HarvardX Capstone Project: Prediction of New York City House Sale Prices

Goodarz Ghanavati

12/13/2020

Contents

1 Introduction

This report describes the method and the code developed for the HarvardX Capstone project. The project goal is to predict housing prices in New York City (NYC). The project dataset available on Kaggle includes the records of one year of building or building unit sales in NYC. It includes various data about each property including the sale price, zip code, block, gross square feet, address, building type, etc. Data exploration and visualization shows that sale price and several predictors, e.g., gross square feet, are correlated. Data cleaning was performed to remove missing values and data that is not helpful for prediction of sale prices as well as mitigating the impact of the skewness of data. Then, the dataset was split into training and test sets. Random Forest and linear regression models trained using the training set were used to predict the sale prices of the test set. The performance of the two models are compared with each other.

2 Method and Analysis

This section explains the process and techniques used in the project.

2.1 Loading Libraries

The first step is to install if needed and import the libraries used in the project.

```
# Install required libraries
if(!require(randomForest)) install.packages("randomForest",
                                            repos = "http://cran.us.r-project.org")
if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret",
                                     repos = "http://cran.us.r-project.org")
if(!require(corrplot)) install.packages("corrplot",
                                        repos = "http://cran.us.r-project.org")
if(!require(e1071)) install.packages("e1071",
                                     repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate",
                                         repos = "http://cran.us.r-project.org")

library(randomForest)
library(tidyverse)
library(caret)
library(corrplot)
library(e1071)
library(lubridate)
```

2.2 Options for Running the Code

By setting is_tune to 1 the code will tune the Random Forest model. If is_tune is 0, the code will train the Random Forest model using the mtry_input parameter. As discussed in the results section, tuning of the Random Forest algorithm for this application takes about 8 hours. Training the Random Forest model with the optimal parameter (mtry=6) takes about 2 hours. The user can set mtry to lower values for a faster run but the model will not be optimal. With mtry of 2, the Random Forest model will be trained within a few minutes.

```
## Options for running the code
# is_tune    1: Tune parameters in the Random Forest model
#            2: Don't tune. Train the Random Forest model using mtry_input
is_tune = 0
# mtry_input sets the mtry parameter for the random forest algorithm:
# the user can select a value from 2 to 6. The optimal value is 6.
# The user can set mtry to lower values for a faster run but the
# model will not be optimal
mtry_input = 6
```

2.3 Download the Dataset

The NYC Property Sales dataset is available on Kaggle [1]. I have copied the dataset to my Github account.

The following command saves the start time of the run in the start_time variable.

```
# Start time of run
start_time <- Sys.time()
```

The following piece of code downloads the dataset from Github and reads the csv file containing the data:

```
# Download dataset from Github and read the csv file
download.file("https://github.com/ghanag/HarvardX-Capstone-Project-NYC-House-Price-Forecast/raw/master/
              "./nyc-rolling-sales.csv")
nyc_house_data <- read.csv("nyc-rolling-sales.csv")
```

The following code shows the dimension, column names and the first few rows of the dataset:

```
# Dimension, Column Names and first few rows of the dataset
dim(nyc_house_data)
```

```
## [1] 83982    22
```

```
names(nyc_house_data)
```

```
##  [1] "X"                                 "BOROUGH"
##  [3] "NEIGHBORHOOD"                      "BUILDING.CLASS.CATEGORY"
##  [5] "TAX.CLASS.AT.PRESENT"              "BLOCK"
##  [7] "LOT"                               "EASE.MENT"
##  [9] "BUILDING.CLASS.AT.PRESENT"         "ADDRESS"
## [11] "APARTMENT.NUMBER"                  "ZIP.CODE"
## [13] "RESIDENTIAL.UNITS"                 "COMMERCIAL.UNITS"
## [15] "TOTAL.UNITS"                       "LAND.SQUARE.FEET"
## [17] "GROSS.SQUARE.FEET"                 "YEAR.BUILT"
## [19] "TAX.CLASS.AT.TIME.OF.SALE"         "BUILDING.CLASS.AT.TIME.OF.SALE"
## [21] "SALE.PRICE"                        "SALE.DATE"
```

```
head(nyc_house_data)
```

```
##   X BOROUGH  NEIGHBORHOOD                    BUILDING.CLASS.CATEGORY
## 1 4       1 ALPHABET CITY 07 RENTALS - WALKUP APARTMENTS
## 2 5       1 ALPHABET CITY 07 RENTALS - WALKUP APARTMENTS
## 3 6       1 ALPHABET CITY 07 RENTALS - WALKUP APARTMENTS
## 4 7       1 ALPHABET CITY 07 RENTALS - WALKUP APARTMENTS
## 5 8       1 ALPHABET CITY 07 RENTALS - WALKUP APARTMENTS
## 6 9       1 ALPHABET CITY 07 RENTALS - WALKUP APARTMENTS
##   TAX.CLASS.AT.PRESENT BLOCK LOT EASE.MENT BUILDING.CLASS.AT.PRESENT
## 1                   2A   392   6        NA                        C2
## 2                    2   399  26        NA                        C7
## 3                    2   399  39        NA                        C7
## 4                   2B   402  21        NA                        C4
## 5                   2A   404  55        NA                        C2
## 6                    2   405  16        NA                        C4
##                 ADDRESS APARTMENT.NUMBER ZIP.CODE RESIDENTIAL.UNITS
## 1           153 AVENUE B                    10009                 5
## 2  234 EAST 4TH   STREET                    10009                28
## 3  197 EAST 3RD   STREET                    10009                16
## 4    154 EAST 7TH STREET                    10009                10
## 5 301 EAST 10TH   STREET                    10009                 6
## 6 516 EAST 12TH   STREET                    10009                20
##   COMMERCIAL.UNITS TOTAL.UNITS LAND.SQUARE.FEET GROSS.SQUARE.FEET YEAR.BUILT
## 1                0           5             1633              6440       1900
## 2                3          31             4616             18690       1900
## 3                1          17             2212              7803       1900
## 4                0          10             2272              6794       1913
## 5                0           6             2369              4615       1900
## 6                0          20             2581              9730       1900
##   TAX.CLASS.AT.TIME.OF.SALE BUILDING.CLASS.AT.TIME.OF.SALE SALE.PRICE
## 1                         2                             C2    6625000
## 2                         2                             C7          -
## 3                         2                             C7          -
## 4                         2                             C4    3936272
```

```
## 5                         2                              C2    8000000
## 6                         2                              C4          -
##             SALE.DATE
## 1 2017-07-19 00:00:00
## 2 2016-12-14 00:00:00
## 3 2016-12-09 00:00:00
## 4 2016-09-23 00:00:00
## 5 2016-11-17 00:00:00
## 6 2017-07-20 00:00:00
```

2.4 Data Cleaning

Examining the dataset shows that some of the columns in the dataset are not helpful for prediction of sale prices. Column X appears to be an ID for each group of transactions, all values in the EASE.MENT column are null. Apartment number is not useful for predicting the price either. Also, although there is some correlation between an apartment address/street and its price, due to the large number of addresses/streets it does not appear that it can be a useful input to a machine learning algorithm. Also, there are several other predictors in the data set, e.g., borough, zip code and neighborhood that are related to the location of a property. They can be used instead of the address as a predictor.

The following code removes the following columns: X, EASE.MENT, APARTMENT.NUMBER and ADDRESS.

```
# Delete Easement Column since all Eastment values are NA and delete column X
# Apartment number and address don't appear to be useful for predicting the price
nyc_house_data <- subset(nyc_house_data, select = -c(EASE.MENT, X,
                                                     APARTMENT.NUMBER, ADDRESS))
```

Checking for duplicate rows shows that there are several identical rows. The following code checks and removes duplicate rows.

```
# Check for duplicate rows and remove them
sum(duplicated(nyc_house_data))
```

```
## [1] 949
```

```
nyc_house_data <- unique(nyc_house_data)
```

Checking the sale price column shows that it is in the character format. Also, there are entries with no ("-") sale price that need to be removed.

```
## CLEANING SALE PRICE COLUMN

# Many rows don't have a sale price
sum(nyc_house_data$SALE.PRICE == " -  ")
```

```
## [1] 13892
```

```
# Removing Rows that don't have a Sale Price
nyc_house_data <- nyc_house_data[!nyc_house_data$SALE.PRICE == " -  ",]
# Change the Sale Price Variable class from Character to Numeric
nyc_house_data$SALE.PRICE <- as.numeric(nyc_house_data$SALE.PRICE)
```

Now, let's look at the sale price values. Some entries are zero. These sales are typically transfer of deeds between parties [1]. For example, parents may transfer the ownership to their children. Some entries have small dollar amounts. They don't seem to be right for properties in New York City. For example, it is not reasonable that price of a condo in Manhattan to be 70,000$. Here, all entries with a sale price less than 250,000$ are removed. On the other hand, properties with very high sale prices in dataset are office buildings, apartment complexes with many units and large garages. These entries seem correct so they are kept.

```
## Check Sale Price Values
# Summary of sale prices
summary(nyc_house_data$SALE.PRICE)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.      NA's
## 0.000e+00 2.300e+05 5.346e+05 1.286e+06 9.500e+05 2.210e+09         1
```

```
# Removing houses with Sale Price less than 250,000$
nyc_house_data <- subset(nyc_house_data,SALE.PRICE > 250000)
# Check Properties with highest Sale Prices
head(nyc_house_data[order(-nyc_house_data$SALE.PRICE),])
```

```
##       BOROUGH          NEIGHBORHOOD                      BUILDING.CLASS.CATEGORY
## 7448        1           MIDTOWN CBD 21 OFFICE BUILDINGS
## 2560        1             FINANCIAL 21 OFFICE BUILDINGS
## 2558        1             FINANCIAL 21 OFFICE BUILDINGS
## 6333        1              KIPS BAY 08 RENTALS - ELEVATOR APARTMENTS
## 2051        1               FASHION 21 OFFICE BUILDINGS
## 35390       3 DOWNTOWN-FULTON FERRY 29 COMMERCIAL GARAGES
##       TAX.CLASS.AT.PRESENT BLOCK LOT BUILDING.CLASS.AT.PRESENT ZIP.CODE
## 7448                     4  1301   1                         O4    10167
## 2560                     4    40   3                         O4    10005
## 2558                     4    29   1                         O4    10004
## 6333                     2   934   1                         D6    10016
## 2051                     4   833  11                         O4    10001
## 35390                    4    54   1                         G7    11201
##       RESIDENTIAL.UNITS COMMERCIAL.UNITS TOTAL.UNITS LAND.SQUARE.FEET
## 7448                  0               35          35            81336
## 2560                  0                1           1            53632
## 2558                  0                1           1            42762
## 6333                894                8         902           141836
## 2051                  0               55          55            30750
## 35390                 0                0           0           134988
##       GROSS.SQUARE.FEET YEAR.BUILT TAX.CLASS.AT.TIME.OF.SALE
## 7448            1586886       1966                         4
## 2560            1617206       1987                         4
## 2558             993569       1983                         4
## 6333             829024       1975                         2
## 2051             645977       1969                         4
## 35390                 0          0                         4
##       BUILDING.CLASS.AT.TIME.OF.SALE SALE.PRICE          SALE.DATE
## 7448                              O4   2.21e+09 2017-05-05 00:00:00
## 2560                              O4   1.04e+09 2017-01-24 00:00:00
## 2558                              O4   6.52e+08 2017-05-24 00:00:00
## 6333                              D6   6.20e+08 2016-12-08 00:00:00
## 2051                              O4   5.65e+08 2016-11-01 00:00:00
## 35390                             G7   3.45e+08 2016-12-20 00:00:00
```

