Statistical Inference W4702 Final Project

Christopher Rusnak, Alimu Mijiti, Christine Lee, Maura Fitzgerald

December 15, 2015

Contents

Introduction

Introduction to our project - why bother to look at airbnb data?

Motivation behind studying airbnb data In recent years, we come across discussions in the media and in our communities of how the use of various sharing economy platforms such as Uber and Airbnb have become more prevalent and might have a positive or possibly less positive impact on how we operate our lives.

We ourselves may be avid users of some of these platforms. For example, as a potential guest or owner of an Airbnb listing, one might be interested in determining whether listing prices are fair given the property, and in particular, what variables might be the most meaningful predictors of listing price. If you are an Airbnb host, you might wonder which variables can be the best predictors of whether you achieve the "superhost" status.

Data set requirements Furthermore, we had in mind some criteria for selecting an interesting set of data to explore. We feel Airbnb's listing data fits our search:

- It is a comprehensive data set with 94 original variables and over 30,000 observations for NYC area listings.
- It contains a rich set of variables, including a range of numerical and categorical variables.
- The data was recently compiled in September 2015 by the author of insideairbnb.com.
- The data contains interesting information on trends in properties and communities in the New York area.

Project questions we wish to explore with statistical modeling techniques

Some questions or insights we hope to address in this project include:

- Exploring the relationships between various predictors and designated Y variables price, or superhost status – how strong are these relationships and do they appear to be linear or non-linear?
- Building a classification model and identifying variables for predicting whether an Airbnb host is likely to be classified as a superhost.
- Building a GAM model and identifying the best predictors for predicting the Y variable as price.

Dataset

Data Source

We used publicly available Airbnb listings data from http://insideairbnb.com/. We limited the scope of our analysis to New York City. The original dataset contained 94 information fields for 30,480 listings in the NYC area.

Prep and Cleaning

Variables Included

From the original dataset we kept the following fields in something close to their original representation:

```
##  [1] "id"                           "listing_url"
##  [3] "scrape_id"                    "last_scraped"
##  [5] "name"                         "summary"
##  [7] "space"                        "description"
##  [9] "experiences_offered"          "neighborhood_overview"
## [11] "notes"                        "transit"
## [13] "thumbnail_url"                "medium_url"
## [15] "picture_url"                  "xl_picture_url"
## [17] "host_id"                      "host_url"
## [19] "host_name"                    "host_since"
## [21] "host_location"                "host_about"
## [23] "host_response_time"           "host_response_rate"
## [25] "host_acceptance_rate"         "host_is_superhost"
## [27] "host_thumbnail_url"           "host_picture_url"
## [29] "host_neighbourhood"           "host_listings_count"
## [31] "host_total_listings_count"    "host_verifications"
## [33] "host_has_profile_pic"         "host_identity_verified"
## [35] "street"                       "neighbourhood"
## [37] "neighbourhood_cleansed"       "neighbourhood_group_cleansed"
## [39] "city"                         "state"
## [41] "zipcode"                      "market"
## [43] "smart_location"               "country_code"
```

Id, host_id, latitude, longitude and is_location_exact were kept for plotting and reference purposes rather than analysis.

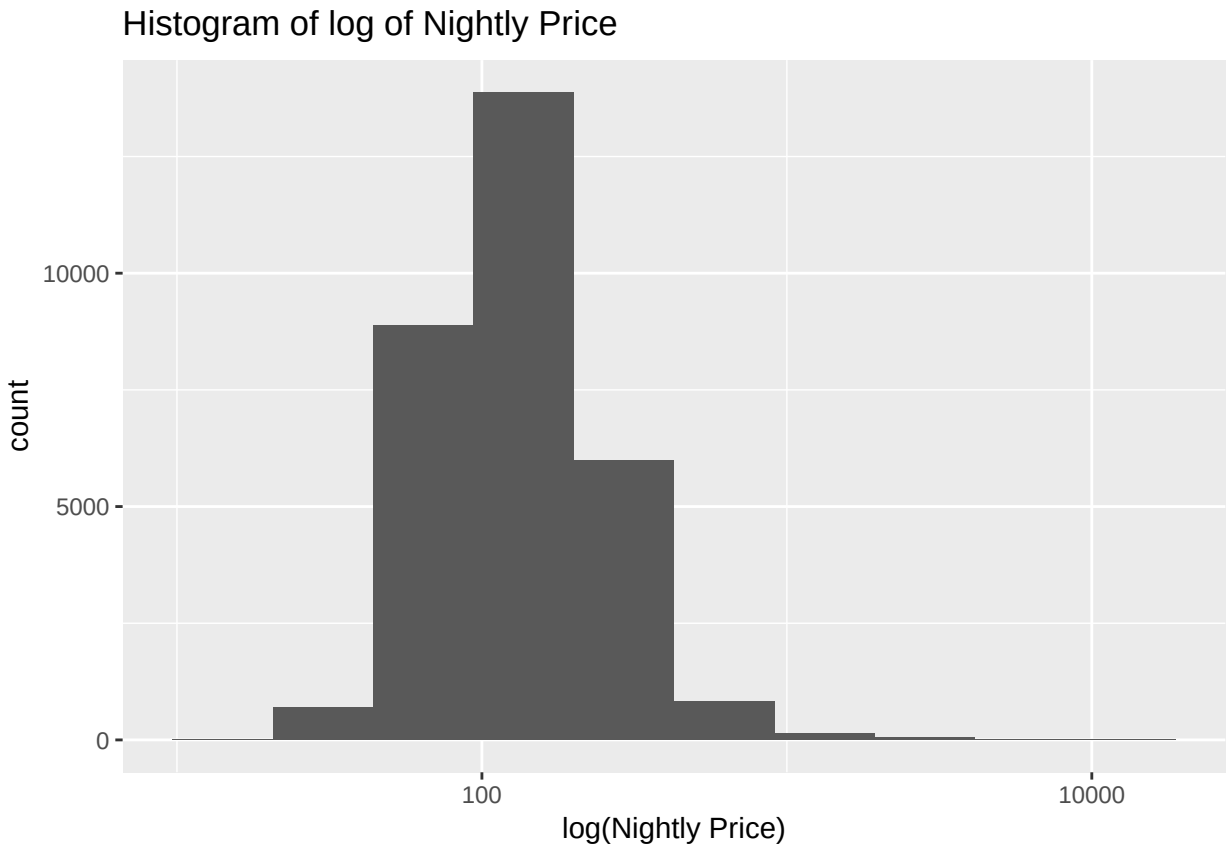
In addition to these fields, we created

- a count of the number of hosts at a location (based on names field)
- the distance (in meters) from a listing (based on lat/long) to the nearest subway station
- indicator variables for amenities in a listing
- indicator variables for host verifications

EDA

Price

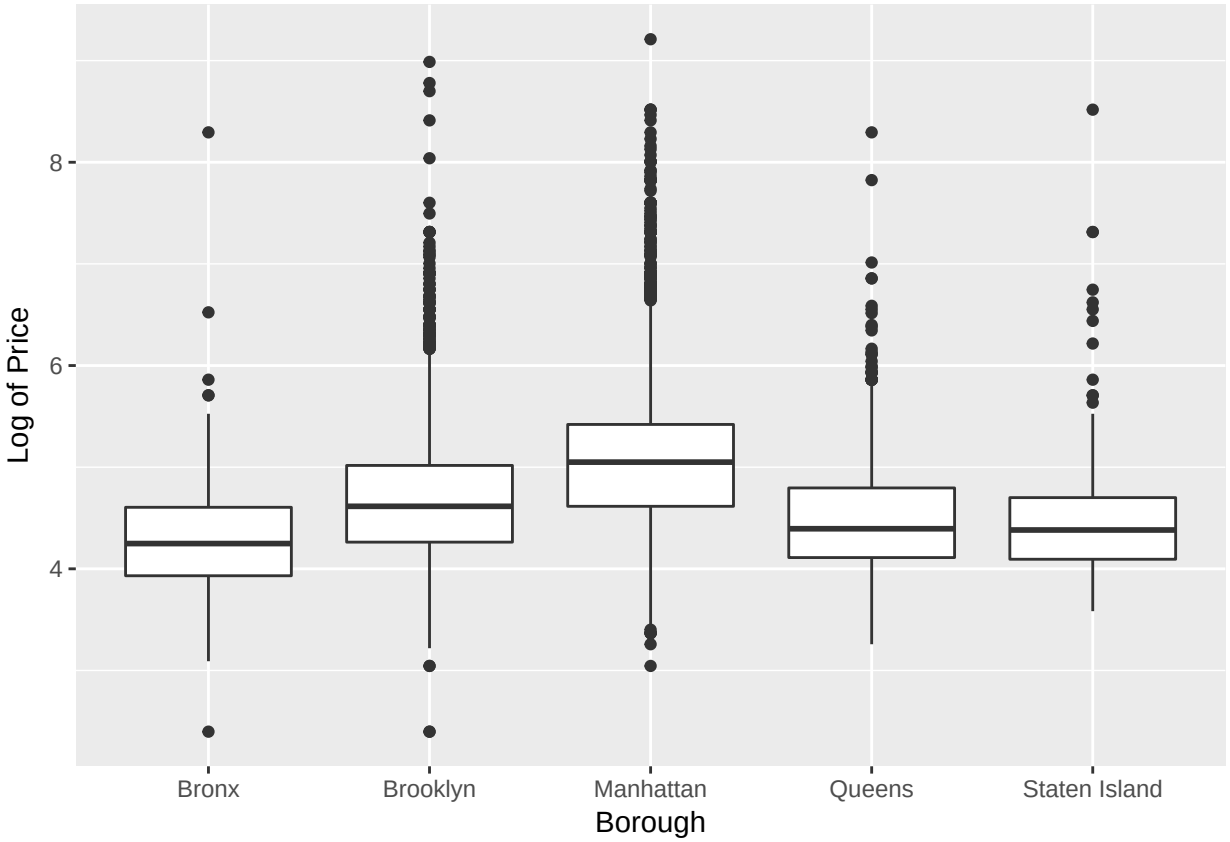
The mean and quartiles of the price are all less than $200, demonstrating that the majority of airbnb in New York City (at least 75%) have a non-expensive nightly price. The cheapest listing is $10 and the most expensive is $10,000. Because of the wide range and the skewed distribution of price values, we take the logarithm of the price to spread out the data.



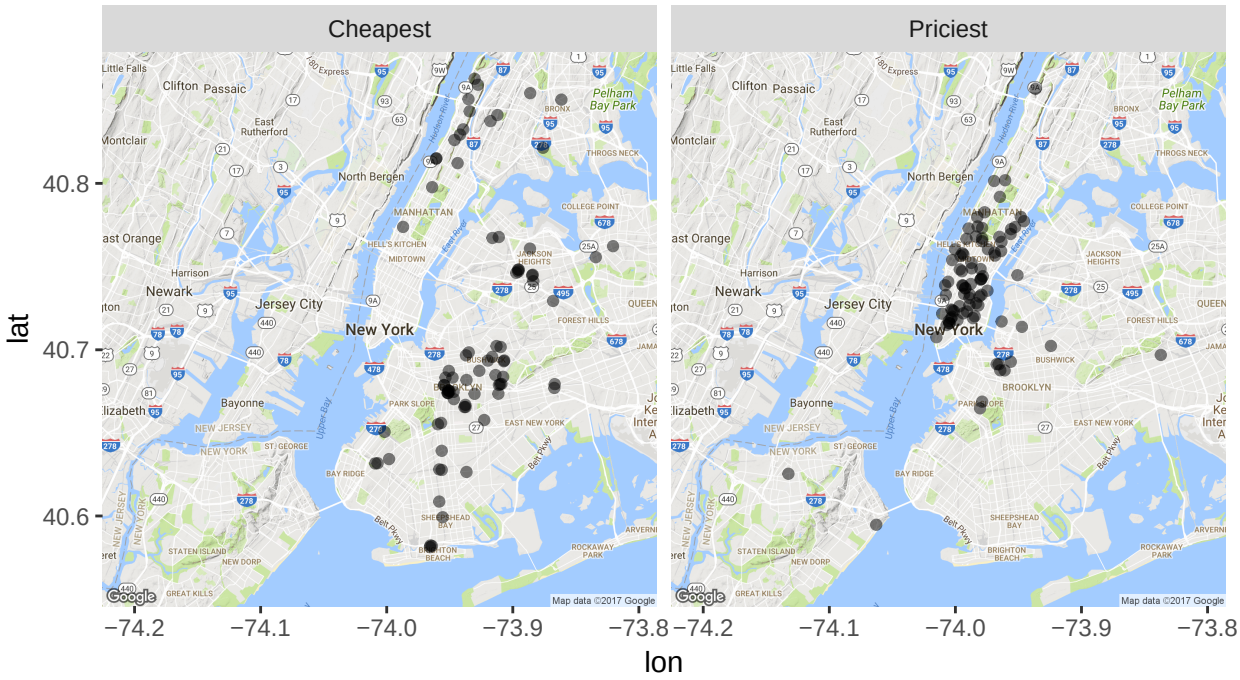
Price Summary Statistics

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    10.0    80.0   125.0   163.6   195.0 10000.0
```

Box plots of nightly price with respect to NYC borough demonstrate that the most expensive listings are in Manhattan, as one would expect.

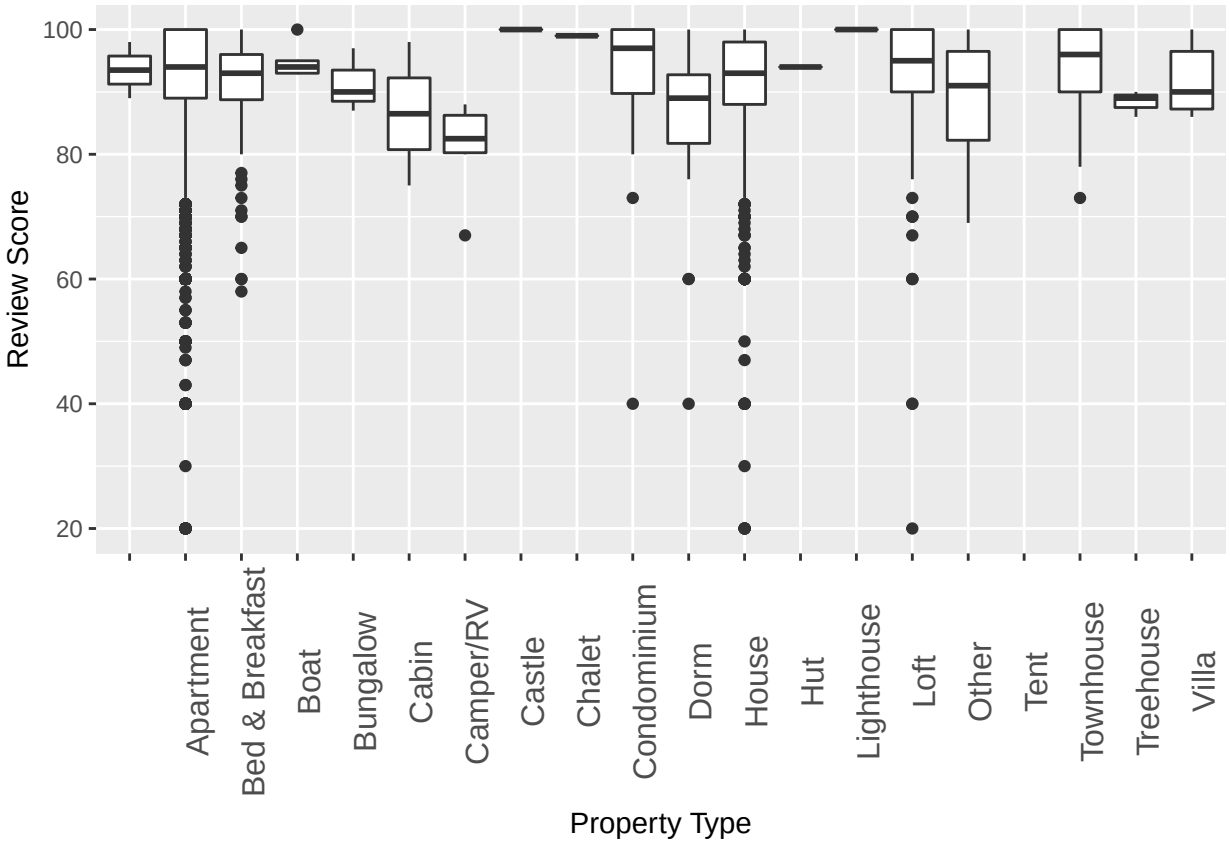


There are 83 listings whose nightly price is $30 or less. Likewise, there are 87 listings whose nightly price is at least $1,500. The least expensive listings are mostly located in Brooklyn neighborhoods such as Bushwick, Bedford-Stuyvesant, and Crown Heights. Over 80% of the most expensive listings are located in Manhattan.



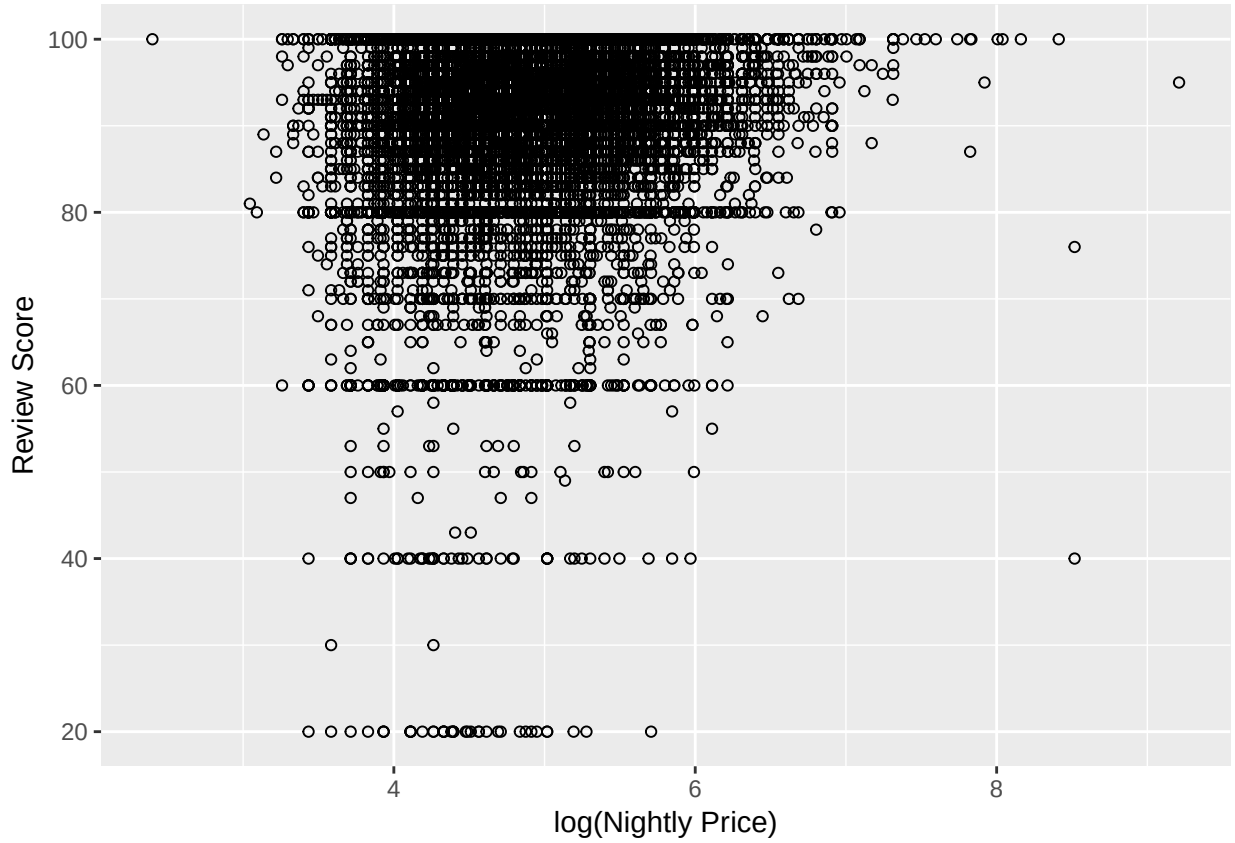
Ratings

Most listings, regardless of property type, tend to have very high reviews.



Ratings and Price

Price is not a strong indicator of the review rating, and the two variables appear to be independent.

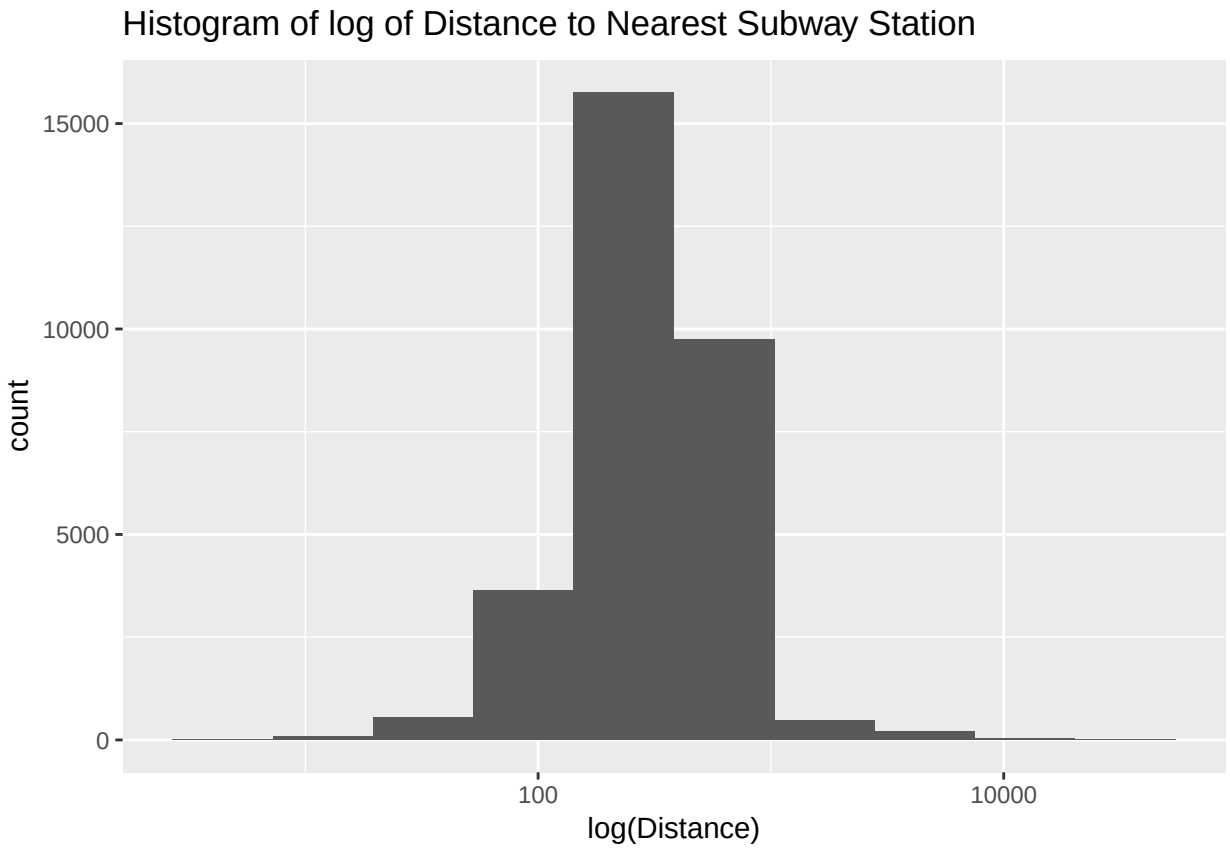


Distance to Nearest Subway

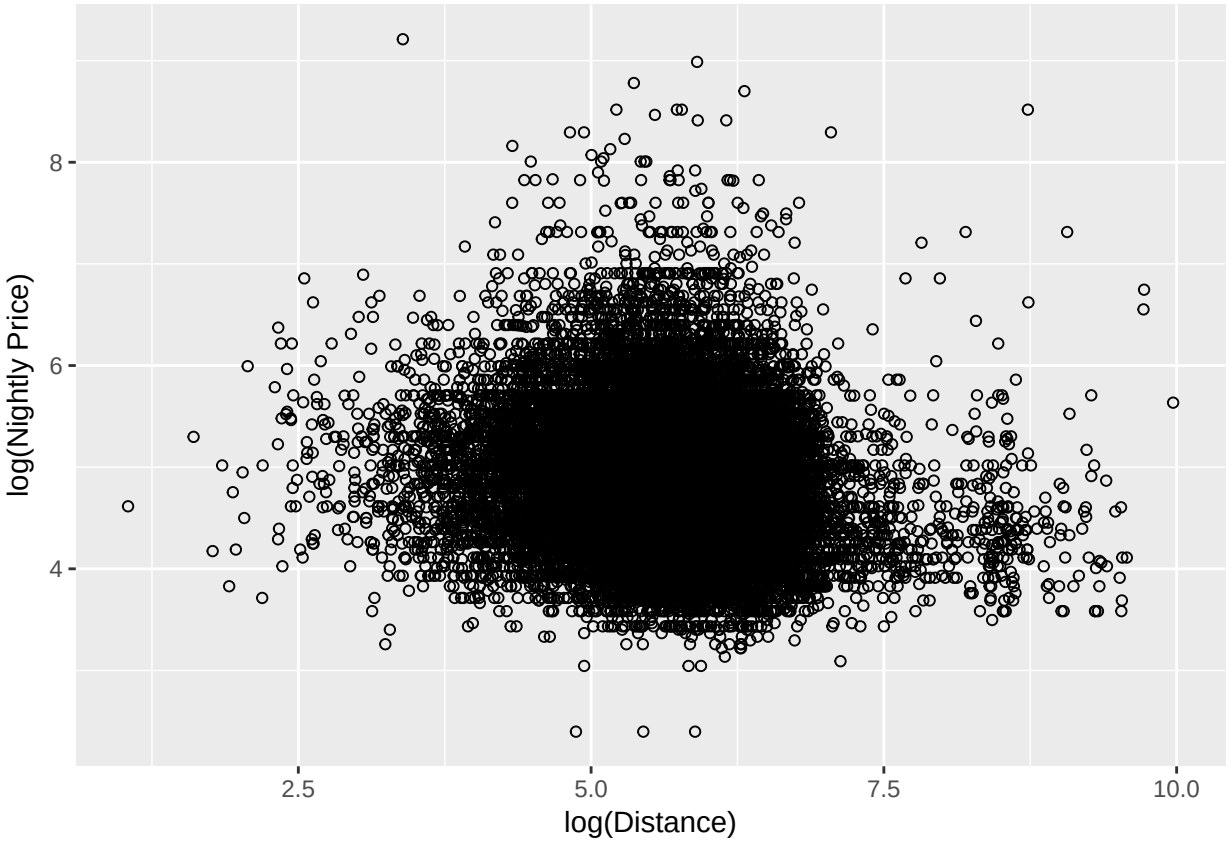
Most listings are within 445 meters from a subway station. The listing closest to a subway station is within one meter, likely in a building right above it. The listing farthest from a subway station is 21,380 meters (~13 miles) away.

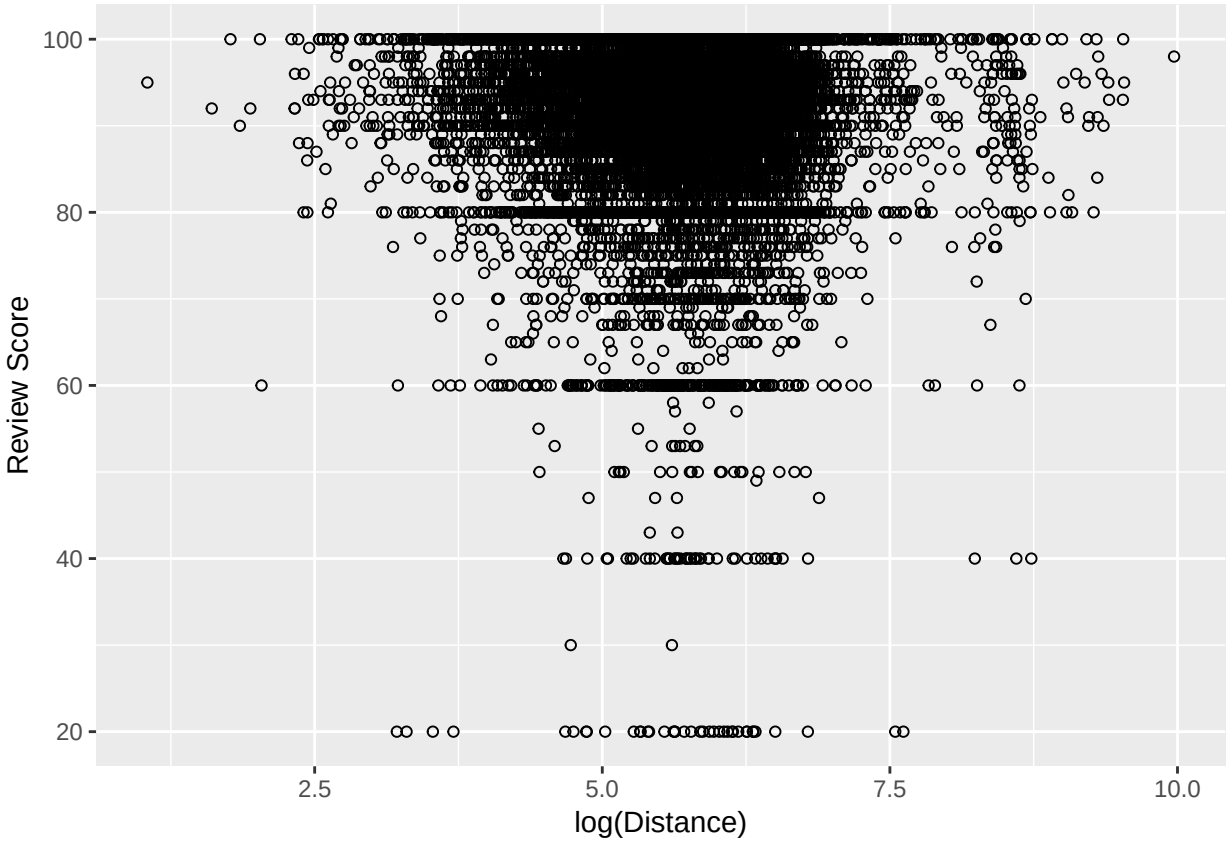
Distance to Subway Summary Statistics

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##     1.846   196.500   303.800   395.600   445.600 21380.000
```



