Predicting Housing Valuations In a Volatile Economy

1 Description

In this project we analyse and predict housing value in a volatile market over a four years window.
The dataset is from kaggle.com, including the characteristics of sold houses and the microeconomics indexes.
While cleaning the data, we use ggplot to plot variables, making 24 graphs (including one interactive plot).
We use Multivariate Imputation by Chained Equations (mice), for missing variables imputation.
Finally we run a random search XGBoost with 1000 draws to find the best model, which outperforms simple regression by about 50 percent.

```
# == Data Visualisation and Wrangling == #
library(tidyverse)
library(data.table)
library(lubridate)
library(ggthemes)

# == Imputing Missing Data == #
library(mice)
library(lattice)

# == Interactive Time series == #
library(dygraphs)
library(xts)

# == XGBoost == #
library(xgboost)
library(Metrics)
```

1.0.0.1 set seed

```
set.seed(1234)
```

2 Loading data and initial prepration

2.0.0.1 set seed

```
set.seed(1234)
```

```
df = read.csv("data.csv" , header= TRUE)
macro = read.csv("macro.csv" , header= TRUE)
```

3 checking the data

The data dimensions

```
dim(df)
```

```
## [1] 30471   292
```

Converting data columns to appropriate format.

```
df$timestamp <- as.Date(df$timestamp)
macro$timestamp <- as.Date(macro$timestamp)
```

We also limit the number of variables/columns as this project is a demonstration and the resources (time/computation) are limited for intended analysis.

```
df <- df %>% select(timestamp,full_sq, life_sq, floor,
                     max_floor, build_year, num_room,
                     kitch_sq, state, material,
                     product_type, full_all, price_doc)

macro_s <- macro %>% select(timestamp,usdrub,unemployment)

dim(df)
```

```
## [1] 30471    13
```

```
dim(macro_s)
```

```
## [1] 2484    3
```

Converting data columns to appropriate format.

```
df$timestamp <- as.Date(df$timestamp)
macro$timestamp <- as.Date(macro$timestamp)
```

We join the data sets.

```
df <- df %>% left_join(macro_s)
dim(df)
```

```
## [1] 30471    15
```

The dataset includes 30471 observations and 292 columns.

```
split <- sample(c(rep(0, 0.75 * nrow(df)), rep(1, 0.25 * nrow(df))))
train = df[split == 0 , ]
test = df[split == 1 , ]
```

```
dim(train)
```

```
## [1] 22854    15
```

```
dim(test)
```

```
## [1] 7617   15
```

4 Explanatory Data Analysis

For aesthetic reasons, some outliers might have been removed from the graphs and they are not demonstrated separatly. As we move forward through data, cleaning might take place as needed.

4.1 internal house charachteritics

Here we list the house internal characteristics and analyse them

4.1.1 full_sq

Definition: total area in square meters, including loggias, balconies and other non-residential areas
Here we table the data and inspect full_Sq values. There are observations with value below 10 square meter and as they are suspicious, so we further investigate them.

```
table(train$full_sq)
```

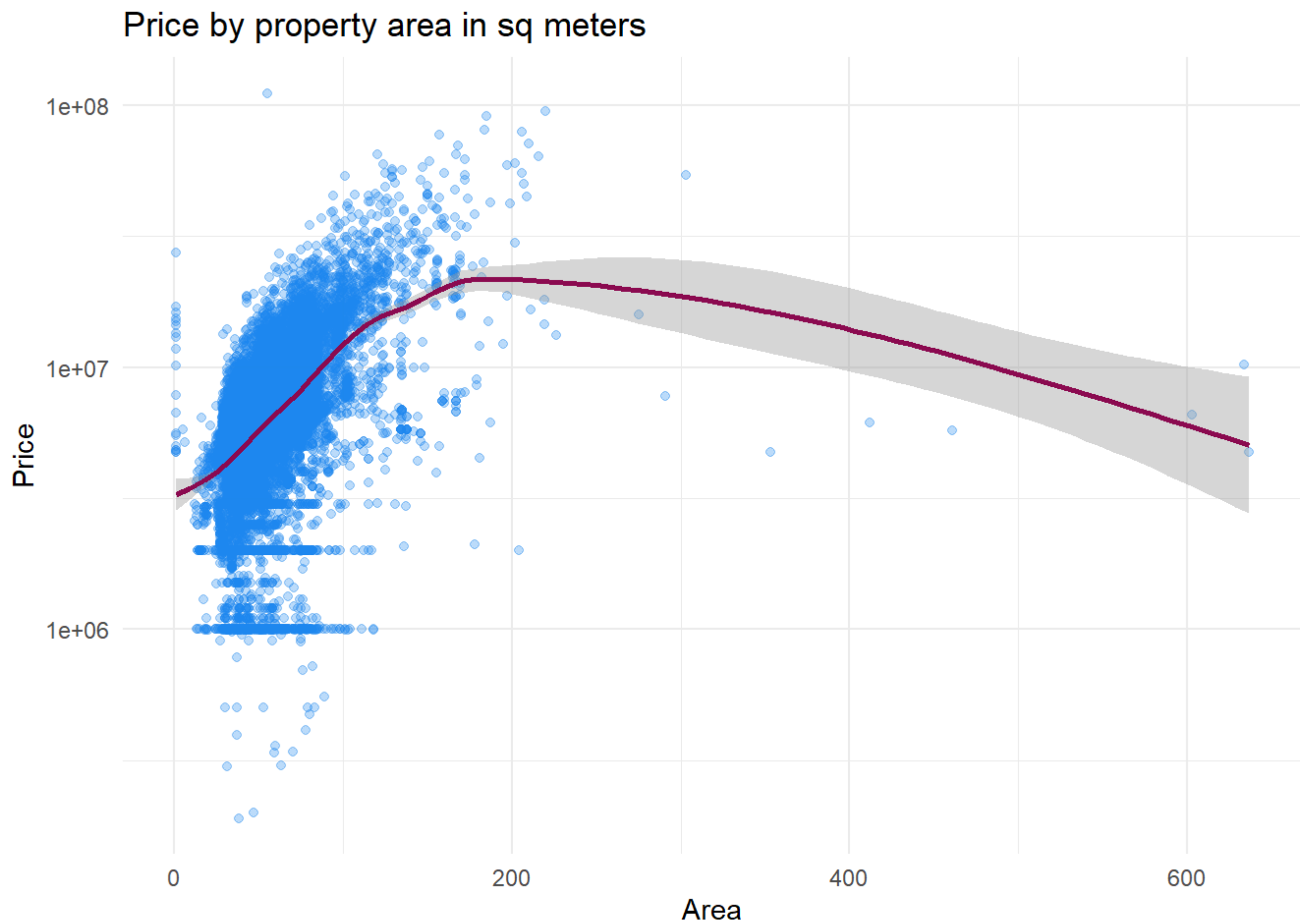
```
##
##    0    1    5    6   12   13   14   15   16   17   18   19   20   21   22   23
##    2   19    1    1    2    7    7   11    7   13   15   19   19    8    6    6
##   24   25   26   27   28   29   30   31   32   33   34   35   36   37   38   39
##   13   19   44   75   64  198  299  404  620  234  475  441  399  882 1404  750
##   40   41   42   43   44   45   46   47   48   49   50   51   52   53   54   55
##  654  581  510  628  768  688  365  265  268  181  292  582  496  493  430  314
##   56   57   58   59   60   61   62   63   64   65   66   67   68   69   70   71
##  347  347  432  427  475  404  362  477  471  228  112  135  110  115  106  118
##   72   73   74   75   76   77   78   79   80   81   82   83   84   85   86   87
##  168  186  350  241  291  333  234  344  128  102  200   92  158   99   53   40
##   88   89   90   91   92   93   94   95   96   97   98   99  100  101  102  103
##   39   47   42   28   34   24   32   35   47   28   30   29   49   36   55   25
##  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119
##   29   16   19   20   13   14   13    7   25   12   11   17   15   17   12   12
##  120  121  122  123  124  125  126  127  129  130  131  132  133  134  135  136
##   14   16    8   14   12   14   15   10    6    2    7    2    7   21   14    8
##  137  138  139  140  141  142  143  144  146  147  148  149  150  151  153  154
##    7   12    1    2    1    6    6    3    8    7    2    2    6    3    1    2
##  155  156  157  158  159  160  161  164  165  166  167  168  169  170  172  173
##    9    8    3    1    4    6    1    1    8    7   12    1    6    4    4    1
##  174  177  178  179  181  182  183  184  185  186  187  195  197  199  202  204
##    1    1    2    2    2    1    1    1    1    1    2    1    2    1    2    1
##  206  207  209  210  211  216  219  220  226  275  291  303  353  412  461  603
##    2    1    1    1    1    1    2    1    1    1    1    1    1    1    1    1
##  634  637 5326
##    1    1    1
```

If the area of a house is zero, we convert it to NA.

```
train[,"full_sq"][train[,"full_sq"] == 0] <- NA
```

The following is a scatter plot of the price by property area.

```
train %>%
    filter(full_sq < 1000) %>%
    ggplot(aes(x=full_sq, y=price_doc)) +
    geom_point(color='dodgerblue2', alpha=0.3) +
    geom_smooth(color='deeppink4') +
    scale_y_log10() +
    labs(x='Area', y='Price', title='Price by property area in sq meters') +
    theme_minimal()
```



