NYC Properties

Vivien Leung

May 21, 2019

Introduction

New York City is a well known to be one of the most expensive cities in the world to buy real estate. In this project, we will use a year's (September 2016 to September 2017) worth of data provided by City of New York on properties sold on the NYC real estate market. This dataset is a record of every building or building unit (apartment, etc.) sold in the New York City property market over a 12-month period. The purpose of this project is to predict house prices using these data such as Borough, Neighborhood, Building Class, etc. We will explore the correlations, and clean and isolate any potential outliers. After doing so, the data will be fitted with different models such as Linear Model, Random Forest, Ridge Regression, Lasso Regression, Elastic Net, GBM. Let's begin!

Analysis

First we will import the following libraries:

```
if(!require(data.table)) install.packages("data.table")
if(!require(tidyr)) install.packages("tidyr")
if(!require(dplyr)) install.packages("dplyr")
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(ggplot2)) install.packages("ggplot2",dependencies=TRUE)
if(!require(lubridate)) install.packages("lubridate")
if(!require(corrplot)) install.packages("corrplot")
if(!require(caret)) install.packages("caret")
if(!require(gbm)) install.packages("gbm")
if(!require(glmnet)) install.packages("glmnet")
if(!require(MASS)) install.packages("MASS")
if(!require(randomForest)) install.packages("randomForest")

library(data.table)
library(tidyr)
library(dplyr)
library(tidyverse)
library(ggplot2)
library(lubridate)
library(corrplot)
library(caret)
library(gbm)
library(glmnet)
library(MASS)
library(Rborist)
library(randomForest)
```

Now we will download the data from GitHub. This is originally from Kaggle but have uploaded to my GitHub account for easier access.

```
download.file("https://github.com/vleung5/NYCPropertySales/raw/master/nyc-property-sales.zip",
              destfile="nyc-property-sales.zip")
```

```
unzip("nyc-property-sales.zip","nyc-rolling-sales.csv")
nyc_prop <- as_data_frame(fread("nyc-rolling-sales.csv"))
```

Looking at this data, we can see it is a 84548 x 22 data frame and we can see the columns as follows:

```
class(nyc_prop)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
dim(nyc_prop)
```

```
## [1] 84548    22
```

```
names(nyc_prop)
```

```
##  [1] "V1"                              "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"
```

We can take a look at the first 6 rows:

head(nyc_prop, 6)

Data Cleaning

Let's start cleaning the data. From the above, we can tell 'V1 'and 'EASE-MENT' are observation numbers or have alot of NULL values, and are not needed. We can remove these. We can also separate the title of building class category from the category number.

```
nyc_property <- as_data_frame(nyc_prop[,-1])
nyc_property <- nyc_property[,c(-7)]

# Separated the columns in 'BUILDING CLASS CATEGORY'
nyc_property <- nyc_property %>%
  separate(col = "BUILDING CLASS CATEGORY", into = c("BUILDING CLASS CATEGORY NUMBER",
                                                     "BUILDING CLASS CATEGORY"), sep = 3) %>%
  separate(col = "SALE DATE", into = c("SALE DATE", "TIME"), sep = " ")
```

There are also duplicate data as well. We can dedupe these.

```
nyc_property %>% filter(duplicated(nyc_property) == TRUE) %>% nrow()
```

```
## [1] 765
```

```
head(nyc_property %>% filter(duplicated(nyc_property) == TRUE))
```

```
## # A tibble: 6 x 22
##   BOROUGH NEIGHBORHOOD `BUILDING CLASS~ `BUILDING CLASS~ `TAX CLASS AT P~
##     <int> <chr>        <chr>            <chr>            <chr>
## 1       1 CHELSEA      "01 "            ONE FAMILY DWEL~ 1
## 2       1 CHELSEA      "01 "            ONE FAMILY DWEL~ 1
```

```
## 3       1 CHELSEA      "10 "            COOPS - ELEVATO~ 2
## 4       1 CHELSEA      "13 "            CONDOS - ELEVAT~ 2
## 5       1 CIVIC CENTER "46 "            CONDO STORE BUI~ 4
## 6       1 CIVIC CENTER "46 "            CONDO STORE BUI~ 4
## # ... with 17 more variables: BLOCK <int>, LOT <int>, `BUILDING CLASS AT
## #   PRESENT` <chr>, ADDRESS <chr>, `APARTMENT NUMBER` <chr>, `ZIP
## #   CODE` <int>, `RESIDENTIAL UNITS` <int>, `COMMERCIAL UNITS` <int>,
## #   `TOTAL UNITS` <int>, `LAND SQUARE FEET` <chr>, `GROSS SQUARE
## #   FEET` <chr>, `YEAR BUILT` <int>, `TAX CLASS AT TIME OF SALE` <int>,
## #   `BUILDING CLASS AT TIME OF SALE` <chr>, `SALE PRICE` <chr>, `SALE
## #   DATE` <chr>, TIME <chr>
```

```
nyc_property <- unique(nyc_property)
```

Let's take a look at the data again:

```
lapply(nyc_property, class)
```

```
## $BOROUGH
## [1] "integer"
##
## $NEIGHBORHOOD
## [1] "character"
##
## $`BUILDING CLASS CATEGORY NUMBER`
## [1] "character"
##
## $`BUILDING CLASS CATEGORY`
## [1] "character"
##
## $`TAX CLASS AT PRESENT`
## [1] "character"
##
## $BLOCK
## [1] "integer"
##
## $LOT
## [1] "integer"
##
## $`BUILDING CLASS AT PRESENT`
## [1] "character"
##
## $ADDRESS
## [1] "character"
##
## $`APARTMENT NUMBER`
## [1] "character"
##
## $`ZIP CODE`
## [1] "integer"
##
## $`RESIDENTIAL UNITS`
## [1] "integer"
##
## $`COMMERCIAL UNITS`
## [1] "integer"
##
## $`TOTAL UNITS`
```

```
## [1] "integer"
##
## $`LAND SQUARE FEET`
## [1] "character"
##
## $`GROSS SQUARE FEET`
## [1] "character"
##
## $`YEAR BUILT`
## [1] "integer"
##
## $`TAX CLASS AT TIME OF SALE`
## [1] "integer"
##
## $`BUILDING CLASS AT TIME OF SALE`
## [1] "character"
##
## $`SALE PRICE`
## [1] "character"
##
## $`SALE DATE`
## [1] "character"
##
## $TIME
## [1] "character"
```

```
names(nyc_property)
```

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

```
head(nyc_property)
```

```
## # A tibble: 6 x 22
##   BOROUGH NEIGHBORHOOD `BUILDING CLASS~ `BUILDING CLASS~ `TAX CLASS AT P~
##     <int> <chr>        <chr>            <chr>            <chr>
## 1       1 ALPHABET CI~ "07 "            RENTALS - WALKU~ 2A
## 2       1 ALPHABET CI~ "07 "            RENTALS - WALKU~ 2
## 3       1 ALPHABET CI~ "07 "            RENTALS - WALKU~ 2
## 4       1 ALPHABET CI~ "07 "            RENTALS - WALKU~ 2B
## 5       1 ALPHABET CI~ "07 "            RENTALS - WALKU~ 2A
## 6       1 ALPHABET CI~ "07 "            RENTALS - WALKU~ 2
## # ... with 17 more variables: BLOCK <int>, LOT <int>, `BUILDING CLASS AT
## #   PRESENT` <chr>, ADDRESS <chr>, `APARTMENT NUMBER` <chr>, `ZIP
## #   CODE` <int>, `RESIDENTIAL UNITS` <int>, `COMMERCIAL UNITS` <int>,
## #   `TOTAL UNITS` <int>, `LAND SQUARE FEET` <chr>, `GROSS SQUARE
## #   FEET` <chr>, `YEAR BUILT` <int>, `TAX CLASS AT TIME OF SALE` <int>,
## #   `BUILDING CLASS AT TIME OF SALE` <chr>, `SALE PRICE` <chr>, `SALE
## #   DATE` <chr>, TIME <chr>
```

We will need to convert some data types to be more suitable for analyses. We can convert Borough, Neighborhood, Building Class Category Number, Building Class Category, Building Class At Present, Zip Code, Tax Class At Time Of Sale, and Building Class At Time Of Sale into factors. Land Square Feet, Gross Square Feet, Year Built, and Sale Price can be converted to numeric. Block and Lot will be converted to characters. Converting fields to numeric gave a warning:

Warning in evalq(as.numeric(`LAND SQUARE FEET`),) :

NAs introduced by coercion

In this case, we will need to clean up these NA in a later step.

I have also separated out the Year and Month from the sale date as this can be significant because there can be a surge in the market for particular months/seasons.

```
fac <- c(1,2,3,4,5,8,11,18,19)
nyc_property <- nyc_property %>% mutate_at(fac, funs(factor(.)))
levels(nyc_property$BOROUGH) <- c("Manhattan", "Bronx", "Brooklyn", "Queens", "Staten Island")

num <- c(15,16,17,20)
nyc_property <- nyc_property %>% mutate_at(num, funs(as.numeric(.)))
```

```
## Warning in evalq(as.numeric(`LAND SQUARE FEET`), <environment>): NAs
## introduced by coercion
```

```
## Warning in evalq(as.numeric(`GROSS SQUARE FEET`), <environment>): NAs
## introduced by coercion
```

```
## Warning in evalq(as.numeric(`SALE PRICE`), <environment>): NAs introduced
## by coercion
```

```
chr <- c(6,7)
nyc_property  <- nyc_property %>% mutate_at(chr, funs(as.character(.)))

nyc_property$'SALE DATE' <- ymd(nyc_property$'SALE DATE')
nyc_property$'SALE YEAR' <- as.factor(format(nyc_property$'SALE DATE',"%Y"))
nyc_property$'SALE MONTH' <- as.factor(format(nyc_property$'SALE DATE',"%m"))
```

Now let's take care of the NAs that were introducted from converting the data types above and check for 0 values as well.

```
colSums(is.na(nyc_property))
```

```
##                         BOROUGH                    NEIGHBORHOOD
##                               0                               0
## BUILDING CLASS CATEGORY NUMBER         BUILDING CLASS CATEGORY
##                               0                               0
##            TAX CLASS AT PRESENT                           BLOCK
##                               0                               0
##                             LOT       BUILDING CLASS AT PRESENT
##                               0                               0
##                         ADDRESS                APARTMENT NUMBER
##                               0                               0
##                        ZIP CODE              RESIDENTIAL UNITS
##                               0                               0
##                COMMERCIAL UNITS                    TOTAL UNITS
##                               0                               0
##                LAND SQUARE FEET              GROSS SQUARE FEET
##                           26054                           27385
```

```
##                      YEAR BUILT      TAX CLASS AT TIME OF SALE
##                               0                              0
## BUILDING CLASS AT TIME OF SALE                     SALE PRICE
##                               0                          14176
##                       SALE DATE                           TIME
##                               0                              0
##                       SALE YEAR                     SALE MONTH
##                               0                              0
```

```
colSums(nyc_property == 0)
```

```
##                          BOROUGH                     NEIGHBORHOOD
##                                0                                0
## BUILDING CLASS CATEGORY NUMBER          BUILDING CLASS CATEGORY
##                                0                                0
##            TAX CLASS AT PRESENT                            BLOCK
##                                0                                0
##                             LOT       BUILDING CLASS AT PRESENT
##                                0                                0
##                         ADDRESS                 APARTMENT NUMBER
##                                0                                1
##                        ZIP CODE                RESIDENTIAL UNITS
##                             971                            24546
##                COMMERCIAL UNITS                     TOTAL UNITS
##                           78777                            19677
##                LAND SQUARE FEET                GROSS SQUARE FEET
##                               NA                               NA
##                      YEAR BUILT      TAX CLASS AT TIME OF SALE
##                             6885                              0
## BUILDING CLASS AT TIME OF SALE                     SALE PRICE
##                                0                               NA
##                       SALE DATE                           TIME
##                                0                              0
##                       SALE YEAR                     SALE MONTH
##                                0                              0
```

```
sum(is.na(nyc_property$'SALE PRICE'))
```

```
## [1] 14176
```

We will remove NAs data. This is important as we will get the following error later:

> Warning in storage.mode(v) <- "double" : NAs introduced by coercion Error in lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) : NA/NaN/Inf in 'y'

```
nyc_property %>% filter(nyc_property$'SALE PRICE' == 0 | is.na(nyc_property$'SALE PRICE')) %>%
  nrow()
```

```
## [1] 24188
```

```
#Remove rows with no sale price
nyc_property <- nyc_property %>% filter((!nyc_property$'SALE PRICE' == 0)
                                        & !is.na(nyc_property$'SALE PRICE'))


#Do the same with land square feet, gross square feet, etc
nyc_property <- nyc_property %>% filter((!nyc_property$'LAND SQUARE FEET' == 0)
                                        & !is.na(nyc_property$'LAND SQUARE FEET'))

nyc_property <- nyc_property %>% filter((!nyc_property$'GROSS SQUARE FEET' == 0)
                                        & !is.na(nyc_property$'GROSS SQUARE FEET'))
```

```
nyc_property <- nyc_property %>% filter((!nyc_property$'ZIP CODE' == 0)
                                        & !is.na(nyc_property$'ZIP CODE'))

nyc_property <- nyc_property %>% filter((!nyc_property$'TOTAL UNITS' == 0)
                                        & !is.na(nyc_property$'TOTAL UNITS'))

nyc_property <- nyc_property %>% filter((!nyc_property$'YEAR BUILT' == 0)
                                        & !is.na(nyc_property$'YEAR BUILT'))
```

Let's get the building age instead of the year built, it is more practical and clearer to understand.

