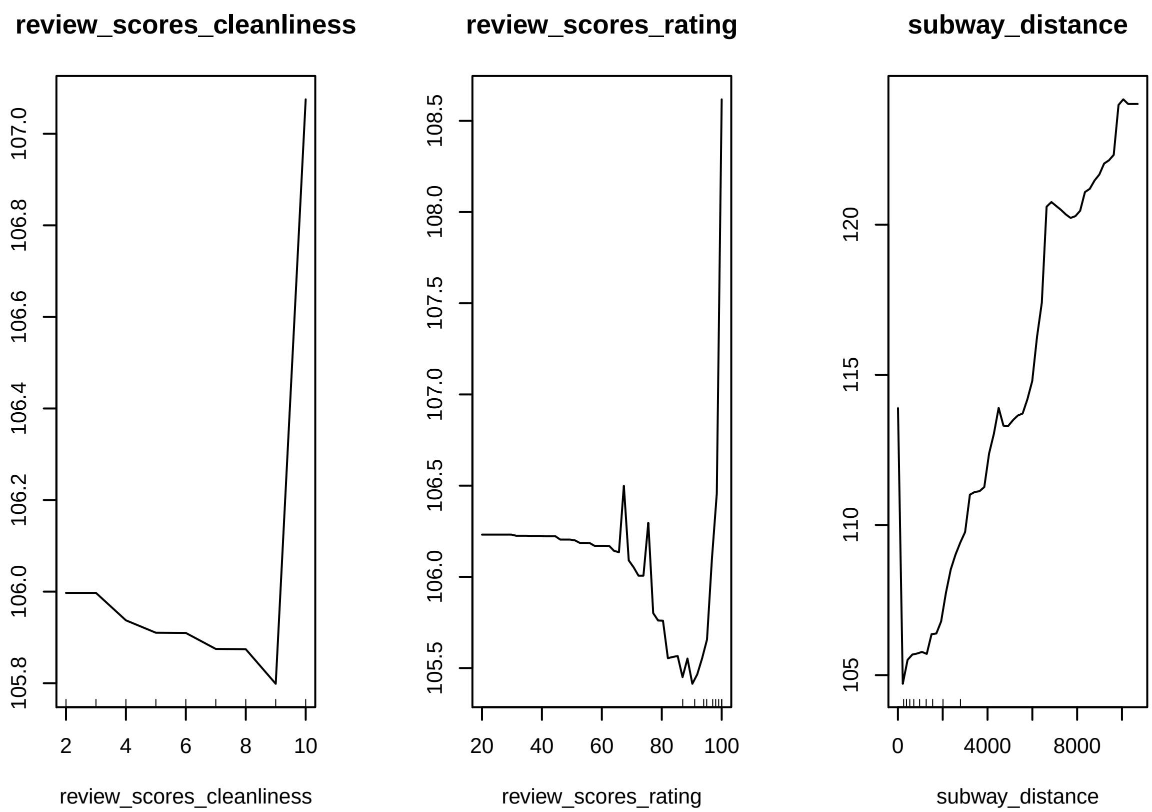
People care about room type (entire place, private bedroom or shared bedroom), number of bedrooms, number of guests accomodated and property type the most. These 4 variables have the largest increase in MSE (if removed). They also make the largest contributions to RSS reductions.

We summarize some interesting findings below:

1. Review scores on checkin is the most important type of review in terms of increase in MSE, but the overall review rating is most important in terms of RSS reducion.
2. Whether a place is instant bookable has little impact. People do not care about whether a place is instant bookable or not. In other words, most people are willing to spend more time waiting for responses/approval from the hosts.
3. How fast a host respond or how often a host respond to messages are relatively unimportant.
4. Even with our prices normalized by region, the distance to subway is still somewhat important in determining the prices.

Relative importance does not tell us the relationship between variables and price. We need to look at partial dependency plots to determine this. We choose some interesting ones to see:





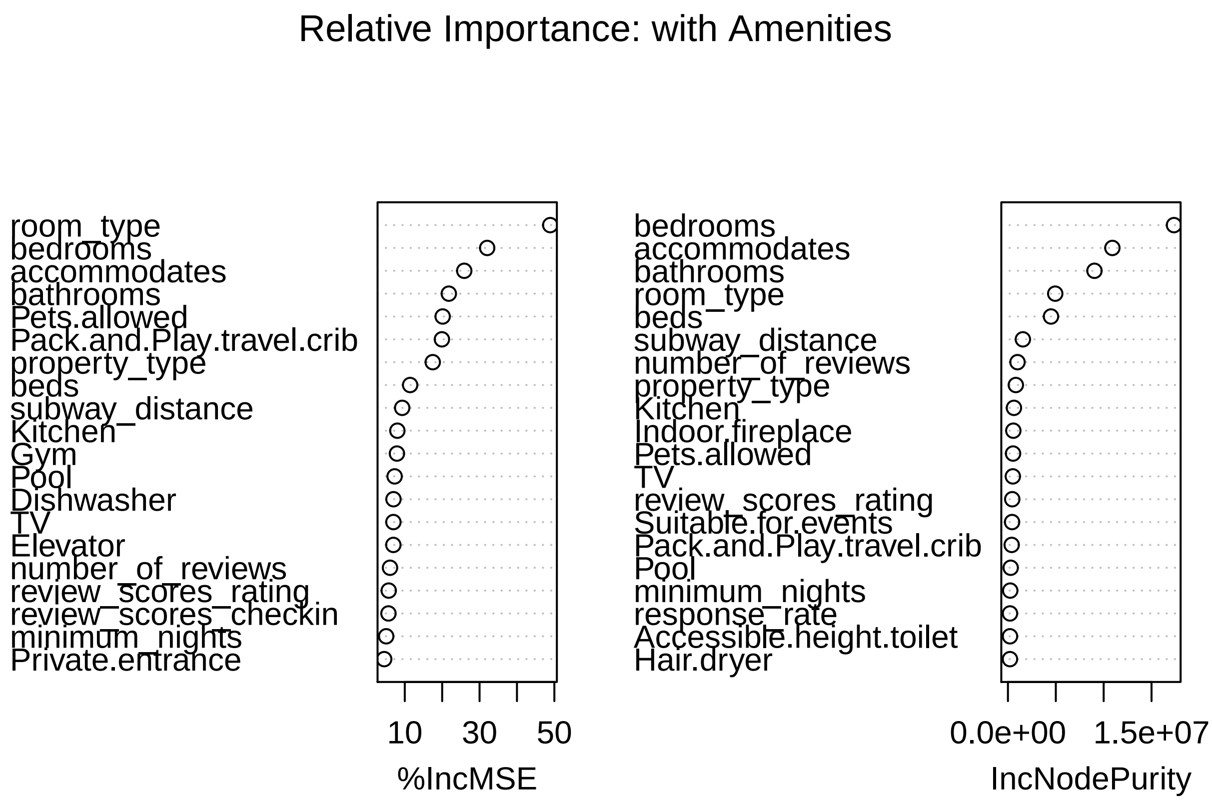
1. The most expensive properties are condos and lofts. This is because the listings for condos and lofts are mostly for entire place (over 87% of listings of both property types are for whole places), whereas for houses, only 37% of listings are for entire places.

2. Review scores on rating and cleanliness have positive effects on price when the ratings are high. This range is also where the majority of ratings reside (93.08% of all 'review scores ratings' are above 85 and 90.77% of reviews on cleanliness are 9 or 10). Thus, generally higher review scores correspond to higher prices. However, we should also note the negative effect on prices when the review scores are lower. This is rather interesting as one might expect a monotone relationship across the entire domain of review scores.

3. Subway distance has a negative impact on price when the distance is short. However, as the distance becomes longer, the effect becomes positive. This is a rather interesting finding, and perhaps we can only explain this in interaction with other terms.

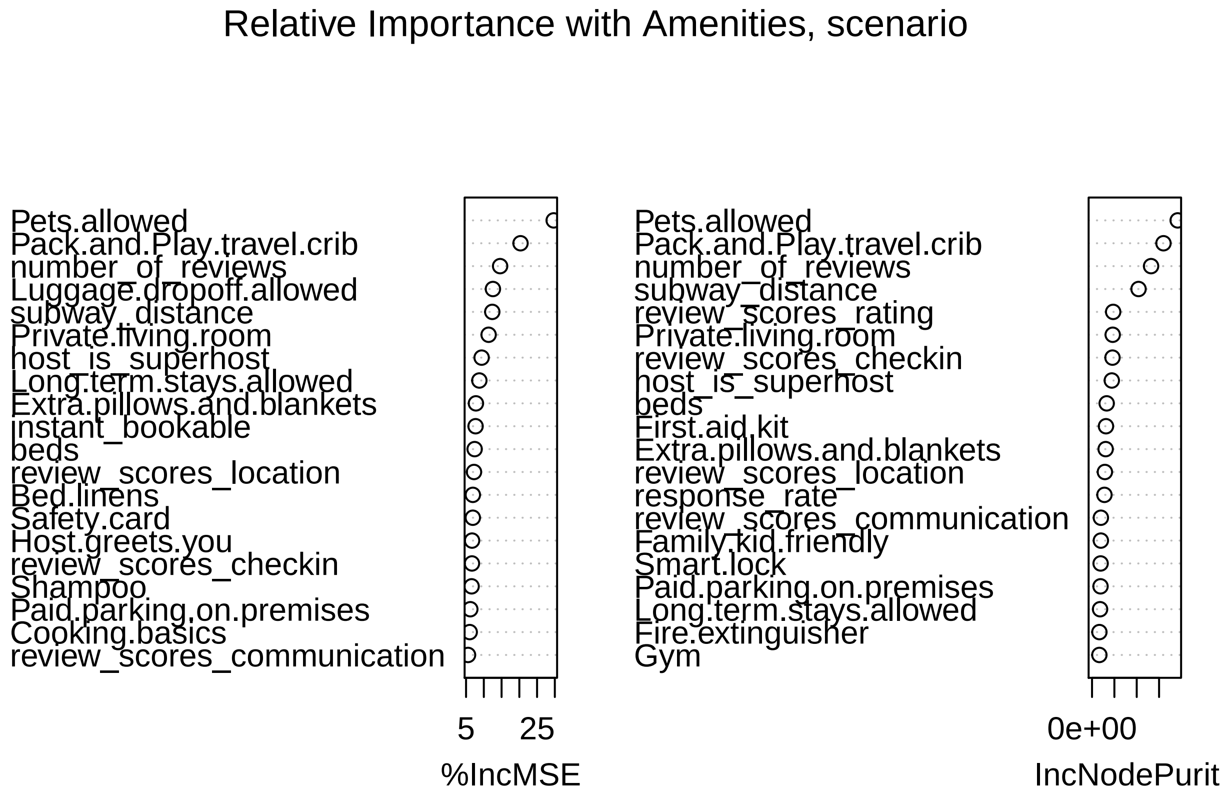
5.4 Variable Importance - with Amenities

We first build a random forest with ntree=500 and maxnode=84, but this time using m=p/2 rather than m=p/3 as we did in section 5.2. Using a larger m will return a better evaluation of relative importance. For the variable importance plot, we only look at the top 20 most important variables.