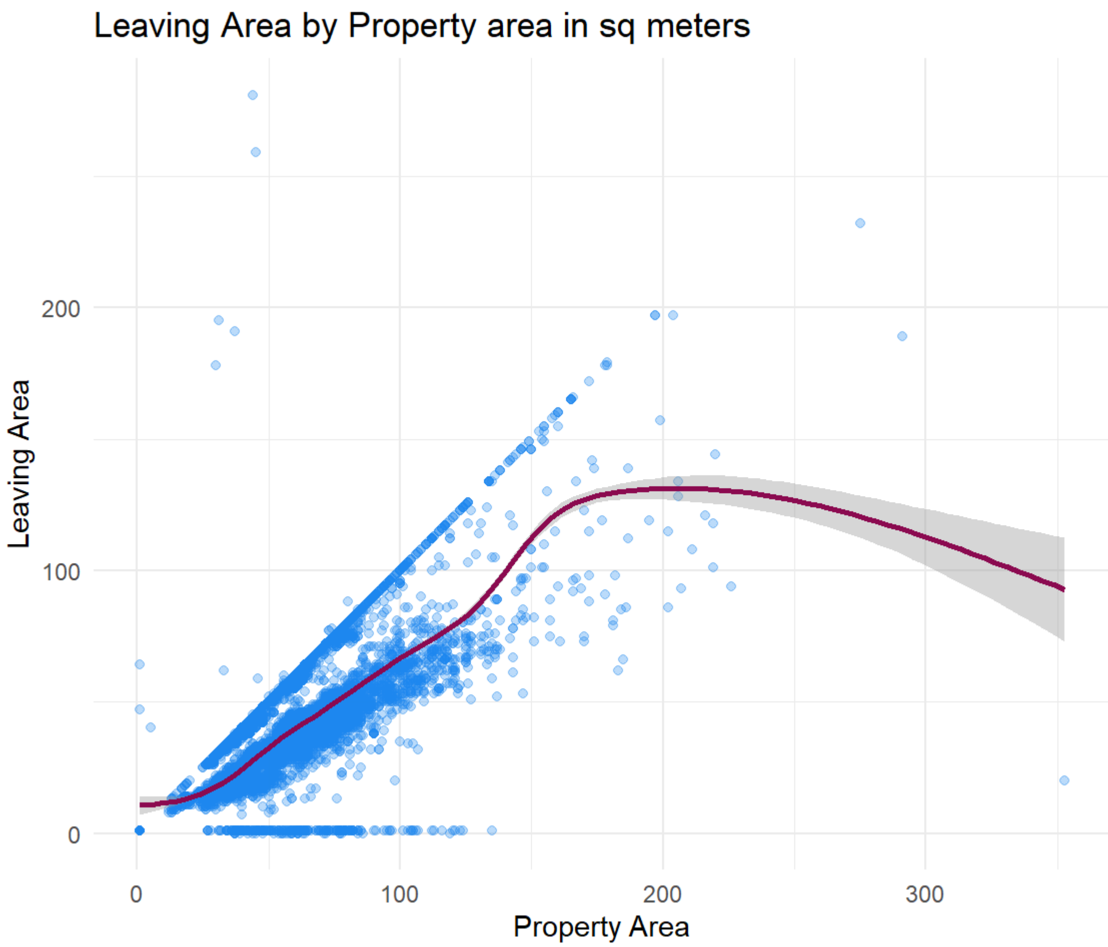
we graph the suspicious properties, those with an area below 20 square meter. As we are not able to further investigate the matter, we let them to stay as they are.

```
train %>%
    filter(full_sq < 20) %>%
    ggplot(aes(x=full_sq, y=price_doc)) +
    geom_point(color='dodgerblue2', alpha=0.4) +
    theme_minimal() +
    labs(x='Area', y='Price', title='Price by property area in sq meters - Properties under 20 m
sq')
```

4.1.2 life_sq

Next we graph leaving area against the full property area, we expect to see all values of living are below that of property area. We remove outliers from the graph to have a better view of the relation.

```
train %>%
    filter(full_sq < 400 & life_sq <300) %>%
    ggplot(aes(y=life_sq, x=full_sq)) +
    geom_point(color='dodgerblue2', alpha=0.3) +
    geom_smooth(color = 'deeppink4') +
    coord_fixed(ratio = 1)+
    labs(y='Leaving Area' , x='Property Area',
         title='Leaving Area by Property area in sq meters') +
    theme_minimal()
```

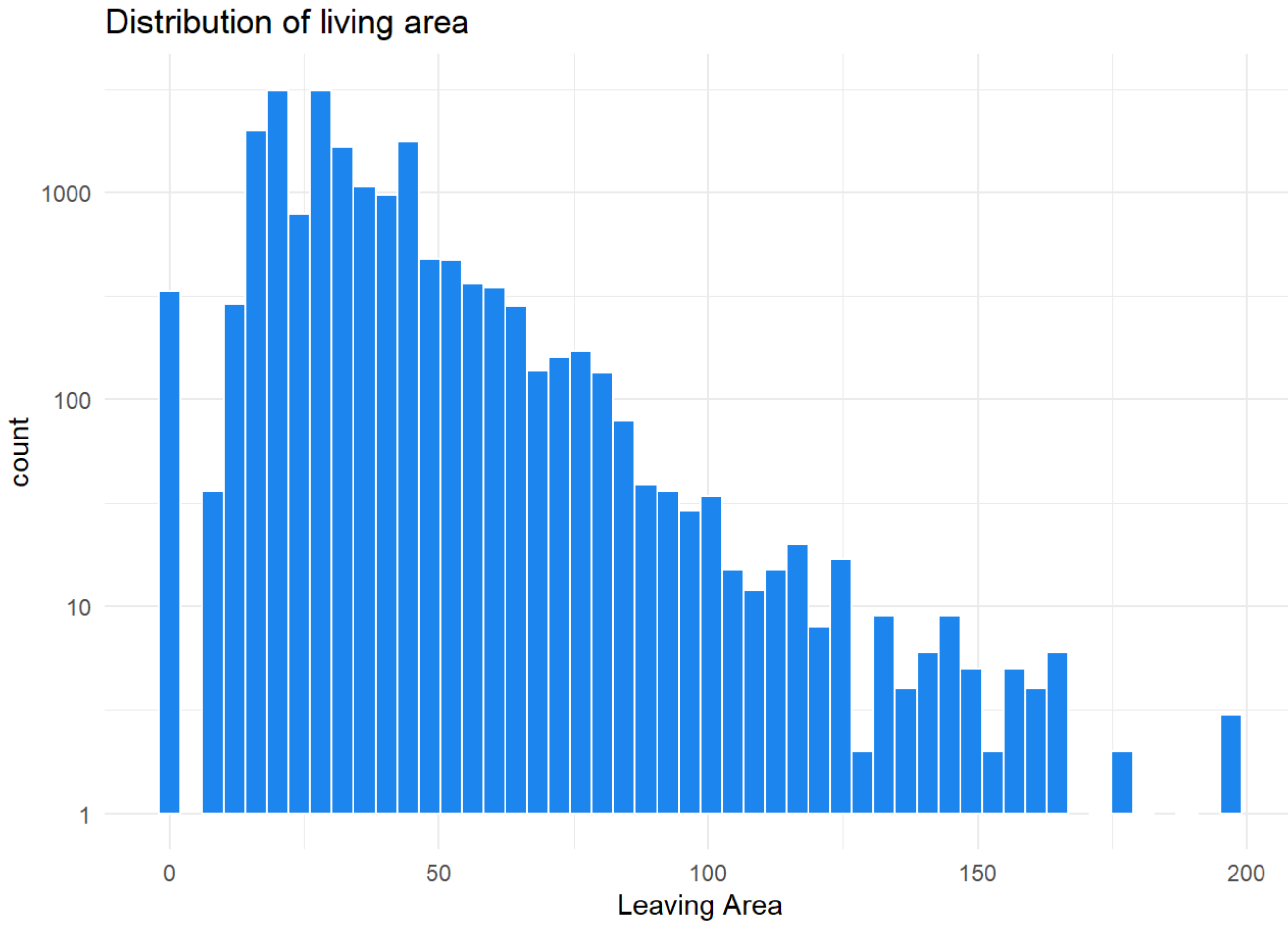


The following line of code removes the living area value of observations in which the property area is smaller than living area, as we are assuming the property value is probably more reliable.

```
train[,"life_sq"][train[,"life_sq"]>train[,"full_sq"]] <- NA
```

Now we take a look at the distribution of the leaving area.

