ISEN 613 |Engineering Data Analysis

PREDICTION OF BROOKLYN HOME SALES PRICES

Table of Contents

EXECUTIVE SUMMARY

For this project, we analyzed a data set corresponding obtained from NYC Department of Finance website to develop important insights between different features and to write a predictive model to estimate the value of a property for Brooklyn City for given values of these parameters.

Planning has become a basic need of this decade. Planning allow both individuals and organizations to avoid undesired situations, gain better economic stability and thus, obtain a better quality in operations and living.

We did feature engineering, to create new attributes which categorized variables in a new way, and thus, makes it easier to understand and straight forward to model. The original data set was cleaned, and variables that not generated added value were removed. After obtaining a new data set, both univariate and bivariate analysis was performed.

In the Univariate Analysis each attribute was, removing outliers and ambiguous values. For example, Sale price values of $1, zip codes with null values and empty cells for any given attribute. Moreover, the Bivariate Analysis consisted in checking for correlation between attributes and dropping a few more predictors which we found to be highly correlated. After the dataset was cleaned, and both Univariate and Bivariate analysis were completed, we continued with the model fitting stage.

For model fitting, we tried different methods. Among those were Tree Regression (Boosting & Random Forest), K Nearest Neighbor, Multiple Linear Regression, Ridge Regression and Lasso Regression.

After fitting and analyzing each model, we found that Boosting and Random forest produce good accuracy, but yield poor interpretability. Whereas, Multiple Linear Regression, Lasso Regression, Ridge Regression produce less accurate result, but more interpretable models.

INTRODUCTION

Today we live in the era of information, a fast-paced and completely dynamic environment, where every choice plan or strategy, either in industry or by individuals, are based on informed decisions. Part of this informed decisions have to do with the ability to identify and predict future behavior of different items and issues.

Brooklyn is the neighborhood of New York City that has developed de most in the last ten years. Since 2006 real estate prices have increased significantly. In 2006 the average price per square foot for condos and townhouses were approximately $330. In 2017 average price per square foot for properties of the same time was approximately $950. The low availability and the increasing demand, has made Brooklyn properties to hit record numbers. In the first quarter of 2018, median selling price rose to $800,000, a 3.5% increase from previous prices.

If in need to purchase a property in the area, it is extremely important to be able to predict what the actual price for a home is, so that one can find better opportunities, and also no overpay for properties with inflated prices.

For this project we analyzed a data set corresponding to Brooklyn, NY housing market. Planning has become a basic need of this decade. Planning allow both individuals and organizations to avoid undesired situations, gain better economic stability and thus, obtain a better quality in operations and living.

The goal of our study is to be able to predict future home prices, and later understand how different attributes positively or negatively affect house prices in the neighborhood of Brooklyn.

PROJECT APPROACH

Data description

The data set corresponds to 2003-2017 sales price of Brooklyn, NY, real estate properties.
This dataset had 390883 observations and 111 attributes.

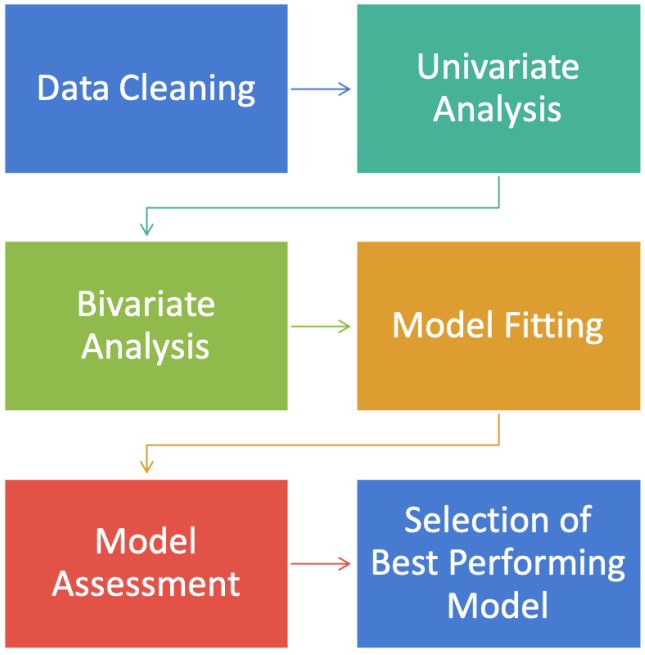
1. Duplicate attributes: the dataset was generated by LEFT JOIN of two different files so both the files had some columns that gave the same information.
2. Irrelevant variables: like Identifiers, Files IDs , codes generated by concatenation were removed.
3. Attributes having large no. of missing values were also removed.
4. For some observations *sale_price* = 0 which implies that the property has been transferred so these values will not help in prediction and were thus removed. This reduced the number of observations to 250K.

After achieving a dimension of 250k X 65. We still had many NA values in our data. I adopted two approaches for missing values:

1. Replaced missing values with median and mode in continuous and categorical variables respectively. This object is named as 'data_stage2'
2. Removed all observations with NA values. This object is 'no_NA_data'

Method

In order to conduct the analysis, we followed the different set of steps depicted in the flowchart below.



DATA CLEANING

The first step, and maybe the most essential was cleaning the data. The data contains a large number of variables which increase the noise and reduce the prediction capability of the model due to increased variance. We need to reduce the number of variables to the maximum before model fitting.
Going through the attribute description (attached in appendix), we can remove many variables on the following grounds.

1. Duplicity -Reading from the description we find that many variables have same information but just different names. This is the case as the dataset is a product of concatenation of two different tables.
2. No Description- Many variables have no proper description. This may be the case as many variables are representations of various IDs of PLUTOMAP files which supposedly have no direct impact on the predictions made.
3. Large percentage of missing data- We have discarded all variables where missing values are more than 75% of all observations.
4. Removal of observations with empty cells.

Variable_Name <fctr>	Total_Missing_Values <dbl>	Percent_Values_missing <dbl>
BoroCode	87155	22.29695
BBL	87155	22.29695
CondoNo	87155	22.29695
Tract2010	87155	22.29695
XCoord	87155	22.29695
YCoord	87155	22.29695
ZoneMap	87155	22.29695
ZMCode	384771	98.43636
Sanborn	87173	22.30156
TaxMap	87173	22.30156
EDesigNum	387329	99.09078
APPBBL	87155	22.29695
APPDate	371624	95.07295

92–104 of 111 rows

```r
data_stage2 = subset(data, select = -c(borough, Borough, V1, UnitsRes, UnitsTotal, LotArea, BldgArea, BldgClass, Easements, easement,
OwnerType, building_class_category, ZipCode, YearBuilt, MAPPLUTO_F, PLUTOMapID, SHAPE_Leng, SHAPE_Area, Address, EDesigNum,Version,
Sanborn, ZoneMap, ZMCode, HistDist,Landmark, APPDate, FIRM07_FLA, PFIRM15_FL, Ext, AreaSource, sale_date,ZoneDist2, ZoneDist3, ZoneDist4,
Overlay1, Overlay2, SPDist1, SPDist2, SPDist3, LtdHeight ))

dim(data_stage2)
```

```
[1] 390883     70
```

5. For some observations sale_price = 0 which implies that the property has been transferred so these values will not help in prediction and were thus removed. This reduced the number of observations to 250K.

```
[1] 250740     70
```

We still have 70 attributes left in our data, which are still way too many for proper prediction. We further analyze remaining attributes and try to reduce even more.

Then we look at the structure of our attributes. We find that both *address* and *apartment_number* entries are exactly the same. Moreover, they only represent the house number and do not give any perception of the area or street or block where the property is present. So we can omit these attributes.

BBL and *APPBBL* also appear to be meant for identification purpose only. It is concatenation of the borough code, tax block and tax lot, which are already available to us separately. After doing this we end up with 65 remaining variables.

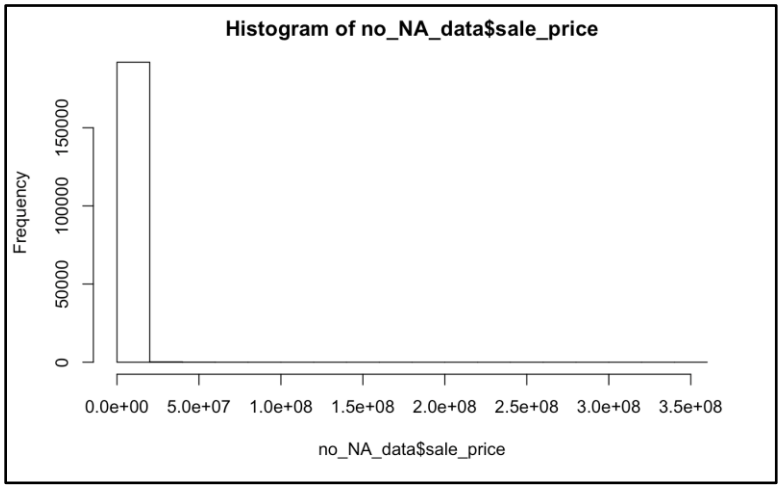
193218 data observations have missing values for one or the other attributes. We remove these observations also.

```
```{r}
dim(no_NA_data)
```

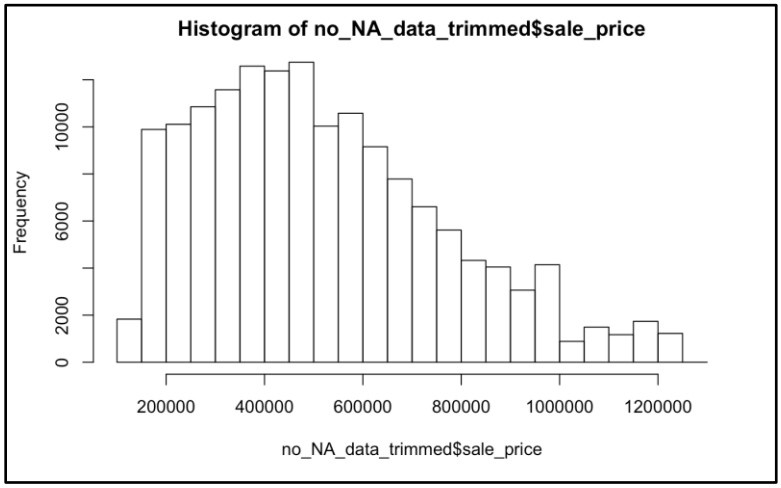
 [1] 192269   65
```

UNIVARIATE ANALYSIS

We want to predict sales price of brooklyn holmes. Let's look at the histogram of the response variable.