Examining the entries of Gross Square Feet and Land Square Feet shows that for certain building types, e.g., housing co-op, both Gross Square Feet and Land Square Feet entries are 0 or blank. Also, for certain properties that don't include buildings, e.g., garages, Gross Square Feet is entered 0 while Land Square Feet is not zero. Due to these reasons, the entries with Gross Square Feet or Land Square Feet of 0 are kept. The following code examines the blank entries of Gross Square Feet and Land Square feet. It also changes the blank ("-") values of Gross Square Feet and Land Square Feet to 0 and converts them to numeric variables.

```r
# Check the entries with Gross square feet equal to -
sum(nyc_house_data$GROSS.SQUARE.FEET == " -  ")
```

```
## [1] 18177
```

```r
head(nyc_house_data[nyc_house_data$GROSS.SQUARE.FEET == " -  ",])
```

```
##    BOROUGH  NEIGHBORHOOD                    BUILDING.CLASS.CATEGORY
## 14       1 ALPHABET CITY 09 COOPS - WALKUP APARTMENTS
## 16       1 ALPHABET CITY 09 COOPS - WALKUP APARTMENTS
## 17       1 ALPHABET CITY 09 COOPS - WALKUP APARTMENTS
## 18       1 ALPHABET CITY 09 COOPS - WALKUP APARTMENTS
## 19       1 ALPHABET CITY 09 COOPS - WALKUP APARTMENTS
## 20       1 ALPHABET CITY 09 COOPS - WALKUP APARTMENTS
##    TAX.CLASS.AT.PRESENT BLOCK LOT BUILDING.CLASS.AT.PRESENT ZIP.CODE
## 14                    2   373  40                       C6    10009
## 16                    2   373  40                       C6    10009
## 17                    2   373  40                       C6    10009
## 18                    2   373  46                       C6    10009
## 19                    2   373  49                       C6    10009
## 20                    2   373  49                       C6    10009
##    RESIDENTIAL.UNITS COMMERCIAL.UNITS TOTAL.UNITS LAND.SQUARE.FEET
## 14                 0                0           0              -
## 16                 0                0           0              -
## 17                 0                0           0              -
## 18                 0                0           0              -
## 19                 0                0           0              -
## 20                 0                0           0              -
##    GROSS.SQUARE.FEET YEAR.BUILT TAX.CLASS.AT.TIME.OF.SALE
## 14                 -       1920                        2
## 16                 -       1920                        2
## 17                 -       1920                        2
## 18                 -       1925                        2
## 19                 -       1920                        2
## 20                 -       1920                        2
##    BUILDING.CLASS.AT.TIME.OF.SALE SALE.PRICE           SALE.DATE
## 14                             C6     499000 2017-03-10 00:00:00
## 16                             C6     529500 2017-06-09 00:00:00
## 17                             C6     423000 2017-07-14 00:00:00
## 18                             C6     501000 2017-03-16 00:00:00
## 19                             C6     450000 2016-09-01 00:00:00
## 20                             C6     510000 2017-08-17 00:00:00
```

```r
# Check the entries with Gross square feet and Land Square Feet equal to -
sum(nyc_house_data$GROSS.SQUARE.FEET == " -  " &
      nyc_house_data$LAND.SQUARE.FEET == " -  ")
```

```
## [1] 17800
```

```r
head(nyc_house_data[nyc_house_data$GROSS.SQUARE.FEET == " -  " &
                      nyc_house_data$LAND.SQUARE.FEET != " -  " ,])
```

```
##      BOROUGH  NEIGHBORHOOD                      BUILDING.CLASS.CATEGORY
## 76         1 ALPHABET CITY 11A CONDO-RENTALS
## 958        1       CHELSEA  29 COMMERCIAL GARAGES
## 959        1       CHELSEA  29 COMMERCIAL GARAGES
```

```
## 1424       1  CIVIC CENTER  22 STORE BUILDINGS
## 1782       1       CLINTON  31 COMMERCIAL VACANT LAND
## 1959       1  EAST VILLAGE  31 COMMERCIAL VACANT LAND
##      TAX.CLASS.AT.PRESENT BLOCK  LOT BUILDING.CLASS.AT.PRESENT ZIP.CODE
## 76                      2   397 1301                       RR    10002
## 958                     4   799   70                       G6    10011
## 959                     4   803   62                       G6    10001
## 1424                    4   133   13                       K9    10007
## 1782                    4  1064    9                       V1    10019
## 1959                    4   463   33                       V1    10003
##      RESIDENTIAL.UNITS COMMERCIAL.UNITS TOTAL.UNITS LAND.SQUARE.FEET
## 76                 132                0         132            33650
## 958                  0                0           0             2125
## 959                  0                0           0             7571
## 1424                 0                2           2             2188
## 1782                 0                0           0             2750
## 1959                 0                0           0             2500
##      GROSS.SQUARE.FEET YEAR.BUILT TAX.CLASS.AT.TIME.OF.SALE
## 76                   -       1989                        2
## 958                  -       1111                        4
## 959                  -          0                        4
## 1424                 -       1900                        4
## 1782                 -          0                        4
## 1959                 -          0                        4
##      BUILDING.CLASS.AT.TIME.OF.SALE SALE.PRICE           SALE.DATE
## 76                               RR   52625000 2016-10-19 00:00:00
## 958                              G6    8208750 2017-04-21 00:00:00
## 959                              G6   60000000 2016-10-07 00:00:00
## 1424                             K9   25000000 2017-04-05 00:00:00
## 1782                             V1   23000000 2017-05-22 00:00:00
## 1959                             V1    6000000 2016-09-23 00:00:00
```

```r
# Convert blank ("-") values of Gross Square Feet and Land Square Feet to 0
nyc_house_data[nyc_house_data$GROSS.SQUARE.FEET == " -  ",
               names(nyc_house_data) == "GROSS.SQUARE.FEET"] = "0"
nyc_house_data[nyc_house_data$LAND.SQUARE.FEET == " -  ",
               names(nyc_house_data) == "LAND.SQUARE.FEET"] = "0"
# Converting Gross Square Feet and Land Square Feet to Numerical variables
nyc_house_data$GROSS.SQUARE.FEET <- as.numeric(nyc_house_data$GROSS.SQUARE.FEET)
nyc_house_data$LAND.SQUARE.FEET  <- as.numeric(nyc_house_data$LAND.SQUARE.FEET)
```

Examining the Year Built and Zip Code columns shows that some of the entries are 0. These rows are removed from the dataset.

```r
# Remove rows that have a year built equal to 0
nyc_house_data <- nyc_house_data[!nyc_house_data$YEAR.BUILT == 0, ]
# Remove rows that have a zip code equal to 0
nyc_house_data <- nyc_house_data[!nyc_house_data$ZIP.CODE == 0, ]
```

Examining the residential, commercial and total unit columns shows that in some rows sum of residential and commercial units is not equal to total units. Also, in some rows both residential and commercial units are 0. These rows don't seem correct. Hence, they are removed from the dataset.

```r
# Remove entries with both Residential and Commerical units equal to 0
nyc_house_data <- filter(nyc_house_data, RESIDENTIAL.UNITS != 0 |
                           COMMERCIAL.UNITS != 0)
```

```
# Number of entries with sum of Residential and Commerical units not equal to Total units
sum(nyc_house_data$RESIDENTIAL.UNITS + nyc_house_data$COMMERCIAL.UNITS
    != nyc_house_data$TOTAL.UNITS)
```

```
## [1] 22
```

```
# Remove entries with sum of Residential and Commerical units not equal to Total units
nyc_house_data <- filter(nyc_house_data, RESIDENTIAL.UNITS+COMMERCIAL.UNITS == TOTAL.UNITS)
```

Examining the Sale Date column shows that time is 00:00:00 for all entries. The following code removes time from Sale Date. It then Convert dates to numeric values.

```
# Time is 00:00:00 for all entries in SALE.DATE. Remove time. Convert dates
# to numeric values.
nyc_house_data <- nyc_house_data %>%
  separate(col = "SALE.DATE", into = c("SALE.DATE", "SALE.TIME"), sep = " ")
nyc_house_data <- subset(nyc_house_data, select = -c(SALE.TIME))
nyc_house_data$SALE.DATE <- as.numeric(ymd(nyc_house_data$SALE.DATE))
```

Now, let's check if there is any NA left in the dataset. The RandomForest algorithm cannot handle NAs.

```
# Check if there is any NA left in the dataset
colSums(is.na(nyc_house_data))
```

```
##                         BOROUGH                    NEIGHBORHOOD
##                               0                               0
##         BUILDING.CLASS.CATEGORY            TAX.CLASS.AT.PRESENT
##                               0                               0
##                           BLOCK                             LOT
##                               0                               0
##       BUILDING.CLASS.AT.PRESENT                        ZIP.CODE
##                               0                               0
##               RESIDENTIAL.UNITS                COMMERCIAL.UNITS
##                               0                               0
##                     TOTAL.UNITS                LAND.SQUARE.FEET
##                               0                               0
##               GROSS.SQUARE.FEET                       YEAR.BUILT
##                               0                               0
##      TAX.CLASS.AT.TIME.OF.SALE BUILDING.CLASS.AT.TIME.OF.SALE
##                               0                               0
##                      SALE.PRICE                       SALE.DATE
##                               0                               0
```

All NA value are removed from the dataset.