There is no clear relationship between the distance of a listing to a subway station and that listing's price or review score.


log(Nightly Price)
8
6
4
2.5
5.0
7.5
10.0
log(Distance)



SuperHosts

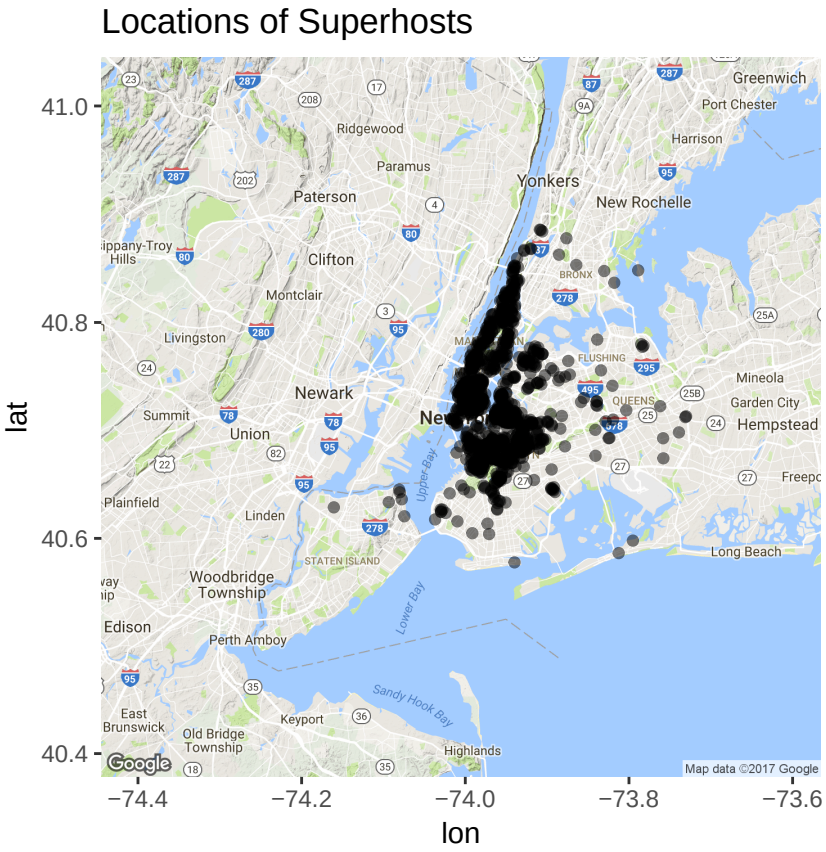
```
##           f     t
##     3 29228  1245
```

Only a small percentage of listings (~4%) are from "superhosts".

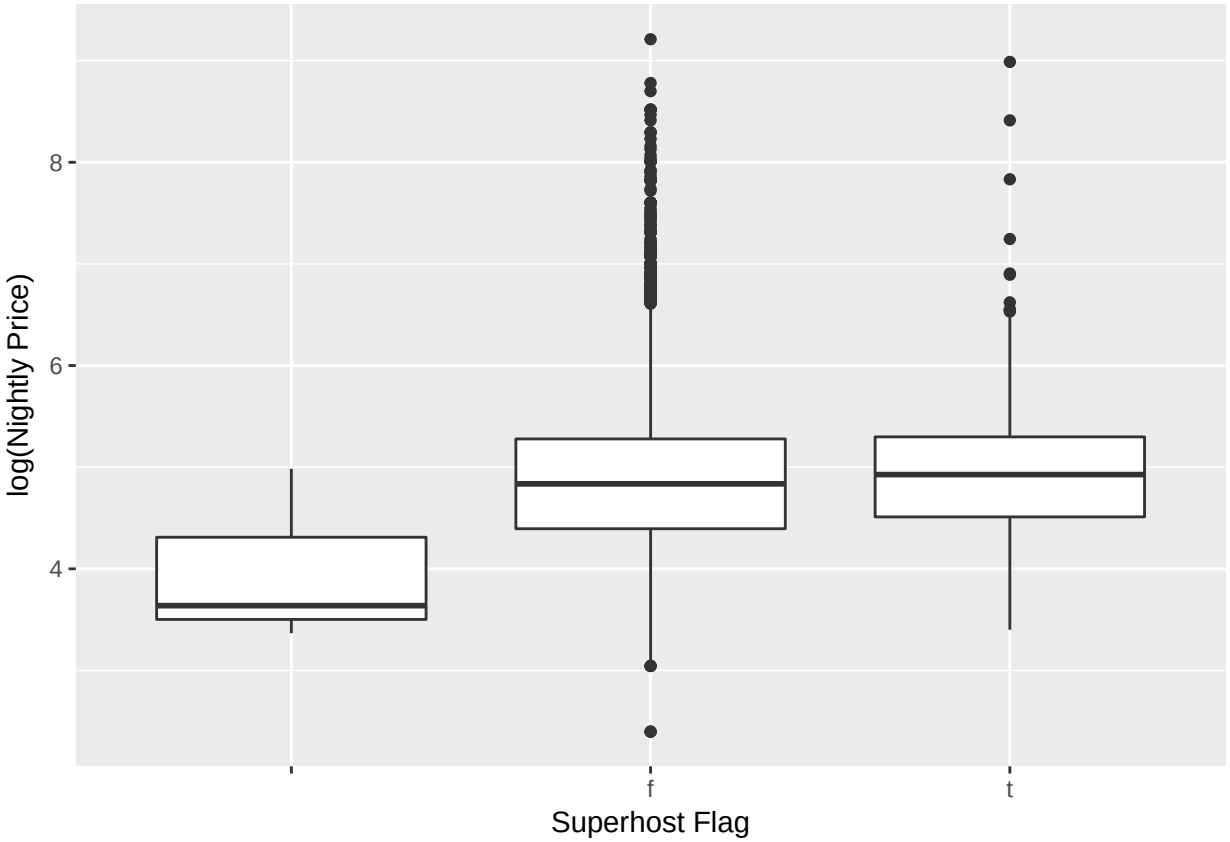
Brooklyn and Manhattan have the most superhosts, with both boroughs being home to nearly 90% of all superhosts in NYC.

Most superhosts in Brooklyn are in Williamsburg and Bedford-Stuyvesant. Within Manhattan, superhosts are more spread out, but tend to cluster in neighborhoods such as Harlem, Chelsea, Hell's Kitchen, and the East Village.

		f	t
Bronx	0	336	9
Brooklyn	2	11113	561
Manhattan	1	15478	552
Queens	0	2160	117
Staten Island	0	141	6

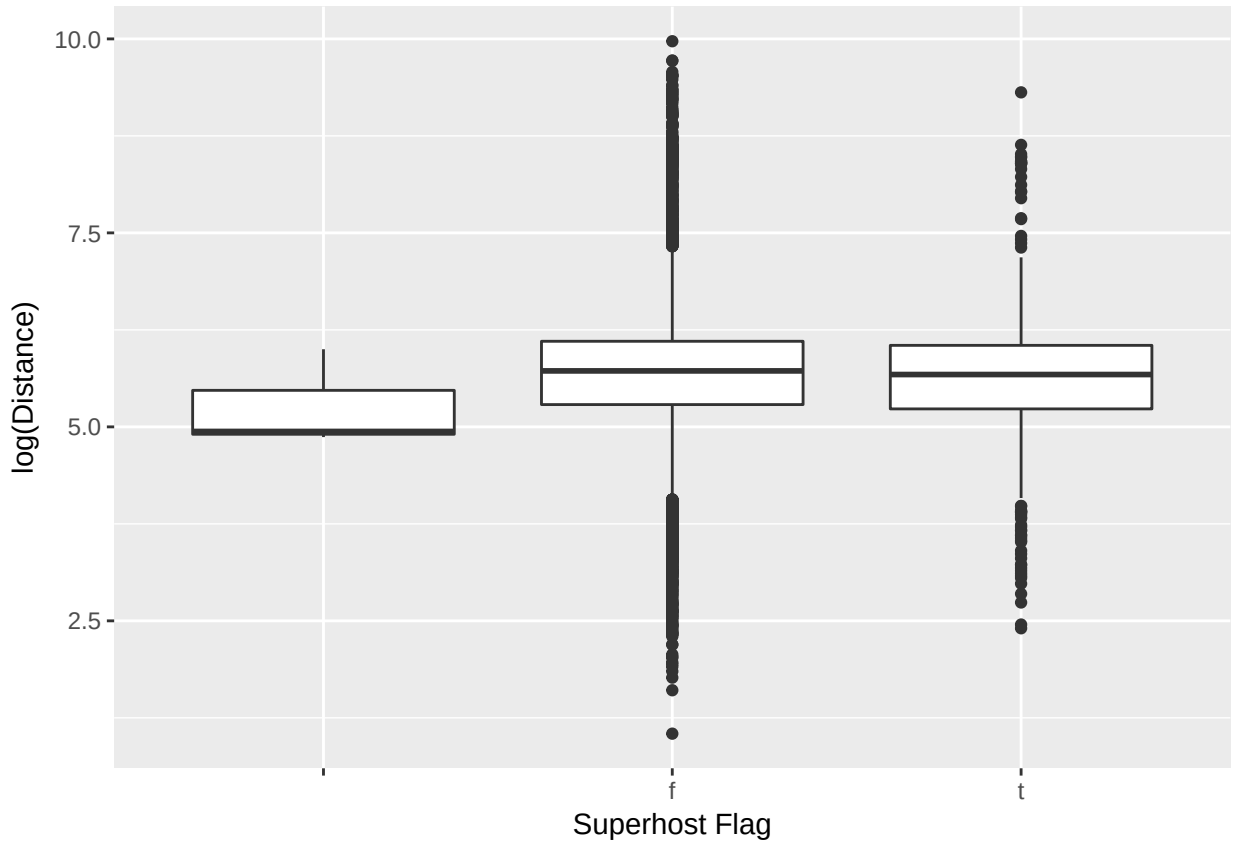


SuperHosts and Price



Box plots of the indicator variable of whether or not a host is a superhost against the log of the nightly price suggest that superhosts and non-superhosts charge similar prices for their listings. We can check this by running the Mann-Whitney Wilcoxon test.

SuperHosts and Subway Distance



Box plots of the indicator variable of whether or not a host is a superhost against the log of the distance to the nearest subway station suggest that superhosts and non-superhosts have similar distributions for the distance to a subway stop. We can check this by running the Mann-Whitney Wilcoxon test.

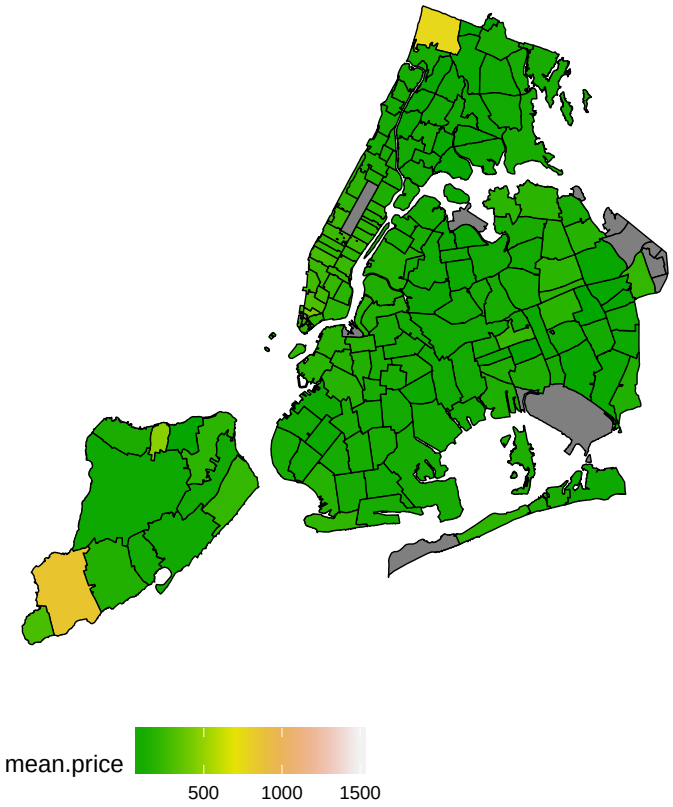
Correlations with Price

Numerical covariates tend to have a low correlation with the price. Please see Correlation results in the Appendix section. We also explore these relationships further in the regression section.

Average Price per Zip Code

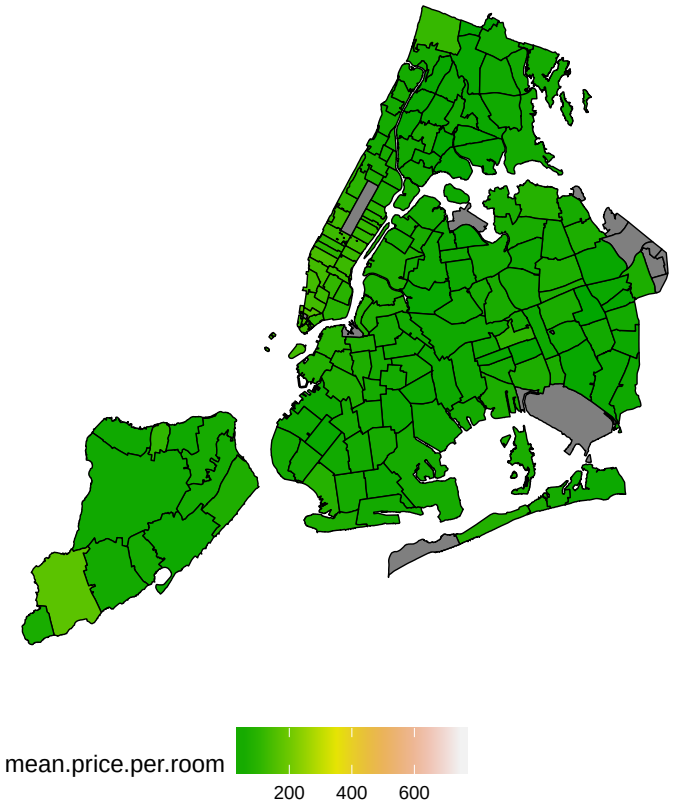
While exploring the data, we tried to find out the average price for different zipcodes in New York City.

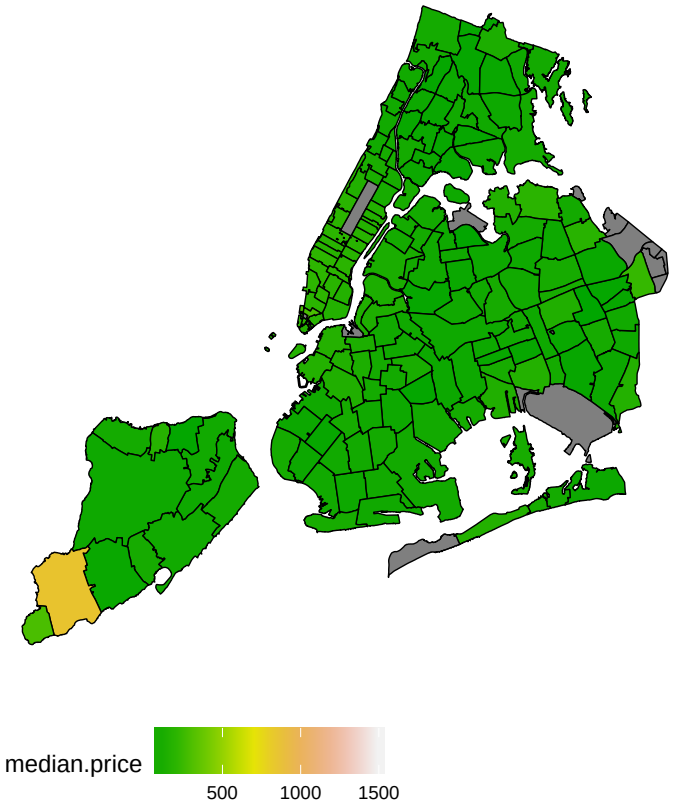
First, we averaged the price per listing, and mapped the average price on the zipcode map below. After mapping the price, we noted that the average listing price for Manhattan areas seem to be the lowest in all of the five boroughs (Manhattan, Brooklyn, Queens, Bronx and Staten Island) of New York.