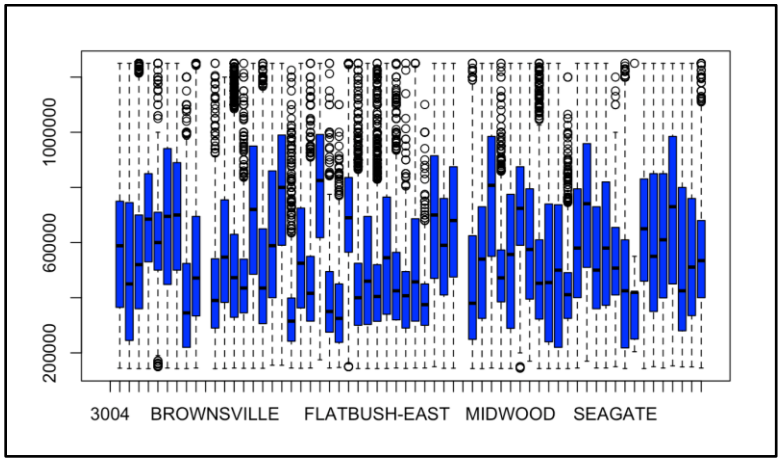


We see some outliers at both the ends. So we take the observations from 10 percentile to 90 percentile of *sale_price*. For the 10% Percentile we get 142,524 and for the 90% Percentile 1,250,500 observations, respectively.



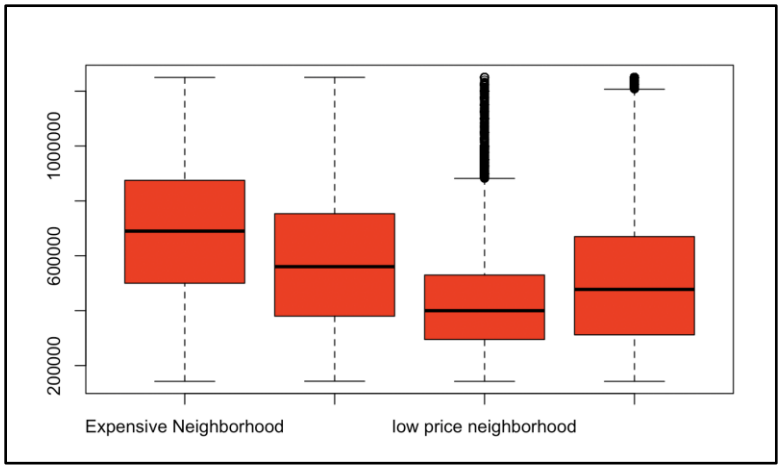
Here we get a fairly uniform distribution of response variable i.e *sale_price.*

Attribute analysis: *Neighborhood*
Department of Finance assessors determine the neighborhood name in the course of valuing properties. The common name of the neighborhood is generally the same as the name Finance designates. However, there may be slight differences in neighborhood boundary lines and some sub-neighborhoods may not be included.



Here we have 63 levels in neighborhood variable. This makes it difficult for the model and may lead to overfitting. So we do feature engineering and create new attribute for categorizing neighborhood into 4 categories.

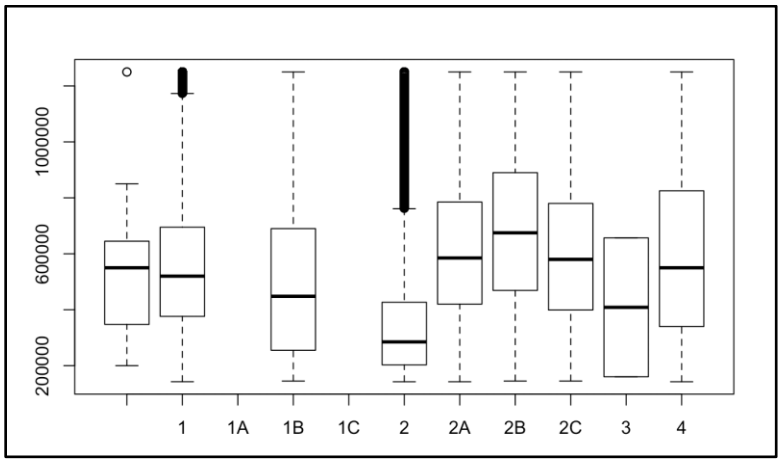
```r
#creating new categories for neighborhood
nb_class$category = ifelse(nb_class$mean>300000 & nb_class$mean<452407.5, "low price neighborhood",ifelse
  (nb_class$mean>452407.5 & nb_class$mean<554618.1, "medium price neighborhood",ifelse
    (nb_class$mean>554618.1 & nb_class$mean<631312.3, "High price neighborhood", "Expensive Neighborhood")))
```



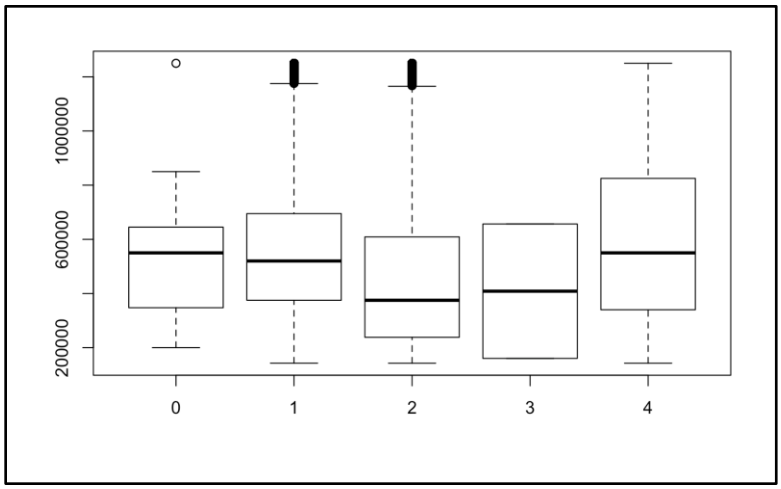
Low price neighborhood has some outliers.

Attribute analysis: *Tax Class*

Every property is assigned a tax class (Classes 1, 2, 3, and 4), based on the use of the property.
. Class 1: Includes most residential property of up to three units (such as one-, two-, and three-family homes and small stores or offices with one or two attached apartments), vacant land that is zoned for residential use, and most condominiums that are not more than three stories.
. Class 2: Includes all other property that is primarily residential, such as cooperatives and condominiums.
. Class 3: Includes property with equipment owned by a gas, telephone or electric company.
. Class 4: Includes all other properties not included in class 1,2, and 3, such as offices, factories, warehouses, garage buildings, etc.



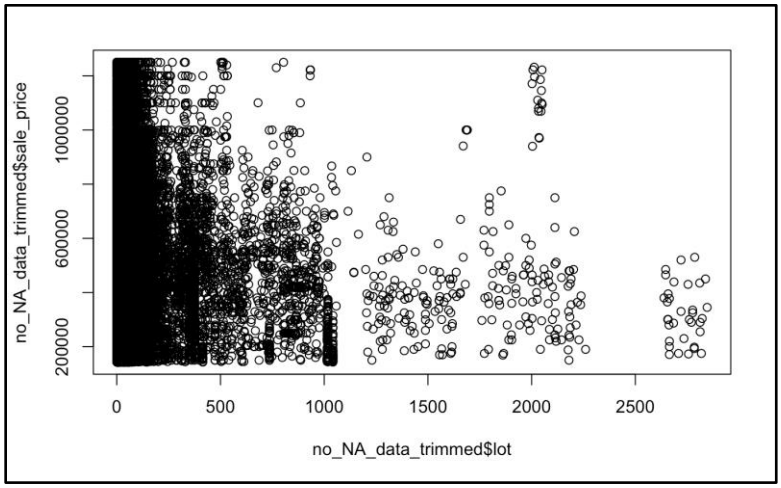
We see that there are 11 factors in Tax Class whereas ideally it should be only 1,2,3 & 4 as per the description. So we merge 1, 1A, 1B, 1C to 1 and 2, 2A, 2B, 2C to 2.



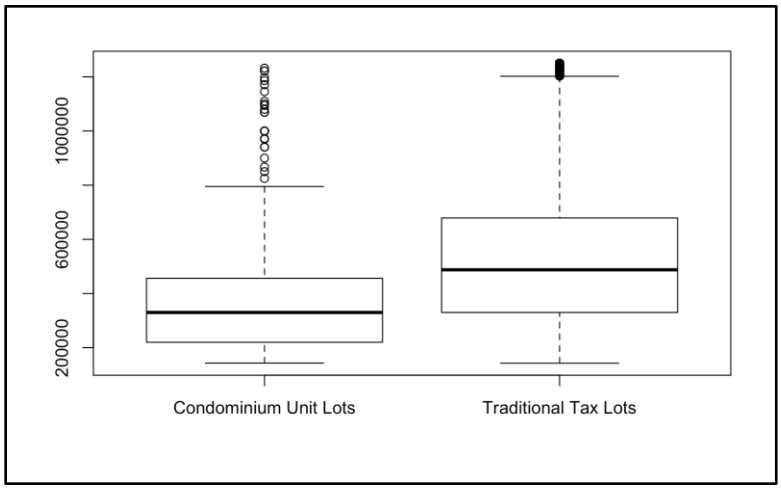
We see that *tax_class* '2' has some outliers and the *sale_price* is randomly distributed among the tax classes.

Attribute Analysis: *Lot*
Often the Tax Lot number can tell you the type of tax lot. The following table identifies some of these tax lot numbering conventions. Of course there are exceptions to each convention.