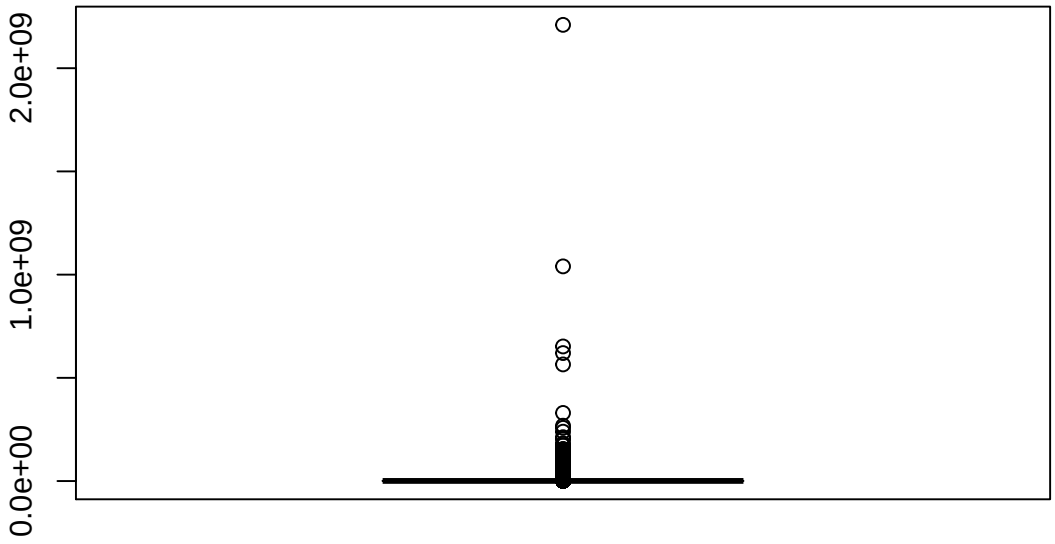
```
nyc_property <- nyc_property %>%
  mutate('BUILDING AGE' = 2017 - nyc_property$'YEAR BUILT')
```

We'll remove columns we don't need

```
nyc_property <- subset(nyc_property, select = -c(which(names(nyc_property) %in% c("YEAR BUILT"))))
nyc_property <- subset(nyc_property, select = -c(which(names(nyc_property) %in% c("SALE DATE"))))
nyc_property <- subset(nyc_property, select = -c(which(names(nyc_property) %in% c("TIME"))))
nyc_property <- subset(nyc_property, select = -c(which(names(nyc_property) %in%
                                                  c("ADDRESS","APARTMENT NUMBER"))))
```

Let's take a look at outliers for sale price and we can filter them out.

```
boxplot(nyc_property$'SALE PRICE')
```



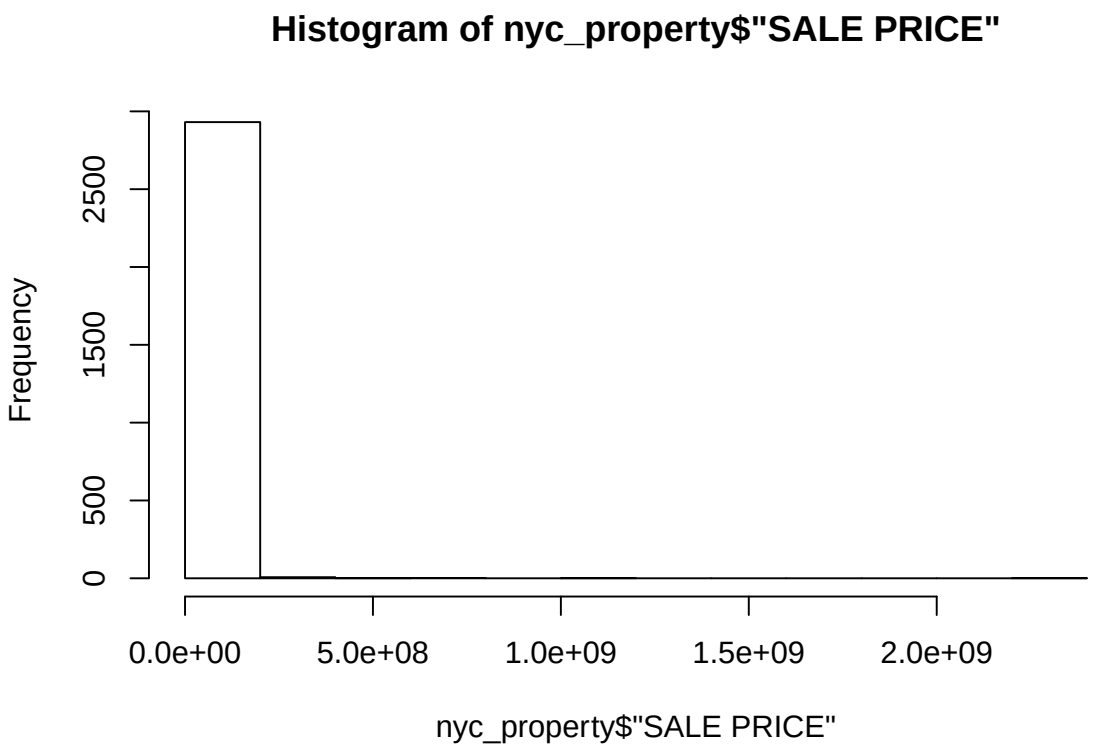

```
outliers = boxplot(nyc_property$'SALE PRICE', plot=FALSE)$out
nyc_property[nyc_property$'SALE PRICE' %in% outliers,]
```

```
## # A tibble: 2,943 x 20
##    BOROUGH NEIGHBORHOOD `BUILDING CLASS~ `BUILDING CLASS~ `TAX CLASS AT P~
##    <fct>   <fct>        <fct>            <fct>            <fct>
##  1 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2A
##  2 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2B
##  3 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2A
##  4 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2B
##  5 Manhat~ ALPHABET CI~ "08 "            RENTALS - ELEVA~ 2
##  6 Manhat~ ALPHABET CI~ "08 "            RENTALS - ELEVA~ 2B
##  7 Manhat~ ALPHABET CI~ "09 "            COOPS - WALKUP ~ 2
##  8 Manhat~ ALPHABET CI~ "14 "            RENTALS - 4-10 ~ 2A
##  9 Manhat~ ALPHABET CI~ "14 "            RENTALS - 4-10 ~ 2A
## 10 Manhat~ ALPHABET CI~ "14 "            RENTALS - 4-10 ~ 2A
## # ... with 2,933 more rows, and 15 more variables: BLOCK <chr>, LOT <chr>,
## #   `BUILDING CLASS AT PRESENT` <fct>, `ZIP CODE` <fct>, `RESIDENTIAL
## #   UNITS` <int>, `COMMERCIAL UNITS` <int>, `TOTAL UNITS` <int>, `LAND
## #   SQUARE FEET` <dbl>, `GROSS SQUARE FEET` <dbl>, `TAX CLASS AT TIME OF
## #   SALE` <fct>, `BUILDING CLASS AT TIME OF SALE` <fct>, `SALE
## #   PRICE` <dbl>, `SALE YEAR` <fct>, `SALE MONTH` <fct>, `BUILDING
## #   AGE` <dbl>
```

```
nyc_property <- nyc_property %>% filter(nyc_property$'SALE PRICE' %in% outliers )
```

Looking at the histogram of Sale Price, the data is skewed. We will do log transformation so we can get better results. This will be applied to other fields as well. Here are the histograms showing before and after the log transformations.

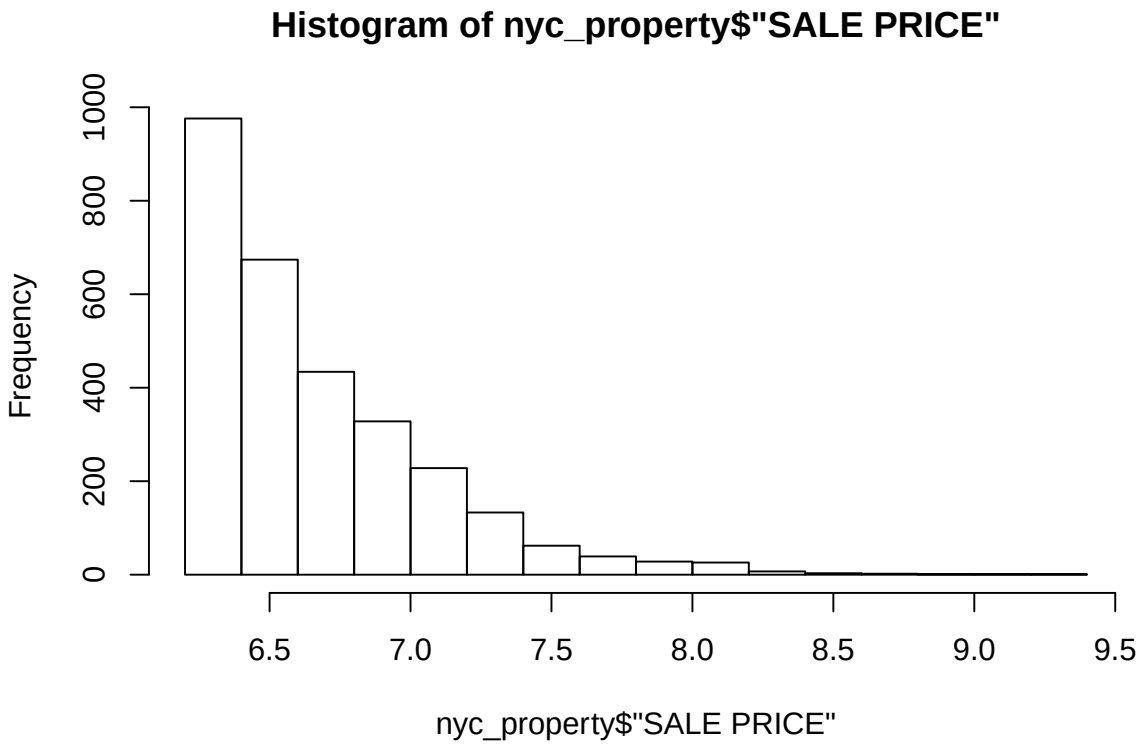
```
hist(nyc_property$'SALE PRICE')
```




```
nyc_property <- nyc_property %>% mutate_at(c(which(names(nyc_property) %in% c("SALE PRICE")))
                                            ,funs(log10(.)))
mean(nyc_property$'SALE PRICE')
```

```
## [1] 6.666753
```

```
hist(nyc_property$'SALE PRICE')
```




```
nyc_property %>% mutate_at(c(which(names(nyc_property) %in%
                                   c("RESIDENTIAL UNITS","COMMERCIAL UNITS","TOTAL UNITS"))),funs(
```

```
## # A tibble: 2,943 x 20
##    BOROUGH NEIGHBORHOOD `BUILDING CLASS~ `BUILDING CLASS~ `TAX CLASS AT P~
##    <fct>   <fct>        <fct>            <fct>            <fct>
##  1 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2A
##  2 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2B
##  3 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2A
##  4 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2B
##  5 Manhat~ ALPHABET CI~ "08 "            RENTALS - ELEVA~ 2
##  6 Manhat~ ALPHABET CI~ "08 "            RENTALS - ELEVA~ 2B
##  7 Manhat~ ALPHABET CI~ "09 "            COOPS - WALKUP ~ 2
##  8 Manhat~ ALPHABET CI~ "14 "            RENTALS - 4-10 ~ 2A
##  9 Manhat~ ALPHABET CI~ "14 "            RENTALS - 4-10 ~ 2A
## 10 Manhat~ ALPHABET CI~ "14 "            RENTALS - 4-10 ~ 2A
## # ... with 2,933 more rows, and 15 more variables: BLOCK <chr>, LOT <chr>,
## #   `BUILDING CLASS AT PRESENT` <fct>, `ZIP CODE` <fct>, `RESIDENTIAL
## #   UNITS` <dbl>, `COMMERCIAL UNITS` <dbl>, `TOTAL UNITS` <dbl>, `LAND
## #   SQUARE FEET` <dbl>, `GROSS SQUARE FEET` <dbl>, `TAX CLASS AT TIME OF
## #   SALE` <fct>, `BUILDING CLASS AT TIME OF SALE` <fct>, `SALE
## #   PRICE` <dbl>, `SALE YEAR` <fct>, `SALE MONTH` <fct>, `BUILDING
## #   AGE` <dbl>
```

```
nyc_property %>% mutate_at(c(which(names(nyc_property) %in%
                                   c("LAND SQUARE FEET", "GROSS SQUARE FEET"))),funs(. + 1))
```

