Econometrics final project

Chan, Kim Fai

12/17/2021

Index

1 Introduction 1

2 Method and Analysis

2.1 Loading Libraries

2.2 Options for Running the Code

2.3 Download the Data set

2.4 Data Cleaning

2.5 Data Exploration and Visualization

2.6 Logarithmic Data Transformation

2.7 Create Training and Test Sets

2.8 Algorithms

3 Results

4 Conclusion

References

1 Introduction

The approach and code developed for the final project are described in this report. The purpose of the research is to forecast house prices in New York City (NYC). The Kaggle project dataset contains data from one year of building or building unit sales in New York City. It provides information such as the sale price, zip code, block, gross square feet, address, and building type for each property. Sale price and numerous factors, such as gross square feet, are correlated, according to data investigation and visualization. Data cleaning was done to remove missing values and data that was not useful in predicting selling prices, as well as to reduce the influence of data skewness. The data was then divided into training and test sets. The sale prices of the test set were predicted using Random Forest and linear regression models trained on the training 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.or
g")
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.or
g")
```

```
                                        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

The Random Forest model will be tuned if is tune is set to 1. If is tune is set to 0, the function will use the mtry input parameter to train the Random Forest model. Tuning the Random Forest method for this application takes roughly 8 hours, as stated in the results section. It takes about 2 hours to train the Random Forest model with the best parameter (mtry=6). For a faster run, the user can set mtry to lower numbers, but the model will not be optimal. The Random Forest model will be trained in a matter of minutes with mtry of 2.

```
## 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. 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://raw.githubusercontent.com/billchancn/finalproject/main/nyc-rolling-sales.csv",
              "./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

When looking at the dataset, it becomes clear that several of the columns are useless for predicting sale prices. All entries in the EASE.MENT column are null, indicating that column X is an ID for each group of transactions. The apartment number is also ineffective in determining the price. Furthermore, while there is a link between an apartment's address/street and its price, due to the enormous number of addresses/streets, it does not appear to be a relevant input to a machine learning method. There are also several other variables in the data set that are related to a property's location, such as borough, zip code, and neighborhood. They can be used as a predictor instead of the address.

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)
```

Let's have a look at the sale price values now. Some of the entries are blank. These transactions usually include the transfer of deeds between parties. Parents, for example, may transfer ownership to their children.

Some of the entries are for tiny sums of money. They don't appear to be appropriate for New York City residences. The price of an apartment in Manhattan, for example, is not realistic at 70,000 dollars. All entries with a sale price of less than $250,000 are eliminated from this list. Office buildings, apartment complexes with multiple units, and vast garages, on the other hand, have very high sale prices in the dataset. These entries appear to be correct, thus they will be maintained.

```
## 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
```

When looking at the Gross Square Feet and Land Square Feet entries, it becomes clear that for some building types, such as housing cooperatives, both Gross Square Feet and Land Square Feet values are 0 or blank. Additionally, for sites that do not include structures, such as garages, Gross Square Feet is set to zero although Land Square Feet is not. The entries with Gross Square Feet or Land Square Feet of 0 are retained for these reasons. The following code looks at the blank Gross Square Feet and Land Square Feet entries. It also converts the blank ("-") values of Gross Square Feet and Land Square Feet to numeric variables by setting them to 0.

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

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

```
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
```

```
# 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
```

```
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
```

```
# 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.

```
# 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, ]
```

The sum of residential and commercial units is not equal to total units in some rows, according to the residential, commercial, and total unit columns. Furthermore, both residential and business units are zero in some rows. These rows do not appear to be correct. As a result, they are no longer included in the dataset.

```
# 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
```

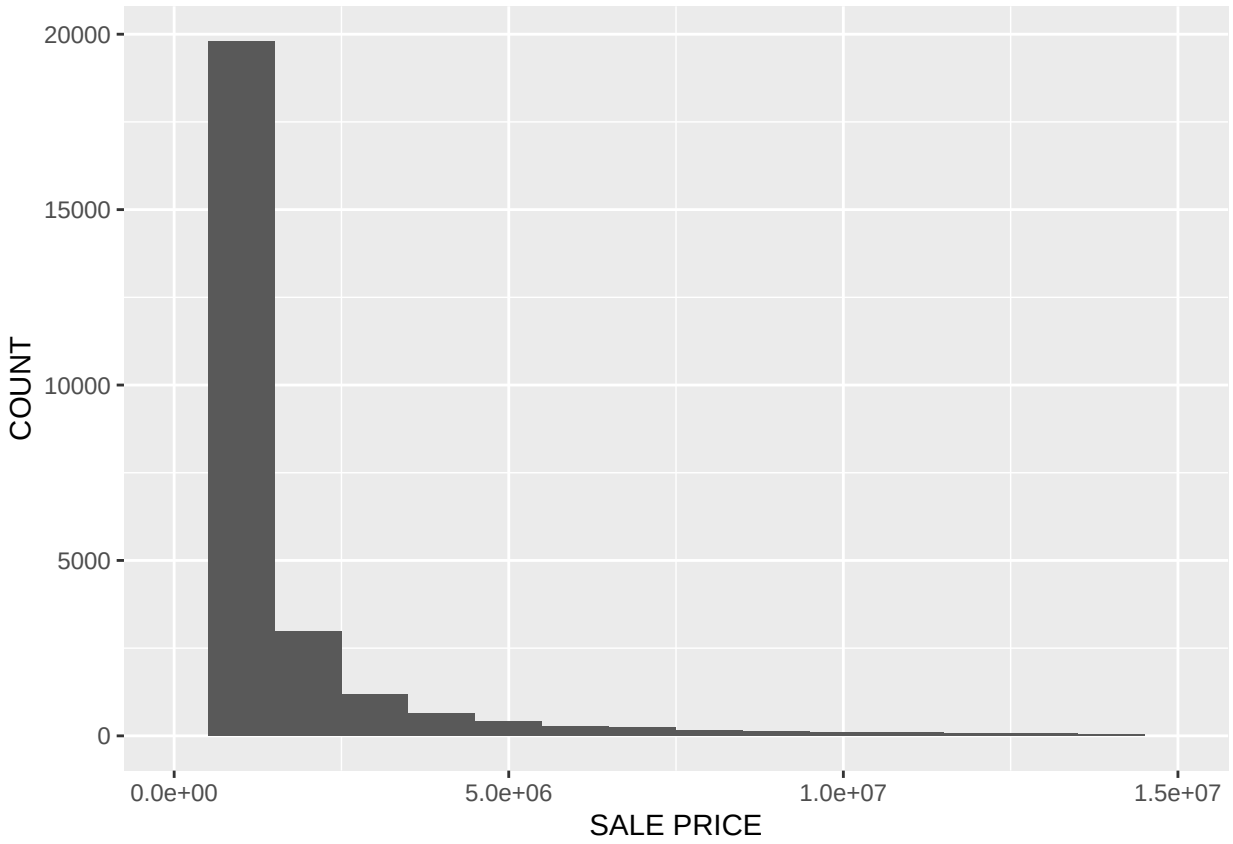
2.5 Data Exploration and Visualization

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)
```

