NYC Property Sale Price Predictor

Donghan Lee

10/27/2021

Introduction

The prediction of price on an asset is of keen interest. Particularly, real estate price is related more-or-less to people's life. Who don't want to live in a cozy house? Thus, I have searched data set for machine learning to build a price predictor. I have found *"NYC Property Sales"* data in Kaggle webpage, which is *"A year's worth of properties sold on the NYC real estate market"* according to the webpage. The source of data is City of New York and the last update is 4 years ago. According to the webpage, the data contains all the building unit sold in NYC property market over a 12 month period.

Variables

The contents of the data include 22 variables such as sale price, address, unit type, borough, square feet, and so on.

Goal

Using the data, I will make a price predictor.

Flow

1. data acquisition.
2. data analysis and build a clean data.
3. training and test sets creation
4. simulation of three different methods.

- Naive approach
    - the expected price is the average
- regression tree model (rpart)
- k-nearest neighbors (kNN) algorithm

##Methos/analysis

Due to the authorizing issue (needs login), a zip file was downloaded from "https://www.kaggle.com/new-york-city/nyc-property-sales" and unziped to "nyc-rolling-sales.csv"

libraries loading

```
library(tidyverse)
library(lubridate)
library(dplyr)
library(knitr)
```

Benchmark RMSE function Root Mean Square Error is defined as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(y_{\hat{u},i} - y_{u,i})^2}$$

where $y_{\hat{u},i}$ and $y_{u,i}$ are predicted and actul values. This RMSE is used as the Benchmark.

```
RMSE <- function(predictions, true_ratings){
  sqrt(mean((true_ratings - predictions)^2))
}
```

After inspecting the file, I found that the delimiter is ",". Thus, tidyverse::read_csv() is used and assigned to sale.

```
sale <- read_csv("nyc-rolling-sales.csv")
```

Data consists of row: 84548 and column: 22.

Inspection of data

```
head(sale)
```

```
## # A tibble: 6 x 22
##    ...1 BOROUGH NEIGHBORHOOD  `BUILDING CLASS CAT~ `TAX CLASS AT PR~ BLOCK   LOT
##   <dbl>   <dbl> <chr>         <chr>                <chr>             <dbl> <dbl>
## 1     4       1 ALPHABET CITY 07 RENTALS - WALKUP~ 2A                  392     6
## 2     5       1 ALPHABET CITY 07 RENTALS - WALKUP~ 2                   399    26
## 3     6       1 ALPHABET CITY 07 RENTALS - WALKUP~ 2                   399    39
## 4     7       1 ALPHABET CITY 07 RENTALS - WALKUP~ 2B                  402    21
## 5     8       1 ALPHABET CITY 07 RENTALS - WALKUP~ 2A                  404    55
## 6     9       1 ALPHABET CITY 07 RENTALS - WALKUP~ 2                   405    16
## # ... with 15 more variables: EASE-MENT <lgl>, BUILDING CLASS AT PRESENT <chr>,
## #   ADDRESS <chr>, APARTMENT NUMBER <chr>, ZIP CODE <dbl>,
## #   RESIDENTIAL UNITS <dbl>, COMMERCIAL UNITS <dbl>, TOTAL UNITS <dbl>,
## #   LAND SQUARE FEET <chr>, GROSS SQUARE FEET <chr>, YEAR BUILT <dbl>,
## #   TAX CLASS AT TIME OF SALE <dbl>, BUILDING CLASS AT TIME OF SALE <chr>,
## #   SALE PRICE <chr>, SALE DATE <dttm>
```

```
tail(sale)
```

```
## # A tibble: 6 x 22
##    ...1 BOROUGH NEIGHBORHOOD `BUILDING CLASS CATE~ `TAX CLASS AT PR~ BLOCK   LOT
##   <dbl>   <dbl> <chr>        <chr>                 <chr>             <dbl> <dbl>
## 1  8408       5 WOODROW      02 TWO FAMILY DWELLI~ 1                  7339    41
## 2  8409       5 WOODROW      02 TWO FAMILY DWELLI~ 1                  7349    34
## 3  8410       5 WOODROW      02 TWO FAMILY DWELLI~ 1                  7349    78
## 4  8411       5 WOODROW      02 TWO FAMILY DWELLI~ 1                  7351    60
## 5  8412       5 WOODROW      22 STORE BUILDINGS    4                  7100    28
## 6  8413       5 WOODROW      35 INDOOR PUBLIC AND~ 4                  7105   679
## # ... with 15 more variables: EASE-MENT <lgl>, BUILDING CLASS AT PRESENT <chr>,
## #   ADDRESS <chr>, APARTMENT NUMBER <chr>, ZIP CODE <dbl>,
## #   RESIDENTIAL UNITS <dbl>, COMMERCIAL UNITS <dbl>, TOTAL UNITS <dbl>,
```

```
## #   LAND SQUARE FEET <chr>, GROSS SQUARE FEET <chr>, YEAR BUILT <dbl>,
## #   TAX CLASS AT TIME OF SALE <dbl>, BUILDING CLASS AT TIME OF SALE <chr>,
## #   SALE PRICE <chr>, SALE DATE <dttm>
```

```
str(sale)
```

```
## spec_tbl_df [84,548 x 22] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ ...1                          : num [1:84548] 4 5 6 7 8 9 10 11 12 13 ...
##  $ BOROUGH                       : num [1:84548] 1 1 1 1 1 1 1 1 1 1 ...
##  $ NEIGHBORHOOD                  : chr [1:84548] "ALPHABET CITY" "ALPHABET CITY" "ALPHABET CITY" "AL
##  $ BUILDING CLASS CATEGORY       : chr [1:84548] "07 RENTALS - WALKUP APARTMENTS" "07 RENTALS - WALKU
##  $ TAX CLASS AT PRESENT          : chr [1:84548] "2A" "2" "2" "2B" ...
##  $ BLOCK                         : num [1:84548] 392 399 399 402 404 405 406 407 379 387 ...
##  $ LOT                           : num [1:84548] 6 26 39 21 55 16 32 18 34 153 ...
##  $ EASE-MENT                     : logi [1:84548] NA NA NA NA NA NA ...
##  $ BUILDING CLASS AT PRESENT     : chr [1:84548] "C2" "C7" "C7" "C4" ...
##  $ ADDRESS                       : chr [1:84548] "153 AVENUE B" "234 EAST 4TH   STREET" "197 EAST 3R
##  $ APARTMENT NUMBER              : chr [1:84548] NA NA NA NA ...
##  $ ZIP CODE                      : num [1:84548] 10009 10009 10009 10009 10009 ...
##  $ RESIDENTIAL UNITS             : num [1:84548] 5 28 16 10 6 20 8 44 15 24 ...
##  $ COMMERCIAL UNITS              : num [1:84548] 0 3 1 0 0 0 0 2 0 0 ...
##  $ TOTAL UNITS                   : num [1:84548] 5 31 17 10 6 20 8 46 15 24 ...
##  $ LAND SQUARE FEET              : chr [1:84548] "1633" "4616" "2212" "2272" ...
##  $ GROSS SQUARE FEET             : chr [1:84548] "6440" "18690" "7803" "6794" ...
##  $ YEAR BUILT                    : num [1:84548] 1900 1900 1900 1913 1900 ...
##  $ TAX CLASS AT TIME OF SALE     : num [1:84548] 2 2 2 2 2 2 2 2 2 2 ...
##  $ BUILDING CLASS AT TIME OF SALE: chr [1:84548] "C2" "C7" "C7" "C4" ...
##  $ SALE PRICE                    : chr [1:84548] "6625000" "-" "-" "3936272" ...
##  $ SALE DATE                     : POSIXct[1:84548], format: "2017-07-19" "2016-12-14" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   ...1 = col_double(),
##   ..   BOROUGH = col_double(),
##   ..   NEIGHBORHOOD = col_character(),
##   ..   `BUILDING CLASS CATEGORY` = col_character(),
##   ..   `TAX CLASS AT PRESENT` = col_character(),
##   ..   BLOCK = col_double(),
##   ..   LOT = col_double(),
##   ..   `EASE-MENT` = col_logical(),
##   ..   `BUILDING CLASS AT PRESENT` = col_character(),
##   ..   ADDRESS = col_character(),
##   ..   `APARTMENT NUMBER` = col_character(),
##   ..   `ZIP CODE` = col_double(),
##   ..   `RESIDENTIAL UNITS` = col_double(),
##   ..   `COMMERCIAL UNITS` = col_double(),
##   ..   `TOTAL UNITS` = col_double(),
##   ..   `LAND SQUARE FEET` = col_character(),
##   ..   `GROSS SQUARE FEET` = col_character(),
##   ..   `YEAR BUILT` = col_double(),
##   ..   `TAX CLASS AT TIME OF SALE` = col_double(),
##   ..   `BUILDING CLASS AT TIME OF SALE` = col_character(),
##   ..   `SALE PRICE` = col_character(),
##   ..   `SALE DATE` = col_datetime(format = "")
##   .. )
```

```
##  - attr(*, "problems")=<externalptr>
```

```
names(sale)
```

```
##  [1] "...1"                           "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"
```

Data contains 22 columns. * "…1" : property index, double * "BOROUGH" : administration district, double * "NEIGHBORHOOD" : neighborhodd name, chr * "BUILDING CLASS CATEGORY" : what kind of property, chr * "TAX CLASS AT PRESENT" : tax category, chr * "BLOCK" : double
* "LOT" : double * "EASE-MENT" : logical
* "BUILDING CLASS AT PRESENT" : chr * "ADDRESS" : chr
* "APARTMENT NUMBER" : chr * "ZIP CODE" : double
* "RESIDENTIAL UNITS" : double * "COMMERCIAL UNITS" : double
* "TOTAL UNITS" : double * "LAND SQUARE FEET" : chr * "GROSS SQUARE FEET" : chr * "YEAR BUILT" : double
* "TAX CLASS AT TIME OF SALE" : double * "BUILDING CLASS AT TIME OF SALE" : chr * "SALE PRICE" : chr * "SALE DATE" : POSIXct

Data cleaning

SALE PRICE is what I want to predict and name is changed to price.

```
price <- sale["SALE PRICE"]
n_distinct(price)
```

```
## [1] 10008
```

Price contains unique 10008 values.

contents inspection

```
head(price)
```

```
## # A tibble: 6 x 1
##   `SALE PRICE`
##   <chr>
## 1 6625000
## 2 -
## 3 -
## 4 3936272
## 5 8000000
## 6 -
```

Price contians "-". "-" is replaced with the average value of prices. price variables are changed to numeric. Both can be achieved with as.numberic().

```
price <- as.numeric(t(price))

# replace NA with the average.

mprice <- round(mean(price, na.rm = T), digits = 0)

price <- sapply(price, function(l) {
  ifelse(!is.na(l), l, mprice)
})
```

price is rounded to 10^4

```
price <- round(price, digits = -4)
```

price is collected to dat.

```
dat <- data.frame(price = price)
```

"...1" is the index for property, propId.

```
propId <- sale["...1"]

n_distinct(propId)
```

```
## [1] 26736
```

propId has unique 26736 values. It is over our price number. Thus, we don't need this.