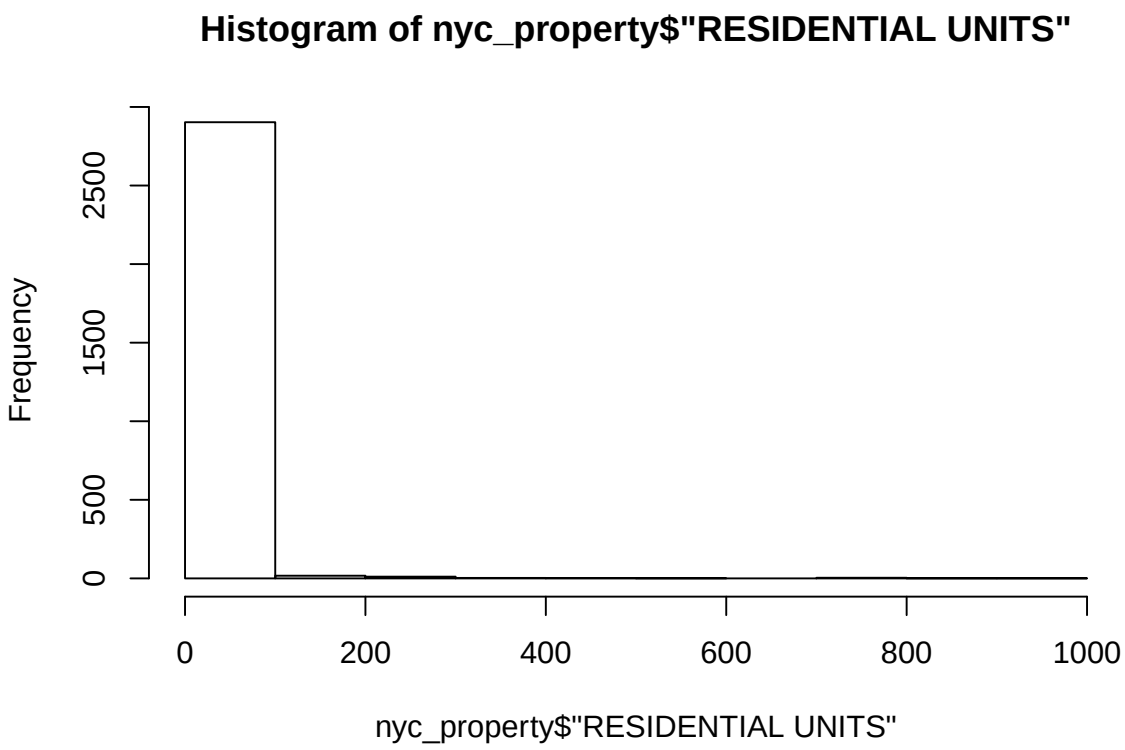
```
## # A tibble: 2,943 x 20
```

```
##    BOROUGH NEIGHBORHOOD `BUILDING CLASS~ `BUILDING CLASS~ `TAX CLASS AT P~
##    <fct>   <fct>        <fct>            <fct>            <fct>
##  1 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2A
##  2 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2B
##  3 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2A
##  4 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2B
##  5 Manhat~ ALPHABET CI~ "08 "            RENTALS - ELEVA~ 2
##  6 Manhat~ ALPHABET CI~ "08 "            RENTALS - ELEVA~ 2B
##  7 Manhat~ ALPHABET CI~ "09 "            COOPS - WALKUP ~ 2
##  8 Manhat~ ALPHABET CI~ "14 "            RENTALS - 4-10 ~ 2A
##  9 Manhat~ ALPHABET CI~ "14 "            RENTALS - 4-10 ~ 2A
## 10 Manhat~ ALPHABET CI~ "14 "            RENTALS - 4-10 ~ 2A
## # ... with 2,933 more rows, and 15 more variables: BLOCK <chr>, LOT <chr>,
## #   `BUILDING CLASS AT PRESENT` <fct>, `ZIP CODE` <fct>, `RESIDENTIAL
## #   UNITS` <int>, `COMMERCIAL UNITS` <int>, `TOTAL UNITS` <int>, `LAND
## #   SQUARE FEET` <dbl>, `GROSS SQUARE FEET` <dbl>, `TAX CLASS AT TIME OF
## #   SALE` <fct>, `BUILDING CLASS AT TIME OF SALE` <fct>, `SALE
## #   PRICE` <dbl>, `SALE YEAR` <fct>, `SALE MONTH` <fct>, `BUILDING
## #   AGE` <dbl>
```

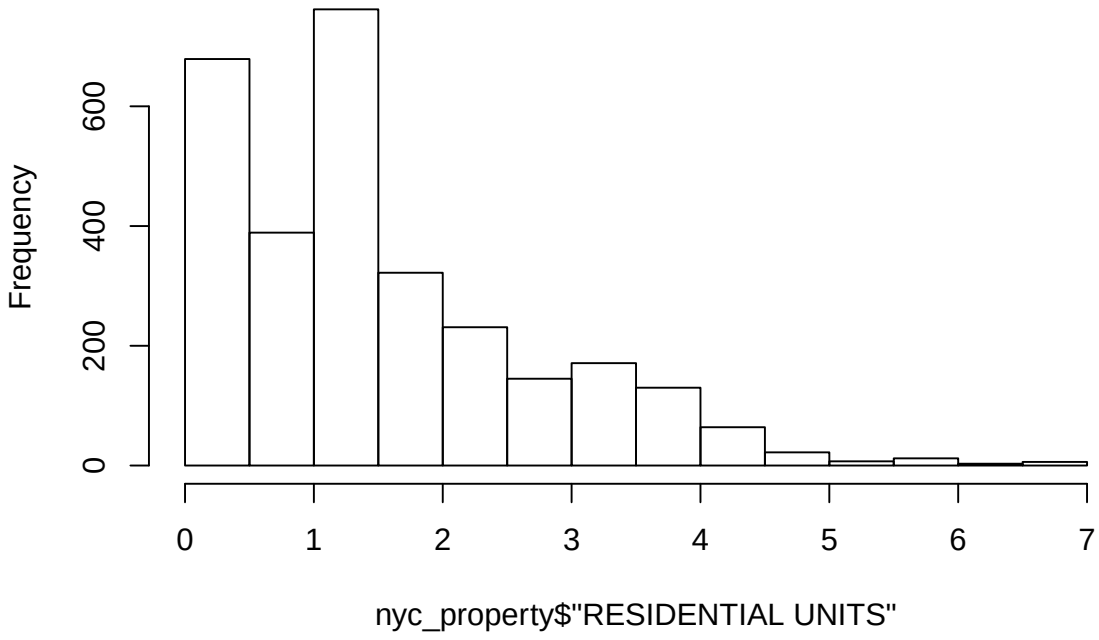
```
nyc_property %>% mutate_at(c(which(names(nyc_property) %in%
                                   c("BUILDING AGE"))),funs(. + 1))
```

```
## # A tibble: 2,943 x 20
##    BOROUGH NEIGHBORHOOD `BUILDING CLASS~ `BUILDING CLASS~ `TAX CLASS AT P~
##    <fct>   <fct>        <fct>            <fct>            <fct>
##  1 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2A
##  2 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2B
##  3 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2A
##  4 Manhat~ ALPHABET CI~ "07 "            RENTALS - WALKU~ 2B
##  5 Manhat~ ALPHABET CI~ "08 "            RENTALS - ELEVA~ 2
##  6 Manhat~ ALPHABET CI~ "08 "            RENTALS - ELEVA~ 2B
##  7 Manhat~ ALPHABET CI~ "09 "            COOPS - WALKUP ~ 2
##  8 Manhat~ ALPHABET CI~ "14 "            RENTALS - 4-10 ~ 2A
##  9 Manhat~ ALPHABET CI~ "14 "            RENTALS - 4-10 ~ 2A
## 10 Manhat~ ALPHABET CI~ "14 "            RENTALS - 4-10 ~ 2A
## # ... with 2,933 more rows, and 15 more variables: BLOCK <chr>, LOT <chr>,
## #   `BUILDING CLASS AT PRESENT` <fct>, `ZIP CODE` <fct>, `RESIDENTIAL
## #   UNITS` <int>, `COMMERCIAL UNITS` <int>, `TOTAL UNITS` <int>, `LAND
## #   SQUARE FEET` <dbl>, `GROSS SQUARE FEET` <dbl>, `TAX CLASS AT TIME OF
## #   SALE` <fct>, `BUILDING CLASS AT TIME OF SALE` <fct>, `SALE
## #   PRICE` <dbl>, `SALE YEAR` <fct>, `SALE MONTH` <fct>, `BUILDING
## #   AGE` <dbl>
```

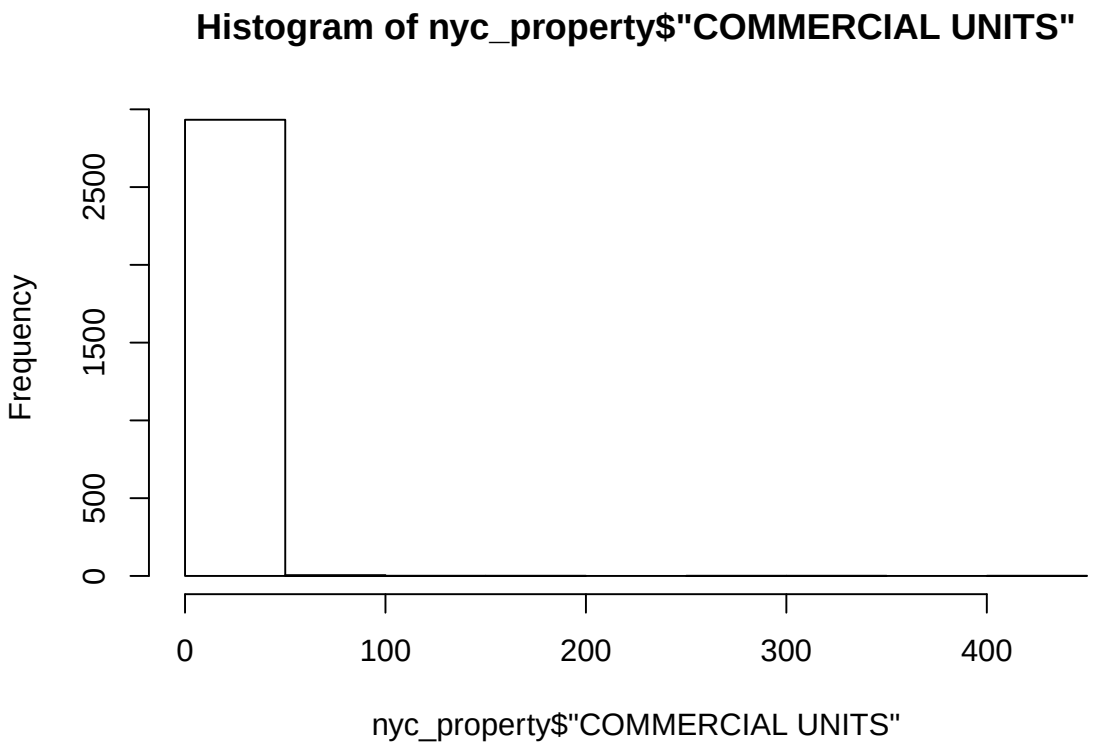
```
hist(nyc_property$'RESIDENTIAL UNITS')
```



