Predicting Price Range of New York City Airbnb listings

Anushka Nair
Laptop: Lenovo Yoga 720 i5
12/14/2020

Q0: *Introduction and Description of classification task:*

Airbnb is a service that has completely disrupted the travel & hospitality markets and this disruption has clearly been welcomed by many travelers, as Airbnb has sustained massive growth since their inception in 2009. Any Airbnb insights that could be gained from analyzing user data may translate into a more efficient travel market overall. This could mean a better experience for both travelers and hosts, as well as a deeper understanding of hospitality services in today's world.

I obtained a large, New York City (NYC) Airbnb listing dataset from kaggle.com, a crowd-sourced, data-sharing platform. This dataset consisted of 16 variables with 48,895 listings, which I ultimately reduced to 10 variables with 38843 listings. This paper presents the considerations, processes, and conclusions derived from one project goal: to predict the price range of an Airbnb listing based on different features. A description of the dataset is provided below.

***Data Description*:**
Number of observations: 38,843
Number of predictors: 9
Number of predictors after one-hot encoding: 15

Categorical:

- Neighbourhood_group: This predictor represents which New York city borough the listing is from. It has 5 categories, namely Bronx, Brooklyn, Manhattan, Queens, and Staten Island.
- Room_type: The room type of the listing, it can either be Entire Home, Shared Room, or a Private Room.

Numerical:

- Latitude :The latitude coordinates of the listing
- Longitude : The longitude coordinates of the listing
- Minimum_nights: The minimum amount of nights required to book this listing.
- Number_of_reviews: The number of reviews
- Reviews_per_month: The number of reviews per month
- Calculated_host_listings_count: The number of listings per host.
- Availability_365: The number of days in a year when the listing is available for booking
- Price: Price in Dollars

The target variable is Price. The price variable in the dataset was in dollars, so 3 categories are created in order to be able to build a classification model. The classes and the number of observations per class are listed below.

Target
Low($0-$79) : 12,804 cases
Medium($80-$149): 13,319 cases
High (>=$150) : 12,720 cases

Data Cleaning:

I deleted certain features because they did not provide any useful information or were hard to integrate into my model. I deleted the *ID*, *Name*, *Host ID*, and *Host Name* because they do not help in the classification task.
Lastly, I deleted the date of the *last review*, because it was too complex to work with.
I also deleted observations with missing data, thus reducing the number of observations from 48,895 to 38,843.

Balancing Class Sizes:

The target class Price had the following classes: Class low consisting of prices <$79, Class Medium consisting of prices <$150 and Class High consisting of Prices >=$150

I chose these specific ranges based on the following:

1. New York hotels range from $74 to $506 per night, with an average cost of $196. The first price range could be thought of as competitive listings that beat the average hotel cost, since most of the values fall below the lower end of an average New York hotel. The medium price range was chosen to be roughly consistent with the first price range(spanning approximately $70-$80 dollars). The high price ranges from $150- $10,000, which is a huge range, however approximately 10,000 observations out of the 12,720 cases that fall under the category High are in the price range $150-$300.
2. The exact cutoff values were picked based on which values provided the best balance between classes. I tried several different values for each cutoff and ultimately decided on the ones stated above.

Categorical to Numerical:

The two categorical variables, neighborhood_group and room_type, were dealt with by using the process of one-hot encoding. One hot encoding transforms a given categorical variable with *n* values into *n* different features where the values can either 1 or 0, 1 indicating the presence of the corresponding category and 0 the absence.

- The feature neighborhood_group consisted of 5 categories: Bronx, Brooklyn, Manhattan, Queens, and Staten Island. The 5 features were one hot encoded and took on the following ID's :
  neighbourhood_group_Bronx, neighbourhood_group_Brooklyn, neighbourhood_group_Manhattan, neighbourhood_group_Queens and neighbourhood_group_Staten Island.

- The feature Room_type consisted of 3 categories: Entire home/apt, Private room and Shared Room. The 3 features were one hot encoded and took on the following ID's :
  room_type_Entire home/apt, room_type_Private room, room_type_Shared room

Standardizing each feature:

I standardized all the features, so each feature has a center of 0 and standard deviation of 1 using an inbuilt R function. Normalizing the data is an important step, because it reduces any bias introduced due to variables being measured at different scales.

Q1.1: *Perform a PCA analysis on standardized features SDATA :*

Next, a Principal Component Analysis is performed on the Standardized data using the prcomp() function. Principal component analysis represents a dataset in a lower dimension by finding a sequence of linear combinations of the variables that capture most of the variation in the data. In general, they are the eigenvectors of the data's covariance matrix. The first principal component of the data is the direction that maximizes the variance. The i^{th} principal component is the direction that has maximal variance orthogonal to the first i-1 principal components. The $x call is used to extract the eigenvectors. The summary of the prcomp() output is calculated, which consists of the standard deviation, proportion of variance explained and the cumulative proportion.

Q1.2: *Present Numerical Results :*

- Below are the two plots for Principal component versus Cumulative % Variance explained and Principal Component versus eigenvalue.

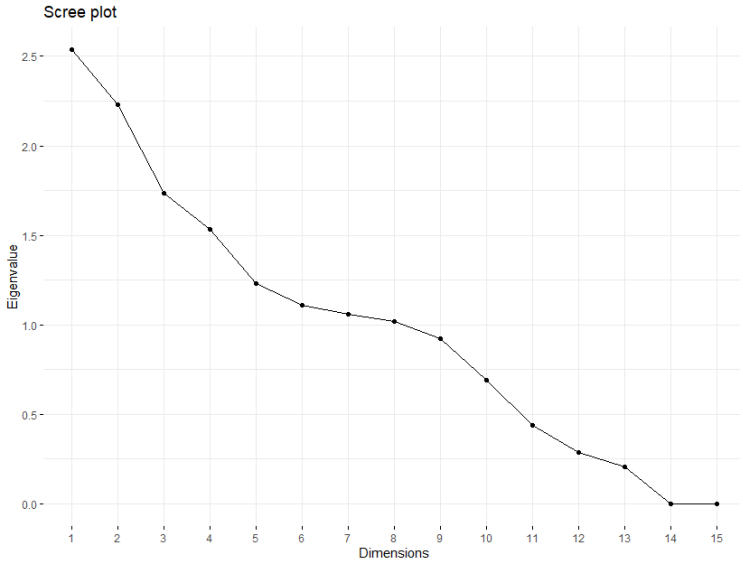

Figure-1: Dimension versus Eigenvalue

The x-axis is the Dimension number and the y-axis is the corresponding eigenvalue

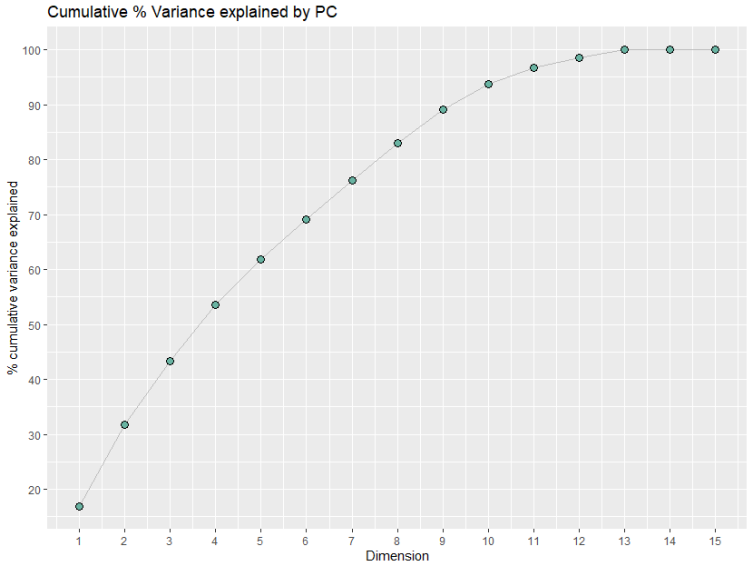

Figure – 2: Dimension versus cumulative % Variance explained

The x-axis is the Dimension number and the y-axis is the cumulative % variance explained.

- From the plot, we can see that the first 11 dimensions explain around 95% of the total variance. According to the plots, the last two dimensions do not contribute to the variation in the data.

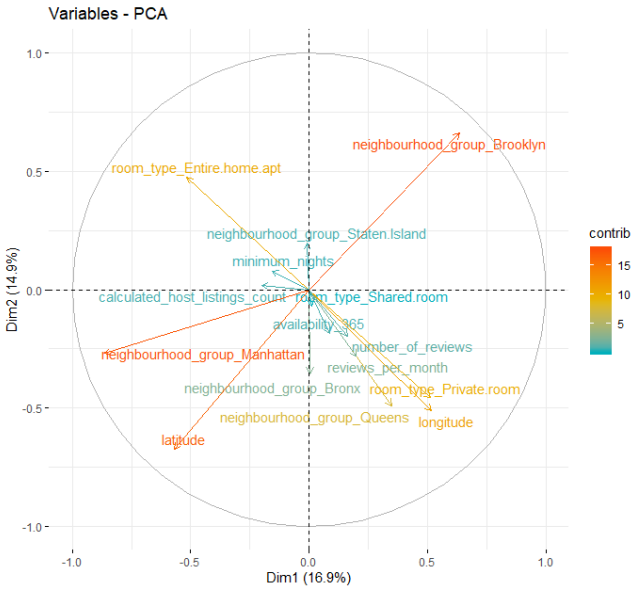

Figure – 3: feature contribution to the first 2 dimensions

The x-axis is Dimension 1 and the y-axis is Dimension 2. The axes range from -1 to 1, indicating correlation with the dimension. The length and color of each feature line represents how much the feature contributes to the respective dimension.