The following code converts borough, neighborhood, building & tax class category variables to factors so they can be used as an input to the Random Forest model.

```
# Converting Borough, neighborhood, building & tax class category variables to factor
nyc_house_data$BOROUGH <- as.factor(nyc_house_data$BOROUGH)
nyc_house_data$BUILDING.CLASS.CATEGORY <-
  as.factor(nyc_house_data$BUILDING.CLASS.CATEGORY)
nyc_house_data$TAX.CLASS.AT.PRESENT <-
  as.factor(nyc_house_data$TAX.CLASS.AT.PRESENT)
nyc_house_data$TAX.CLASS.AT.TIME.OF.SALE <-
  as.factor(nyc_house_data$TAX.CLASS.AT.TIME.OF.SALE)
nyc_house_data$NEIGHBORHOOD <-
  as.factor(nyc_house_data$NEIGHBORHOOD)
```

```
nyc_house_data$BUILDING.CLASS.AT.TIME.OF.SALE <-
  as.factor(nyc_house_data$BUILDING.CLASS.AT.TIME.OF.SALE)
nyc_house_data$BUILDING.CLASS.AT.PRESENT <-
  as.factor(nyc_house_data$BUILDING.CLASS.AT.PRESENT)
```

The following code shows the dimension of the dataset at the end of data cleaning:

```
# Dimension of the dataset
dim(nyc_house_data)
```

```
## [1] 35953    18
```

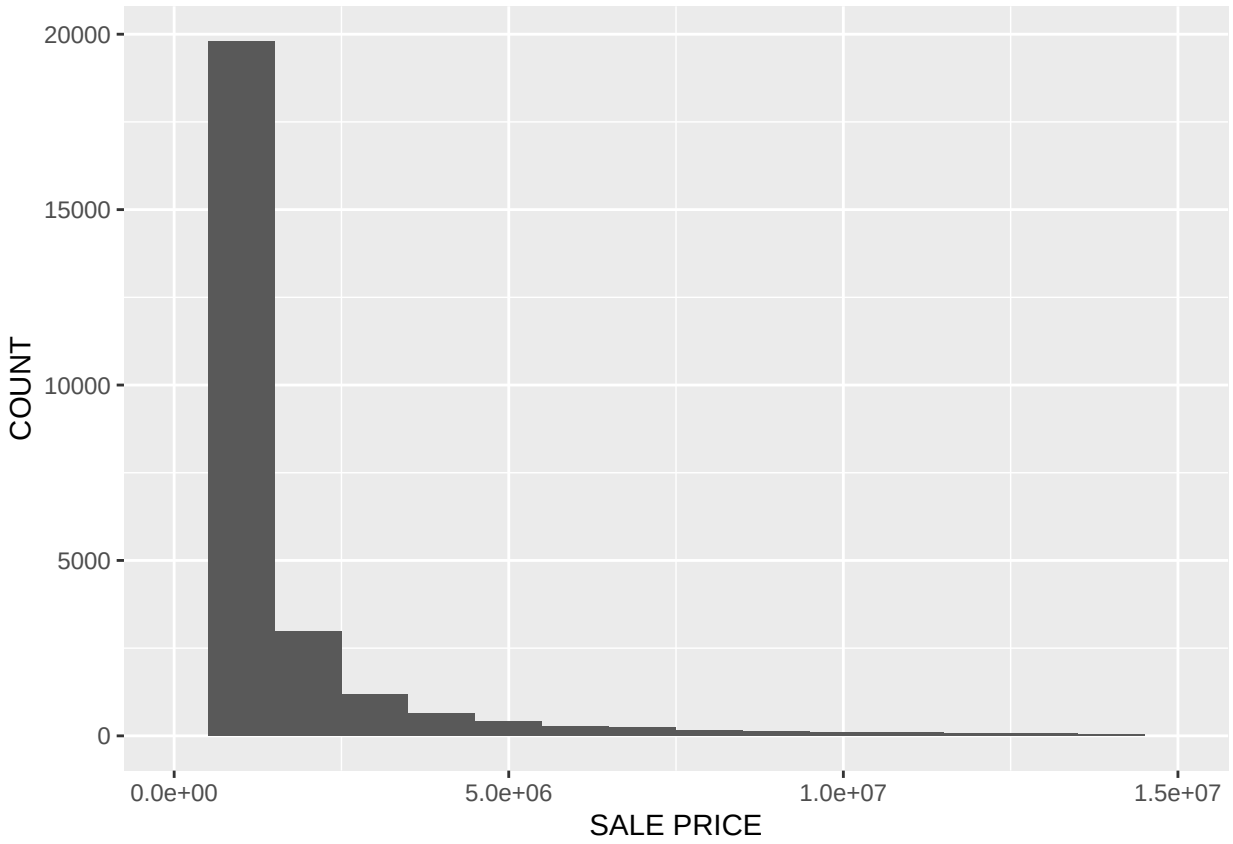
2.5 Data Exploration and Visualization

The following code calculates the summary statistics of Sale Price and its skewness and plots its histogram. The histogram shows that the distribution of Sale Price is right-skewed. The calculated skewness shows that the distribution of Sale Price is extremely skewed. This is due to the heavy tail of the distribution. Note that the skewness of a normal distribution is almost 0. See ITL NIST Handbook for more information on Skewness.

```
# Summary Statistics of Sale Prices
summary(nyc_house_data$SALE.PRICE)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 2.500e+05 4.994e+05 7.390e+05 1.879e+06 1.230e+06 2.210e+09
```

```
# Plot Histogram of Sale Prices
nyc_house_data %>%
  ggplot(aes(SALE.PRICE)) +
  geom_histogram(binwidth = 1e6) +
  scale_x_continuous(limit = c(0, 15e6), name = "SALE PRICE") +
  scale_y_continuous(name = "COUNT")
```




```
# Calculate Skewness of Sale Price
skewness(nyc_house_data$SALE.PRICE)
```

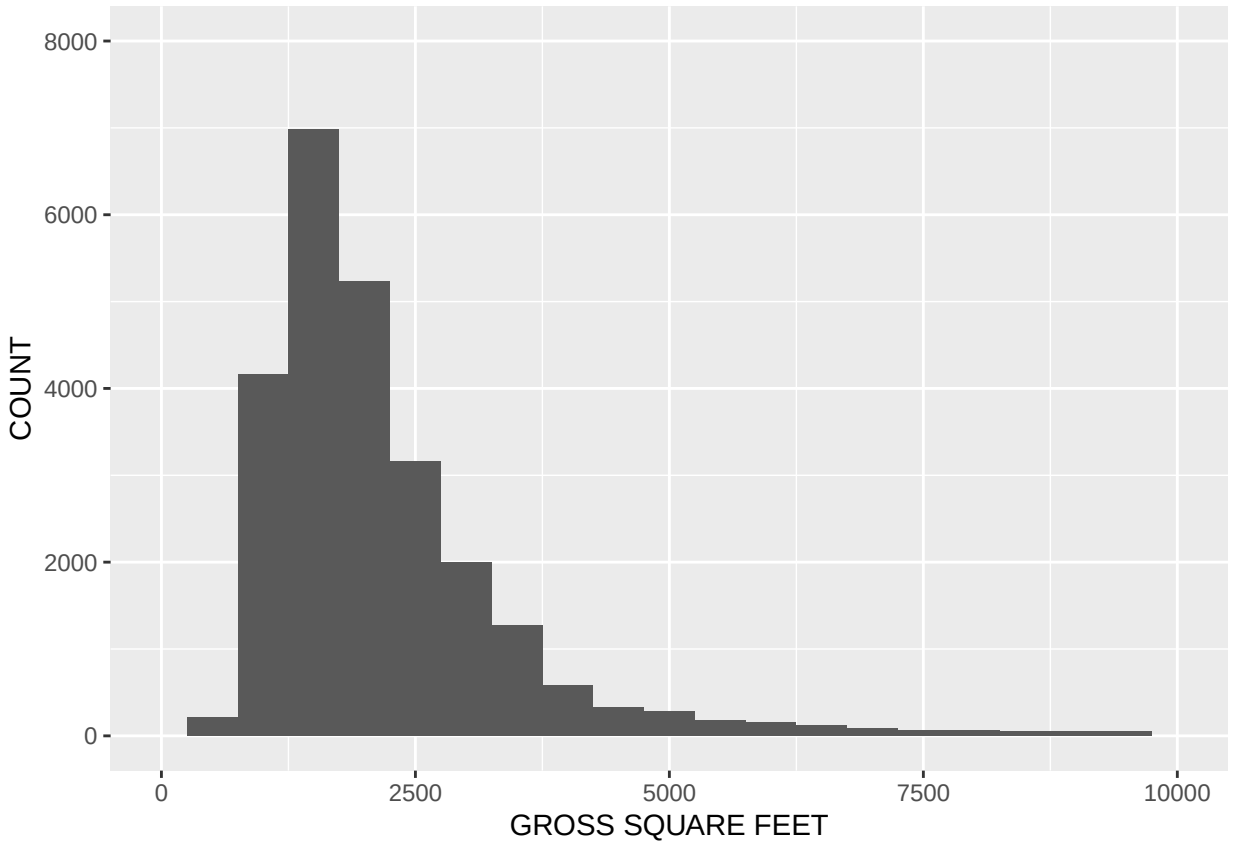
```
## [1] 96.02103
```

Similarly, the following code calculates the summary statistics of Gross Square Feet and its skewness and plots its histogram. The histogram shows that the distribution of Gross Square Feet is right-skewed. The calculated skewness shows that the distribution of Gross Square Feet is extremely skewed similar to Sale Price.

```
# Summary Statistics of Gross Square Feet
summary(nyc_house_data$GROSS.SQUARE.FEET)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0       0    1490    3094    2300 3750565
```

```
# Plot Histogram of Gross Square Feet
nyc_house_data %>%
  ggplot(aes(GROSS.SQUARE.FEET)) +
  geom_histogram(binwidth = 500) +
  scale_x_continuous(limit = c(0, 1e4), name = "GROSS SQUARE FEET") +
  scale_y_continuous(limits = c(0,8e3), name = "COUNT")
```