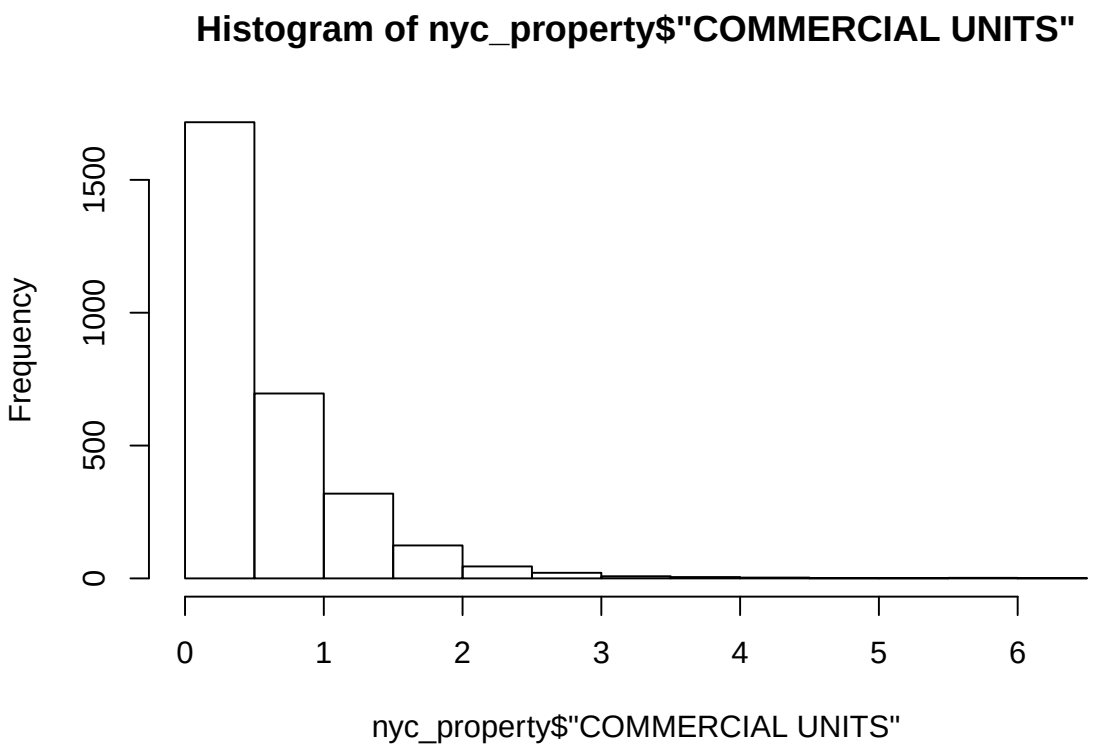

```
nyc_property <- nyc_property %>% mutate_at(c(which(names(nyc_property) %in%
                                                    c("RESIDENTIAL UNITS"))),funs(log1p(.)))
hist(nyc_property$'RESIDENTIAL UNITS')
```



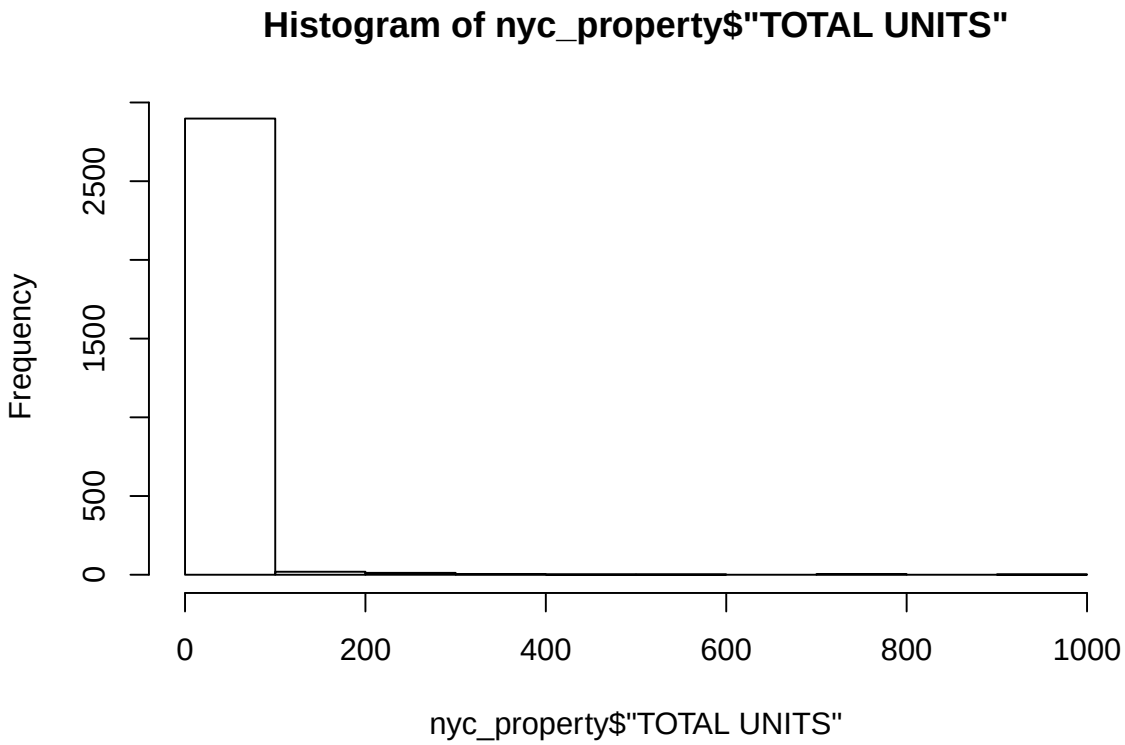
```
hist(nyc_property$'COMMERCIAL UNITS')
```



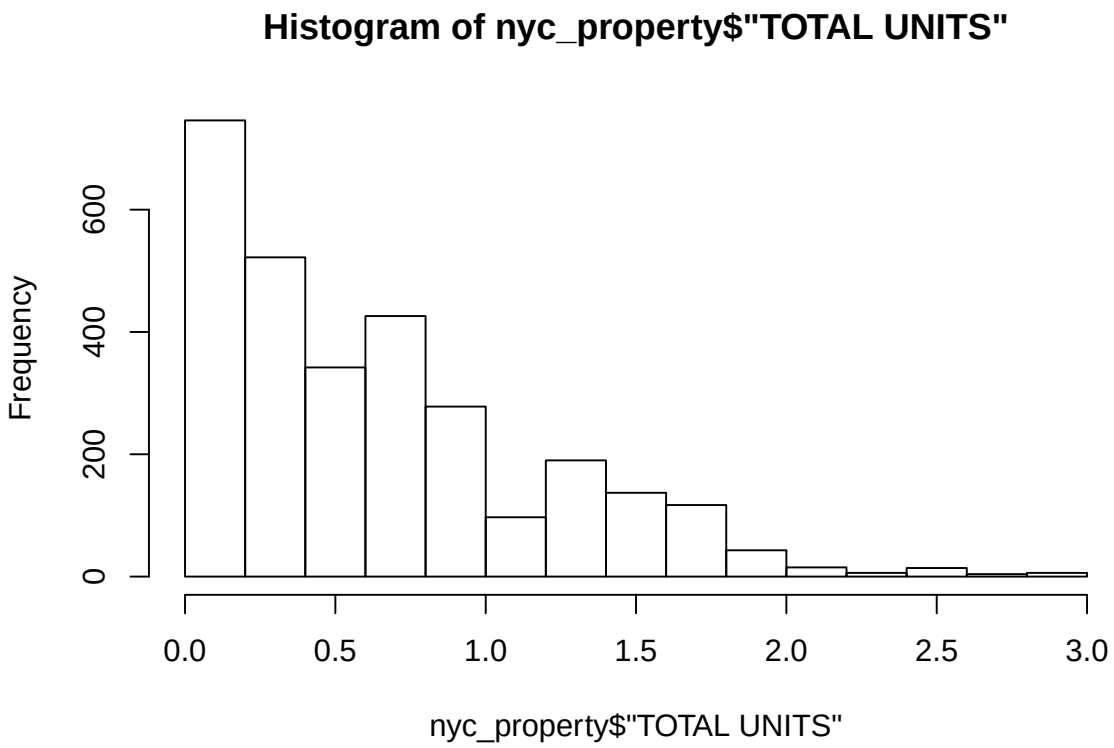

```
nyc_property <- nyc_property %>% mutate_at(c(which(names(nyc_property) %in%
                                                    c("COMMERCIAL UNITS"))),funs(log1p(.)))
hist(nyc_property$'COMMERCIAL UNITS')
```



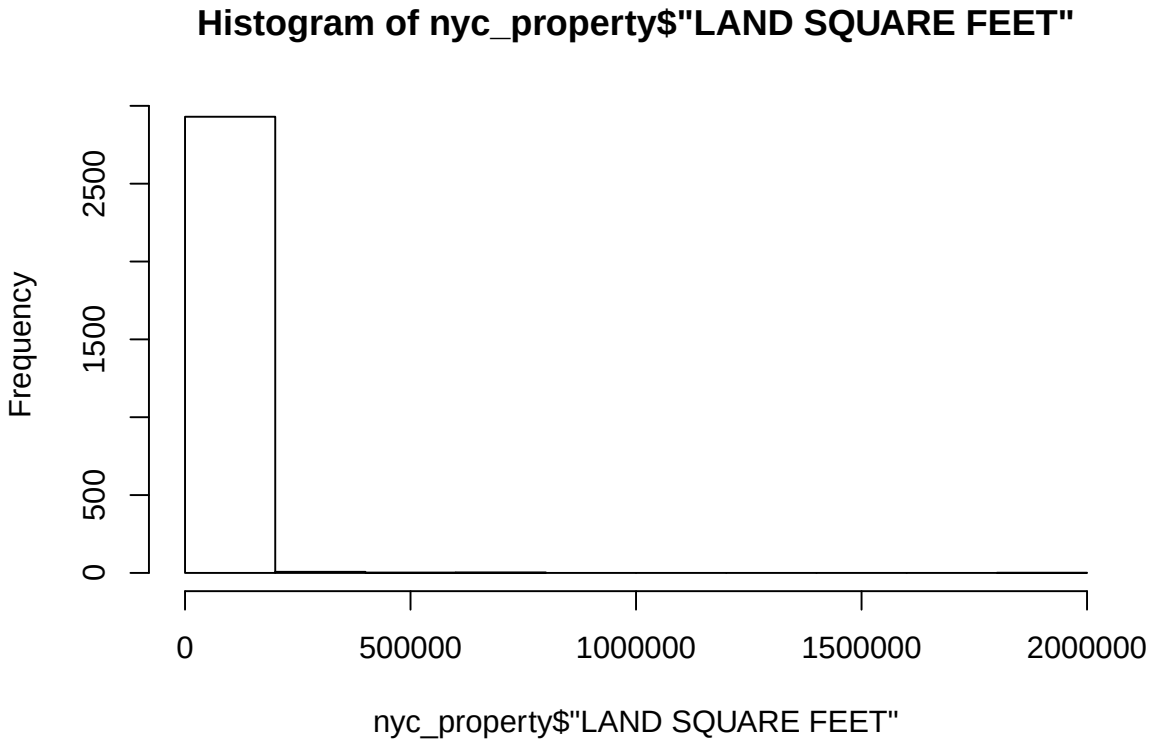

```
hist(nyc_property$'TOTAL UNITS')
```



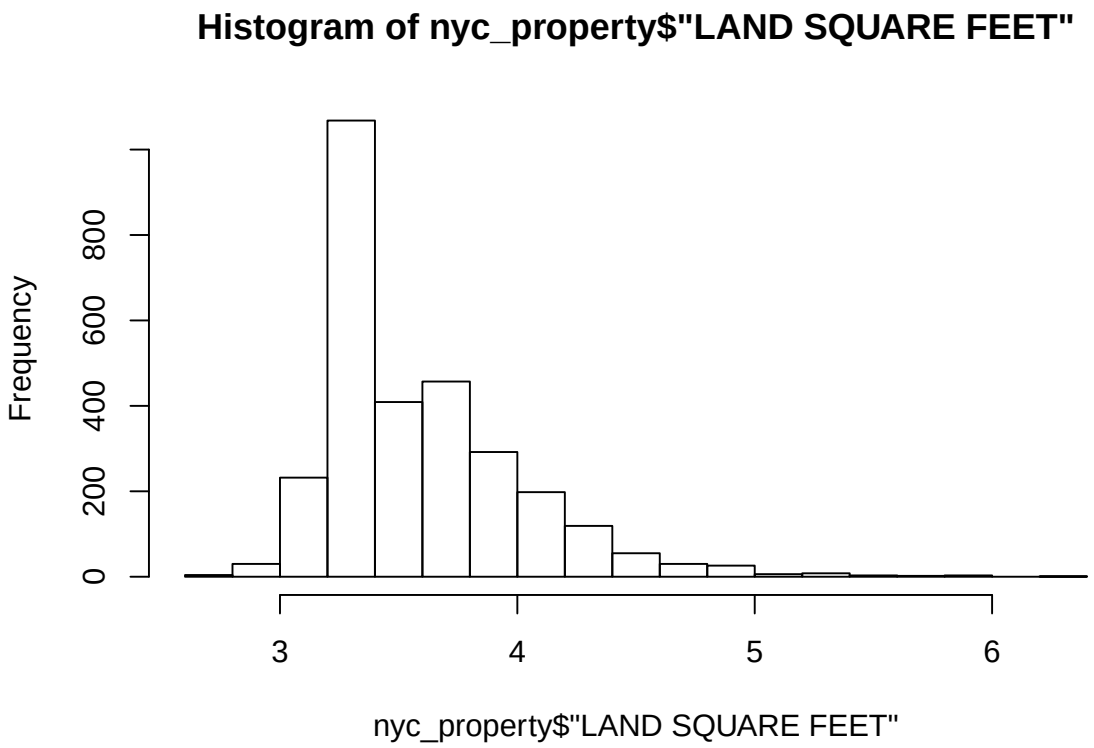

```
nyc_property <- nyc_property %>% mutate_at(c(which(names(nyc_property) %in%
                                                    c("TOTAL UNITS"))),funs(log10(.)))
hist(nyc_property$'TOTAL UNITS')
```