Figure 3 is a plot to visualize the correlation between the features and the first two dimensions. *Neighborhood group Brooklyn*, *Neighborhood group Manhattan*, *latitude* and *room_type_Entire_home* contribute the most to the first two dimensions, while *minimum_nights*, *neighbourhood_group.Staten.Island* and *room_type_Shared.room* contribute the least.

Q1.3: *First 3 Eigenvalues ,%Total Variance and %Cumulative Variance:*

- To obtain the eigenvalues, PVE and Cumulative PVE, the get_eig function is utilized. Below are the associated values of the first 3 dimensions. As we can see, the first three dimensions explain almost half of the variability in the data.

Dimension	Eigenvalues	PVE	Cumulative PVE
1	2.54	17%	17%
2	2.23	15%	32%
3	1.73	12%	44%

Q1.4: *3D Projection for each pair of classes:*

- The first three eigenvectors are extracted along with the associated target variable column. Three subsets of the eigenvectors are created using the subset() function, one for each pair of classes: Low and Medium, Low and High, Medium and High. Below are the three plots of the 3D projection for each pair of classes.

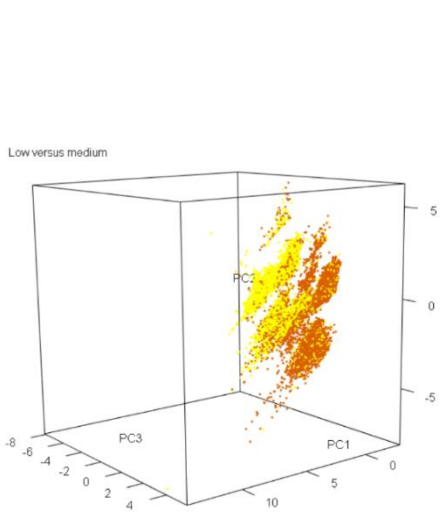

Figure – 4 3D Projection of Low versus Medium

Class Low is Orange and class medium is Yellow

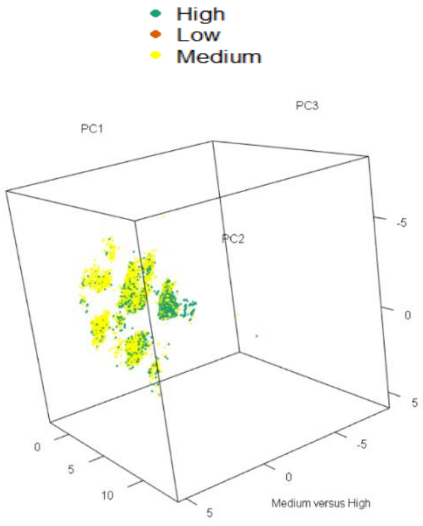

Figure – 5 3D Projection of Medium versus High

Class Medium is Yellow and class High is Green.

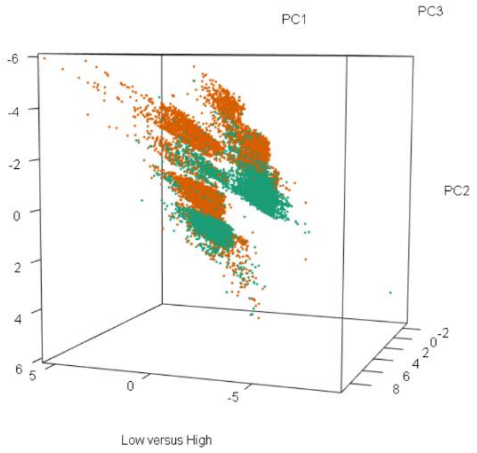

Figure – 6 3D Projection of Low versus High

Class Low is Orange and class High is Green

- The three plots are used to understand how distinct and separable each class is. The Low versus Medium plot looks moderately separable, with some overlap. Visually Medium versus High has a lot of overlap and looks the least separable compared to the other two class pairs. Low versus high looks the most separable. Overall, Low versus High has distinct and separable cases, with not too much overlap, which makes sense intuitively, since the class Medium acts like a buffer.

Q2.0 *K-means algorithm:*

The K-means clustering is an unsupervised learning algorithm used to split data into a predefined(k) number of clusters. The goal is to partition observations into distinct groups such that the observation within each cluster are similar to each other while the observations in different clusters are very different from each other. The K-means algorithm has the following steps:

1) Random Initialization: K random centroids are initialized.
2) Assign observations to the closest centroid, where closeness is defined by the Euclidean distance.
3) Recalculate the cluster centroid by taking the average of all data points belonging to each cluster Clu_j (j=1…k).
4) Repeat step 2 and 3 until the centroid value stabilizes.

A few advantages of K-means clustering is that it is robust, efficient and easy to understand. A few disadvantages is that it is not efficient in handling outliers and noise, and the performance is highly dependent on choosing the best initial cluster assignment in step 1. A work-around for the

second disadvantage is to run the k-means algorithm for multiple k values and pick the value that minimizes the gini index and the within sum of squares, which is done in the next section.

Q2.1 Calculate *Reduction of variance Perf(K) and Gini index G(K) K:*

In order to calculate the reduction of variance Perf(k), we first need the clustering quality Q(k). Q(k) is calculated by dividing the within cluster sum of squares(SWS) by the total within cluster sum squares(SWS(1)), where SWS(1) is the dispersion of the whole dataset around the center. In R, we extract SWS by calling the $tot.withinss function, and SWS(1) by calling the $totss function. Finally, the reduction of variance performance is calculated using the formula Perf(k) = 1-Q(k).

The Gini index is used to evaluate the purity of classification. To find the gini index of each k value, the impurity of each cluster is calculated. The gini index of each cluster is calculated using the formula F1(j)*(1-F1(j)) + ... + Fm(j)*(1-Fm(j)) where Fmj represents the % cases of class(m) within cluster j where j=(1…k). Next, these gini indexes are added up and divided by the maximum gini value attainable for each k, which is calculated using the formula (1-1/m)*j

Q2.2 : *Plot of Reduction of variance Perf(K) and Gini index G(K) versus K:*

- To select the optimal number of clusters, the Reduction of Variance and gini index are calculated for several k values and plotted below. Repeated k-means is performed with several k-values in the range of 0-100 with nstart set to 100 and iter.max set to 50.
- **Computing time: 8.21 Minutes**

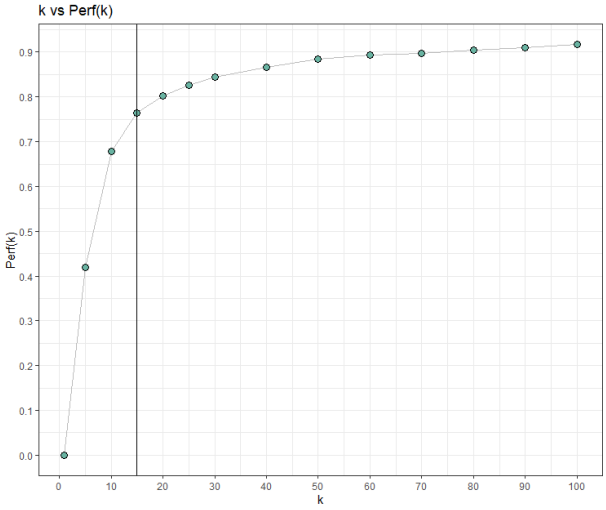

Figure – 7 k vs Perf(k)

The x-axis is the number of clusters k and the y-axis is the reduction of variance performance. The vertical line represents the "ideal" k value.

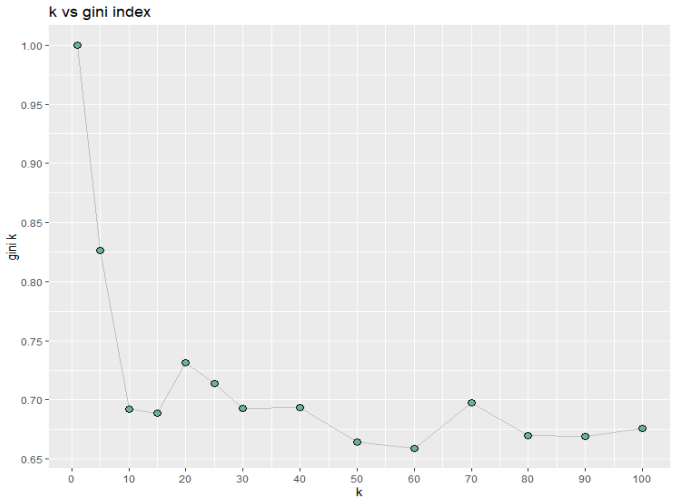

Figure – 8 k vs gini index

The x-axis is the number of clusters k and the y-axis is the corresponding gini-index.