After further analysis, we found out that some of the listings have more than one bedroom and some of the listings are actually listed as the entire house or apartment. We reasoned that the average price in Manhattan is the lowest of all the boroughs is because of the type of listing or the number of bedrooms in those listings.

As such, we tried to map the average price per room (for every listing, divide the price by the number of the bedrooms) on the zipcode map. While doing this, we encountered a problem: some of the listings have zero bedrooms, such that it is impossible to divide by the number of rooms for those listings. In this case, we used a method similar to the Laplace method from Bayes classification. Since we are considering the relative price of these regions, we added one to the number of bedrooms for every listing. Then we mapped the average price for bedroom on the zipcode map, and the result is largely the same. We also tried to map the median price for the every zipcode, giving similar results.


mean.price.per.room
200
400
600



The results seem counterintuitive. The renting price in Manhattan is high, similar to the price of hotels, so we expected the average price in Manhattan to be more expensive than other boroughs. However, after further analysis, we think the reason the average price in Manhattan is higher is that there are more hosts in Manhattan and there are more listings in Manhattan. Airbnb is a market and the competitive market in Manhattan might force the price down, which is why the average price in Manhattan is lower. Our reasoning is supported by the fact that there many more hosts in Manhattan than in any other borough. We infer that the listing market in Manhattan is much more competitive and there are more listings in Manhattan. Also, the hosts in other areas (especially the hosts in more distant parts of Brooklyn and the Bronx) might just list their home and put little effort in the listing market.

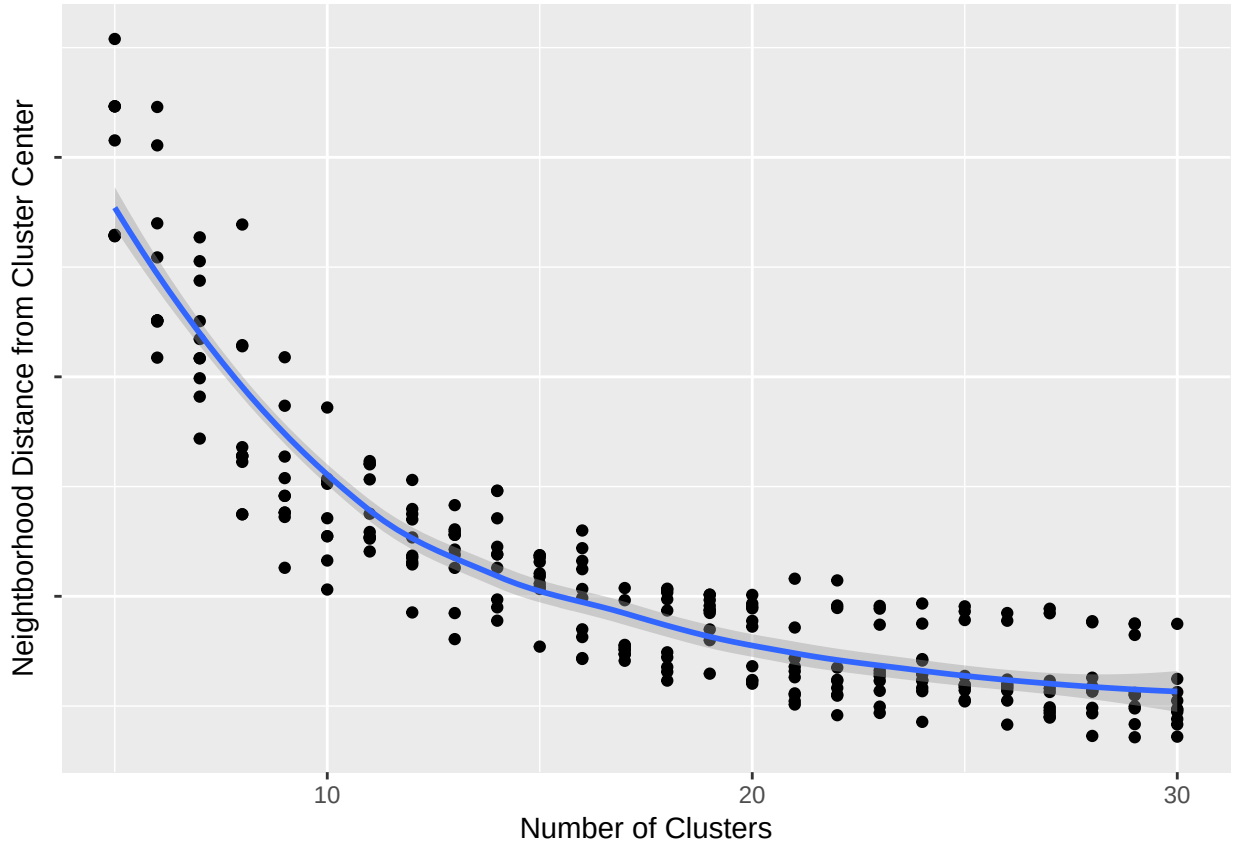
Treatment of Neighborhoods

The original dataset contains 205 distinct neighborhoods with anywhere from 1 to >2,500 listings. Despite the size of the data, 205 neighborhoods is too many for us to use in models. Inference on so many factors is outside the scope of the project and due to the unbalanced distribution of listings (many neighborhoods have fewer than 20), models built with the original neighborhoods are likely to overfit the training set and hold up poorly to cross-validation.

Even though we aren't using the original neighborhoods, we would still like to use some information from this field in our analysis. One possible solution to this dilemma is clustering neighborhoods based on shared characteristics.

We clustered the 205 neighborhoods based on location (latitude, longitude and borough), mix of property types, and average distance to a subway station. In a more comprehensive analysis we would test clustering based on external metrics such as population density, household income, and zoning information (% residential) or using these in the models directly.

K-means was chosen as our clustering algorithm with $k = 10$. 10 was chosen based on the reduction in distances from points to their cluster centers as well as cluster sizes (At least 25 listings/cluster). After choosing K, we ran the algorithm a few times and chose the output with at least 25 listings per cluster (K-means will often pick a local minimum so running the algorithm a few times can give different solutions).



Neighborhood to cluster mapping and number of listings in appendix

Model Results and Diagnostics

1) Mann-Whitney Wilcoxon test

Mann-Whitney Wilcoxon for Superhost & price

We use the Mann-Whitney Wilcoxon test to test at a rejection level of 0.05 the null hypothesis that superhost and non-superhosts have prices from the same distribution against the alternative hypothesis that they are derived from different distributions. This test results in a p-value of 1.217×10^{-5}. Therefore, we can reject the null hypothesis, as it is likely that superhosts and non-superhosts charge prices from different distributions.

Mann-Whitney Wilcoxon for Superhost & distance to subway

We use the Mann-Whitney Wilcoxon test to test at a rejection level of 0.05 the null hypothesis that superhost and non-superhosts have distances to the nearest subway from the same distribution against the alternative hypothesis that they are derived from different distributions. This test results in a p-value of 0.004824. Therefore, we can reject the null hypothesis, as it is likely that superhosts and non-superhosts have nearest subway stop distances from different distributions.

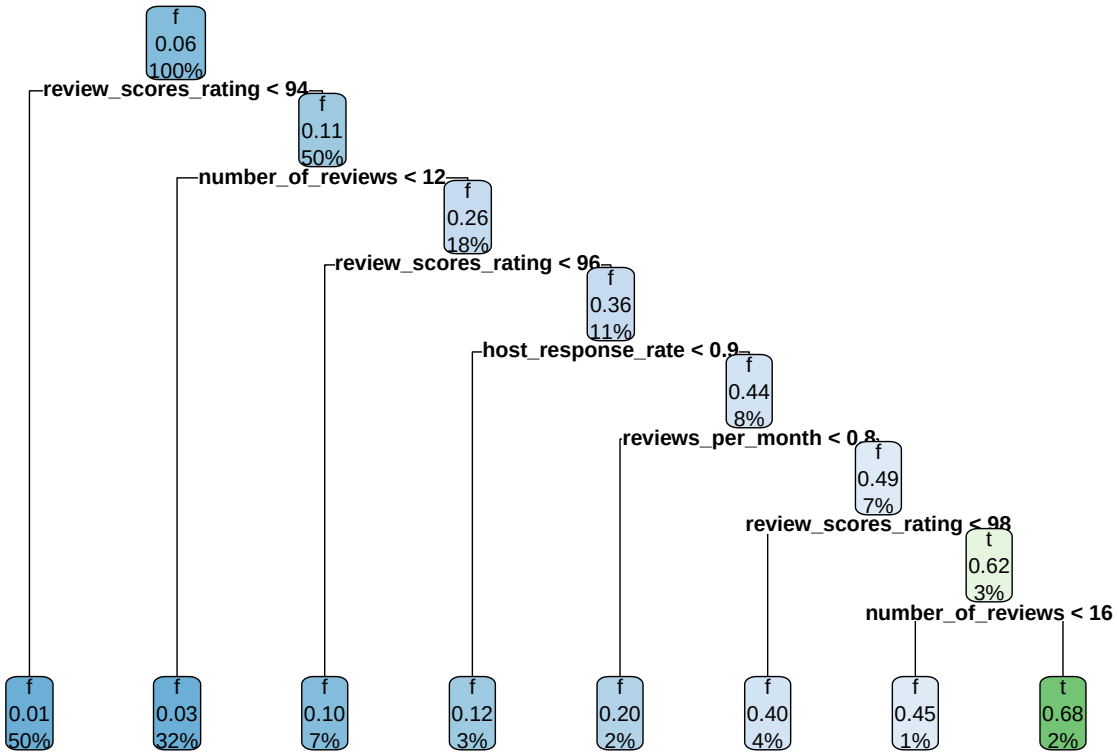
2) Classification for predicting y:superhost and neighborhood

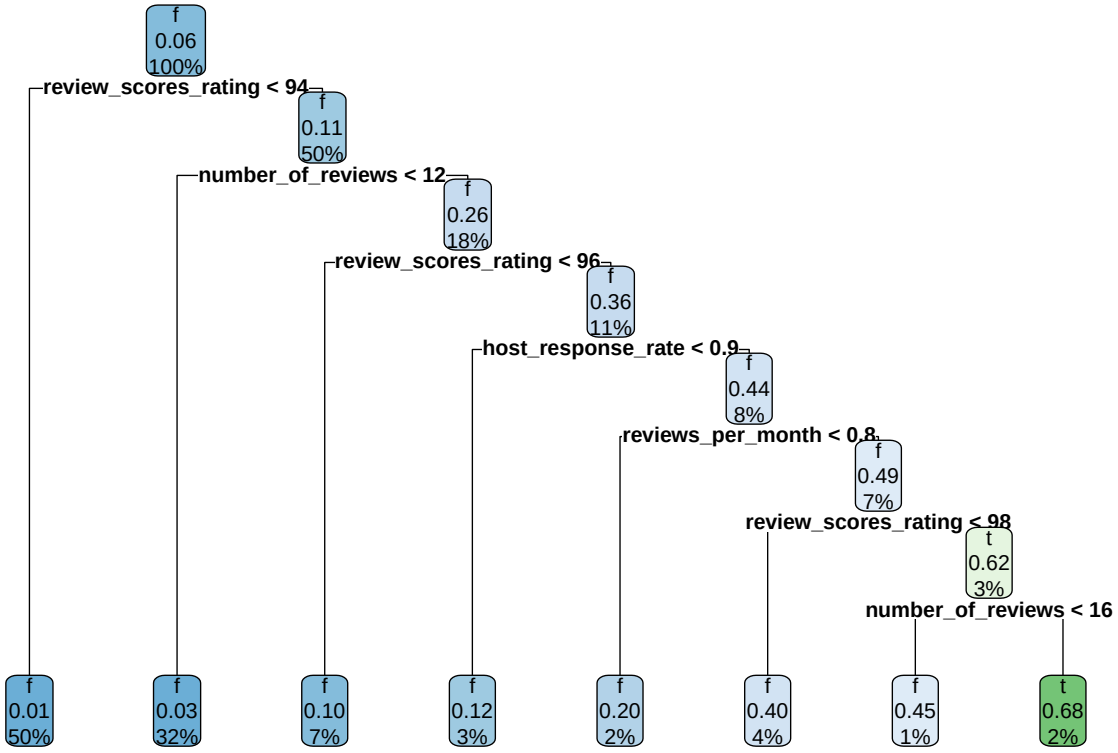
As a classification method, we attempted Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA) on the dataset. As with the tree classification, we predicted whether or not a host was a "superhost". However, since QDA and LDA use Bayes' theorem, and because most of the variables in our dataset are numeric and have wide ranges, we have deduced that QDA and LDA are not appropriate classification methods for our data. Comparing QDA, LDA and the decision tree, the tree classification is the most efficient one. As such, we recommend using the tree classification method for predicting whether the host is a superhost.

Using a decision tree, we classified each listing for whether or not the host is a "superhost". After the initial EDA, we chose 16 variables for training the tree model. Since we have a sufficiently large number of observations in our dataset, we randomly divided the dataset into two partitions: training set and test set. About $\frac{5}{6}$ of the data is used for training and the remaining $\frac{1}{6}$ of the data is used for testing.

When training the model, we used Gini index and Information Gain as entropy measures for developing the tree.

After training the model, we tested it on the testing dataset, resulting in an overall accuracy rate of 96.4% for both the Gini and Information Gain splitting indices.

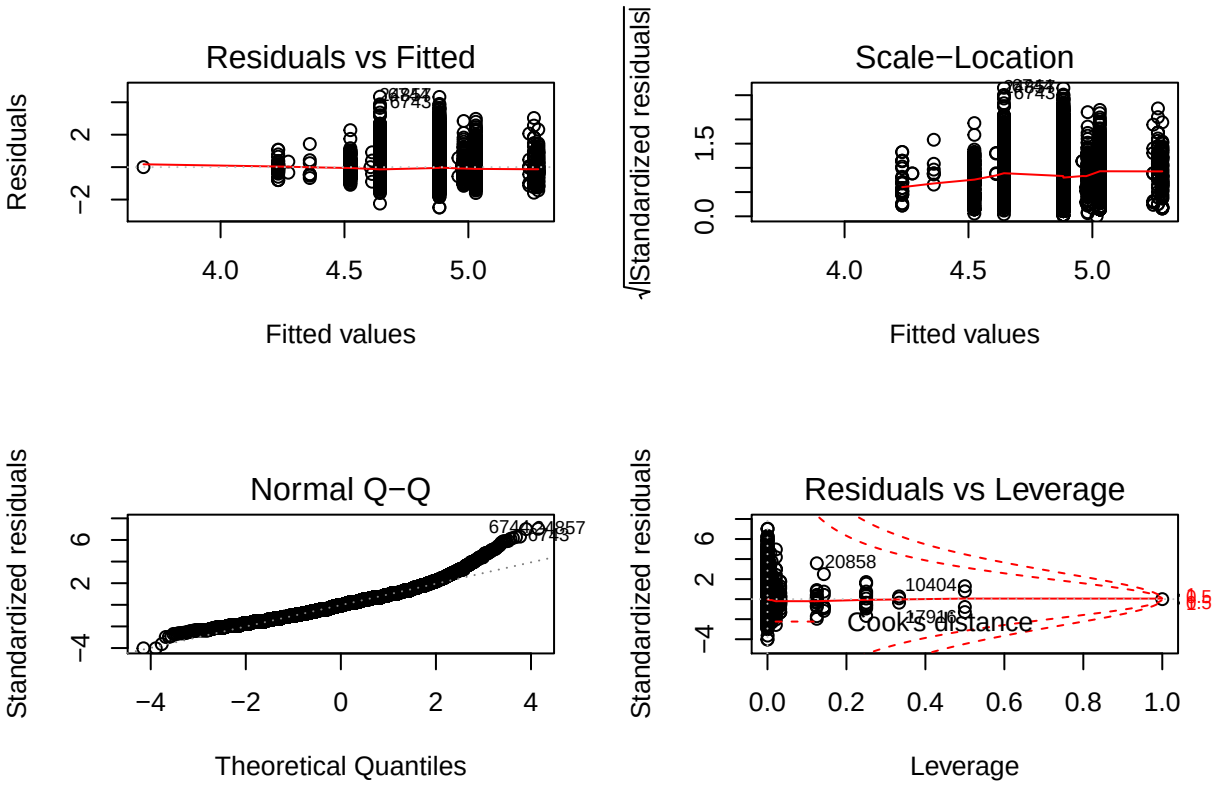




3) ANOVA analysis

One Way Anova for price by property type

We apply one-way Analysis of Variance (ANOVA) to test at a rejection level of 0.05 the null hypothesis that different property types have the same mean price against the alternative hypothesis that at least one pair of property types has different mean prices. This test is done on the assumption that the prices are derived from a Gaussian distribution with an unknown but fixed variance. This test results in a p-value of less than 2×10^{-16}. Therefore, we can reject the null hypothesis, as the different property types likely have different mean prices.



The visualization of the residuals vs. fitted model results shows that the residual error increases as the fitted value increases. These results suggest that there is a non-constant variance of the error terms, such that the assumption of homoscedasticity is violated.

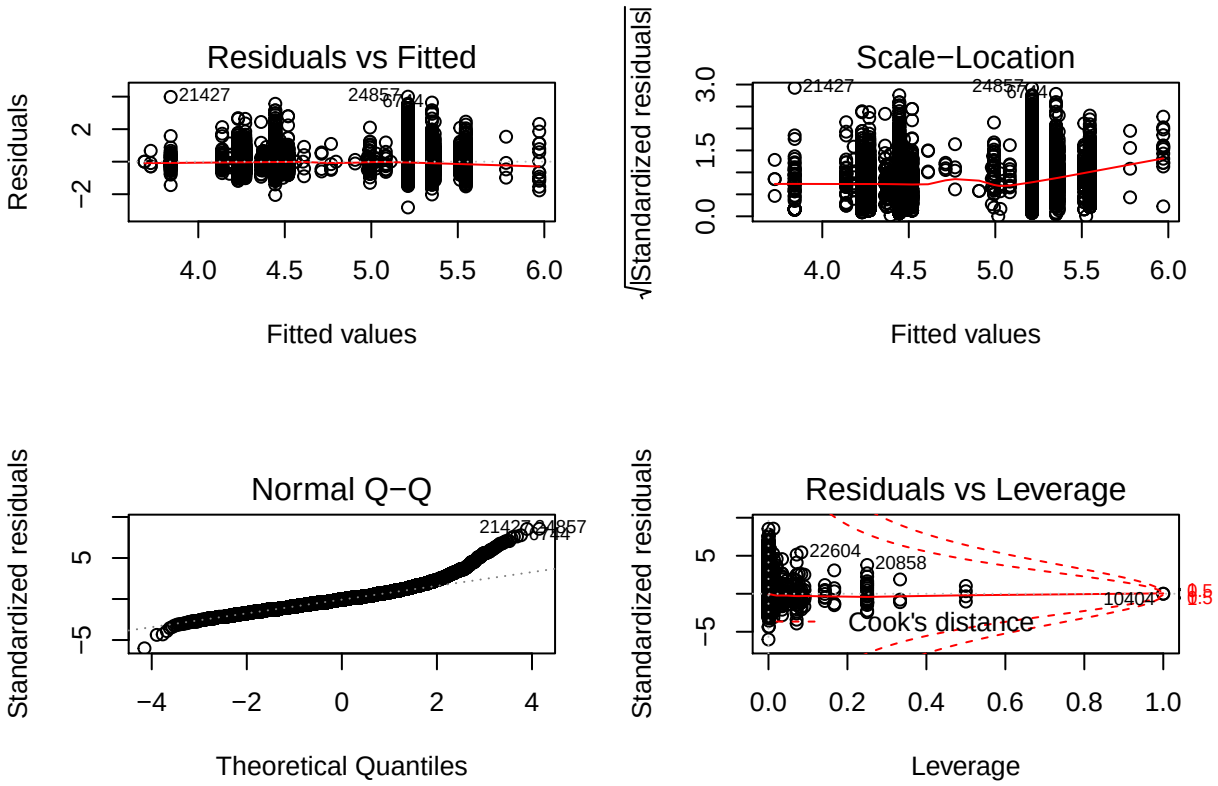
The normal Q-Q plot shows a curved fit that is close to the diagonal line, suggesting that the residual data is close to a Normal distribution.

The residuals vs. leverage plot shows that the data points with the least leverage have the most variance in their residual error. This finding suggests that the ANOVA model does not fit the price for more common residence types as well as less common residence types.

The AIC value of this model is 57142.26, and the BIC value of this model is 57317.08.

Two Way Anova for Price vs. Property Type & Room Type

We apply two-way Analysis of Variance (ANOVA) to test at a rejection level of 0.05 the null hypothesis that different property types and room types have the same mean price against the alternative hypothesis that at least one pair of property types & room types has different mean prices. This test is done on the assumption that the prices are derived from a Gaussian distribution with an unknown but fixed variance. This test results in a p-value of less than 2×10^{-16}. Therefore, we can reject the null hypothesis, as the different property types and room types likely have different mean prices.



The visualization of the residuals vs. fitted model results shows that the residual error has roughly the same range as the fitted value increases.These results suggest that there is a constant variance of the error terms, such that the assumption of homoscedasticity is not violated.

The normal Q-Q plot shows a curved fit that is close to the diagonal line for most residual points, suggesting that the residual data is close to a Normal distribution.

The residuals vs. leverage plot shows that the data points with the least leverage have the most variance in their residual error. This finding suggests that the ANOVA model does not fit the price for more common property type & room type combinations as well as less common property type & room type combinations.

The AIC value of this model is 40287.79, and the BIC value of this model is 40637.43.

4) Lasso regression for predicting y:price

Applying the Lasso regression for informing subset selection ideas for predictors variables

To aid our subset selection process for regressions on predicting one of our y variables - *price* or log(*price*), instead of studying the relationships of several hundred individual predictor variables (after dummifying all factors) with the dependent varaible via scatterplots or correlation matrices, we also wish to rely on the Lasso regression with regularization, a technique involving all p predictors where some coefficients without much explanatory power would be effectively constrained to 0. We outline the implementation process and outcome below.

Challenges we overcame in implementing the Lasso with glmnet in R

We overcame a few challenges before being able to implement the Lasso regression with the `glmnet()` function.