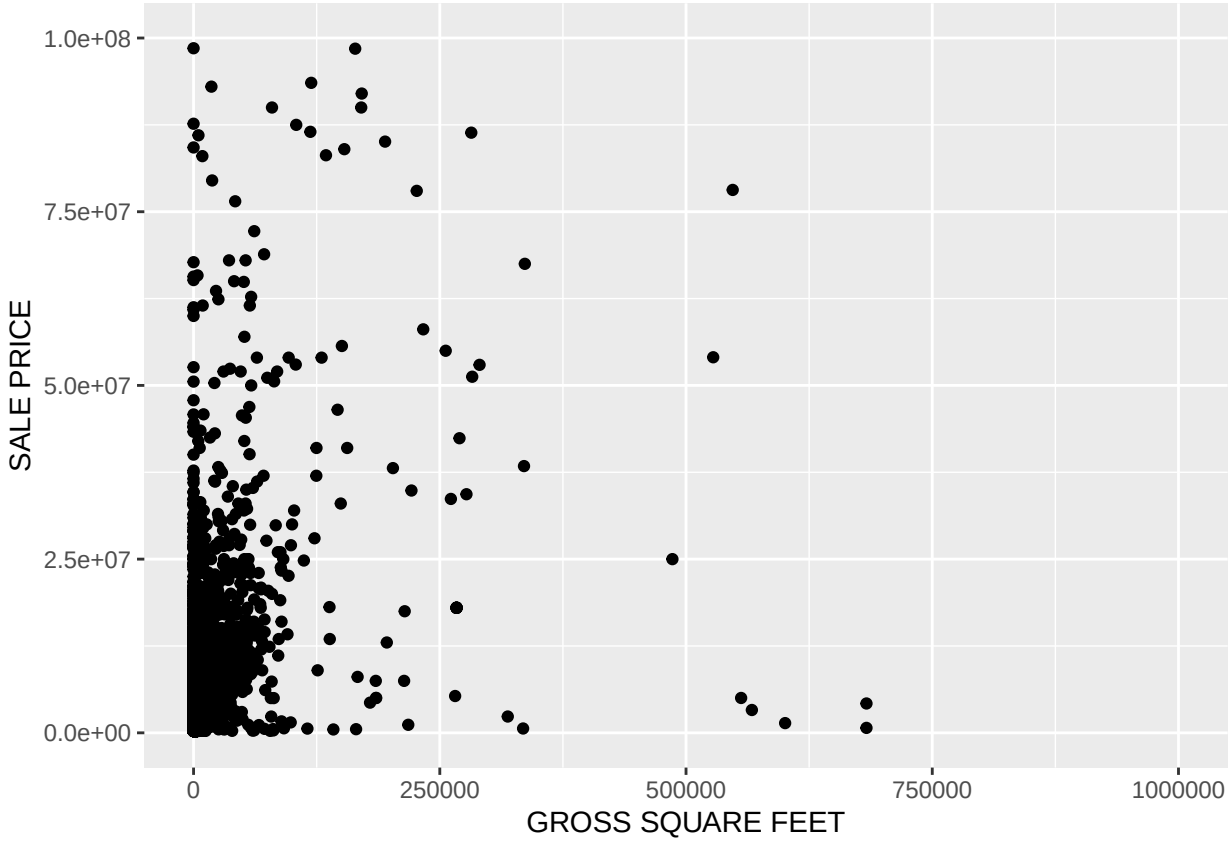

```
# Calculate Skewness of Gross Square Feet
skewness(nyc_house_data$GROSS.SQUARE.FEET)
```

```
## [1] 76.17532
```

The following code plots Sale Price versus Gross Square Feet.

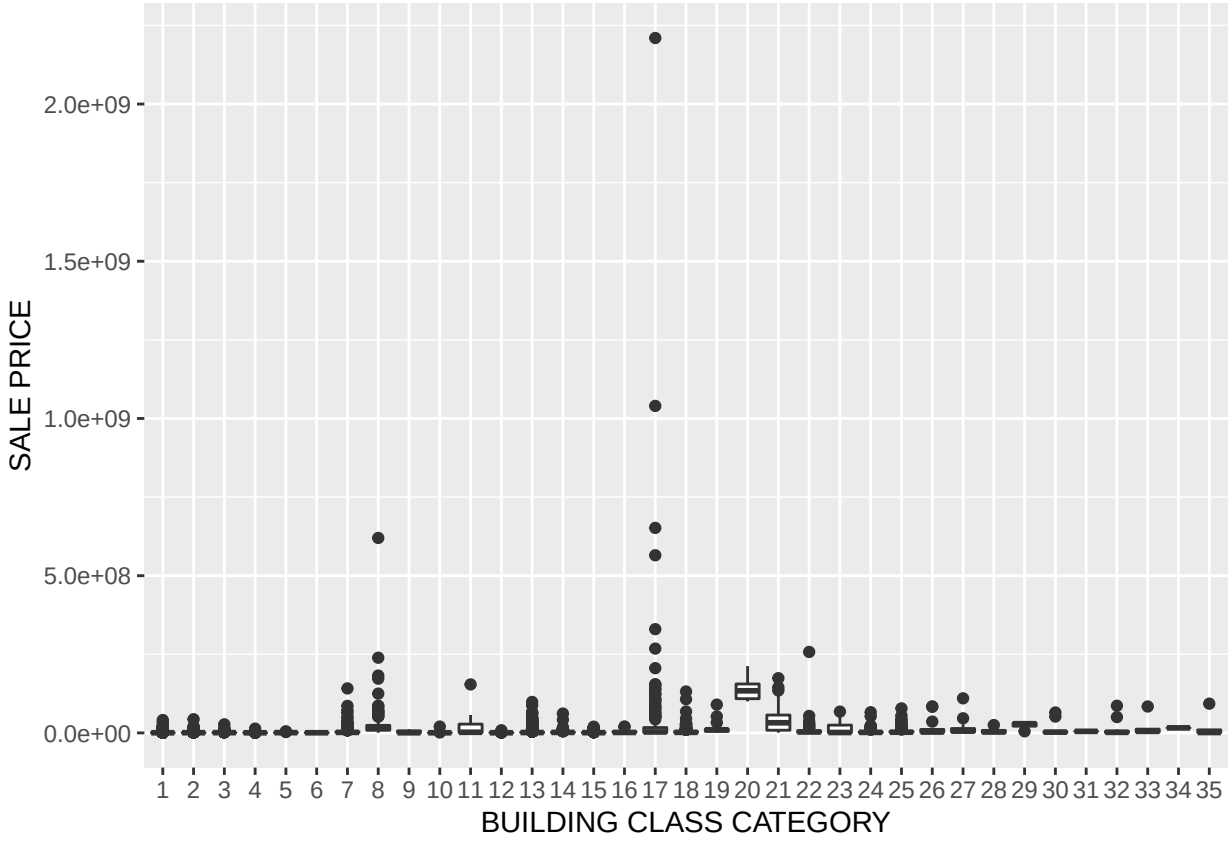
```
# Plot Sale Price vs Gross Square Feet
nyc_house_data %>%
  ggplot(aes(x=GROSS.SQUARE.FEET,y=SALE.PRICE))+
  geom_point()+
  scale_x_continuous(name = "GROSS SQUARE FEET") +
  scale_y_continuous(name = "SALE PRICE")
```


```
# Plot Sale Price vs Gross Square Feet with different axes limits
nyc_house_data %>%
  ggplot(aes(x=GROSS.SQUARE.FEET,y=SALE.PRICE))+
  geom_point()+
  scale_x_continuous(limits = c(0,1e6), name = "GROSS SQUARE FEET") +
  scale_y_continuous(limits = c(0,1e8), name = "SALE PRICE")
```



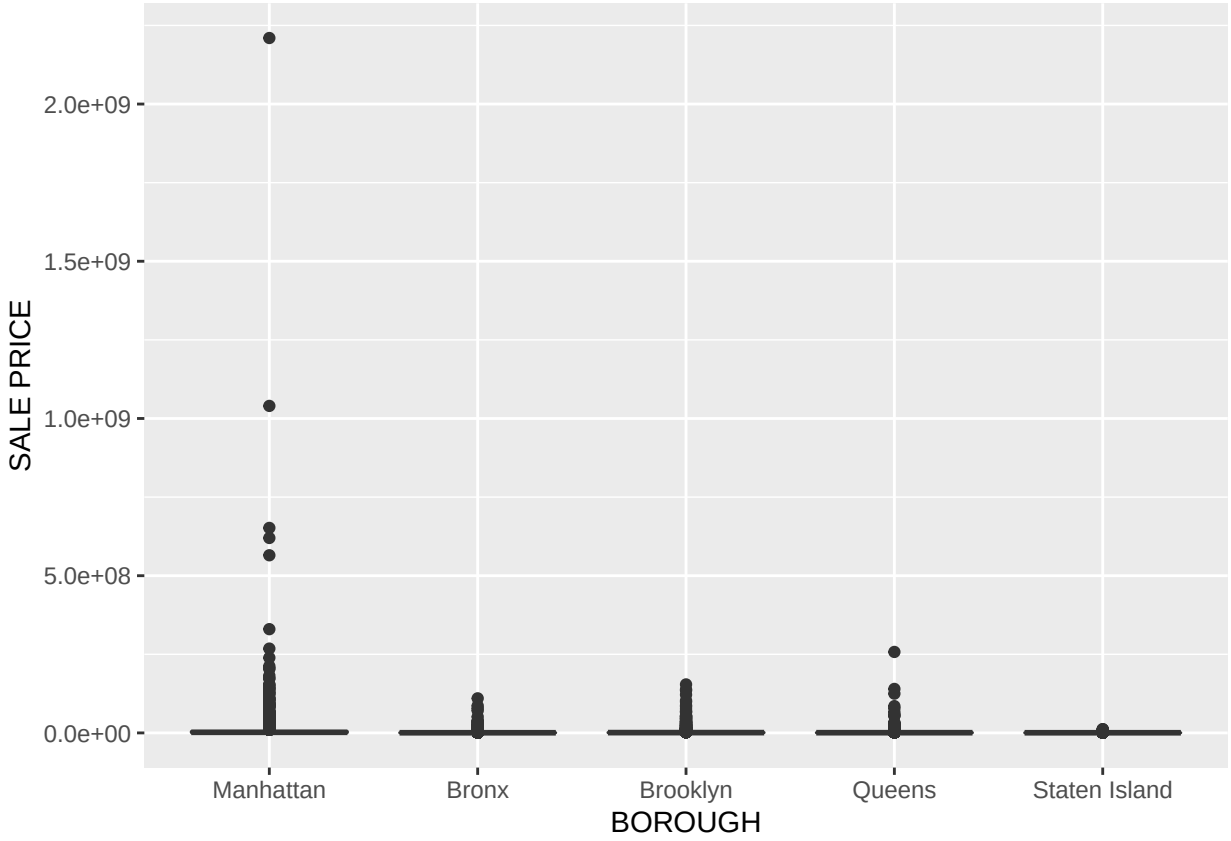
The following plot shows the boxplot of sale prices for each building class category. The plot shows that there are many outliers in sale prices for each catergory. However, although these value are considered outliers statistically, as discussed earlier the high sale prices seem valid data points.

```
# Plot Sale Price vs Building Class Category
nyc_house_data %>%
  ggplot(aes(x=BUILDING.CLASS.CATEGORY,y= SALE.PRICE))+
  geom_boxplot()+
  scale_x_discrete(name = "BUILDING CLASS CATEGORY",
                   labels=seq(1,43,1)) +
  scale_y_continuous(name = "SALE PRICE")
```