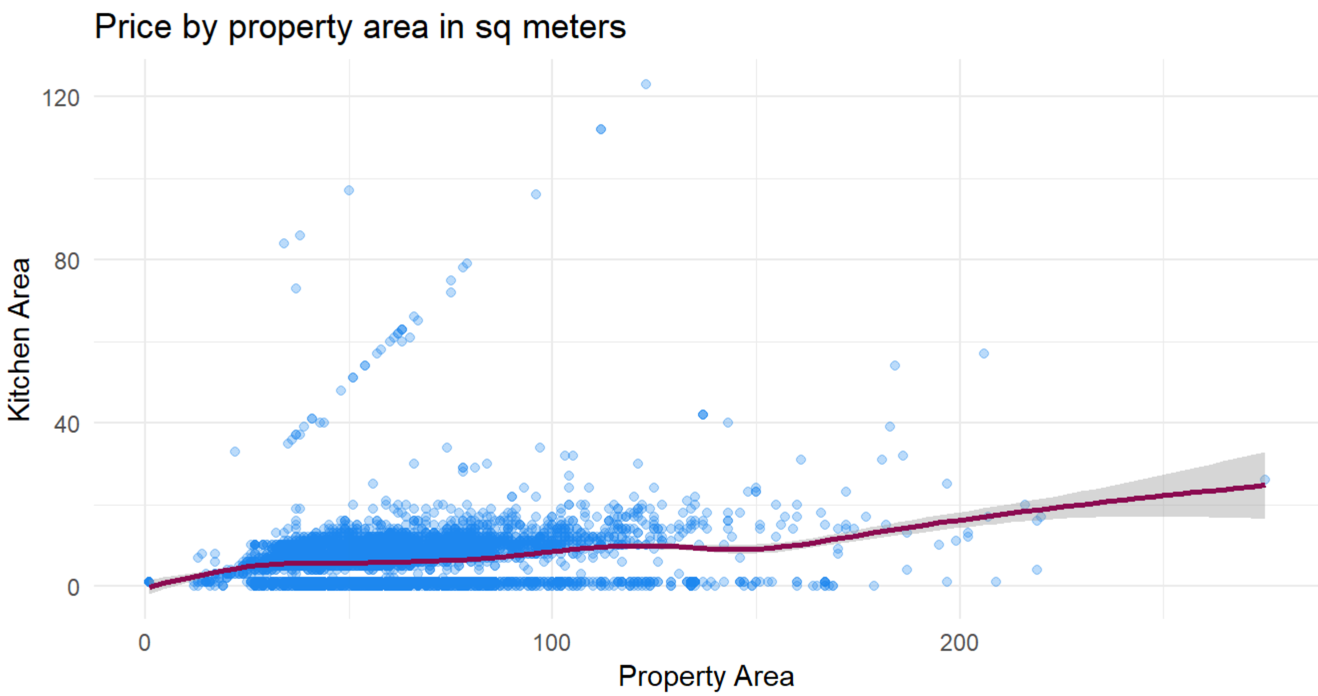
```
train %>%
    filter(full_sq < 1000 & life_sq < 200) %>%
    ggplot(aes(x=life_sq)) +
    geom_histogram(color= "white" ,fill='dodgerblue2', bins=50) +
    scale_y_log10()+
    labs(x='Leaving Area',
         title='Distribution of living area') +
    theme_minimal()
```



4.1.3 kitch_sq

we graph the area of kitchen against the property area. As one could easily justify it, the kitchen area, increases with a small slope.

```
train %>%
    filter(full_sq < 300 & kitch_sq <500) %>%
    ggplot(aes(y=kitch_sq, x=full_sq)) +
    geom_point(color='dodgerblue2', alpha=0.3) +
    geom_smooth(color = 'deeppink4') +
    coord_fixed(ratio = 1) +
    labs(y='Kitchen Area', x='Property Area',
         title='Price by property area in sq meters')+
    theme_minimal()
```

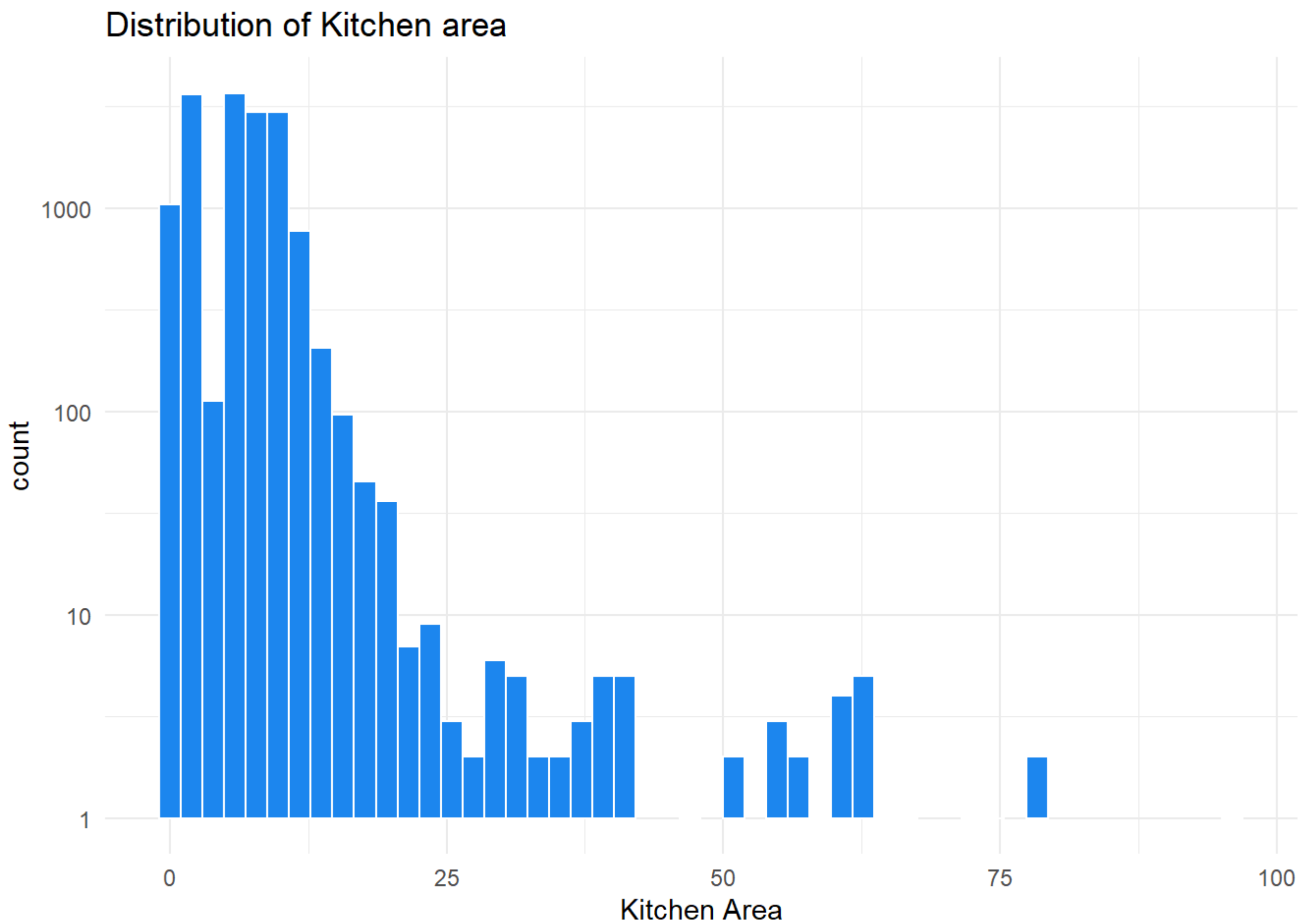


We remove kitchen values bigger than the prperty area.

```
train[,"kitch_sq"][train$kitch_sq>train$full_sq] <- NA
```

Here we have the histogram of kitchen area.