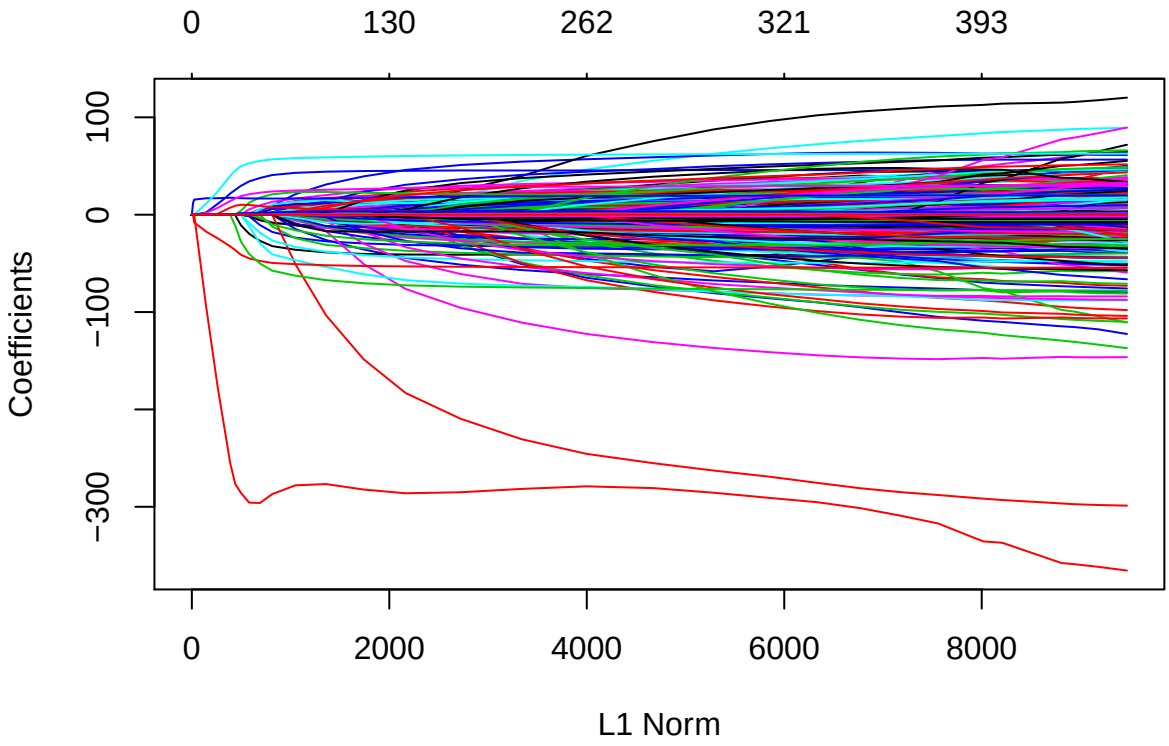
1.glmnet() function's inability to handle NA values

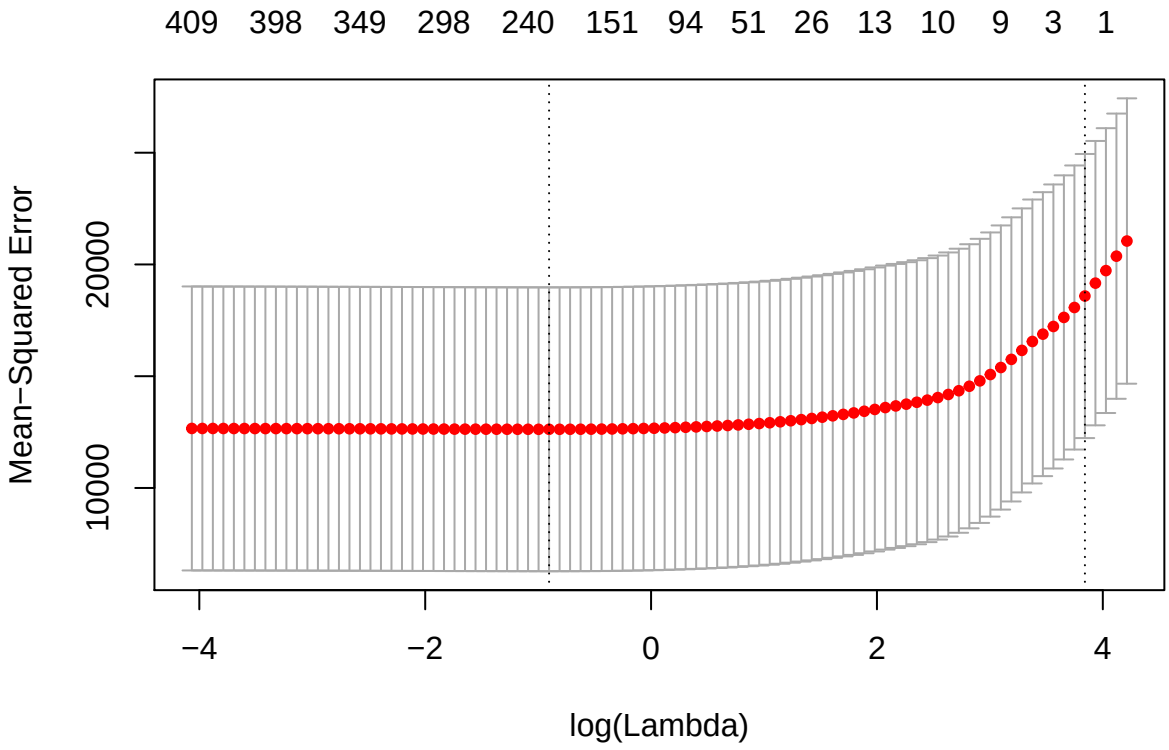
In the data cleaning stage, for some numerical variables where it made sense to insert \$0 values for *N/A* or missing values we did so, such as for security fee and cleaning fee. For other variables such as different ratinges, we left the *N/A* values for missing inputs. However,feeding our matrix of predictor variables into `glmnet()` with *N/A* values appear to lead to the following error:`"NA/NaN/Inf in foreign function call"` even after trying to pass the option of `"na.action=na.pass"` into the function.

We proceeded to exclude row items with *N/A*s or missing values before feeding the data into our predictor variable matrix with the `model.matrix()` function. Fortunately, excluding observations with *N/A*s from our original data set still leaves us with more than 19,000 observations for our test and training sets.

2.Lasso regression appear to be thrown off by select unnecessary categorical variables

Our first attempt at applying the Lasso regression on the entire set of p original predictor variables (92 total) appeared to be unsuccessful given the very limited number of predictor variables that came out with a positive coefficient. We suspected including a number of not predictively meaningful x variables such as IDs and a problematic date variable, all of which have a high number of levels, may have led to problems for Lasso regression's implementation. We therefore decided to exclude such variables from our predictor matrix, afterwhich the Lasso regression results turn out to be more sensible.





Lasso subset selection suggests hosts' info and neighborhood varaibles are associated with price

Highlights of predictor variables with explanatory power

We examined the list of predictor variables with non-zero coefficients after running the Lasso with an optimal Lambda selected by cross-validation at 0.4. The first list of coefficients shown below involves those variables whose coefficients have been shrunken to 0 by the regularization technique. The second list shows non-zero coefficients from our Lasso regression.

This list of non-zero coefficients, suggesting the associated predictors have some degree of association with our dependent variable y of price, provide some interesting insights about which variables explain the movement of price.

The first set of variables involve information about the host. In particular, the categorical variable *host_is_superhost* (whether a listing's host is a "superhost") is suggested to have a meaningful positive relationship with the price variable. Another is *host_has_profile_pict*. The second set of variables with non-zero coefficients come from neighborhood factor variables, especially for *neighbourhood_cleansedBull's Head*, *neighbourhood_cleansedBelle Harbor*, and *neighbourhood_cleansedBronxdale*, which have sizeable positive or negative cofficients. The coefficients suggest listings being situated in these select neighborhoods have a meaningful impact on the price level for these listings.

```
##                                   (Intercept)
##                                 -1.954594e+04
##             host_response_timea few days or more
##                                 -3.232595e-01
##                               host_response_timeN/A
```

```
##                                     0.000000e+00
##                   host_response_timewithin a day
##                                     0.000000e+00
##             host_response_timewithin a few hours
##                                     0.000000e+00
##                 host_response_timewithin an hour
##                                     3.620674e+00
##                               host_response_rate
##                                     5.322428e+00
##                            host_acceptance_rate
##                                     3.977790e+00
##                              host_is_superhostf
##                                    -1.770251e+01
##                              host_is_superhostt
##                                     0.000000e+00
##                       host_total_listings_count
##                                     0.000000e+00
##                           host_has_profile_picf
##                                    -3.569533e+00
##                           host_has_profile_pict
##                                     2.826174e-10
##                         host_identity_verifiedf
##                                    -2.277885e+00
##                         host_identity_verifiedt
##                                     0.000000e+00
##              neighbourhood_cleansedArden Heights
##                                     0.000000e+00
##                   neighbourhood_cleansedArrochar
##                                     0.000000e+00
##                    neighbourhood_cleansedArverne
##                                     0.000000e+00
##                    neighbourhood_cleansedAstoria
##                                     0.000000e+00
##                 neighbourhood_cleansedBath Beach
##                                     0.000000e+00
##          neighbourhood_cleansedBattery Park City
##                                     0.000000e+00
##                  neighbourhood_cleansedBay Ridge
##                                    -1.622415e+01
##                neighbourhood_cleansedBay Terrace
##                                    -2.572030e+01
## neighbourhood_cleansedBay Terrace, Staten Island
##                                     0.000000e+00
##                     neighbourhood_cleansedBayside
##                                     0.000000e+00
##                   neighbourhood_cleansedBayswater
##                                     0.000000e+00
##         neighbourhood_cleansedBedford-Stuyvesant
##                                    -1.335128e+01
##                neighbourhood_cleansedBelle Harbor
##                                     0.000000e+00
##                   neighbourhood_cleansedBellerose
##                                     0.000000e+00
##                     neighbourhood_cleansedBelmont
```

```
##                                   1.067359e+00
##               neighbourhood_cleansedBensonhurst
##                                  -1.122646e+01
##              neighbourhood_cleansedBergen Beach
##                                   0.000000e+00
##               neighbourhood_cleansedBoerum Hill
##                                  -2.235038e+00
##              neighbourhood_cleansedBorough Park
##                                   0.000000e+00
##                 neighbourhood_cleansedBriarwood
##                                   0.000000e+00
##            neighbourhood_cleansedBrighton Beach
##                                   0.000000e+00
##                 neighbourhood_cleansedBronxdale
##                                  -4.066371e+01
##          neighbourhood_cleansedBrooklyn Heights
##                                   0.000000e+00
##              neighbourhood_cleansedBrownsville
##                                  -2.598075e+00
##              neighbourhood_cleansedBull's Head
##                                   0.000000e+00
##                neighbourhood_cleansedBushwick
##                                   0.000000e+00
##           neighbourhood_cleansedCambria Heights
##                                   0.000000e+00
##                neighbourhood_cleansedCanarsie
##                                  -5.807999e+01
##           neighbourhood_cleansedCarroll Gardens
##                                   1.875496e+00
##              neighbourhood_cleansedCastle Hill
##                                   0.000000e+00
##         neighbourhood_cleansedCastleton Corners
##                                   0.000000e+00
##                  neighbourhood_cleansedChelsea
##                                   2.539421e+01
##                neighbourhood_cleansedChinatown
##                                  -1.084797e+01
##              neighbourhood_cleansedCity Island
##                                   0.000000e+00
##              neighbourhood_cleansedCivic Center
##                                  -8.186506e+00
```

```
##                             (Intercept)
##                           -1.954594e+04
##    host_response_timea few days or more
##                           -3.232595e-01
##        host_response_timewithin an hour
##                            3.620674e+00
##                      host_response_rate
##                            5.322428e+00
##                     host_acceptance_rate
##                            3.977790e+00
##                       host_is_superhostf
##                           -1.770251e+01
```

```
##                     host_has_profile_picf
##                              -3.569533e+00
##                     host_has_profile_pict
##                               2.826174e-10
##                   host_identity_verifiedf
##                              -2.277885e+00
##          neighbourhood_cleansedBay Ridge
##                              -1.622415e+01
##        neighbourhood_cleansedBay Terrace
##                              -2.572030e+01
## neighbourhood_cleansedBedford-Stuyvesant
##                              -1.335128e+01
##            neighbourhood_cleansedBelmont
##                               1.067359e+00
##        neighbourhood_cleansedBensonhurst
##                              -1.122646e+01
##        neighbourhood_cleansedBoerum Hill
##                              -2.235038e+00
##          neighbourhood_cleansedBronxdale
##                              -4.066371e+01
##        neighbourhood_cleansedBrownsville
##                              -2.598075e+00
##           neighbourhood_cleansedCanarsie
##                              -5.807999e+01
##    neighbourhood_cleansedCarroll Gardens
##                               1.875496e+00
##            neighbourhood_cleansedChelsea
##                               2.539421e+01
##          neighbourhood_cleansedChinatown
##                              -1.084797e+01
##       neighbourhood_cleansedCivic Center
##                              -8.186506e+00
```

Comparing our Lasso vs. Ridge regression performance

```
## [1] "MSE for Lasso Regression model:"
```

```
## [1] 8409.719
```

```
## [1] "MSE for Ridge Regression model:"
```

```
## [1] 8397.412
```

Ridge regression has slightly more predictive power than Lasso

Finally, we wished to run the ridge regression also to compare whether one model may have better explanatory ability than the other. Our results on mean squared error for the two models shown above suggests our ridge regression model has a lower MSE and hence better explanatory ability. We believe this is because the lasso regularization technique's subset selection ability forces unnecessary variables' coefficients to zero and as a result returns a more predictively meaningful subset of predictors for the regression and thus enhances the model performance.

5) Non-linear models: splines + GAM

Exploring GAM regression models with linear & non-linear variables on Y var as price

Given our vast list of predictor variables, we wished to construct a model incorporating non-linear approaches on some of our predictors. We therefore turned to the Generalized Additive Model (GAM) approach. We first set out to construct an appropriate GAM model with the dependent variable as price. In the next section, we explored GAM models with the dependent variable as $\log(y)$.

Initial iterations: Testing whether some variables have a non-linear relationship with Y

We experimented with our first 2 models *gam.2* and *gam.3* to explore whether several predictor variables - *review_scores_rating*, *num_amenities*, *security_deposit*, *distance_in_meters* - have a non-linear relationship with our y variable. Within the GAM model, we apply a natural spline technique to these variables in question.

With the below ANOVA test on *gam.2* and *gam.3* models, where in *gam.2* we did not apply the natural spline to the variables in question, we find that the very small p-value observed for *gam.3* model provides compelling evidence that a GAM with non-linear modeling of these variables is a better model than a GAM with only linear modeling of these variables.

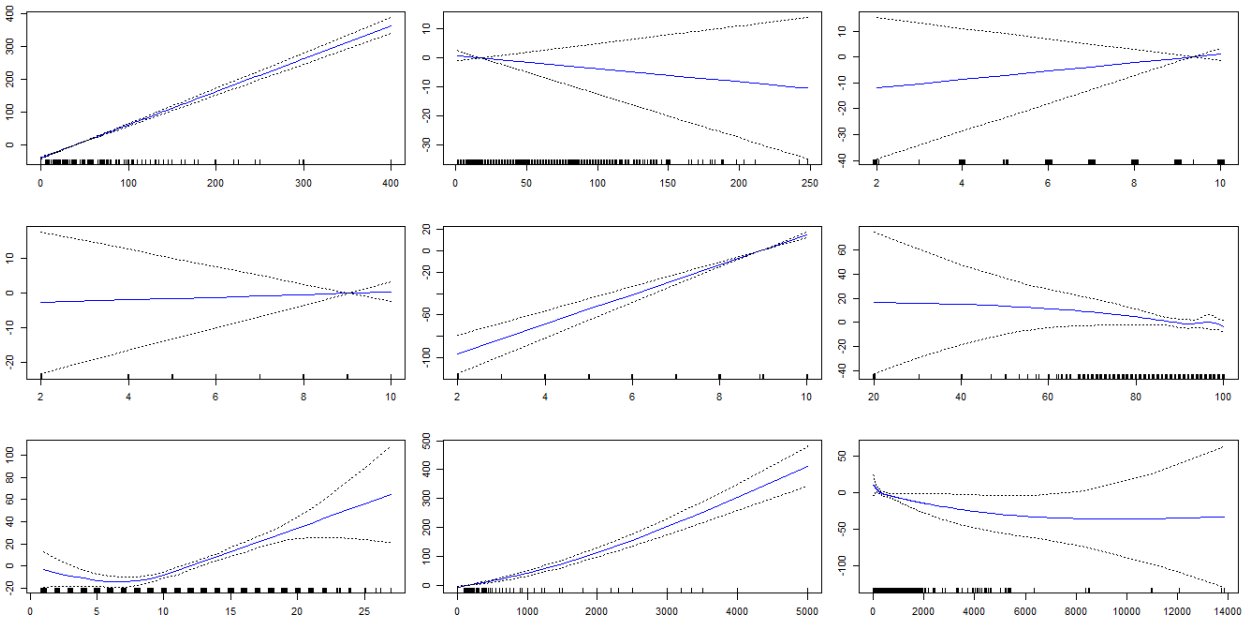
With this in mind, we move ahead to build on *gam.3* by adding more categorical variables to our *gam.4* model, whose categorical variables are shown in the *gam.4* plot below.

ANOVA on gam.2 and gam.3 models to assess non-linearity

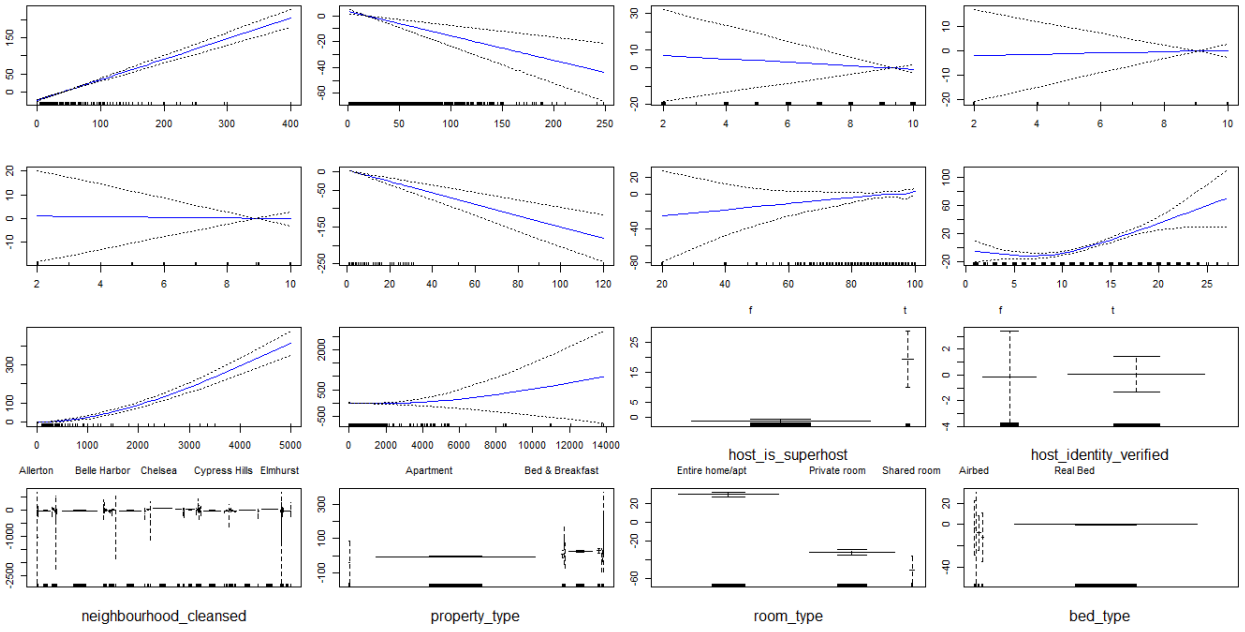
```
anova(gam.2, gam.3, test = "F")
```

```
## Analysis of Deviance Table
##
## Model 1: price ~ review_scores_rating + num_amenities + number_of_reviews +
##     review_scores_accuracy + review_scores_cleanliness + review_scores_location +
##     distance_in_meters
## Model 2: price ~ number_of_reviews + review_scores_accuracy + review_scores_cleanliness +
##     review_scores_location + ns(review_scores_rating, 4) + ns(num_amenities,
##     4) + ns(distance_in_meters, 4)
##   Resid. Df Resid. Dev Df Deviance      F    Pr(>F)
## 1     19382  374396690
## 2     19373  372083000  9  2313690 13.385 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Gam.3 plot: Plots of the relationship between each feature and the response in the fitted model



Gam.4 plot: Plots of the relationship between each feature and the response in the fitted model



Next iteration: Assessing different versions of GAM models

We explored different versions of GAM models by layering on more variables, as well as variables modeled in a non-linear approach to earlier models. In other words, our earlier GAM versions are subsets of the later GAM versions so we can sequentially compare the simpler model to the more complex models. This must be so in order to apply the `ANOVA()` function to test the null hypothesis that a model M_1 is sufficient to explain the data against the alternative hypothesis that a more complex model M_2 is required.

We finally arrived at a subset of variables for our last *gam.6* model by adding a few more predictors which we learned from the Lasso regression earlier carried meaningfully large non-zero coefficients, namely *host_response_time* and *host_has_profile_pic*.

Overview of four different GAM models we explored

```
> gam.2=gam(price ~
+                  review_scores_rating+
+                  Num_Amenities+
+                  cleaning_fee+
+                  number_of_reviews+
+                  review_scores_accuracy+
+                  review_scores_cleanliness+
+                  review_scores_location+
+                  distance_in_meters,
+                 data=datagam)

> gam.3=gam(price~
+                  cleaning_fee+
+                  number_of_reviews+
+                  review_scores_accuracy+
+                  review_scores_cleanliness+
+                  review_scores_location+
+                  ns(review_scores_rating,4)+ #set to 4 deg of freedom
+                  ns(Num_Amenities,4)+
+                  ns(security_deposit,4)+
+                  ns(distance_in_meters,4)

> gam.4=gam(price~
+              cleaning_fee+
+              number_of_reviews+
+              review_scores_accuracy+
+              review_scores_cleanliness+
+              review_scores_location+
+              minimum_nights+
+              ns(review_scores_rating,4)+ #set to 4 deg of freedom
+              ns(Num_Amenities,4)+
+              ns(security_deposit,4)+
+              ns(distance_in_meters,4)+
+              host_is_superhost+
+            host_identity_verified+
+            neighbourhood_cleansed+
+            property_type+
+            room_type+
+            bed_type,
+            data=datagam)