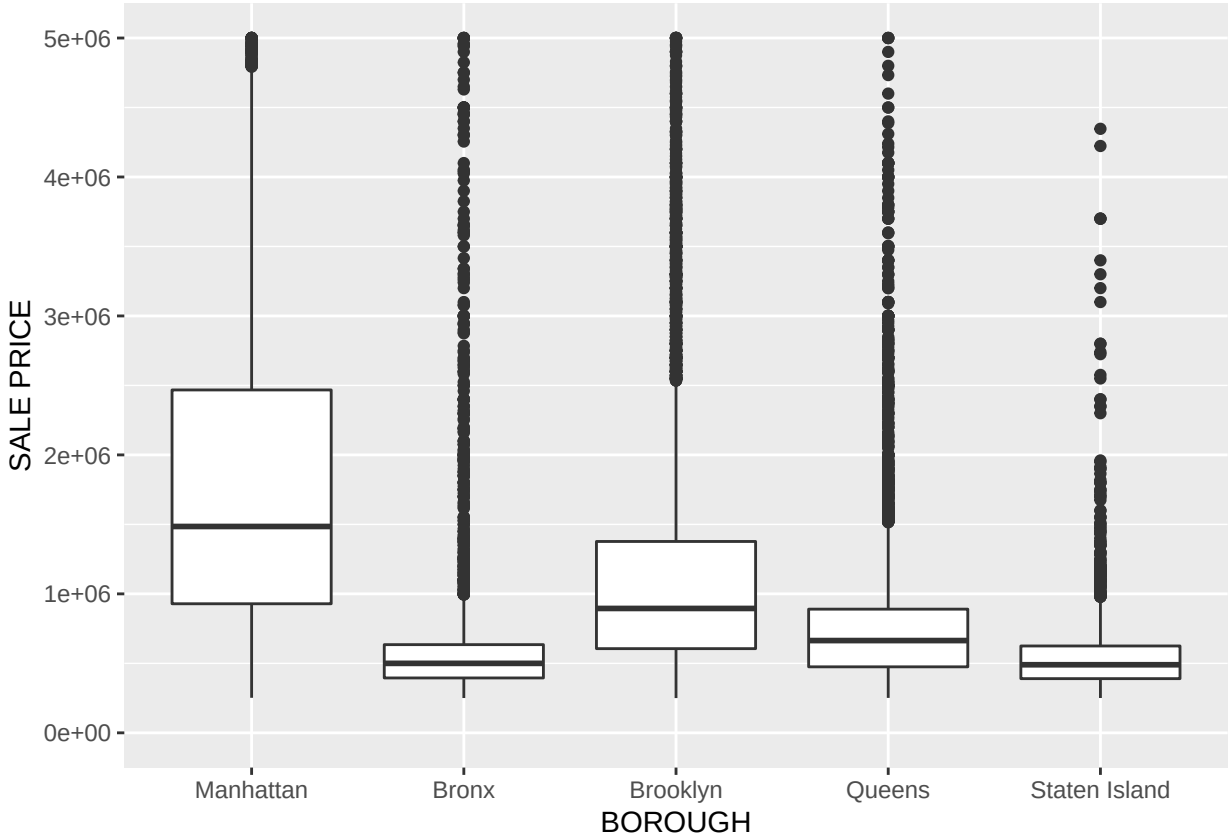
Similarly, the following plot shows the boxplot of sale prices for each borough (borough names are defined in [1]). This plot also shows that there are many outliers in sale prices in each borough. Particularly, there are some very high property sale prices in Manhattan.

```
# Plot Sale Price vs Borough
nyc_house_data %>%
  ggplot(aes(x=as.factor(BOROUGH),y=SALE.PRICE))+
  geom_boxplot()+
  scale_y_continuous(name = "SALE PRICE")+
  scale_x_discrete(name= "BOROUGH",
                   labels=c("Manhattan","Bronx","Brooklyn","Queens","Staten Island"))
```



The following code is similar to the above but with lower sale price limit. The plot shows Manhattan has the highest average sale prices while sale prices in Bronx and Staten Island are the lowest.

```
# Plot Sale Price vs BOROUGH (lower Sale price limit)
nyc_house_data %>%
  ggplot(aes(x=as.factor(BOROUGH),y=SALE.PRICE))+
  geom_boxplot()+
  scale_y_continuous(limit = c(0, 5e6), name = "SALE PRICE")+
  scale_x_discrete(name= "BOROUGH",
                   labels=c("Manhattan","Bronx","Brooklyn","Queens","Staten Island"))
```

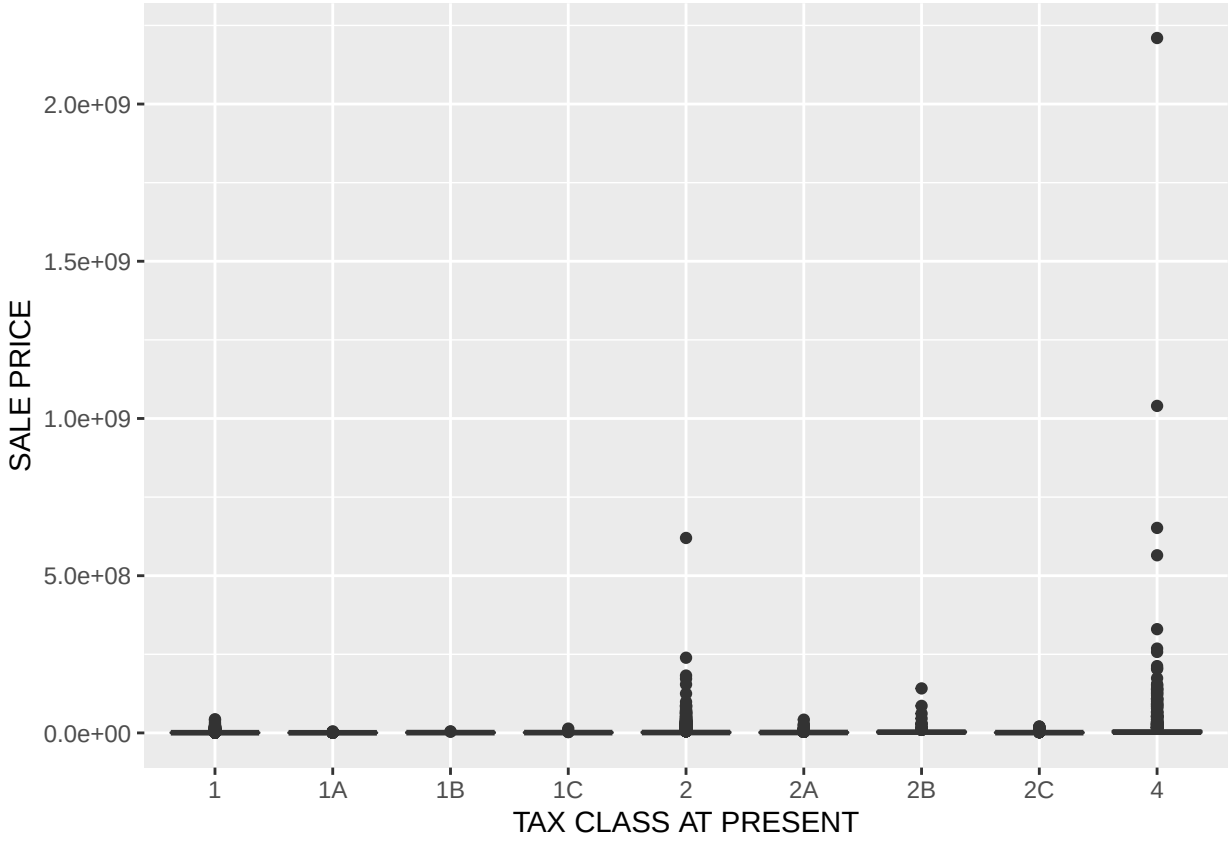


The following code shows the categories of TAX CLASS AT PRESENT and shows the boxplot of SALE PRICE for each category. There are entries with very high sale prices for tax classes of 4 and 2. Similar to previous plots, this plot also confirms that there are many outliers in the data and the distribution of sale prices has a fat tail.

```
# Tax Class at Present levels
levels(nyc_house_data$TAX.CLASS.AT.PRESENT)
```

```
## [1] "1"  "1A" "1B" "1C" "2"  "2A" "2B" "2C" "4"
```

```
# Plot Sale Price vs Tax Class at Present
nyc_house_data %>%
  ggplot(aes(x=TAX.CLASS.AT.PRESENT,y=SALE.PRICE))+
  geom_boxplot()+
  scale_y_continuous(name = "SALE PRICE")+
  scale_x_discrete(name= "TAX CLASS AT PRESENT")
```

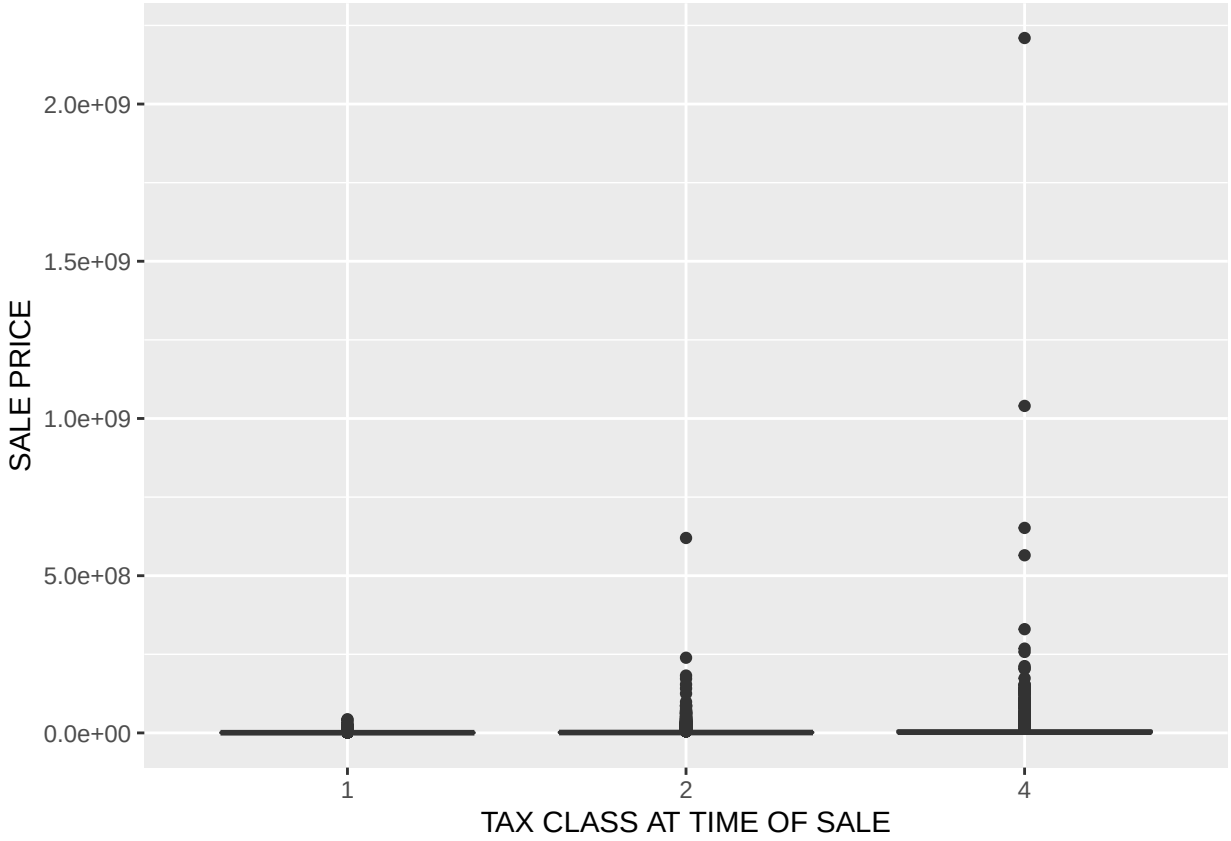


Similarly, the following code shows the categories of TAX CLASS AT TIME OF SALE and shows the boxplot of SALE PRICE for each category.

```
# Tax Class at Time of Sale levels
levels(nyc_house_data$TAX.CLASS.AT.TIME.OF.SALE)
```