TAX LOT NUMBER	TYPE OF LOT
1-999	Traditional Tax Lots
1001-6999	Condominium Unit Lots
7501-7599	Condominium Billing Lots
8000-8899	Subterranean Tax lots
8900-8999	DTM Dummy Tax Lots
9000-9899	Air Rights Tax Lots



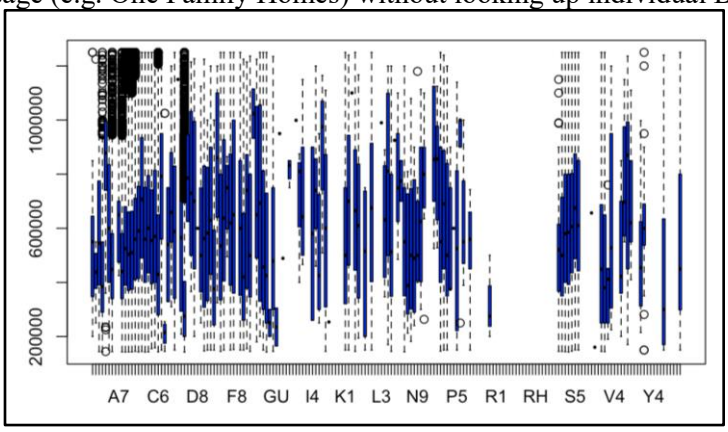
No correlation found. in this raw plot. But we can create categories of TYPE OF PLOT as defined in the description.



From the above boxplot we can infer that the median of sale price of Traditional Tax lots is higher than the median sale price of condominium plots.

Attribute analysis: *Building Class*

This is a field that we are including so that users of the Rolling Sales Files can easily identify similar properties by broad usage (e.g. One Family Homes) without looking up individual Building Classes.



We can see that some building classes do not form any box. For better understanding, we count the number of rows in each column to check the discrepancy.

```r
bc = data.frame(Building_Class = levels(no_NA_data_trimmed$building_class), count = tabulate(no_NA_data_trimmed$building_class))

bc[order(bc$count),]
```

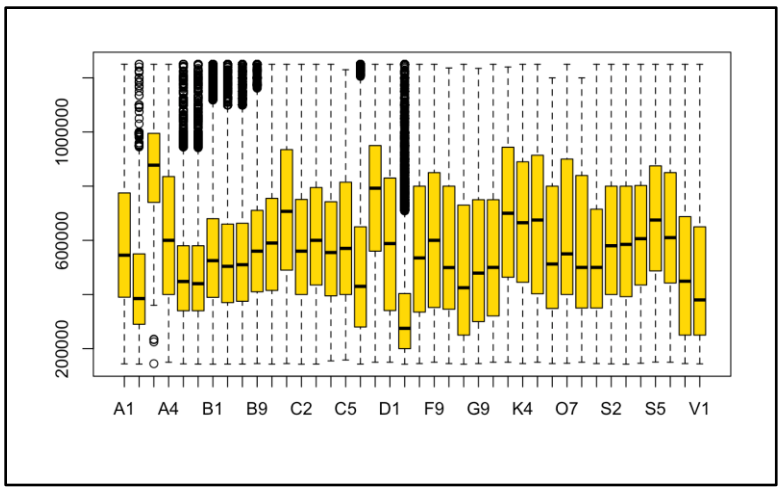
	Building_Class <fctr>	count <int>
8	A6	0
45	F8	0
60	H3	0
62	H5	0
66	HB	0
67	I1	0
74	J3	0
75	J6	0
76	J8	0
77	J9	0

1–10 of 180 rows Previous 1 2 3 4 5 6 ... 18 Next

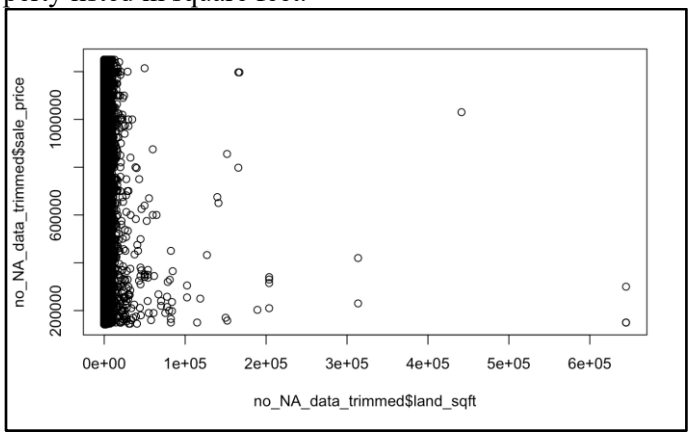
Now, We find that many factors have been retained even after subsetting or indexing the data frame from the original data. These factors provide no information as they don't contain any data. We decide to drop them. We just need to apply factor again to our subsetted dataset.

We further remove building classes that have less than 100 count as they will not give a good estimate of the sale price.

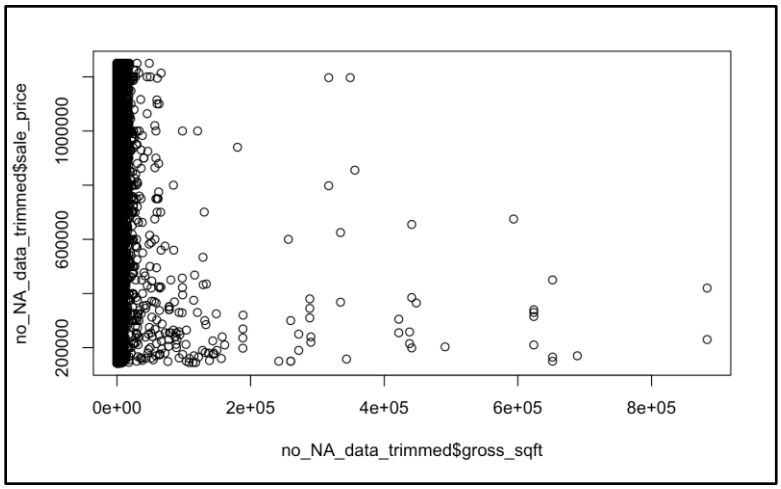
We get our final boxplot as follows:



Attribute analysis: *Land_Sqft*
The land area of the property listed in square feet.

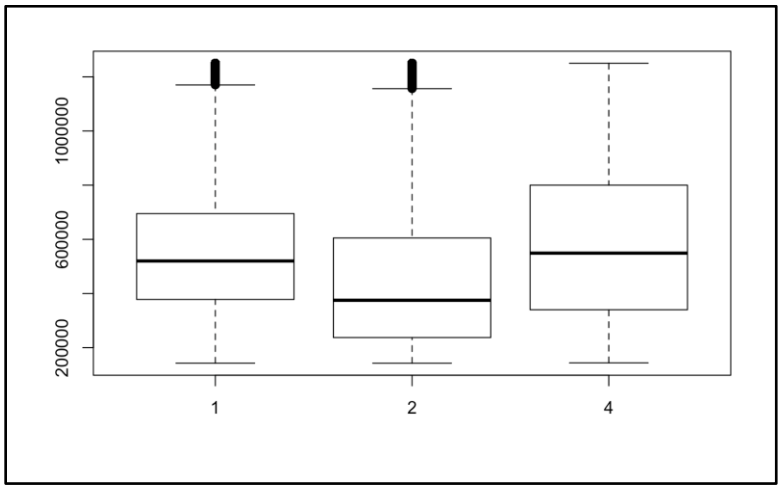


Attribute analysis: *gross_sqft*
The land area of the property listed in square feet.



As land square feet area and gross square feet area are correlated. We can omit gross_sqft.

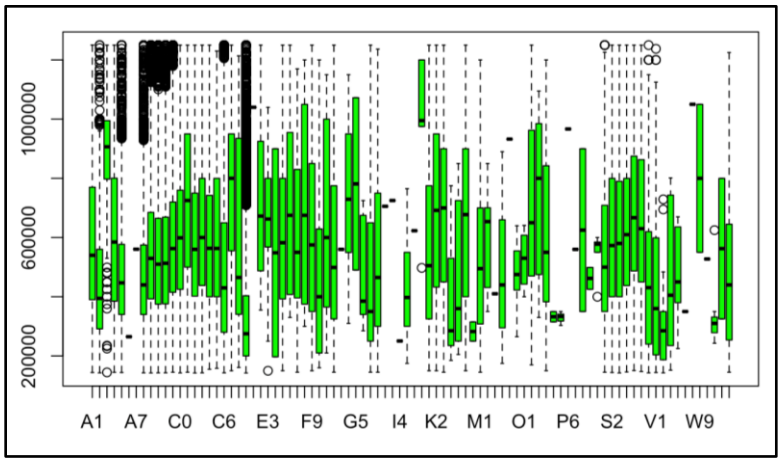
Attribute Analysis: ***Tax Class at Sale-***



This plot is exactly same as Tax Class. So we remove this too.

Attribute analysis: ***Building_class_at_sale***

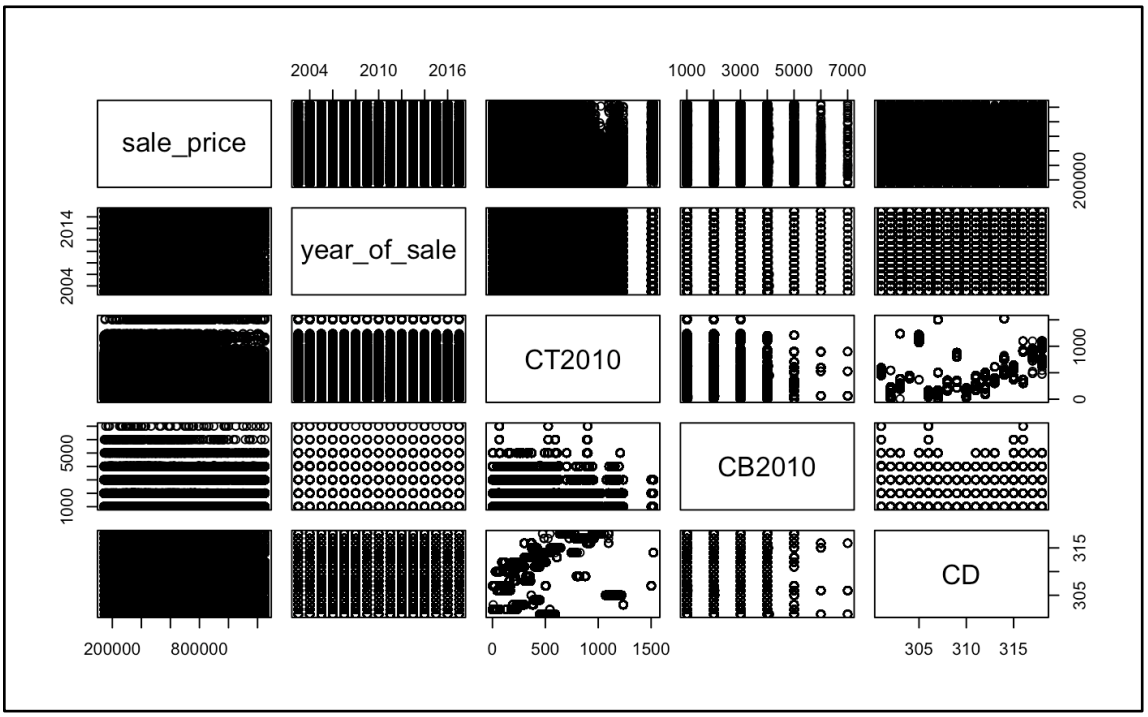
The Building Classification is used to describe a property's constructive use. The first position of the Building Class is a letter that is used to describe a general class of properties (for example "A" signifies one-family homes, "O" signifies office buildings. "R" signifies condominiums). The second position, a number, adds more specific information about the property's use or construction style (using our previous examples "A0" is a Cape Cod style one family home, "O4" is a tower type office building and "R5" is a commercial condominium unit). The term Building Class as used by the Department of Finance is interchangeable with the term Building Code as used by the Department of Buildings.