- Looking at the k versus Perf(k) plot, we see that we can see that K = 15 is where the curve begins to flatten and hence, the best K value (k*) using the elbow method is 15. The gini-index plot also confirms that K=15 is the best K value. Although, the gini-index plot reduces even further with higher k values, K=15 is the best value when considering Perf(k).

Q3.1: Describe each cluster Cl_j (j=1,2,...15) by center, size, dispersion, gini and class frequency:

Based on the gini index and Perf(k), K=15 is the best K value. The K-means algorithm is then applied using k=15, and the 15 clusters are described in this section.

Gini index: The closer the gini index is to 0, the higher the purity of classification. If all the elements are linked with a single class, then it would have a gini index of 0. The lowest gini index in our case is Cluster #13, with a gini index of 0.02, which indicates the cluster is almost completely pure. Since this result is alarming, the purest cluster is investigated further. The Airbnb listings within this cluster are all associated to only 2 hosts, due to which the data points were similar. 233 observations are assigned to target price High, and 2 are assigned to target price Medium. All the listings are located in Manhattan, with majority of the room type *Entire_Home*. However, the *number of reviews*, *availability 365*, *reviews per month* and *minimum nights* differ for these listings, from which we can infer that the location and room type are mainly considered for this cluster. The Principal component analysis conducted earlier also suggested that *room_type.Entire home* and *neighbourhood_group_Manhattan* contribute the most to the first two dimensions, suggesting these features are important to attain a good clustering.

Dispersion: The k-means algorithm seeks to minimize the dispersion, also referred to as the within-cluster sum of squares(WSS), which measures the amount by which the observations within each cluster differ from each other. A lower within-cluster sum of squares indicates compactness. The lowest dispersion is associated with cluster #13, the same cluster with the lowest gini index.

The first 3 clusters size, gini index, within sum of squares and class frequency is shown in the table down below. The remaining values are in the excel sheet titled “best k(15) cluster description”.

Cluster #	WSS	Size	Gini	Class High Freq	Class Low Freq	Class Medium Fr
1	17616.73149	5784	0.60273972	13%	40%	47%
2	2074.392146	314	0.567264392	13%	57%	30%
3	5547.297526	4954	0.570000647	46%	8%	46%

Center coordinates: The kxp(15x15) center coordinates are extracted using the $centers call and placed in the excel sheet titled “Best k cluster centers”. In order to visualize the cluster centers , a principal component analysis is performed on the centers, and the first three eigenvectors are extracted. The plot suggests that the 15 clusters are spread out.

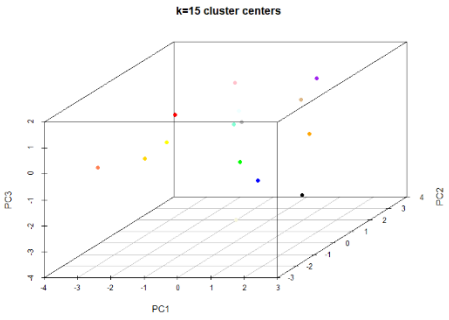

Figure –9: Plot visualizing best K Cluster centers

Q3.2: *Display the 3D projection of cluster CLj on eigenvectors V1 V2 V3*

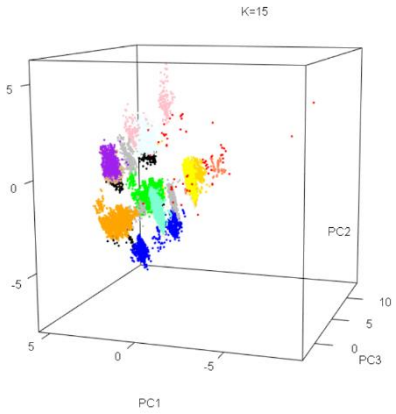

Figure –10: 3D Projection of Clj on first 3 eigenvectors

To display the 3D Projection of cluster CL_j on the eigenvectors V1,V2,V3, the plot3d library is used in R. The Visually, majority of the 15 clusters seem well defined and compact, with some visible noise.

Q3.2: *K-means results:*

While performing a k-means clustering wasn't required, I did so to see how well it performed. The confusion matrix is placed in the excel sheet titled "K means 15". The algorithm is 62% accurate in the classification task. It performs the best in classifying class Low, with an accuracy rate of 71%, followed by class High, with an accuracy rate of 65%, and finally class Medium, with an accuracy of 50%.

Q4: *Trainset and test-set distribution*:

The dataset is divided into three subsets corresponding to the target classes: LOWPRICE, MEDPRICE, HIGHPRICE. Each subset is randomly split into a training and testing set, with each train set consisting of 80% of the corresponding subset and each test set consisting of 20% of the corresponding subset. The three train sets are combined into one and the three test sets are combined into one, naming them TRAIN_SET and TEST_SET respectively. The classes are approximately balanced within the train and test set, so there is no need to clone the data.

Q5: *Random Forest algorithm:*

Random Forest is a supervised learning algorithm that can be used for classification and regression tasks. The algorithm utilizes several decision trees on various subsets of the data to produce a class prediction in two stages.

Stage 1: Create the random forest ensemble, where each time a tree split is considered, a random subset of sqrt(p) predictors from the full set is picked.

Stage 2: Make a prediction from the random forest classifier created in the first stage. Random forest prediction takes the test features and uses the rules of each randomly created decision tree to predict and store the predicted outcome. Each tree gives a class prediction and the class with the most votes becomes the model's prediction.

The advantages of using the random forest algorithm is that it is efficient , highly accurate and is not sensitive to outliers. Since random forest predicts based on input from several trees, individual decision tree errors are minimized, thus reducing overall variance and error and leading to a more generalized and less overfit model. The disadvantages of the random forest algorithm are that it requires several parameters. Further, it requires a lot of memory and takes long to process.

Q5.1: *Apply Random Forest to SDATA:*

The random forest algorithm is applied to the training set with ntree = {100,200,300,400}and ntry = $\sqrt{15} \approx 4$. Below is the train and test set accuracy curves for the 4 trees.

Computing time: 1.9 mins

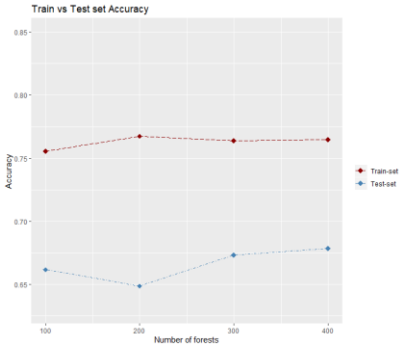

Figure – 11 Train versus test set accuracy over multiple tree values

The x-axis is the number of trees. The y-axis is the accuracy rate. The red line represents the train set accuracy, and the blue line represents the test set accuracy

The train set consistently performs better than the test set with an approximate accuracy of 76% . Predicting power is lost in all three classes when we move from the train set to the test set. The four test set accuracy rates are in the range of 64%-67%. The best ntree value looking at this plot is 400.

Below are the four confusion matrices corresponding to the four tree values:

Total :66% CI: (64%-66%)		TEST TRUE		
	nTree = 100	**LOW**	**MEDIUM**	**HIGH**
PRED	**LOW**	81%	26%	2%
	MEDIUM	18%	43%	23%
	HIGH	4%	31%	75%

Total :65% CI: (64%-66%)		TEST TRUE		
	nTree = 200	**LOW**	**MEDIUM**	**HIGH**
PRED	**LOW**	82%	27%	3%
	MEDIUM	13%	37%	20%
	HIGH	5%	36%	77%

Total :67% CI: (66%-68%)		TEST TRUE		
	nTree = 300	**LOW**	**MEDIUM**	**HIGH**
PRED	**LOW**	82%	25%	4%
	MEDIUM	15%	43%	21%
	HIGH	3%	32%	75%

Total :68% CI: (66%-68%)		TEST TRUE		
	nTree = 100	**LOW**	**MEDIUM**	**HIGH**
PRED	**LOW**	82%	25%	4%
	MEDIUM	13%	47%	22%
	HIGH	5%	28%	76%

The four classifiers associated with the four ntrees are approximately 82% accurate at classifying "Low", 42% accurate at predicting "Medium", and 75% accurate at predicting "High" with an overall approximate accuracy of 66%. They consistently perform well on predicting class "Low", with class "Medium" and class "High" causing the varying accuracy rates. Ntree=400 performs the best and Ntree=200 performs the worst in terms of overall accuracy.
Ntree=400 is better at classifying "Medium" than the other number of trees, however, some predicting power is lost in the classification of "High".
Ntree = 200 performs the worst in classifying "Medium" , with an accuracy of 37%.
Based on the confusion matrix, ntree=400 is the best number of trees. It classifies "Low" 82% accurately, with a false positive rate of 29% and false negative rate of 18%. It classifies "Medium" 47% accurately, with a false positive rate of 35% and false negative rate of 53%. It classifies "High" 76% accurately, with a false positive rate of 33% and a false negative rate of 26%.