"BOROUGH" administration district, double –> borough, chr according to "A Property's Borough, Block and Lot Number". NYC.gov. City of New York. 1. Manhattan (New York County) 2. Bronx (Bronx County) 3. Brooklyn (Kings County) 4. Queens (Queens County) 5. Staten Island (Richmond County)

```
boro <- sale["BOROUGH"]
n_distinct(boro)
```

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

borough number is changed to a name accordingly.

```
boro <- sapply(boro, function(l) {
  ifelse(l == 1, "Manhattan",
         ifelse(l == 2, "Bronx",
                ifelse(l == 3, "Brooklyn",
                       ifelse(l == 4, "Queens", "Staten Island")
                )
         )
  )
})

boro <- factor(boro)
```

collecting borough information

```
dat <- dat %>% mutate(boro = boro)
```

table for the distribution

```
tab <- dat %>% count(boro)
tab %>% kable()
```

boro	n
Bronx	7049
Brooklyn	24047
Manhattan	18306
Queens	26736
Staten Island	8410

"NEIGHBORHOOD" : neighborhodd name, chr –> neigh

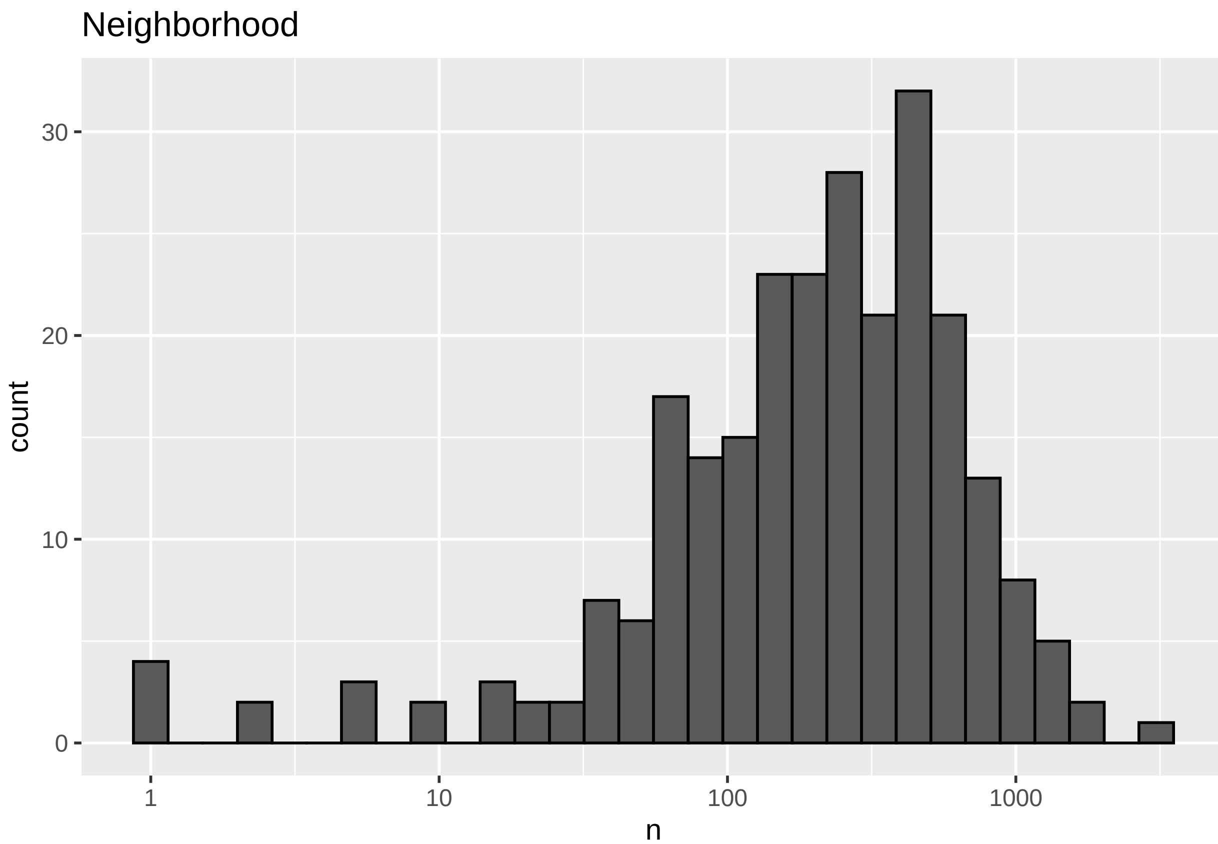
```
neigh <- as.matrix(sale["NEIGHBORHOOD"])
n_distinct(neigh)
```

```
## [1] 254
```

```
neigh <- factor(neigh)
```

In neighorhood, 254 categories exist. Thus, a histogram may be easier than a table.

```
dat <- dat %>% mutate(neigh = neigh)

dat %>%
  count(neigh) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Neighborhood")
```



"BUILDING CLASS CATEGORY" : what kind of property, chr -> b_cat

```
b_cat <- as.matrix(sale["BUILDING CLASS CATEGORY"])
n_distinct(b_cat)
```

```
## [1] 47
```

```
b_cat <- factor(b_cat)
```

collecting b_cat & distribution table

```
dat <- dat %>% mutate(b_cat = b_cat)
tab <- dat %>% count(b_cat)
tab %>% kable()
```

b_cat	n
01 ONE FAMILY DWELLINGS	18235
02 TWO FAMILY DWELLINGS	15828
03 THREE FAMILY DWELLINGS	4384
04 TAX CLASS 1 CONDOS	1656
05 TAX CLASS 1 VACANT LAND	1248
06 TAX CLASS 1 - OTHER	182
07 RENTALS - WALKUP APARTMENTS	3466
08 RENTALS - ELEVATOR APARTMENTS	382

b_cat	n
09 COOPS - WALKUP APARTMENTS	2767
10 COOPS - ELEVATOR APARTMENTS	12902
11 SPECIAL CONDO BILLING LOTS	2
11A CONDO-RENTALS	43
12 CONDOS - WALKUP APARTMENTS	926
13 CONDOS - ELEVATOR APARTMENTS	12989
14 RENTALS - 4-10 UNIT	671
15 CONDOS - 2-10 UNIT RESIDENTIAL	1281
16 CONDOS - 2-10 UNIT WITH COMMERCIAL UNIT	96
17 CONDO COOPS	1201
18 TAX CLASS 3 - UNTILITY PROPERTIES	4
21 OFFICE BUILDINGS	350
22 STORE BUILDINGS	935
23 LOFT BUILDINGS	46
25 LUXURY HOTELS	12
26 OTHER HOTELS	114
27 FACTORIES	201
28 COMMERCIAL CONDOS	30
29 COMMERCIAL GARAGES	587
30 WAREHOUSES	326
31 COMMERCIAL VACANT LAND	463
32 HOSPITAL AND HEALTH FACILITIES	59
33 EDUCATIONAL FACILITIES	69
34 THEATRES	12
35 INDOOR PUBLIC AND CULTURAL FACILITIES	32
36 OUTDOOR RECREATIONAL FACILITIES	14
37 RELIGIOUS FACILITIES	100
38 ASYLUMS AND HOMES	25
39 TRANSPORTATION FACILITIES	2
40 SELECTED GOVERNMENTAL FACILITIES	2
41 TAX CLASS 4 - OTHER	158
42 CONDO CULTURAL/MEDICAL/EDUCATIONAL/ETC	13
43 CONDO OFFICE BUILDINGS	475
44 CONDO PARKING	1441
45 CONDO HOTELS	211
46 CONDO STORE BUILDINGS	154
47 CONDO NON-BUSINESS STORAGE	377
48 CONDO TERRACES/GARDENS/CABANAS	47
49 CONDO WAREHOUSES/FACTORY/INDUS	30

"TAX CLASS AT PRESENT" : tax category, chr -> tax_c_a_p

```
tax_c_a_p <- as.matrix(sale["TAX CLASS AT PRESENT"])
n_distinct(tax_c_a_p)
```

```
## [1] 11
```

```
tax_c_a_p <- tax_c_a_p
```

collecting tax_c_a_p & distribution table

```
dat <- dat %>% mutate(tax_c_a_p = tax_c_a_p)
tab <- dat %>% count(tax_c_a_p)
tabtax <- tab
tab %>% kable()
```

tax_c_a_p	n
1	38633
1A	1444
1B	1234
1C	186
2	30919
2A	2521
2B	814
2C	1915
3	4
4	6140
NA	738

Tax class has four categories. But there are NA and sub-categories in tax_c_a_p. sub categories will be merged and NA should be removed. this is done after collecting all the variables.

"BLOCK" : double -> block

```
block <- as.matrix(sale["BLOCK"])

n_distinct(block)
```

```
## [1] 11566
```

block has 11566, which is over the number of price unique values. Thus, block won't be used in predicton.