This is same as building_class so we can remove this attribute.

Bivariate Analysis

We checked the correlation between different variables and removed the ones found to be correlated



After conducting univariate and bivariate analysis for all attributes, we finalized the following 30 attributes for our model fitting to predict the sale price of residential units.

```r
final_data = subset ( no_NA_data_trimmed, select = c(sale_price, building_class, category, tax_class, type_of_lot,zip_code,
residential_units, gross_sqft,year_built, year_of_sale, CD, Council, LandUse,ResArea, GarageArea, StrgeArea, NumBldgs,NumFloors,
BldgFront, BldgDepth, ProxCode,IrrLotCode, LotType, AssessTot, ExemptTot, YearAlter1, YearAlter2, BuiltFAR, XCoord, YCoord) )
```

```r
dim(final_data)
```

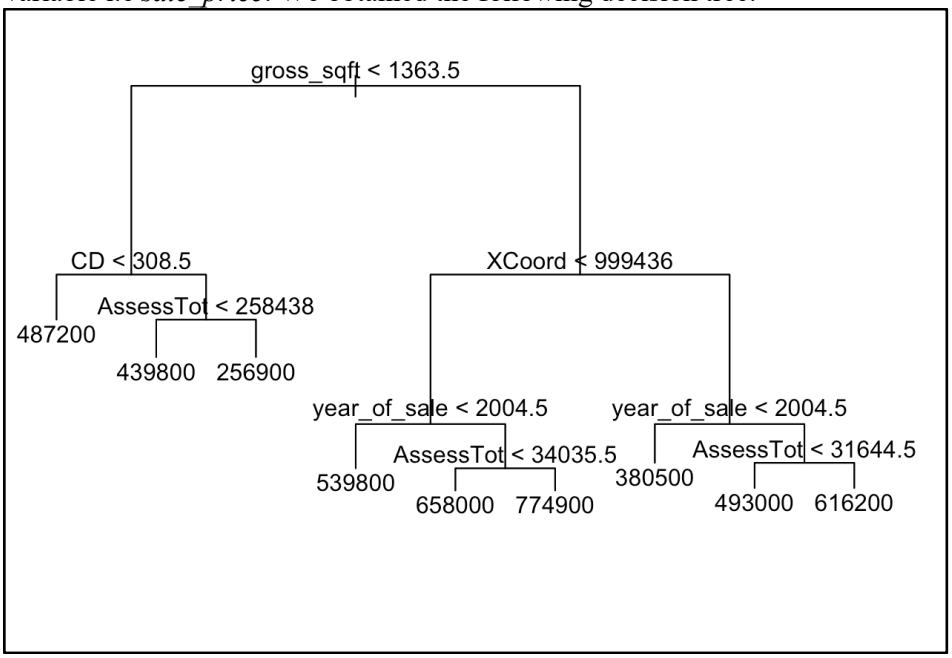
```
[1] 150542     30
```

MODEL FITTING

We started our model implementation with decision trees because they give us an important parameter of feature importance which can be used to find the most important variables of the dataset. This will help us in further reducing the number of variables while implementing less complex algorithms like multiple linear regression, lasso and ridge.

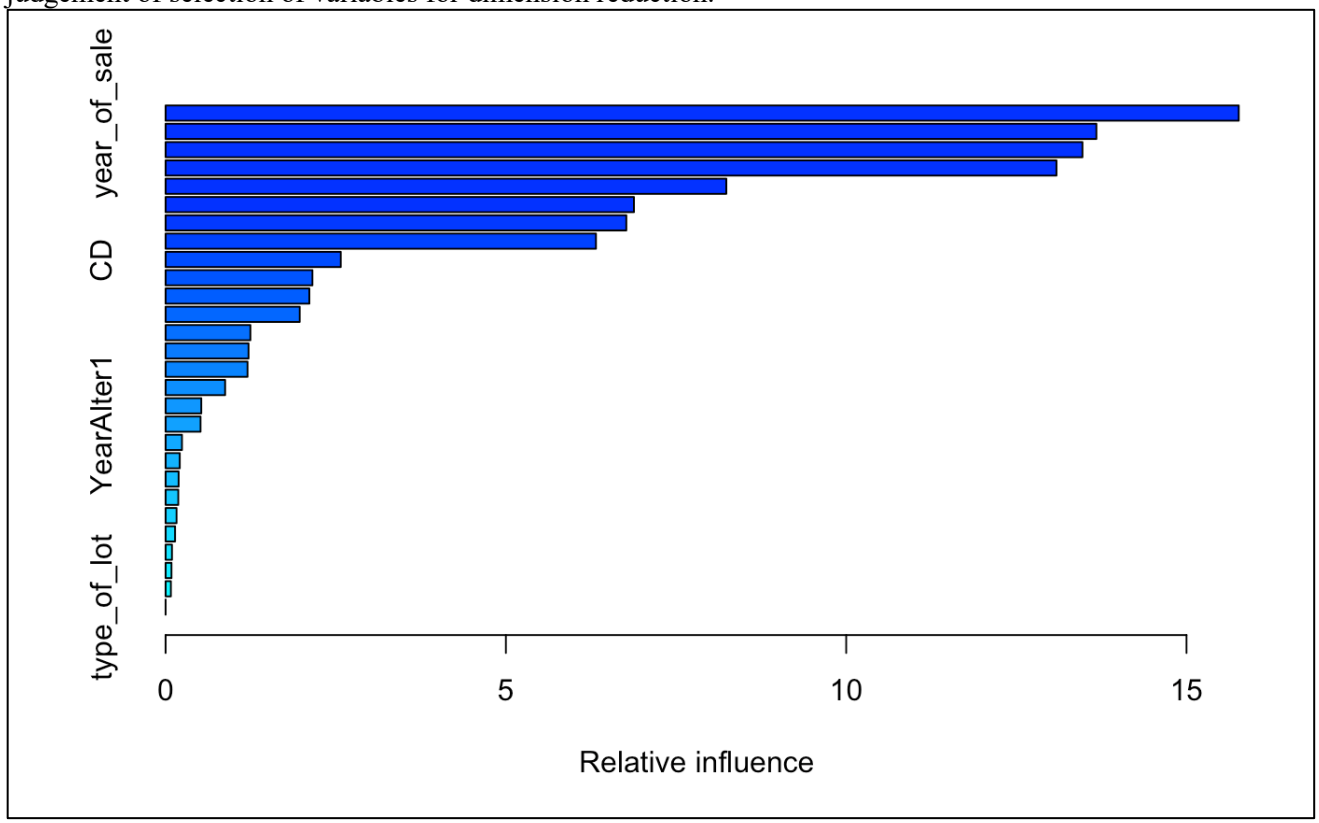
Regression Trees

Decision trees was implemented on the dataset having 150542 observations and 30 variables to predict the response variable i.e *sale_price*. We obtained the following decision tree:



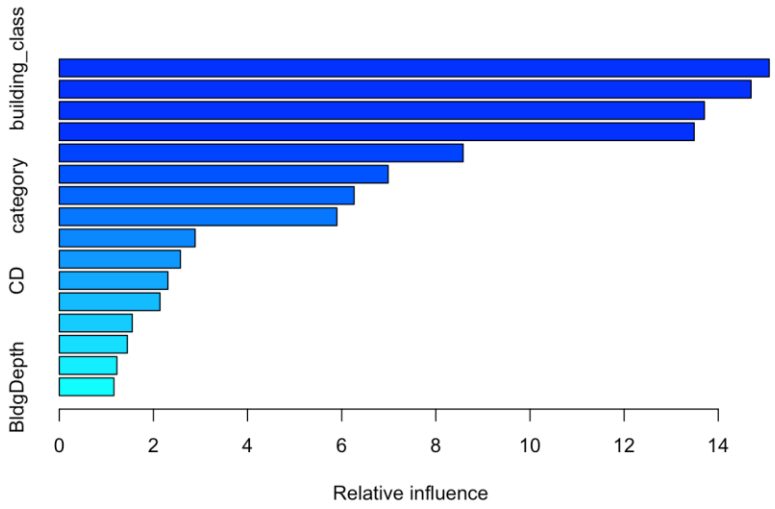
The decision tree gives us 9 different regions with the predicted sale_prices as the mean of the sale_price denoted by 9 terminal nodes.

Summary of the model also provides us with the importance of each feature, which helps us in better judgement of selection of variables for dimension reduction.



We remove all the variables whose importance was found to be less than 1%. This leaves us with only 16 predictors. We create a new object final_data2 with 150542 observations and 17 variables. This has been used for next models.

```{r}
final_data2=subset ( final_data, select = c(sale_price, building_class, category,zip_code, gross_sqft,year_built, year_of_sale, CD,
Council,ResArea, BldgFront, BldgDepth, AssessTot, ExemptTot, BuiltFAR, XCoord, YCoord) )
```



Data Splitting

We split our data in training and testing subsets to compute Mean Squared Errors and R-squared Errors for different algorithms and choose the best predictive model.