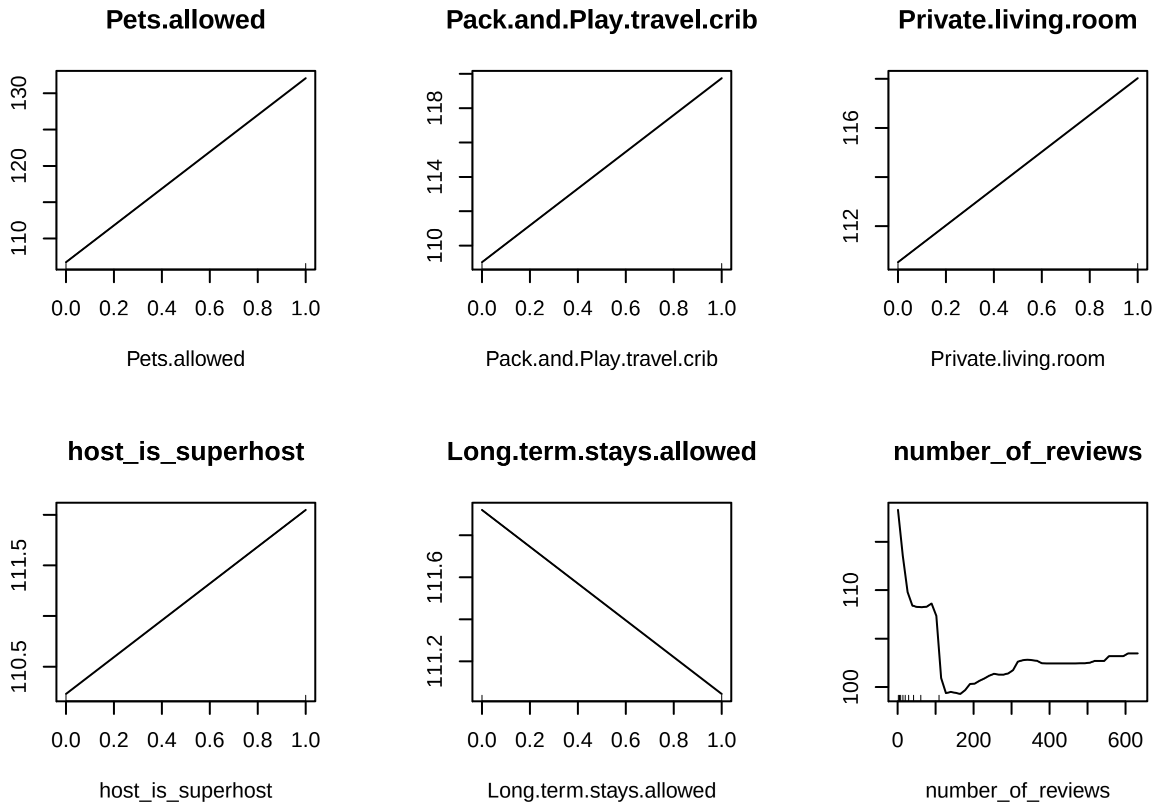


The top 4 variables are still room type, bedrooms, accommodates and bathrooms. On the other hand, a lot of amenities have entered the top 20 list.
For increase in MSE, `Pets.allowed` comes in 5th, followed by `Pack.and.Play.travel.crib`. These two variables both have similar importance as the number of bathrooms. From this observation we can conjecture that a lot of people are travelling with pets, and there are also many people who travel with babies. Other amenities in the top 20 include `kitchen`, `gym`, `pool`, `dishwasher`, `TV`, `elevator` and `private.entrance`.
In terms of reduction in RSS, we have `kitchen` as the most important amenity, followed by `indoor.fireplace`, `pets.allowed`, `TV`, `Pack and Play travel crib`, `Pool`, `Accessible height toilet` and `Hair dryer`.
Interestingly, the number of reviews becomes more important (in terms of RSS reduction) when we include amenities in our model.

We would like to further explore the relative important of amenities by fixing the important non-amenity variables. Imagine that a couple or a small group of friends (2 or 3 people) are looking to rent a one bedroom condo in Toronto for 3 nights. They start a search with property type set to condos, room type set to "Entire home/apt", number of bedrooms set to 1, accomodates set to at least 2, and they don't care about the number of bathrooms. What would the relative importance of amenities be in this scenario?
To accomplish the above task, we set the variables to their respective values (minimum nights set to at most 3), and drop the bathrooms column:



We also look at some of the partial dependency plots:



Observations (the below discussions apply only to the scenario we described):
1. It looks like many people in our scenario are likely to travel with pets and/or babies, as `pet.allowed` and `pack.and.play.travel.crib` are the most important variables using either criterion. If we look at the y scale of the two partial dependency plots, having these features will make the price much higher as well.
2. `number_of_reviews` becomes the 3rd most important feature. The relationship between price and number of reviews is interesting, with negative effect for smaller than 100 reviews, and positive effect afterwards. A possible explanation is that when the number of reviews is low, it is actually the lower price that attract more people and result in more reviews. In other words, when a place is overpriced, it attracts few customers and thus has few reviewers.
3. Surprisingly (and somewhat strangely), `private.living.room` is a rather important amenity, even when we're talking about renting entire condos. This does not make sense intuitively, since most condos should come with living rooms. The marginal effect (in terms of price different) is also quite significant.
4. `host.is.superhost` is relatively important. 'Superhost' is a program of Airbnb where they recognize exceptional hosts who meet certain standards.
On Airbnb's webpage `https://www.airbnb.ca/superhost` they claim that superhosts often make much more money. We have somewhat proven the part about making more money, but from our model it looks like the "much more" part is not true - the marginal effect is only $2 dollars.
5. `long.term.stays.allowed` are among the most important amenities, which indicates that quite some people use Airbnb to find longer stays instead of subletting. According to Airbnb, long term stays are stays that are longer than 2 weeks. However, the marginal effect is quiet small.

6 Boosting

We repeat the same four tasks with random forests, namely prediction and variable importance, both with and without amenities.

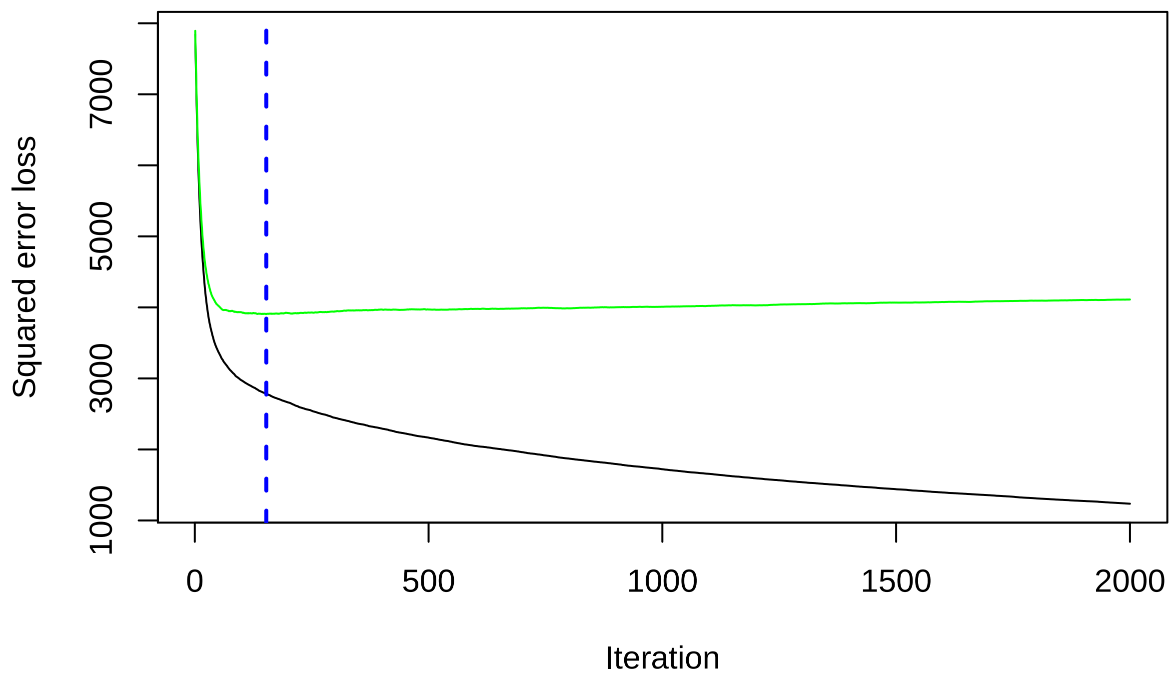
6.1 Prediction - without amenities

Note that for prediction, we use the training set to perform model selection with cv, and use the test set to evaluate performance.

For boosting, we have 4 hyperparameters to tune, namely d = interaction depth, ν = learning rate (aka shrinkage parameter), η = bag fraction, and M = number of trees.
We first fix number of trees at 1000. We would like to experiment (using 5 fold CV) values of d in (8, 9, 10, 11, 12), values of ν in (0.001, 0.005, 0.01, 0.05, 0.1), and values of η in (0.5, 0.6, 0.7, 0.8, 0.9). All combinations of each of these variables are experimented with, and the average cv error of each combination is recorded. We use the combination that produces the minimum average cv error:

```
##    inderaction depth shrinkage bag fraction cv error
## 19                 8      0.05          0.8 3599.777
```

We use these hyperparameters, then find the optimal number of trees (experimenting with more than 1000):




```
## [1] 153
```

The MSPE of this model on the test set is:

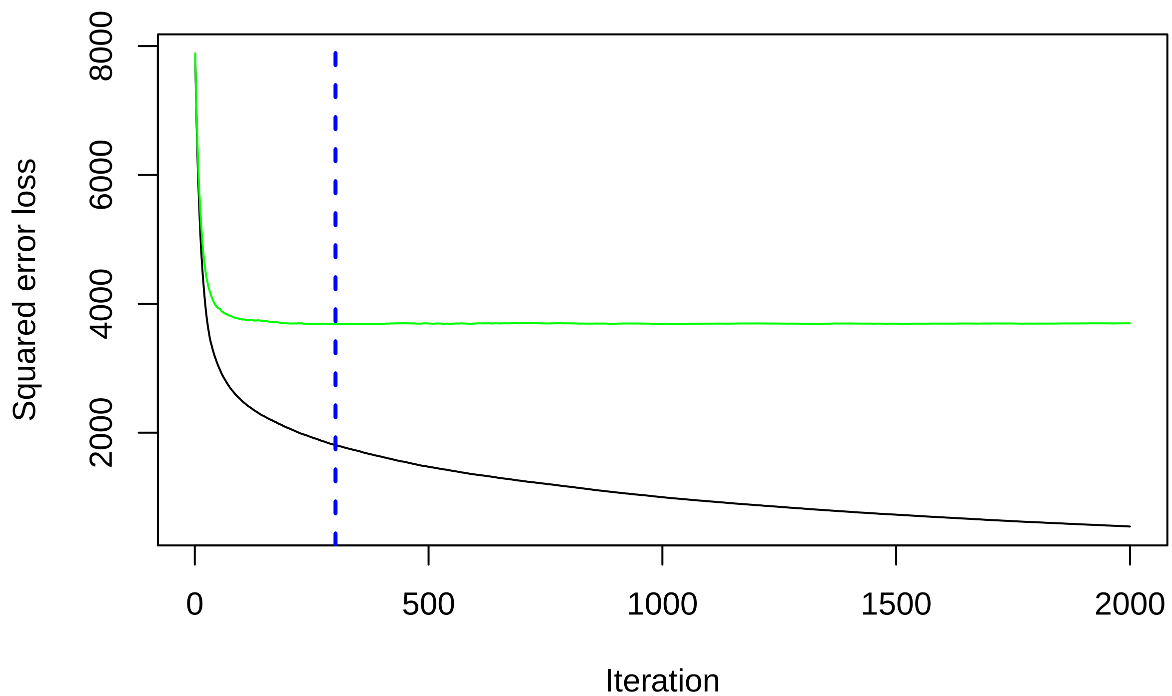
```
## [1] 4964.739
```

6.2 Prediction - with Amenities

We repeat the procedure above to find the optimal hyperparameters.

```
##    inderaction depth shrinkage bag fraction cv error
## 59                12      0.05          0.8 3372.262
```

Fixing these hyperparameters to find the optimal number of trees:




```
## [1] 301
```

Prediction sum of squares on the test set is The MSPE of this model on the test set is:

```
## [1] 3790.608
```

6.3 Variable Importance - No Amenities

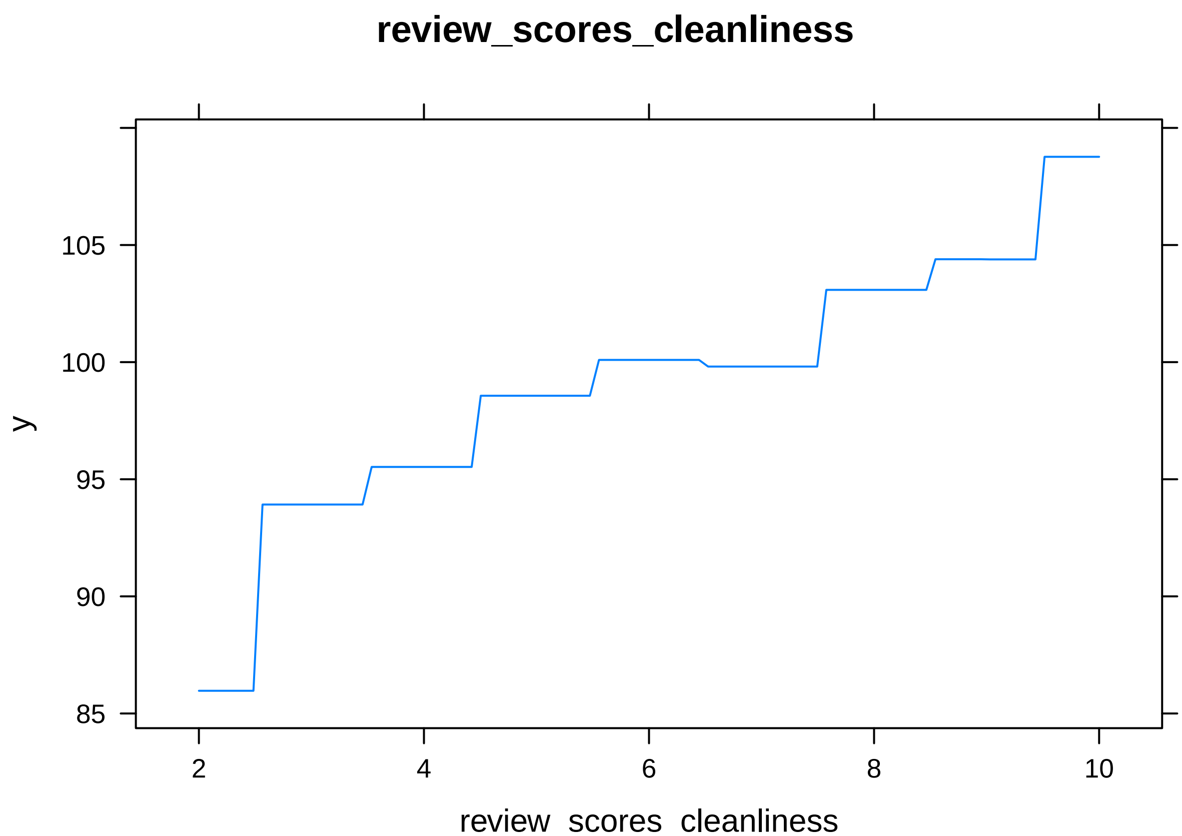
We obtain the following relative importance of variables (as measured by percentage reduction in RSS):

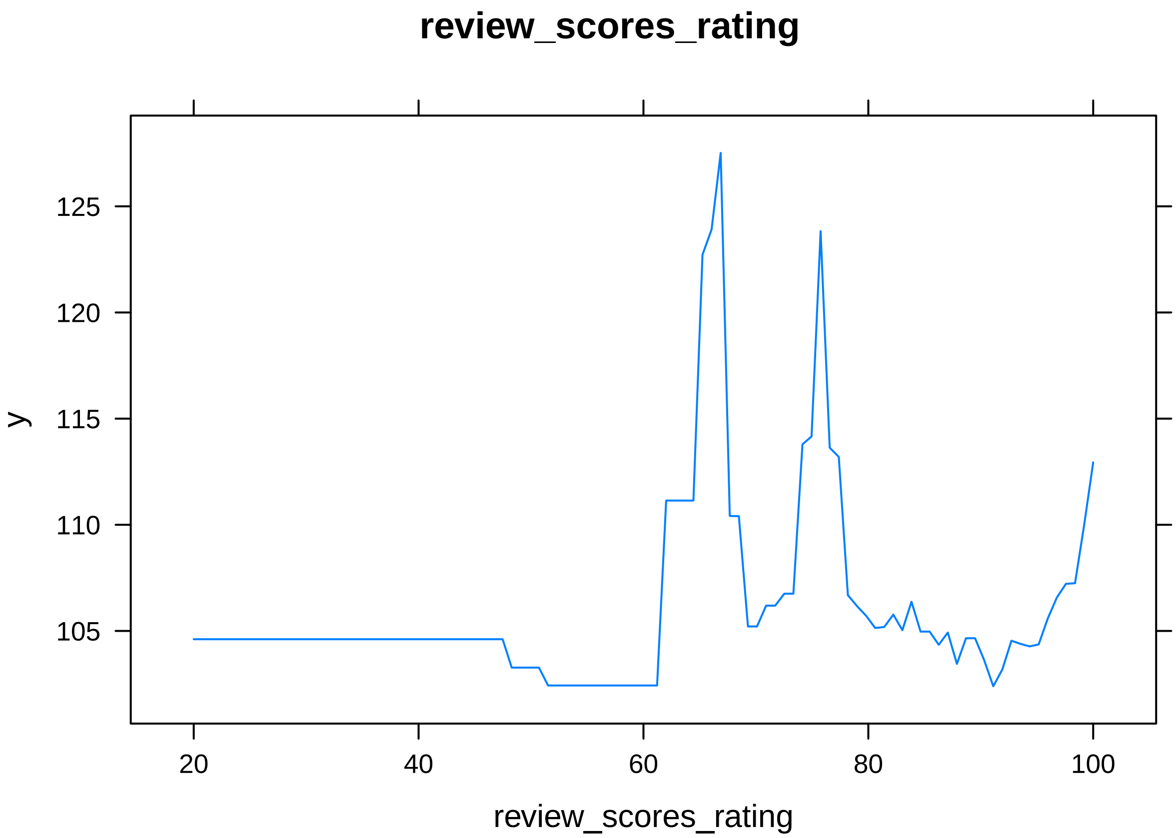
```
##                              var    rel.inf
## 1                       bedrooms 28.5308540
## 2                   accommodates 16.4108034
## 3                      bathrooms 13.1927355
## 4                subway_distance 10.2800820
## 5                      room_type  6.5048134
## 6              number_of_reviews  6.0665076
## 7           review_scores_rating  3.9607799
## 8                  property_type  3.1595705
## 9                           beds  2.7327402
## 10                minimum_nights  1.5906941
## 11   review_scores_cleanliness  1.5443509
## 12                 response_rate  1.3386831
## 13                 response_time  1.2788233
## 14              instant_bookable  1.0196611
## 15      review_scores_location  0.8760350
```