"LOT" : double -> lot

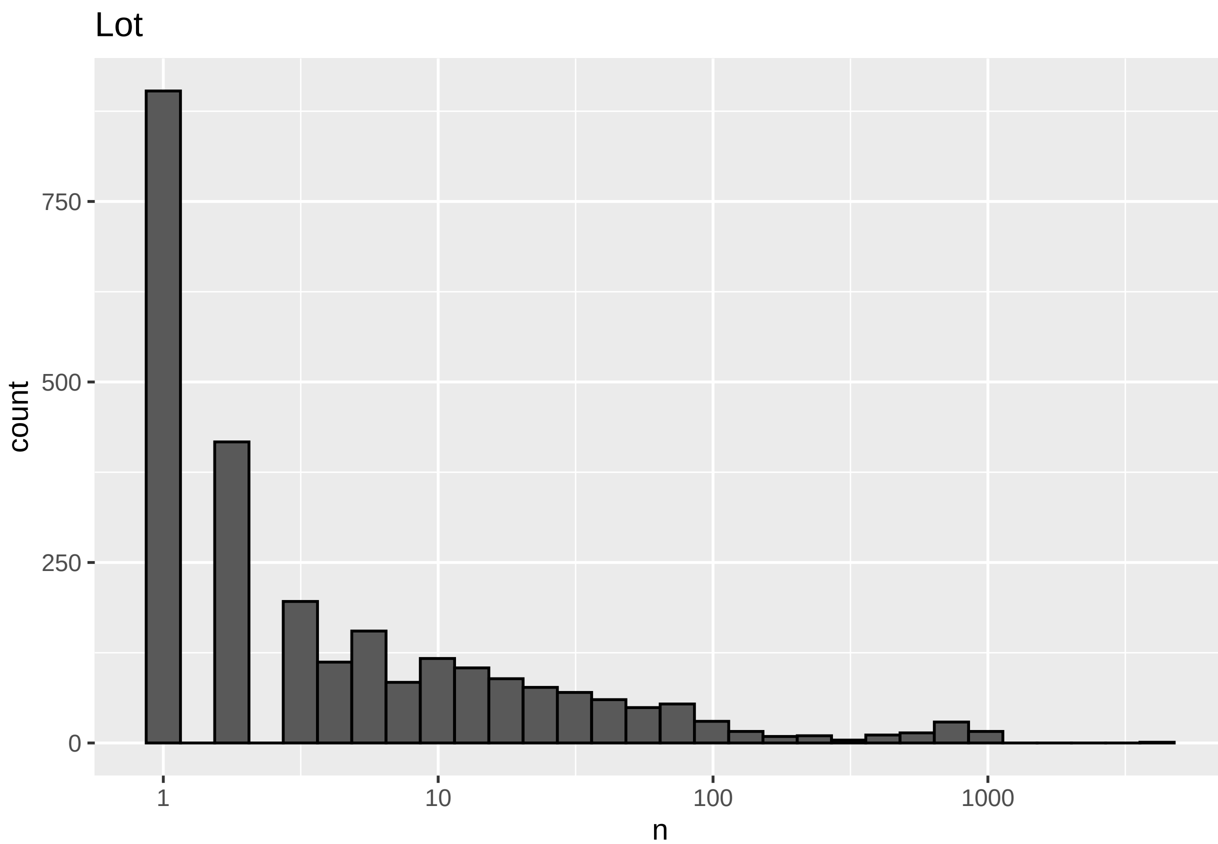
```
lot <- as.matrix(sale["LOT"])
n_distinct(lot)
```

```
## [1] 2627
```

```
lot <- as.numeric(lot)
```

In lot, 2627 categories exist. Thus, a histogram may be easier than a table.

```
datx <- dat %>% mutate(lot = lot)

datx %>%
  count(lot) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Lot")
```



"BUILDING CLASS AT PRESENT" : chr –> b_c_a_p

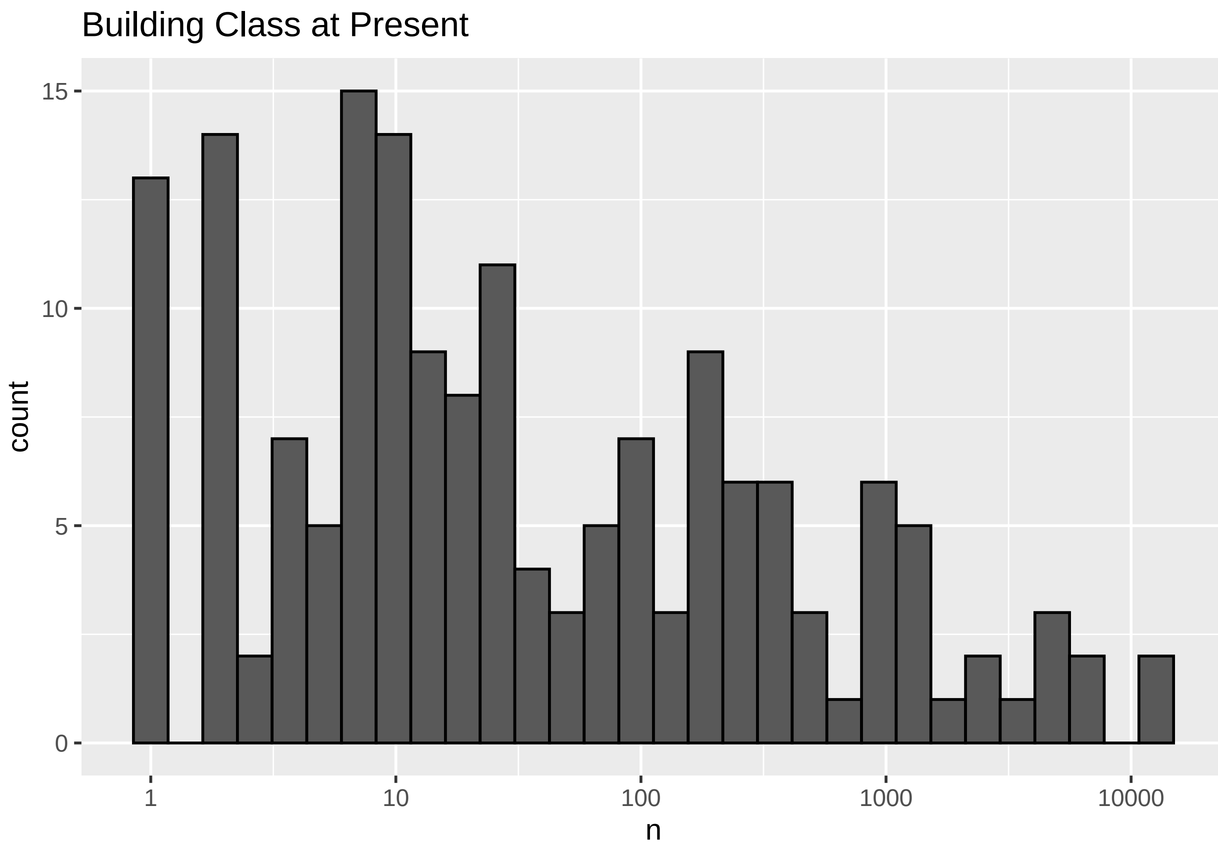
```
b_c_a_p <- as.matrix(sale["BUILDING CLASS AT PRESENT"])
n_distinct(b_c_a_p)
```

```
## [1] 167
```

```
b_c_a_p <- factor(b_c_a_p)
```

In lot, 167 categories exist. Thus, a histogram may be easier than a table.

```
dat <- dat %>% mutate(b_c_a_p = b_c_a_p)

dat %>%
  count(b_c_a_p) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Building Class at Present")
```



"ADDRESS" : chr –> address

```
address <- as.matrix(sale["ADDRESS"])

n_distinct(address)
```

```
## [1] 67563
```

address has 67563, which is over price unique value. Thus, address won't be used in predicton.

"APARTMENT NUMBER" : chr Because address is discarded, apartment number is useless.

"ZIP CODE" : double –> zip

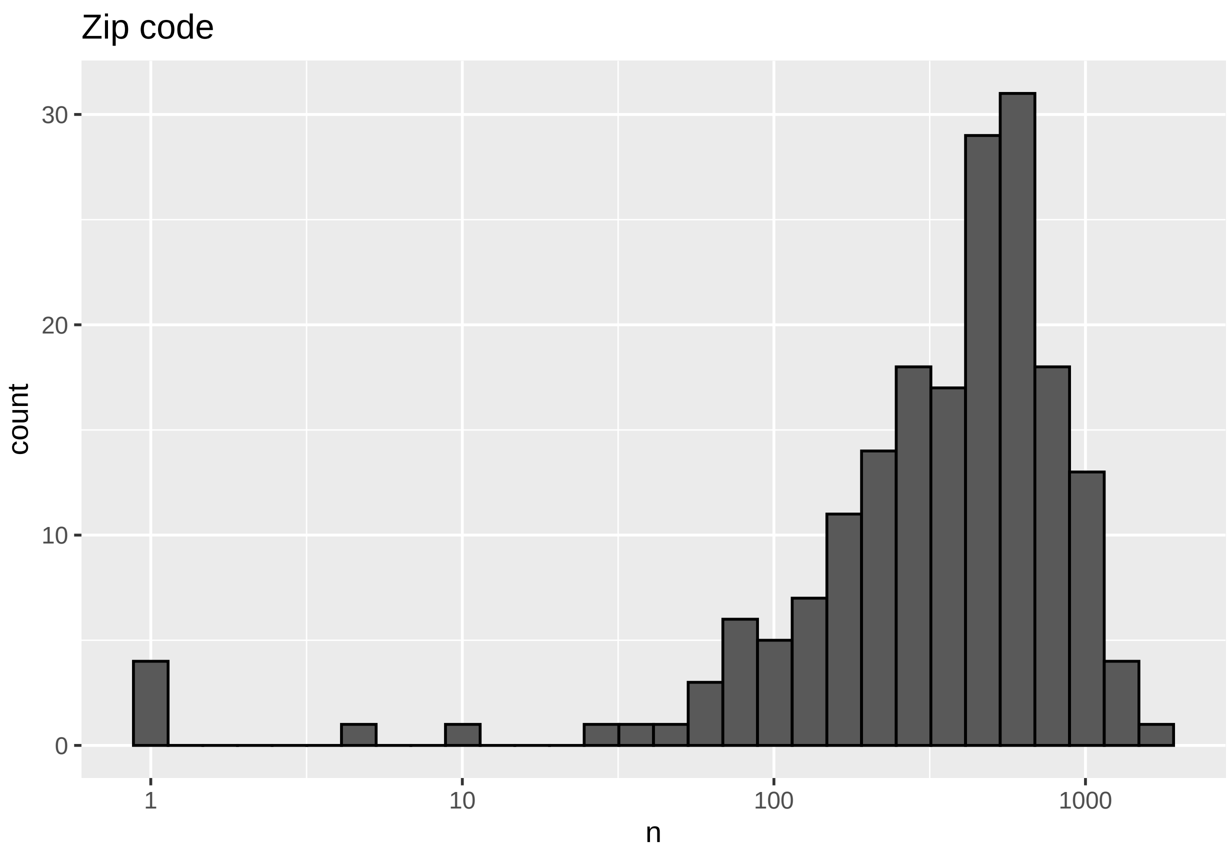
```
zip <- as.matrix(sale["ZIP CODE"])

n_distinct(zip)
```

```
## [1] 186
```

```
zip <- factor(zip)
```