We take 70% of observations in training subset and 30% of observations in testing subset.

```{r}
set.seed(100)
train=sample(1:nrow(final_data2), 0.7*(nrow(final_data2)))
test=final_data2[-train,"sale_price"]
dim(final_data2[train,])
dim(final_data2[-train,])
```

```
[1] 105379     17
[1] 45163    17
```

Regression Trees: Boosting

The first algorithm we applied to training dataset is boosting using 5000 trees. Predicting the sale_price values for the test dataset we obtained the following mean squared error and R-squaered error.

```
Test MSE - 26418815335
R-squared - 0.5782046
```

For Boosting we used n.trees= 5000, which means that 5000 samples were sequentially generated using training dataset taking feedback from the previous tree. Final prediction was derived from the mean of the predictions from each sample. This reduces the variance and improves the prediction accuracy of the model. However, the interpretability of the model is sacrificed as compared to decision trees. Higher the number of trees, higher the accuracy.

Regression Trees: Random Forest

Next algorithm we tried on our training dataset is of Random Forest. Predicting the sale_price values for the test dataset we obtained the following mean squared error and R-squared error.

```
Test MSE -
R-squared -
```

For Random Forest we used *n.trees= 10*, which has the same implication as mentioned for boosting. We also use one more argument *m.try*, which implies the number of variables selected for fitting the tree. This reduces the effect that a strong predictor can have on our predictions as compared to a single decision tree, or in other words reduces the effect of correlation. This results in further reduction of variance and hence, improvement in accuracy as compared to a single tree.

K-Nearest Neighbors(KNN)

Next we used KNN algorithm to predict the sale prices for the test dataset .
We used 7 different values of K viz. K = 1, K=5, K=10, K=50, K= 100, K=1000.

We obtained the following R-squared values from each K value.

K= 1	K=5	K=10	K=50	K=100	K=1000
-0.04880005	0.34868700	0.38856917	0.38991741	0.38617096	0.32505479

We see that K=1 gives us a very poor result as it is highly flexible and thus overfits the training data. As the flexibility goes on decreasing the accuracy improves and we get the R-squared value for K=50. We can say that KNN with K=50 gives us the best prediction out of all othe K values. K=50 implies that the KNN predictor utilizes the 50 nearest neighbors to predict the value of the sale price. The prediction is the mean of the 50 nearest values.

Multiple Linear Regression

This is the most simple algorithm of all the algorithms that we have used. This method gives good interpretability but the accuracy is not good as compared to the regression trees. Fitting a model to the training data and predict the sale prices for the test data, we obtain the following R-squared and MSE values.

```
Residual standard error: 203300 on 105322 degrees of freedom
Multiple R-squared:  0.3353,    Adjusted R-squared:  0.3349
F-statistic: 948.6 on 56 and 105322 DF,  p-value: < 2.2e-16

[1] 0.3378145
```

Multiple regression also provides good interpretability of the data or better understanding of the direction of relation of each attribute to the sale price. The co-efficient of different attributes tell us if is positively or negatively related to the sale price and the effect of a unit change in variable on the sale price.

Following is a summary of the fitted multiple linear model.