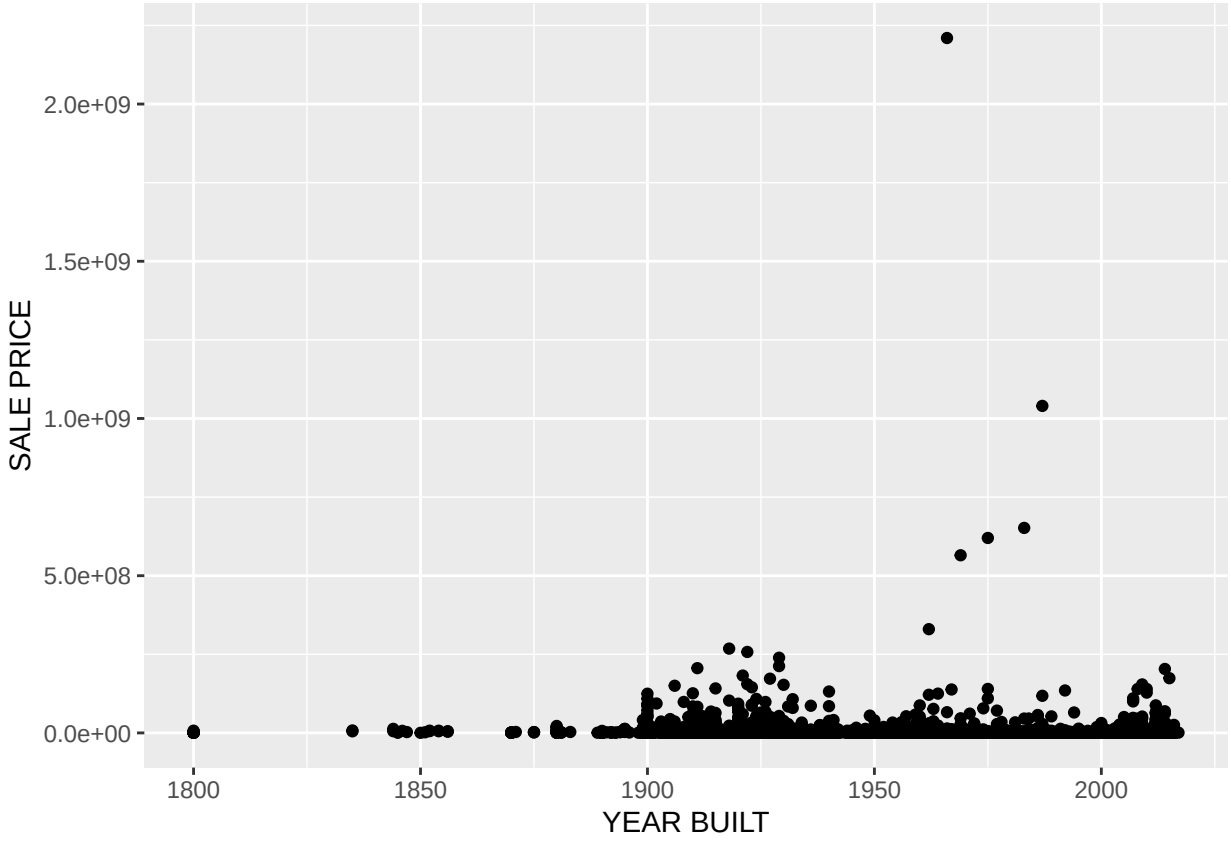
```
## [1] "1" "2" "4"
```

```
# Plot Sale Price vs Tax Class at Time of Sale
nyc_house_data %>%
  ggplot(aes(x=as.factor(TAX.CLASS.AT.TIME.OF.SALE),y=SALE.PRICE))+
  geom_boxplot()+
  scale_y_continuous(breaks= seq(0,5e6,0.2e6))+
  scale_y_continuous(name = "SALE PRICE")+
  scale_x_discrete(name= "TAX CLASS AT TIME OF SALE")
```

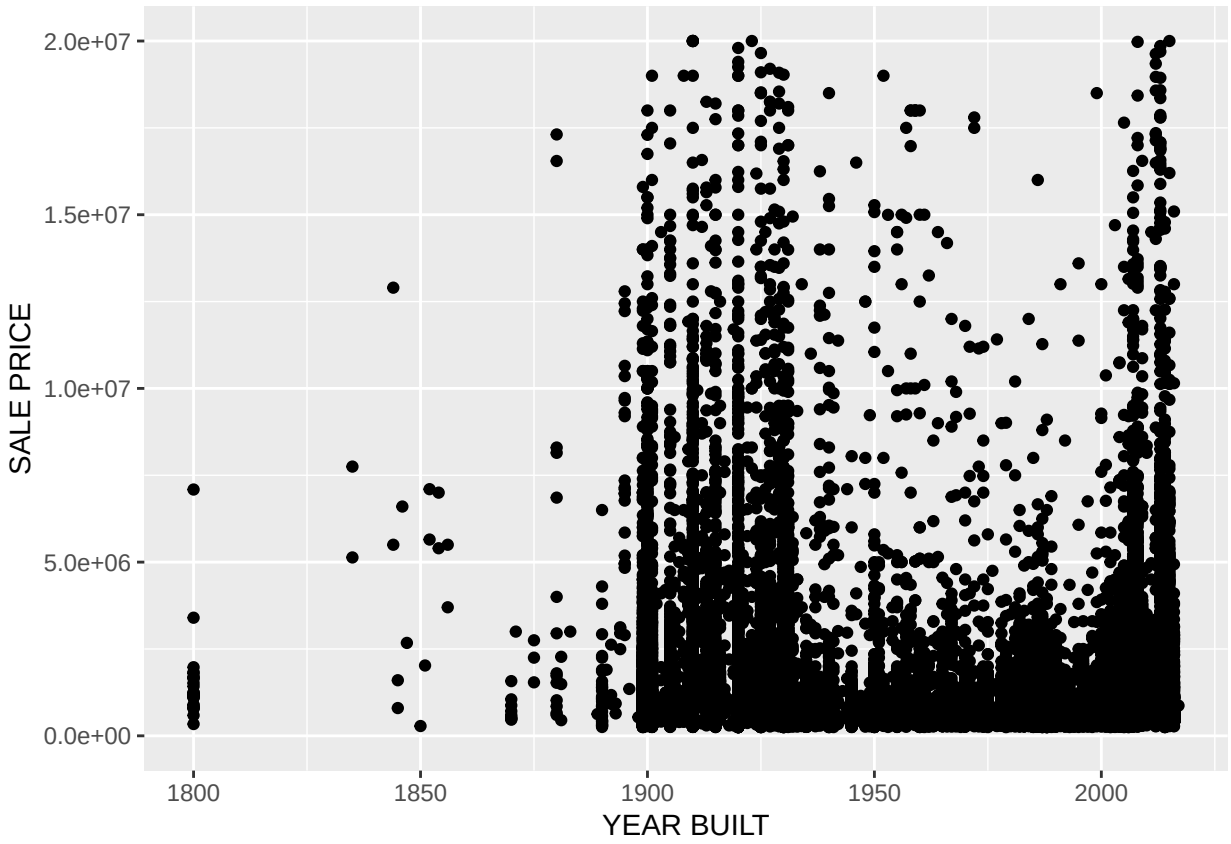


The following plots show Sale Price versus Year Built. It does not seem that there is a strong correlation between the two.

```
# Plot Sale Price vs Year Built
nyc_house_data %>%
  ggplot(aes(x=YEAR.BUILT,y=SALE.PRICE))+
  geom_point()+
  scale_x_continuous(name = "YEAR BUILT")+
  scale_y_continuous(name = "SALE PRICE")
```

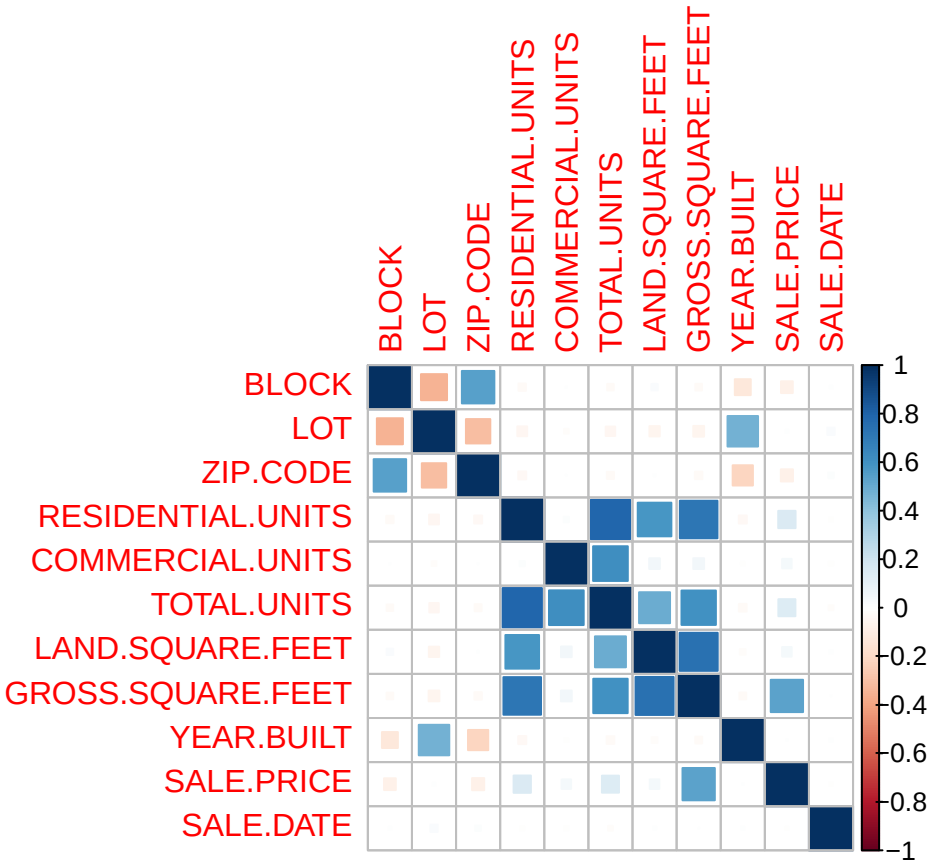



```
# Plot Sale Price vs Year Built with lower sale price limit
nyc_house_data %>%
  ggplot(aes(x=YEAR.BUILT,y=SALE.PRICE))+
  geom_point()+
  scale_x_continuous(name = "YEAR BUILT")+
  scale_y_continuous(name = "SALE PRICE", limit = c(0, 2e7))
```