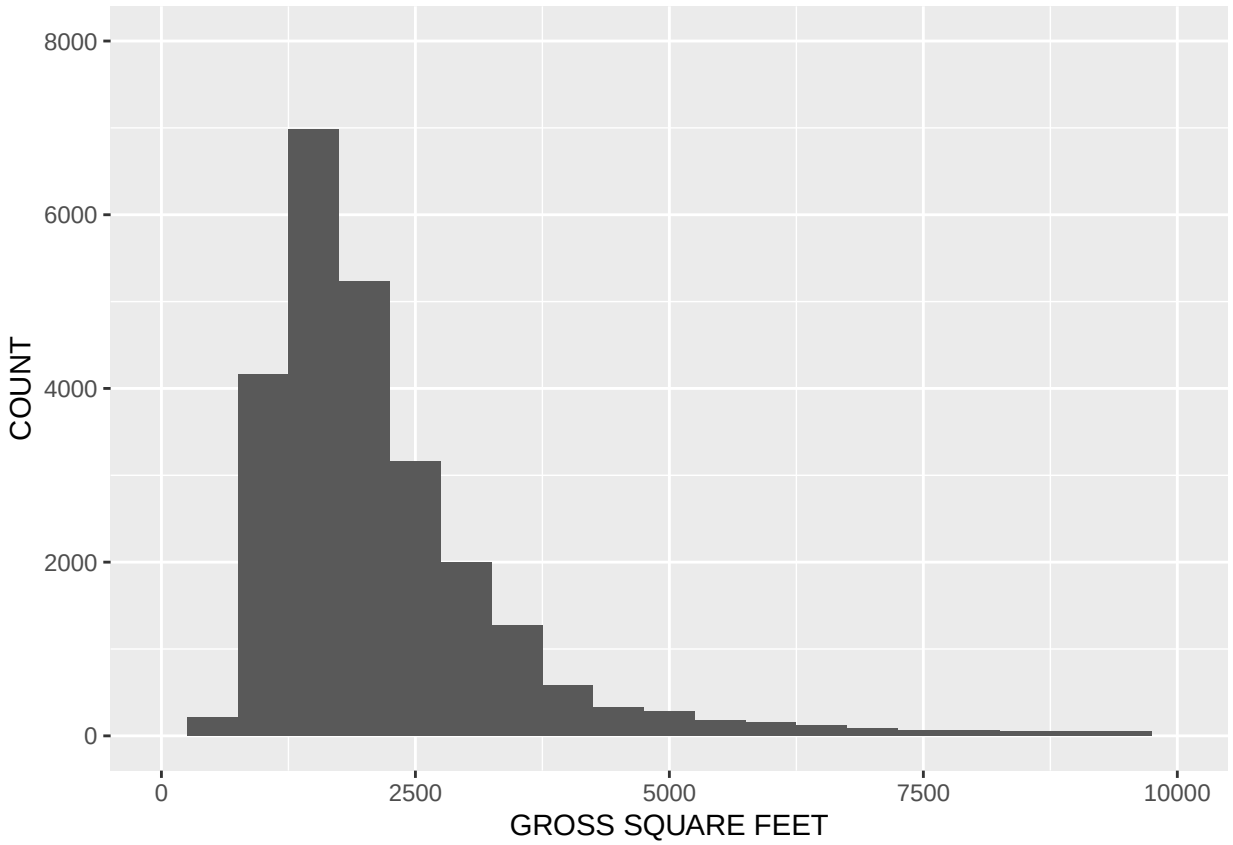
```
## [1] 96.02103
```

Similarly, the following code computes Gross Square Feet summary statistics and skewness, as well as plotting its histogram. The distribution of Gross Square Feet is right-skewed, as seen by the histogram. The estimated skewness demonstrates that the Gross Square Footage distribution is very skewed, similar to the Sale Price distribution.

```
# 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")
```



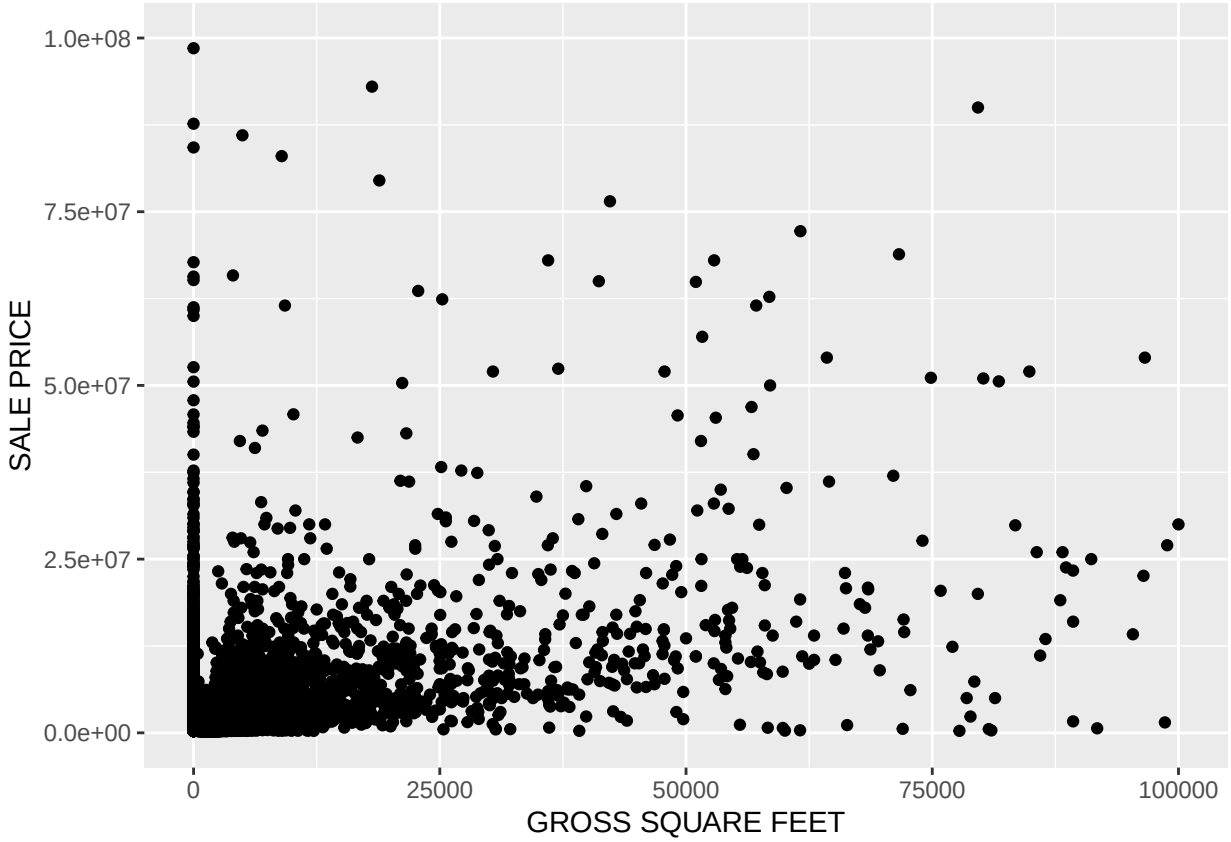

```
# Calculate Skewness of Gross Square Feet
skewness(nyc_house_data$GROSS.SQUARE.FEET)
```

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

The following code plots Sale Price versus Gross Square Feet.