Q6:Display 3 curves corresponding to the 3 target classes:

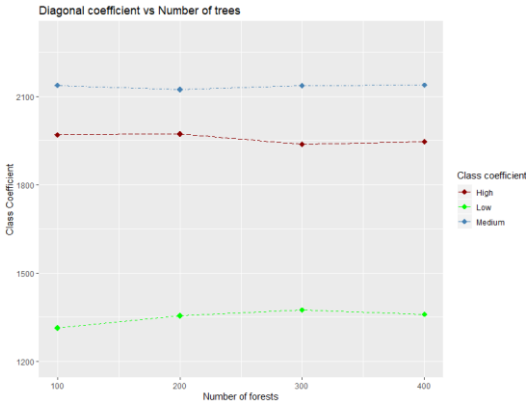

Figure – 12 Number of trees vs diagonal coefficients

The x-axis is the number of trees. The y-axis are the diagonal coefficients for each class. The red line is class High, the green line is class Low, the blue line is class Medium

From the plot above, we see that the number of correctly classified "Low" prediction increases as ntree increases. The number of correctly classified "High" prediction slightly decreases with more ntrees. The number of correctly classified "Medium" prediction increases with a higher number of trees. From this, we can conclude that with a higher number of trees, class High loses some predicting power and Class Medium and High gain predictive power.

Q7: *Compute Importance and display in decreasing order*:

The importance() function provides two measures for determining feature importance, mean decrease impurity and mean decrease accuracy. Mean decrease impurity (Gini importance) is calculated by the total amount by which the Gini decreases over the specific variable, averaged across all trees. If a variable is important, the tree split over that variable causes the Gini index to decrease the most. The larger the value of the mean decrease in Gini, the more important the variable. The second measure is the mean decrease accuracy, which directly measures the impact each feature has on the accuracy of the model. It is computed by permuting the values of each feature in the out-of-bag samples and measuring how much the permutation decreases the accuracy of the model. This decrease is averaged over all trees. Important variables should lead to a larger mean decrease in accuracy. To compute these importance features, the function importance() is used. The random forest algorithm is applied using bntree=400 and ntry = 4 (BESTRF). Below is the table of the first 3 important features based on Mean decrease in Accuracy:

Features	High	Low	Medium	MeanDecreaseAccuracy	MeanDecreaseGini
latitude	14.20	29.27	16.25	40.96	1432.17
availability_365	13.61	23.86	22.65	39.37	694.57
longitude	22.56	27.31	20.05	36.86	1636.20

The remaining values are in the excel sheet titled "Variable Importance". Next, to visualize the feature importance, the varImpPlot() function is used.

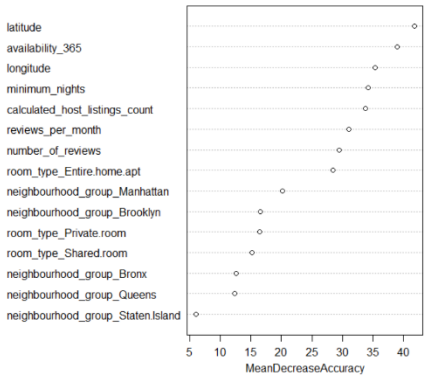

Figure – 13: Mean Decrease in Accuracy

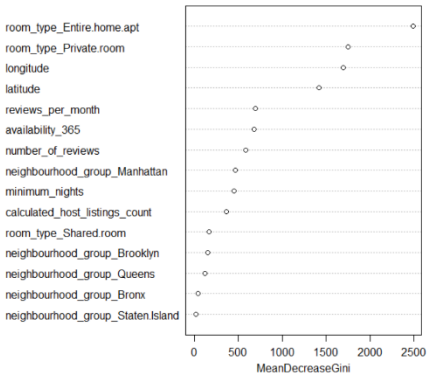

Figure – 14: Mean Decrease in Gini

The gini index plot indicates that room type entire home and private room are the most important in terms of reducing the gini-index, while they are not as important in terms of overall accuracy. The two measures seem to agree that *neighbourhood_group.Staten.Island*,

neighbourhood_group.Bronx and *neighbourhood_group.Queens* are the least important features. According to the mean decrease Accuracy plot, *latitude* is the most important feature and *neighbourhood_group.Staten.Island* is the least important feature.

Q8: KS test for the most(latitude) and least(Staten Island) important feature:
In order to compare the features visually, I divided the data into three subsets corresponding to the three classes: LOWPRICE, MEDPRICE and HIGHPRICE. Next, I plotted the histogram for the most important and least important feature and conducted a ks.test. The ks test compares two datasets and indicates whether they are drawn from the same distribution by estimating and comparing the distance between the distribution of the two samples.

Below are the results:

- **Latitude**:

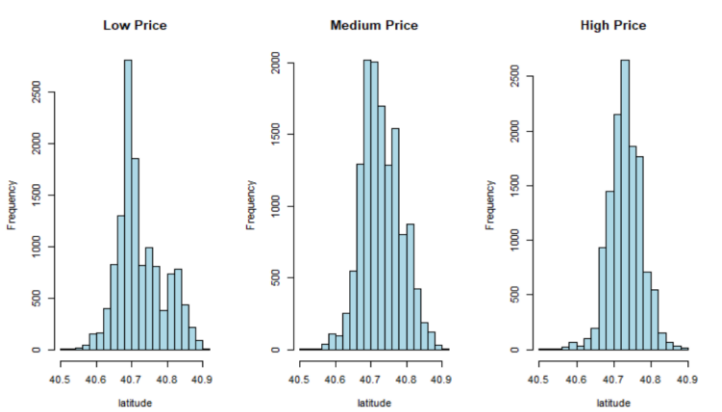

Figure – 15: Histogram of latitude

The x-axis are the latitude values and the y axis is the frequency.

The three histograms follow a generally normal distribution. Visually, Price low seems to be bimodal, with one mode at approximately x=40.7 and one at approximately x=40.85.
Medium price also peaks around x=40.7, but it doesn't have a sharp drop in frequency like price low, instead it has a gradual drop in frequency. Price High has a peak x=40.75 and does not drop in frequency as sharply as the histogram for low price.

KS-test:
H_0: Both samples come from a population with the same distribution
H_a: Both samples do not come from a population with the same distribution

LATITUDE		
Class-Pair	P-value	D
(Low,Medium)	<2.2e-16	0.15
(Medium,High)	<2.2e-16	0.12
(Low,High)	<2.2e-16	0.26

As we can see, for all three class pairs, we reject the null hypothesis since the p-value< 0.01 and conclude that the 3 pairs of classes do not come from a population with the same distribution. The Distance measure makes sense intuitively, as we can see that the distribution for the class pair (Low,High) is the highest. This is in agreement with the interpretation in **section 1.4** (*3D*

Projection for each pair of classes) that the class pair (Low,High) are highly separable and distinct, compared to the other two class-pairs.

- **Neighbourhood_group.Staten.Island**:

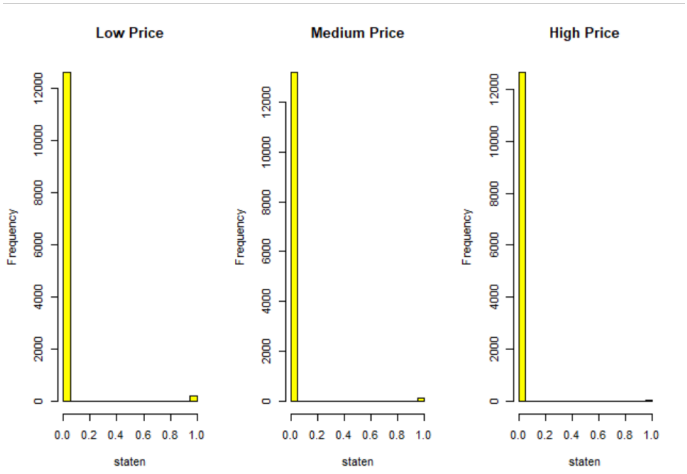

Figure – 16: Histogram of Staten Island

The x-axis represents the value of Staten Island and the y axis is the frequency.

Visually, we can see that the three histograms are almost identical. Staten Island is a one-hot encoded variable, hence taking on the values of 0 and 1 indicating presence or absence. The reason this feature is not that useful could be attributed to the fact that there are barely any Airbnb listings in Staten Island, comprising 314 out of 38843 of the listings.

KS-test:
H_0: Both samples come from a population with the same distribution
H_a: Both samples do not come from a population with the same distribution

Staten_Island		
Class-Pair	**P-value**	**D**
(Low,Medium)	0.9195	0.006
(Medium,High)	0.9999	0.003
(Low,High)	0.442	0.011

For all the three class pairs, we fail to reject the null hypothesis since the p-value>0.01 and conclude that the 3 pairs of classes come from a population with the same distribution. Again, we can see that the greatest distance is between Low and High.

The ks test indicates that the feature latitude is significantly different across the three classes while the feature Staten_Island is not significantly different across the three classes. We can conclude that latitude has high discriminating power and Staten Island has low discriminating power.

Q9: *RF on cluster with lowest Gini*:

The clustering with the minimum gini-index computed in Q3 is associated with K=15.
In order to compare. The random forest algorithm is applied to the 15 clusters to see if there is any advantage to training the RF classifier over the lowest gini-index clustering.
The clusters that were extremely pure and consisted majority of one class(>70%) were replaced by a predictor which always predicts the majority class. Below is a table consisting of the clusters that used the blind majority class predictor.