In zip code case, 186 categories exist. Thus, a histogram may be easier than a table.

```r
dat <- dat %>% mutate(zip = zip)

dat %>%
  count(zip) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Zip code")
```



"RESIDENTIAL UNITS" : double –> resi

```r
resi <- as.matrix(sale["RESIDENTIAL UNITS"])

n_distinct(resi)
```

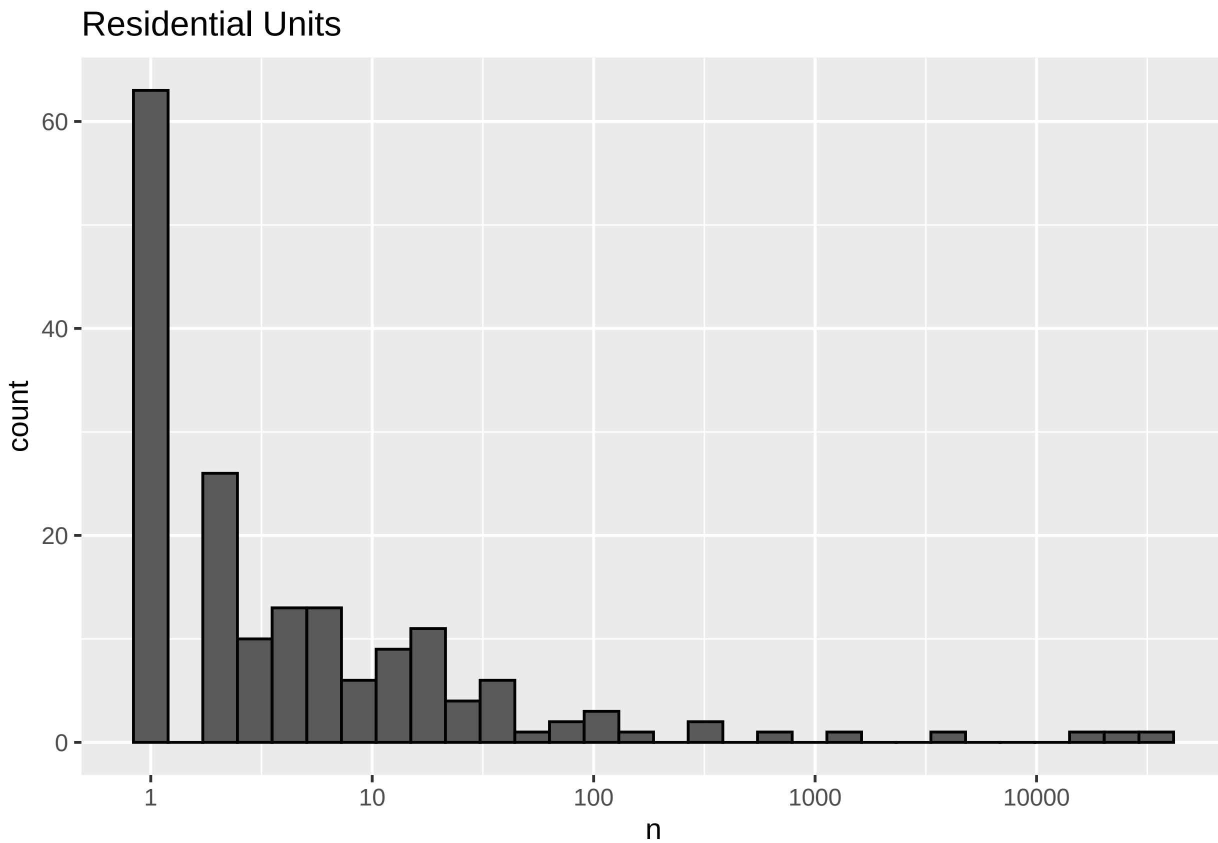
```
## [1] 176
```

```r
resi <- factor(resi)
```

In residential unit case, 176 categories exist. Thus, a histogram may be easier than a table.

```r
dat <- dat %>% mutate(resi = resi)

dat %>%
  count(resi) %>%
```

```
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Residential Units")
```



"COMMERCIAL UNITS" : double –> comm

```
comm <- as.matrix(sale["COMMERCIAL UNITS"])
n_distinct(comm)
```

```
## [1] 55
```

```
comm <- factor(comm)
```

In commerical unit case, 55 categories exist.

```
dat <- dat %>% mutate(comm = comm)
tab <- dat %>% count(comm)
tab %>% kable()
```

comm	n
0	79429
1	3558

comm	n
2	817
3	259
4	137
5	74
6	70
7	31
8	26
9	20
10	17
11	10
12	12
13	4
14	6
15	11
16	2
17	6
18	3
19	3
20	4
21	1
22	3
23	1
24	1
25	2
26	2
27	1
28	1
30	1
31	1
32	1
34	1
35	4
38	1
42	3
49	1
51	1
52	1
55	1
56	1
59	1
62	1
67	1
73	1
91	1
126	2
147	1
172	1
184	1
254	4
318	1
422	2
436	2

comm	n
2261	1

"TOTAL UNITS" : double -> tot

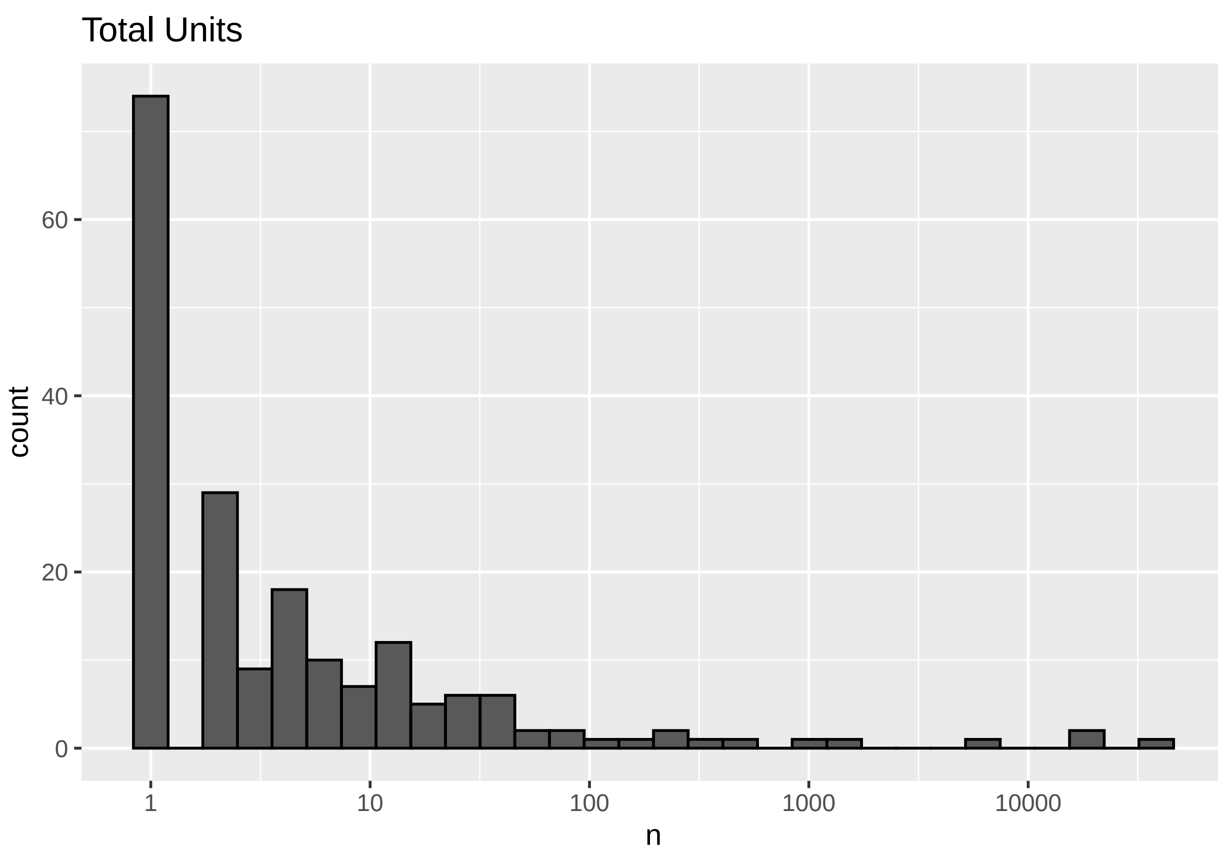
```
tot <- as.matrix(sale["TOTAL UNITS"])
n_distinct(tot)
```

```
## [1] 192
```

```
tot <- factor(tot)
```

In total unit case, 192 categories exist. Thus, a histogram may be easier than a table.

```
dat <- dat %>% mutate(tot = tot)

dat %>%
  count(tot) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Total Units")
```



"LAND SQUARE FEET" : chr –> l_sqf

```r
l_sqf <- as.matrix(sale["LAND SQUARE FEET"])

n_distinct(l_sqf)
```

```
## [1] 6062
```

Land Square feet has 6062 unique values, which is more than a half of price unique values. Thus, not used. However, one can round the numbers.

```r
l_sqf <- as.numeric(l_sqf)
```