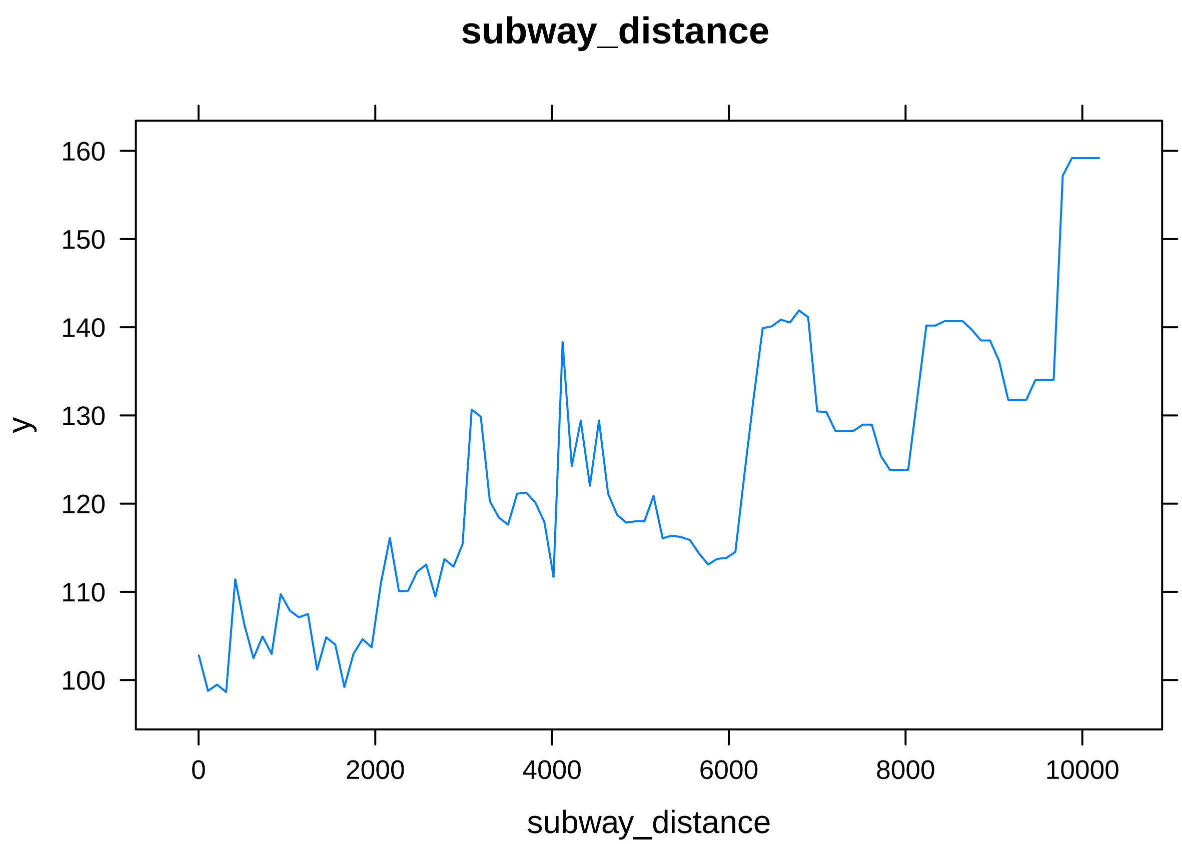
```
## 16           host_is_superhost  0.4648347
## 17       review_scores_checkin  0.4082385
## 18      review_scores_accuracy  0.3215071
## 19 review_scores_communication  0.3182856
```

This order of variable importance is quite similar to the one (RSS reduction) given by the random forest we built.
The top 4 variables are still `bedrooms`, `acomodates`, `bathrooms` and `room_type`, and they follow this exact order as they do in the random forest.
A notable difference is that the `beds` feature has dropped from 5th place (in random forest) to 8th place here.

We take a look at the partial dependence plots of the same variables we explored in section 5.3:




review_scores_rating
125
120
115
110
105
y
20
40
60
80
100
review_scores_rating



1. The marginal effect of review scores on cleanliness is very different than in random forest. The price is now (almost) monotonely increasing in cleanliness, and the spread of the y-axis is also much larger now.
2. The marginal effect of review scores on rating has a rather weird and complicated shape. Nonetheless we see a positive trend for high review scores of over 90.
3. The marginal effect of subway distance on price is generally positive, similar to what we have seen before, and again, very interesting but hard to explain.

6.4 Variable Importance - with amenities

We only look at the top 30 variables:

```
##                          var    rel.inf
## 1                   bedrooms 26.2031094
## 2               accommodates 13.4285970
## 3                  bathrooms 11.1126334
## 4            subway_distance  6.5109718
## 5                  room_type  5.8658439
## 6          number_of_reviews  4.4954010
## 7       review_scores_rating  2.8373254
## 8              property_type  2.4031216
## 9                       beds  1.8913485
## 10  review_scores_cleanliness  1.4333292
## 11           Indoor.fireplace  1.3388348
## 12             minimum_nights  1.2989978
## 13              response_rate  0.8941625
```

```
## 14             response_time  0.8271344
## 15        Family.kid.friendly  0.7323772
## 16           instant_bookable  0.7134504
## 17       Lock.on.bedroom.door  0.6658162
## 18                       Pool  0.6337666
## 19              First.aid.kit  0.6064097
## 20           Fire.extinguisher  0.5899580
## 21 Extra.pillows.and.blankets  0.5777951
## 22               Pets.allowed  0.5695917
## 23     review_scores_location  0.5594518
## 24           Private.entrance  0.5562100
## 25                         TV  0.5446521
## 26  Children.s.books.and.toys  0.4285684
## 27    Long.term.stays.allowed  0.3881392
## 28                 Beachfront  0.3793105
## 29                    Bathtub  0.3352898
## 30                 Dishwasher  0.3313349
```

The only amenity that made it to the top 10 is `indoor.fireplace`, which is a variable highly correlated with `property_type`. 21.7% of houses have fireplace, while only 7.8% of other properties have it.

7 Statistical Conclusions

In this section we compare the prediction powers (measured by MSPE) of the models and evaluate the best one:

7.1 Without Amenities

Model	Notes	MSPE
OLS	N/A	4573.585
Smoothing Splines	Using only 4 explanatory variables	10166.770
Random Forests	Maxnode=74, m=10	4520.163
Boosting	interaction depth = 8, learning rate = 0.05, bag fracion = 0.8	4964.739

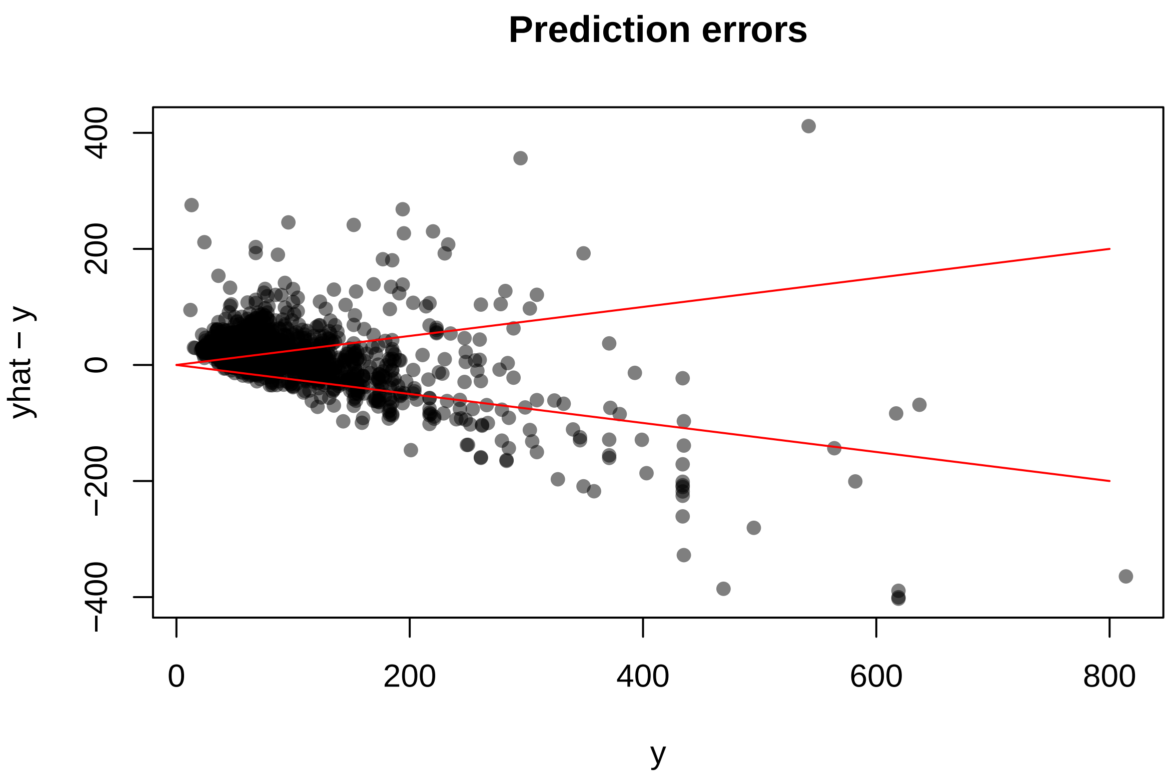
7.2 With Amenities

Model	Notes	MSPE
OLS	N/A	4266.040
Smoothing Splines	NA	NA
Random Forests	Maxnode=84, m=35	4098.838
Boosting	interaction depth = 12, learning rate = 0.05, bag fracion = 0.8	3790.608

The best model is produced by using boosting and including all amenities variables.
The mean absolute prediction error (measured by $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$) is

```
## [1] 39.2134
```

The following plot $(\hat{y} - y)$ v.s. y. The red lines denote $\pm 25\%$ error range:



The proportion of predictions that fall within the $\pm 25\%$ error range is:

```
## [1] 0.4137416
```

Points in the upper left corner are severely underpriced (these are listings like entire houses with 4 bedrooms at $13 a night) and the ones in the lower right corner are overpriced.

8 Practical Conclusions

We have built a model to predict an Airbnb's price, and this model has an avearge miss of $39.2.
This model can be used to

1. Calculate the (actual:predicted) ratio to obtain a measure of value (in terms of cost-effectiveness).
2. Detect outliers and inform owners if their places are severely over-priced or under-priced. For overpriced listings, owners may want to lower the price to attract more customers. For underpriced listings, many of them were accidentally set to a low price by the owners (like the 4-bedroom house at $13 a night... unless the owner is in for charity), so when they were booked at such low prices, the owners had to manually cancel the bookings. This creates additional cost for the owner, and also when people see that many bookings were cancelled for a place, they would not want to book it anymore. Thus our model can be used to create reminders for outlier listings.

It was found that the number of guests a place can accomodate has large impact on the price, even when we account for number of bedrooms. Perhaps owners can consider adding sofa beds or air mattresses in order to accomodate more guests and receive more income.
We have also discovered some amenities that have large importance when determining the price. For example,

a pet-friendly condo earns 20$ more than a non-pet friendly condo. Having ‘pack.and.play.travel.crib’ also increases price by around $10. For those owners who are hesitant to include certain amenities, these extra income could be a good motivataion.

9 Future Work

9.1 Data Processing

Our model is affected by outliers and there are quite a number of them, especially underpriced listings. We could have done a more detailed look at these listings and identify those that were accidentally set at a wrong number (by looking at its price a few months later and see if there’s any big changes).

9.2 Missing explanatory variables

Out model contains a huge set of explanatory variables (including 100 amenities), but we’re still missing some important ones that was not measured. For example, a suite on the 30-th floor of a condo will have a different price than a suite on the 1st floor of this building, because the rooms on higher floors offer better views. Some other important factors include how new a place is, how does it looks like, etc.
Some of these variables (such as design and decorations) are not directly measurable, but we can come up with ways to quantify them. For example, we can give pictures ratings from 1 to 10 in terms of moderness or design, and use this as one of our explanatory variables.

9.3 Explaining the unexpected relationship

We saw a number of partial dependence plots in which the relationships was not what we expected. In this project we did not try to explain these more complicated relationships (for example, increasing price in distance to subways, weird shapes of the marginal effects of review ratings, etc.). It would be valuable to further dig into these observations and come up with possible explanations.