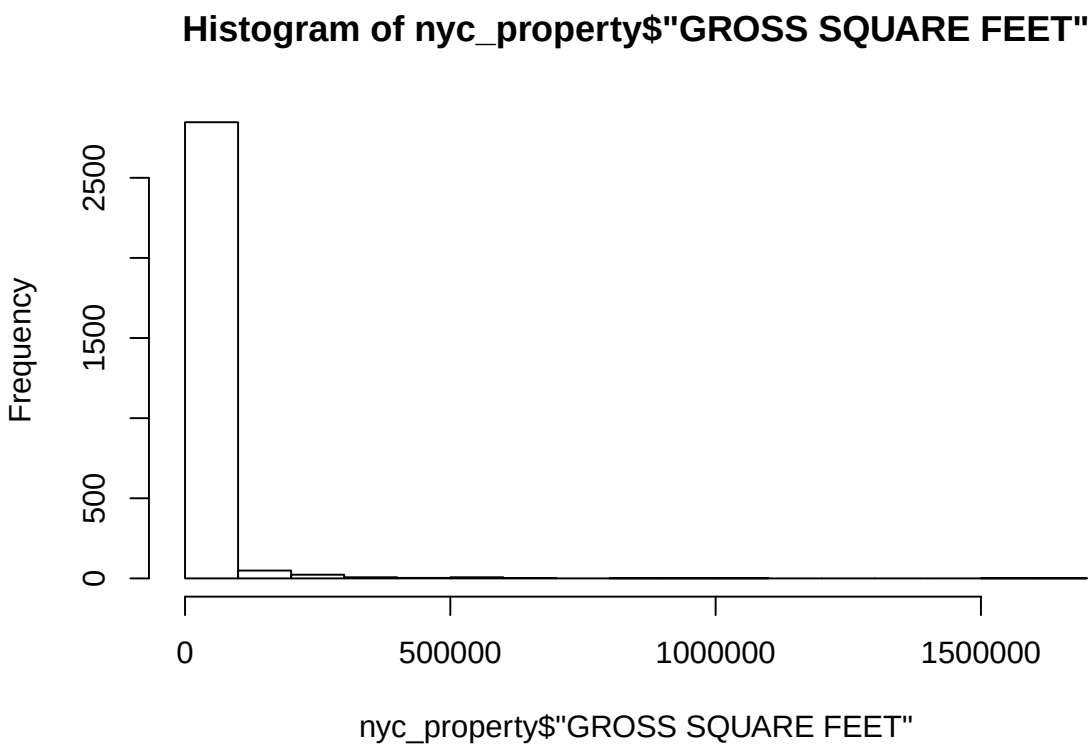

```
hist(nyc_property$'LAND SQUARE FEET')
```



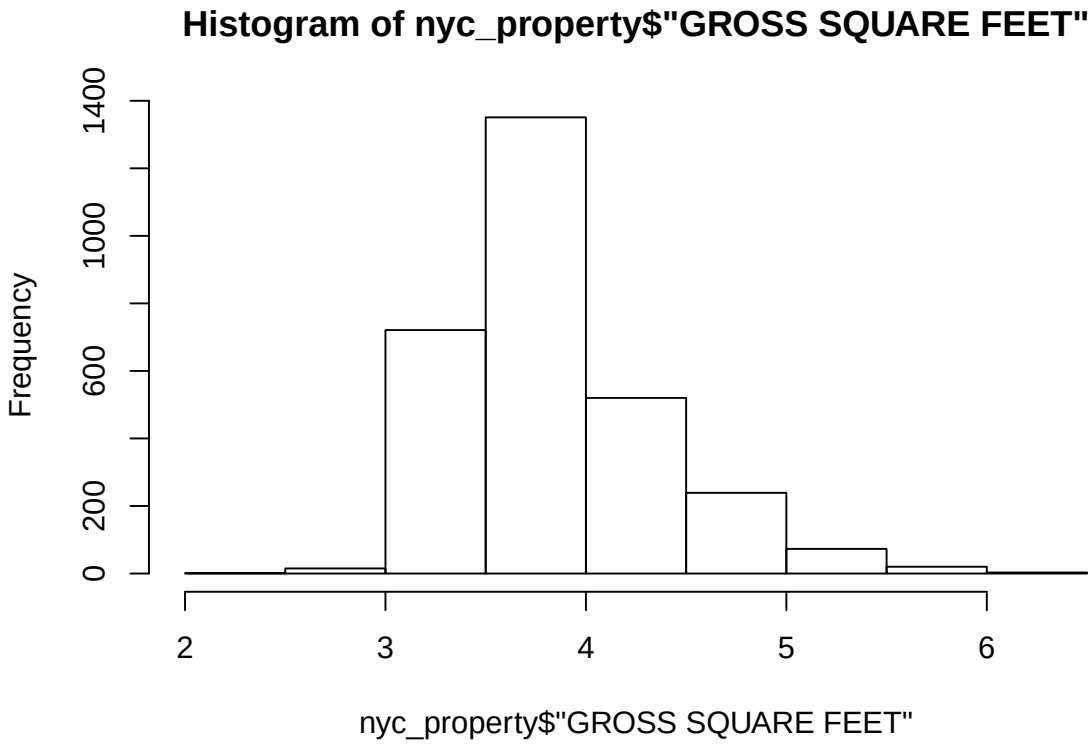
```
nyc_property <- nyc_property %>% mutate_at(c(which(names(nyc_property) %in%
                                                    c("LAND SQUARE FEET"))),funs(log10(.)))
hist(nyc_property$'LAND SQUARE FEET')
```




```
hist(nyc_property$'GROSS SQUARE FEET')
```




```
nyc_property <- nyc_property %>% mutate_at(c(which(names(nyc_property) %in%
                                                    c("GROSS SQUARE FEET"))),funs(log10(.)))
hist(nyc_property$'GROSS SQUARE FEET')
```



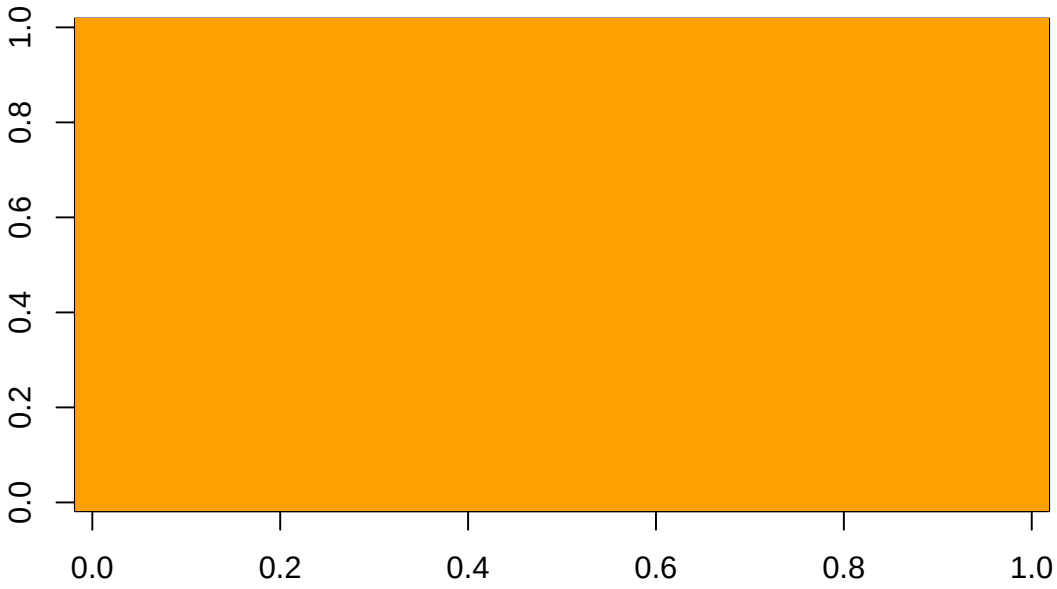
Analysis

We can start the analysis by splitting my data set into an 80-20% training and test set.

```
set.seed(23)
index <- sample(nrow(nyc_property),nrow(nyc_property)*0.80)
nyc_property.train <- nyc_property[index,]
nyc_property.test<- nyc_property[-index,]
```

Identify any near zero variance predictors

```
nzv <- nearZeroVar(nyc_property.train)
image(matrix(1:784 %in% nzv, 28, 28))
```

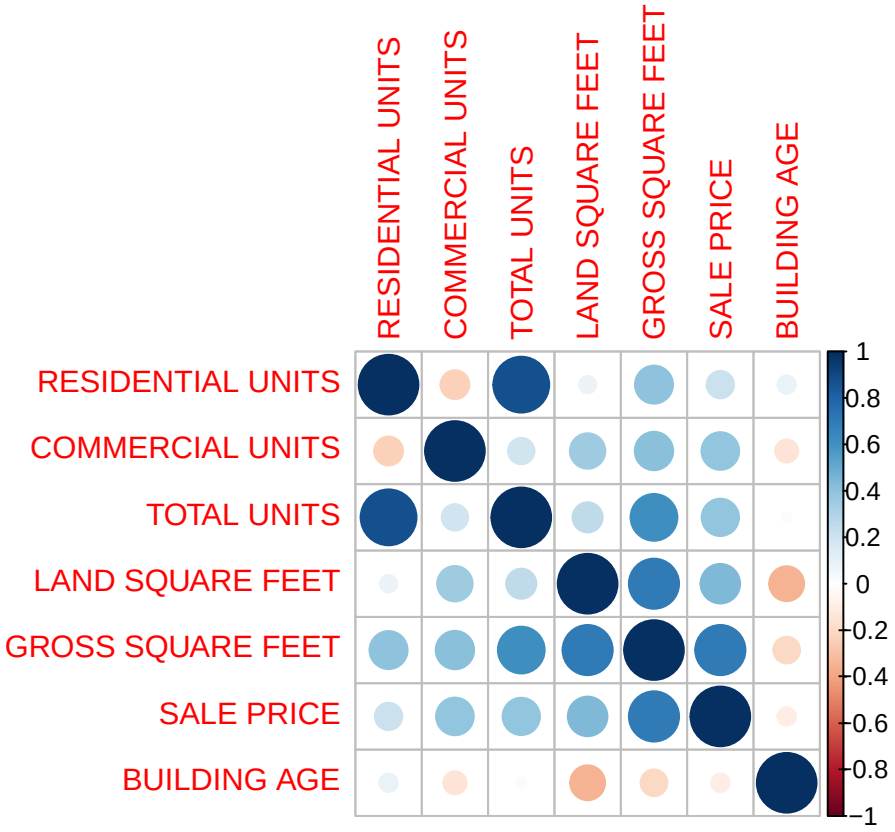



```
col_index <- setdiff(1:ncol(nyc_property.train), nzv)
length(col_index)
```

```
## [1] 20
```