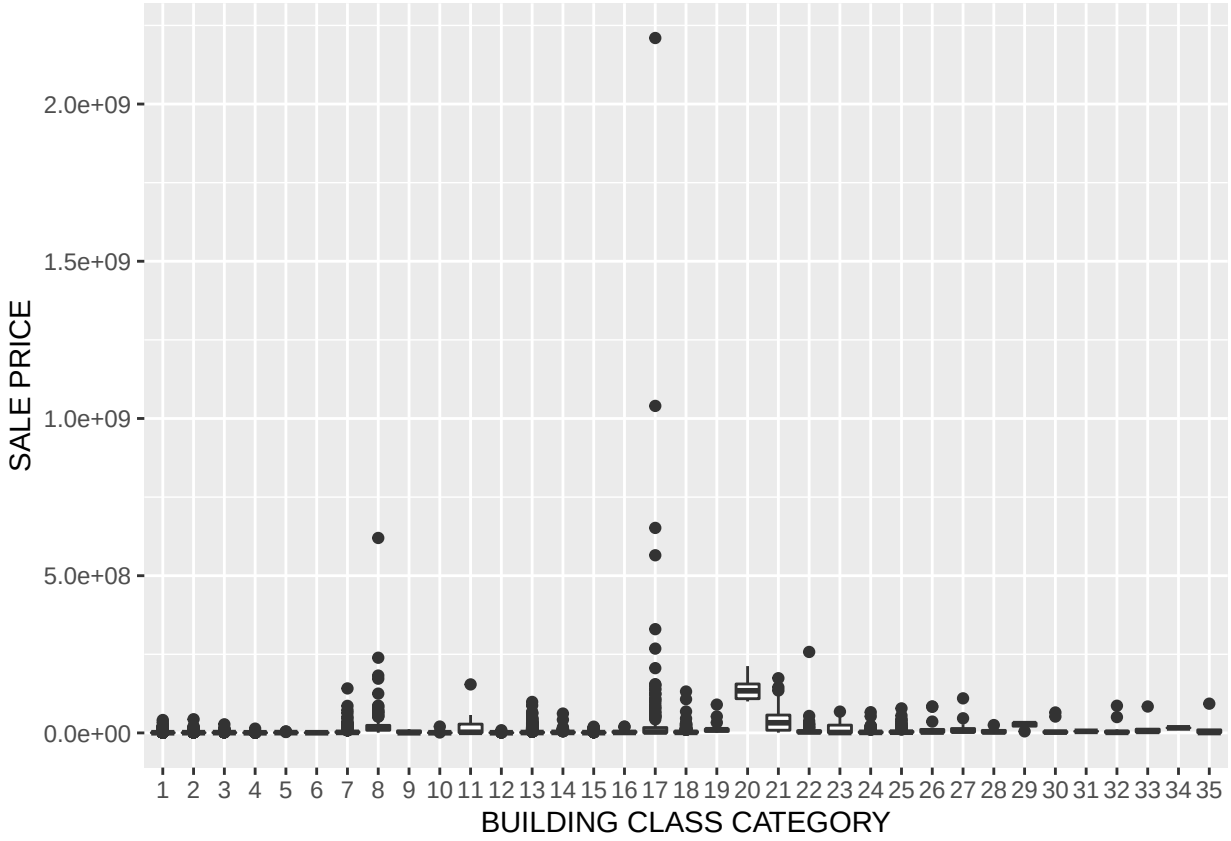
```
# Plot Sale Price vs Gross Square Feet
nyc_house_data %>%
  ggplot(aes(x=GROSS.SQUARE.FEET,y=SALE.PRICE))+
  geom_point()+
  scale_x_continuous(name = "GROSS SQUARE FEET") +
  scale_y_continuous(name = "SALE PRICE")
```


```
# 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(0,1e5), name = "GROSS SQUARE FEET") +
  scale_y_continuous(limits = c(0,1e8), name = "SALE PRICE")
```



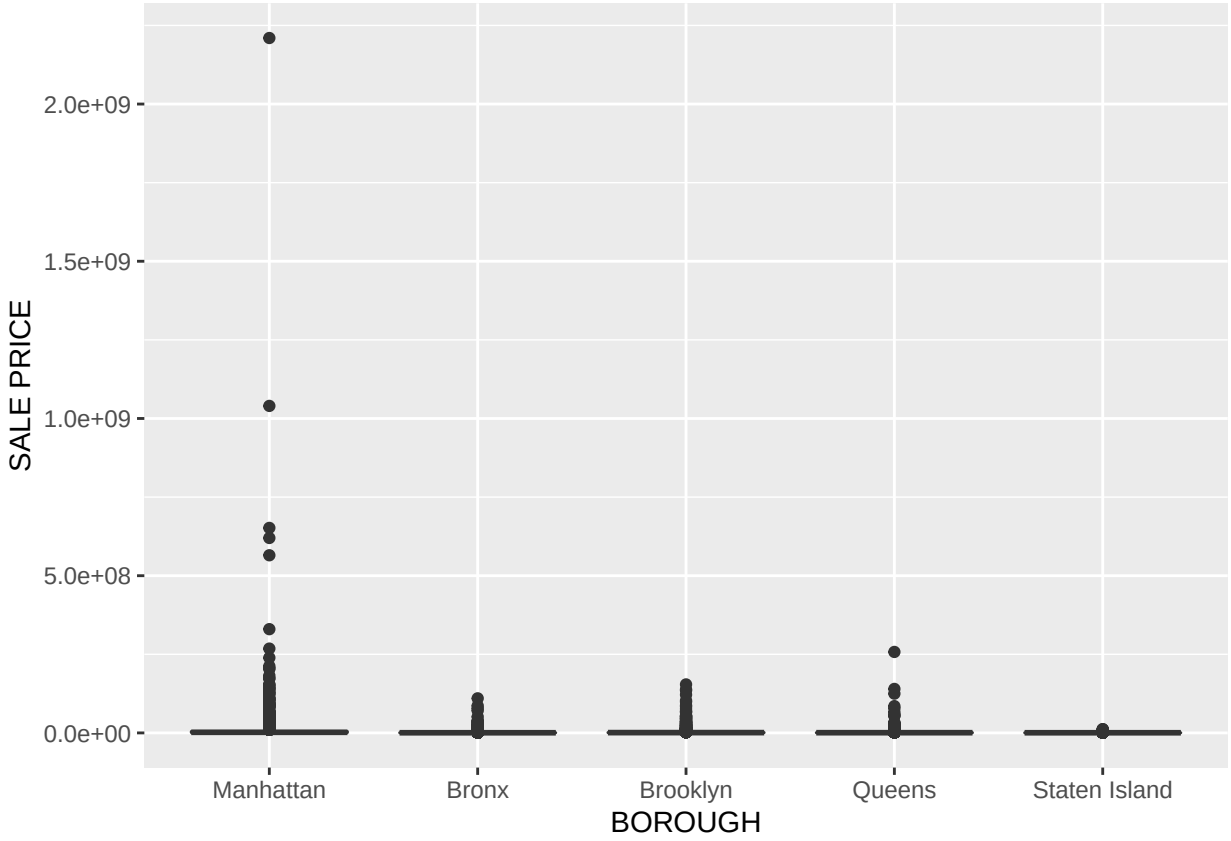
The graph below shows a boxplot of sale prices for each building class category. As demonstrated in the graph, there are multiple outliers in sale prices for each category. The high sale prices appear to be reliable data points, despite the fact that these numbers are statistical outliers.

```
# 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")
```



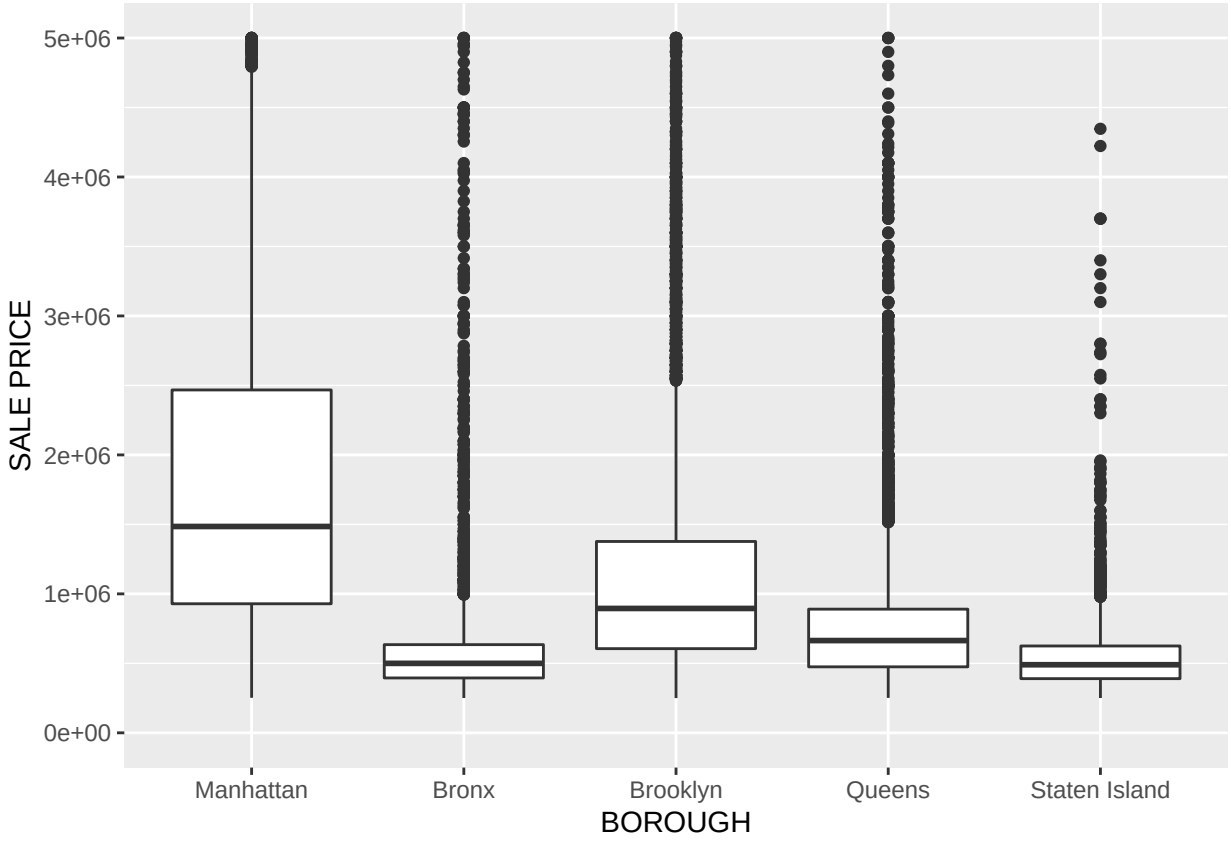
Similarly, the boxplot of sale prices for each borough is shown in the graph below. This graph also reveals that there are numerous outliers in each borough's sale prices. In Manhattan, in particular, there are some extremely expensive property sale prices.

```
# 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 code below is similar to the previous one, however it has a lower sale price limit. The graph demonstrates that Manhattan has the highest average sale prices, while the Bronx and Staten Island have the lowest average sale prices.