The following plot shows the correlation of numerical variables.

```
# Plot Correlations of numerical variables
num_col <- sapply(nyc_house_data, is.numeric) # find numeric columns
corrs   <- cor(nyc_house_data[,num_col])
corrplot(corrs, method="square")
```



2.6 Logarithmic Data Transformation

As discussed earlier, examining some of the numerical variables showed that they are very skewed. The problem with skewed data is that it could result in training the machine learning algorithm on a large number of moderately priced houses. This could result in significant error in predicting prices of expensive properties [2]. Logarithmic transformation is one of the best methods to handle skewed data. The following code calculates the skewness of all except four numerical variables and applies Logarithmic transformation if the skewness is larger than 0.75. Residential Units, Commercial Units, Land Square Feet and Gross Square Feet are not transformed since they have entries of 0.

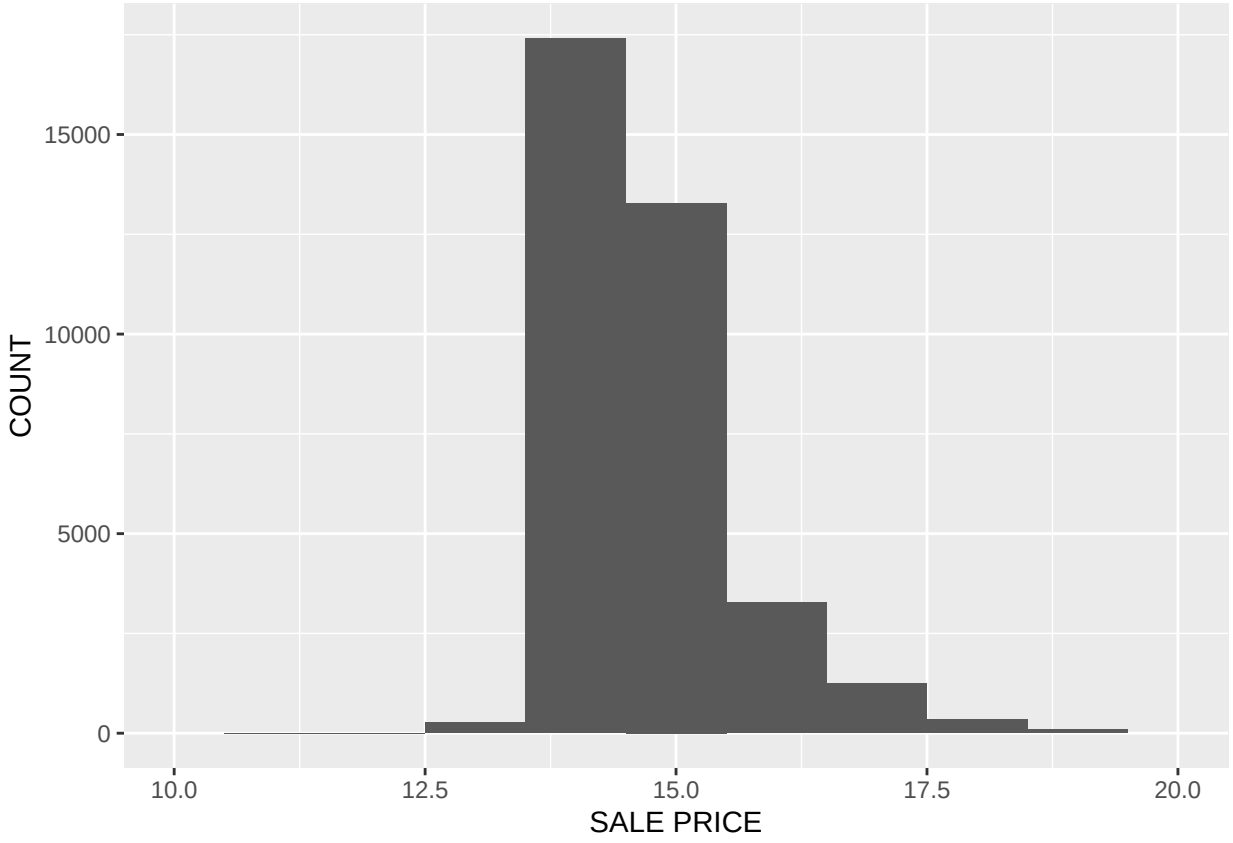
```
# # Apply log transformation to Numerical Variable with skewness > 0.75
for(x in seq(1,ncol(nyc_house_data)))
{
  if(is.numeric(nyc_house_data[,x]))
  {
    skew <- skewness(nyc_house_data[,x],na.rm = T)
    if (skew > 0.75 &
        names(nyc_house_data[x]) != "RESIDENTIAL.UNITS" &
        names(nyc_house_data[x]) != "COMMERCIAL.UNITS" &
        names(nyc_house_data[x]) != "LAND.SQUARE.FEET" &
        names(nyc_house_data[x]) != "GROSS.SQUARE.FEET"
        )
    {
      print(skew)
      print(names(nyc_house_data[x]))
      nyc_house_data[,x] <- log(nyc_house_data[,x])+1
    }
```

```
  }
}
```

```
## [1] 0.9159831
## [1] "BLOCK"
## [1] 2.532948
## [1] "LOT"
## [1] 69.05576
## [1] "TOTAL.UNITS"
## [1] 96.02103
## [1] "SALE.PRICE"
```

The plot below shows the histogram of sale prices after the log transformation. The distribution of sale prices is still right-skewed but its skewness reduces after transformation.

```
# Plot Histogram of Sale Prices after log transformation
nyc_house_data %>%
  ggplot(aes(SALE.PRICE)) +
  geom_histogram(binwidth = 1) +
  scale_x_continuous(limit = c(10, 20), name = "SALE PRICE") +
  scale_y_continuous(name = "COUNT")
```



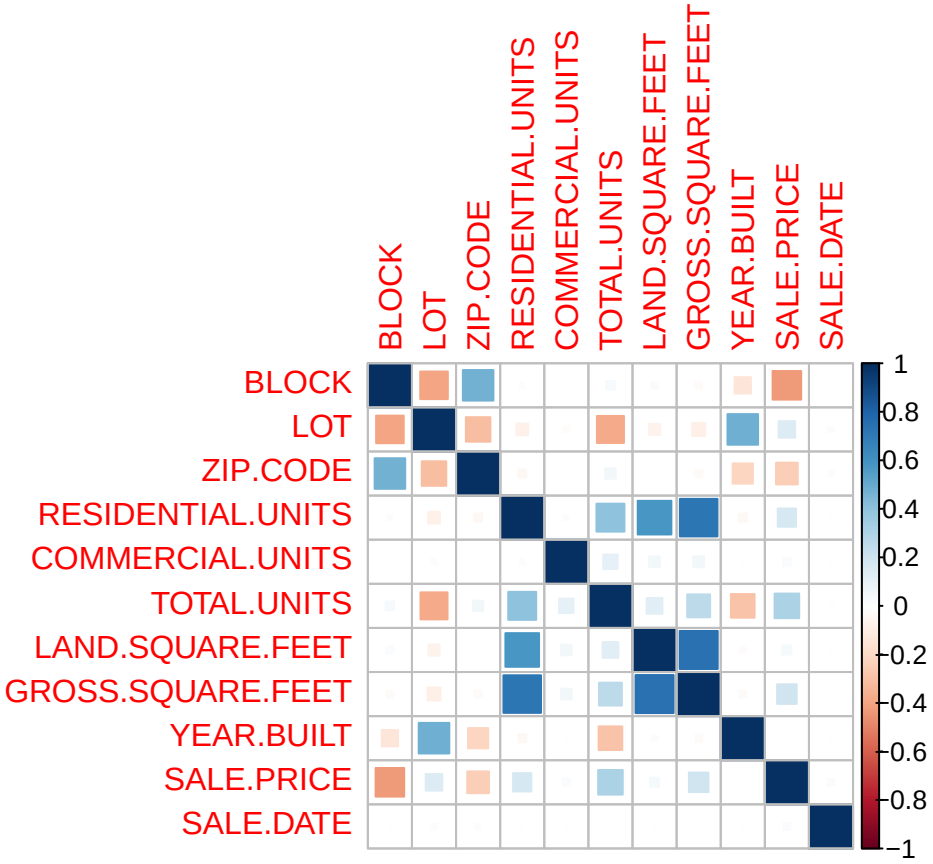
The following code calculates the skewness of Sale Price. It confirms that the skewness of Sale Price is significantly less than its skewness before transformation.

```
# Calculate Skewness of Sale Price after log transformation
skewness(nyc_house_data$SALE.PRICE)
```

```
## [1] 1.707472
```

The following plot shows the correlation of numerical variables after the log transformation. The plot shows that after transformation the correlation of sale price with Total Units increases. Also, there seems to be a fairly strong negative correlation between Sale Price and Block. However, the correlation of Sale Price and Gross Square Feet is less than the one before transformation.

```
# Plot Correlations of numerical variables after log transformation
num_col <- sapply(nyc_house_data, is.numeric) # find numeric columns
corrs   <- cor(nyc_house_data[,num_col])
corrplot(corrs, method="square")
```



2.7 Create Training and Test Sets

The code below creates the training and test datasets from the nyc_house_data dataset. Test set is selected to be 10% of the original dataset. This is selected as a typical choice for the size of the test set. Considering the size of the dataset (almost 35000 instances), the relative size of the train and test sets does not have a significant impact on the performance of the machine learning model. Also, cross validation is used in training the models.

```
# Create training and test sets
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = nyc_house_data$SALE.PRICE, times = 1,
                                  p = 0.1, list = FALSE)
train_set <- nyc_house_data[-test_index,]
test_set  <- nyc_house_data[test_index,]
```