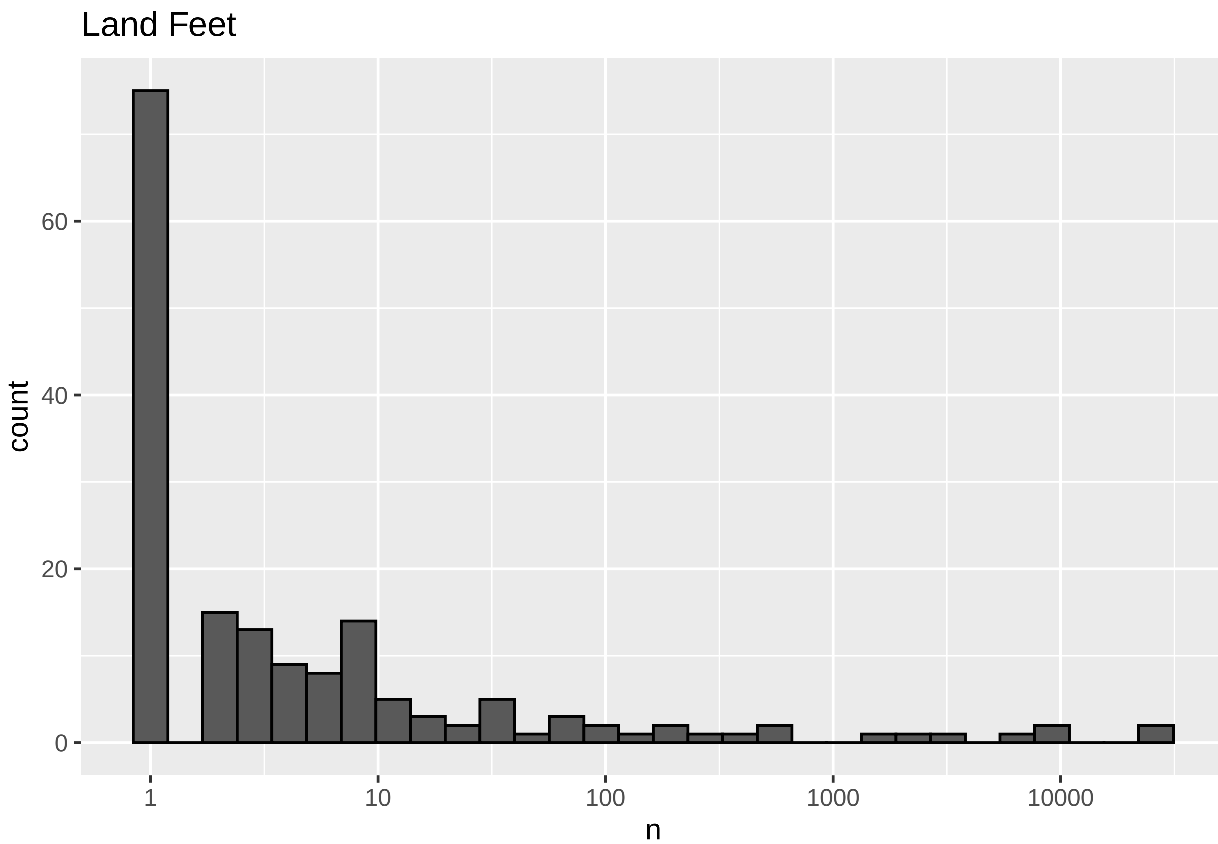
```
## Warning: NAs introduced by coercion
```

```r
l_sqf <- round(l_sqf, digits = -3)

n_distinct(l_sqf)
```

```
## [1] 170
```

unique number is reduced to 170. Thus, a histogram may be easier than a table.

```r
datx <- dat %>% mutate(l_sqf = l_sqf)

datx %>%
  count(l_sqf) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Land Feet")
```



NA is replaced with the average.

```
ml_sqf = mean(l_sqf, na.rm = T)

l_sqf <- sapply(l_sqf, function(l) {
  ifelse(!is.na(l), l, ml_sqf)
})

dat <- dat %>% mutate(l_sqf = l_sqf)
```

"GROSS SQUARE FEET" : chr –> g_sqrf

```
g_sqrf <- as.matrix(sale["GROSS SQUARE FEET"])
n_distinct(g_sqrf)
```

```
## [1] 5691
```

gross square feet has 5691 unique values, which is more than a half of price unique values. Thus, not used. however, one can round the numbers.

```
g_sqrf <- as.numeric(g_sqrf)
```

```
## Warning: NAs introduced by coercion
```

```r
g_sqrf <- round(g_sqrf, digits = -3)

n_distinct(g_sqrf)
```

```
## [1] 239
```

unique number is 239

NA is replaced with the average.

```r
mg_sqf = mean(g_sqrf, na.rm = T)

g_sqrf <- sapply(g_sqrf, function(l) {
  ifelse(!is.na(l), l, mg_sqf)
})

dat <- dat %>% mutate(g_sqrf = g_sqrf)
```

"YEAR BUILT" : double –> year_buit

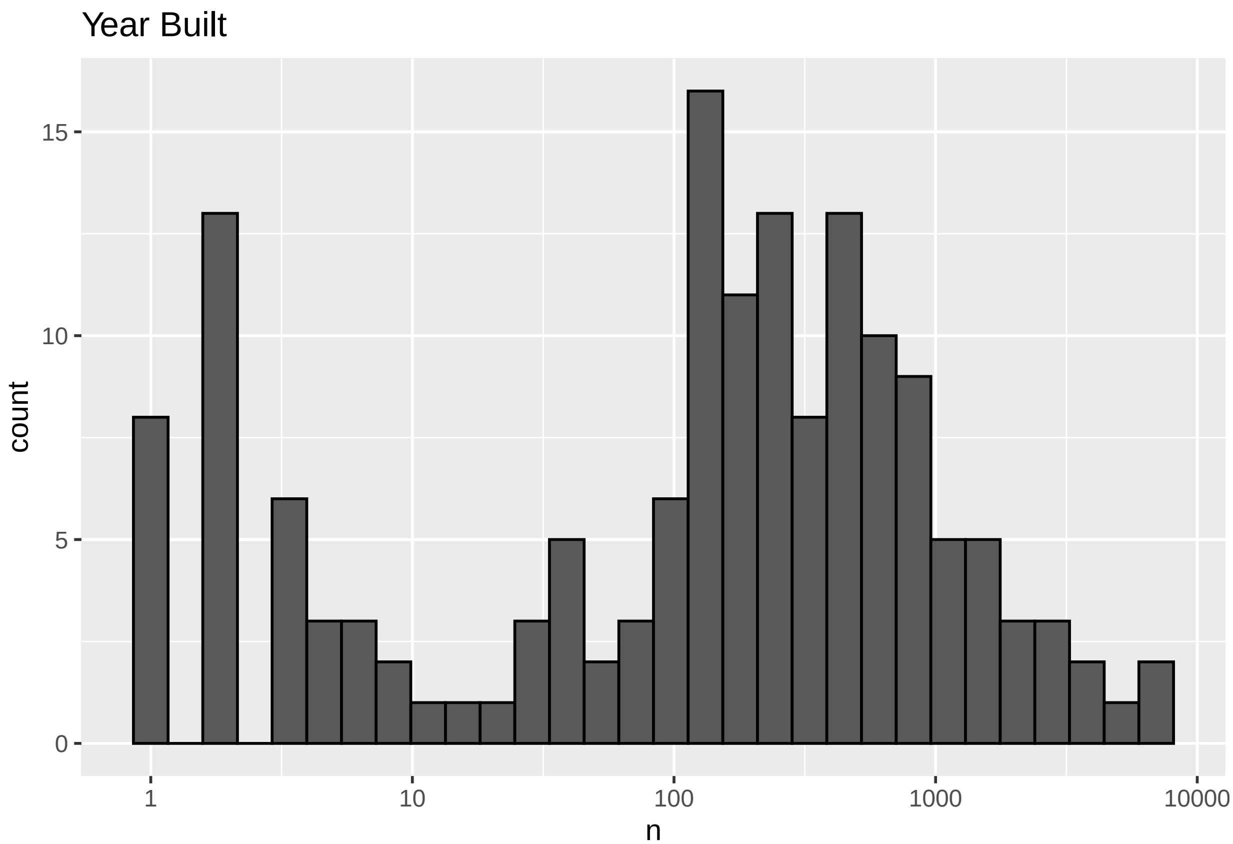
```r
year_buit <- as.matrix(sale["YEAR BUILT"])

n_distinct(year_buit)
```

```
## [1] 158
```

```r
year_buit <- factor(year_buit)
```

In year built case, 158 categories exist. Thus, a histogram may be easier than a table.

```r
dat <- dat %>% mutate(year_buit = year_buit)

dat %>%
  count(year_buit) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Year Built")
```



"TAX CLASS AT TIME OF SALE" : double –> t_c_a_s

```
t_c_a_s <- as.matrix(sale["TAX CLASS AT TIME OF SALE"])
n_distinct(t_c_a_s)
```

```
## [1] 4
```

```
t_c_a_s <- factor(as.character(t_c_a_s))
```

Tax class at the time of sale has 4 univque values. table is shown.

```
dat <- dat %>% mutate(t_c_a_s = t_c_a_s)
tab <- dat %>% count(t_c_a_s)
tab %>% kable()
```

t_c_a_s	n
1	41533
2	36726
3	4
4	6285

"BUILDING CLASS AT TIME OF SALE" : chr –> b_c_a_s

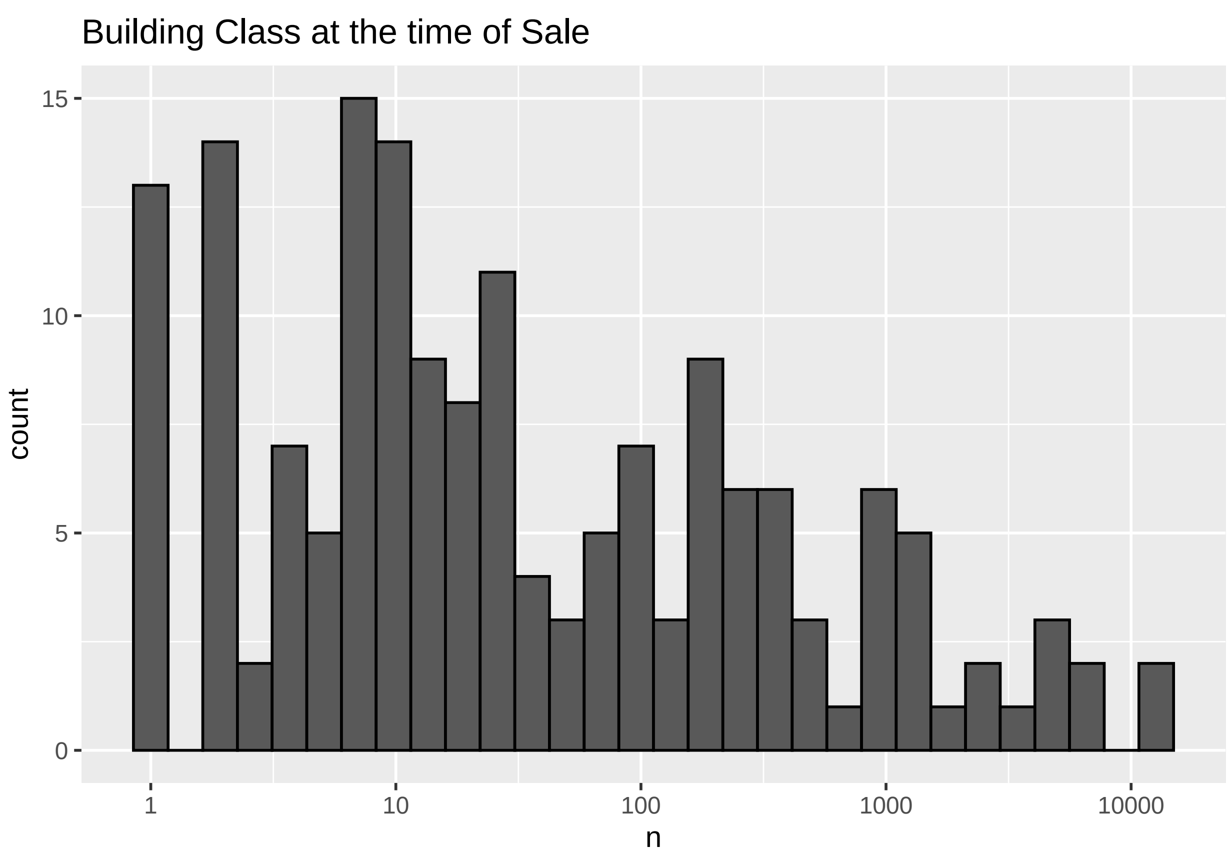
```r
b_c_a_s <- as.matrix(sale["BUILDING CLASS AT TIME OF SALE"])
n_distinct(b_c_a_s)
```

```
## [1] 166
```

```r
b_c_a_s <- factor(b_c_a_p)
```