Cluster #	4	6	7	8	9	11	13	14
Vote	High	Low	Low	High	Low	Low	High	Low

As we can see, none of the majority vote predictors are used to classify "Medium".
The remainder of the clusters were trained on the random forest algorithm. Below is the process applied to Cluster #1. First, the data assigned to cluster 1 is extracted and named Clu1. Next, Clu1 is divided into three subsets corresponding to the target classes: clu1_low,clu1_medium, clu1_high. Next, the three subsets are randomly split into a training and testing set, with the train set consisting of 80% of the corresponding subset and each test set consisting of 20% of the corresponding subset.

CLASS	TOTAL	TRAIN	TEST
LOW	2271	1816	455
MEDIUM	2747	2197	550
HIGH	766	612	154

Since the classes are unbalanced, smote() is applied to balance the classes.
The smote () function takes two parameters: K and dup_size. It uses existing minority instances and synthesizes new instances at some distance towards one of the neighbors. The distance is specified using the K parameter, where k = 1 is the closest neighbor of the same class. The dup_size is the amount of times the smote() function should look through the existing instances. The dup_size for class "High" is set to 2 and the k parameter is set to 4. Below is the train and test set distribution after applying smote()

CLASS	TRAIN	TEST
LOW	1816	455
MEDIUM	2197	550
HIGH	1836	462

After balancing the classes, the random forest algorithm is applied using bntree=400 and ntry=4.
The remaining clusters follow the same procedure.

Cluster#1 Test Set versus SDATA Test Set

Computing time: 1.9 mins

Total :58% Smote(K=4)		TEST TRUE		
	nTree = 400	**LOW**	**MEDIUM**	**HIGH**
PRED	**LOW**	64%	21%	5%
	MEDIUM	35%	74%	58%
	HIGH	1%	5%	37%

Total :68% SDATA		TEST TRUE		
	nTree = 400	**LOW**	**MEDIUM**	**HIGH**
PRED	**LOW**	82%	25%	4%
	MEDIUM	13%	47%	22%
	HIGH	5%	28%	76%

95% Confidence Interval(56%,61%)

95% Confidence Interval(66%,68%)

The confusion matrix on the left is the RF classifier trained and tested on Cluster #1. The confusion matrix on the right is the random forest algorithm trained on SDATA in Q5. Since the majority class in this cluster prior to cloning was "Medium", we see that Medium performs the best. "Medium" in SDATA performs worse, since it was the class with the most noise associated with it. Applying the Random forest algorithm in this situation seems beneficial since we can see that our worst performing class in SDATA("Medium") is performing well when the algorithm is trained on its majority cluster. The remaining clusters confusion matrices are in the excel sheet titled "Q9 Cluster RF"

Q10: *RF on cluster with lowest Gini*:

In this step, all the 15 classifiers corresponding to the 15 clusters from Q9 are combined using a for loop and tested on the SDATA Test Set. For instance, if the test-set case belongs to cluster #1, then apply the classifier associated with cluster#1 as explained in the previous step. Below are the two confusion matrices for RF applied on the minimum gini clustering and RF applied on K-means.

BESTRF Test-set versus Minimum Gini

Total :68% **SDATA**		TEST TRUE		
	nTree = 400	**LOW**	**MEDIUM**	**HIGH**
PRED	**LOW**	82%	25%	4%
	MEDIUM	13%	47%	22%
	HIGH	5%	28%	76%

Total :71% **MIN GINI** **K=15**		TEST TRUE		
	nTree = 400	**LOW**	**MEDIUM**	**HIGH**
PRED	**LOW**	73%	16%	2%
	MEDIUM	23%	67%	26%
	HIGH	4%	17%	72%

95% Confidence Interval(66%,68%)

95% Confidence Interval(70%,72%)

The RF Classifier applied to the minimum-gini index clustering(MIN_GINI_RF) performs better than the RF algorithm applied to SDATA. The MIN_GINI_RF slightly loses predicting power in

Class “Low” and “High”, but significantly improves in classifying “Medium”. This result suggests that training the random forest algorithm on the clusters reduced the noise caused by class Medium. The confidence intervals do not overlap, which suggests that training on the minimum gini clustering improves accuracy. The combination of K-means and RF perform better than K means by itself, which was 62% accurate overall(*Q3.2*: *K-means results)* .The benefit of this method from my understanding is that it can identify more distinct cases within the noisy class, since the clustering tries to group by similarity.

Q11: *Train Linear SVM on two classes*:

Support Vector Machine is an algorithm that finds a hyperplane in an n dimensional space(n=number of features) that differentiates between the two classes. The ideal hyperplane is the maximal margin hyperplane, which maximizes the distance to the closest data points from both classes. A few advantages of the SVM algorithm is that it is extremely effective in high dimensional data, is memory efficient and is not impacted by outliers. A few disadvantages of the SVM algorithm is that it is a time-consuming algorithm and it does not perform well in the case of overlapping classes.
Following the same steps and 80-20 split as *Q4*: *Trainset and test-set distribution,* three new train sets and three new test-sets are created, namely (trainset_LOW, testset_LOW), (trainset_MED, testset_MED) and (trainset_HIGH, testset_HIGH).
Next, three pairs of train and test sets are created for the binary classification using SVM. (TRAINSET_LM, TESTSET _LM), (TRAINSET_MH, TESTSET _MH), (TRAINSET_LH, TESTSET_LH).
The tune() function from the library e1071 is used in order to compare and pick the best cost value. By default, tune() performs ten-fold cross-validation on a set of models of interest and helps determine the best cost parameter . The cost values tested are {0.001,0.01,0.1,1,5,10}. The best cost function is determined to be cost=5 for all three binary classifiers. Below are the confusion matrices for the three classifiers:
Computation time: 12.46 minutes

Total: 91%	TRUE	
SVM (Cost=5)	**LOW**	**HIGH**
LOW	91%	8%
HIGH	9%	92%

Total:77%	TRUE	
SVM (Cost=5)	**LOW**	**MEDIUM**
LOW	72%	18%
MEDIUM	28%	82%

Total: 67%	TRUE	
SVM (Cost=5)	**MEDIUM**	**HIGH**
MEDIUM	51%	17%
HIGH	49%	83%

The SVM performs exceptionally well in predicting class Low versus High, with a global accuracy of 91%. This result makes sense since in Q1.4: *3D Projection for each pair of classes*, Class Low versus Class High seemed highly separable. The SVM also performed well in predicting Class Low versus Medium, with an overall accuracy of 77%. For Class Medium versus High, the SVM had a low overall accuracy, however it performed well(83%) in predicting Class High.

As part of my midterm, I performed a KNN for the same three two-class problems. The confusion matrices using the KNN algorithm are down below:

Total: 92%	TRUE	
KNN K=21	**LOW**	**HIGH**
LOW	91%	8%
HIGH	9%	93%

Total: 80%	TRUE	
KNN K=21	**LOW**	**MEDIUM**
LOW	81%	21%
MEDIUM	19%	79%

Total: 74%	TRUE	
KNN K=21	**MEDIUM**	**HIGH**
MEDIUM	73%	26%
HIGH	27%	74%

The KNN algorithm performed better than the SVM algorithm in all three binary classification problems. The binary classification of Low versus High perform almost identically, indicating that the two classes are distinct and easy to classify. In the binary classification problem of Medium vs High, the SVM algorithm performs terribly in classifying "Medium" compared to the KNN algorithm. In Q1.4: *3D Projection for each pair of classes* I stated that the pair of Medium versus High looks the least separable visually with several overlapping cases, and as stated earlier one of SVMS cons is that it does not perform as well in the case of overlapping classes. This might be the reason of the terrible performance in the binary problem of Medium versus High.

*Conclusion***:**

First, a principal component analysis was performed to the standardized data to visualize the separability of the classes. The K-means algorithm is performed, where K=15 is found to be the "best" number of clusters. Next, the random forest algorithm is applied, where ntree=400 performs the best. Using the importance() feature, latitude is identified as the most important feature and neighbourhood_group.Staten.Island is identified as the least important feature. Next, within each cluster of best K-means, either the random forest algorithm or majority vote classification is applied based on class distribution and used to predict on the test-set of SDATA. This combination of K-means and Random forest performs better in terms of accuracy than the two algorithms on their own. Finally, SVM is applied to the three binary problems, where Low versus High performs the best. The random forest algorithm accuracy could be improved by exploring and testing more values for the number of forests. The SVM algorithm accuracy could be increased by using a non-linear kernel, especially in the case of Low versus Medium and Medium versus High. One way to improve the classification task is by picking a new cut-off value for class "Medium", since it is the class generating the most noise. One possibility that can be explored is to test the cases Low versus (Medium+High), Medium versus (Low+High) and High versus (Low+Medium).