> gam.6=gam(price~
+              cleaning_fee+
+              number_of_reviews+
+              review_scores_accuracy+
+              review_scores_cleanliness+
+              review_scores_location+
+              minimum_nights+
+              ns(review_scores_rating,4)+
+              ns(Num_Amenities,4)+
+              ns(security_deposit,4)+
+              ns(distance_in_meters,4)+
```

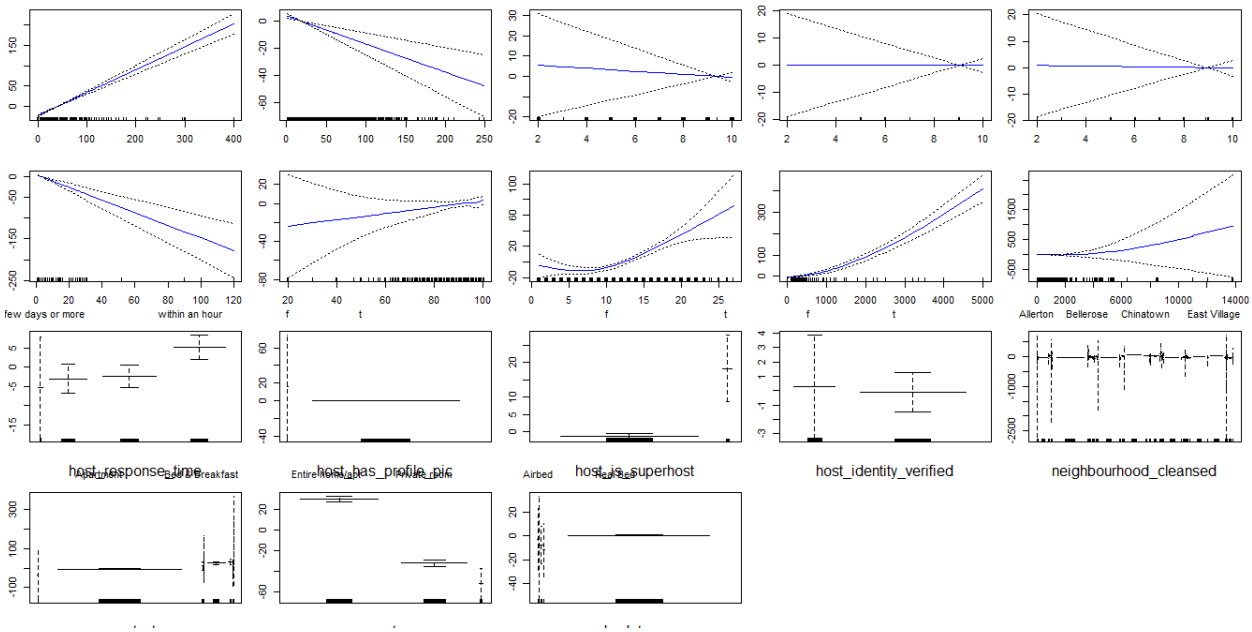


Figure 1: gam6 plot

```
+              host_response_time+ # added in gam.6 due to Lasso results
+              host_has_profile_pic+ # added in gam.6 due to Lasso results
+              host_is_superhost+
+              host_identity_verified+
+              neighbourhood_cleansed+
+              property_type+
+              room_type+
+              bed_type,
+            data=datagam)
```

Gam.6 model with most complete model subset: Plots of the relationship between each feature and the response in the fitted model

ANOVA model evaluation of above four GAM models

Based on the results of an ANOVA analysis of four GAM models - *gam.2, gam.3, gam.4*, and *gam.6* - we find that the miniscule p-value for *gam.4* of $< 2.2 \times 10^{-16}$ provides evidence the *gam.4* model, which contains an additional set of categorical predictors vs. *gam.3*'s subset, offers a better model fit than the earlier models.

Meanwhile, the p-value for *gam.6* of 0.025 in the ANOVA results suggests there is some evidence that the addition of the last model's predictors (suggested by our Lasso regression) collectively contribute to a better model fit, vs. the previous model *gam.4* which did not include those predictors.

```
anova(gam.2, gam.3, gam.4, gam.6, test = "F")
```

```
## Analysis of Deviance Table
##
## Model 1: price ~ review_scores_rating + num_amenities + number_of_reviews +
##     review_scores_accuracy + review_scores_cleanliness + review_scores_location +
##     distance_in_meters
## Model 2: price ~ number_of_reviews + review_scores_accuracy + review_scores_cleanliness +
##     review_scores_location + ns(review_scores_rating, 4) + ns(num_amenities,
```

```
##     4) + ns(distance_in_meters, 4)
## Model 3: price ~ number_of_reviews + review_scores_accuracy + review_scores_cleanliness +
##     review_scores_location + minimum_nights + ns(review_scores_rating,
##     4) + ns(num_amenities, 4) + ns(distance_in_meters, 4) + host_is_superhost +
##     host_identity_verified + property_type + room_type + bed_type
## Model 4: price ~ number_of_reviews + review_scores_accuracy + review_scores_cleanliness +
##     review_scores_location + minimum_nights + ns(review_scores_rating,
##     4) + ns(num_amenities, 4) + ns(distance_in_meters, 4) + host_response_time +
##     host_has_profile_pic + host_is_superhost + host_identity_verified +
##     property_type + room_type + bed_type
##   Resid. Df Resid. Dev Df Deviance        F  Pr(>F)
## 1     19382  374396690
## 2     19373  372083000  9  2313690  15.9711 < 2e-16 ***
## 3     19346  311493584 27 60589416 139.4136 < 2e-16 ***
## 4     19342  311336020  4   157564   2.4472 0.04418 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Selecting best model on performance based on AIC criteria

Finally, to apply another model performance assessment measure, we compared the AIC score across the above 4 GAM models. Our last model *gam.6* with the largest set of predictors achieved the lowest AIC score, suggesting this model, among the four we explored, is potentially GAM model offering the best model fit and predictive ability for y variable price.

```
AIC(gam.2)
```

```
## [1] 246391
```

```
AIC(gam.3)
```

```
## [1] 246288.8
```

```
AIC(gam.4)
```

```
## [1] 242896.5
```

```
AIC(gam.6)
```

```
## [1] 242894.7
```

Summary of GAM model fit for gam.6

```
summary(gam.6)
```

```
##
## Call: gam(formula = price ~ number_of_reviews + review_scores_accuracy +
##     review_scores_cleanliness + review_scores_location + minimum_nights +
##     ns(review_scores_rating, 4) + ns(num_amenities, 4) + ns(distance_in_meters,
##     4) + host_response_time + host_has_profile_pic + host_is_superhost +
##     host_identity_verified + property_type + room_type + bed_type,
##     data = gam.dataset)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -270.75  -47.09  -11.21   23.54 9679.81
##
## (Dispersion Parameter for gaussian family taken to be 16096.37)
##
```

```
##     Null Deviance: 395431690 on 19389 degrees of freedom
## Residual Deviance: 311336020 on 19342 degrees of freedom
## AIC: 242894.7
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                               Df    Sum Sq  Mean Sq   F value    Pr(>F)
## number_of_reviews              1    193024   193024   11.9918 0.0005355
## review_scores_accuracy         1    817072   817072   50.7612 1.080e-12
## review_scores_cleanliness      1    477260   477260   29.6502 5.238e-08
## review_scores_location         1   9092579  9092579  564.8838 < 2.2e-16
## minimum_nights                 1      2547     2547    0.1582 0.6907945
## ns(review_scores_rating, 4)    4    775125   193781   12.0388 9.005e-10
## ns(num_amenities, 4)           4  10438569  2609642  162.1261 < 2.2e-16
## ns(distance_in_meters, 4)      4   1561110   390277   24.2463 < 2.2e-16
## host_response_time             3     96642    32214    2.0013 0.1114545
## host_has_profile_pic           1       404      404    0.0251 0.8741061
## host_is_superhost              1      3373     3373    0.2096 0.6471223
## host_identity_verified         1     96754    96754    6.0109 0.0142263
## property_type                 18   1750919    97273    6.0432 5.939e-15
## room_type                      2  58608144 29304072 1820.5390 < 2.2e-16
## bed_type                       4    182148    45537    2.8290 0.0232587
## Residuals                  19342 311336020    16096
##
## number_of_reviews           ***
## review_scores_accuracy      ***
## review_scores_cleanliness   ***
## review_scores_location      ***
## minimum_nights
## ns(review_scores_rating, 4) ***
## ns(num_amenities, 4)        ***
## ns(distance_in_meters, 4)   ***
## host_response_time
## host_has_profile_pic
## host_is_superhost
## host_identity_verified      *
## property_type               ***
## room_type                   ***
## bed_type                    *
## Residuals
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Generalized Additive Models for Log Price

Why Model Logged Price

Due to the nature of housing prices, it makes sense intuitively to test a multiplicative relationship between predictors rather than something strictly additive (For example, having an extra bedroom might be correlated with a larger (absolute dollar amount) increase in price in a pricy neighborhood than in a cheaper one.). Because the raw price variable looks closer to an exponential distribution than a normal one, we logged it in many of our earlier visualizations and it looks like some of the variables have a relationship with the log providing further motivation for using the log of price as the dependant variable.

(Because the dependent variable has been logged, individual variable contributions should be interpreted as lifts percentage-wise to the non-logged price rather than dollar contributions.)

Variable and Parameter Selection

In order to better understand the relationships between explanatory variables and log price, we wanted to test for non-linear relationships as well as linear ones. We decided to try a generalized additive (in relation to the log) model in oto due this.

Before looking into non-linearity in the predictors, we first used an OLS regression to help us pick predictors. Due to the size of the dataset and number of predictors, many stepwise and subset selection models were too computationally expensive and those that did run (forward stepwise, lasso) did not result in usable subets. Therefore, we ran a linear regression using all the variables and looked at their statistical significance. (Fees and Neighborhoods were not included in this regression.)

Table 2: Highly Significant OLS Variables

X	Estimate	Std..Error	t.value	Pr...t..
(Intercept)	3.19	0.24	13.43	0
neighbourhood_group_cleansedManhattan	0.55	0.03	20.74	0
room_typePrivate room	-0.57	0.01	-97.87	0
room_typeShared room	-0.88	0.02	-55.58	0
accommodates	0.06	0.00	24.90	0
bathrooms	0.12	0.01	18.30	0
bedrooms	0.15	0.00	31.96	0
minimum_nights	-0.01	0.00	-11.80	0
availability_60	0.00	0.00	29.28	0
review_scores_location	0.10	0.00	38.30	0
review_scores_value	-0.06	0.00	-16.95	0
Amenities.airconditioning	0.08	0.01	10.86	0
Amenities.cabletv	0.05	0.00	10.82	0
Amenities.doorman	0.12	0.01	12.71	0
Amenities.indoorfireplace	0.13	0.01	11.87	0
neighborhood_clustercluster.5	0.19	0.02	10.81	0

The original baseline GAM Model included all variables which were highly significant (absolute value of T-Stat >10) in the OLS regression, except for neighborhood clusters (Some of the clusters had too few listings after missing ratings scores were removed to hold up to cross validation.). It also included all ratings and distance from subway because we were interested in transformations of those.

Next, we ran a GAM model and looked at it's RMSE as well as a testing RMSE (Average of 10 10-fold cross validation RMSEs). We tested models with different tranformations and compared both testing and training RMSEs

Table 3: Summary of Models Tested

Model	Training RMSE	Testing RMSE
All Linear	0.3222	0.3275
Base + Logged Distance	0.3222	0.3277
Base + Smooth Distance 2DF	0.3222	0.3277
Base + Smooth Distance 3DF	0.3221	0.328
Base + Smooth Distance 4DF	0.3221	0.3275
Base + Smooth Distance 5DF	0.322	0.3275

Model	Training RMSE	Testing RMSE
Base + Smooth Distance 6DF	0.3219	0.3275
Base + Log Accomodates	0.3216	0.327
Base + Smooth Accomodates 2 DF	0.3218	0.3275
Base + Smooth Accomodates 3 DF	0.3214	0.3272
Base + Smooth Accomodates 4 DF	0.3211	0.3267
Base + Smooth Accomodates 5 DF	0.3208	0.3263
Base + Smooth Accomodates 6 DF	0.3206	0.3264
Base + Smooth Accomodates 7 DF	0.3205	0.3265
Base + Log Location Review +Smooth Accomodates 5 DF	0.3233	0.3295
Base + Log Value Review +Smooth Accomodates 5 DF	0.3207	0.3266
Base + Log Communication Review +Smooth Accomodates 5 DF	0.3207	0.3268
Base + Log Check in Review +Smooth Accomodates 5 DF	0.3208	0.3263
Base + Log Cleanliness Review +Smooth Accomodates 5 DF	0.321	0.3266
Base + Log Accuracy Review +Smooth Accomodates 5 DF	0.3208	0.3269
Base + Log Rating +Smooth Accomodates 5 DF	0.3211	0.3268
Base + Log Min Nights +Smooth Accomodates 5 DF	0.3209	0.3268
Base + Smooth Min Nights 2 DF + Smooth Accomodates 5 DF	0.3205	0.326
Base + Smooth Min Nights 3 DF + Smooth Accomodates 5 DF	0.3205	0.3262
Base + Smooth Min Nights 4 DF + Smooth Accomodates 5 DF	0.3204	0.3262
Base + Smooth Min Nights 5 DF + Smooth Accomodates 5 DF	0.3204	0.3262
Base + log Bedrooms + Smooth Min Nights 2 DF + Smooth Accomodates 5 DF	0.3226	0.3286
Base + log Bathrooms + Smooth Min Nights 2 DF + Smooth Accomodates 5 DF	0.3208	0.3266
Base + Smooth Min Nights 2 DF + Smooth Accomodates 5 DF - Check in Review - accuracy review	0.3206	0.3258
Base - Check in Review - accuracy review	0.3222	0.3276

We tested logs as well as smoothing parameters (on variables with ranges large enough to support them) for quantitative variables. Most did not have a large enough/noticable effect on the testing RMSE to justify them (even if non-linearity was statistically sigificant), but a few did. Two of the ratings variables were never significant in the model and removed (which reduced testing error). Final model output is below:

Table 4: Anova for Parametric Effects

X	Df	Sum.Sq	Mean.Sq	F.value	Pr..F.
room_type	2	3330.36	1665.18	16185.68	0
neighbourhood_group_cleansed	4	515.83	128.96	1253.47	0

X	Df	Sum.Sq	Mean.Sq	F.value	Pr..F.
bedrooms	1	704.94	704.94	6852.09	0
bathrooms	1	101.39	101.39	985.55	0
s(minimum_nights, 2)	1	14.92	14.92	145.06	0
review_scores_rating	1	61.99	61.99	602.52	0
review_scores_cleanliness	1	3.33	3.33	32.40	0
review_scores_communication	1	0.95	0.95	9.23	0
review_scores_location	1	152.93	152.93	1486.47	0
review_scores_value	1	44.41	44.41	431.63	0
Amenities.indoorfireplace	1	19.58	19.58	190.35	0
Amenities.airconditioning	1	34.61	34.61	336.36	0
Amenities.cabletv	1	33.33	33.33	323.94	0
Amenities.doorman	1	52.39	52.39	509.28	0
s(accommodates, 5)	1	96.86	96.86	941.45	0
distance_in_meters	1	3.53	3.53	34.29	0
Residuals	21664	2228.79	0.10	NA	NA

Table 5: Anova for Non-Parametric Effects

X	Npar.Df	Npar.F	Pr.F.
(Intercept)	NA	NA	NA
room_type	NA	NA	NA
neighbourhood_group_cleansed	NA	NA	NA
bedrooms	NA	NA	NA
bathrooms	NA	NA	NA
s(minimum_nights, 2)	1	32.24	0
review_scores_rating	NA	NA	NA
review_scores_cleanliness	NA	NA	NA
review_scores_communication	NA	NA	NA
review_scores_location	NA	NA	NA
review_scores_value	NA	NA	NA
Amenities.indoorfireplace	NA	NA	NA
Amenities.airconditioning	NA	NA	NA
Amenities.cabletv	NA	NA	NA
Amenities.doorman	NA	NA	NA
s(accommodates, 5)	4	47.11	0
distance_in_meters	NA	NA	NA

The AIC of this model is 12,254 and the BIC is 12,430. (Without non-parametric effects, AIC = 12,467 and BIC = 12,642.) All variables and effects are statistically significant.

Residuals on the log price are fairly normally distributed. Looking at the predicted value in terms of the actual price, the model pretty clearly doesnt predict higher prices as accurately as lower ones (a function of the log transformation.)

Looking at the model coefficients, space (accomodates, bedrooms, bathrooms, privacy) are predictors of higher prices. So are indoor fireplaces, doormen, Cable TV and Air Conditioning. Most of the review scores are positive predictors. The few that are negative have small coefficients and are probably balancing out the other review variables (all are highly correlated). Being closer to Manhatten and the subway are predictors of pricier listings. Minimum nights is a negative predictor of price.

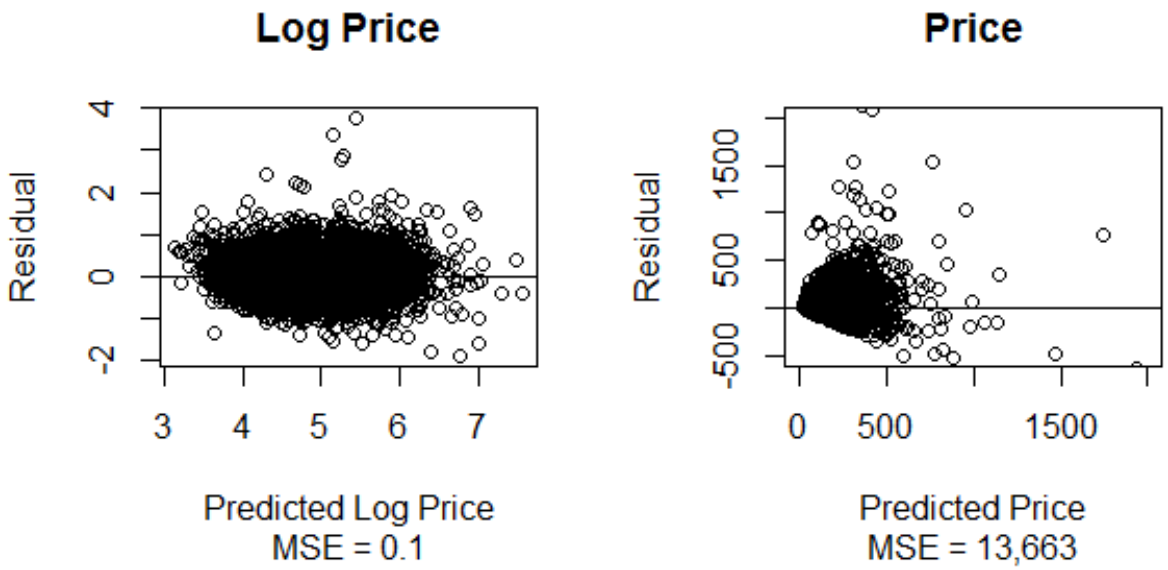


Figure 2: GAM Residuals

Findings Summary: Comparing the models

Some counter-intuitive insights gleaned from Explorative Data Analysis and Regression analyses

- The majority of airbnb listings have a very high review rating score of 90+ - does this suggest review ratings overall may not help distinguish between the true attractiveness or quality of listings as much as we would have thought?
- There is not a high association between the variables of price and review rating score per our regression coefficient output;
- While our custom-created variable *Distance to Nearest Subway* does not appear to be a meaningful predictor of either *Price* or *Superhost* status, we noticed in our GAM regression that *review_scores_location* variable has a very statistically significant positive coefficient in its relationship to *Price*. Perhaps the preference for proximity to public transportation is reflected instead in this variable;
- To our surprise, the average Airbnb rental price for listings in Manhattan is lower than those for other boroughs, we suspect due to a much higher number of host listings and thus more intense competition within the Manhattan market.

Inference-based analyses on the Price variable vs. Property/Room Type and Superhost Status

- Our ANOVA analysis demonstrates a real difference in price based on both property type and room type.
- The Mann-Whitney-Wilcox tests points to a difference in price range/distribution based on whether or not a host is a superhost.
- We can infer from the decision trees models value ranges that would lead to a host being a superhost: high number of reviews, high review score, high host response rate, and high rate of reviews per month.

Regression Models: Linear Regression with Lasso & General Addictive Models

- Applying a Lasso regression with regularization to infer about the subset of predictor variables having non-zero explanatory power for the Price variable, we learned that two major groups of predictors appear to have more meaningful coefficients than the rest - predictors on host information and neighborhoods.
- From our GAM models on the Price variable, we can see from the Summary output of our highest performing gam.6 model that the most meaningful predictors with the highest relevance in this regression are: *room_type*, *review_scores_location*, *ns(security_deposit, 4)*, *review_scores_accuracy*, and *ns(Num_Amenities, 4)*. [Note: `ns()` indicates a natural spline technique on the variable was applied]. Furthermore, we were able to conclude with an ANOVA analysis on the different models that several variables do appear to have a non-linear relationship with the y variable.
- We were also able to fit a highly significant GAM model on logged price. While the MSE of this model is larger than the MSE of the price model alone when predictions are put on a non-logged scale, we would recommend further analysis before drawing conclusions that a linear relationship is better than a log-linear one here. (It's possible that high residuals on the outliers make MSE a less reliable comparison if we care less about the outliers).

Conclusions

The results of this project are based on our collaborative work over the course of a short period of time. Given more time, we would like to test additional information such as neighborhood population density, household income, and zoning information (% residential) as well as use more sophisticated techniques such as machine learning and natural language processing.

Appendix

R Code

General Data Exploration

Correlation of each Variable with Price

host_response_rate	1.00
host_acceptance_rate	0.07
bathrooms	0.01
bedrooms	0.04
beds	0.06
guests_included	0.04
minimum_nights	-0.02
maximum_nights	0.00
availability_365	-0.06
number_of_reviews	0.11
review_scores_rating	0.04
review_scores_accuracy	0.03
review_scores_cleanliness	0.09
review_scores_checkin	0.07
review_scores_communication	0.09
review_scores_location	-0.02
review_scores_value	0.05
reviews_per_month	0.19
Host.Ver.email	0.05

Host.Ver.facebook	-0.03
Host.Ver.jumio	0.06
Host.Ver.kba	0.02
Host.Ver.linkedin	0.05
Host.Ver.none	NA
Host.Ver.phone	0.00
Host.Ver.sentid	0.01
Host.Ver.weibo	0.01
distance_in_meters	0.03

Data Manipulation and Cleaning

General Data Cleaning

Variables Excluded

Because image and text analysis are out of scope for this project, the following variables in the original dataset were not considered for this analysis:

```
##  [1] "listing_url"           "summary"
##  [3] "description"           "notes"
##  [5] "thumbnail_url"         "picture_url"
##  [7] "host_url"              "host_thumbnail_url"
##  [9] "name"                  "space"
## [11] "neighborhood_overview" "transit"
## [13] "medium_url"            "xl_picture_url"
## [15] "host_about"            "host_picture_url"
```

Some variables were excluded due to having little or no information in New York listings:

```
## [1] "jurisdiction_names"  "requires_license"    "experiences_offered"
## [4] "square_feet"         "license"
```

Others were redundant or not relevant for this analysis.