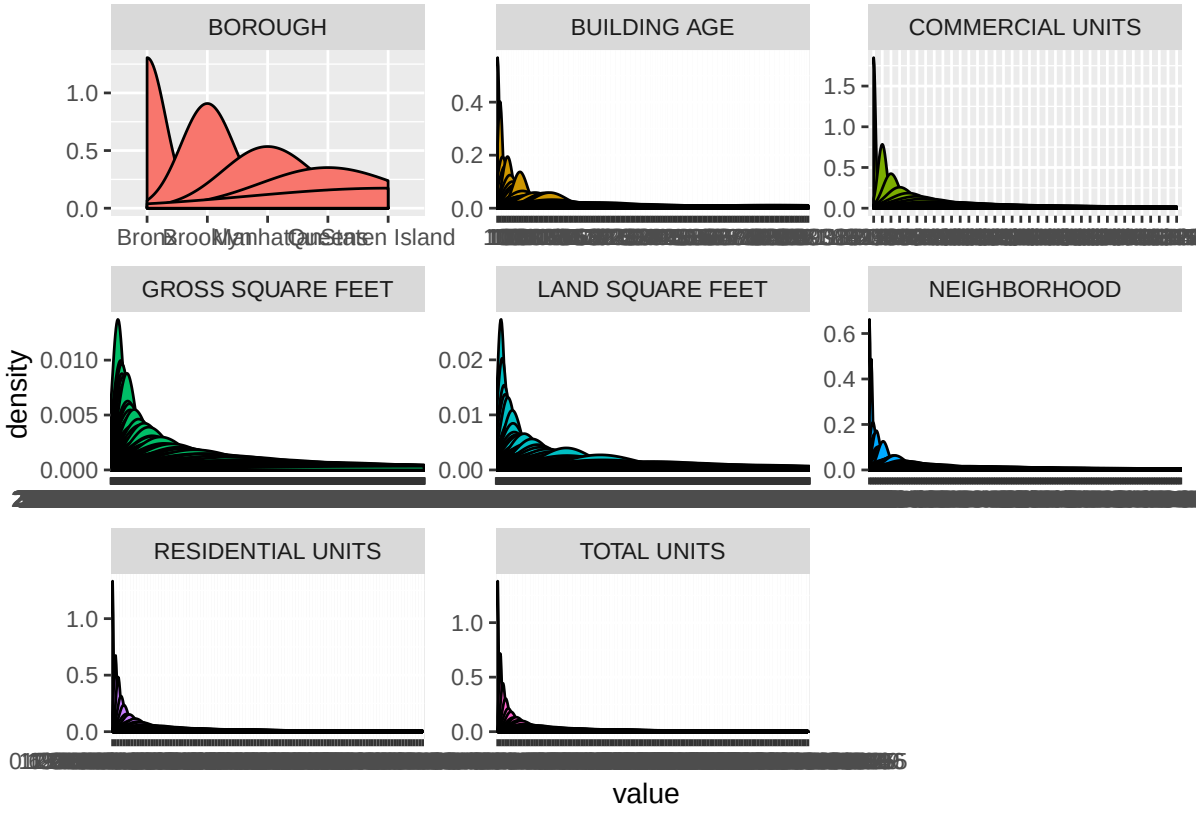
Let's plot the correlations among numerical fields.

```
nyc_property.train %>%
  select_if(is.numeric) %>%
  cor() %>%
  corrplot::corrplot()
```



Let's plot the metrics for these fields

```
nyc_property.train %>%
  dplyr::select(`BOROUGH`, `NEIGHBORHOOD`,`RESIDENTIAL UNITS`,`COMMERCIAL UNITS`,`TOTAL UNITS`,
         `LAND SQUARE FEET`,`GROSS SQUARE FEET`, `BUILDING AGE`) %>%
  gather(metric, value) %>%
  ggplot(aes(value, fill = metric)) +
  geom_density(show.legend = FALSE) +
  facet_wrap(~ metric, scales = "free")
```



There are neighborhood in test but not in train and will throw the following error later on. So let's add this to the train data.

> Error in model.frame.default(Terms, newdata, na.action = na.action, xlev = object$xlevels) : factor NEIGHBORHOOD has new levels ARROCHAR, BULLS HEAD, CONCORD, GRANT CITY, MARINERS HARBOR, MORRIS PARK/VAN NEST, SOUTH BEACH, SOUTH OZONE PARK, WOODLAWN

Let's verify if the above error is true and there are neighorhoods in test but not in train. We will put them in the training set.

```
nyc_property.train %>% filter(`NEIGHBORHOOD` == "ARROCHAR") #0 row
```

```
## # A tibble: 0 x 20
## # ... with 20 variables: BOROUGH <fct>, NEIGHBORHOOD <fct>, `BUILDING
## #   CLASS CATEGORY NUMBER` <fct>, `BUILDING CLASS CATEGORY` <fct>, `TAX
## #   CLASS AT PRESENT` <fct>, BLOCK <chr>, LOT <chr>, `BUILDING CLASS AT
## #   PRESENT` <fct>, `ZIP CODE` <fct>, `RESIDENTIAL UNITS` <dbl>,
## #   `COMMERCIAL UNITS` <dbl>, `TOTAL UNITS` <dbl>, `LAND SQUARE
## #   FEET` <dbl>, `GROSS SQUARE FEET` <dbl>, `TAX CLASS AT TIME OF
## #   SALE` <fct>, `BUILDING CLASS AT TIME OF SALE` <fct>, `SALE
## #   PRICE` <dbl>, `SALE YEAR` <fct>, `SALE MONTH` <fct>, `BUILDING
## #   AGE` <dbl>
```

```
nyc_property.test %>% filter(`NEIGHBORHOOD` == "ARROCHAR") #1 row
```

```
## # A tibble: 1 x 20
##   BOROUGH NEIGHBORHOOD `BUILDING CLASS~ `BUILDING CLASS~ `TAX CLASS AT P~
##   <fct>   <fct>        <fct>            <fct>            <fct>
## 1 Staten~ ARROCHAR     "14 "            RENTALS - 4-10 ~ 2A
## # ... with 15 more variables: BLOCK <chr>, LOT <chr>, `BUILDING CLASS AT
## #   PRESENT` <fct>, `ZIP CODE` <fct>, `RESIDENTIAL UNITS` <dbl>,
```

```
## #   `COMMERCIAL UNITS` <dbl>, `TOTAL UNITS` <dbl>, `LAND SQUARE
## #   FEET` <dbl>, `GROSS SQUARE FEET` <dbl>, `TAX CLASS AT TIME OF
## #   SALE` <fct>, `BUILDING CLASS AT TIME OF SALE` <fct>, `SALE
## #   PRICE` <dbl>, `SALE YEAR` <fct>, `SALE MONTH` <fct>, `BUILDING
## #   AGE` <dbl>
```

```
nyc_property.temp <- nyc_property.test

nyc_property.test <- nyc_property.temp %>%
  semi_join(nyc_property.train, by = "NEIGHBORHOOD") %>%
  semi_join(nyc_property.train, by = "BUILDING CLASS CATEGORY NUMBER") %>%
  semi_join(nyc_property.train, by = "BLOCK") %>%
  semi_join(nyc_property.train, by = "LOT") %>%
  semi_join(nyc_property.train, by = "BUILDING CLASS CATEGORY") %>%
  semi_join(nyc_property.train, by = "ZIP CODE") %>%
  semi_join(nyc_property.train, by = "BUILDING CLASS AT TIME OF SALE")

removed <- anti_join(nyc_property.temp, nyc_property.test)
```

```
## Joining, by = c("BOROUGH", "NEIGHBORHOOD", "BUILDING CLASS CATEGORY NUMBER", "BUILDING CLASS CATE
```

```
nyc_property.train <- rbind(nyc_property.train, removed)

rm(nyc_property.temp, removed)
```

Predictive Analysis

We will first create a function to calculate the RMSE.

```
RMSE <- function(true_sale_price, predicted_sale_price){
  sqrt(mean((true_sale_price - predicted_sale_price)^2))
}
```

First, we will begin with fitting Linear model using all fields. This gives a warning that the prediction from a rank-deficient fit may be misleading. This means that there are collinear covariates. Using the summary function, we see that there are NAs for Estimate, Std. Error and t value. From recommendations from mentors, I deduced that the warning maybe or may not be important but it is probably benign. But we will do another LM model without these fields and compare the results.

```
m <- lm(nyc_property.train$'SALE PRICE' ~ . - `SALE PRICE`, data = nyc_property.train)

predict_model_lm_m <- predict(m, nyc_property.test)
```

```
## Warning in predict.lm(m, nyc_property.test): prediction from a rank-
## deficient fit may be misleading
```

```
length(m$coefficients) > m$rank
```

```
## [1] TRUE
```

This means there are collinear covariates. Using summary, we see that there are NAs for Estimate, Std. Error and t value. (I will not show results here as this will create a large output.)

The warning may or may not be important but it's probably benign. But we can do another LM model without these fields and compare results.

```
summary(m)
```

```
model_1_rmse_m <- RMSE(predict_model_lm_m, nyc_property.test$'SALE PRICE')
model_1_rmse_m
```

```
## [1] 0.5844569
```

```
rmse_results <- data_frame(method = "LM", RMSE = model_1_rmse_m)
rmse_results %>% knitr::kable()
```

method	RMSE
LM	0.5844569

Now let's remove the collinear covariates and try the linear model again.

```
m <- lm(`SALE PRICE` ~ . - `NEIGHBORHOOD` - `BUILDING CLASS CATEGORY NUMBER` -
          `BUILDING CLASS CATEGORY` - `TAX CLASS AT PRESENT` - `BLOCK` -
          `BUILDING CLASS AT PRESENT` - `ZIP CODE` - `TAX CLASS AT TIME OF SALE` -
          `SALE PRICE` - `BUILDING CLASS AT TIME OF SALE` - `SALE MONTH` ,
        data = nyc_property.train)

predict_model_lm_m <- predict(m, nyc_property.test)


model_1_rmse_m <- RMSE(predict_model_lm_m, nyc_property.test$'SALE PRICE')
model_1_rmse_m
```

```
## [1] 0.2353085
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="LM with selected columns",
                                     RMSE = model_1_rmse_m ))
rmse_results %>% knitr::kable()
```

method	RMSE
LM	0.5844569
LM with selected columns	0.2353085

Next, we will try the Gradient Boosting Machine (GBM) model. This is quite memory intensive so we will remove any global data/variables before running this.

```
rm(m, model_1_rmse_m, predict_model_lm_m, outliers, nzv, fac, index, nyc_prop)
```

When running a full model, it gives an error that variables need to be of type numeric, ordered or factor (not characters).

```
gbm.fit <- gbm(
   formula = nyc_property.train$'SALE PRICE' ~ . - `SALE PRICE`,
   distribution = "gaussian",
   data = nyc_property.train,
   n.trees = 1000,
   interaction.depth = 1,
   shrinkage = 0.001,
   cv.folds = 5,
   n.cores = NULL, # will use all cores by default
   verbose = FALSE
 )
```

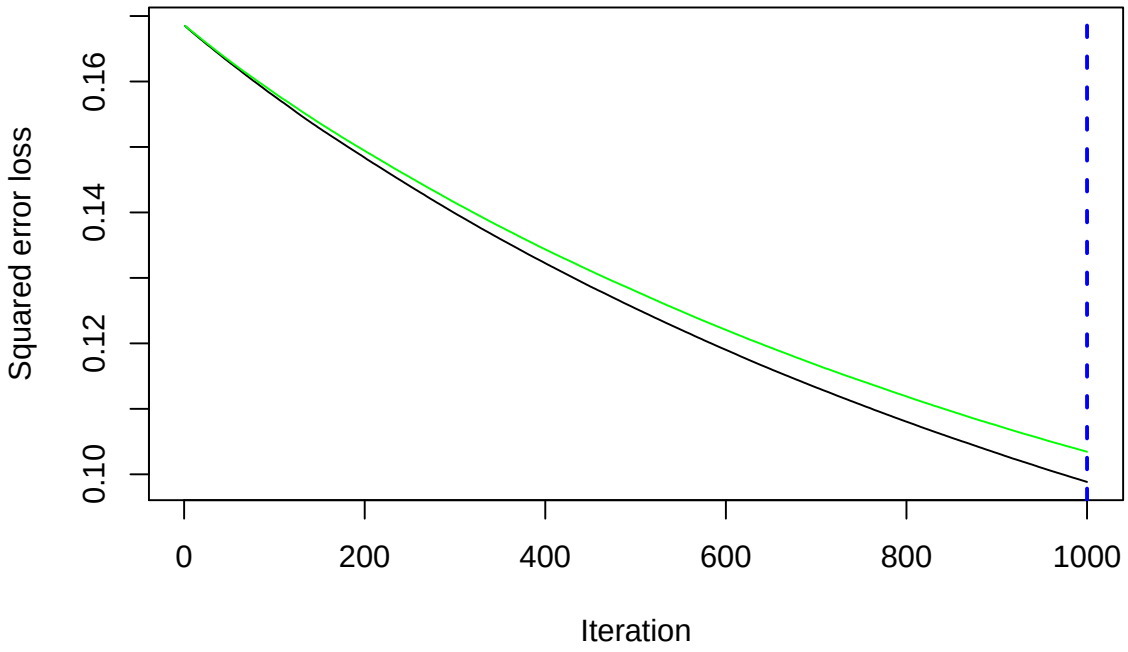
Error in checkForRemoteErrors(val) : 5 nodes produced errors; first error: variable 6: BLOCK is not of type numeric, ordered, or factor.