CODE

```
#######CLEAN#####
library(data.table)
library(mltools)
library(standardize)
library("dplyr")
library(factoextra)
#############Q0#################
AB_NYC_2019 <- read.csv("C:/Users/User/OneDrive/Desktop/MSDS/DS/archive/AB_NYC_2019.csv")
NYC_DATA_CLEAN= AB_NYC_2019[,-c(1:4,6,13)]
NYC_DATA_CLEAN$neighbourhood_group<-as.factor(NYC_DATA_CLEAN$neighbourhood_group)
NYC_DATA_CLEAN= na.omit(NYC_DATA_CLEAN)##No NA values
View(NYC_DATA_CLEAN)
#adding the target variable
NYC_TARGET<-NYC_DATA_CLEAN
NYC_TARGET$target = NA

#Creating the target class
for (i in 1:nrow(NYC_TARGET)){
  if(NYC_TARGET$price[i]<80){
    NYC_TARGET$target[i]="Low"
  }else if(NYC_TARGET$price[i]<150){
    NYC_TARGET$target[i]="Medium"
  }else{
    NYC_TARGET$target[i]="High"
  }
}
NYC_TARGET$target=as.factor(NYC_TARGET$target)
summary(NYC_TARGET$target)
#one hot encoding categorical
S_NYC<-one_hot(as.data.table(NYC_TARGET), cols = c("room_type","neighbourhood_group"))
```

```r
S_NYC = data.frame(S_NYC)
##Standardizing
S_NYC1<-S_NYC%>%mutate_if(is.numeric,scale)
S_NYC1<-S_NYC1[,-11]#get rid of original price var
S_NYC_features = S_NYC1[,-16]
S_NYC_label = data.frame(S_NYC1[,16])

#############Q1#############
#PCA
AIRBNB.pca <- prcomp(S_NYC_features,scale=TRUE) # pca of features
PC = data.frame(AIRBNB.pca$x)
fviz_eig(AIRBNB.pca)#plot eigenvalues

fviz_pca_var(AIRBNB.pca,
             col.var = "contrib", # Color by contributions to the PC
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE     # Avoid text overlapping
)
summary(AIRBNB.pca) #11 PCS explain 95% variation
########Q1.1############
percent_var = summary(AIRBNB.pca)
fviz_eig(AIRBNB.pca,geom = "line",choice = "eigenvalue",ncp = 15)
fviz_eig(AIRBNB.pca,geom = "line",choice = "variance",ncp = 15)

Prop_Var = data.frame(percent_var$importance[3,])
Prop_Var = round(100*Prop_Var,2)
library(ggplot2)
library(dplyr)
library(hrbrthemes)
plot(seq(1,15),Prop_Var$percent_var.importance.3...)


ggplot(Prop_Var, aes(x=seq(1,15), y=percent_var.importance.3...)) +
  geom_line(color="grey") +
  geom_point(shape=21, color="black", fill="#69b3a2", size=3)+
  scale_x_continuous(breaks = scales::pretty_breaks(n = 15)) +
  scale_y_continuous(breaks = scales::pretty_breaks(n = 10)) +
  ggtitle("Cumulative % Variance explained by PC")+
  xlab("Dimension")+
  ylab("% cumulative variance explained")
###########Q1.2############
PC = cbind(PC,S_NYC_label)
PC = PC[,c(1,2,3,16)]
#LOW,MEDIUM
PC_LM = subset(PC,PC$S_NYC1...16. == "Low" | PC$S_NYC1...16. == "Medium")
#MEDIUM HIGH
PC_MH = subset(PC,PC$S_NYC1...16. == "High" | PC$S_NYC1...16. == "Medium")
#LOW HIGH
PC_LH = subset(PC,PC$S_NYC1...16. == "Low" | PC$S_NYC1...16. == "High")
##plotting Low versus Medium
#displaying 3D graph of each vector
library(factoextra)
library(plot3D)
library(rgl)
library(scatterplot3d)
colors <- c("#1B9E77", "#D95F02", "yellow") # creating color function
colors <- colors[as.numeric(PC_LM$S_NYC1...16.)] # adding the colors numbers to each CL
#side plot
scat2<-scatterplot3d(PC_LM[,1:3], pch = 16, color=colors, main = "3")
legend(scat2$xyz.convert(-30,1,15),legend = levels(PC_LM$S_NYC1...16.),col =c("#1B9E77",
#3D plot
plot3d(PC_LM[,1:3], col=colors, main="Low versus medium")
```

```r
#side plot
scat2<-scatterplot3d(PC_MH[,1:3], pch = 16, color=colors, main = "3")
legend(scat2$xyz.convert(-25,1,15),legend = levels(PC_LM$S_NYC1...16.),col =c("#1B9E77", "#D95F02", "yellow"),p
#3D plot
plot3d(PC_MH[,1:3], col=colors, main="Medium versus High")
#Low versus high
colors <- c("#1B9E77", "#D95F02","yellow") # creating color function
colors <- colors[as.numeric(PC_LH$S_NYC1...16.)] # adding the colors numbers to each CL
#side plot
scat2<-scatterplot3d(PC_LH[,1:3], pch = 16, color=colors, main = "3")
legend(scat2$xyz.convert(-25,1,15),legend = levels(PC_LH$S_NYC1...16.),col =c("#1B9E77", "#D95F02", "#7570B3"),
#3D plot
plot3d(PC_LH[,1:3], col=colors, main="Low versus High")
###########Q2############
#apply K means clustering
start = Sys.time()
k.set= c(1,5,10,15,20,25,30,40,50,60,70,80,90,100)
Qm <- numeric(length(k.set))
Perf.m <- numeric(length(k.set))
wss = numeric(length(k.set))
for (k in 1:length(k.set)){
  km.airbnb <- kmeans(S_NYC_features, k.set[k], nstart = 100,iter.max=50)
  #wss[k] = km.airbnb$tot.withinss
  Q <- km.airbnb$tot.withinss/km.airbnb$totss
  Perf <- 1-(km.airbnb$tot.withinss/km.airbnb$totss)
  Qm[k] = Q
  Perf.m[k] = Perf
}
end = Sys.time()
end-start
```

```r
ggplot(data.frame(cbind(k.set,Perf.m)),aes(x=k.set, y=Perf.m)) +
  geom_line(color="grey") +
  geom_point(shape=21, color="black", fill="#69b3a2", size=3)+
  scale_x_continuous(breaks = scales::pretty_breaks(n = 15)) +
  scale_y_continuous(breaks = scales::pretty_breaks(n = 10)) +
  geom_vline(xintercept = 15)+
  xlab("k")+
  ylab("Perf(k)")+
  ggtitle("k vs Perf(k)")+
  theme_bw()
##################Q2.1 ##############
k.set= c(1,5,10,15,20,25,30,40,50,60,70,80,90,100)
nstart=30
impk = numeric(length(k.set))
gini_fin = numeric(length(k.set))
gin = numeric(length(k.set))
full_data= S_NYC1
##cluster #8 is purest
for(k in 1:length(k.set)){
  full_data = S_NYC1
  km_explore<- kmeans(full_data[,-16],k.set[k], iter.max = 50)
  full_data$cluster = NA
  full_data$cluster=as.factor(km_explore$cluster)
  clus_Size = km_explore$size #5 clusters when k=5
  gini_cl = numeric(k.set[k])
  impk = 0
  for(i in 1:k.set[k]){#go through each cluster
    clu<-subset(full_data,full_data$cluster == i)
    clu_1 = table(clu$target)[1]/clus_Size[i]
    clu_2=table(clu$target)[2]/clus_Size[i]
    clu_3=table(clu$target)[3]/clus_Size[i]
```

```
    gini_cl = clu_1*(1-clu_1)+clu_2*(1-clu_2)+clu_3*(1-clu_3)
    impk = impk+gini_cl

  }
  gin[k] = impk
  #print(gini_cl)
}
gini_final = numeric(length(k.set))
for(i in 1:length(k.set)){
  gini_final[i] = gin[i]/(.6667*k.set[i])
}
gini_plot=data.frame(gini_final)
ggplot(gini_plot , aes(x=k.set, y=gini_plot$gini_final)) +
  geom_line(color="grey") +
  geom_point(shape=21, color="black", fill="#69b3a2", size=3)+
  scale_x_continuous(breaks = scales::pretty_breaks(n = 15)) +
  scale_y_continuous(breaks = scales::pretty_breaks(n = 10)) +
  xlab("k")+
  ylab("gini k")+
  ggtitle("k vs gini index")
###################Q3###########
##k=15
kmbest<- kmeans(S_NYC_features,15,nstart = 100,iter.max=100)
kmbest.centers = kmbest$centers
write.table(kmbest.centers, file = "K Means Centriod values.csv", sep = ",", row.names = TRUE)
kmbest$withinss
kmbest$tot.withinss
kmbest$size

full_data= S_NYC1




full_data= S_NYC1
full_data$cluster = NA
full_data$cluster=as.factor(kmbest$cluster)
gini_cl = numeric(15)
clu_11 = numeric(15)
clu_22= numeric(15)
clu_33 = numeric(15)
clus_Siz = numeric(15)
wss = numeric(15)
clus_Size = kmbest$size
for(i in 1:15){
  clus_Siz[i]= kmbest$size#go through each cluster
  clu<-subset(full_data,full_data$cluster == i)
  clu_1 = table(clu$target)[1]/clus_Size[i]
  clu_11[i] =  clu_1
  clu_2=table(clu$target)[2]/clus_Size[i]
  clu_22[i] =  clu_2
  clu_3=table(clu$target)[3]/clus_Size[i]