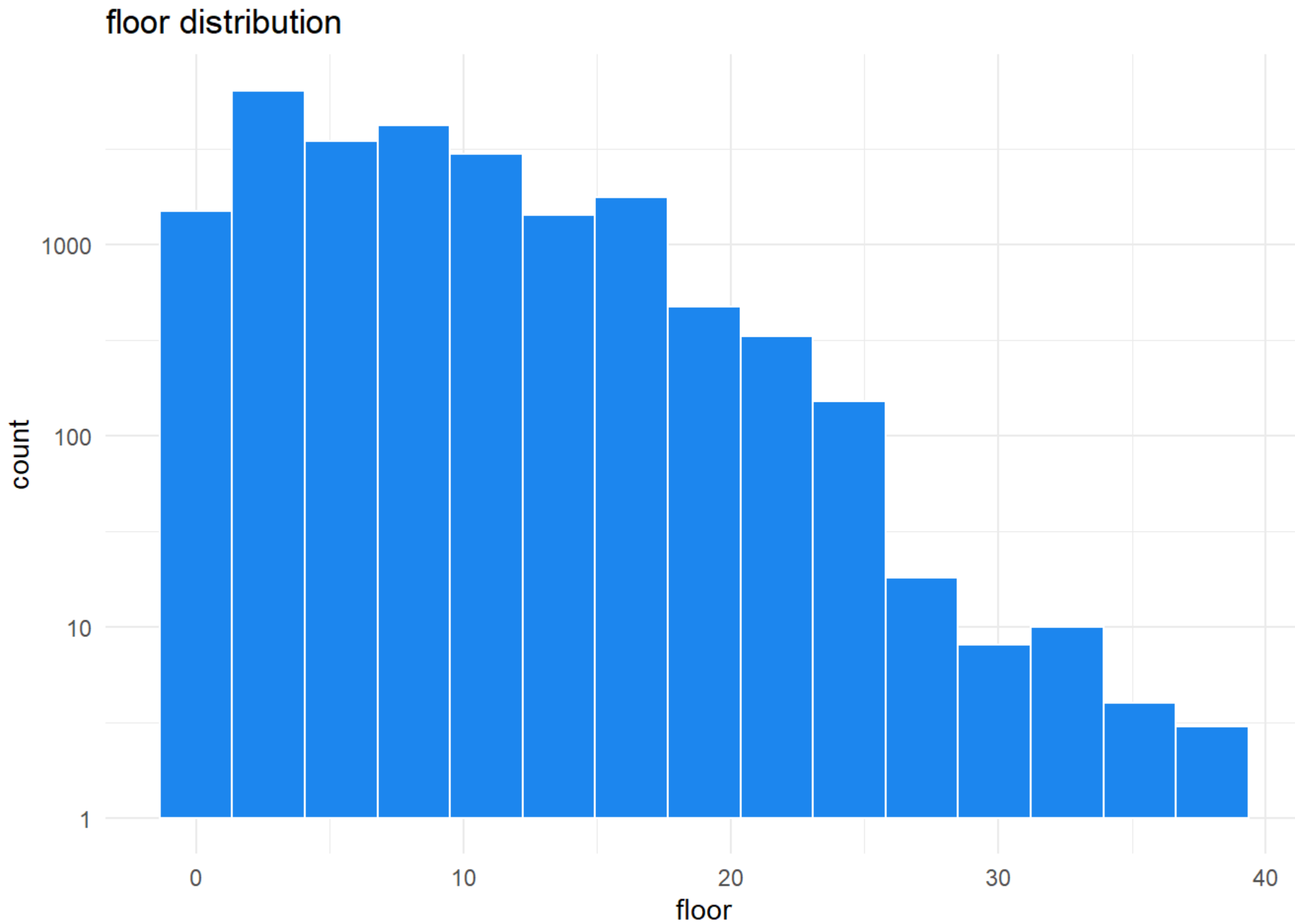
```
train %>%
    filter(kitch_sq < 100 ) %>%
    ggplot(aes(x=kitch_sq)) +
    geom_histogram(color= "white" ,fill='dodgerblue2', bins=50) +
    scale_y_log10() +
    labs(x='Kitchen Area',
         title='Distribution of Kitchen area') +
    theme_minimal()
```



4.1.4 floor

Here we have the distribution of variable floor.

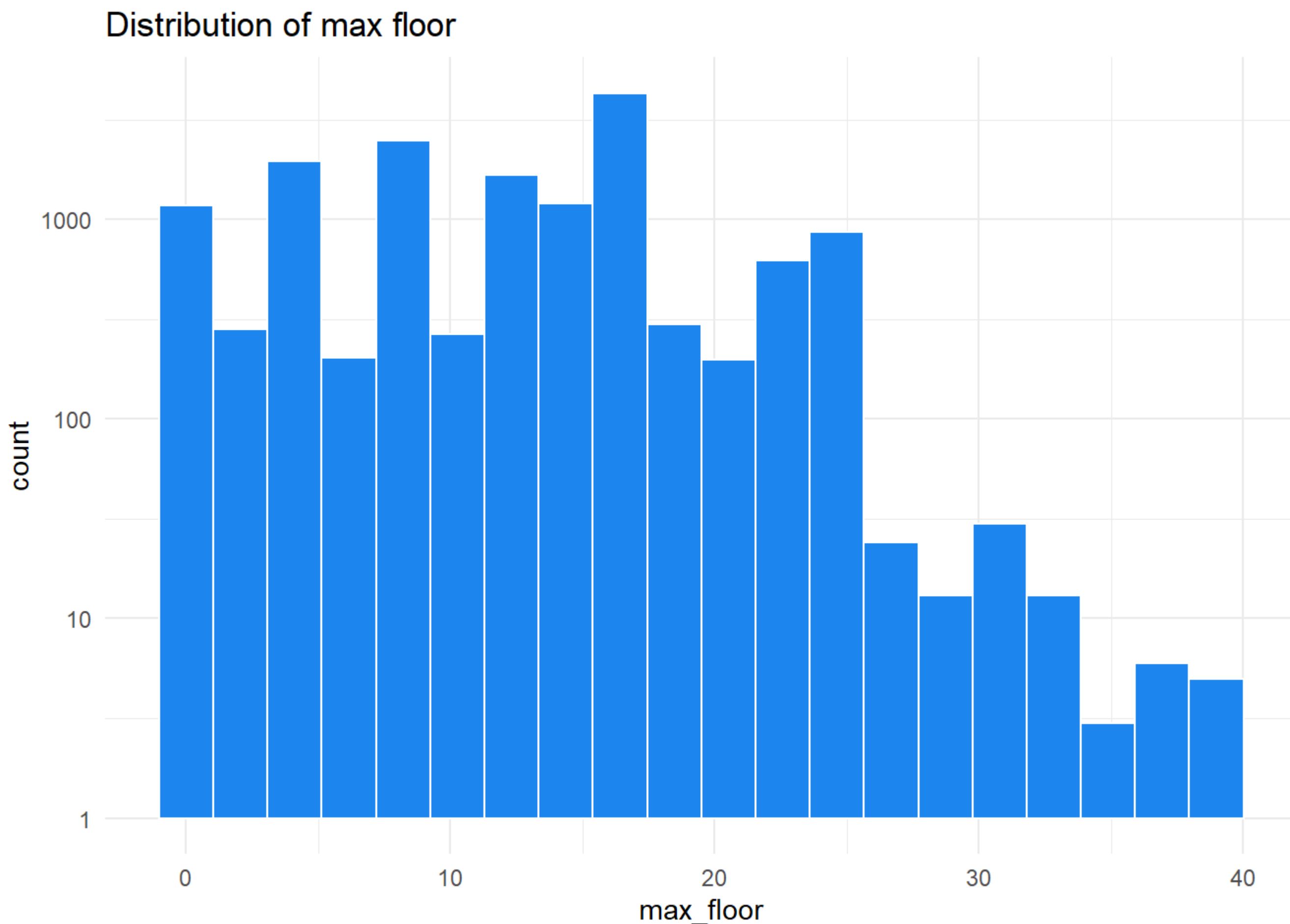
```
train %>%
    filter(floor < 40) %>%
    ggplot(aes(x=floor)) +
    geom_histogram(color= "white" ,fill='dodgerblue2', bins=15) +
    scale_y_log10() +
    labs(x='floor',
         title='floor distribution') +
    theme_minimal()
```



###

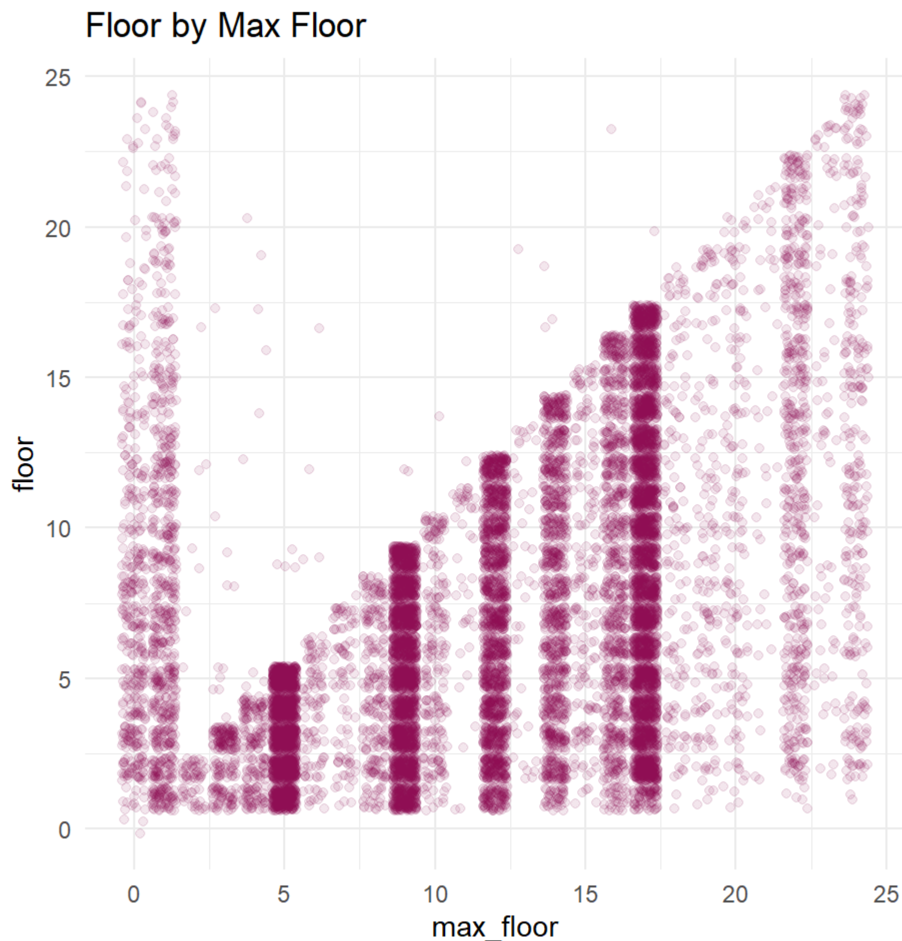
max_floor Here the max floor

```
train %>%
    filter(max_floor < 40) %>%
    ggplot(aes(x=max_floor)) +
    geom_histogram(color= "white" ,fill='dodgerblue2', bins=20) +
    scale_y_log10() +
    ggtitle('Distribution of max floor')+
    theme_minimal()
```