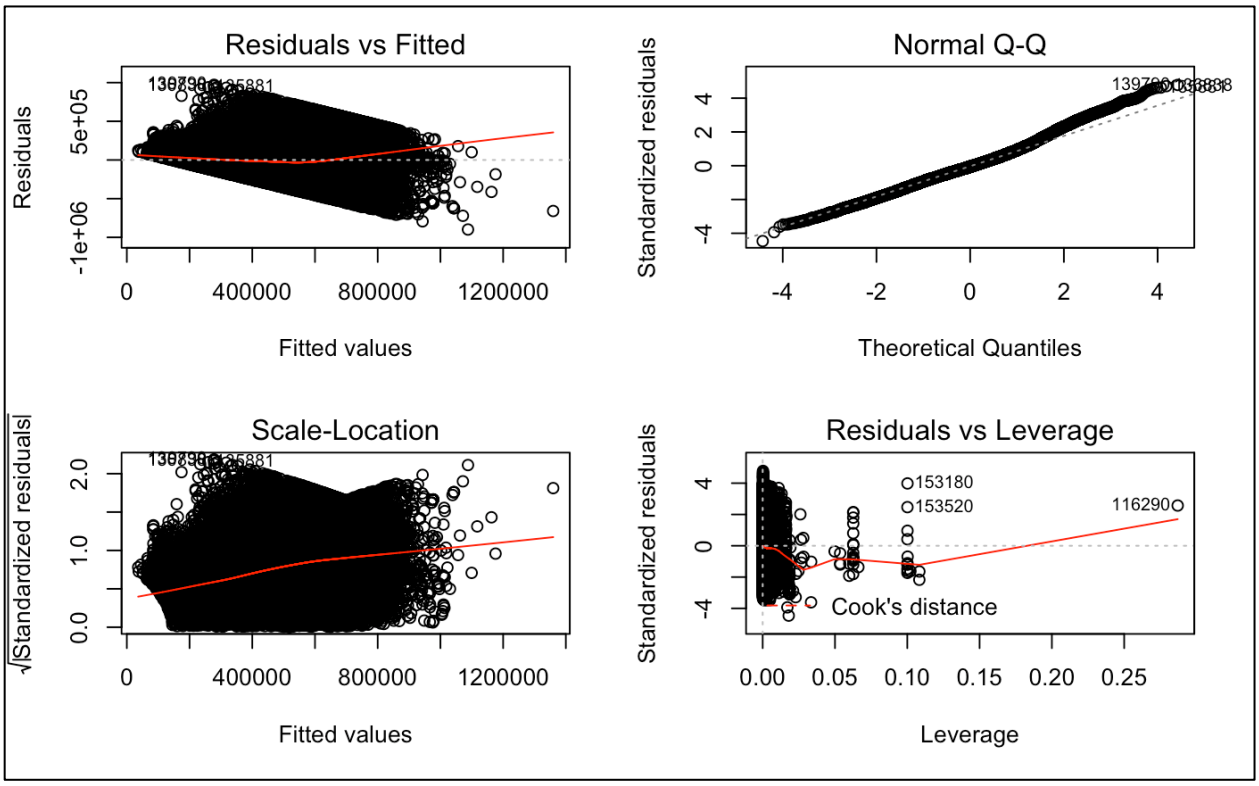
```
Call:
lm(formula = sale_price ~ ., data = final_data2[train, ])

Residuals:
   Min     1Q  Median     3Q    Max
```

```
-898887 -129073  -11685  116914  971924

Coefficients:
                                  Estimate Std. Error t value Pr(>|t|)
(Intercept)                     -2.709e+07  7.311e+05 -37.056  < 2e-16 ***
building_classA2                -1.258e+05  6.879e+03 -18.279  < 2e-16 ***
building_classA3                 1.910e+05  1.464e+04  13.049  < 2e-16 ***
building_classA4                -1.503e+04  8.354e+03  -1.799 0.072079 .
building_classA5                -1.200e+05  3.769e+03 -31.846  < 2e-16 ***
building_classA9                -1.098e+05  3.938e+03 -27.879  < 2e-16 ***
building_classB1                -4.328e+04  3.363e+03 -12.868  < 2e-16 ***
building_classB2                -5.044e+04  3.663e+03 -13.769  < 2e-16 ***
building_classB3                -6.690e+04  3.660e+03 -18.278  < 2e-16 ***
building_classB9                -4.806e+04  4.059e+03 -11.841  < 2e-16 ***
building_classC0                -1.130e+04  3.477e+03  -3.250 0.001156 **
building_classC1                 7.137e+04  6.841e+03  10.432  < 2e-16 ***
building_classC2                -2.693e+04  4.917e+03  -5.478 4.32e-08 ***
building_classC3                 8.921e+03  4.741e+03   1.882 0.059898 .
building_classC4                 1.467e+04  1.984e+04   0.739 0.459777
building_classC5                 6.024e+02  1.783e+04   0.034 0.973043
building_classC6                -2.096e+05  4.573e+03 -45.833  < 2e-16 ***
building_classC7                 1.255e+05  1.496e+04   8.390  < 2e-16 ***
building_classD1                 1.460e+04  2.164e+04   0.675 0.499874
building_classD4                -3.250e+05  5.347e+03 -60.773  < 2e-16 ***
building_classE9                -3.643e+04  1.381e+04  -2.638 0.008346 **
building_classF9                -2.187e+04  1.559e+04  -1.403 0.160664
building_classG2                -4.074e+04  1.831e+04  -2.225 0.026079 *
building_classG7                -1.609e+05  5.090e+04  -3.161 0.001572 **
building_classG9                -5.696e+04  1.528e+04  -3.727 0.000193 ***
building_classK1                -3.689e+04  1.084e+04  -3.402 0.000668 ***
building_classK2                 5.361e+04  1.578e+04   3.396 0.000683 ***
building_classK4                 1.432e+04  1.236e+04   1.159 0.246644
building_classK9                 8.987e+04  1.501e+04   5.986 2.15e-09 ***
building_classM1                -5.050e+04  2.338e+04  -2.160 0.030758 *
building_classO7                 6.539e+04  2.165e+04   3.021 0.002523 **
building_classO9                -1.321e+04  1.881e+04  -0.702 0.482602
building_classS1                -8.288e+04  7.186e+03 -11.533  < 2e-16 ***
building_classS2                -2.405e+04  5.072e+03  -4.742 2.12e-06 ***
building_classS3                -1.711e+04  1.044e+04  -1.639 0.101155
building_classS4                 2.694e+02  1.033e+04   0.026 0.979200
building_classS5                 4.927e+04  1.060e+04   4.648 3.35e-06 ***
building_classS9                 2.616e+04  9.204e+03   2.842 0.004488 **
building_classV0                -6.505e+04  2.579e+04  -2.523 0.011651 *
building_classV1                -1.781e+05  6.437e+04  -2.768 0.005648 **
categoryHigh price neighborhood  -5.569e+04  2.290e+03 -24.325  < 2e-16 ***
categorylow price neighborhood   -2.214e+05  2.240e+03 -98.848  < 2e-16 ***
categorymedium price neighborhood -1.252e+05  2.080e+03 -60.177  < 2e-16 ***
zip_code                        -2.680e+01  6.099e+01  -0.440 0.660285
gross_sqft                       4.330e-01  7.559e-02   5.728 1.02e-08 ***
year_built                      -1.512e+01  2.571e+01  -0.588 0.556658
year_of_sale                     1.478e+04  1.415e+02 104.410  < 2e-16 ***
CD                              -3.734e+03  2.034e+02 -18.352  < 2e-16 ***
Council                          1.438e+03  3.810e+02   3.773 0.000161 ***
ResArea                         -3.731e-01  2.135e-02 -17.474  < 2e-16 ***
BldgFront                       -2.539e+02  1.907e+01 -13.319  < 2e-16 ***
BldgDepth                        7.330e+01  2.115e+01   3.465 0.000531 ***
AssessTot                        1.546e-02  6.160e-04  25.096  < 2e-16 ***
ExemptTot                       -2.568e-02  2.349e-03 -10.930  < 2e-16 ***
BuiltFAR                         1.470e+04  5.747e+02  25.572  < 2e-16 ***
```

```
XCoord                -1.731e-01  3.786e-02  -4.573 4.82e-06 ***
YCoord                -1.572e+00  1.349e-01 -11.658  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```



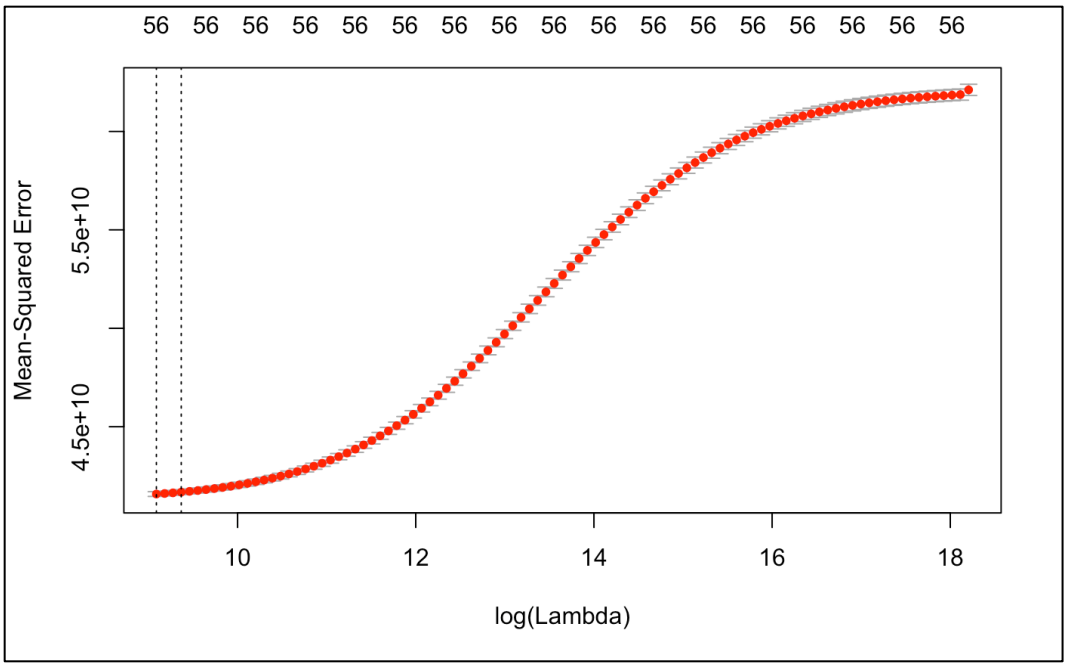
Ridge Regression

We want to improve the multiple linear regression model, by replacing the ordinary least square (OLS) fitting with some alternative fitting procedure. ¬ Two reasons for improving the OLS model are :
1.Prediction Accuracy 2.Model Interpretability

When we have a large number of *X* variables in the model there will generally be many that have little or no effect on *Y*. The model would be easier to interpret by removing the unimportant variables (i.e., setting the coefficients of those variables to zero). This can increase bias but substantially reduce the variance.

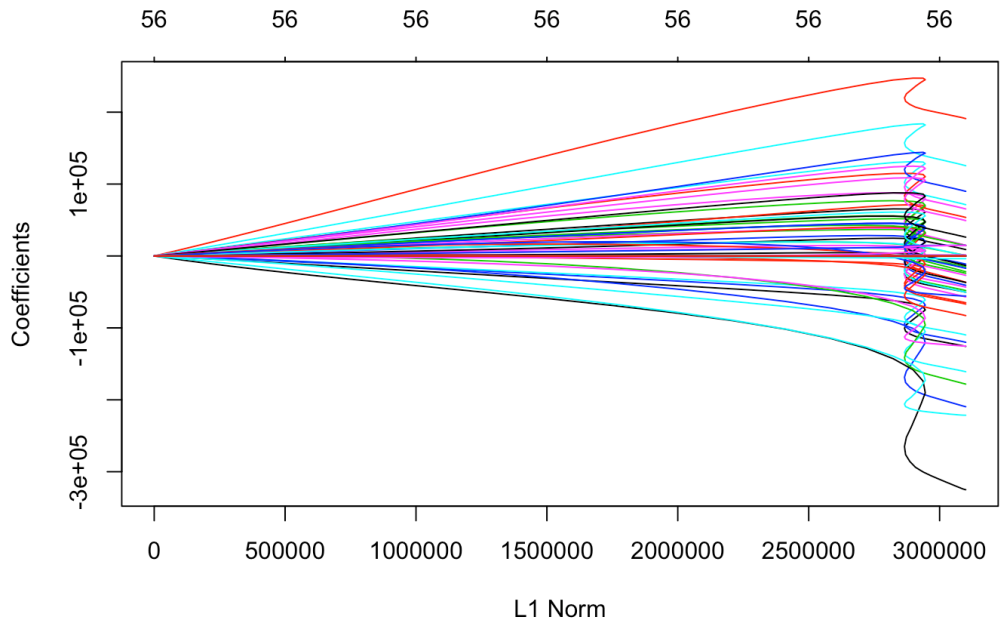
Ridge regression involves shrinking the estimates of coefficients toward zero, which in turn, reduces the variance. Applying this method to our training dataset and selecting for the best tuning parameter(lambda) we get the following plot. We get the best value of lambda as

8851.734



Predicting the sale prices using the best value of lambda for ridge regression we get the following R-squared value

R-squared : 0.3332837

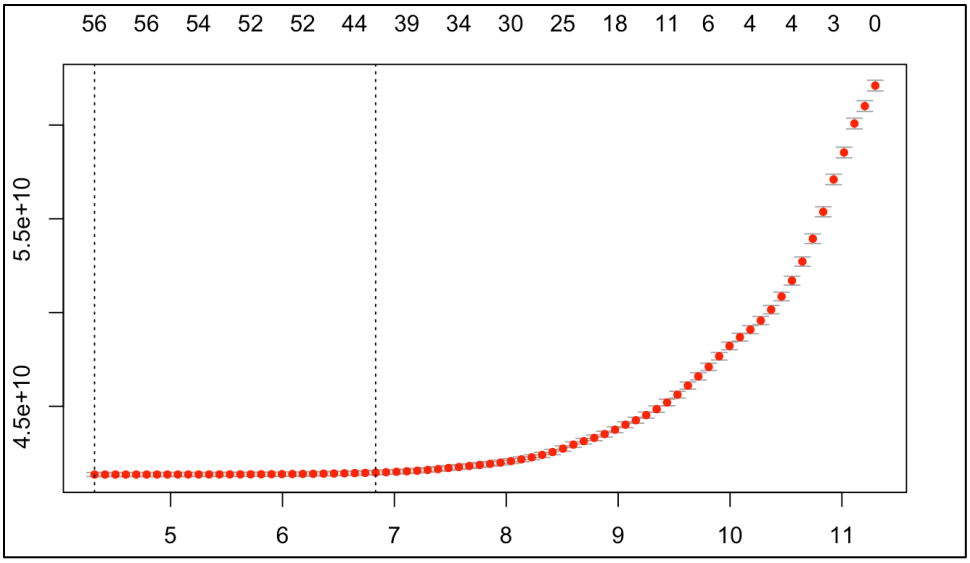


The Lasso Regression

Lasso also has the same effects on the co-efficient of the different predictors as the ridge but the ridge regression never reduces the co-efficient to be **exactly zero**. Lasso is an alternative with a solution to this problem and we can produce a model that has high predictive power and it is simple to interpret.

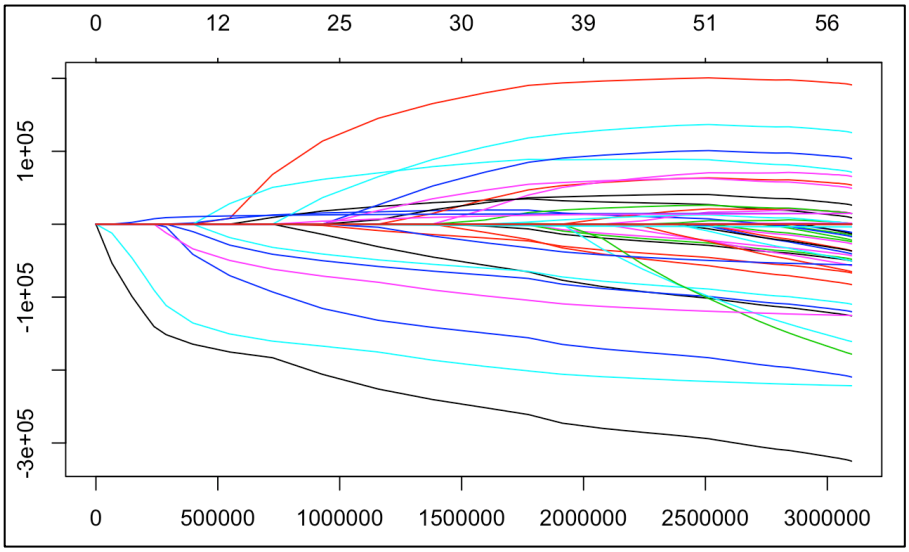
Applying Lasso method to our training dataset and selecting for the best tuning parameter(lambda) we get the following plot. We get the best value of lambda as

75.21791



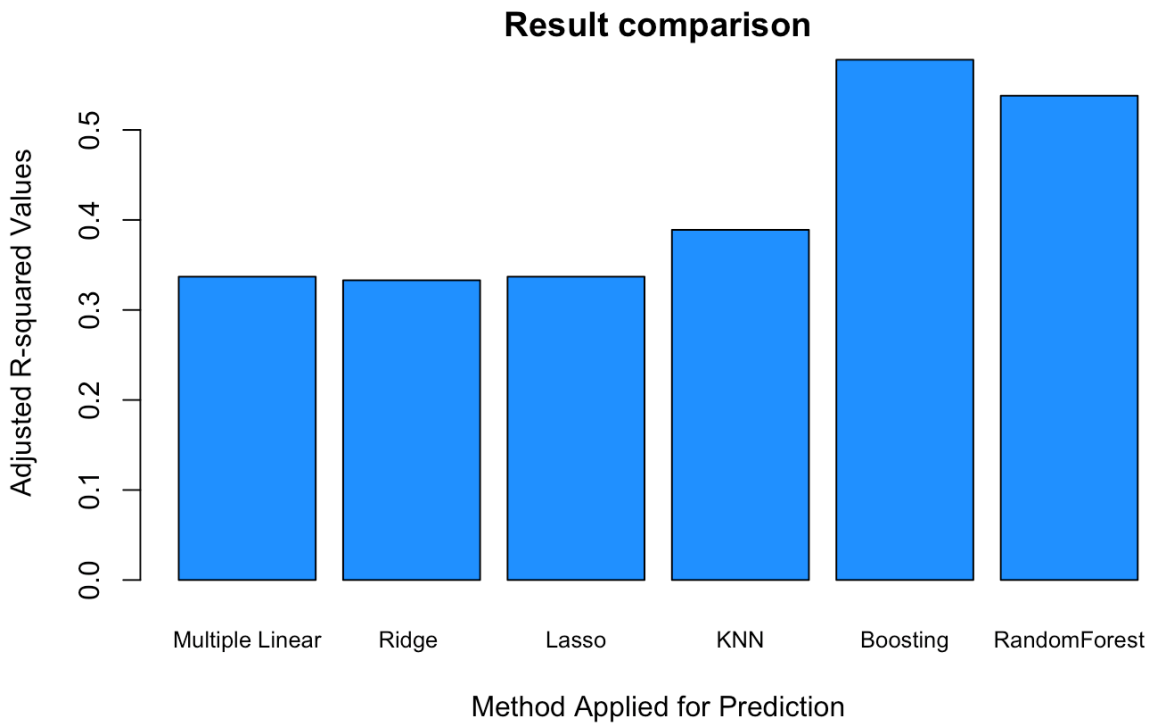
Predicting the sale prices using the best value of lambda for ridge regression we get the following R-squared value

R-squared : 0. 0.3377732



MODEL ASSESMENT AND SELECTION OF BEST MODEL

Different models can be compared using adjusted R-squared obtained from each algorithm and select the one with the maximum R-squared value. We have used adjusted R- squared as some of the models used lesser number of variables as compared to models that use all the variables.



The bar plot above shows that Boosting has given the best results out all the regression models. Random Forest was the second leading model. We can say that both boosting and Random forest gave accurate models, however this is at the cost of interpretability.

While Linear Regression, Ridge Regression and Lasso Regression were easier to interpret, they also gave least accurate results.

CONCLUSIONS

- Many irrelevant attributes were present in the original data set.
- building_class, year_of_sale and gross_sqft, were the variables that had the most impact in the overall prediction models, which seems to be intuitively true.
- Sale Price increase from 2003 to 2009, then they remain almost constant till 2013 and then they start increasing again and we can find a steady increase in sale prices till 2017.
- Sales prices are highest for building classes A3, C3, C7, V0 and V1.
- The following neighborhoods were find to be most expensive in terms of sales prices :
  BOROUGH PARK, CARROLL GARDENS, COBBLE HILL-WEST, PARK SLOPE SOUTH, SUNSET PARK, GREENPOINT, MANHATTAN BEACH, MILL BASIN, BOERUM HILL, GOWANUS , WILLIAMSBURG-NORTH, DOWNTOWN-FULTON FERRY, BENSONHURST, COBBLE HILL, DYKER HEIGHTS

REFERENCES

CATE CORCORAN, “From $320 to $908 a Square Foot: A Look Back at Housing Prices in Brooklyn 2006-2016”, Brownstoner Real Estate Market. Nov 22, 2016. https://www.brownstoner.com/real-estate-market/brooklyn-real-estate-home-prices-history-timeline/

DAVID JEANS, “Brooklyn again sets record-high sales prices. But can it continue?”. The Real Deal, New York Real Estate News. April 12, 2018. https://therealdeal.com/2018/04/12/brooklyn-again-sets-record-high-sales-prices-but-can-it-continue/

LUCY COHEN BLATTER, “Here's how much apartment prices have risen across NYC in the last 10 years”. Brick Underground. Feb 8, 2017. https://www.brickunderground.com/buy/how-prices-have-changed-over-10-years-NYC

NA ZOU, “Advanced Data Analysis Class Slides”. Industrial and Systems Engineering, Texas A&M University.

APPENDIX

Code Used:

```
#Change the path before running the code chunk
path = "C:/Users/saksh/Downloads/ISEN 613 Notes/Project"
setwd(path)
install.packages("data.table")
install.packages("bit64")
library(data.table)
library(bit64)
data = fread("brooklyn_sales_map.csv")
dim(data)
```

```
#Writing the data in dataframe format and finding the number and percentage of missing values for each attribute.
data = as.data.frame(data)
NAs = numeric()
percent_NAs = numeric()
for ( i in 1:ncol(data)){
   NAs[i] =sum(is.na(data[,i]))
   percent_NAs[i] = (NAs[i]/390883)*100
}
Missing_values= data.frame(Variable_Name = names(data), Total_Missing_Values = NAs, Percent_Values_missing
=percent_NAs)
Missing_values
```

```
data_stage2 = subset(data, select = -c(borough, Borough, UnitsRes, UnitsTotal, LotArea, BldgArea, BldgClass, Easements,
easement, OwnerType, building_class_category, ZipCode, YearBuilt, MAPPLUTO_F, PLUTOMapID, SHAPE_Leng,
SHAPE_Area, Address, EDesigNum,Version, Sanborn, ZoneMap, ZMCode, HistDist,Landmark, APPDate, FIRM07_FLA,
PFIRM15_FL, Ext, AreaSource, sale_date,ZoneDist2, ZoneDist3, ZoneDist4, Overlay1, Overlay2, SPDist1, SPDist2, SPDist3,
LtdHeight ))     #removing duplicate, irrelevant attributes and ones above 75% NAs
dim(data_stage2)   #dimensions of data_stage2
```

```
transferred_properties = which(data_stage2$sale_price == 0)       #observations with sale price = 0 removed
data_stage2 = data_stage2[-(transferred_properties),]
dim(data_stage2)
data_stage2 = subset(data_stage2, select = -c(OwnerName))      #OwnerName variable removed
str(data_stage2)                              #structure of object data_stage2
```

```
# converting necessary attributes to factors
data_stage2$neighborhood = as.factor(data_stage2$neighborhood)
data_stage2$tax_class = as.factor(data_stage2$tax_class)
data_stage2$building_class = as.factor(data_stage2$building_class)
data_stage2$address = as.factor(data_stage2$apartment_number)
data_stage2$building_class_at_sale =  as.factor(data_stage2$building_class_at_sale)
data_stage2$FireComp = as.factor(data_stage2$FireComp)
data_stage2$SanitSub = as.factor(data_stage2$SanitSub)
data_stage2$ZoneDist1 = as.factor(data_stage2$ZoneDist1)
data_stage2$SplitZone = as.factor(data_stage2$SplitZone)
data_stage2$IrrLotCode = as.factor(data_stage2$IrrLotCode)
data_stage2$apartment_number = as.factor(data_stage2$apartment_number)
str(data_stage2)
```

```
data_stage2 = subset(data_stage2, select = -c(BBL, APPBBL))         #removing BBL and APPBBL
summary(data_stage2$address)
summary(data_stage2$apartment_number)
data_stage2 = subset(data_stage2, select = -c(address, apartment_number))      #removing address & apartment number
```

```
no_NA_data = na.omit(data_stage2)          #removed observations with missing values
dim(no_NA_data)
hist(no_NA_data$sale_price)
```

```
quantile(no_NA_data$sale_price,c(0.1,0.9))    #to check top 10 and 90 percentiles
no_NA_data_trimmed = subset(no_NA_data, sale_price>142524 & sale_price<1250500)    #observations above 90 & below 10 percentiles remove
dim(no_NA_data_trimmed)
hist(no_NA_data_trimmed$sale_price) # plotting histogram for trimmed data
str(no_NA_data_trimmed)
```

```
boxplot(sale_price~ neighborhood, no_NA_data_trimmed, col = "blue")
dt = data.table(no_NA_data_trimmed)
nb_class =as.data.frame( dt[,list(mean= mean(sale_price)), by = neighborhood])
nb_class
#creating new attribute as category of neighborhood
nb_class$category = ifelse(nb_class$mean>300000 & nb_class$mean<452407.5, "low price neighborhood",ifelse
  (nb_class$mean>452407.5 & nb_class$mean<554618.1, "medium price neighborhood",ifelse
    (nb_class$mean>554618.1 & nb_class$mean<631312.3, "High price neighborhood", "Expensive Neighborhood")))
```

```
#Univariate analysis of different attributes
no_NA_data_trimmed = merge(no_NA_data_trimmed, nb_class[,c("neighborhood","category")])
boxplot(sale_price~category , no_NA_data_trimmed, col= "red")
boxplot(sale_price~tax_class, no_NA_data_trimmed
no_NA_data_trimmed$tax_class = ifelse ( no_NA_data_trimmed$tax_class %in% c("1","1A", "1B","1C"),1,
ifelse(no_NA_data_trimmed$tax_class %in% c("2","2A", "2B","2C"),2,
ifelse(no_NA_data_trimmed$tax_class ==3,3, ifelse(no_NA_data_trimmed$tax_class == 4,4,0))))
boxplot(sale_price~tax_class, no_NA_data_trimmed)
plot(no_NA_data_trimmed$block, no_NA_data_trimmed $sale_price)
plot(no_NA_data_trimmed$lot, no_NA_data_trimmed $sale_price)
no_NA_data_trimmed$type_of_lot = ifelse( no_NA_data_trimmed$lot %in% 1:999, "Traditional Tax Lots", "Condominium Unit Lots")
boxplot(sale_price~ type_of_lot, no_NA_data_trimmed)
boxplot(sale_price~building_class, no_NA_data_trimmed, col = "blue")
```

```
#checking attribute building_class
bc = data.frame(Building_Class = levels(no_NA_data_trimmed$building_class), count =
tabulate(no_NA_data_trimmed$building_class))
bc[order(bc$count),]
no_NA_data_trimmed$building_class = factor(no_NA_data_trimmed$building_class)
bc = data.frame(building_class = levels(no_NA_data_trimmed$building_class), count =
tabulate(no_NA_data_trimmed$building_class))
bc[order(bc$count),]
```