10 Contributions

- Wenqi Wu:
    1. Manual selection of meaningful amenities
    2. Obtaining GPS locations of subway stations
    3. Obtaining housing indices of Toronto neighbourhoods
    4. Model analysis and writeup
- Jiawei Yu:
    1. Data preprocessing and visualizations
    2. Code writing
    3. Model analysis and writeup

Reference

Inside Airbnb. Adding data to the debate. (n.d.). Retrieved August 9, 2019, from http://insideairbnb.com/index.html

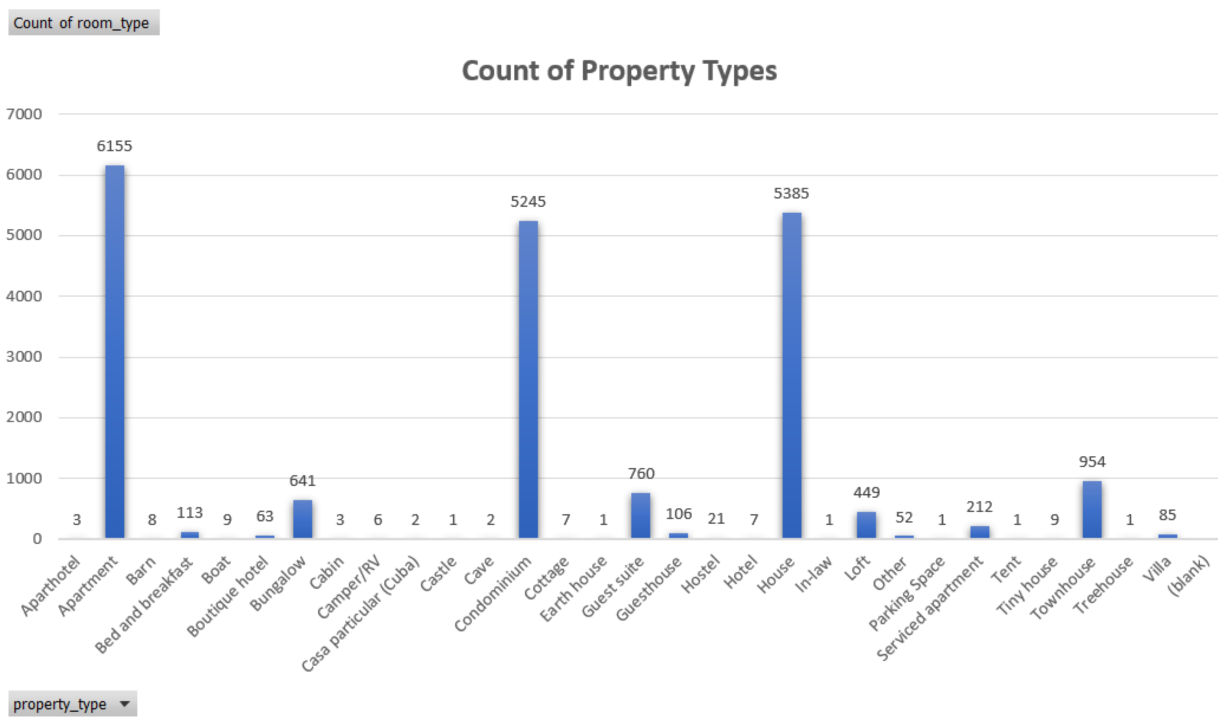
The Hottest Toronto Neighbourhoods. (n.d.). Retrieved August 9, 2019, from https://www.zolo.ca/toronto-real-estate/neighbourhoodsTTC

Operating Statistics. (n.d.). Retrieved August 9, 2019, from https://www.ttc.ca/About_the_TTC/Operating_Statistics/2017/section_two.jsp

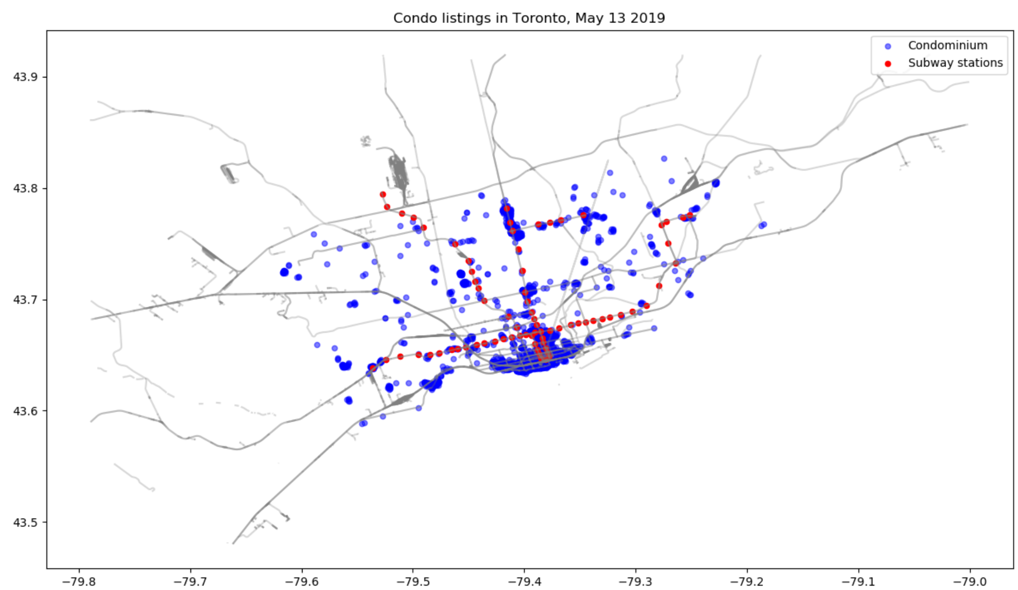
Vacation Homes & Condo Rentals. (n.d.). Retrieved August 9, 2019, from https://www.airbnb.ca/superhost