In building class at the time of sale case, 166 categories exist. Thus, a histogram may be easier than a table.

```r
dat <- dat %>% mutate(b_c_a_s = b_c_a_s)

dat %>%
  count(b_c_a_s) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Building Class at the time of Sale")
```



checking the collection, dat.

```r
names(dat)
```

```
##  [1] "price"     "boro"      "neigh"     "b_cat"     "tax_c_a_p" "b_c_a_p"
##  [7] "zip"       "resi"      "comm"      "tot"       "l_sqf"     "g_sqrf"
## [13] "year_buit" "t_c_a_s"   "b_c_a_s"
```

Now, the tax class at present (tax_c_a_p) need to be modified. first remove NA and associated values.

```
dat <- dat %>% filter(tax_c_a_p != "NA")
```

original table

```
tabtax %>% kable()
```

tax_c_a_p	n
1	38633
1A	1444
1B	1234
1C	186
2	30919
2A	2521
2B	814
2C	1915
3	4
4	6140
NA	738

table after removing NA

```
tab <- dat %>% count(tax_c_a_p)
tabtax <- tab
tab %>% kable()
```

tax_c_a_p	n
1	38633
1A	1444
1B	1234
1C	186
2	30919
2A	2521
2B	814
2C	1915
3	4
4	6140

Now, combine 1A, 1B, and 1C to 1. 2A, 2B, and 2C to 2

table for comparison

```
dat$tax_c_a_p[dat$tax_c_a_p %in%  c("1A", "1B", "1C")] <- "1"
#dat$tax_c_a_p[dat$tax_c_a_p == "1B"] <- "1"
#dat$tax_c_a_p[dat$tax_c_a_p == "1C"] <- "1"
dat$tax_c_a_p[dat$tax_c_a_p %in%  c("2A", "2B", "2C")] <- "2"
#dat$tax_c_a_p[dat$tax_c_a_p == "2B"] <- "2"
#dat$tax_c_a_p[dat$tax_c_a_p == "2C"] <- "2"
```

```
tax_c_a_p <- as.numeric(dat$tax_c_a_p)
tax_c_a_p <- as.character(tax_c_a_p)
tax_c_a_p <- factor(tax_c_a_p)

dat <- dat %>% select(price, boro, neigh, b_cat, b_c_a_p,
                      zip, resi, comm, tot, l_sqf,
                      g_sqrf, year_buit, t_c_a_s, b_c_a_s)

dat <- dat %>% mutate(tax_c_a_p = tax_c_a_p)

tab <- dat %>% count(tax_c_a_p)
tabtax <- tab
tab %>% kable()
```

tax_c_a_p	n
1	41497
2	36169
3	4
4	6140

**Final colled data checking

```
str(dat)
```

```
## 'data.frame':    83810 obs. of  15 variables:
##  $ price    : num  6620000 1280000 1280000 3940000 8000000 ...
##  $ boro     : Factor w/ 5 levels "Bronx","Brooklyn",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ neigh    : Factor w/ 254 levels "AIRPORT LA GUARDIA",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ b_cat    : Factor w/ 47 levels "01 ONE FAMILY DWELLINGS",..: 7 7 7 7 7 7 7 7 8 8 ...
##  $ b_c_a_p  : Factor w/ 166 levels "A0","A1","A2",..: 16 21 21 18 16 18 18 21 30 34 ...
##  $ zip      : Factor w/ 186 levels "0","10001","10002",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ resi     : Factor w/ 176 levels "0","1","2","3",..: 6 29 17 11 7 21 9 45 16 25 ...
##  $ comm     : Factor w/ 55 levels "0","1","2","3",..: 1 4 2 1 1 1 1 3 1 1 ...
##  $ tot      : Factor w/ 192 levels "0","1","2","3",..: 6 32 18 11 7 21 9 47 16 25 ...
##  $ l_sqf    : num  2000 5000 2000 2000 2000 3000 2000 5000 2000 4000 ...
##  $ g_sqrf   : num  6000 19000 8000 7000 5000 10000 4000 21000 9000 19000 ...
##  $ year_buit: Factor w/ 158 levels "0","1111","1680",..: 41 41 41 54 41 41 61 41 61 61 ...
##  $ t_c_a_s  : Factor w/ 4 levels "1","2","3","4": 2 2 2 2 2 2 2 2 2 2 ...
##  $ b_c_a_s  : Factor w/ 166 levels "A0","A1","A2",..: 16 21 21 18 16 18 18 21 30 34 ...
##  $ tax_c_a_p: Factor w/ 4 levels "1","2","3","4": 2 2 2 2 2 2 2 2 2 2 ...
```

Correlation among variables correlation correlation among parameters and between parameters and price. For this, polycor library is needed for the use of hetcor() because the data contains numeric and caterorical factor.

Unfortunately, the hetcor() takes too long. Instead, I have used correlate(), which ignores non-numeric parameters for the correlation coefficiency calculation.

```
library(lsr)
```

```r
x <- correlate(dat)
x
```

```
##
## CORRELATIONS
## ============
## - correlation type:  pearson
## - correlations shown only when both variables are numeric
##
##           price boro neigh b_cat b_c_a_p zip resi comm tot l_sqf g_sqrf
## price         .    .     .     .       .   .    .    .   . 0.052  0.346
## boro          .    .     .     .       .   .    .    .   .     .      .
## neigh         .    .     .     .       .   .    .    .   .     .      .
## b_cat         .    .     .     .       .   .    .    .   .     .      .
## b_c_a_p       .    .     .     .       .   .    .    .   .     .      .
## zip           .    .     .     .       .   .    .    .   .     .      .
## resi          .    .     .     .       .   .    .    .   .     .      .
## comm          .    .     .     .       .   .    .    .   .     .      .
## tot           .    .     .     .       .   .    .    .   .     .      .
## l_sqf     0.052    .     .     .       .   .    .    .   .     .  0.526
## g_sqrf    0.346    .     .     .       .   .    .    .   . 0.526      .
## year_buit     .    .     .     .       .   .    .    .   .     .      .
## t_c_a_s       .    .     .     .       .   .    .    .   .     .      .
## b_c_a_s       .    .     .     .       .   .    .    .   .     .      .
## tax_c_a_p     .    .     .     .       .   .    .    .   .     .      .
##           year_buit t_c_a_s b_c_a_s tax_c_a_p
## price             .       .       .         .
## boro              .       .       .         .
## neigh             .       .       .         .
## b_cat             .       .       .         .
## b_c_a_p           .       .       .         .
## zip               .       .       .         .
## resi              .       .       .         .
## comm              .       .       .         .
## tot               .       .       .         .
## l_sqf             .       .       .         .
## g_sqrf            .       .       .         .
## year_buit         .       .       .         .
## t_c_a_s           .       .       .         .
## b_c_a_s           .       .       .         .
## tax_c_a_p         .       .       .         .
```

Final data check before predictors

```
names(dat)
dim(dat)
```

final dat has 83810 rows and 15 columns. column names are “price”, “boro”, “neigh”, “b_c_at”, “b_c_a_p”, “zip”, “resi”, “comm”, “tot”, “l_sqf”,“g_sqrf”, “year_buit”, “t_c_a_s”, “b_c_a_s”, “tax_c_a_p”.

Generating trainset and testset for predictors.

with a laptop computer, predictors are extremely slow. In order to check the validity, 1000 data are sampled from dat and named dats.

```
set.seed(1, sample.kind = "Rounding")
```

```
dat_index <- sample(seq_len(nrow(dat)), size = 1000)

dats <- dat[dat_index,]
```

dats split for a hang-out test

load caret library

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked _by_ '.GlobalEnv':
##
##     RMSE

## The following object is masked from 'package:purrr':
##
##     lift
```

set random seed to 1

```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
```

split sale into two data sets, train and a hold-out test sets. 90% trainset and 10% testset

```
test_index <- createDataPartition(y = dats$price, times = 1, p = 0.1, list = FALSE)

trainset <- dats[-test_index,]
testset <- dats[test_index,]
```

check the trainset and testset size