So we will remove these fields.

```
gbm.fit <- gbm(
  formula = `SALE PRICE` ~ . - `BLOCK` - `LOT` - `SALE PRICE`,
  distribution = "gaussian",
  data = nyc_property.train,
  n.trees = 1000,
  interaction.depth = 1,
  shrinkage = 0.001,
  cv.folds = 5,
  n.cores = NULL, # will use all cores by default
  verbose = FALSE
)

gbm.fit
```

```
## gbm(formula = `SALE PRICE` ~ . - BLOCK - LOT - `SALE PRICE`,
##     distribution = "gaussian", data = nyc_property.train, n.trees = 1000,
##     interaction.depth = 1, shrinkage = 0.001, cv.folds = 5, verbose = FALSE,
##     n.cores = NULL)
## A gradient boosted model with gaussian loss function.
## 1000 iterations were performed.
```




```
## The best cross-validation iteration was 1000.
## There were 17 predictors of which 5 had non-zero influence.
```

```
#The best cross-validation iteration was 7407

# get MSE and compute RMSE
```

```
gbm_model_rmse <- sqrt(min(gbm.fit$cv.error))
gbm_model_rmse
```

```
## [1] 0.3216269
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="GBM model",
                                     RMSE = gbm_model_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
LM	0.5844569
LM with selected columns	0.2353085
GBM model	0.3216269

```
rm(gbm.fit, gbm_model_rmse)
```

Since this is memory intensive, I have limited the n.trees to 1000 to make this reproducable. Otherwise, increasing this to 10000 will give this a better rmse.

Now we will try Random Forest:

Random forest have issues with the spaces in column names as they are illegal. So we will make illegal variable names to legal names first.

```
nyc_property.train_new <- nyc_property.train
names(nyc_property.train_new) <- make.names(names(nyc_property.train_new))

nyc_property.test_new <- nyc_property.test
names(nyc_property.test_new) <- make.names(names(nyc_property.test_new))
```

Random Forest also have issues with factors with more than 53 categories. Let's identify them and exclude them from the model.

Error: Can not handle categorical predictors with more than 53 categories.

```
names(Filter(is.factor, nyc_property.train_new))
```

```
##  [1] "BOROUGH"                         "NEIGHBORHOOD"
##  [3] "BUILDING.CLASS.CATEGORY.NUMBER" "BUILDING.CLASS.CATEGORY"
##  [5] "TAX.CLASS.AT.PRESENT"            "BUILDING.CLASS.AT.PRESENT"
##  [7] "ZIP.CODE"                        "TAX.CLASS.AT.TIME.OF.SALE"
##  [9] "BUILDING.CLASS.AT.TIME.OF.SALE" "SALE.YEAR"
## [11] "SALE.MONTH"
```

```
length(unique(nyc_property.train_new$BOROUGH))
```

```
## [1] 5
```

```
length(unique(nyc_property.train_new$NEIGHBORHOOD))
```

```
## [1] 193
```

```
length(unique(nyc_property.train_new$BUILDING.CLASS.CATEGORY.NUMBER))
```

```
## [1] 29
```

```
length(unique(nyc_property.train_new$BUILDING.CLASS.CATEGORY))
```

```
## [1] 29
```

```
length(unique(nyc_property.train_new$TAX.CLASS.AT.PRESENT))
```

```
## [1] 5
```

```
length(unique(nyc_property.train_new$BUILDING.CLASS.AT.PRESENT))
```

```
## [1] 115
```

```
length(unique(nyc_property.train_new$ZIP.CODE))
```

```
## [1] 164
```

```
length(unique(nyc_property.train_new$TAX.CLASS.AT.TIME.OF.SALE))
```

```
## [1] 3
```

```
length(unique(nyc_property.train_new$BUILDING.CLASS.AT.TIME.OF.SALE))
```

```
## [1] 118
```

```
length(unique(nyc_property.train_new$SALE.YEAR))
```

```
## [1] 2
```

```
length(unique(nyc_property.train_new$SALE.MONTH))
```

```
## [1] 12
```

```
rf_model <- randomForest(`SALE.PRICE` ~ . - `NEIGHBORHOOD` - `BUILDING.CLASS.AT.PRESENT` -
                           `ZIP.CODE` - `COMMERCIAL.UNITS` - `BUILDING.CLASS.AT.TIME.OF.SALE` -
                           `SALE.PRICE`,
                         data=nyc_property.train_new,
                         importance=TRUE, na.action=na.omit)

predictions_rf_model1 <- predict(rf_model, nyc_property.test_new)

rf_model_1_rmse <-RMSE(predictions_rf_model1, nyc_property.test_new$'SALE.PRICE')
rf_model_1_rmse
```

```
## [1] 0.1805946
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Random Forest model",
                                     RMSE = rf_model_1_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
LM	0.5844569
LM with selected columns	0.2353085
GBM model	0.3216269
Random Forest model	0.1805946

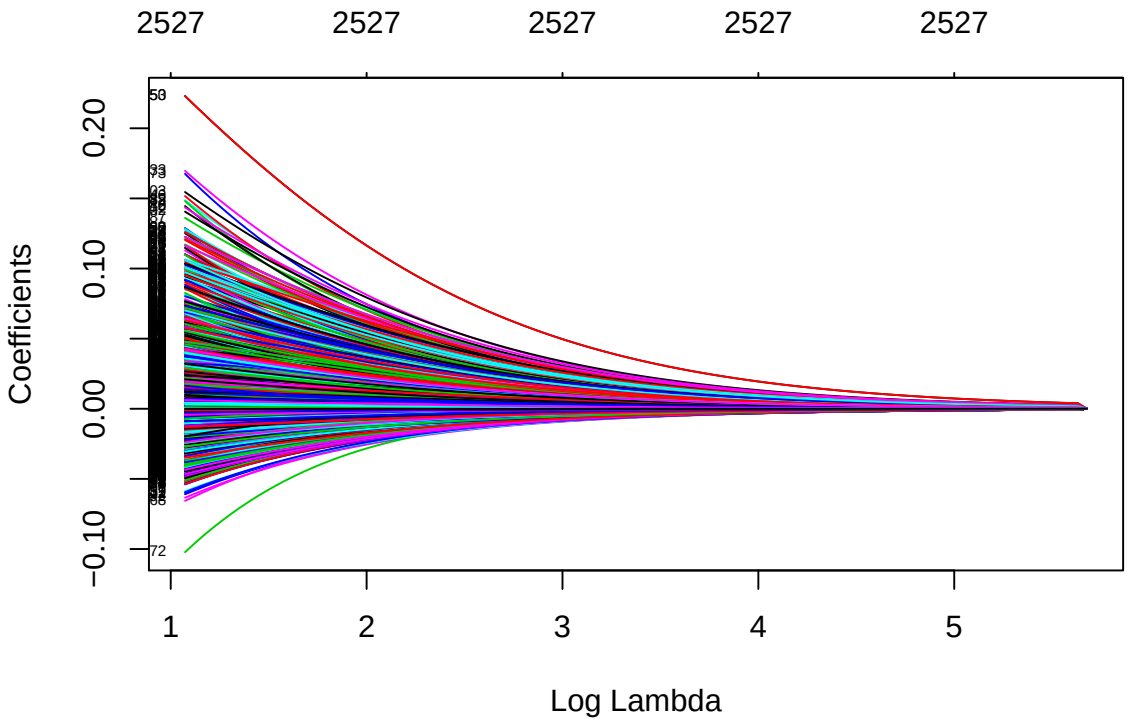
```
rm(rf_model, predictions_rf_model1, rf_model_1_rmse)
rm(nyc_property.train_new, nyc_property.test_new)
```

Here is GLM with Ridge Regression. Since Ridge Regression needs the data in a matrix instead of dataframe, we will transform the data to a matrix first and once again splitting the training and testing data.

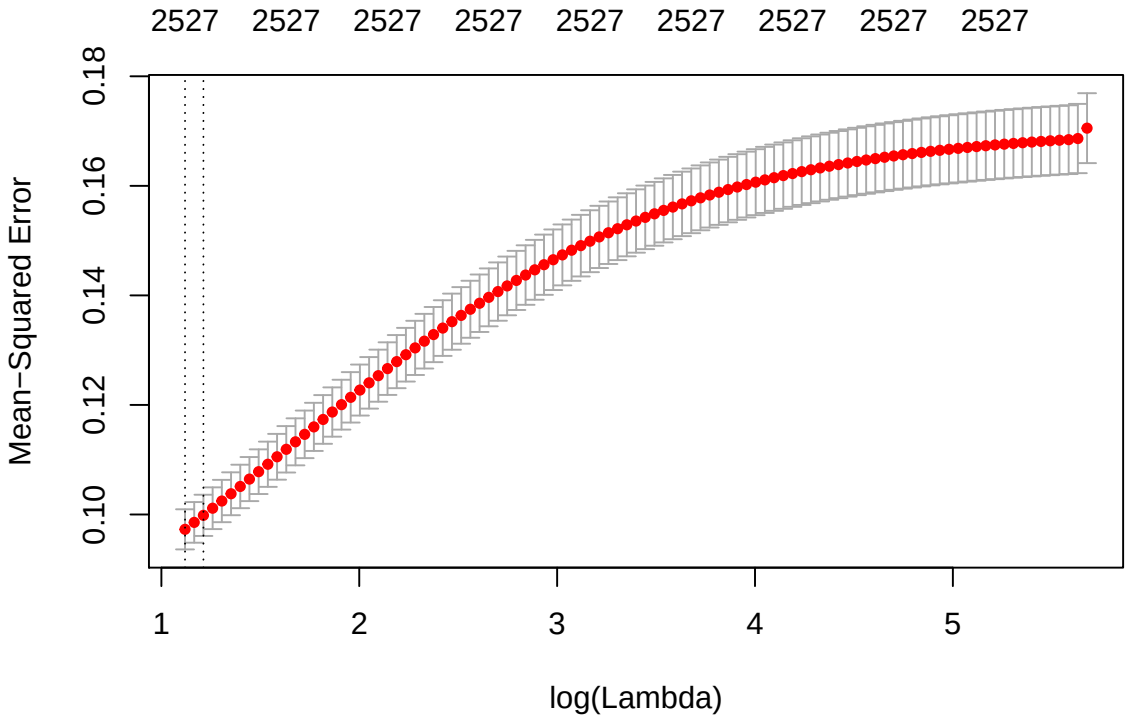
```
x=model.matrix(nyc_property$'SALE PRICE' ~ . - `SALE PRICE`,data=nyc_property)
y=nyc_property$'SALE PRICE'
set.seed(23)
```

```
index <- sample(nrow(x),nrow(x)*0.80)
mtx_train_x <- x[index,]
mtx_train_y <- y[index]
mtx_test_x<- x[-index,]
mtx_test_y <- y[-index]

ridge <- glmnet(mtx_train_x,mtx_train_y,alpha=0)
plot(ridge,xvar="lambda",label=TRUE)
```




```
cv.ridge=cv.glmnet(mtx_train_x,mtx_train_y,alpha=0)
plot(cv.ridge)
```




```
predictions_ridge_model <- predict(ridge, mtx_test_x)

ridge_model_1_rmse <-RMSE(predictions_ridge_model, mtx_test_y)
ridge_model_1_rmse
```

```
## [1] 0.3714356
```