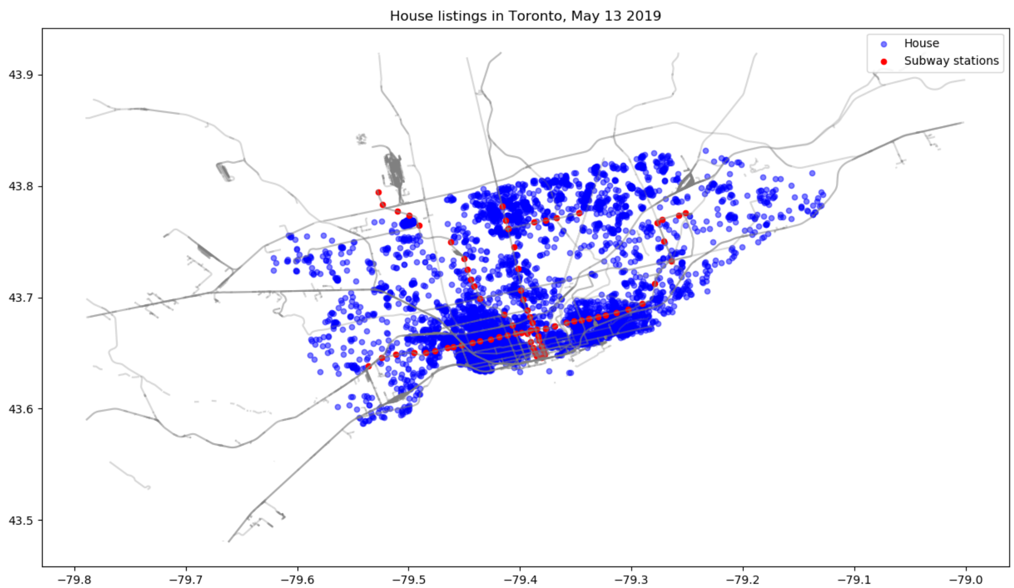
Appendix

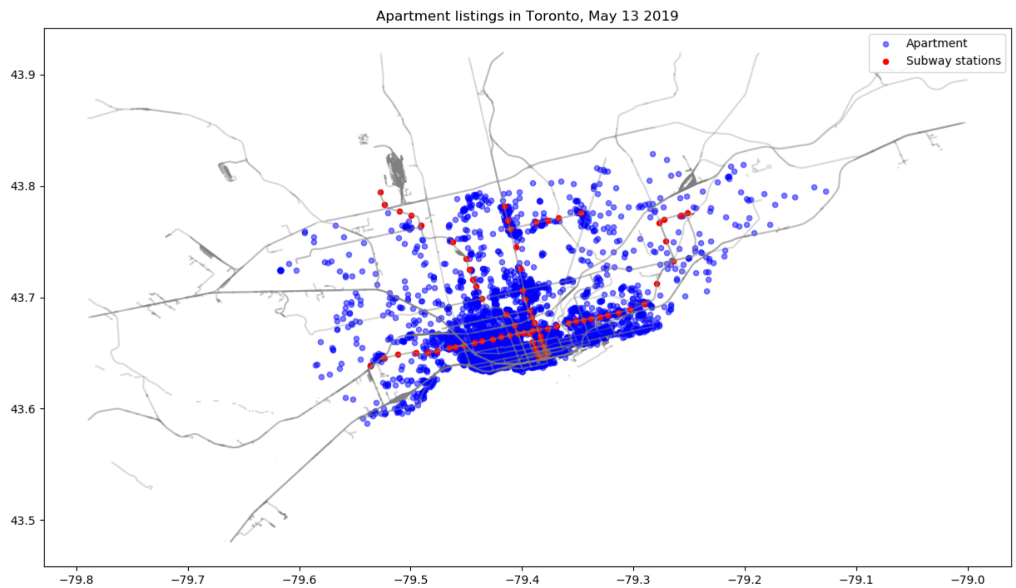
Appendix 1: Property types and Counts

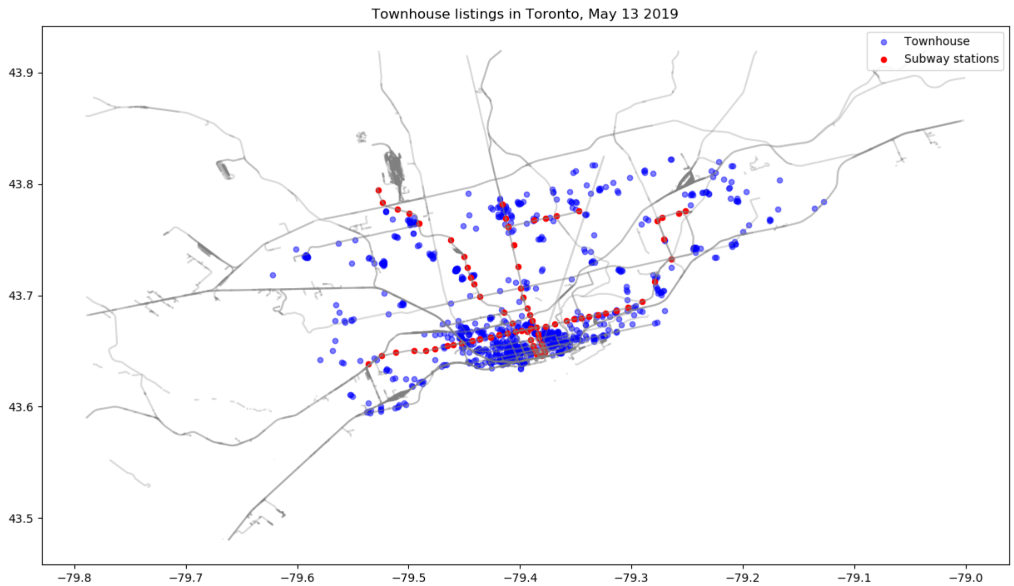


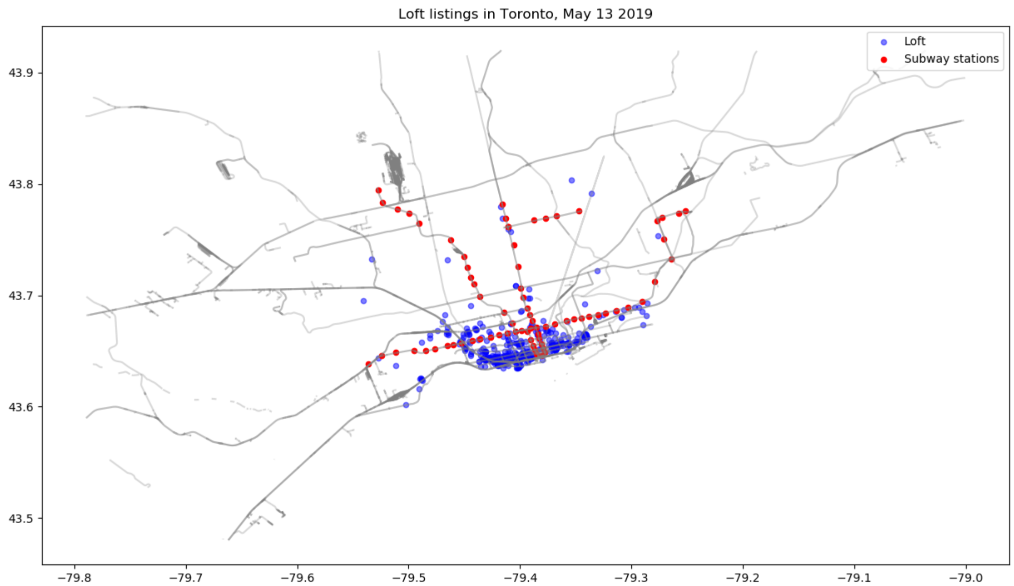
Appendix 2: Distribution of listings (by property type) across Toronto




House listings in Toronto, May 13 2019
House
Subway stations
43.9
43.8
43.7
43.6
43.5
−79.8
−79.7
−79.6
−79.5
−79.4
−79.3
−79.2
−79.1
−79.0


Apartment listings in Toronto, May 13 2019
Apartment
Subway stations
43.9
43.8
43.7
43.6
43.5
−79.8
−79.7
−79.6
−79.5
−79.4
−79.3
−79.2
−79.1
−79.0


Townhouse listings in Toronto, May 13 2019
Townhouse
Subway stations
43.9
43.8
43.7
43.6
43.5
−79.8
−79.7
−79.6
−79.5
−79.4
−79.3
−79.2
−79.1
−79.0


Loft listings in Toronto, May 13 2019
Loft
Subway stations
43.9
43.8
43.7
43.6
43.5
−79.8
−79.7
−79.6
−79.5
−79.4
−79.3
−79.2
−79.1
−79.0

Appendix 3: Amenities & Counts (Preprocessed Data)

There were 86 unique amenities selected.

24-hour check-in : 1985
Accessible-height toilet : 633
Air conditioning : 17365
BBQ grill : 1574
Baby bath : 182
Baby monitor : 72
Bathroom essentials : 969
Bathtub : 2437
Beachfront : 1389
Bed linens : 7519
Breakfast : 1820
Building staff : 827
Buzzer/wireless intercom : 1559
Cat(s) : 418
Children's books and toys : 1238
Cleaning before checkout : 904
Coffee maker : 6954
Cooking basics : 7778
Crib : 393
Disabled parking spot : 400
Dishes and silverware : 8322
Dishwasher : 5627
Dog(s) : 470
Doorman : 1147
Dryer : 16215
EV charger : 174
Elevator : 8466
Essentials : 19266
Ethernet connection : 1258
Extra pillows and blankets : 5647
Family/kid friendly : 6612
Fire extinguisher : 10511
First aid kit : 7724
Free parking on premises : 8145
Free street parking : 1989
Full kitchen : 392
Game console : 295
Garden or backyard : 2566
Patio or balcony : 4456
Gym : 6207
Hair dryer : 14860
Hangers : 17133
Heating : 19650
High chair : 732
Host greets you : 3207
Hot tub : 3136
Hot water kettle : 171
Indoor fireplace : 2025
Iron : 14245
Keypad : 2052

Kitchen : 18787
Lock on bedroom door : 6025
Lockbox : 3258
Long term stays allowed : 6324
Luggage dropoff allowed : 3828
Microwave : 8268
Netflix : 149
Other : 1627
Oven : 7945
Pack and Play/travel crib : 884
Paid parking off premises : 4676
Paid parking on premises : 2571
Pets allowed : 2575
Pets live on this property : 1076
Pool : 3770
Private entrance : 6680
Private living room : 2490
Refrigerator : 9008
Room-darkening shades : 1055
Safety card : 2747
Self check-in : 7003
Shampoo : 16167
Single level home : 1371
Smart TV : 114
Smart lock : 879
Smoke detector : 35224
Smoking allowed : 664
Stove : 8167
Suitable for events : 468
TV : 19118
Washer : 16482
Waterfront : 594
Well-lit path to entrance : 1868
Wheelchair accessible : 13244
Wifi : 25253
Window guards : 151