```
nrow(trainset)
```

```
## [1] 898
```

```
nrow(testset)
```

```
## [1] 102
```

trainset 75427 and testset 8383.

short version 898 trainset 102 testset.

ratio = 9 which is exactly what is expected (0.9/0.1 = 9).

short version, ratio = 8.803922.

Average model

In this model, one expects an average price.

regression tree model (rpart)

The model minizes against muliple class variable using the following formula.

$$\sum_{1:x,R_1(j,s)} (y_i - y\hat{_{R_1}})^2 + \sum_{1:x,R_2(j,s)} (y_i - y\hat{_{R_2}})^2$$

This is implemented in rpart package.

k-nearest neighbors (kNN) algorithm

kNN method is based on conditional probability.

$$p(x_1, x_2) = Pr(Y = 1|X_1 = x_1, X_2 = x_2)$$

This is calculated between two variabilities. Thus, it can be used for multiple classes. This algorithm is implemented in knn3 package.

Results

1. Average Model

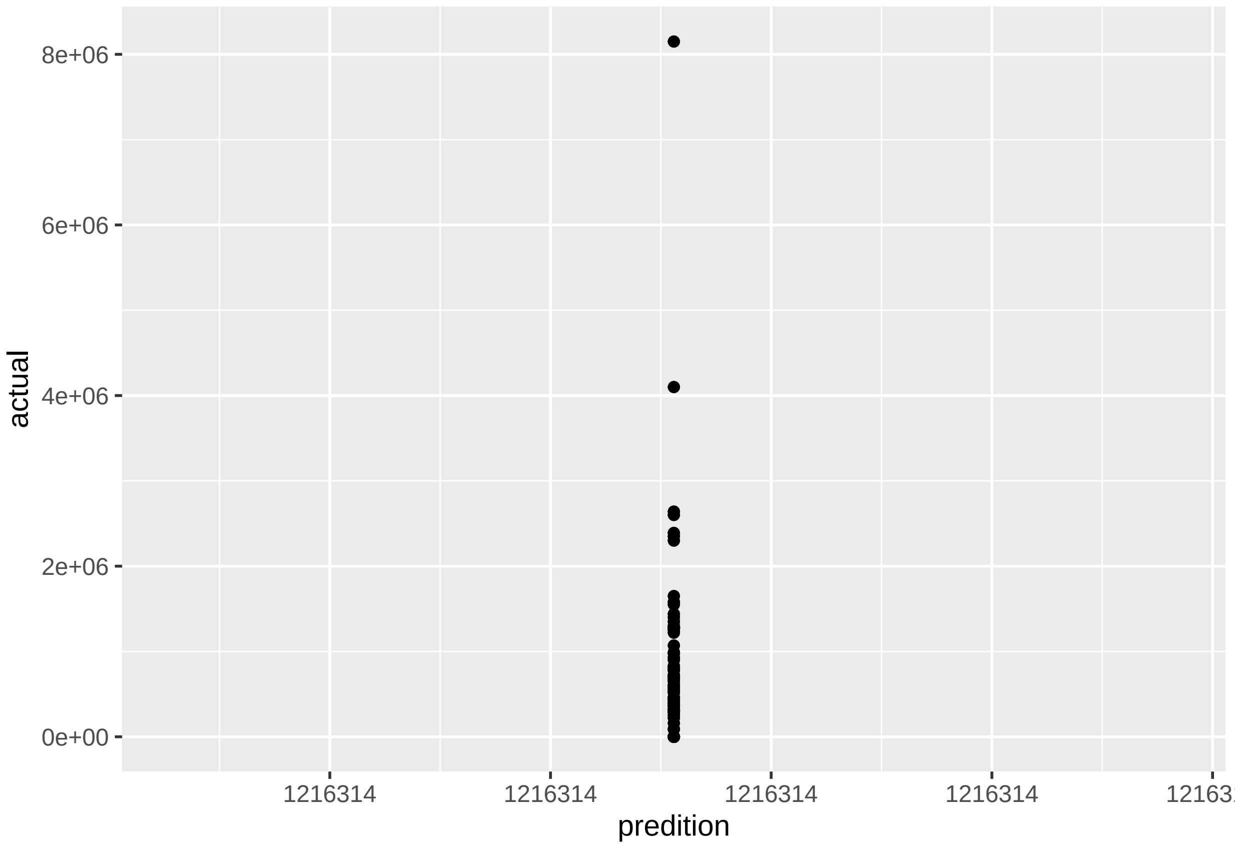
```
mu <- mean(trainset$price)
```

Thurs, prediction; rating is expected to be average for all.

```
predictions <- rep(mu, nrow(testset))
```

scatterplot prediction and actual

```
data.frame(predition = predictions, actual = testset$price) %>%
ggplot(aes(predition, actual)) +
geom_point()
```



rmse from average

```
naive_rmse <- RMSE(predictions, testset$rating)
```

storing the rmse to rmse_results

```
rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

table for rmse

```
rmse_results %>% kable()
```

method	RMSE
Just the average	NaN

Regress tree model and k-nearst neighbor algorythm

Since the calculation of these methods is slow even if smaller data, parallel computing is implemented.

How many cores in computer

```
library(parallel)

detectCores()
```

```
## [1] 4
```

My computer has 4 cores.

caret package parallel computing

```
library(doParallel)
```

```
## Loading required package: foreach
```

```
##
## Attaching package: 'foreach'
```

```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
## Loading required package: iterators
```

start of parallel computing

```
rt_f <- makePSOCKcluster(4)
registerDoParallel(rt_f)
```

2. Predictor using rpart method

```
fit_rt <- train(price ~ ., method = "rpart",
                tuneGrid = data.frame(cp = seq(0.0, 0.1, len = 25)),
                data = trainset)
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

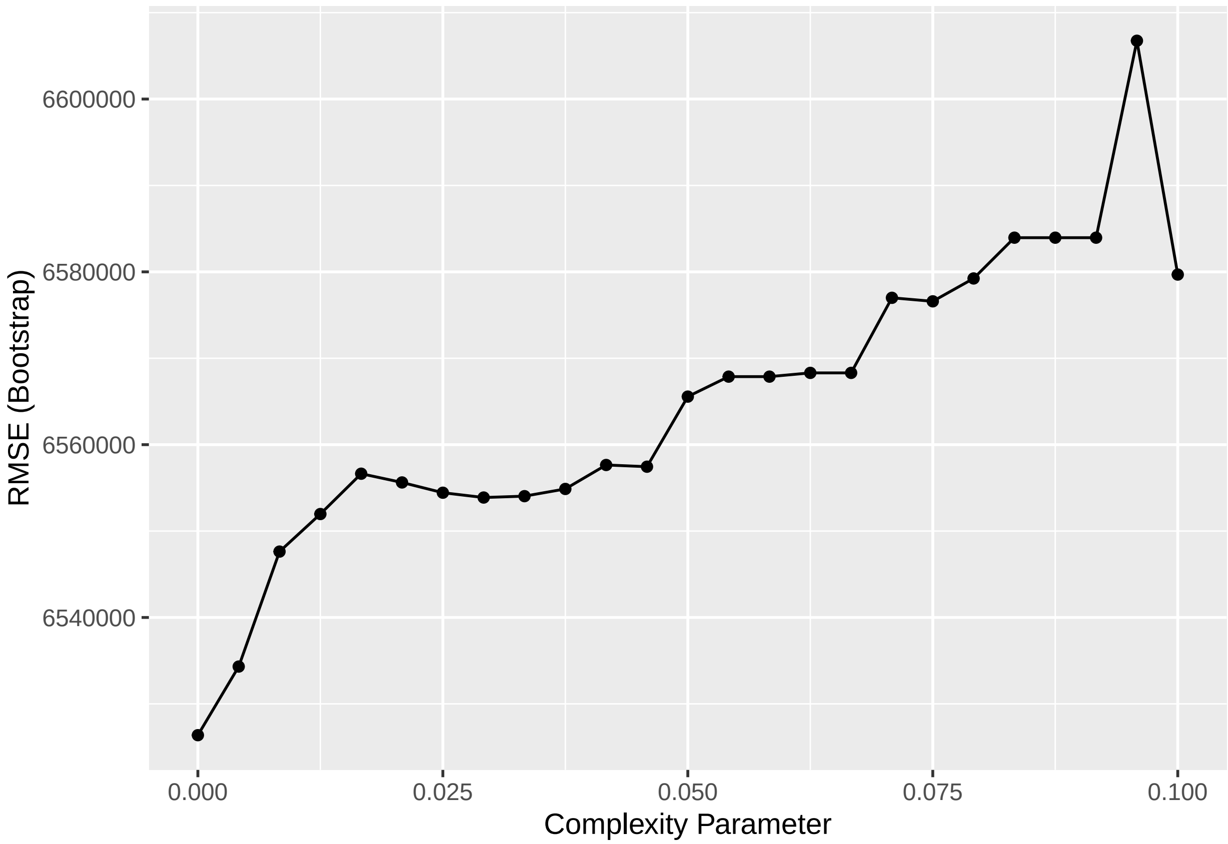
```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

When the calculation is finished. close the parallel computing

```
stopCluster(rt_f)
```

plot optimization

```r
ggplot(fit_rt)
```

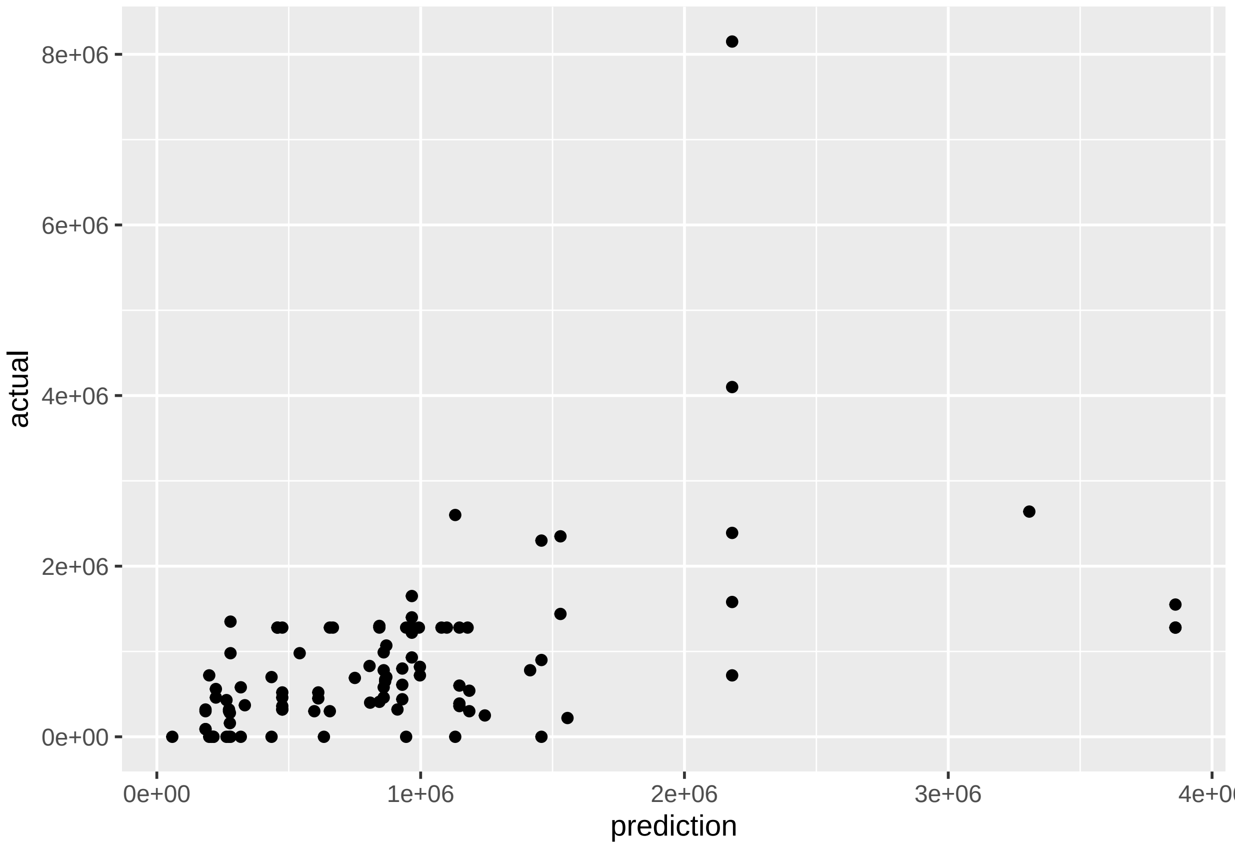


predicting price

```r
y_hat_rt <- predict(fit_rt, testset)
```

plot of prediction and actual price data.

```r
data.frame(prediction = y_hat_rt, actual = testset$price) %>%
  ggplot(aes(prediction, actual)) +
  geom_point()
```