We check the property floor against the maximum number of floors. we cap the graph axises on 25 floors and 25 max floors.

```
train %>%
  filter(max_floor < 25 & floor < 25) %>%
  ggplot(aes(y= floor , x= max_floor)) +
  geom_jitter(color='deeppink4', alpha=0.1) +
  coord_fixed(ratio = 1) +
  labs(x='max_floor', y='floor', title='Floor by Max Floor')+
    theme_minimal()
```



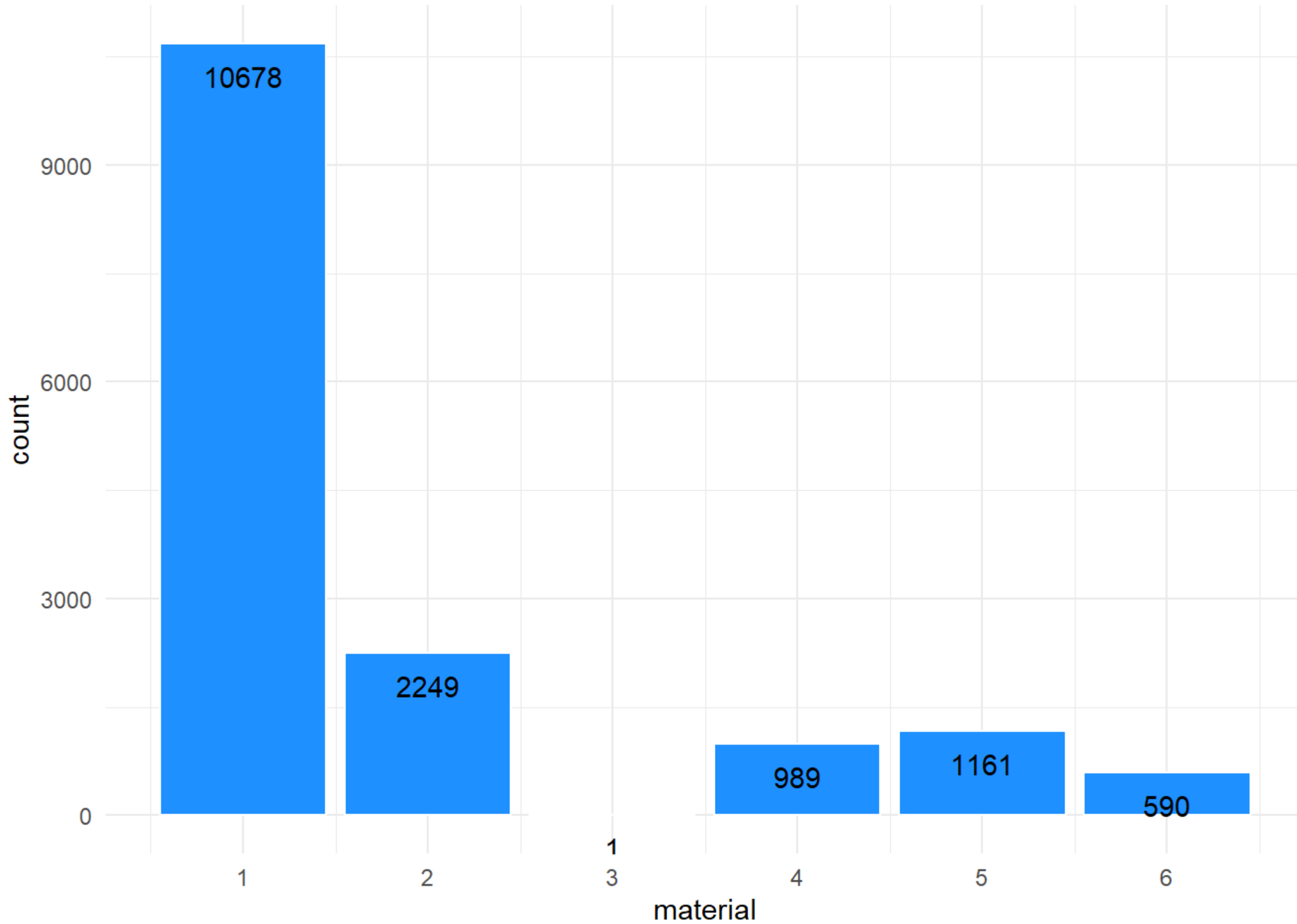
We remove max_floors that are smaller than floors.

```
train$max_floor[train$max_floor<train$floor] <- NA
```

4.1.5 material

Here we table the material of the each house. We don’t have list to know what the materials actually are./ There is only one observation with material 1.

```
train %>%
  ggplot( aes(x=material)) +
  geom_bar(fill = "dodgerblue1", color = "white") +
  scale_x_continuous(breaks = seq(1,6,1)) +
  geom_text(stat='count', aes(label=..count..), vjust=2)+
    theme_minimal()
```



4.1.6 build_year

We first inspect the data using table command.

```
table(train$build_year)
```

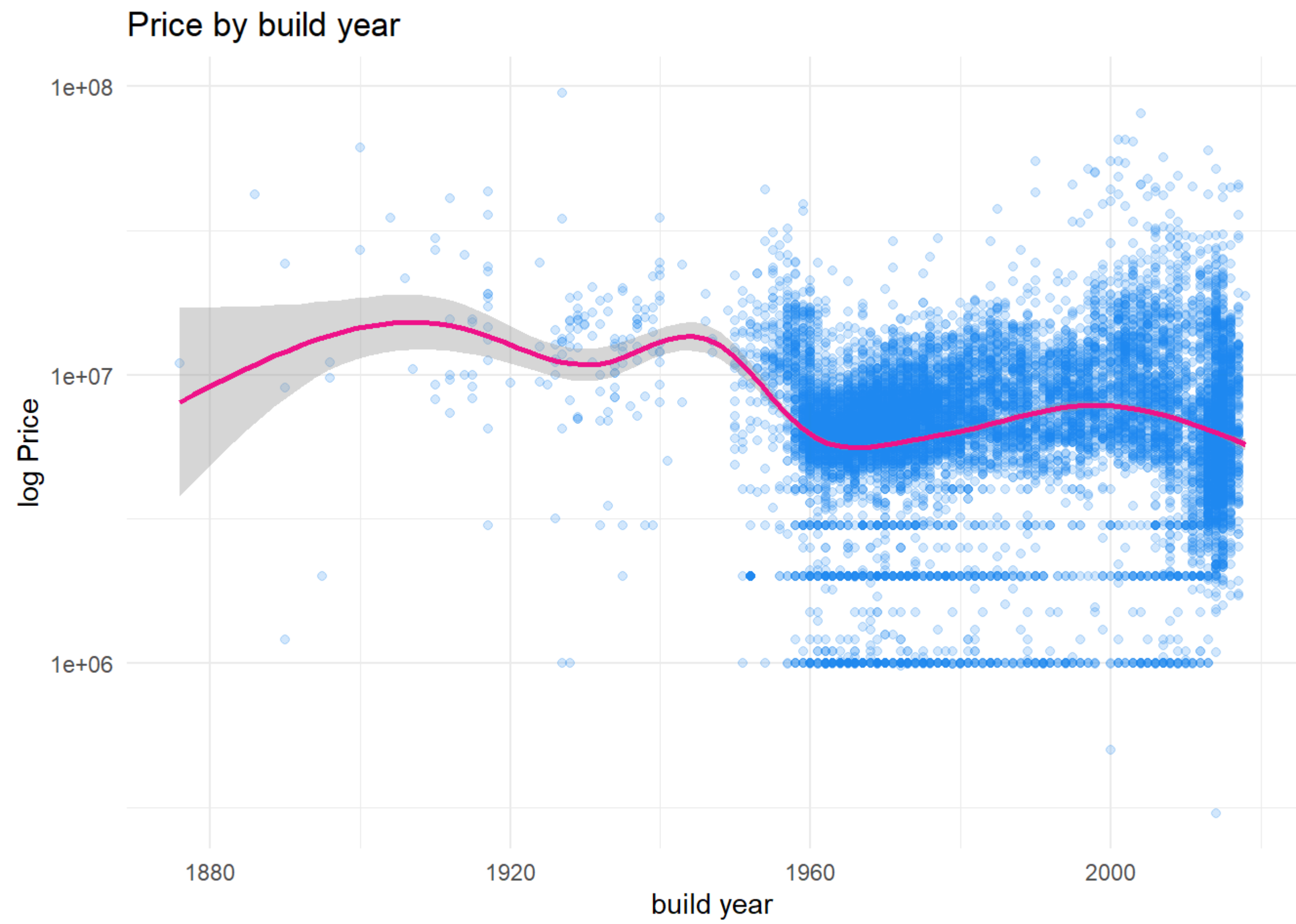
```
##
##        0        1        3       71      215     1876     1886     1890
##      406      270        1        1        1        1        1        3
##     1895     1896     1900     1904     1906     1907     1910     1912
##        1        2        2        1        1        1        4        5
##     1914     1915     1917     1920     1924     1925     1926     1927
##        2        5       13        1        3        1        6        8
##     1928     1929     1930     1931     1932     1933     1934     1935
##        9       10        3        4        6        5        9        8
##     1936     1937     1938     1939     1940     1941     1943     1946
##        2       10        4        6       12        1        2        2
##     1947     1949     1950     1951     1952     1953     1954     1955
##        3        1       17       18       33       18       29       44
##     1956     1957     1958     1959     1960     1961     1962     1963
##       38       85      135      166      262      226      268      230
##     1964     1965     1966     1967     1968     1969     1970     1971
##      239      280      256      289      276      312      313      259
##     1972     1973     1974     1975     1976     1977     1978     1979
##      276      254      270      229      204      200      164      168
##     1980     1981     1982     1983     1984     1985     1986     1987
##      174      143      142      131      130      134       98      122
##     1988     1989     1990     1991     1992     1993     1994     1995
##      127      124       89       75      101       88      117      118
##     1996     1997     1998     1999     2000     2001     2002     2003
##      117      111      111       94       95      132      160      147
##     2004     2005     2006     2007     2008     2009     2010     2011
##      156      134      171      155      179      135       94      123
##     2012     2013     2014     2015     2016     2017     2018     4965
##      180      351      709      608      276      123        1        1
## 20052009
##        1
```