Code: R Code for Analysis

```r
# remove columns that we do not not use as explanatory variables
id.col <- which(names(toronto) == 'id')
price.col <- which(names(toronto) == 'price')
toronto <- toronto[,-c(id.col, price.col)]
price_counts <- rep(0, 7)
for (i in 1:7){
  price_counts[i] <- sum(toronto$normalized_price >= price_ranges[i] &
                         toronto$normalized_price < price_ranges[i+1])
}
barplot(price_counts, names.arg = labels, cex.names=0.75, xlab='Price(Adjusted) Range',
        ylab='frequency', main = 'Frequencies by price(adjusted) ranges',
        col='cornflowerblue')
```

```r
amenities <- names(toronto)[23:ncol(toronto)]
n_amenities <- length(amenities)
amenities_count = rep(n_amenities, 0)
for (i in 1:n_amenities){
  a = amenities[i]
  amenities_count[i] = sum(toronto[,i+22])
}
amenities <- amenities[order(amenities_count, decreasing=TRUE)]
amenities_count = sort(amenities_count,decreasing=TRUE)
barplot(amenities_count[1:8], names.arg = amenities[1:8], cex.names=0.71, xlab='Amenity',
        ylab='frequency', main = 'Most common amenities', col='chartreuse3')

amenities <- amenities[order(amenities_count)]
amenities_count = sort(amenities_count)
barplot(amenities_count[1:8], names.arg = amenities[1:8], cex.names=0.55, xlab='Amenity',
        ylab='frequency', main = 'Rarest amenities', col='aquamarine3')
```

```r
train <- toronto[1:10000,]
test <- toronto[10001:nrow(toronto), ]
lr1 <- lm(normalized_price~., data=train)
sMSE.1 <- mean((lr1$fitted.values - train$normalized_price)^2)
MSPE.1 <- mean((predict(lr1, newdata=test) - test$normalized_price)^2)
```

```r
c(sMSE.1, MSPE.1)
```

```r
train.sparse <- train[,1:20]
test.sparse <- test[,1:20]
lr2 <- lm(normalized_price~., data=train.sparse)
summary(lr2)
sMSE.2 <- mean((lr2$fitted.values - train$normalized_price)^2)
MSPE.2 <- mean((predict(lr2, newdata=test.sparse) - test.sparse$normalized_price)^2)
```

```r
c(sMSE.2, MSPE.2)
```

```r
par(mfrow=c(1,2))
standard.res <- rstandard(lr2)
plot(lr2$residuals~predict(lr2), main='Residuals V.S. Fitted')
plot(standard.res~predict(lr2), main='Standardized Resid. V.S. Fitted')
abline(h=1.96)
abline(h=-1.96)
```

```r
library(mgcv)
set.seed(1000)
subset <- train[train$property_type == 'Condominium' &
                  train$room_type == 'Entire home/apt' & train$minimum_nights == 1,]
subset <- subset[sample(nrow(subset)),]
valid_size <- nrow(subset)%/%5

cv.errors.interaction <- rep(0,5)
cv.errors.no_interaction <- rep(0,5)

for (i in 0:4){
  subset.test <- subset[(i*valid_size + 1) : min((i+1)*valid_size, nrow(subset)),]
  if (i == 0) {
    subset.train <- subset[(valid_size+1) : nrow(subset),]
```

```r
  } else if (i == 4) {
    subset.train <- subset[(4*valid_size + 1) : nrow(subset), ]
  } else if (i == 4) {
    subset.train <- subset[c((1:i*valid_size), ((i+1)*valid_size : nrow(subset))), ]
  }
  s1 <- gam(normalized_price ~ s(bathrooms, k=1) +
              s(bedrooms, k=1) + s(number_of_reviews, bs='cr') +
              s(subway_distance,bs='cr') + ti(bathrooms, bedrooms), data=subset.train)
  s2 <- gam(normalized_price ~ s(bathrooms, k=1) +
              s(bedrooms, k=1) + s(number_of_reviews, bs='cr') +
              s(subway_distance,bs='cr'), data=subset.train)

  cv.errors.interaction[i+1] <- mean((predict(s1, newdata=subset.test)
                                      - subset.test$normalized_price)^2)
  cv.errors.no_interaction[i+1] <- mean((predict(s2, newdata=subset.test)
                                         - subset.test$normalized_price)^2)
}
```

```r
cv.errors.interaction
```

```r
cv.errors.no_interaction
```

```r
c(mean(cv.errors.interaction), mean(cv.errors.no_interaction))
```

```r
ss <- gam(normalized_price ~ s(bathrooms, k=1) + s(bedrooms, k=1) +
            s(number_of_reviews, bs='cr') +
            s(subway_distance,bs='cr'), data=subset)
summary(ss)
```

```r
test.subset <- test[test$property_type == 'Condominium' & test$room_type == 'Entire home/apt'
                    & test$minimum_nights == 1,]
MSPE <- mean((predict(ss, newdata=test.subset)-test.subset$normalized_price)^2)
MSPE
```

```r
lr.3 <- lm(normalized_price ~ bathrooms + bedrooms + number_of_reviews + subway_distance,
           data=subset)
MSPE <- mean((predict(lr.3, newdata=test.subset) - test.subset$normalized_price)^2)
MSPE
```

```r
library(randomForest)
p <- ncol(train.sparse) - 1

set.seed(444)
OOB_errs <- rep(0, 8)
max_nodes <- c(5, 15, 25, 35, 45, 55, 65, 75)

par(mfrow=c(2,2))
for (i in 1:length(max_nodes)){
  n <- max_nodes[i]
  rf <- randomForest(data=train.sparse, normalized_price~., importance=FALSE,
                     ntree=500, mtry=p/3, keep.forest=TRUE, maxnodes=n)
  plot(rf, main=paste('maxnode=', n))
  OOB_errs[i] <- mean((rf$predicted - train.sparse$normalized_price)^2)
}
```

```r
names(OOB_errs) <- max_nodes
OOB_errs
```

```r
set.seed(444)
OOB_errs <- rep(0, 6)
max_nodes <- c(56, 59, 62, 65, 68, 71, 74)
for (i in 1:length(max_nodes)){
  n <- max_nodes[i]
  rf <- randomForest(data=train.sparse, normalized_price~., importance=FALSE,
                     ntree=500, mtry=p/3, keep.forest=TRUE, maxnodes=n)
  OOB_errs[i] <- mean((rf$predicted - train.sparse$normalized_price)^2)
}
names(OOB_errs) <- max_nodes
OOB_errs
```

```r
set.seed(444)
OOB_errs <- rep(0, 6)
ms <- c(1, round(p/5), round(p/4), round(p/3), round(p/2), p)
for (i in 1:length(ms)){
  m <- ms[i]
  rf <- randomForest(data=train.sparse, normalized_price~., importance=FALSE,
                     ntree=500, mtry=m, keep.forest=TRUE, maxnodes=74)
  OOB_errs[i] <- mean((rf$predicted - train.sparse$normalized_price)^2)
}
```

```r
names(OOB_errs) <- c(1, 'p/5', 'p/4', 'p/3', 'p/2', 'p')
OOB_errs
```

```r
set.seed(100)
rf <- randomForest(data=train.sparse, normalized_price~., importance=FALSE,
                   ntree=1000, mtry=round(p/2), keep.forest=TRUE, maxnodes=74)
MSPE <- mean((predict(rf, newdata=test.sparse) - test.sparse$normalized_price)^2)
MSPE
```

```r
p <- ncol(train) - 1

set.seed(444)
OOB_errs <- rep(0, 8)
max_nodes <- c(15, 25, 35, 45, 55, 65, 75, 85)

par(mfrow=c(2,2))
for (i in 1:length(max_nodes)){
  n <- max_nodes[i]
  rf <- randomForest(data=train, normalized_price~., importance=FALSE,
                     ntree=500, mtry=p/3, keep.forest=TRUE, maxnodes=n)
  plot(rf, main=paste('maxnode=', n))
  OOB_errs[i] <- mean((rf$predicted - train$normalized_price)^2)
}
```

```r
names(OOB_errs) <- max_nodes
OOB_errs
```

```r
set.seed(444)
OOB_errs <- rep(0, 6)
max_nodes <- c(66, 69, 72, 75, 78, 81, 84)
for (i in 1:length(max_nodes)){
```

```r
  n <- max_nodes[i]
  rf <- randomForest(data=train, normalized_price~., importance=FALSE,
                     ntree=500, mtry=p/3, keep.forest=TRUE, maxnodes=n)
  OOB_errs[i] <- mean((rf$predicted - train$normalized_price)^2)
}
names(OOB_errs) <- max_nodes
OOB_errs
```

```r
set.seed(444)
OOB_errs <- rep(0, 6)
ms <- c(1, round(p/5), round(p/4), round(p/3), round(p/2), p)
for (i in 1:length(ms)){
  m <- ms[i]
  rf <- randomForest(data=train, normalized_price~., importance=FALSE,
                     ntree=500, mtry=m, keep.forest=TRUE, maxnodes=84)
  OOB_errs[i] <- mean((rf$predicted - train$normalized_price)^2)
}
```

```r
names(OOB_errs) <- c(1, 'p/5', 'p/4', 'p/3', 'p/2', 'p')
OOB_errs
```

```r
set.seed(100)
rf <- randomForest(data=train, normalized_price~., importance=FALSE,
                   ntree=500, mtry=round(p/3), keep.forest=TRUE, maxnodes=84)
MSPE <- mean((predict(rf, newdata=test) - test$normalized_price)^2)
MSPE
```

```r
set.seed(100)
toronto.sparse <- toronto[,1:20]
p <- ncol(toronto.sparse) - 1
rf <- randomForest(data=toronto.sparse, normalized_price~., importance=TRUE,
                   ntree=500, mtry=round(p/2), keep.forest=TRUE, maxnodes=74)
```

```r
library(randomForest)
varImpPlot(rf, main='Relative Importance: Without Amenities')
```

```r
importance(rf)[order(importance(rf)[,1], decreasing = TRUE),]
```

```r
partialPlot(rf, pred.data=toronto.sparse, x.var=property_type, main='property_type', cex.names = 0.7)
```

```r
par(mfrow=c(1,3))
partialPlot(rf, pred.data=toronto.sparse, x.var=review_scores_checkin,
            main='review_scores_checkin')
partialPlot(rf, pred.data=toronto.sparse, x.var=review_scores_rating,
            main='review_scores_rating')
partialPlot(rf, pred.data=toronto.sparse, x.var=subway_distance,
            main='subway_distance')
```

```r
set.seed(100)
p <- ncol(toronto) - 1
rf <- randomForest(data=toronto, normalized_price~., importance=TRUE,
                   ntree=500, mtry=round(p/2), keep.forest=TRUE, maxnodes=84)
```

```r
varImpPlot(rf, main='Relative Importance: with Amenities', n.var=20)
```

```r
set.seed(100)
scenario <- toronto[toronto$property_type == 'Condominium' & toronto$bedrooms == 1
                    & toronto$accommodates >= 2 & toronto$room_type == 'Entire home/apt' &
                      toronto$minimum_nights <= 3,]
scenario <- subset(scenario, select=-c(property_type, bedrooms, accommodates,
                                       bathrooms, room_type, minimum_nights))
p <- ncol(scenario) - 1
rf <- randomForest(data=scenario, normalized_price~., importance=TRUE,
                   ntree=500, mtry=round(p/2), keep.forest=TRUE, maxnodes=84)
```

```r
varImpPlot(rf, main='Relative Importance with Amenities, scenario', n.var=20)
```

```r
par(mfrow=c(2,3))
partialPlot(rf, pred.data=scenario, x.var=Pets.allowed, main='Pets.allowed')
partialPlot(rf, pred.data=scenario, x.var=Pack.and.Play.travel.crib,
            main='Pack.and.Play.travel.crib')
partialPlot(rf, pred.data=scenario, x.var=Private.living.room,
            main='Private.living.room')
partialPlot(rf, pred.data=scenario, x.var=host_is_superhost,
            main='host_is_superhost')
partialPlot(rf, pred.data=scenario, x.var=Long.term.stays.allowed,
            main='Long.term.stays.allowed')
partialPlot(rf, pred.data=scenario, x.var=number_of_reviews, main='number_of_reviews')
```

```r
library(gbm)
set.seed(100)
p <- ncol(train.sparse) - 1
cv.errors <- matrix(data=NA, nrow=5**3, ncol=4)
irow <- 1
for (d in 8:12){
  for (nu in c(0.001, 0.005, 0.01, 0.05, 0.1)){
    for (eta in c(0.5, 0.6, 0.7, 0.8, 0.9)) {
      boost <- gbm(data = train.sparse, normalized_price~., distribution = 'gaussian',
                   n.trees=1000, interaction.depth = d, shrinkage=nu,
                   bag.fraction=eta, cv.folds=5)
      cv.errors[irow,] <- c(d, nu, eta, min(boost$cv.error))
      irow <- irow + 1
    }
  }
}
```

```r
cv.errors <- data.frame(cv.errors)
names(cv.errors) <- c('inderaction depth', 'shrinkage', 'bag fraction', 'cv error')
min.row <- cv.errors[which(cv.errors$'cv error' == min(cv.errors$'cv error')),]
min.row
```

```r
library(gbm)
set.seed(100)
d = min.row[1]
nu = min.row[2]
eta = min.row[3]
boost <- gbm(data = train.sparse, normalized_price~., distribution = 'gaussian',
                   n.trees=2000, interaction.depth = d, shrinkage=nu,
                   bag.fraction=eta, cv.folds=5)
gbm.perf(boost, method='cv')
```

```r
library(gbm)
n.trees <- which(boost$cv.error == min(boost$cv.error))
MSPE <- mean((predict(boost, newdata = test.sparse, n.trees = n.trees) -
                test.sparse$normalized_price)^2)
MSPE
```

```r
library(gbm)
set.seed(100)
p <- ncol(train) - 1
cv.errors.2 <- matrix(data=NA, nrow=4**3, ncol=4)
irow <- 1
for (d in 9:12){
  for (nu in c(0.005, 0.01, 0.05, 0.1)){
    for (eta in c(0.6, 0.7, 0.8, 0.9)) {
      # print(irow)
      boost.2 <- gbm(data = train, normalized_price~., distribution = 'gaussian',
                     n.trees=1000, interaction.depth = d, shrinkage=nu,
                     bag.fraction=eta, cv.folds=5)
      cv.errors.2[irow,] <- c(d, nu, eta, min(boost.2$cv.error))
      irow <- irow + 1
    }
  }
}
```

```r
cv.errors.2 <- data.frame(cv.errors.2)
names(cv.errors.2) <- c('inderaction depth', 'shrinkage', 'bag fraction', 'cv error')
min.row.2 <- cv.errors.2[which(cv.errors.2$'cv error' == min(cv.errors.2$'cv error')),]
min.row.2
```

```r
set.seed(100)
d = min.row.2[1]
nu = min.row.2[2]
eta = min.row.2[3]
boost.2 <- gbm(data = train, normalized_price~., distribution = 'gaussian',
                   n.trees=2000, interaction.depth = d, shrinkage=nu,
                   bag.fraction=eta, cv.folds=5)
gbm.perf(boost.2, method='cv')
```

```r
library(gbm)
n.trees <- which(boost.2$cv.error == min(boost.2$cv.error))
MSPE <- mean((predict(boost.2, newdata = test, n.trees = n.trees) -
                test$normalized_price)^2)
MSPE
```

```r
imp.1 <- summary(boost, plotit = FALSE)
rownames(imp.1) <- 1:19
imp.1
```

```r
par(mfrow=c(1,3))
plot(boost, i.var=15, main='review_scores_checkin')
plot(boost, i.var=12, main='review_scores_rating')
plot(boost, i.var=19, main='subway_distance')
```

```r
imp.2 <- summary(boost.2, plotit = FALSE)
imp.2 <- imp.2[1:30,]
rownames(imp.2) <- 1:30
```

```r
imp.2
```

```r
library(gbm)
l1_errors <- abs(predict(boost.2, newdata = test, n.trees = n.trees) -
                   test$normalized_price)
MAPE <- mean(l1_errors)
pct_error <- l1_errors/test$normalized_price
MAPE
```

```r
plot(y=predict(boost.2, newdata = test, n.trees = n.trees) - test$normalized_price,
     x=test$normalized_price, main = 'Prediction errors',
     ylab = 'yhat - y', xlab='y', col = adjustcolor('black',alpha=0.5), pch=16)
lines(x=c(0,800), y=c(0,200), col='red')
lines(x=c(0,800), y=c(0,-200), col='red')
```

```r
l1_errors_pct <- l1_errors / test$normalized_price
mean(l1_errors_pct <= 0.25)
```