```
# 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"))
```

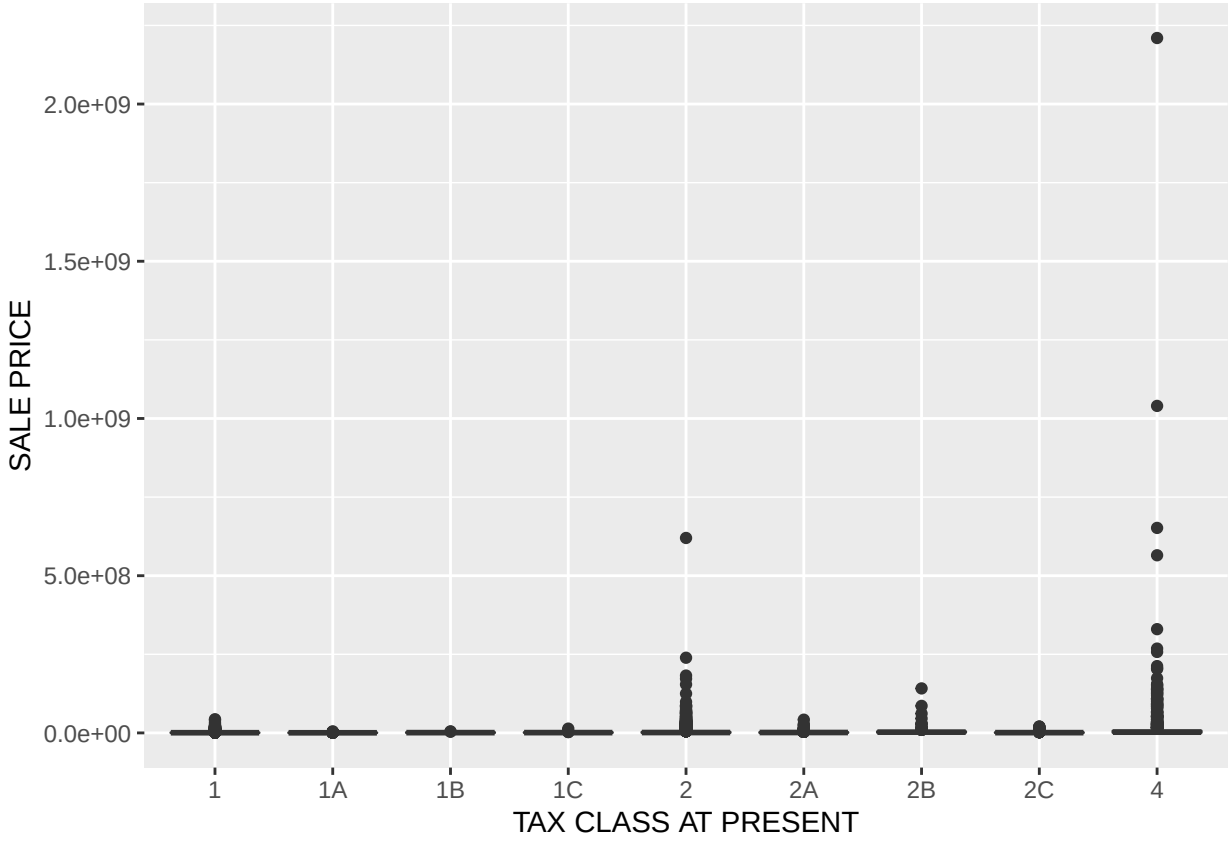


The following code displays the categories of TAX CLASS AT THE MOMENT, as well as the SALE PRICE boxplot for each category. For tax classes 4 and 2, there are entries with extremely high sale prices. This image, like previous plots, indicates that the data contains numerous outliers and that the distribution 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")
```

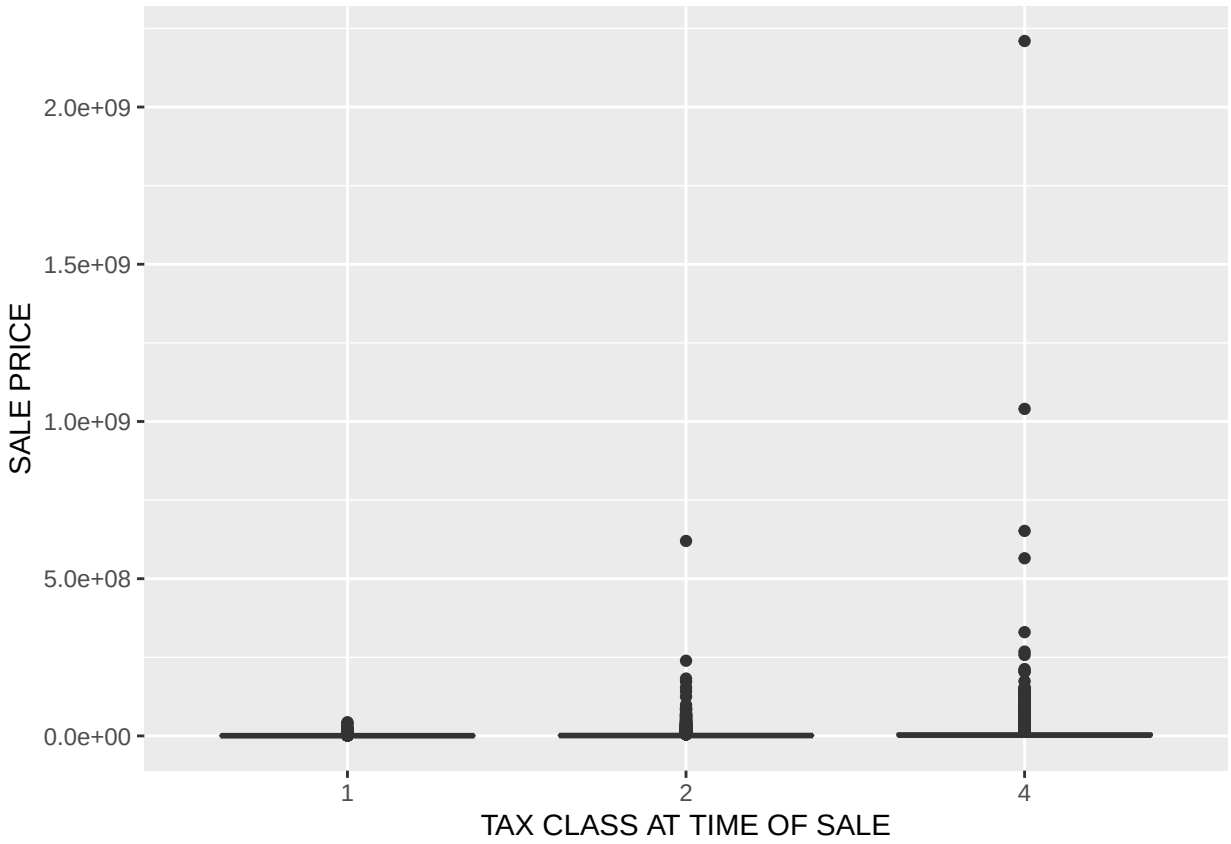


Similarly, the following code shows the categories of TAX CLASS AT TIME OF SALE and shows the boxplot of SALE PRICE for each category.