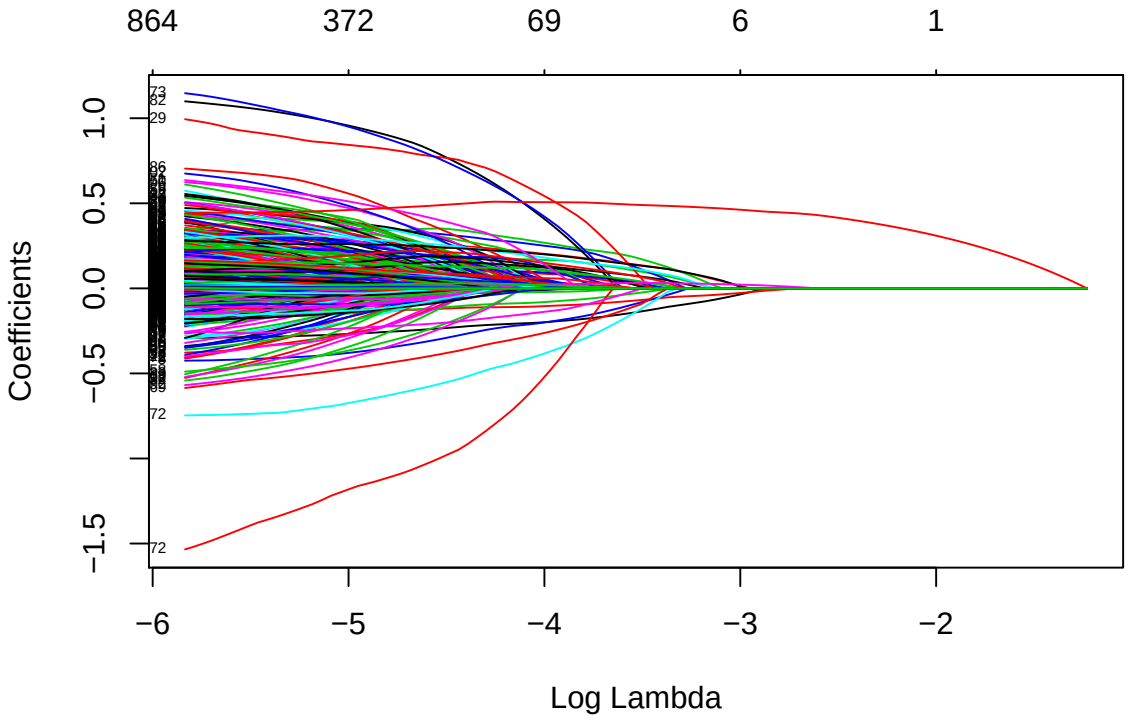
```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="GLM with Ridge Regression model",
                                     RMSE = ridge_model_1_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
LM	0.5844569
LM with selected columns	0.2353085
GBM model	0.3216269
Random Forest model	0.1805946
GLM with Ridge Regression model	0.3714356

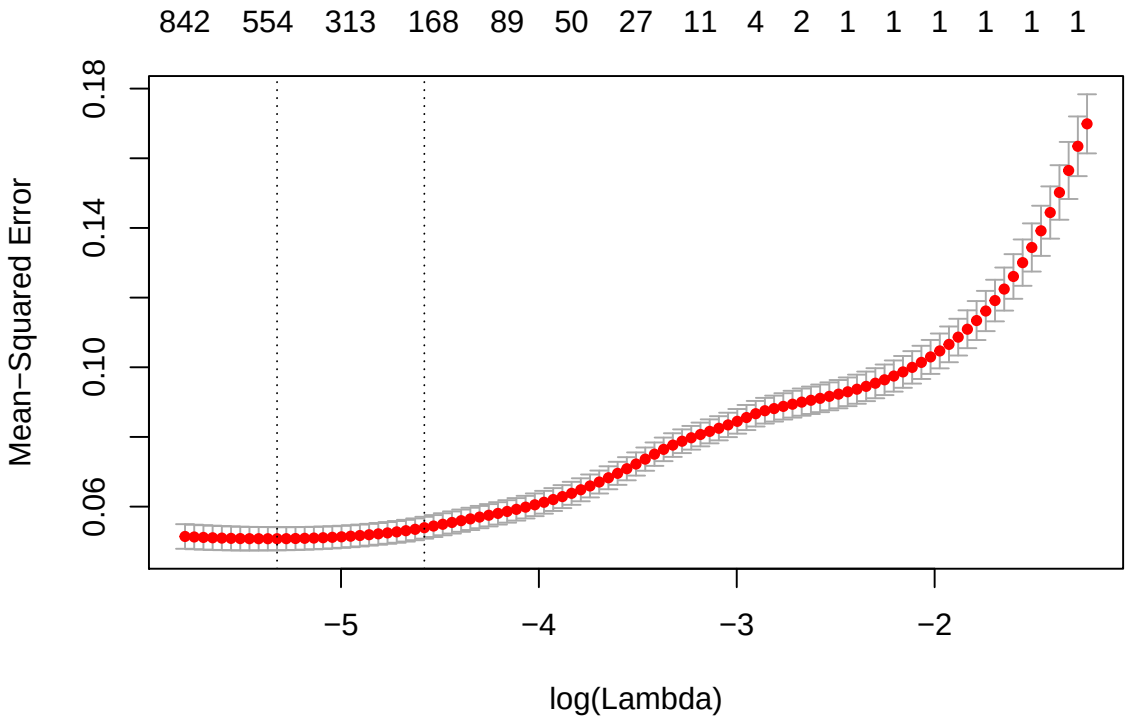
```
rm(ridge, predictions_ridge_model, ridge_model_1_rmse, cv.ridge)
```

GLM with Lasso Regression

```
#lasso
lasso <- glmnet(mtx_train_x,mtx_train_y,alpha=1)
plot(lasso,xvar="lambda",label=TRUE)
```




```
cv.lasso=cv.glmnet(mtx_train_x,mtx_train_y,alpha=1)
plot(cv.lasso)
```




```
predictions_lasso_model <- predict(lasso, mtx_test_x)
lasso_model_1_rmse <-RMSE(predictions_lasso_model, mtx_test_y)
```

```
lasso_model_1_rmse
```

```
## [1] 0.2651776
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="GLM with Lasso Regression model",
                                     RMSE = lasso_model_1_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
LM	0.5844569
LM with selected columns	0.2353085
GBM model	0.3216269
Random Forest model	0.1805946
GLM with Ridge Regression model	0.3714356
GLM with Lasso Regression model	0.2651776

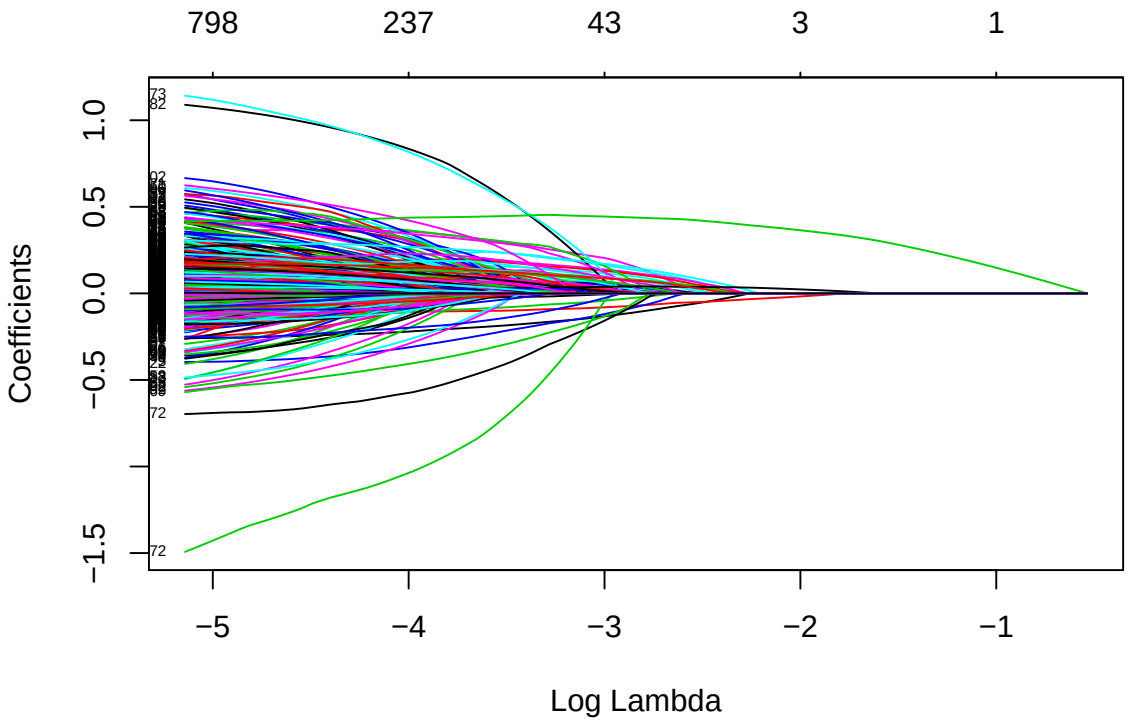
```
lam.best=lasso$lambda[order(lasso_model_1_rmse)[1]]
lam.best
```

```
## [1] 0.2923926
```

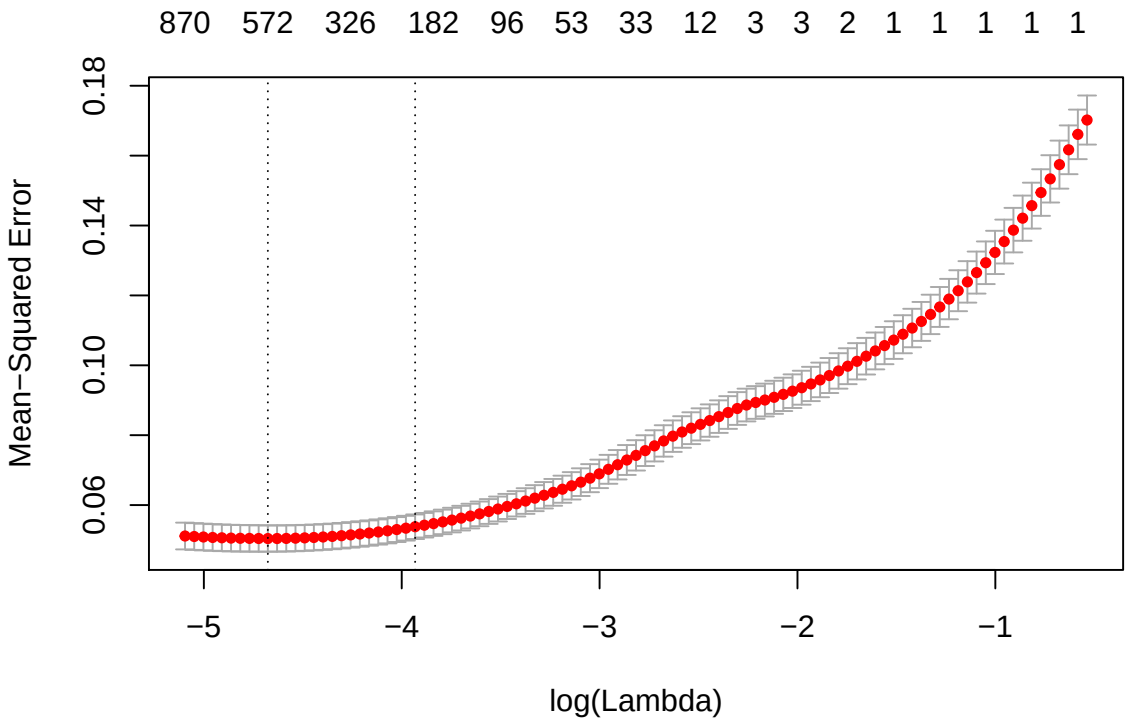
```
rm(lasso, predictions_lasso_model, lasso_model_1_rmse, lam.best, cv.lasso)
```

GLM with Elastic Net Regression

```
#elastic net
elnet <- glmnet(mtx_train_x,mtx_train_y,alpha=0.5)
plot(elnet,xvar="lambda",label=TRUE)
```



```
cv.elnet=cv.glmnet(mtx_train_x,mtx_train_y,alpha=0.5)
plot(cv.elnet)
```




```
predictions_elnet_model <- predict(elnet, mtx_test_x)
elnet_model_1_rmse <-RMSE(predictions_elnet_model, mtx_test_y)
elnet_model_1_rmse
```

```
## [1] 0.2727173
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="GLM with Elastic Net Regression model",
                                     RMSE = elnet_model_1_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
LM	0.5844569
LM with selected columns	0.2353085
GBM model	0.3216269
Random Forest model	0.1805946
GLM with Ridge Regression model	0.3714356
GLM with Lasso Regression model	0.2651776
GLM with Elastic Net Regression model	0.2727173

```
lam.best=elnet$lambda[order(elnet_model_1_rmse)[1]]
lam.best
```

```
## [1] 0.5847851
```

```
rm(elnet, predictions_elnet_model, elnet_model_1_rmse, lam.best, cv.elnet)
rm(mtx_train_x, mtx_train_y, mtx_test_x, mtx_test_y, x, y)
```

Results

From running Linear Model, Random Forest, Ridge Regression, Lasso Regression, Elastic Net, GBM, the best RMSE results was Random Forest with 0.1805946. Second best is Lasso Regression.

Conclusion

There was alot of data cleaning for this data set and trying to get the fields in the right data type and shape was a hurdle. There is also the limitation on computer memory to improve on the model like using GBM. But having a 0.18 RMSE for Random Forest is pretty good and other models giving similar results reenforce this prediction. In the future, it is possible to improve on this by looking for outliers on other fields and using models such as KNN.