  clu_33[i] =  clu_3
  gini_cl[i] = clu_1*(1-clu_1)+clu_2*(1-clu_2)+clu_3*(1-clu_3)
}


wss=kmbest$withinss
desc = data.frame(cbind(wss,clus_Size,gini_cl,clu_11,clu_22,clu_33))
###3D projection3
########ACCURACY OF K=15###########
S_NYC_15 = S_NYC1
S_NYC_15$cluster<-as.factor(kmbest$cluster)
```

```r
#######gini index of best k#####
full_data= S_NYC1
full_data$cluster = NA
full_data$cluster=as.factor(kmbest$cluster)
gini_cl = numeric(15)
clu_11 = numeric(15)
clu_22= numeric(15)
clu_33 = numeric(15)
clus_Siz = numeric(15)
clus_Size = kmbest$size
for(i in 1:15){
  clu<-subset(full_data,full_data$cluster == i)
  clu_1 = table(clu$target)[1]/clus_Size[i]
  clu_11[i] =  clu_1
  clu_2=table(clu$target)[2]/clus_Size[i]
  clu_22[i] =  clu_2
  clu_3=table(clu$target)[3]/clus_Size[i]
  clu_33[i] =  clu_3
  gini_cl[i] = clu_1*(1-clu_1)+clu_2*(1-clu_2)+clu_3*(1-clu_3)
}

#######Q4######
set.seed(1)
LOWPRICE = S_NYC1[which(S_NYC1$target =="Low"),]
set.seed(1)
MEDPRICE = S_NYC1[which(S_NYC1$target =="Medium"),]
set.seed(1)
HIGHPRICE = S_NYC1[which(S_NYC1$target =="High"),]



##TRAIN/TEST SET: LOW
n<-nrow(LOWPRICE)
train<-sample(1:n, 0.8*n)
trainset_LOW <- LOWPRICE[train,]
testset_LOW <- LOWPRICE[-train,]

##TRAIN/TEST SET: MEDIUM
n<-nrow(MEDPRICE)
train<-sample(1:n, 0.8*n)
trainset_MED <- MEDPRICE[train,]
testset_MED <- MEDPRICE[-train,]

##TRAIN/TEST SET: HIGH
n<-nrow(HIGHPRICE)
train<-sample(1:n, 0.8*n)
trainset_HIGH <- HIGHPRICE[train,]
testset_HIGH <- HIGHPRICE[-train,]
#combined train set
TRAIN_SET = rbind(trainset_LOW,trainset_MED,trainset_HIGH)#High:10176  Low: 10243  Medium:10655
TEST_SET = rbind(testset_LOW,testset_MED,testset_HIGH)#High: 2544   Low:2561   Medium:2664

TRAIN_predictor <- TRAIN_SET[,-c(16)]
TRAIN_label <- TRAIN_SET[, "target"]
TEST_predictor <- TEST_SET[,-c(16)]
TEST_label <- TEST_SET[,"target"]

TRAIN_predictor=as.data.frame(TRAIN_predictor)
TEST_predictor=as.data.frame(TEST_predictor)
TRAIN_label=as.data.frame(TRAIN_label)
TEST_label=as.data.frame(TEST_label)
```

```r
library(randomForest)
library(caret)
TRAIN_SET$target<- factor(TRAIN_SET$target) # changing label to factors
TEST_SET$target<- factor(TEST_SET$target)
set.seed(1)
start = Sys.time() ####1.706 mins
RF_100 <- randomForest(target ~., data=TRAIN_SET,ntree = 100, ntry=4
                       ,importance=T)
RF_200 <- randomForest(target ~., data=TRAIN_SET,ntree = 200, ntry=4
                       ,importance=T)
RF_300 <- randomForest(target ~., data=TRAIN_SET,ntree = 300, ntry=4
                       ,importance=T)
RF_400 <- randomForest(target ~., data=TRAIN_SET,ntree = 400, ntry=4
                       ,importance=T)
end=Sys.time()
Time=end-start
predSET_100tr<-predict(RF_100,newdata = TRAIN_SET)
predSET_100test<-predict(RF_100,newdata = TEST_SET)
results_100tr = confusionMatrix(predSET_100tr, TRAIN_SET$target)
results_100test = confusionMatrix(predSET_100test, TEST_SET$target)

predSET_200tr<-predict(RF_200,newdata = TRAIN_SET)
results_200tr = confusionMatrix(predSET_200tr, TRAIN_SET$target)
predSET_200test<-predict(RF_200,newdata = TEST_SET)
results_200test = confusionMatrix(predSET_200test, TEST_SET$target)

predSET_300tr<-predict(RF_300,newdata = TRAIN_SET)
results_300tr = confusionMatrix(predSET_300tr, TRAIN_SET$target)
predSET_300test<-predict(RF_300,newdata = TEST_SET)
results_300test = confusionMatrix(predSET_300test, TEST_SET$target)


predSET_400tr<-predict(RF_400,newdata = TRAIN_SET)
results_400tr = confusionMatrix(predSET_400tr, TRAIN_SET$target)
predSET_400test<-predict(RF_400,newdata = TEST_SET)
results_400test = confusionMatrix(predSET_400test, TEST_SET$target)

Train_Acc = c(results_100tr$overall[1],results_200tr$overall[1],results_300tr$overall[1],results_400tr$overall[1])
Test_Acc = c(results_100test$overall[1],results_200test$overall[1],results_300test$overall[1],results_400test$overall[1])
nfor = c(100,200,300,400)

ggplot(data.frame(cbind(ntree,Train_Acc,Test_Acc)), aes(x=ntree))+
  geom_point(aes(y = Train_Acc, color = "darkred"),shape=18,size=3)+
  geom_line(aes(y = Train_Acc,color = "darkred"),linetype="longdash") +
  geom_point(aes(y = Test_Acc,color = "steelblue"),shape=18,size=3)+
  geom_line(aes(y = Test_Acc,color="steelblue"), linetype="dotdash")+
  ylim(.63,.85)+
  scale_color_identity(name = "",
                       labels = c("Train-set", "Test-set"),
                       guide = "legend") +
  xlab("Number of forests")+
  ylab("Accuracy")+
  ggtitle("Train vs Test set Accuracy")

####Q6:
#high:cl1,low # cl2, medium cl3
CL1_coef =c(results_100test$table[1],results_200test$table[1],results_300test$table[1],results_400test$table[1])
CL2_coef =  c(results_100test$table[5],results_200test$table[5],results_300test$table[5],results_400test$table[5])
CL3_coef = c(results_100test$table[9],results_200test$table[9],results_300test$table[9],results_400test$table[9])
```

```
ggplot(data.frame(cbind(nfor,CL1_coef,CL2_coef,CL3_coef)), aes(x=nfor))+
  geom_point(aes(y = CL1_coef, color = "darkred"),shape=18,size=3)+
  geom_line(aes(y = CL1_coef,color = "darkred"),linetype="longdash") +
  geom_point(aes(y = CL2_coef,color = "steelblue"),shape=18,size=3)+
  geom_line(aes(y = CL2_coef,color="steelblue"), linetype="dotdash")+
  geom_point(aes(y = CL3_coef, color = "green"),shape=18,size=3)+
  geom_line(aes(y = CL3_coef,color = "green"),linetype="dashed") +
  scale_color_identity(name = "Class coefficient",
                       labels = c("High", "Low","Medium"),
                       guide = "legend") +
  xlab("Number of forests")+
  ylab("Class Coefficient")+
  ggtitle("Diagonal coefficient vs Number of trees")+
  ylim(1200,2300)

############################Q7#####################################################
#Bntr = 400

RF_200 <- randomForest(target ~., data=TRAIN_SET,ntree = 400, ntry=4
                       ,importance=T)

predSET_400tr<-predict(RF_400,newdata = TRAIN_SET)
results_400tr = confusionMatrix(predSET_400tr, TRAIN_SET$target)
predSET_400test<-predict(RF_400,newdata = TEST_SET)
results_400test = confusionMatrix(predSET_400test, TEST_SET$target)


varImpPlot((RF_400))
Importance = data.frame(importance(RF_400))
Importance_features = Importance[order(Importance$MeanDecreaseAccuracy,decreasing = TRUE),]
View(Importance_features)
write.csv(Importance_features,"Importance.csv", row.names = TRUE)

#Most important: latitude

set.seed(1)
LOWPRICE = S_NYC[which(S_NYC$target =="Low"),]
set.seed(1)
MEDPRICE = S_NYC[which(S_NYC$target =="Medium"),]
set.seed(1)
HIGHPRICE = S_NYC[which(S_NYC$target =="High"),]
###Compare histograms visually and conduct ks-test
par(mfrow = c(1,3))
hist(LOWPRICE$latitude, main = "Low Price" , col = "lightblue", xlab = "latitude")
hist(MEDPRICE$latitude, main = "Medium Price" , col = "lightblue", xlab = "latitude")
hist(HIGHPRICE$latitude, main = "High Price" , col = "lightblue", xlab = "latitude")

ks.test(LOWPRICE$latitude,MEDPRICE$latitude)
ks.test(LOWPRICE$latitude,HIGHPRICE$latitude)
ks.test(MEDPRICE$latitude,HIGHPRICE$latitude)