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

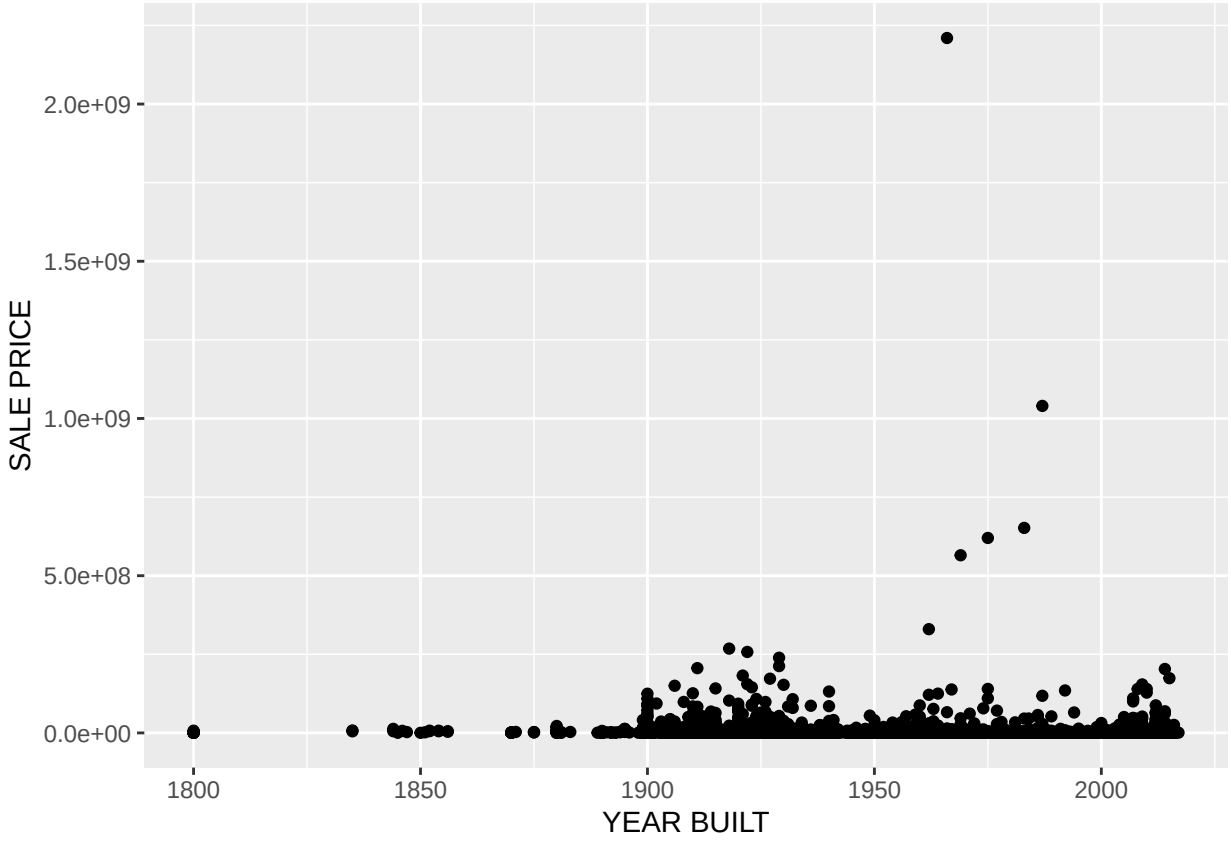
```
## [1] "1" "2" "4"
```

```
# 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")
```

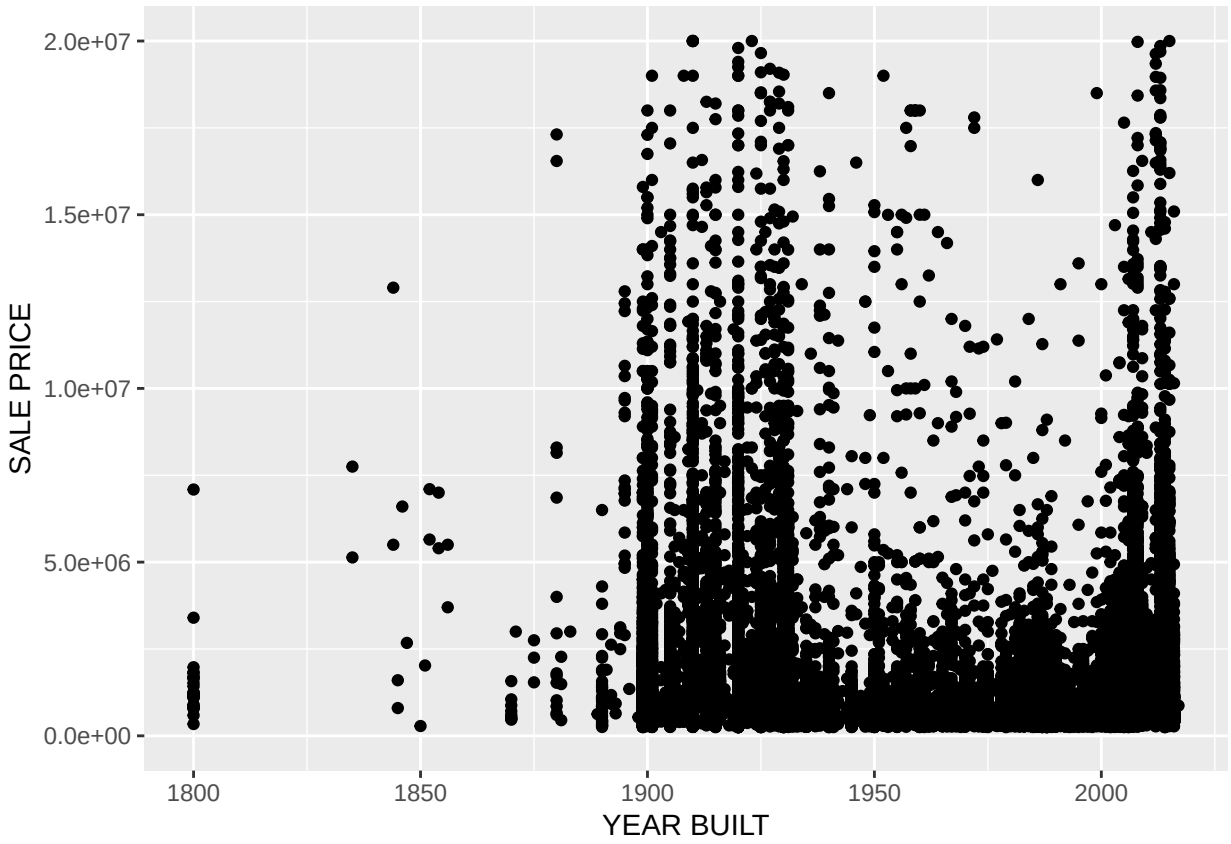


The following plots show Sale Price versus Year Built. It does not seem that there is a strong correlation between the two.

```
# 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")
```