In main dataset we set the build years before 1860 and after 2018 to NA

```
train$build_year[train$build_year<1860 |train$build_year> 2018 ] <- NA
```

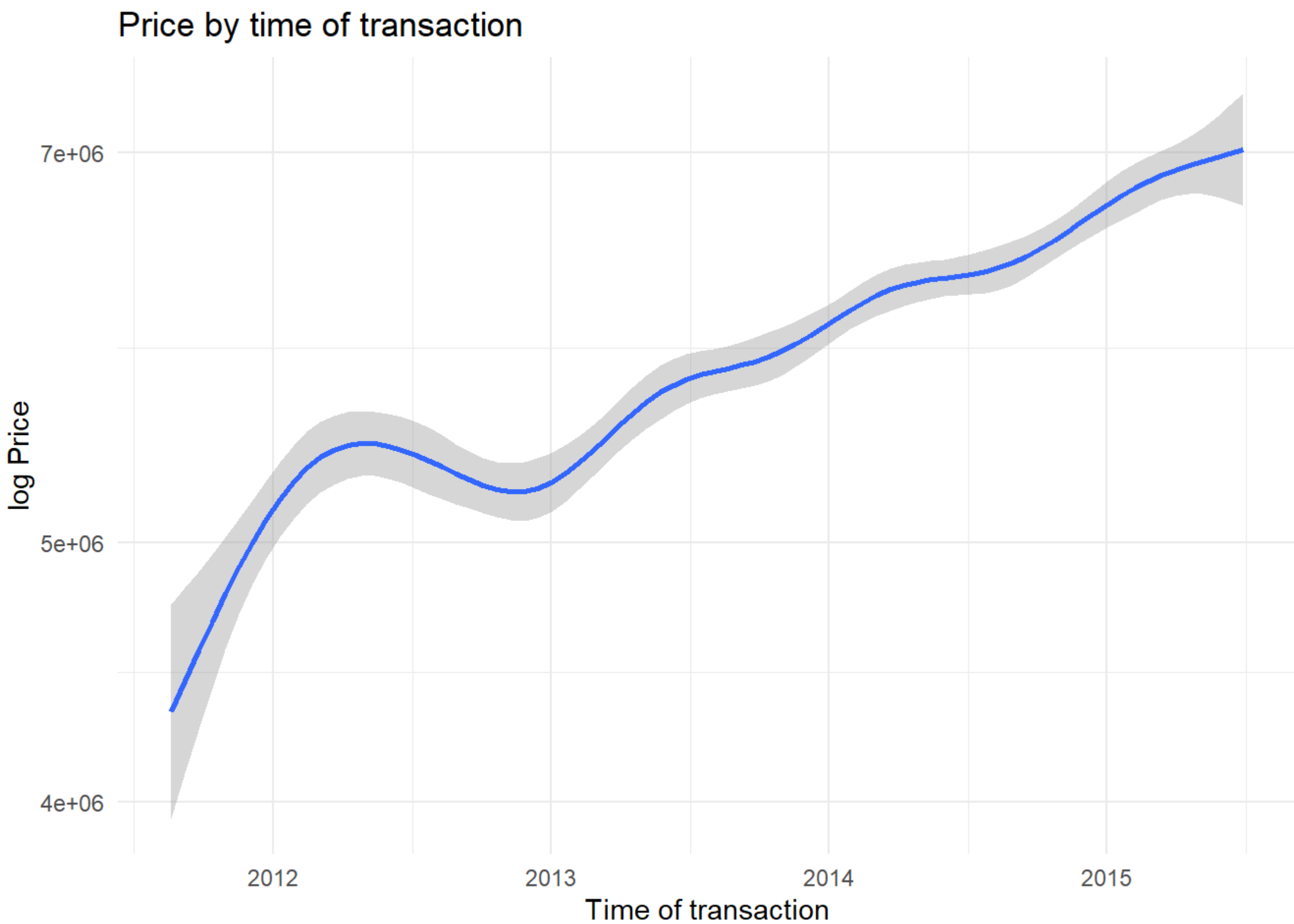
The plot of price against the built year is as follows. As it can been seen some properties values have been rounded (either by operator or sellers)

```
train %>%
    filter(build_year >1860) %>%
    ggplot(aes(y=price_doc, x=build_year)) +
    geom_point(color = 'dodgerblue2' ,alpha = .2)+
    geom_smooth(color = 'deeppink2') +
    scale_y_log10()+
    labs(x='build year', y='log Price', title='Price by build year')+
    theme_minimal()
```



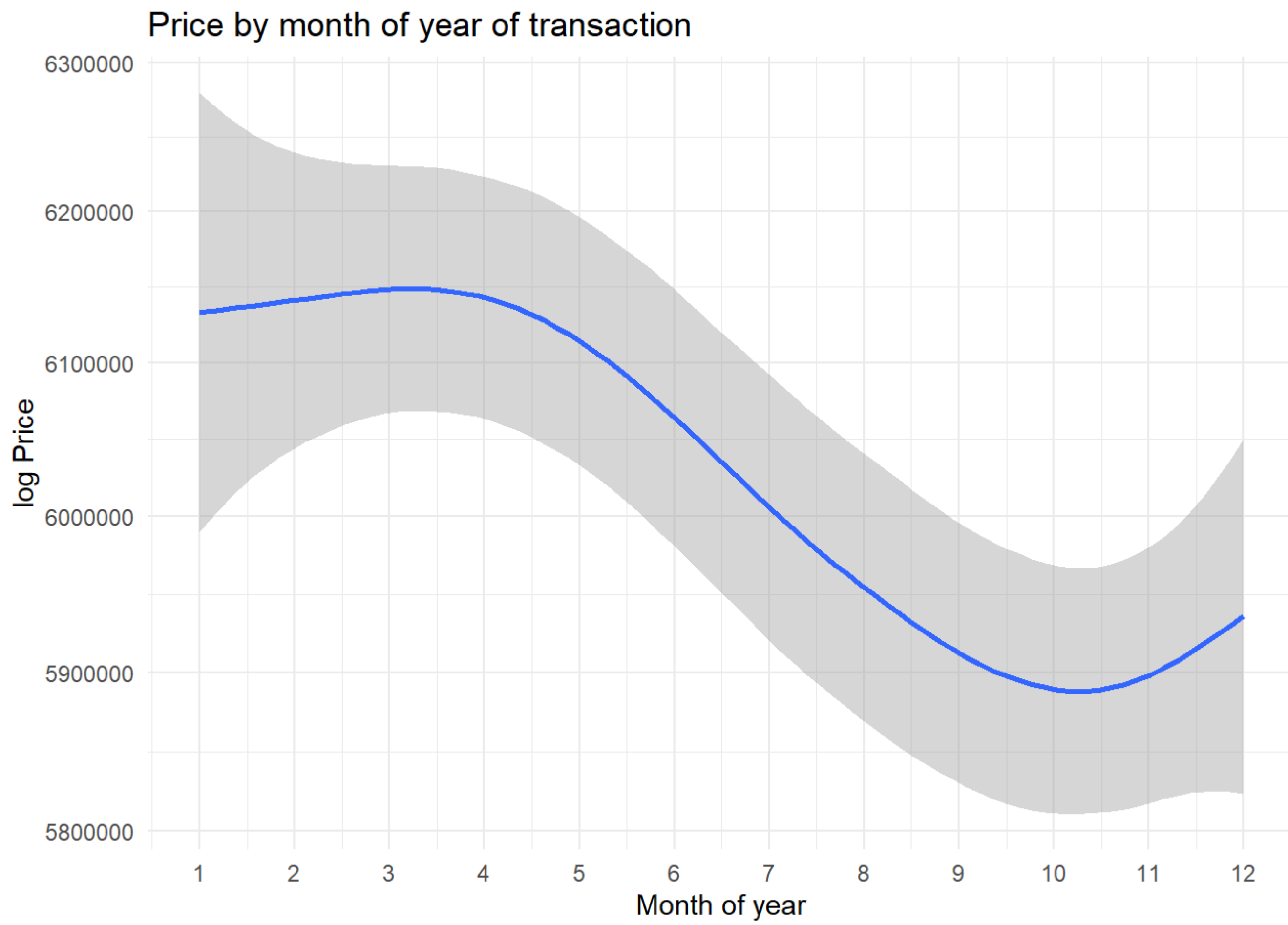
Here we check the price trend in our dataset and as we see the transaction value is continuously increasing.

```
train %>%
  ggplot(aes(y=price_doc , x= (timestamp) )) +
  geom_smooth()+
  scale_y_log10()+
  labs(x='Time of transaction', y='log Price', title='Price by time of transaction')+
    theme_minimal()
```



Now we check the scatter plot of price by month of transaction, to check seasonality. The transactions in spring are of a higher value compared to winter.