```
##  [1] "scrape_id"             "host_location"
##  [3] "neighbourhood"         "smart_location"
##  [5] "country"               "weekly_price"
##  [7] "calendar_last_scraped" "availability_30"
##  [9] "last_review"           "city"
## [11] "host_neighbourhood"    "last_scraped"
## [13] "street"                "market"
## [15] "country_code"          "calendar_updated"
## [17] "monthly_price"         "has_availability"
## [19] "availability_90"       "first_review"
## [21] "host_listings_count"   "state"
## [23] "host_name"
```

General Data Cleaning

```
# Read in Airbnb listings data
listings <- read.csv("listings.csv", stringsAsFactors = FALSE)

# Create Subset for Analysis
drops <- c("listing_url", "scrape_id", "last_scraped",
```

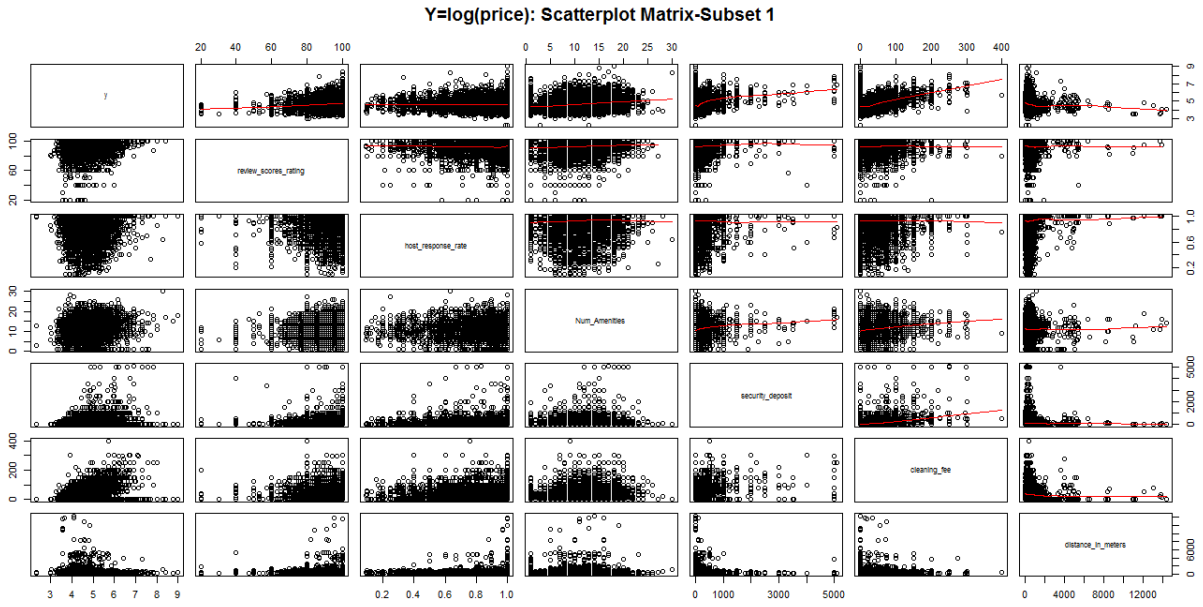


Figure 3: Scatterplot xsubset1

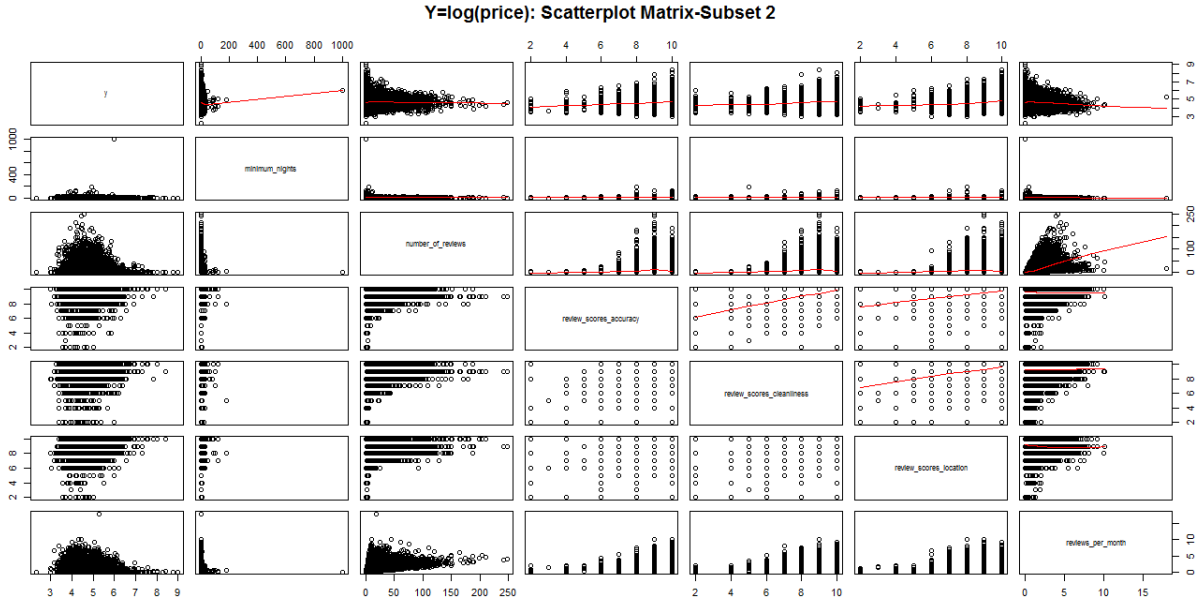


Figure 4: Scatterplot xsubset2

```
    "name", "summary", "space", "description", "neighborhood_overview",
    "notes", "transit", "thumbnail_url", "medium_url",
    "picture_url", "xl_picture_url", "host_url", "host_location",
    "host_about", "host_thumbnail_url", "host_picture_url",
    "street", "neighbourhood", "market", "smart_location",
    "country_code", "country", "calendar_updated",
    "weekly_price", "monthly_price", "calendar_last_scraped",
    "has_availability", "availability_30", "availability_90",
    "first_review", "last_review", "jurisdiction_names",
    "requires_license", "license", "experiences_offered",
    "host_listings_count", "city", "state", "host_neighbourhood",
    "square_feet")

air.bnb.dataset <- listings[, !(names(listings) %in%
    drops)]

# Identify Factor Variables
air.bnb.dataset$room_type <- (as.factor(air.bnb.dataset$room_type))
air.bnb.dataset$is_location_exact <- (as.factor(air.bnb.dataset$is_location_exact))
air.bnb.dataset$property_type <- (as.factor(air.bnb.dataset$property_type))
air.bnb.dataset$bed_type <- (as.factor(air.bnb.dataset$bed_type))
air.bnb.dataset$host_response_time <- (as.factor(air.bnb.dataset$host_response_time))
air.bnb.dataset$host_is_superhost <- (as.factor(air.bnb.dataset$host_is_superhost))
air.bnb.dataset$host_identity_verified <- (as.factor(air.bnb.dataset$host_identity_verified))
air.bnb.dataset$host_has_profile_pic <- (as.factor(air.bnb.dataset$host_has_profile_pic))
air.bnb.dataset$neighbourhood_cleansed <- (as.factor(air.bnb.dataset$neighbourhood_cleansed))
air.bnb.dataset$neighbourhood_group_cleansed <- (as.factor(air.bnb.dataset$neighbourhood_group_cleansed
air.bnb.dataset$instant_bookable <- (as.factor(air.bnb.dataset$instant_bookable))
air.bnb.dataset$cancellation_policy <- (as.factor(air.bnb.dataset$cancellation_policy))
air.bnb.dataset$require_guest_profile_picture <- (as.factor(air.bnb.dataset$require_guest_profile_pictu
air.bnb.dataset$require_guest_phone_verification <- (as.factor(air.bnb.dataset$require_guest_phone_veri

# Dates
air.bnb.dataset$host_since <- (as.Date(air.bnb.dataset$host_since))

# Numbers with formatting
remove.punct.from.num <- function(num_field) {
    # Remove punctuation from numbers with formatting
    # (ex. '86%')
    temp <- gsub("[[:punct:]]", "", num_field)
    if (is.na(num_field)) {
        return(num_field)
    }
    if (is.na(temp)) {
        return(num_field)
    }
    if (temp == "NA") {
        return(NA)
    } else {
        return(as.numeric(temp))
    }
}
```

```r
air.bnb.dataset$host_response_rate <- sapply(air.bnb.dataset$host_response_rate,
    FUN = remove.punct.from.num)/100
air.bnb.dataset$host_acceptance_rate <- sapply(air.bnb.dataset$host_acceptance_rate,
    FUN = remove.punct.from.num)/100
air.bnb.dataset$price <- sapply(air.bnb.dataset$price,
    FUN = remove.punct.from.num)
air.bnb.dataset$price <- as.numeric(air.bnb.dataset$price)/100
air.bnb.dataset$security_deposit <- sapply(air.bnb.dataset$security_deposit,
    FUN = remove.punct.from.num)/100
air.bnb.dataset$cleaning_fee <- sapply(air.bnb.dataset$cleaning_fee,
    FUN = remove.punct.from.num)/100
air.bnb.dataset$extra_people <- sapply(air.bnb.dataset$extra_people,
    FUN = remove.punct.from.num)/100

# Deal with NAs in optional costs
air.bnb.dataset$cleaning_fee[is.na(air.bnb.dataset$cleaning_fee)] <- 0
air.bnb.dataset$security_deposit[is.na(air.bnb.dataset$security_deposit)] <- 0
air.bnb.dataset$extra_people[is.na(air.bnb.dataset$extra_people)] <- 0


# Other numeric variables
air.bnb.dataset$host_total_listings_count <- as.numeric(air.bnb.dataset$host_total_listings_count)
air.bnb.dataset$guests_included <- as.numeric(air.bnb.dataset$guests_included)
air.bnb.dataset$extra_people <- as.numeric(air.bnb.dataset$extra_people)
air.bnb.dataset$minimum_nights <- as.numeric(air.bnb.dataset$minimum_nights)
air.bnb.dataset$maximum_nights <- as.numeric(air.bnb.dataset$maximum_nights)
air.bnb.dataset$number_of_reviews <- as.numeric(air.bnb.dataset$number_of_reviews)
air.bnb.dataset$review_scores_rating <- as.numeric(air.bnb.dataset$review_scores_rating)
air.bnb.dataset$review_scores_accuracy <- as.numeric(air.bnb.dataset$review_scores_accuracy)
air.bnb.dataset$review_scores_cleanliness <- as.numeric(air.bnb.dataset$review_scores_cleanliness)
air.bnb.dataset$review_scores_checkin <- as.numeric(air.bnb.dataset$review_scores_checkin)
air.bnb.dataset$review_scores_communication <- as.numeric(air.bnb.dataset$review_scores_communication)
air.bnb.dataset$review_scores_location <- as.numeric(air.bnb.dataset$review_scores_location)
air.bnb.dataset$review_scores_value <- as.numeric(air.bnb.dataset$review_scores_value)
air.bnb.dataset$calculated_host_listings_count <- as.numeric(air.bnb.dataset$calculated_host_listings_c
air.bnb.dataset$reviews_per_month <- as.numeric(air.bnb.dataset$reviews_per_month)

# Remove zip codes from outside NYC
to.remove <- which(air.bnb.dataset$zipcode %in% c("07030",
    "07105", "07712", "12335", "12337", "94103", "99135"))
air.bnb.dataset <- air.bnb.dataset[-to.remove, ]

# Deal with bad zip codes
bad.zip <- function(zip) {
    if (nchar(zip) == 5) {
        return(zip)
    } else if (zip == "8456422473 call for more details") {
        return("")
    } else if (zip == "1m") {
        return("")
    } else if (nchar(zip) > 5) {
        zip <- substr(zip, 1, 5)
        return(zip)
```

```
    } else {
        return(zip)
    }

}

air.bnb.dataset$zipcode <- sapply(air.bnb.dataset$zipcode,
    FUN = bad.zip)

# Use Laplace method to calculate price per room
air.bnb.dataset$price.per.room <- air.bnb.dataset$price/(air.bnb.dataset$bedrooms +
    1)
```

Number of hosts in a listing

```
# Number of hosts in a listing

find.all.hosts <- function(host_names) {
    # Compute total number of hosts in a listing
    n.hosts <- 0

    # Find number of '&' or ' And ' strings
    n.amp <- length(grep("&", host_names))
    n.and <- length(grep(" And ", host_names))
    if (n.amp > 0 | n.and > 0) {
        # If there is a match, then there are at least two
        # hosts per listing More hosts may appear in a
        # comma-separated list Find the number of commas as
        # a proxy for the remaining hosts
        n.commas <- attr(regexpr(", ", host_names),
            "match.length")
        if (n.commas > 0) {
            n.hosts <- 2 + n.commas
        } else {
            n.hosts <- 2
        }
    } else {
        n.hosts <- 1
    }
    return(n.hosts)
}



air.bnb.dataset$number.of.hosts <- sapply(air.bnb.dataset$host_name,
    FUN = find.all.hosts)

air.bnb.dataset$host_name <- NULL
```

Amenities and Host Verifications

```
# Amenities and Host Verifications
```

```r
# Turn spaces into periods and commas into spaces
# in amenities
air.bnb.dataset$amenities <- gsub(" ", ".", air.bnb.dataset$amenities)
air.bnb.dataset$amenities <- gsub(",", " ", air.bnb.dataset$amenities)

# Create Bags of words
hostVers <- create_matrix(air.bnb.dataset$host_verifications,
    language = "english", removeNumbers = TRUE, removeStopwords = TRUE,
    removePunctuation = TRUE, removeSparseTerms = FALSE,
    toLower = TRUE, stemWords = FALSE)

amenities <- create_matrix(air.bnb.dataset$amenities,
    language = "english", removeNumbers = TRUE, removeStopwords = TRUE,
    removePunctuation = TRUE, removeSparseTerms = FALSE,
    toLower = TRUE, stemWords = FALSE)


# Join to dataset
hVers <- (as.data.frame(as.matrix(hostVers)))
names(hVers) <- paste("Host.Ver.", names(hVers), sep = "")

amen <- (as.data.frame(as.matrix(amenities)))
names(amen) <- paste("Amenities.", names(amen), sep = "")

# Calculate total number of amenities per listing
num_amen <- apply(amen, FUN = sum, MARGIN = 1)

air.bnb.dataset <- cbind(air.bnb.dataset, hVers)
air.bnb.dataset$num_amenities <- num_amen
```

Distance to Closest Subway Station

```r
# This is the R code for calculating the distance
# to the closest subway station.  Data Source:
# subway
# stations:https://data.ny.gov/Transportation/NYC-Transit-Subway-Entrance-And-Exit-Data/i9wp-a4ja
# airbnb listing:
# http://insideairbnb.com/get-the-data.html


# Based on:
# https://www.r-bloggers.com/great-circle-distance-calculations-in-r/
# and
# https://github.com/caesar0301/omniR/blob/master/R/gcd.R

# Convert degrees to radians
deg2rad <- function(deg) {
    return(deg * pi/180)
}

# Calculates the geodesic distance between two
# points specified by radian latitude/longitude
# using Vincenty inverse formula for ellipsoids
```

```r
# (vif)
gcd.vif.customized <- function(long1, lat1, long2,
    lat2) {



    # Convert degrees to radians
    long1 <- deg2rad(long1)
    lat1 <- deg2rad(lat1)
    long2 <- deg2rad(long2)
    lat2 <- deg2rad(lat2)



    # WGS-84 ellipsoid parameters
    a <- 6378137  # length of major axis of the ellipsoid (radius at equator)
    b <- 6356752.314245  # length of minor axis of the ellipsoid (radius at the poles)
    f <- 1/298.257223563  # flattening of the ellipsoid

    L <- long2 - long1  # difference in longitude
    U1 <- atan((1 - f) * tan(lat1))  # reduced latitude
    U2 <- atan((1 - f) * tan(lat2))  # reduced latitude
    sinU1 <- sin(U1)
    cosU1 <- cos(U1)
    sinU2 <- sin(U2)
    cosU2 <- cos(U2)

    cosSqAlpha <- NULL
    sinSigma <- NULL
    cosSigma <- NULL
    cos2SigmaM <- NULL
    sigma <- NULL

    lambda <- L
    lambdaP <- 0
    iterLimit <- 100
    while (abs(lambda - lambdaP) > 1e-12 & iterLimit >
        0) {
        sinLambda <- sin(lambda)
        cosLambda <- cos(lambda)
        sinSigma <- sqrt((cosU2 * sinLambda) * (cosU2 *
            sinLambda) + (cosU1 * sinU2 - sinU1 * cosU2 *
            cosLambda) * (cosU1 * sinU2 - sinU1 * cosU2 *
            cosLambda))
        if (sinSigma == 0)
            return(0)  # Co-incident points
        cosSigma <- sinU1 * sinU2 + cosU1 * cosU2 *
            cosLambda
        sigma <- atan2(sinSigma, cosSigma)
        sinAlpha <- cosU1 * cosU2 * sinLambda/sinSigma
        cosSqAlpha <- 1 - sinAlpha * sinAlpha
        cos2SigmaM <- cosSigma - 2 * sinU1 * sinU2/cosSqAlpha
        if (is.na(cos2SigmaM))
```

```
            cos2SigmaM <- 0  # Equatorial line: cosSqAlpha=0
        C <- f/16 * cosSqAlpha * (4 + f * (4 - 3 *
            cosSqAlpha))
        lambdaP <- lambda
        lambda <- L + (1 - C) * f * sinAlpha * (sigma +
            C * sinSigma * (cos2SigmaM + C * cosSigma *
                (-1 + 2 * cos2SigmaM * cos2SigmaM)))
        iterLimit <- iterLimit - 1
    }
    if (iterLimit == 0)
        return(NA)  # formula failed to converge
    uSq <- cosSqAlpha * (a * a - b * b)/(b * b)
    A <- 1 + uSq/16384 * (4096 + uSq * (-768 + uSq *
        (320 - 175 * uSq)))
    B <- uSq/1024 * (256 + uSq * (-128 + uSq * (74 -
        47 * uSq)))
    deltaSigma = B * sinSigma * (cos2SigmaM + B/4 *
        (cosSigma * (-1 + 2 * cos2SigmaM^2) - B/6 *
            cos2SigmaM * (-3 + 4 * sinSigma^2) * (-3 +
            4 * cos2SigmaM^2)))
    s <- b * A * (sigma - deltaSigma)

    return(s)  # Distance in meters
}


distance_vector_function <- function(listing_longitude,
    listing_latitude, station_longitude, station_latitude) {
    # Calculate the shortest distance between a listing
    # and a subway station
    n.list.lat <- length(listing_latitude)
    n.list.long <- length(listing_longitude)
    stopifnot(n.list.lat == n.list.long)

    n.station.lat <- length(station_latitude)
    n.station.long <- length(station_longitude)
    stopifnot(n.station.lat == n.station.long)

    # Stores distances between a given listing and all
    # stations
    distance_temp <- rep(0, n.station.lat)
    # Stores min distance from a listing to a station
    distance_vector <- rep(0, n.list.lat)

    # Loop through all listings
    for (i in 1:n.list.lat) {
        # Loop through all stations
        for (j in 1:n.station.lat) {
            # Calculate distance between listing and station
            distance_temp[j] <- gcd.vif.customized(listing_longitude[i],
                listing_latitude[i], station_longitude[j],
                station_latitude[j])
        }
```

```
        # Preserve distance to station that is the shortest
        distance_vector[i] <- min(distance_temp)
    }
    return(distance_vector)
}

# Idea: Calculate Voronoi diagram of subway
# stations, using geodesic distance as distance
# metric Then, map each listing into Voronoi
# diagram. The station corresponding to the Voronoi
# cell in which each listing is located will be
# considered the closest subway station. Then
# calculate the geodesic distance between each
# listing and its closest station.