Python Code for Preprocessing

```
"""
STAT 444 Final Project
Data Preprocessing
"""

from collections import defaultdict

import pandas as pd
import numpy as np
from geopy.distance import distance
from tqdm import tqdm


"""
Cleaning (Removal of bad data or missing data)
"""
overpriced = [12068731, 13379170, 25032692, 17330866]


def clean(data):
    data = data[data['availability_365'] > 0]
    data = data[data['number_of_reviews'] > 0]
    data = data[(data['id'] != overpriced[0]) & (data['id'] != overpriced[1])
                & (data['id'] != overpriced[2]) & (data['id'] != overpriced[3])]
    data = data[data['price'] > 0]

    return data


def remove_missing(data):
    for col in data.columns:
        data = data[pd.notnull(data[col])]
    return data


"""
Processing Amenities
"""


def trim_string(string):
    if len(string) == 0:
        return ''
    while string[0] in ['"', '{', ' ']:
        string = string[1:]
        if len(string) == 0:
            return ''
    while string[-1] in ['"', '}', ' ']:
        string = string[:-1]
        if len(string) == 0:
            return ''
```

```
    return string


def process_amenities_string(amenities):
    amenities = trim_string(amenities)
    amenities = amenities.split(',')
    amenities = [trim_string(s) for s in amenities]
    amenities = ["Children's books and toys"
    if s.startswith('Children') else s for s in amenities]
    amenities = ["Pack and Play/travel crib"
    if s.startswith('Pack') else s for s in amenities]
    return amenities


def count_amenities(data):
    amenities_lists = data['amenities']
    all_amenities = defaultdict(int)
    for amenities in amenities_lists:
        amenities = process_amenities_string(amenities)
        for a in amenities:
            all_amenities[a] += 1

    i = 0
    for a in sorted(all_amenities.keys()):
        print('{}: {}  '.format(a, all_amenities[a]))
        if all_amenities[a] >= 1:
            i += 1
    print(i)


def process_amenities(data):
    amenities_file = 'amenities.txt'
    # a txt file that contains the manually selected amenities to be included
    amenities_lists = data['amenities']
    selected_amenities = []
    with open(amenities_file, 'r') as infile:
        for line in infile:
            line = line.split(',')
            counts = [int(item.split(':')[1]) for item in line]
            line = [item.split(':')[0] for item in line]
            print('{} : {}  '.format(line[0], sum(counts)))
            selected_amenities.append(line)
    print('{} amenities selected'.format(len(selected_amenities)))

    for amenity in tqdm(selected_amenities, 'amenities'):
        has_feature = [1 if len(set(amenity).intersection(process_amenities_string(str))) > 0
                       else 0 for str in amenities_lists]
        data[amenity[0]] = has_feature

    return data


"""
```

```python
Only keep condos, houses, apartments, townhouses, bungalows, lofts and guest suites
"""


def process_property_types(data):
    data = data[(data['property_type'] == 'House') |
    (data['property_type'] == 'Apartment') |
    (data['property_type'] == 'Condominium') |
    (data['property_type'] == 'Bungalow') |
    (data['property_type'] == 'Townhouse') |
    (data['property_type'] == 'Guest suite') |
    (data['property_type'] == 'Loft')]
    return data


"""
Location - distance to nearest subway station
"""
subway_GPS = "./TorontoSubwayGPS.csv"


def distance_to_subway(data):
    lat, lon = data['latitude'], data['longitude']
    loc = list(zip(lat, lon))
    subway_gps_data = pd.read_csv(subway_GPS)
    subway_lat, subway_lon = subway_gps_data['latitude'], subway_gps_data['longitude']
    subway_loc = list(zip(subway_lat, subway_lon))
    subway_distance = []
    for place in tqdm(loc, 'listings'):
        shortest_dist = -1
        for subway in subway_loc:
            dist = distance(subway, place).m
            if shortest_dist == -1 or dist < shortest_dist:
                shortest_dist = dist
        subway_distance.append(int(shortest_dist))

    data['subway_distance'] = subway_distance
    return data


"""
Location - normalize prices by local factor
"""


def normalize_price(data):
    neigh_data = pd.read_csv('neighbourhoods_cost.csv')
    neighborhoods, houses, condos = neigh_data['neighbourhood'].tolist(), \
                                    neigh_data['3-bed house'].tolist(),\
                                    neigh_data['2-bed condo'].tolist()
    costs = {}
    for i, n in enumerate(neighborhoods):
        if not np.isnan(houses[i]):
```

```
            costs[n] = houses[i]
        else:
            costs[n] = 2 * condos[i]
    mean_price = sum(costs.values())/len(costs.values())
    for key in costs:
        costs[key] = costs[key]/mean_price

    neighborhoods, prices = data['neighbourhood'].tolist(), data['price'].tolist()
    normalized_price = [int(prices[i]/costs[neighborhoods[i]]) for i in range(len(prices))]
    data['normalized_price'] = normalized_price

    return data


"""
Converts categorical values of t and f to 1 and 0
"""


def process_bool(data):
    bool_variables = ['host_is_superhost', 'instant_bookable']
    for var in bool_variables:
        values = data[var].tolist()
        new_values = [1 if v == 't' else 0 for v in values]
        data[var] = new_values
    return data


"""
Perform all preprocessing
"""


def preprocess(data):
    data = clean(data)
    data = process_property_types(data)
    data = normalize_price(data)
    data = distance_to_subway(data)
    data = process_bool(data)
    data = process_amenities(data)
    data = data.drop(['amenities', 'latitude', 'longitude', 'review_scores_value',
                      'neighbourhood', 'listing_url', 'availability_365'], axis=1)
    data = remove_missing(data)
    return data


if __name__ == '__main__':
    data = pd.read_csv('toronto_selected_features.csv', encoding='latin1')
    data = preprocess(data)
    data.to_csv('./preprocessed.csv', encoding='latin1', index=False)
```

Python Code for Visualizations

```
"""
Stat 444 Final Project
Data Visualizations
"""

import geopandas as gpd
from shapely.geometry import Point, Polygon
import descartes
import matplotlib.pyplot as plt
import pandas as pd


"""
Reads Data and Preprocessing
"""
data = pd.read_csv('toronto_selected_features.csv', encoding='latin1')


"""
Plot listings on map
"""
# Reads subway locations
subway_gps_data = pd.read_csv("./TorontoSubwayGPS.csv")
subway_lat, subway_lon = subway_gps_data['latitude'], subway_gps_data['longitude']
geometry = [Point(xy) for xy in zip(subway_lon, subway_lat)]
subway_df = gpd.GeoDataFrame(pd.DataFrame({'lat':subway_lat}),
crs={'init': 'epsg:4326'}, geometry=geometry)

# Reads data and map shapes
toronto_map = gpd.read_file('Toronto-shp/shape/railways.shp')
geometry = [Point(xy) for xy in zip(data['longitude'], data['latitude'])]
geo_df = gpd.GeoDataFrame(data, crs={'init': 'epsg:4326'}, geometry=geometry)

# plot houses
fig,ax = plt.subplots(figsize=(15, 15))
toronto_map.plot(ax=ax, alpha=0.3, color='grey')
geo_df[geo_df['property_type'] == 'House'].plot(
    ax=ax, markersize=20, alpha=0.5, color='blue', marker='o', label='House')
subway_df.plot(ax=ax, markersize=20, alpha=1, color='red', marker='o',
label='Subway stations')
plt.title('House listings in Toronto, May 13 2019')
plt.legend()
plt.savefig('Houses.png')

# Plot condos
plt.figure()
fig,ax = plt.subplots(figsize=(15, 15))
toronto_map.plot(ax=ax, alpha=0.3, color='grey')
geo_df[geo_df['property_type'] == 'Condominium'].plot(
    ax=ax, markersize=20, alpha=0.5, color='blue', marker='o',
    label='Condominium')
```

```
subway_df.plot(ax=ax, markersize=20, alpha=1, color='red', marker='o',
label='Subway stations')
plt.title('Condo listings in Toronto, May 13 2019')
plt.legend()
plt.savefig('Condos.png')

# Plot apartments
plt.figure()
fig,ax = plt.subplots(figsize=(15, 15))
toronto_map.plot(ax=ax, alpha=0.3, color='grey')
geo_df[geo_df['property_type'] == 'Apartment'].plot(
    ax=ax, markersize=20, alpha=0.5, color='blue', marker='o',
    label='Apartment')
subway_df.plot(ax=ax, markersize=20, alpha=1, color='red', marker='o',
label='Subway stations')
plt.title('Apartment listings in Toronto, May 13 2019')
plt.legend()
plt.savefig('Apartments.png')

# Plot apartments
plt.figure()
fig,ax = plt.subplots(figsize=(15, 15))
toronto_map.plot(ax=ax, alpha=0.3, color='grey')
geo_df[geo_df['property_type'] == 'Townhouse'].plot(
    ax=ax, markersize=20, alpha=0.5, color='blue', marker='o', label='Townhouse')
subway_df.plot(ax=ax, markersize=20, alpha=1, color='red', marker='o',
label='Subway stations')
plt.title('Townhouse listings in Toronto, May 13 2019')
plt.legend()
plt.savefig('Townhouses.png')

# Plot apartments
plt.figure()
fig,ax = plt.subplots(figsize=(15, 15))
toronto_map.plot(ax=ax, alpha=0.3, color='grey')
geo_df[geo_df['property_type'] == 'Loft'].plot(
    ax=ax, markersize=20, alpha=0.5, color='blue', marker='o', label='Loft')
subway_df.plot(ax=ax, markersize=20, alpha=1, color='red', marker='o',
label='Subway stations')
plt.title('Loft listings in Toronto, May 13 2019')
plt.legend()
plt.savefig('Lofts.png')
```