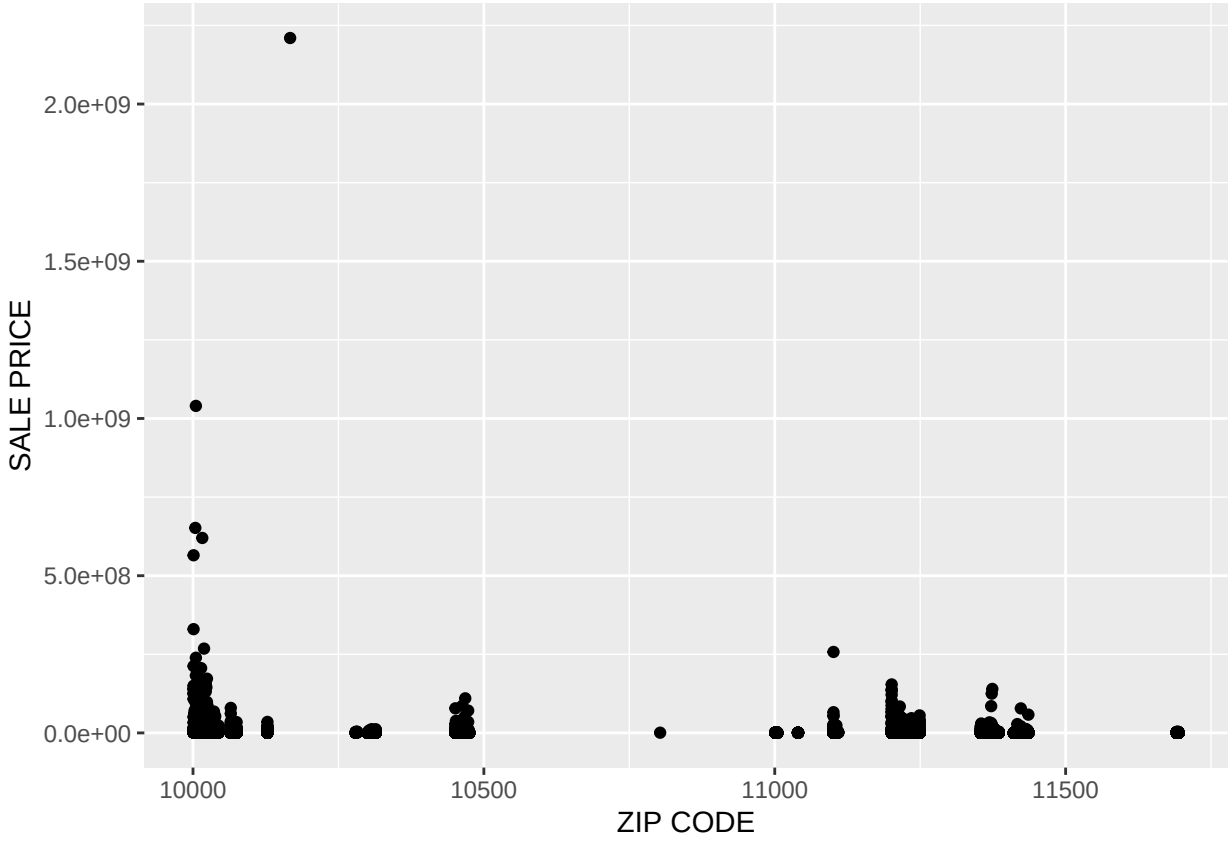



```
# Plot Sale Price vs Year Built with lower sale price limit
nyc_house_data %>%
  ggplot(aes(x=YEAR.BUILT,y=SALE.PRICE))+
  geom_point()+
  scale_x_continuous(name = "YEAR BUILT")+
  scale_y_continuous(name = "SALE PRICE", limit = c(0, 2e7))
```

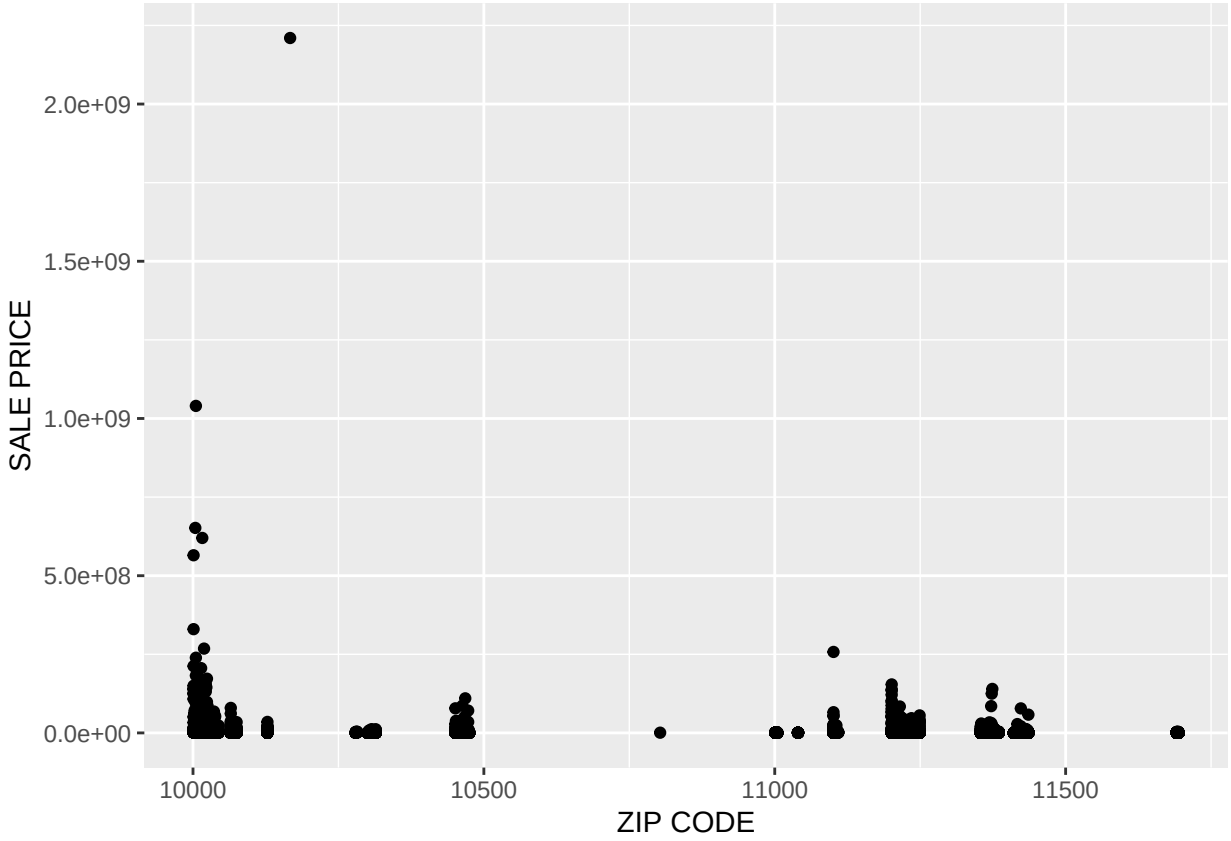


The following plots show Sale Price versus ZIP code. The dots near 10000 is near Manhattan, which has the highest price; the dots near 10500 are in Bronx, the dots between 11000 and 11500 are located in Queens. those are most dense poppulation area.

```
# Plot Sale Price vs ZIP code
nyc_house_data %>%
  ggplot(aes(x=ZIP.CODE,y=SALE.PRICE))+
  geom_point()+
  scale_x_continuous(name = "ZIP CODE")+
  scale_y_continuous(name = "SALE PRICE")
```