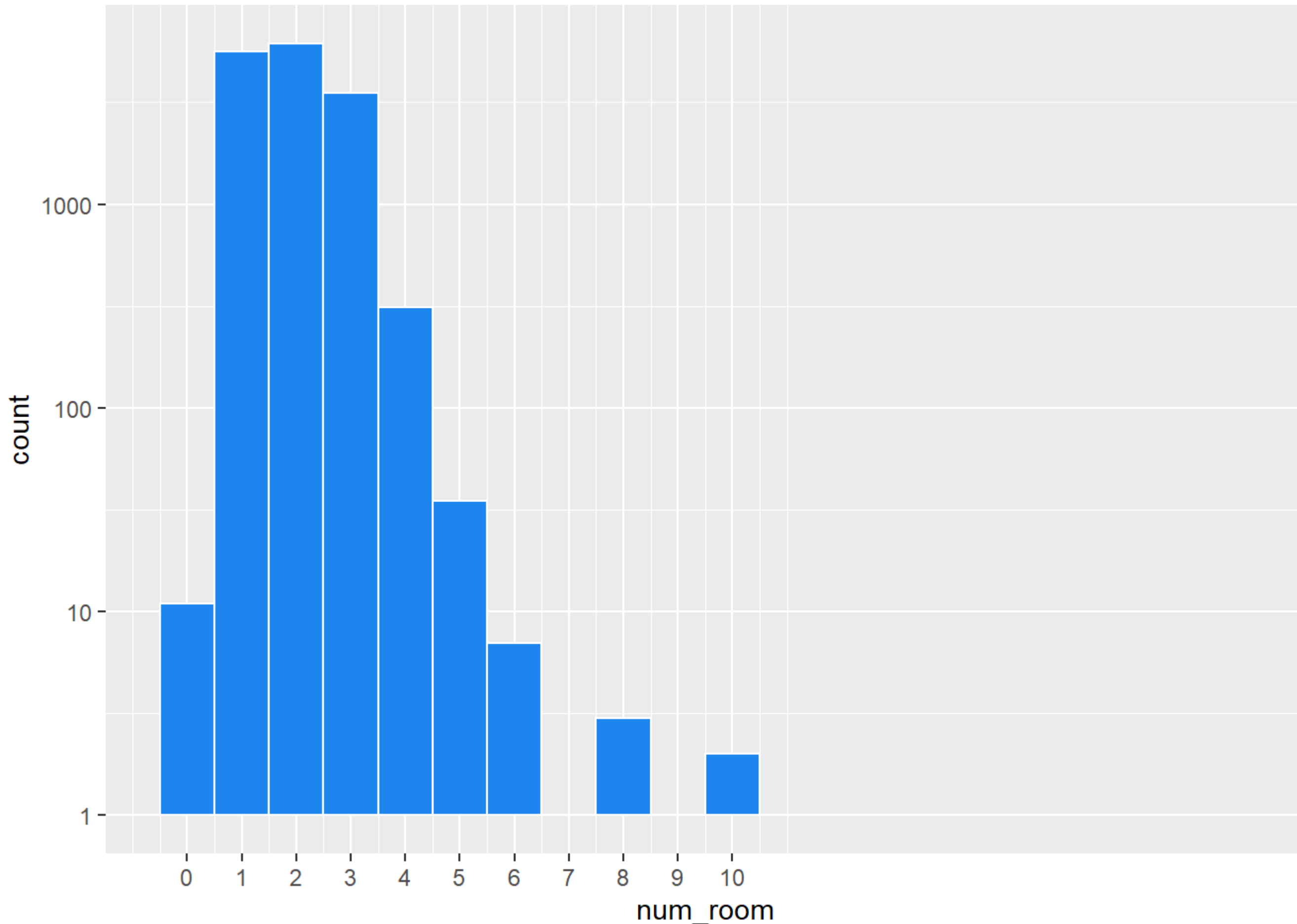
```
train %>%
  mutate(year = year(timestamp)) %>%
  ggplot(aes(y=price_doc , x= month(timestamp) , color = year)) +
  geom_smooth()+
  scale_y_log10()+
  scale_x_continuous(breaks = seq(1,12,1)) +
  labs(x='Month of year', y='log Price', title='Price by month of year of transaction')+
    theme_minimal()
```



4.1.7 num_room

We use a histogram to investigate the number of rooms.

```
train %>%
  ggplot(aes(x=num_room)) +
  geom_histogram(fill = "dodgerblue2", color = "white" ,bins=20) +
  scale_y_log10() +
  scale_x_continuous(breaks = seq(0,10,1))
```

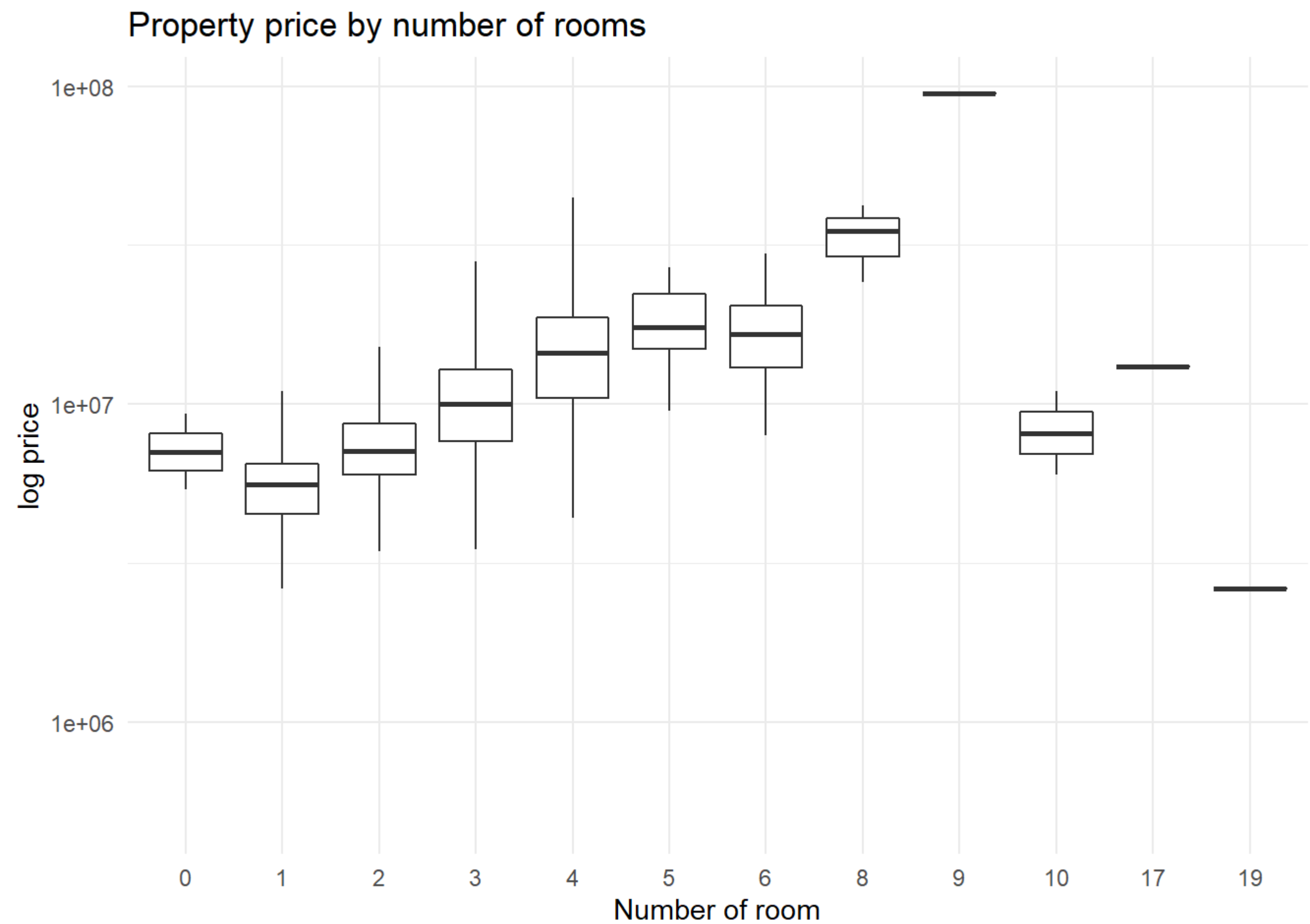



```
  labs(x='Number of Rooms', y='Count', title='number of room log scaled histogram distribution')
+
    theme_minimal()
```

```
## NULL
```

We check the property price by number of rooms, as expected there is a positive correlation.