calcuate rmse

```
rmse_rt <- RMSE(y_hat_rt, testset$price)
```

store rmse

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="rpart",
                                     RMSE = rmse_rt ))
```

print table for RMSE

```
rmse_results %>% kable()
```

method	RMSE
Just the average	NaN
rpart	912810.3

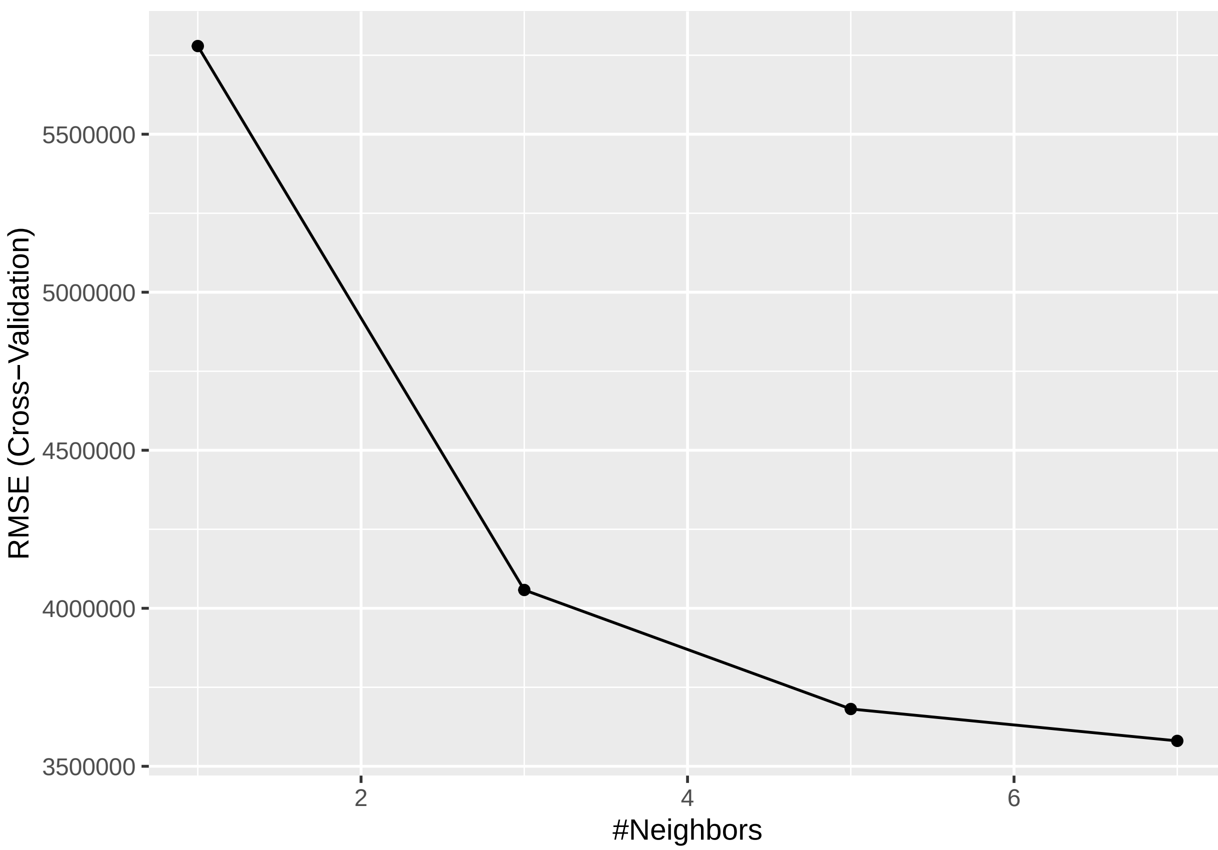
3. knn3 parameter optimization

```
control <- trainControl(method = "cv", number = 10, p = .9)
```

```
# starting parallel computing

knn_f <- makePSOCKcluster(4)
registerDoParallel(knn_f)

# Predictor using knn3

fit_knn <- train(price ~ . , method = "knn",
                 tuneGrid = data.frame(k = c(1,3,5,7)),
                 trControl = control,
                 data = trainset)
```

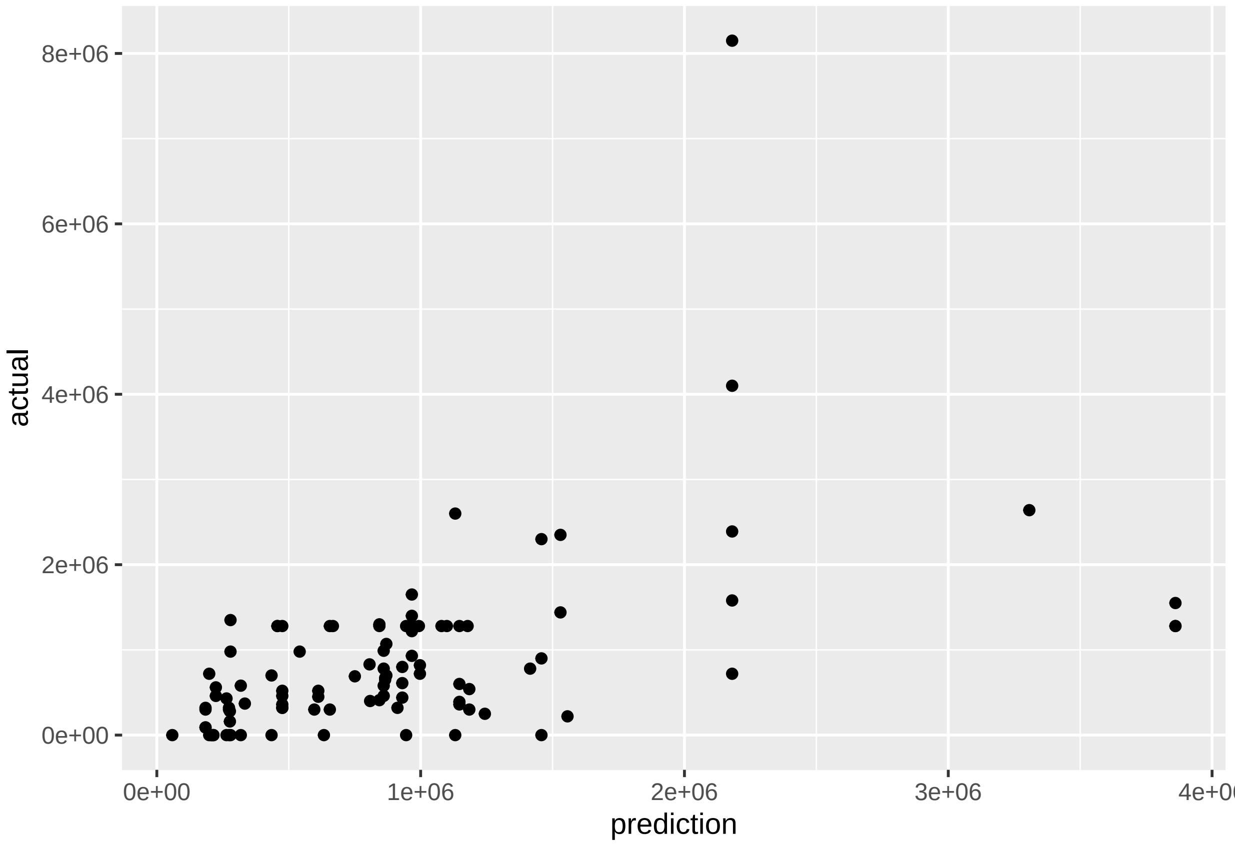
```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
## When the calculation is finished.
stopCluster(knn_f)

ggplot(fit_knn)
```




```
# predicting price
y_hat_knn <- predict(fit_knn, testset)

# plot of prediction and actual price
data.frame(prediction = y_hat_rt, actual = testset$price) %>%
```

```
ggplot(aes(prediction, actual)) +
geom_point()
```




```
# calculate rmse
rmse_knn <- RMSE(y_hat_knn, testset$price)

# store rmse
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="knn",
                                     RMSE = rmse_knn ))

# print table for RMSE

rmse_results %>% kable()
```

method	RMSE
Just the average	NaN
rpart	912810.3
knn	863218.4

Conclusion

rpart and knn3 algorythm works fine. Pricing based on other factors is significant to economics. Thus, the starting of the price predictor is quite important. However, with my current implementation, RMSE (912810 and 863218 for rpart and knn3, respectively) and predicted value are quite different from the actual value. Thus, the predictor is not even close to a half way of being perfect. This results may be due to data size because the computational power limits me to subsetting the data. Furthermore, compared to movielens, the data for NYC real estate is still small.

Limitation

Below is only limited to my implementation.

The algorythms are slow and thus, multiple cpu computer are necessary. Furthermore, the caret parallel computing package only uses CPU based. CPU is optimized for the sequential event. GPU is better fit for the parallel computing than CPU. Thus, GPU based parallel computing may be better.

Future work

Other methods, such as randomnforest, and GPU based parallel computing can be implemented.