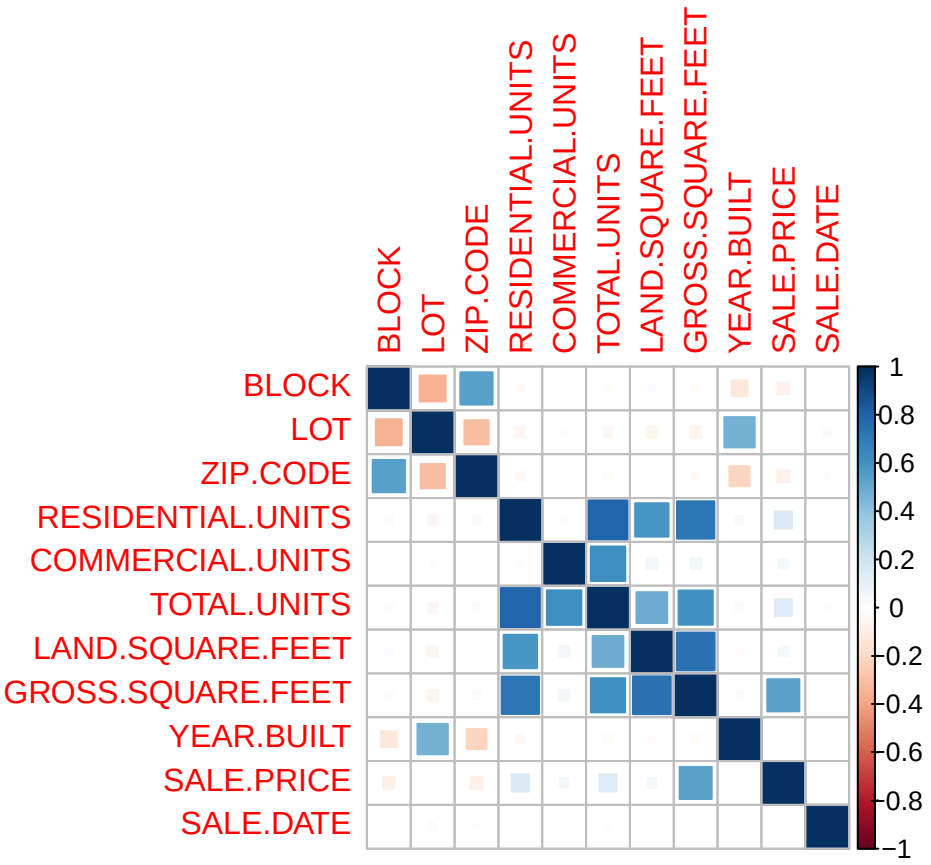



```
# Plot Sale Price vs ZIP code with lower sale price limit
nyc_house_data %>%
  ggplot(aes(x=ZIP.CODE,y=SALE.PRICE))+
  geom_point()+
  scale_x_continuous(name = "ZIP CODE")+
  scale_y_continuous(name = "SALE PRICE")
```



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

Examining several of the numerical variables revealed that they are highly skewed, as previously mentioned. The difficulty with skewed data is that it may lead to the machine learning algorithm being trained on a high number of reasonably priced homes. This might lead to a large mistake in anticipating the prices of pricey homes. One of the most effective approaches for dealing with skewed data is logarithmic transformation. If the skewness is more than 0.75, the following code determines the skewness of all numerical variables except four and performs Logarithmic transformation. Because they contain 0 entries, Residential Units, Commercial Units, Land Square Feet, and Gross Square Feet are not changed.

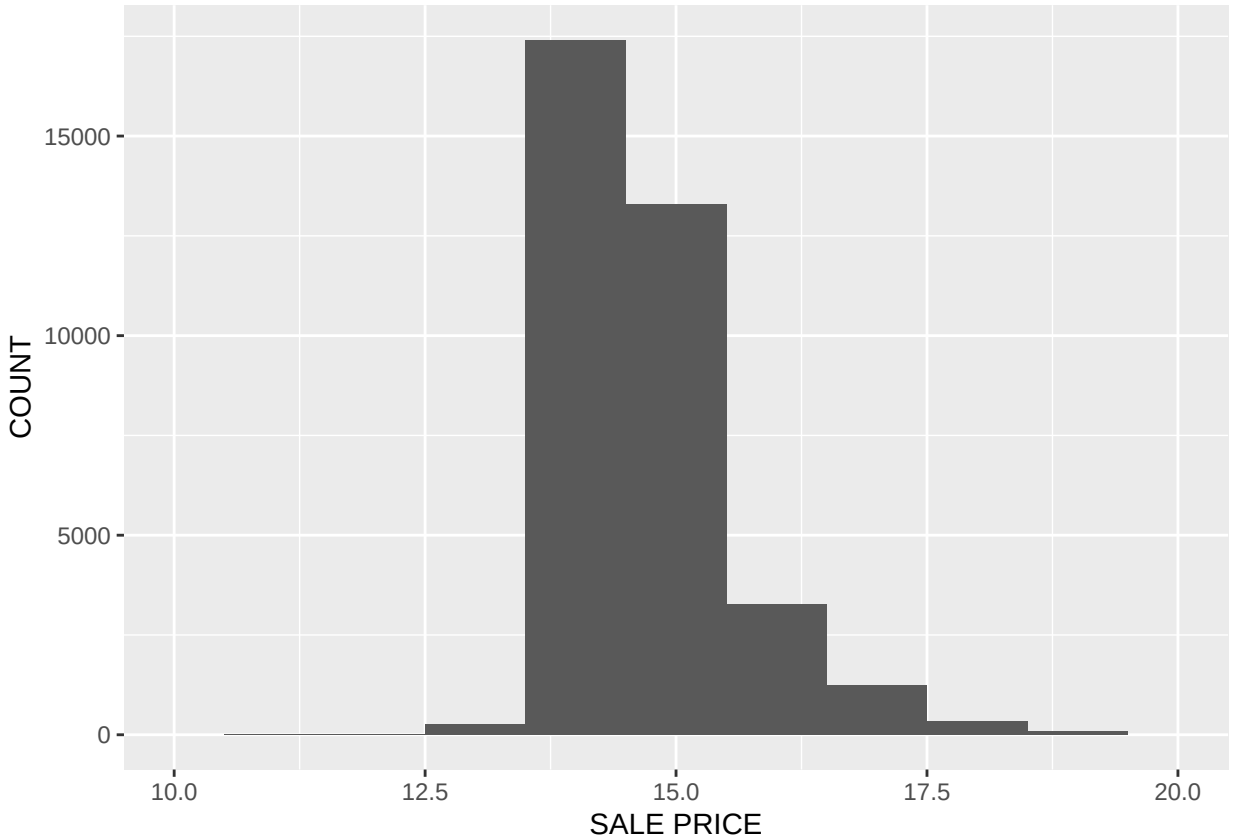
```
# # Apply log transformation to Numerical Variable with skewness > 0.75
for(x in seq(1,ncol(nyc_house_data)))
{
  if(is.numeric(nyc_house_data[,x]))
  {
    skew <- skewness(nyc_house_data[,x],na.rm = T)
    if (skew > 0.75 &
        names(nyc_house_data[x]) != "RESIDENTIAL.UNITS" &
        names(nyc_house_data[x]) != "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 histogram of selling prices following the log transformation is shown in the graph below. The right-skewed distribution of selling prices remains after transformation, although the skewness decreases.