#diff population

par(mfrow = c(1,3))
hist(LOWPRICE$neighbourhood_group_Staten.Island, main = "Low Price" , col = "yellow", xlab = "staten")
hist(MEDPRICE$neighbourhood_group_Staten.Island, main = "Medium Price" , col = "yellow", xlab = "staten")
hist(HIGHPRICE$neighbourhood_group_Staten.Island, main = "High Price" , col = "yellow", xlab = "staten")

ks.test(LOWPRICE$neighbourhood_group_Staten.Island,MEDPRICE$neighbourhood_group_Staten.Island)
ks.test(LOWPRICE$neighbourhood_group_Staten.Island,HIGHPRICE$neighbourhood_group_Staten.Island)


########Q9#######
full_data= S_NYC1
full_data$cluster = NA
full_data$cluster=as.factor(kmbest$cluster)
cl11<-subset(full_data,full_data$cluster == 11)
table(cl4$target) # High:33, Low: 663, Medium:101
size = kmbest$size
cl11 = clu
###TRAIN/TEST SET
cl11_low = cl11[which(cl4$target == "Low"),]
cl11_medium = cl11[which(cl4$target == "Medium"),]
cl11_high = cl11[which(cl4$target == "High"),]
#low
n<-nrow(cl11_low)
train<-sample(1:n, 0.8*n)
trainset_cl11_low <- cl11_low[train,]
testset_cl11_low <- cl11_low[-train,]
#undersample
n<-nrow(trainset_cl11_low)
train<-sample(1:n, 0.5*n)
trainset_cl11_low_under = trainset_cl11_low[train,]

n<-nrow(testset_cl11_low)
train<-sample(1:n, 0.5*n)
testset_cl11_low_under = testset_cl11_low[train,]
```

```
n<-nrow(testset_cl11_low)
train<-sample(1:n, 0.5*n)
testset_cl11_low_under = testset_cl11_low[train,]

##TRAIN/TEST SET: MEDIUM
n<-nrow(cl4_medium)
train<-sample(1:n, 0.8*n)
trainset_cl4_medium <- cl4_medium[train,]
testset_cl4_medium<- cl4_medium[-train,]

##TRAIN/TEST SET: HIGH
n<-nrow(cl4_high)
train<-sample(1:n, 0.8*n)
trainset_cl4_high <- cl4_high[train,]
testset_cl4_high<- cl4_high[-train,]

#combined train and test set
TRAIN_SET_CL4 = rbind(trainset_cl4_low_under,trainset_cl4_medium,trainset_cl4_high)#High:26  Low: 530  Medium:80
#after undersample: 26,265 and 80
TEST_SET_CL4 = rbind(testset_cl4_low_under,testset_cl4_medium,testset_cl4_high)#High: 7   Low:133   Medium:21
#after undersample: 7,66 and 21
table(TEST_SET_CL4$target)#26:high,#530 low,#80 medium
cloned_tr = TRAIN_SET_CL4[,-17]
library("smotefamily")
table(cloned_tr$target)#H:128,LOW:1548,MEDIUM:926
train.CLONED = SMOTE(cloned_tr[,-c(16)], # feature values
                     as.numeric(cloned_tr[,16]), # class labels
                     K = 4, dup_size = 7)

TRAIN_CLONED<-train.CLONED$data
table(TRAIN_CLONED$class) #high:1024,low:1548,medium:926
trainCLONED = SMOTE(TRAIN_CLONED[,-c(16)], # feature values
                    as.numeric(TRAIN_CLONED[,16]), # class labels
                    K = 4, dup_size = 2)
TRAIN_CLONED<-trainCLONED$data
table(TRAIN_CLONED$class) #208,265 and 2778

######cloning for cluster RF##############
cloned_ts = TEST_SET_CL4[,-17]
library("smotefamily")
table(cloned_ts$target)#7,66,21

test.CLONED = SMOTE(cloned_ts[,-c(16)], # feature values
                    as.numeric(cloned_ts[,16]), # class labels
                    K = 4, dup_size = 7)

TEST_CLONED<-test.CLONED$data
table(TEST_CLONED$class) #high:182,low:265,medium:
testCLONED = SMOTE(TEST_CLONED[,-c(16)], # feature values
                   as.numeric(TEST_CLONED[,16]), # class labels
                   K = 4, dup_size = 2)
TEST_CLONED<-testCLONED$data
table(TEST_CLONED$class) #h56,m66 and l63
TRAIN_CLONED_CL4 = data.frame(TRAIN_CLONED)
TRAIN_CLONED_CL4$class = as.factor(TRAIN_CLONED_CL4$class)
TEST_CLONED_CL4=data.frame(TEST_CLONED)
TEST_CLONED_CL4$class = as.factor(TEST_CLONED_CL4$class)
```

```
TRAIN_CLONED_CL4$class = as.factor(TRAIN_CLONED_CL4$class)
TEST_CLONED_CL4=data.frame(TEST_CLONED)
TEST_CLONED_CL4$class = as.factor(TEST_CLONED_CL4$class)
#######Q10######
RF_400_CL4 <- randomForest(class ~., data=TRAIN_CLONED_CL4,ntree = 400, ntry=4
                        ,importance=T)


pred_400_CL4_ts<-predict(RF_400_CL4,newdata = TEST_CLONED_CL4)
pred_400_CL4_tr<-predict(RF_400_CL4,newdata = TRAIN_CLONED_CL4)
results_400_CL4_tr = confusionMatrix(pred_400_CL4_tr, TRAIN_CLONED_CL4$class)
results_400_CL4_ts = confusionMatrix(pred_400_CL4_ts, TEST_CLONED_CL4$class)

#on SDATA
TEST_SET_Q10 = TEST_SET
TEST_SET_LABEL = TEST_SET
for (i in 1:nrow(TEST_SET_Q10)){
  if(TEST_SET_label$target[i]=="Low"){
    TEST_SET_Q10$class[i]="2"
  }else if(TEST_SET_label$target[i]=="Medium"){
    TEST_SET_Q10$class[i]="3"
  }else{
    TEST_SET_Q10$class[i]="1"
  }
}

TEST_SET_Q10$class = as.factor(TEST_SET_Q10$class)
TEST_SET_Q10 = TEST_SET_Q10[,-16]
pred_400_CL4_ts<-predict(RF_400_CL11,newdata = TEST_SET_Q10)
results= confusionMatrix(pred_400_CL11_ts, TEST_SET_Q10$class)


###############Q11#######################
set.seed(2)
LOWPRICE = S_NYC1[which(S_NYC1$target =="Low"),]
set.seed(2)
MEDPRICE = S_NYC1[which(S_NYC1$target =="Medium"),]
set.seed(2)
HIGHPRICE = S_NYC1[which(S_NYC1$target =="High"),]

##TRAIN/TEST SET: LOW
n<-nrow(LOWPRICE)
train<-sample(1:n, 0.8*n)
trainset_LOW <- LOWPRICE[train,]
testset_LOW <- LOWPRICE[-train,]

##TRAIN/TEST SET: MEDIUM
n<-nrow(MEDPRICE)
train<-sample(1:n, 0.8*n)
trainset_MED <- MEDPRICE[train,]
testset_MED <- MEDPRICE[-train,]

##TRAIN/TEST SET: HIGH
n<-nrow(HIGHPRICE)
train<-sample(1:n, 0.8*n)
trainset_HIGH <- HIGHPRICE[train,]
testset_HIGH <- HIGHPRICE[-train,]
```

```
#######Q11#########
##class low versus medium####
TRAIN_SET_LM = rbind(trainset_LOW,trainset_MED)#low:10243, medium:10655
TEST_SET_LM = rbind(testset_LOW,testset_MED)#low:2561, medium:2664

##class medium versus high###
TRAIN_SET_MH = rbind(trainset_MED,trainset_HIGH)#medium:10655, high:10176
TEST_SET_MH = rbind(testset_MED,testset_HIGH)#medium:2664, #HIGH:2544

###class low versus high###
TRAIN_SET_LH = rbind(trainset_LOW,trainset_HIGH)#LOW:10243, high:10176
TEST_SET_LH = rbind(testset_LOW,testset_HIGH)#low:2544, #HIGH:2544

TRAIN_SET_LH$target = droplevels.factor(TRAIN_SET_LH$target)
TEST_SET_LH$target = droplevels.factor(TEST_SET_LH$target)

#######TRAIN SVM######
library(e1071)
library("caret")
start = Sys.time()#12.466148 mins
set.seed(1)
#1.69 mins
###time:
tune.out=tune(method="svm",
              target~.,
              data=TRAIN_SET_LH,
              kernel="linear",
              ranges = list(cost = c(0.001,0.01, 0.1, 1,5,10)))

bestmod = tune.out$best.model
summary(tune.out)


##final svm
start = Sys.time()
best = svm(formula = as.factor(target) ~ ., data = TRAIN_SET_LH, kernel = "linear", cost = 5)

plot(best,TRAIN_SET_LH,room_type_Entire.home.apt~reviews_per_month,fill=TRUE,col = c(gray(0.8),"pink"))

##medium low best cost 5

LM = svm(formula = as.factor(target) ~ ., data = TRAIN_SET_LM, kernel = "linear", cost = 5)
ypredLM <- predict(LM,TEST_SET_LM)
confusionMatrix(ypredLM,TEST_SET_LM$target)

MEDIUM HIGH best cost 5
MH = svm(formula = as.factor(target) ~ ., data = TRAIN_SET_MH, kernel = "linear", cost = 5)
ypred <- predict(MH,TEST_SET_MH)
confusionMatrix(ypred,TEST_SET_MH$target)

######
LH = svm(formula = as.factor(target) ~ ., data = TRAIN_SET_LH, kernel = "linear", cost = 5)
ypredLH <- predict(LH,TEST_SET_LH)
confusionMatrix(ypredLH,TEST_SET_LH$target)
```