```
no_NA_data_trimmed =  merge (no_NA_data_trimmed, bc , by = "building_class")
no_NA_data_trimmed = subset(no_NA_data_trimmed, no_NA_data_trimmed$count>100)
no_NA_data_trimmed$building_class = factor (no_NA_data_trimmed$building_class)
boxplot(sale_price~building_class,no_NA_data_trimmed, col= "gold")
plot(no_NA_data_trimmed$zip_code, no_NA_data_trimmed$sale_price )
```

```
no_NA_data_trimmed =  subset(no_NA_data_trimmed , zip_code != 0)
no_NA_data_trimmed$building_class = factor(no_NA_data_trimmed$building_class)
plot(no_NA_data_trimmed$zip_code, no_NA_data_trimmed$sale_price )
plot( no_NA_data_trimmed$residential_units, no_NA_data_trimmed$sale_price)
plot( no_NA_data_trimmed$commercial_units, no_NA_data_trimmed$sale_price)
plot(no_NA_data_trimmed$total_units, no_NA_data_trimmed$sale_price)
plot(no_NA_data_trimmed$land_sqft, no_NA_data_trimmed$sale_price)
```

```
no_NA_data_trimmed =  subset(no_NA_data_trimmed, year_built > 1800 )          #removing observations before 1800s
no_NA_data_trimmed$building_class = factor(no_NA_data_trimmed$building_class)      #changing variables to factors
plot(no_NA_data_trimmed$year_built, no_NA_data_trimmed$sale_price)
boxplot(sale_price~ tax_class_at_sale, no_NA_data_trimmed)
no_NA_data_trimmed$building_class_at_sale = factor(no_NA_data_trimmed$building_class_at_sale)
boxplot(sale_price~ building_class_at_sale, no_NA_data_trimmed, col = "green")
pairs(sale_price ~ year_of_sale+CT2010+ CB2010+CD, no_NA_data_trimmed)
names(no_NA_data_trimmed)
```

```
final_data = subset ( no_NA_data_trimmed, select = c(sale_price, building_class, category, tax_class, type_of_lot,zip_code,
residential_units, gross_sqft,year_built, year_of_sale, CD, Council, LandUse,ResArea, GarageArea, StrgeArea,
NumBldgs,NumFloors, BldgFront, BldgDepth, ProxCode,IrrLotCode, LotType, AssessTot, ExemptTot, YearAlter1, YearAlter2,
BuiltFAR, XCoord, YCoord) )  #final data with 17 variables
attach(no_NA_data_trimmed)
```

```
#Decison tree
library(tree)
set.seed(1)
tree.brooklyn=tree(sale_price~. -building_class, data=final_data)
summary(tree.brooklyn)
plot(tree.brooklyn)
text(tree.brooklyn, pretty=0)
```

```
#Boosting to check variable importance
install.packages("gbm")
final_data$category=as.factor(final_data$category)
final_data$type_of_lot=as.factor(final_data$type_of_lot)
library (gbm)
 set.seed (1)
 boost.final_data =gbm(sale_price~. -tax_class,data=final_data, distribution="gaussian",n.trees =5000 ,interaction.depth =4)
 summary(boost.final_data)
#Removed the variable which are less important and obtained final data  with 17 attributes
final_data2=subset ( final_data, select = c(sale_price, building_class, category,zip_code, gross_sqft,year_built, year_of_sale, CD,
Council,ResArea, BldgFront, BldgDepth, AssessTot, ExemptTot, BuiltFAR, XCoord, YCoord) )
```

```
#Using 70% of data as training set to fit the model and test the model on the remaining 30% data
set.seed(100)
train=sample(1:nrow(final_data2), 0.7*(nrow(final_data2)))
test=final_data2[-train,"sale_price"]
dim(final_data2[train,])
dim(final_data2[-train,])
n=45163
```

```
#Fitting the boosting model
set.seed (1)
 boost.final_data2 =gbm(sale_price~.,data=final_data2[train,], distribution="gaussian",n.trees =5000 ,interaction.depth =4)
 summary(boost.final_data2)
 p=length(boost.final_data2$var.names)
#Predicting sales price from he model
predict.boost=predict(boost.final_data2,newdata = final_data2[-train ,],
n.trees =5000)
mean(( predict.boost -test)^2)
#Calculating R-Square Adjusted
actual=final_data2$sale_price
r2.gradient_boosting=1-((sum((actual[-train] - predict.boost)^2)/(n-p-1))/(sum((actual[-train] - mean(actual[-train]))^2)/(n-1)))
r2.gradient_boosting
```

```
#Fit the random forest model
final_data2$category=as.factor(final_data2$category)
library(randomForest)
set.seed(1)
bag.brooklyn=randomForest(sale_price~.,data=final_data2, subset=train, mtry=4, ntree=10, importance=TRUE)
bag.brooklyn
p=16
#Predict sales price from the model
predict.bag = predict (bag.brooklyn,newdata =final_data2[-train ,])
plot(predict.bag ,final_data2$sale_price[-train])
abline (0,1)
mean((predict.bag -final_data2$sale_price[-train])^2)
#Calulating R-Square ajusted
r2.random_forest =1-((sum((actual[-train] - predict.bag)^2)/(n-p-1))/(sum((actual[-train] - mean(actual[-train]))^2)/(n-1)))
```

```
r2.random_forest
```

```
#Fitting KNN regression
install.packages("FNN")
R2.knn = numeric()
x = model.matrix(sale_price~., final_data2)[,-1]
y= final_data2$sale_price
knn001= FNN::knn.reg(train = x[train,], test = x[-train,], y = y[train], k = 1)
knn005= FNN::knn.reg(train = x[train,], test = x[-train,], y = y[train], k = 5)
knn010= FNN::knn.reg(train = x[train,], test = x[-train,], y = y[train], k = 10)
knn050= FNN::knn.reg(train = x[train,], test = x[-train,], y = y[train], k = 50)
knn060= FNN::knn.reg(train = x[train,], test = x[-train,], y = y[train], k = 60)
knn100= FNN::knn.reg(train = x[train,], test = x[-train,], y = y[train], k = 100)
knn1000= FNN::knn.reg(train = x[train,], test = x[-train,], y = y[train], k = 1000)
#Calculating R-square adjusted
R2.knn[1] =1 - ((sum((actual[-train] - knn001$pred)^2)/(n-p-1))/(sum((actual[-train] - mean(actual[-train]))^2)/(n-1)))
R2.knn[2] =1 - ((sum((actual[-train] - knn005$pred)^2)/(n-p-1))/(sum((actual[-train] - mean(actual[-train]))^2)/(n-1)))
R2.knn[3] =1 - ((sum((actual[-train] - knn010$pred)^2)/(n-p-1))/(sum((actual[-train] - mean(actual[-train]))^2)/(n-1)))
R2.knn[4] =1 - ((sum((actual[-train] - knn050$pred)^2)/(n-p-1))/(sum((actual[-train] - mean(actual[-train]))^2)/(n-1)))
R2.knn[5] =1 - ((sum((actual[-train] - knn060$pred)^2)/(n-p-1))/(sum((actual[-train] - mean(actual[-train]))^2)/(n-1)))
R2.knn[6] =1 - ((sum((actual[-train] - knn100$pred)^2)/(n-p-1))/(sum((actual[-train] - mean(actual[-train]))^2)/(n-1)))
R2.knn[7] =1 - ((sum((actual[-train] - knn1000$pred)^2)/(n-p-1))/(sum((actual[-train] - mean(actual[-train]))^2)/(n-1)))
R2.knn
```

```
#Fit the  multiple linear regression model
install.packages("boot")
library(boot)
set.seed(10)
lm.fit2=lm(sale_price~.,data=final_data2[train,])
summary(lm.fit2)
#Predict the model
p=16
predict.lm2=predict(lm.fit2 , final_data2[-train,])
r2.lm=1-((sum((actual[-train] - predict.lm2)^2)/(n-p-1))/(sum((actual[-train] - mean(actual[-train]))^2)/(n-1)))
r2.lm
```

```
library(glmnet)
grid = 10^seq(10,-2, length = 100)
#fitting ridge  regression model
ridge.sale_price = glmnet (x[train,],y[train],alpha= 0 , lambda = grid)
coef(ridge.sale_price)
p=12
#selecting best value of lambda for ridge
set.seed(1)
cv.out = cv.glmnet(x[train,],y[train], alpha = 0)
plot(cv.out)
bestlam.ridge = cv.out$lambda.min
bestlam.ridge
#Predict sales price using ridge regression
ridge.pred = predict (ridge.sale_price,s = bestlam.ridge, newx = x[-train,])
mean((ridge.pred - y[-train])^2)
#Calulating R-Square adjusted
R2.ridge =1-((sum((actual[-train] - ridge.pred)^2)/(n-p-1))/(sum((actual[-train] - mean(actual[-train]))^2)/(n-1)))
R2.ridge
```

```
#fitting the lasso regression
lasso.sale_price = glmnet(x[train,], y[train], alpha = 1, lambda = grid)
names(lasso.sale_price)
#selecting best value of lambda for lasso
set.seed(1)
cv.out = cv.glmnet(x[train,],y[train], alpha = 1)
plot(cv.out)
bestlam.lasso = cv.out$lambda.min
```

```
bestlam.lasso
#Predict sales price using lasso regression
lasso.pred = predict (lasso.sale_price, s = bestlam.lasso, newx = x[-train,])
mean((lasso.pred - y[-train])^2)
#Calculating R-Square adjusted
R2.lasso = 1-((sum((actual[-train] - lasso.pred)^2)/(n-p-1))/(sum((actual[-train] - mean(actual[-train]))^2)/(n-1)))
R2.lasso
```

```
Result_Comparison = data.frame( "Method" = c("Multiple Linear", "Ridge ", "Lasso", "KNN", "Boosting", "RandomForest"),
"Adjusted_R-squared_Values"= c(r2.lm,R2.ridge,R2.lasso,R2.knn[4],r2.gradient_boosting,r2.random_forest))  #Comparing
results

barplot(Result_Comparison$Adjusted_R.squared_Values, col= "dodgerblue", names.arg = Result_Comparison$Method, main =
"Result comparison", cex.names = 0.8, ylab = "Adjusted R-squared Values",xlab="Method applied for prediction") #Barplot
```