```
# 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")
```



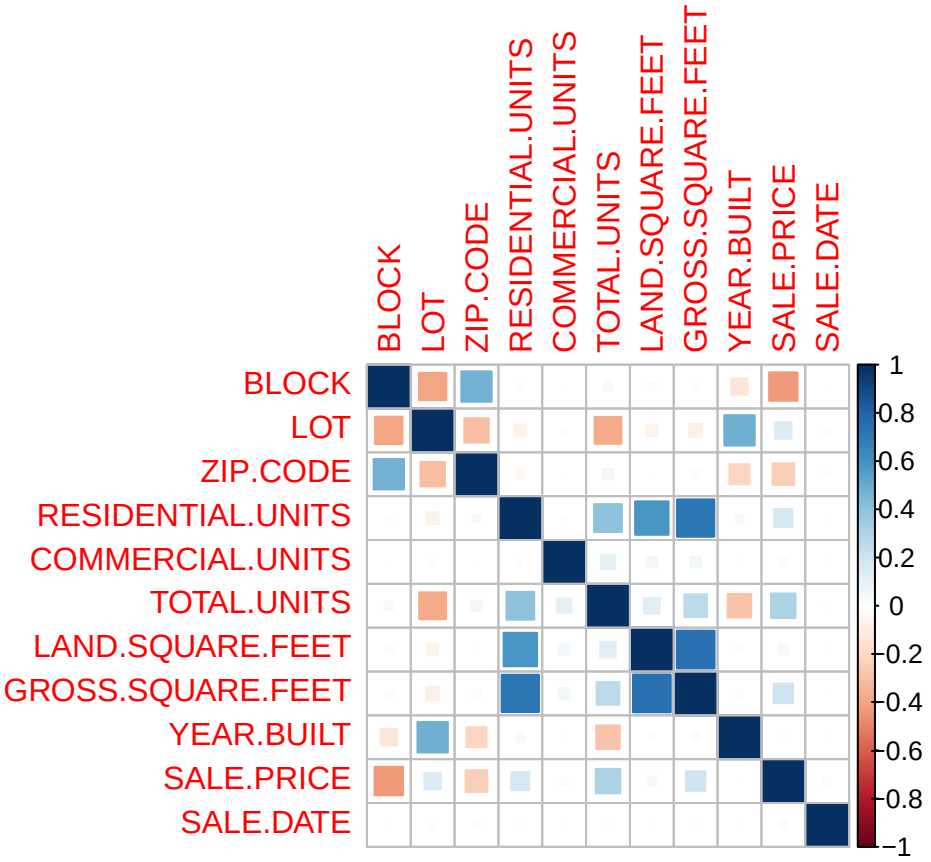
The skewness of Sale Price is calculated using the code below. It confirms that Sale Price's skewness is much lower than its skewness prior to transformation.

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

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

The correlation of numerical variables after the log transformation is seen in the graph below. The graphic illustrates that the association between sale price and total units improves after transformation. In addition, there appears to be a substantial negative relationship between Sale Price and Block. The relationship between Sale Price and Gross Square Feet, on the other hand, is weaker than it was before the transition.

```
# 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 uses the nyc house data dataset to build the training and test datasets. The test set is chosen to be 10% of the original dataset. This was used as an example of a common test set size. The relative size of the train and test sets has no meaningful influence on the performance of the machine learning model, given the size of the dataset (almost 35000 instances). The models are also trained via cross validation.

```
# 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 RandomForest package's random forest implementation can deal with categorical variables with up to 53 levels. The categorical variables with more than 53 levels are printed using the following code.

```
# 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
```

Because the Random Forest implementation in R cannot accept category variables with more than 53 levels, the following code eliminates them.

```
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
                                              ))
```

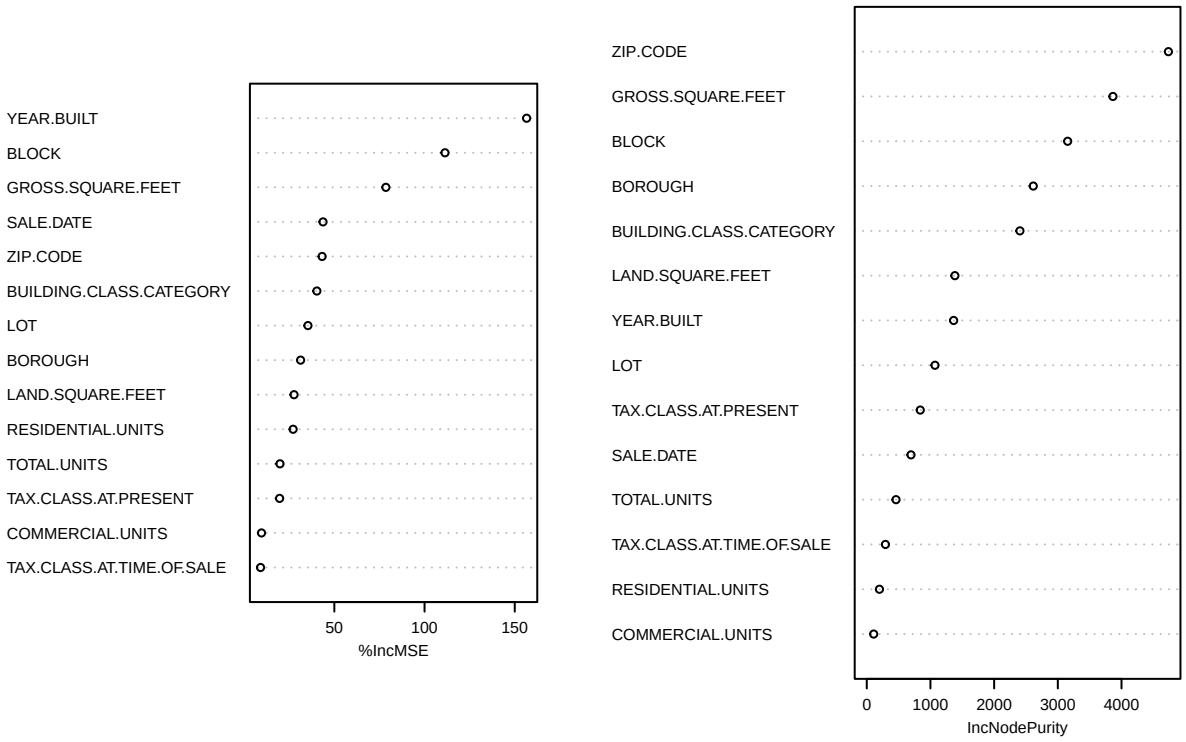
The code below trains the Random Forest model, forecasts sale prices, and compares them to the test set prices. If is tune is set to 1, the code will alter the mtry parameter, which is the number of variables randomly sampled at each split, to tune the algorithm. The code examines the following mtry values: 2, 3, 4, 5, and 6. If is tune is set to 0, the model will be trained using the user-supplied mtry input parameter. The random forest's variable significance chart is likewise plotted by the code.

```
# 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.

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

```
## [1] "Run time is"      "5.56241971439785"
```

3 Results

The algorithms Random Forest and Linear Regression were employed to forecast the sale values of NYC properties. RandomForest performs better than Linear Regression. RandomForest has an RMSE of 3.78e+06 for selling price prediction, but Linear Regression has an RMSE of 4.17e+06.

To identify the best parameter for this application, the RandomForest method was tweaked. The number of variables randomly picked as candidates at each split is the mtry tuning parameter of RandomForest. The square root of the number of predictors is usually a good place to start when tweaking mtry. This figure is nearly 4 in this application (3.87). Tuning was done with five different mtry values: 2, 3, 4, 5, and 6. On my machine, tuning the algorithm took roughly 8 hours. The RandomForest method takes longer to train as mtry grows. Due to computational constraints, the maximum value of mtry examined in this research was restricted to 6.

The RandomForest technique requires much more computer time to train than Linear Regression. The final RandomForest method (mtry = 6) took roughly 2 hours to train, while Linear Regression took about 8 minutes.

4 Conclusion

The purpose of this study is to forecast New York City property selling prices. Random Forest and Linear Regression were employed as machine learning techniques.

The Random Forest model performs better. This is because Random Forest is resistant to overfitting, whereas Linear Regression without regularization can become excessively flexible, especially for datasets with a lot of outliers, like the one employed in this study.

To clean the Kaggle dataset, many processes were used. The dataset's sale prices, as well as several of the other numerical variables, are highly skewed. To increase the model's performance, log transformation was applied to convert these variables. The dataset also contains a large number of outliers. Entries with extremely low prices were excluded from the dataset since they did not appear to be realistic. However, extremely expensive properties were preserved because they appeared to be possible, despite the fact that they are statistical outliers.

Other machine learning methods can be explored in the future to see if they can produce better results. Gradient boosting machine, lasso regression, and other techniques have been utilized in the literature to solve comparable challenges.

Examining and modifying items with Gross Square Feet and/or Land Square Feet of 0 or blank may also be beneficial. This will need real estate understanding, such as the various building classifications. Some of the entries may be incorrect, and eliminating them from the model might enhance its 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.

6- tax class, https://www1.nyc.gov/site/finance/taxes/definitions-of-property-assessment-terms.page