2.8 Algorithms

2.8.1 Random Forest

The random forest impelementation in the RandomForest package can handle categorical variables with up to 53 levels. The following code prints the categrical variables with more than 53 levels.

```
# Print categorical predictors that have more than 53 levels/categories
for(x in seq(1,ncol(nyc_house_data)))
{
  if(is.factor(nyc_house_data[,x]) & length(unique(nyc_house_data[,x])) > 53)
  {
    print(names(nyc_house_data[x]))
    print(length(unique(nyc_house_data[,x])))
  }
}
```

```
## [1] "NEIGHBORHOOD"
## [1] 247
## [1] "BUILDING.CLASS.AT.PRESENT"
## [1] 130
## [1] "BUILDING.CLASS.AT.TIME.OF.SALE"
## [1] 131
```

The following code removes categorical variables with more than 53 levels since the Random Forest implementation in R cannot handle such variables.

```
test_set_rf <- subset(test_set, select = -c(NEIGHBORHOOD,BUILDING.CLASS.AT.PRESENT,
                                            BUILDING.CLASS.AT.TIME.OF.SALE
  ))
train_set_rf <- subset(train_set, select = -c(NEIGHBORHOOD,BUILDING.CLASS.AT.PRESENT,
                                              BUILDING.CLASS.AT.TIME.OF.SALE
                                              ))
```

The following code trains the Random Forest model, predicts sale prices and compares them with the prices in the test set. If is_tune is equal to 1, the code will tune the algorithm by adjusting the mtry parameter, which is the number of variables randomly sampled at each split. The code tests five values of mtry: 2, 3, 4, 5, 6. If is_tune is 0, the model will be trained with mtry_input parameter provided by the user. The code also plots the variable importance chart measured by the random forest.

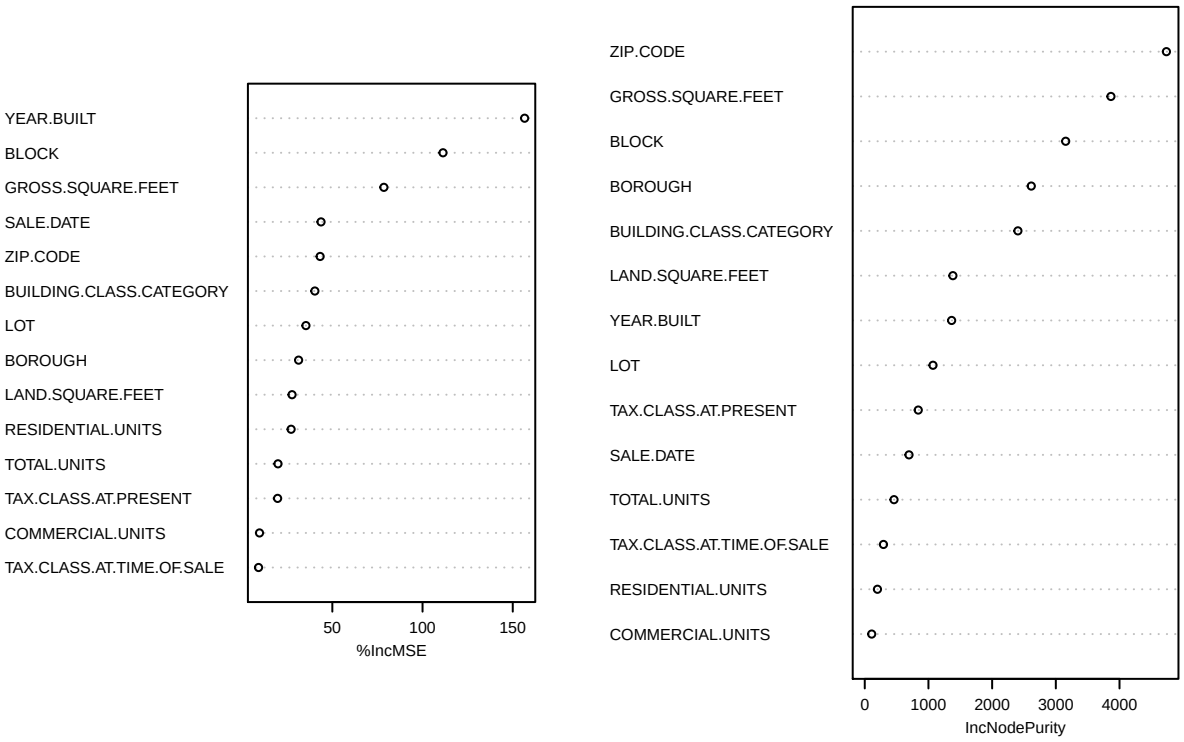
```
# If is_tune = 1, the code tunes the mtry parameter, which is the number of variables
# randomly sampled at each split

if(is_tune == 1) {
  nyc_house_forest1 <- train(
    SALE.PRICE ~ .,
    data = train_set_rf,
    method = "rf",
    tuneGrid = data.frame(mtry = seq(2, 6, 1))
  )
  print(nyc_house_forest1$finalModel)         # Final model information
  print(nyc_house_forest1$bestTune)           # Print the optimal mtry value
  print(nyc_house_forest1$results)
  print(importance(nyc_house_forest1))        # Variable Importance measure
  varImpPlot(nyc_house_forest1, cex = 0.5)    # Chart of Variable Importance
  fit1 <- predict(nyc_house_forest1, newdata = test_set_rf)
  data1 = data.frame(obs=test_set_rf$SALE.PRICE, pred=fit1)
```

```
  defaultSummary(exp(data1-1))       # Compare model prediction with test_set
} else{
  # Random Forest Training with the mtry_input value
  nyc_house_forest2 <- randomForest(SALE.PRICE ~ . ,
                                    data = train_set_rf, mtry = mtry_input, importance = T)
  print(nyc_house_forest2)
  print(importance(nyc_house_forest2))           # Variable Importance measure
  varImpPlot(nyc_house_forest2, cex = 0.5)       # Chart of Variable Importance

  fit2 <- predict(nyc_house_forest2, newdata = test_set_rf)
  data2 = data.frame(obs=test_set_rf$SALE.PRICE, pred=fit2)
  defaultSummary(exp(data2-1))               # Compare model prediction with test_set
}
```

```
##
## Call:
##  randomForest(formula = SALE.PRICE ~ ., data = train_set_rf, mtry = mtry_input,      importance = T)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##           Mean of squared residuals: 0.1336771
##                     % Var explained: 81.97
##                                %IncMSE IncNodePurity
## BOROUGH                     31.305532     2614.7077
## BUILDING.CLASS.CATEGORY     40.319722     2404.8905
## TAX.CLASS.AT.PRESENT        19.686255      839.7871
## BLOCK                      111.337234     3154.8424
## LOT                         35.382667     1071.2008
## ZIP.CODE                    43.244709     4736.5651
## RESIDENTIAL.UNITS           27.165752      199.2345
## COMMERCIAL.UNITS             9.645590      108.8912
## TOTAL.UNITS                 19.891886      459.1645
## LAND.SQUARE.FEET            27.683725     1383.8998
## GROSS.SQUARE.FEET           78.568363     3865.3642
## YEAR.BUILT                 156.573487     1363.9582
## TAX.CLASS.AT.TIME.OF.SALE    9.116391      293.4677
## SALE.DATE                   43.744631      694.5820
```




```
##          RMSE      Rsquared           MAE
## 3.783477e+06 5.556054e-01 6.191281e+05
```

2.8.2 Linear Regression

The following code uses Linear Regression to predict sale prices.

```
# Use Linear Regression to Predict Property Sale Prices

# Linear Regression
model_lm=train(SALE.PRICE~.,
               data=train_set,
               method="lm"
)

fit_lm <- predict(model_lm, newdata = test_set)
data = data.frame(obs=test_set$SALE.PRICE, pred=fit_lm)
defaultSummary(exp(data-1))       # Compare model prediction with test_set
```

```
##          RMSE      Rsquared           MAE
## 4.170694e+06 5.063688e-01 8.176480e+05
```

The following command saves the end time of the run in the end_time variable and prints the total run time.

```
end_time <- Sys.time()
run_time = end_time - start_time
print(c("Run time is" ,run_time))
```

```
## [1] "Run time is"      "2.30822981721825"
```

3 Results

Random Forest and Linear Regression algorithms were used to predict the NYC property sale prices. RandomForest has a better performance than Linear Regression. The RMSE of sale price prediction is 3.78e+06 for RandomForest compared to 4.17e+06 for Linear Regression.

The RandomForest algorithm was tuned to find the optimal parameter for this application. The tuning parameter of RandomForest is mtry, which is the number of variables randomly sampled as candidates at each split. According to [3], the best starting point for tuning mtry is typically the square root of number of predictors. In this application, this value is almost 4 (3.87). Five values of mtry were used for tuning: 2, 3, 4, 5, 6. Tuning of the algorithm took about 8 hours on my computer. As mtry increases, training of the RandomForest algorithm takes more time. Maximum value of mtry tested was limited to 6 in this project due to computational limitations.

The computational time for training the RandomForest algorithm is significantly higher than Linear Regression. Training took about 2 hr for the final RandomForest algorithm (mtry = 6) while it took about 8 minutes for Linear Regression.

4 Conclusion

The goal of this project is to predict the property sale prices in New York City. Two machine learning algorithms were used: Random Forest and Linear Regression.

Random Forest model has a better performance. This is due to the fact that Random Forest is robust to overfitting while Linear Regression without regularization can become too flexible specially for datasets with many outliers such as the one used in this project.

Various steps were taken to clean the dataset from Kaggle. Sale prices and some of the other numerical variables of the dataset are very skewed. Log transformation was used to transform these variables to improve the model performance. There are also many outliers in the dataset. Entries with very low prices were removed from the dataset as they did not seem reasonable. However, very expensive properties were kept as they seemed feasible although they are considered outliers statistically.

In the future, other machine learning algorithms can be tested to see if they can provide a better performance. In literature Gradient Boosting machine [4], lasso regression [5], etc., are used for similar problems.

It might also be helpful to examine and modify entries with Gross Square Feet and/or Land Square Feet of 0 or blank. This will require knowledge of the real estate, e.g., different building categories. Some of the entries may not be accurate and removing them could improve the model performance.

References

1- https://www.kaggle.com/new-york-city/nyc-property-sales

2- https://opendatascience.com/transforming-skewed-data-for-machine-learning

3- Manual On Setting Up, Using, And Understanding Random Forests V3.1.

4- Winky K.O. Ho , Bo-Sin Tang & Siu Wai Wong, “Predicting property prices with machine learning algorithms”, Journal of Property Research, Oct 2020.

5- Yichen Zhou, “Housing Sale Price Prediction Using Machine Learning Algorithms”, UCLA Master’s thesis, 2020.