# Alternative: 'Color-code' subway stations and
# listings, then plot them all on same Voronoi
# diagram

listing_location <- air.bnb.dataset[, c("latitude",
    "longitude")]
row_station <- read.csv("NYC_Transit_Subway_Entrance_And_Exit_Data.csv")
station <- row_station[, c("Station.Latitude", "Station.Longitude")]

distances <- distance_vector_function(air.bnb.dataset$longitude,
    air.bnb.dataset$latitude, row_station$Station.Longitude,
    row_station$Station.Latitude)

air.bnb.dataset$distance_in_meters <- distances


air.bnb.dataset <- droplevels(air.bnb.dataset)
```

EDA

Tables and Summary Statistics

```
# Summary statistics of price
summary(air.bnb.dataset$price)
# Basic histogram of price
ggplot(data = air.bnb.dataset, aes(log(price + 1))) +
    geom_histogram() + labs(x = "log(Price)", title = "Histogram of log of Nightly Price")

# Basic summary statistics of superhost indicator
# variable
summary(air.bnb.dataset$host_is_superhost)

# Basic scatter plot of price & rating
ggplot(data = air.bnb.dataset, aes(x = log(price +
    1), y = review_scores_rating)) + geom_point(shape = 1) +
    labs(x = "log(Nightly Price)", y = "Review Score")

# Summary statistics & histogram of distance to
```

```r
# nearest subway
summary(air.bnb.dataset$distance_in_meters)
ggplot(data = air.bnb.dataset, aes(log(distance_in_meters +
    1))) + geom_histogram() + labs(x = "log(Distance)",
    title = "Histogram of log of Distance to Nearest Subway Station")

# Basic scatter plot of distance to nearest subway
# & price
ggplot(air.bnb.dataset, aes(x = log(distance_in_meters +
    1), y = log(price + 1))) + geom_point(shape = 1) +
    labs(x = "log(Distance)", y = "log(Nightly Price)")

# Basic scatter plot of distance to nearest subway
# & review score
ggplot(air.bnb.dataset, aes(x = log(distance_in_meters +
    1), y = review_scores_rating)) + geom_point(shape = 1) +
    labs(x = "log(Distance)", y = "Review Score")

# Box plots of superhost & price (justification of
# Mann-Whitney Wilcoxon test)
ggplot(air.bnb.dataset, aes(x = host_is_superhost,
    y = log(price + 1))) + geom_boxplot() + labs(x = "Superhost Flag",
    y = "log(Nightly Price)")

# Box plots of superhost & distance to subway
# (justification of Mann-Whitney Wilcoxon test)
ggplot(air.bnb.dataset, aes(x = host_is_superhost,
    y = log(distance_in_meters + 1))) + geom_boxplot() +
    labs(x = "Superhost Flag", y = "log(Distance)")

# Plot of NYC borough & price
ggplot(air.bnb.dataset, aes(x = neighbourhood_group_cleansed,
    y = log(price + 1), fill = neighbourhood_group_cleansed)) +
    geom_boxplot() + ggtitle("Boxplots of prices for each Borough") +
    ylab("Log of Prices") + xlab("Borough")


# Table of NYC borough & superhosts
table(air.bnb.dataset$neighbourhood_group_cleansed,
    air.bnb.dataset$host_is_superhost)
```

Locations of Superhosts

```r
# Plot locations of superhosts
air.bnb.dataset.Superhost.Loc <- air.bnb.dataset[which(air.bnb.dataset$is_location_exact ==
    "t" & air.bnb.dataset$host_is_superhost == "t"),
    c("latitude", "longitude")]


map <- get_map(location = "New York City, NY", zoom = 10)
mapPoints <- ggmap(map) + geom_point(aes(x = longitude,
    y = latitude), data = air.bnb.dataset.Superhost.Loc,
    alpha = 0.5) + labs(title = "Locations of Superhosts")
```

```
mapPoints
```

Locations of most and least expensive

```
# Plot locations of cheapest & most expensive
# locations
air.bnb.dataset.Cheap.Loc <- air.bnb.dataset[which(air.bnb.dataset$is_location_exact ==
    "t" & air.bnb.dataset$price <= 30), c("latitude",
    "longitude")]
No_Cheap <- nrow(air.bnb.dataset.Cheap.Loc)
air.bnb.dataset.Pricey.Loc <- air.bnb.dataset[which(air.bnb.dataset$is_location_exact ==
    "t" & air.bnb.dataset$price >= 1500), c("latitude",
    "longitude")]
No_Pricey <- nrow(air.bnb.dataset.Pricey.Loc)
air.bnb.dataset.Cheap.Loc <- cbind(air.bnb.dataset.Cheap.Loc,
    rep("Cheapest", No_Cheap))
names(air.bnb.dataset.Cheap.Loc)[3] <- "Cost"
air.bnb.dataset.Pricey.Loc <- cbind(air.bnb.dataset.Pricey.Loc,
    rep("Priciest", No_Pricey))
names(air.bnb.dataset.Pricey.Loc)[3] <- "Cost"
air.bnb.dataset.AllPrices <- rbind(air.bnb.dataset.Cheap.Loc,
    air.bnb.dataset.Pricey.Loc)

map <- get_map(location = "New York City, NY", zoom = 11)
mapPoints <- ggmap(map) + geom_point(aes(x = longitude,
    y = latitude), data = air.bnb.dataset.AllPrices,
    alpha = 0.5) + facet_grid(. ~ Cost)

mapPoints
```

Correlations with Price and Review Scores

```
# Boxplots of Property Types vs. Review Scores
layout(c(1, 1))

ggplot(air.bnb.dataset, aes(x = property_type, y = review_scores_rating)) +
    geom_boxplot() + theme(axis.text.x = element_text(angle = 90,
    size = 12)) + labs(x = "Property Type", y = "Review Score")

NumericVariables <- c("distance_in_meters", "Amenities.wirelessinternet",
    "Couple", "Host.Ver.email", "Host.Ver.facebook",
    "Host.Ver.googl", "Host.Ver.jumio", "Host.Ver.kba",
    "Host.Ver.linkedin", "Host.Ver.manualofflin", "Host.Ver.manualonlin",
    "Host.Ver.none", "Host.Ver.phone", "Host.Ver.photograph",
    "Host.Ver.review", "Host.Ver.sentid", "Host.Ver.weibo",
    "Amenities.aircondit", "Amenities.breakfast", "Amenities.buzzerwirelessintercom",
    "Amenities.cabletv", "Amenities.carbonmonoxidedetector",
    "Amenities.cat", "Amenities.dog", "Amenities.doorman",
    "Amenities.dryer", "Amenities.elevatorinbuild",
    "Amenities.essenti", "Amenities.familykidfriend",
    "Amenities.fireextinguish", "Amenities.firstaidkit",
    "Amenities.freeparkingonpremis", "Amenities.gym",
    "Amenities.heat", "Amenities.hottub", "Amenities.indoorfireplac",
```

```
    "Amenities.internet", "Amenities.kitchen", "Amenities.otherpet",
    "Amenities.petsallow", "Amenities.petsliveonthisproperti",
    "Amenities.pool", "Amenities.safetycard", "Amenities.shampoo",
    "Amenities.smokedetector", "Amenities.smokingallow",
    "Amenities.suitableforev", "Amenities.washer",
    "Amenities.washerdry", "Amenities.wheelchairaccess",
    "calculated_host_air.bnb.dataset_count", "reviews_per_month",
    "number_of_reviews", "review_scores_rating", "review_scores_accuracy",
    "review_scores_cleanliness", "review_scores_checkin",
    "review_scores_communication", "review_scores_location",
    "review_scores_value", "minimum_nights", "maximum_nights",
    "availability_30", "availability_365", "guests_included",
    "bathrooms", "bedrooms", "beds", "host_total_air.bnb.dataset_count",
    "host_response_rate", "host_acceptance_rate")

air.bnb.dataset.Numeric <- air.bnb.dataset[, (names(air.bnb.dataset) %in%
    NumericVariables)]
AllCor <- cor(air.bnb.dataset.Numeric, air.bnb.dataset[,
    c("price")])
# Correlation between each numeric variable and
# price
AllCor
```

```
## Attempt 1: Map average price of listings per zip
## code
price_average <- aggregate(air.bnb.dataset$price, by = list(air.bnb.dataset$zipcode),
    FUN = "mean", na.rm = TRUE)
names(price_average) <- c("zipcode", "mean.price")
price_average$mean.price <- round(price_average$mean.price,
    2)

# Download zip code boundaries for NYC
url <- paste("https://data.cityofnewyork.us/", "api/views/i8iw-xf4u/files/",
    "YObIR0MbpUVA0EpQzZSq5x55FzKGM2ejSeahdvjqR20?",
    "filename=ZIP_CODE_040114.zip", sep = "")
fil <- basename(url)
base.loc <- regexec("(?<=filename=).*[.]zip", fil,
    perl = TRUE)
base.str <- substr(fil, base.loc, nchar(fil))
if (!file.exists(base.str)) {
    download.file(url, base.str)
}

fils <- unzip(base.str, exdir = "nyc_zip")

# Read shapefiles
nyc <- readOGR("nyc_zip", ogrListLayers(fils[1])[1],
    stringsAsFactors = FALSE)
nyc@data$id <- rownames(nyc@data)
nyc.points <- fortify(nyc, region = "id")

# Partition price data into ranges/intervals
cut.dat <- cut(price_average$mean.price, breaks = nclass.FD(price_average$mean.price),
    include.lowest = TRUE)
```

```
ranges <- table(cut.dat)

price_average$price.range <- as.integer(cut.dat)

# Join price data to shapefile data
nyc@data <- merge(price_average, nyc@data, by.x = "zipcode",
    by.y = "ZIPCODE", all.y = TRUE)

nyc.df <- merge(nyc.points, nyc@data, by = "id")


gg <- ggplot()
gg <- gg + geom_map(data = nyc.df, map = nyc.df, aes(x = long,
    y = lat, map_id = id, fill = mean.price), color = "black",
    size = 0.25)
gg <- gg + coord_equal()
gg <- gg + theme_map()
gg <- gg + theme(legend.position = "bottom")
gg <- gg + scale_fill_gradientn(colours = terrain.colors(10))

gg
```

```
## Attempt 2: Map average price of listings per zip
## code, dividing price by (number of bedrooms plus
## 1)


price_per_room_average <- aggregate(air.bnb.dataset$price.per.room,
    by = list(air.bnb.dataset$zipcode), FUN = "mean",
    na.rm = TRUE)
names(price_per_room_average) <- c("zipcode", "mean.price.per.room")
price_per_room_average$mean.price.per.room <- round(price_per_room_average$mean.price.per.room,
    2)


# Join price data to shapefile data
nyc@data <- merge(price_per_room_average, nyc@data,
    by = "zipcode", all.y = TRUE)

nyc.df <- merge(nyc.points, nyc@data, by = "id")


gg <- ggplot()
gg <- gg + geom_map(data = nyc.df, map = nyc.df, aes(x = long,
    y = lat, map_id = id, fill = mean.price.per.room),
    color = "black", size = 0.25)
gg <- gg + coord_equal()
gg <- gg + theme_map()
gg <- gg + theme(legend.position = "bottom")
gg <- gg + scale_fill_gradientn(colours = terrain.colors(10))

gg
```

```
## Attempt 3: Map median price of listings per zip
## code

price_median <- aggregate(air.bnb.dataset$price, by = list(air.bnb.dataset$zipcode),
    FUN = "median", na.rm = TRUE)
names(price_median) <- c("zipcode", "median.price")
price_median$median.price <- round(price_median$median.price,
    2)

# Join price data to shapefile data
nyc@data <- merge(price_median, nyc@data, by = "zipcode",
    all.y = TRUE)

nyc.df <- merge(nyc.points, nyc@data, by = "id")



gg <- ggplot()
gg <- gg + geom_map(data = nyc.df, map = nyc.df, aes(x = long,
    y = lat, map_id = id, fill = median.price), color = "black",
    size = 0.25)
gg <- gg + coord_equal()
gg <- gg + theme_map()
gg <- gg + theme(legend.position = "bottom")
gg <- gg + scale_fill_gradientn(colours = terrain.colors(10))

gg
```

K-means for Neighborhood Clusters

```
inclusions <- c("neighbourhood_cleansed", "property_type",
    "neighbourhood_group_cleansed", "latitude", "longitude",
    "distance_in_meters")

cluster.dataset <- air.bnb.dataset[, (names(air.bnb.dataset) %in%
    inclusions)]
tmp <- cbind(cluster.dataset$neighbourhood_cleansed,
    as.data.frame(as.matrix(model.matrix(neighbourhood_cleansed ~
        ., cluster.dataset)))[, -1])
tmp <- as.data.table(melt(tmp))

# Rollup
tmp <- tmp[, mean(value), by = c("cluster.dataset$neighbourhood_cleansed",
    "variable")]
names(tmp)[1] <- "nhood"

# Mean Center
tmp <- tmp[, `:=`(v2, (V1)/mean(V1)), by = c("variable")]

# Normalize
tmp <- tmp[, `:=`(v3, (V1 - mean(V1))/sd(V1)), by = c("variable")]

# Put on zero-one scale so we don't overweight rare
```

```
# categories
tmp <- tmp[, `:=`(v4, (V1)/max(V1)), by = c("variable")]

# Try Mean-Centered and not mean centered
cl.data1 <- as.data.frame(dcast(tmp, nhood ~ variable,
    value.var = "V1"))
cl.data2 <- as.data.frame(dcast(tmp, nhood ~ variable,
    value.var = "v2"))
cl.data3 <- as.data.frame(dcast(tmp, nhood ~ variable,
    value.var = "v3"))
cl.data4 <- as.data.frame(dcast(tmp, nhood ~ variable,
    value.var = "v4"))
row.names(cl.data1) <- cl.data1$nhood
row.names(cl.data2) <- cl.data2$nhood
row.names(cl.data3) <- cl.data3$nhood
row.names(cl.data4) <- cl.data4$nhood
cl.data1 <- cl.data1[, -1]
cl.data4 <- cl.data4[, -1]
cl.data3 <- cl.data3[, -1]
cl.data2 <- cl.data2[, -1]
cl.data5 <- cl.data1[, -1]

# Normalize distance
cl.data5$distance_in_meters <- (cl.data5$distance_in_meters -
    mean(cl.data5$distance_in_meters))/sd(cl.data5$distance_in_meters)

NumClust <- c()
Clustin <- c()
Clustbet <- c()
MC <- c()

i <- 1

for (m in 1:5) {
    if (m == 2) {
        cl.data <- cl.data2
    } else if (m == 3) {
        cl.data <- cl.data3
    } else if (m == 4) {
        cl.data <- cl.data4
    } else if (m == 5) {
        cl.data <- cl.data5
    } else {
        cl.data <- cl.data1
    }

    # test 5-25 Clusters
    for (k in 5:30) {
        # Kmeans can converge to local maximums so run each
        # cluster a few times
        for (j in 1:10) {
            tmp.cl <- kmeans(cl.data, k)
            NumClust[i] <- k
```

```
            Clustin[i] <- tmp.cl$tot.wi
            Clustbet[i] <- tmp.cl$betw
            MC[i] <- m
            i <- i + 1
        }
    }
}

chooseK <- as.data.frame(cbind(NumClust, Clustin, Clustbet,
    MC))
qplot(NumClust, Clustin/100, data = chooseK[MC == 5,
    ]) + geom_smooth(method = "loess") + xlab("Number of Clusters") +
    ylab("Neightborhood Distance from Cluster Center") +
    theme(axis.text.y = element_blank())

use.clusters <- kmeans(cl.data5, 10)
table(test, use.clusters$cluster[test])
air.bnb.dataset$neighborhood_cluster <- as.factor(paste("cluster.",
    use.clusters$cluster[air.bnb.dataset$neighbourhood_cleansed],
    sep = ""))
```

Models

Mann Whitney Wilcoxon

```
# Mann Whitney Wilcoxon Superhost & price
air.bnb.dataset.Superhost <- air.bnb.dataset[air.bnb.dataset$host_is_superhost ==
    "t", c("price")]
air.bnb.dataset.NotSuperhost <- air.bnb.dataset[air.bnb.dataset$host_is_superhost ==
    "f", c("price")]
wilcox.test(air.bnb.dataset.Superhost, air.bnb.dataset.NotSuperhost)
```

```
# Mann Whitney Wilcoxon Superhost & distance to
# subway
air.bnb.dataset.Superhost <- air.bnb.dataset[air.bnb.dataset$host_is_superhost ==
    "t", c("distance_in_meters")]
air.bnb.dataset.NotSuperhost <- air.bnb.dataset[air.bnb.dataset$host_is_superhost ==
    "f", c("distance_in_meters")]
wilcox.test(air.bnb.dataset.Superhost, air.bnb.dataset.NotSuperhost)
```

```
fra.selected.tree <- air.bnb.dataset[, c("host_response_rate",
    "host_acceptance_rate", "host_is_superhost", "host_identity_verified",
    "accommodates", "bathrooms", "bedrooms", "beds",
    "number_of_reviews", "review_scores_rating", "review_scores_cleanliness",
    "review_scores_accuracy", "review_scores_value",
    "instant_bookable", "reviews_per_month", "distance_in_meters")]

# av365 <- listings[which(listings$id %in%
# air.bnb.dataset$id),'availability_365']

# fra.selected.tree <-
# cbind(fra.selected.tree,av365)

# Remove points that do not have superhost status
```

```r
# labeled
to.remove <- which(fra.selected.tree$host_is_superhost ==
    "")
fra.selected.tree <- fra.selected.tree[-to.remove,
    ]

fra.selected.tree <- droplevels(fra.selected.tree)

train <- sample(1:nrow(listings), 26000)
test <- (-train)
tree.training <- fra.selected.tree[train, ]
tree.testing <- fra.selected.tree[test, ]



## lda used for classification
lda_fit <- lda(formula = tree.training$host_is_superhost ~
    ., data = tree.training, na.action = "na.omit")
fitted.results.lda <- predict(lda_fit, newdata = tree.testing,
    type = "response")
```

```r
## qda used for classification
qda_fit <- qda(formula = tree.training$host_is_superhost ~
    ., data = tree.training, na.action = "na.omit")
fitted.results.qda <- predict(qda_fit, newdata = tree.testing,
    type = "response")
```

```r
## In this part, I included the distance variable in
## the model.

#### Gini Index

tree.is_super_host <- rpart(tree.training$host_is_superhost ~
    ., data = tree.training, method = "class", na.action = na.omit)
is_super_host.fit.ig <- predict(tree.is_super_host,
    tree.testing, type = "class")
ig.proportion.correct <- sum(tree.testing$host_is_superhost ==
    is_super_host.fit.gini)/length(tree.testing$host_is_superhost)
print(ig.proportion.correct)

par(xpd = TRUE)
plot(tree.is_super_host, main = "Tree using Information Gain(distance included)")
text(tree.is_super_host, use.n = TRUE)

#### Information gain


tree.is_super_host <- rpart(tree.training$host_is_superhost ~
    ., data = tree.training, method = "class", na.action = na.omit,
    parms = list(split = "information"))
is_super_host.fit.ig <- predict(tree.is_super_host,
    tree.testing, type = "class")
ig.proportion.correct <- sum(tree.testing$host_is_superhost ==
    is_super_host.fit.gini)/length(tree.testing$host_is_superhost)
```

```
print(ig.proportion.correct)

par(xpd = TRUE)
plot(tree.is_super_host, main = "Tree using Information Gain(distance included)")
text(tree.is_super_host, use.n = TRUE)



## Subway station distance variable is not included
## here

#### This is using the gini index for the tree
#### partition.
tree.training$distance_in_meters <- NULL
tree.testing$distance_in_meters <- NULL
tree.is_super_host <- rpart(tree.training$host_is_superhost ~
    ., data = tree.training, method = "class", na.action = na.omit)
is_super_host.fit.gini <- predict(tree.is_super_host,
    tree.testing, type = "class")
gini.proportion.correct <- sum(tree.testing$host_is_superhost ==
    is_super_host.fit.gini)/length(tree.testing$host_is_superhost)
print(gini.proportion.correct)

par(xpd = TRUE)
plot(tree.is_super_host)
text(tree.is_super_host, use.n = TRUE)


#### This is using the information gain for the tree
#### partition.

tree.is_super_host <- rpart(tree.training$host_is_superhost ~
    ., data = tree.training, method = "class", na.action = na.omit,
    parms = list(split = "information"))
is_super_host.fit.ig <- predict(tree.is_super_host,
    tree.testing, type = "class")
ig.proportion.correct <- sum(tree.testing$host_is_superhost ==
    is_super_host.fit.gini)/length(tree.testing$host_is_superhost)
print(ig.proportion.correct)

par(xpd = TRUE)
plot(tree.is_super_host)
text(tree.is_super_host, use.n = TRUE)
```

Anova

One Way Anova for price by property type

```
# One Way Anova (Completely Randomized Design)
# Price vs. Property Type
air.bnb.dataset.Review <- air.bnb.dataset[complete.cases(air.bnb.dataset$price),
    c("review_scores_rating", "property_type", "room_type",
```

```
        "price")]
one_way.fit <- aov(log(price + 1) ~ property_type,
    data = air.bnb.dataset.Review)
layout(matrix(c(1, 2, 3, 4), 2, 2))
plot(one_way.fit)
summary(one_way.fit)
AIC(one_way.fit)
BIC(one_way.fit)
```

Two Way Anova for Price vs. Property Type & Room Type

```
# Two Way Factorial Design Price vs. Property Type
# & Room Type
two_way.fit <- aov(log(price + 1) ~ property_type *
    room_type, data = air.bnb.dataset.Review)
layout(matrix(c(1, 2, 3, 4), 2, 2))
plot(two_way.fit)
summary(two_way.fit)
AIC(two_way.fit)
BIC(two_way.fit)
```

Lasso Regression

```
### TODO: Match created training set with original
### training set (in features) as much as possible
### Ex. Create field that counts number of amenities

## 1.glmnet() function's inability to handle NA
## values
air.bnb.dataset.complete <- air.bnb.dataset[complete.cases(air.bnb.dataset),
    ]


## 2.Lasso regression appear to be thrown off by
## select unnecessary categorical variables
to.drop <- c("id", "host_id", "host_since", "host_verifications",
    "host_name", "amenities", "Host.Ver.facebook",
    "Host.Ver.google", "Host.Ver.jumio", "Host.Ver.email",
    "Host.Ver.kba", "Host.Ver.linkedin", "Host.Ver.manualoffline",
    "Host.Ver.manualonline", "Host.Ver.none", "Host.Ver.phone",
    "Host.Ver.photographer", "Host.Ver.reviews", "Host.Ver.sentid",
    "Host.Ver.weibo", "price.per.room", "number.of.hosts",
    "num_amenities", "distance_in_meters")

air.bnb.dataset.complete <- air.bnb.dataset.complete[,
    !(names(air.bnb.dataset.complete) %in% to.drop)]
predictors <- model.matrix(price ~ ., data = air.bnb.dataset.complete)[,
    -1]
y <- air.bnb.dataset.complete[rownames(predictors),
    "price"]

# Training set
train <- sample(1:nrow(predictors), 15000)
```

```
x.train <- predictors[train, ]
y.train <- y[train]

# Testing set
test <- (-train)
x.test <- predictors[test, ]
y.test <- y[test]
grid <- 10^seq(10, -2, length = 100)
lasso.mod <- glmnet(x.train, y.train, alpha = 1, lambda = grid)
cv.out <- cv.glmnet(x.train, y.train, alpha = 1)
plot(lasso.mod)
plot(cv.out)
bestlam <- cv.out$lambda.min
bestlam


# 2.5 Ridge Regression
ridge.mod <- glmnet(x.train, y.train, alpha = 0, lambda = grid)
cv.ridge.out <- cv.glmnet(x.train, y.train, alpha = 0)
plot(ridge.mod)
plot(cv.ridge.out)
bestlamridge <- cv.ridge.out$lambda.min
bestlamridge
```

```
# Highlights of predictor variables with
# explanatory power
out <- glmnet(predictors, y, alpha = 1, lambda = bestlam)
lasso.coef <- predict(out, type = "coefficients", s = bestlam)[1:50,
    ]
print(lasso.coef)
coeflist <- lasso.coef[lasso.coef != 0]
print(coeflist)
```

```
# Comparing our Lasso vs. Ridge regression
# performance
lasso.pred <- predict(lasso.mod, s = bestlam, newx = x.test)
mean((lasso.pred - y.test)^2)

ridge.pred <- predict(ridge.mod, s = bestlamridge,
    newx = x.test)
mean((ridge.pred - y.test)^2)
```

```
library(data.table)
library(dplyr)
library(ggplot2)
getwd()
setwd("c:/Users/Xtine Lee/Documents/R")
data.dir <- "."
raw <- fread(sprintf("%s/W4702/airbnb/abdatav1_10k.csv",
    data.dir), header = TRUE)
View(raw)
class(raw$host_since)
class(raw$reviews_per_month)
# Converting NAs into Os in raw data
```

```
re_levels <- function(col) {
    if (is.factor(col))
        levels(col) <- c(levels(col), "0")
    col
}
df <- sapply(raw, re_levels)
df[is.na(df)] <- 0
View(df)
df <- as.data.frame(df)
class(df$host_total_listings_count)
df$price <- (as.numeric(df$price))
df$num_host <- (as.numeric(df$num_host))  # Not found
df$host_since <- (as.numeric(df$host_since))
df$host_response_rate <- (as.numeric(df$host_response_rate))
df$host_acceptance_rate <- (as.numeric(df$host_acceptance_rate))
df$host_total_listings_count <- (as.numeric(df$host_total_listings_count))
df$Num_Amenities <- (as.numeric(df$Num_Amenities))  # Not found
df$security_deposit <- (as.numeric(df$security_deposit))
df$cleaning_fee <- (as.numeric(df$cleaning_fee))
df$guests_included <- (as.numeric(df$guests_included))
df$extra_people <- (as.numeric(df$extra_people))
df$minimum_nights <- (as.numeric(df$minimum_nights))
df$maximum_nights <- (as.numeric(df$maximum_nights))
df$availability_365 <- (as.numeric(df$availability_365))  # Not found
df$number_of_reviews <- (as.numeric(df$number_of_reviews))
df$review_scores_rating <- (as.numeric(df$review_scores_rating))
df$review_scores_accuracy <- (as.numeric(df$review_scores_accuracy))
df$review_scores_cleanliness <- (as.numeric(df$review_scores_cleanliness))
df$review_scores_checkin <- (as.numeric(df$review_scores_checkin))
df$review_scores_communication <- (as.numeric(df$review_scores_communication))
df$review_scores_location <- (as.numeric(df$review_scores_location))
df$review_scores_value <- (as.numeric(df$review_scores_value))
df$calculated_host_listings_count <- (as.numeric(df$calculated_host_listings_count))
df$reviews_per_month <- (as.numeric(df$reviews_per_month))
x = model.matrix(df$price ~ ., df)[, -1]  #fix x matrix
View(x)
summary(x)
dim(x)
length(y)
y <- df$price
train <- sample(1:nrow(x), 8000)
test <- (-train)
x.train <- x[train, ]
x.test <- x[test, ]
y.train <- y[train]
y.test <- y[test]
class(x.train)
class(x.test)
class(y.test)
# Performing the Lasso
library(glmnet)
grid = 10^seq(10, -2, length = 100)
lasso.mod = glmnet(x.train, y.train, alpha = 1, lambda = grid)  #x not work
```

```
plot(lasso.mod)
dim(coef(lasso.mod))
set.seed(1)
cv.out = cv.glmnet(x.train, y.train, alpha = 1)
plot(cv.out)
bestlam = cv.out$lambda.min
bestlam
log(bestlam)
lasso.pred = predict(lasso.mod, s = bestlam, newx = x.test)  #x not work
mean((lasso.pred - y.test)^2)  #x not work
out = glmnet(x, y, alpha = 1, lambda = bestlam)
lasso.coef = predict(out, type = "coefficients", s = bestlam)[1:50,
    ]
coeflist <- lasso.coef[lasso.coef != 0]
```

Log Price GAM Model

```
exclusions <- c("id", "host_id", "host_since", "host_name",
    "latitude", "longitude", "cleaning_fee", "security_deposit",
    "extra_people", "neighbourhood_cleansed", "is_location_exact",
    "host_verifications", "amenities")
gam.dataset <- air.bnb.dataset[, !(names(air.bnb.dataset) %in%
    exclusions)]

gam.dataset <- gam.dataset[complete.cases(gam.dataset),
    ]

train <- sample(1:nrow(gam.dataset), 10000, replace = FALSE)
gam.dataset.sample <- gam.dataset[train, ]
model1 <- lm(log(price) ~ ., gam.dataset.sample)

# Variables with very high statistical significance
# (high absolute t-value)
row.names(summary(model1)$coe[which(abs(summary(model1)$coe[,
    3]) > 10), ])

# Use these + ratings and distance which are prime
# candidates for testing for non-linearity
base.formula <- log(price) ~ room_type + accommodates +
    bedrooms + availability_60 + price.per.room + num_amenities +
    distance_in_meters + review_scores_rating + review_scores_accuracy +
    review_scores_cleanliness + review_scores_checkin +
    review_scores_communication + review_scores_location +
    review_scores_value



toPlot <- c("accommodates", "distance_in_meters", "review_scores_rating",
    "review_scores_accuracy", "review_scores_cleanliness",
    "review_scores_checkin", "review_scores_communication",
    "review_scores_location", "review_scores_value")
plotsy <- air.bnb.dataset[, (names(air.bnb.dataset) %in%
    toPlot)]
```

```
plotsy$log.price <- log(air.bnb.dataset$price)
panel.cor <- function(x, y, digits = 2, prefix = "",
    cex.cor, ...) {
    usr <- par("usr")
    on.exit(par(usr))
    par(usr = c(0, 1, 0, 1))
    r <- abs(cor(x, y))
    txt <- format(c(r, 0.123456789), digits = digits)[1]
    txt <- paste0(prefix, txt)
    if (missing(cex.cor))
        cex.cor <- 0.8/strwidth(txt)
    text(0.5, 0.5, txt, cex = cex.cor * r)
}

pairs(plotsy[complete.cases(plotsy), ], upper.panel = panel.cor,
    lower.panel = panel.smooth)
```

```
# 10 fold regression cv
gam.cv <- function(model, traindata2) {
    breaks <- length(traindata2[, 1])
    cv.tmp <- sample(1:breaks, breaks, replace = FALSE)
    br2 <- round(breaks/10)
    predictions <- cv.tmp * 0

    for (i in 1:10) {
        ds.indices <- cv.tmp[(1 + (i - 1) * br2):(i *
            br2)]
        train.indices <- which(!(cv.tmp %in% ds.indices))
        ds <- traindata2[ds.indices, ]
        train <- traindata2[train.indices, ]
        modi <- gam(model, data = train)
        predictions[ds.indices] <- predict(modi, newdata = ds)
    }
    return(predictions)
}
```

```
#First lets try something completely linear
gam.1 <- gam(base.formula, data=gam.dataset)
summary(gam.1)

par(mfrow=c(1,1))
plot.gam(gam.1)

#Because there's a random component, lets average 10 iterations
testErr <- c()
for(i in 1:10){
  a.tm <- gam.cv(base.formula, gam.dataset)
  testErr[i] <- sqrt(mean((a.tm-log(gam.dataset$price))^2, na.rm=TRUE))
}

RMSETest <- mean(testErr)
RMSETrain <- sqrt(mean(gam.1$residuals^2, na.rm=TRUE))

#store result (to have a record)
```

```
results <- as.data.frame(cbind(c("All Linear"), round(RMSETrain, 4), round(RMSETest, 4)))
names(results) <- c("formula", "MSE.Training","MSE.Testing")

results

#Work with this model and append to results

g.formula <- log(price)~room_type + neighbourhood_group_cleansed +
  bedrooms + bathrooms + s(minimum_nights, 2) +
  review_scores_rating + review_scores_cleanliness +
  review_scores_communication + review_scores_location +
  review_scores_value + s(accommodates,5) + distance_in_meters

gam.2 <- gam(g.formula, data=gam.dataset)

testErr <- c()
for(i in 1:10){
  a.tm <- gam.cv(g.formula, gam.dataset)
  testErr[i] <- sqrt(mean((a.tm-log(gam.dataset$price))^2, na.rm=TRUE))
}

RMSETest <- mean(testErr)
RMSETrain <- sqrt(mean(gam.2$residuals^2, na.rm=TRUE))

#store result (to have a record)
results2 <-as.data.frame(cbind(c("Base - Check in Review - accuracy review"),
                               round(RMSETrain,4), round(RMSETest,4)))
names(results2) <- c("formula", "MSE.Training","MSE.Testing")
results <- as.data.frame(rbind(results, results2))


k <- summary(gam.2)$par
k
summary(gam.2)$r.squared
AIC(gam.2)
names(summary.gam(gam.2))

write.csv(results,"gam_results.csv")
write.csv(summary(gam.2)$par,"gam_parametric_anova.csv")
write.csv(summary(gam.2)$anova,"gam_nonparametric_anova.csv")

#residuals base price
g.predicted <- (gam.2$fitted.values)
g.actual <- (gam.2$y)
g.residual <- g.actual-g.predicted
plot(g.predicted,g.residual, xlab="Predicted log price", ylab="Residual")
abline(0,0)

eg.predicted <- exp(gam.2$fitted.values)
eg.actual <- exp(gam.2$y)
eg.residual <- eg.actual-eg.predicted
plot(eg.predicted,eg.residual)
abline(0,0)
```

```
plot(eg.predicted,eg.residual, xlim=c(0,2000), ylim=c(-500, 2000))
abline(0,0)
mean(eg.residual^2)
mean(g.residual^2)
par(mfrow=c(1,2))

plot(g.predicted,g.residual, xlab="Predicted Log Price",
     ylab="Residual", main="Log Price", sub="MSE = 0.1")
abline(0,0)
plot(eg.predicted,eg.residual,  xlab="Predicted Price",
     ylab="Residual", xlim=c(0,2000), ylim=c(-500, 2000),
     main="Price", sub="MSE = 13,663")
abline(0,0)
coef(gam.2)
```

```
gam.2 <- gam(price ~  review_scores_rating+  num_amenities+  number_of_reviews+
               review_scores_accuracy+  review_scores_cleanliness+
               review_scores_location+  distance_in_meters,  data=gam.dataset)

gam.3 <- gam(price~   number_of_reviews+  review_scores_accuracy+
               review_scores_cleanliness+  review_scores_location+
               ns(review_scores_rating,4)+ #set to 4 deg of freedom
               ns(num_amenities,4)+  ns(distance_in_meters,4),  data=gam.dataset)


gam.4 <- gam(price~  number_of_reviews+  review_scores_accuracy+
               review_scores_cleanliness+  review_scores_location+
               minimum_nights+  ns(review_scores_rating,4)+ #set to 4 deg of freedom
               ns(num_amenities,4)+  ns(distance_in_meters,4)+
               host_is_superhost+ host_identity_verified+ property_type+
               room_type+ bed_type, data=gam.dataset)


gam.6 <- gam(price~ number_of_reviews+ review_scores_accuracy+
               review_scores_cleanliness+  review_scores_location+
               minimum_nights+  ns(review_scores_rating,4)+  ns(num_amenities,4)+
               ns(distance_in_meters,4)+  host_response_time+ # added in gam.6 due to Lasso results
              host_has_profile_pic+ # added in gam.6 due to Lasso results
              host_is_superhost+  host_identity_verified+  property_type+
               room_type+ bed_type,  data=gam.dataset)

anova(gam.2, gam.3, test="F")

anova(gam.2, gam.3, gam.4, gam.6, test="F")
```

Tables

Neighborhood Clusters

Number of listings by neighbohood and cluster

	1	10	2	3	4	5	6	7	8	9
Allerton	0	13	0	0	0	0	0	0	0	0
Arden Heights	0	0	0	0	0	0	0	0	3	0
Arrochar	0	0	7	0	0	0	0	0	0	0
Arverne	0	0	0	0	0	0	0	28	0	0
Astoria	0	0	0	0	0	0	0	518	0	0
Bath Beach	0	0	0	0	4	0	0	0	0	0
Battery Park City	0	0	0	0	0	114	0	0	0	0
Bay Ridge	0	0	0	0	0	0	62	0	0	0
Bay Terrace	5	0	0	0	0	0	0	0	0	0
Bay Terrace, Staten Island	0	0	0	0	0	0	0	0	2	0
Bayside	11	0	0	0	0	0	0	0	0	0
Bayswater	0	0	0	0	0	0	0	6	0	0
Bedford-Stuyvesant	0	0	0	0	0	0	1734	0	0	0
Belle Harbor	0	0	0	0	0	0	0	2	0	0
Bellerose	4	0	0	0	0	0	0	0	0	0
Belmont	0	0	0	0	0	0	0	0	0	3
Bensonhurst	0	0	0	0	34	0	0	0	0	0
Bergen Beach	0	0	0	0	3	0	0	0	0	0
Boerum Hill	0	0	0	0	0	0	155	0	0	0
Borough Park	0	0	0	0	87	0	0	0	0	0
Briarwood	0	0	0	0	0	0	0	14	0	0
Brighton Beach	0	0	0	0	0	0	49	0	0	0
Bronxdale	0	0	0	0	0	0	5	0	0	0
Brooklyn Heights	0	0	0	0	0	0	129	0	0	0
Brownsville	0	0	0	0	0	0	26	0	0	0
Bull's Head	0	0	0	0	0	0	0	0	2	0
Bushwick	0	0	0	0	0	0	1250	0	0	0
Cambria Heights	3	0	0	0	0	0	0	0	0	0
Canarsie	0	40	0	0	0	0	0	0	0	0
Carroll Gardens	0	0	0	0	0	0	185	0	0	0
Castle Hill	0	2	0	0	0	0	0	0	0	0
Castleton Corners	0	0	0	1	0	0	0	0	0	0
Chelsea	0	0	0	0	0	955	0	0	0	0
Chinatown	0	0	0	0	0	314	0	0	0	0
City Island	0	8	0	0	0	0	0	0	0	0
Civic Center	0	0	0	0	0	38	0	0	0	0
Claremont Village	0	0	0	0	0	0	9	0	0	0
Clason Point	0	9	0	0	0	0	0	0	0	0
Clifton	0	0	6	0	0	0	0	0	0	0
Clinton Hill	0	0	0	0	0	0	462	0	0	0
Co-op City	0	0	0	0	4	0	0	0	0	0
Cobble Hill	0	0	0	0	0	0	64	0	0	0
College Point	6	0	0	0	0	0	0	0	0	0
Columbia St	0	0	0	0	0	0	0	0	0	30
Concord	0	0	8	0	0	0	0	0	0	0
Concourse	0	0	0	0	0	0	19	0	0	0
Concourse Village	0	0	0	0	0	0	0	0	0	16

	1	10	2	3	4	5	6	7	8	9
Coney Island	0	0	0	0	7	0	0	0	0	0
Corona	0	0	0	0	0	0	0	22	0	0
Country Club	0	0	0	0	0	0	0	0	0	1
Crown Heights	0	0	0	0	0	0	948	0	0	0
Cypress Hills	0	32	0	0	0	0	0	0	0	0
Ditmars Steinway	0	0	0	0	0	0	0	177	0	0
Dongan Hills	0	0	0	2	0	0	0	0	0	0
Downtown Brooklyn	0	0	0	0	0	0	57	0	0	0
DUMBO	0	0	0	0	0	0	29	0	0	0
Dyker Heights	0	0	0	0	0	0	17	0	0	0
East Elmhurst	0	0	0	0	0	0	0	72	0	0
East Flatbush	0	0	0	0	154	0	0	0	0	0
East Harlem	0	0	0	0	0	752	0	0	0	0
East Morrisania	0	0	0	0	2	0	0	0	0	0
East New York	0	0	0	0	46	0	0	0	0	0
East Village	0	0	0	0	0	1741	0	0	0	0
Eastchester	0	4	0	0	0	0	0	0	0	0
Elmhurst	0	0	0	0	0	0	0	74	0	0
Eltingville	0	0	0	0	0	0	0	0	2	0
Emerson Hill	0	0	0	4	0	0	0	0	0	0
Far Rockaway	0	0	0	0	0	0	0	13	0	0
Fieldston	0	0	0	0	4	0	0	0	0	0
Financial District	0	0	0	0	0	337	0	0	0	0
Flatbush	0	0	0	0	0	0	369	0	0	0
Flatiron District	0	0	0	0	0	93	0	0	0	0
Flatlands	0	0	0	0	31	0	0	0	0	0
Flushing	0	0	0	0	0	0	0	97	0	0
Fordham	0	0	0	0	0	0	0	0	0	11
Forest Hills	0	0	0	0	0	0	0	74	0	0
Fort Greene	0	0	0	0	0	0	361	0	0	0
Fort Hamilton	0	0	0	0	0	0	23	0	0	0
Fresh Meadows	6	0	0	0	0	0	0	0	0	0
Gerritsen Beach	0	2	0	0	0	0	0	0	0	0
Glendale	0	0	0	0	0	0	0	8	0	0
Gowanus	0	0	0	0	0	0	126	0	0	0
Gramercy	0	0	0	0	0	266	0	0	0	0
Graniteville	0	0	0	0	0	0	0	0	3	0
Grant City	0	0	0	1	0	0	0	0	0	0
Gravesend	0	0	0	0	0	0	18	0	0	0
Great Kills	0	0	0	0	0	0	0	0	3	0
Greenpoint	0	0	0	0	0	0	688	0	0	0
Greenwich Village	0	0	0	0	0	396	0	0	0	0
Grymes Hill	0	0	0	1	0	0	0	0	0	0
Harlem	0	0	0	0	0	1667	0	0	0	0
Hell's Kitchen	0	0	0	0	0	1210	0	0	0	0
Highbridge	0	6	0	0	0	0	0	0	0	0
Hollis	0	0	0	0	0	0	0	3	0	0
Hollis Hills	1	0	0	0	0	0	0	0	0	0
Holliswood	0	0	0	0	0	0	0	3	0	0
Howard Beach	0	0	0	0	0	0	0	2	0	0
Hunts Point	0	4	0	0	0	0	0	0	0	0
Inwood	0	0	0	0	0	149	0	0	0	0

	1	10	2	3	4	5	6	7	8	9
Jackson Heights	0	0	0	0	0	0	0	101	0	0
Jamaica	37	0	0	0	0	0	0	0	0	0
Jamaica Estates	0	0	0	0	0	0	0	9	0	0
Jamaica Hills	0	0	0	0	0	0	0	1	0	0
Kensington	0	0	0	0	0	0	95	0	0	0
Kew Gardens	0	0	0	0	0	0	0	18	0	0
Kew Gardens Hills	0	0	0	0	0	0	0	7	0	0
Kingsbridge	0	0	0	0	0	0	0	0	0	23
Kips Bay	0	0	0	0	0	402	0	0	0	0
Laurelton	1	0	0	0	0	0	0	0	0	0
Lighthouse Hill	0	0	0	0	0	0	0	0	1	0
Little Italy	0	0	0	0	0	76	0	0	0	0
Long Island City	0	0	0	0	0	0	0	342	0	0
Longwood	0	0	0	0	17	0	0	0	0	0
Lower East Side	0	0	0	0	0	894	0	0	0	0
Manhattan Beach	0	0	0	0	0	0	7	0	0	0
Marble Hill	0	0	0	0	0	7	0	0	0	0
Mariners Harbor	0	0	0	0	0	0	0	0	6	0
Maspeth	0	0	0	0	0	0	0	34	0	0
Melrose	0	0	0	0	0	0	0	0	0	1
Middle Village	0	0	0	0	0	0	0	14	0	0
Midland Beach	0	0	0	9	0	0	0	0	0	0
Midtown	0	0	0	0	0	772	0	0	0	0
Midwood	0	0	0	0	0	0	52	0	0	0
Mill Basin	0	1	0	0	0	0	0	0	0	0
Morningside Heights	0	0	0	0	0	233	0	0	0	0
Morris Heights	0	0	0	0	0	0	0	0	0	6
Morris Park	0	6	0	0	0	0	0	0	0	0
Morrisania	0	0	0	0	0	0	0	0	0	3
Mott Haven	0	0	0	0	26	0	0	0	0	0
Mount Hope	0	0	0	0	22	0	0	0	0	0
Murray Hill	0	0	0	0	0	217	0	0	0	0
Navy Yard	0	0	0	0	10	0	0	0	0	0
Neponsit	1	0	0	0	0	0	0	0	0	0
New Brighton	0	0	0	7	0	0	0	0	0	0
New Dorp Beach	0	0	0	3	0	0	0	0	0	0
New Springville	0	0	0	0	0	0	0	0	2	0
NoHo	0	0	0	0	0	64	0	0	0	0
Nolita	0	0	0	0	0	225	0	0	0	0
North Riverdale	0	3	0	0	0	0	0	0	0	0
Norwood	0	0	0	0	0	0	0	0	0	15
Oakwood	0	0	0	2	0	0	0	0	0	0
Ozone Park	0	0	0	0	0	0	0	16	0	0
Park Slope	0	0	0	0	0	0	396	0	0	0
Parkchester	0	0	0	0	0	0	0	0	0	14
Pelham Bay	0	0	0	0	4	0	0	0	0	0
Pelham Gardens	0	0	0	0	5	0	0	0	0	0
Port Morris	0	0	0	0	14	0	0	0	0	0
Port Richmond	0	0	0	2	0	0	0	0	0	0
Prospect-Lefferts Gardens	0	0	0	0	0	0	284	0	0	0
Prospect Heights	0	0	0	0	0	0	248	0	0	0
Queens Village	9	0	0	0	0	0	0	0	0	0

	1	10	2	3	4	5	6	7	8	9
Randall Manor	0	0	0	6	0	0	0	0	0	0
Red Hook	0	0	0	0	0	0	57	0	0	0
Rego Park	0	0	0	0	0	0	0	35	0	0
Richmond Hill	0	0	0	0	0	0	0	23	0	0
Ridgewood	0	0	0	0	0	0	0	175	0	0
Riverdale	0	0	0	0	0	0	0	0	0	8
Rockaway Beach	0	0	0	0	0	0	0	26	0	0
Roosevelt Island	0	0	0	0	0	47	0	0	0	0
Rosebank	0	0	1	0	0	0	0	0	0	0
Rosedale	1	0	0	0	0	0	0	0	0	0
Sea Gate	0	4	0	0	0	0	0	0	0	0
Sheepshead Bay	0	0	0	0	0	0	47	0	0	0
Shore Acres	0	0	6	0	0	0	0	0	0	0
SoHo	0	0	0	0	0	340	0	0	0	0
Soundview	0	0	0	0	0	0	0	0	0	3
South Ozone Park	5	0	0	0	0	0	0	0	0	0
South Slope	0	0	0	0	0	0	196	0	0	0
Springfield Gardens	17	0	0	0	0	0	0	0	0	0
Spuyten Duyvil	0	0	0	0	5	0	0	0	0	0
St. Albans	16	0	0	0	0	0	0	0	0	0
St. George	0	0	33	0	0	0	0	0	0	0
Stapleton	0	0	5	0	0	0	0	0	0	0
Stuyvesant Town	0	0	0	0	0	95	0	0	0	0
Sunnyside	0	0	0	0	0	0	0	142	0	0
Sunset Park	0	0	0	0	0	0	173	0	0	0
Theater District	0	0	0	0	0	111	0	0	0	0
Throgs Neck	0	2	0	0	0	0	0	0	0	0
Todt Hill	0	0	0	1	0	0	0	0	0	0
Tompkinsville	0	0	11	0	0	0	0	0	0	0
Tottenville	0	0	0	0	0	0	0	0	1	0
Tremont	0	0	0	0	4	0	0	0	0	0
Tribeca	0	0	0	0	0	146	0	0	0	0
Two Bridges	0	0	0	0	0	42	0	0	0	0
Unionport	0	0	0	0	1	0	0	0	0	0
University Heights	0	4	0	0	0	0	0	0	0	0
Upper East Side	0	0	0	0	0	1382	0	0	0	0
Upper West Side	0	0	0	0	0	1499	0	0	0	0
Van Nest	0	0	0	0	0	0	0	0	0	1
Vinegar Hill	0	0	0	0	0	0	24	0	0	0
Wakefield	0	5	0	0	0	0	0	0	0	0
Washington Heights	0	0	0	0	0	640	0	0	0	0
West Brighton	0	0	2	0	0	0	0	0	0	0
West Farms	0	0	0	0	0	0	0	0	0	1
West Village	0	0	0	0	0	807	0	0	0	0
Westchester Square	0	0	0	0	0	0	14	0	0	0
Westerleigh	0	0	0	2	0	0	0	0	0	0
Whitestone	1	0	0	0	0	0	0	0	0	0
Williamsbridge	0	15	0	0	0	0	0	0	0	0
Williamsburg	0	0	0	0	0	0	2777	0	0	0
Windsor Terrace	0	0	0	0	0	0	83	0	0	0
Woodhaven	0	0	0	0	0	0	0	14	0	0
Woodlawn	0	3	0	0	0	0	0	0	0	0

	1	10	2	3	4	5	6	7	8	9
Woodrow	0	0	0	0	0	0	0	0	2	0
Woodside	0	0	0	0	0	0	0	83	0	0