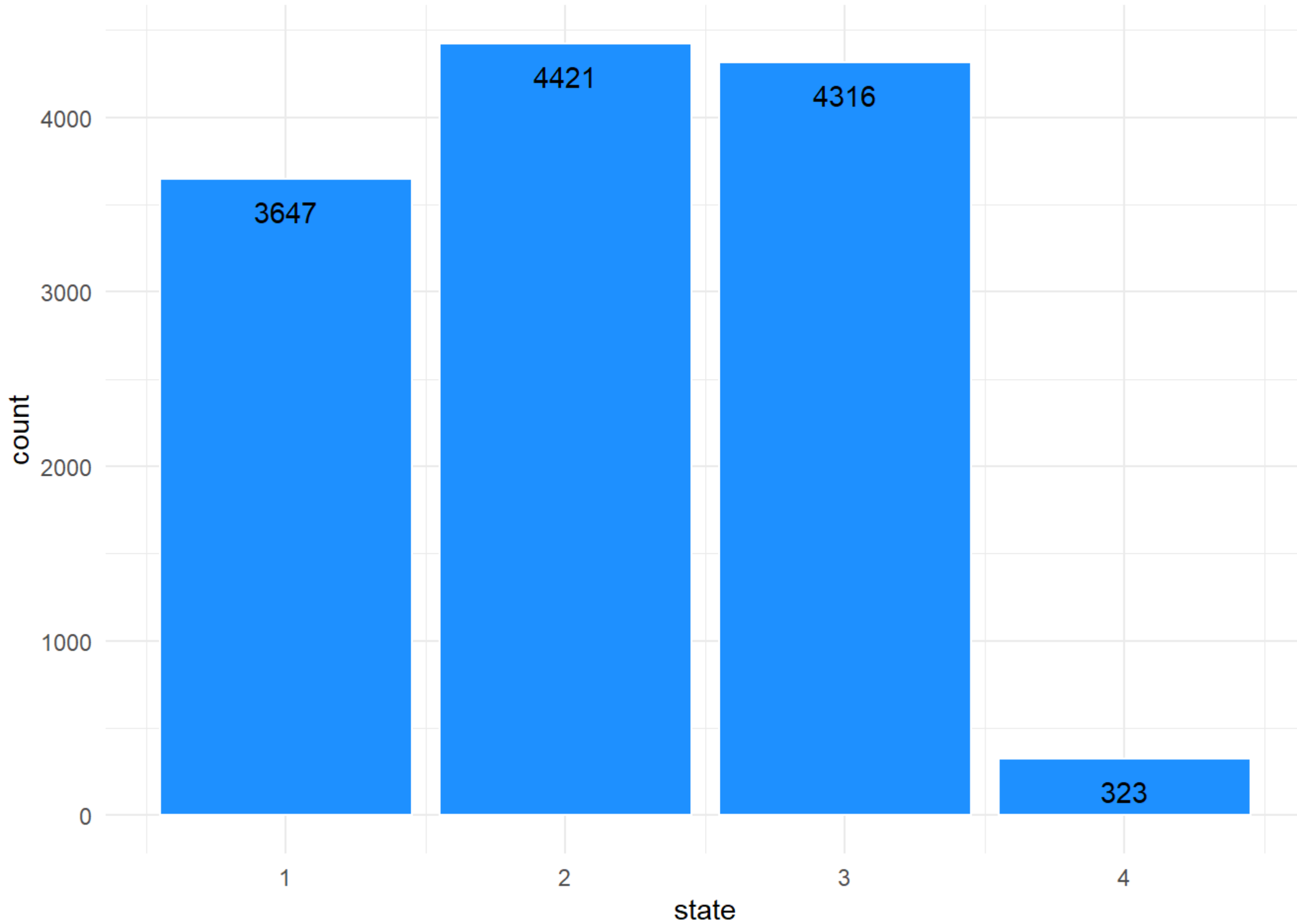
```
train %>%
  na.omit() %>%
  ggplot(aes(y=price_doc ,x=as.factor(num_room))) +
  geom_boxplot(outlier.shape = NA) +
  scale_y_log10()+
  labs(x='Number of room', y='log price', title='Property price by number of rooms')+
    theme_minimal()
```



4.1.8 state

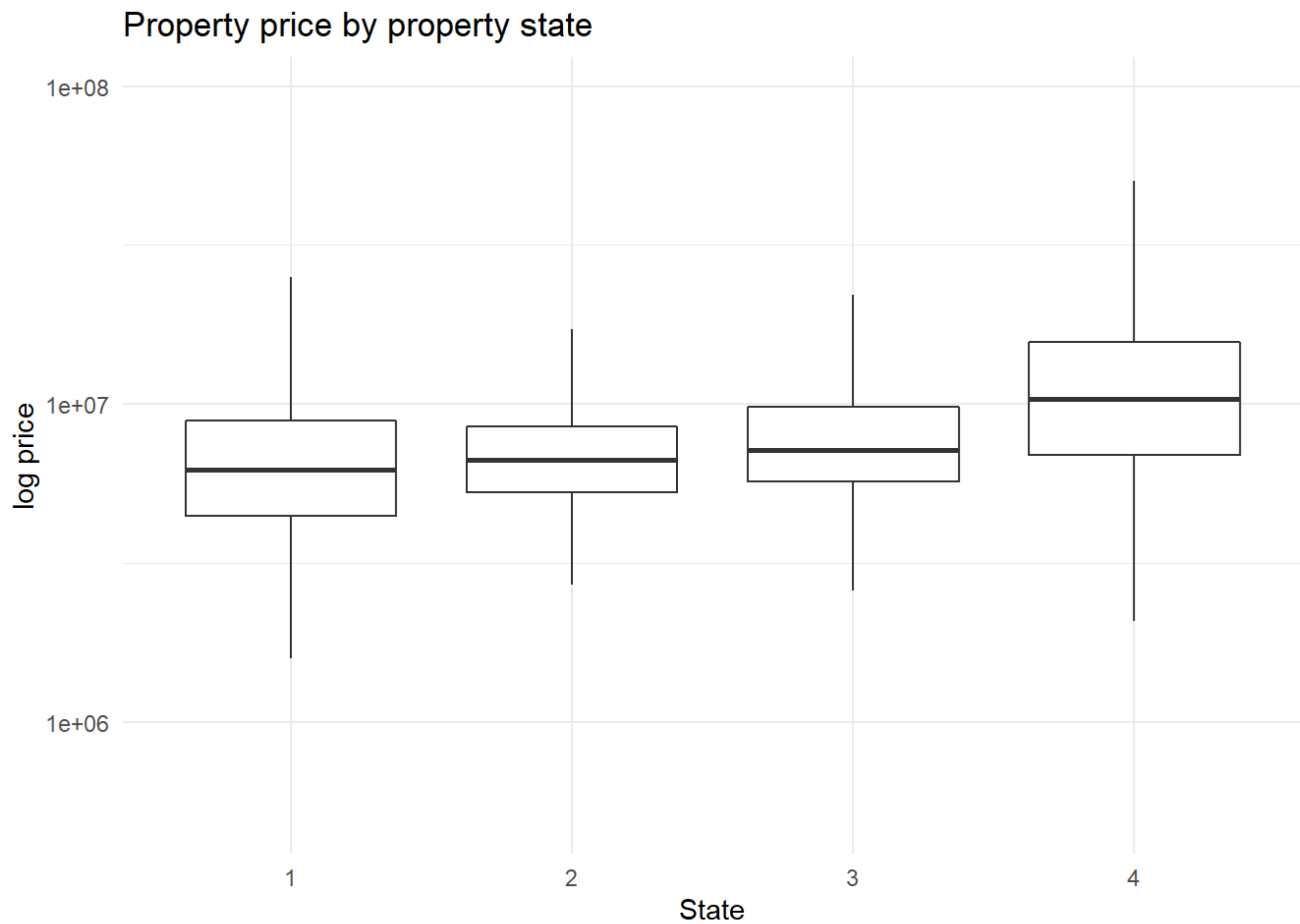
here we check the apartment condition, we also set it to factor as we don't know wheter it is orderd or not./ About hald the data contains unknown state.

```
train$state[train$state == 33] <- 3
train %>%
  ggplot( aes(x=state)) +
  geom_bar(fill = "dodgerblue1", color = "white") +
  geom_text(stat='count', aes(label=..count..), vjust=2)+
    theme_minimal()
```



We see a slight increase in the price by state.

```
train %>%
  na.omit() %>%
  ggplot(aes(y=price_doc ,x=as.factor(state))) +
  geom_boxplot(outlier.shape = NA) +
  scale_y_log10()+
  labs(x='State', y='log price', title='Property price by property state')+
    theme_minimal()
```



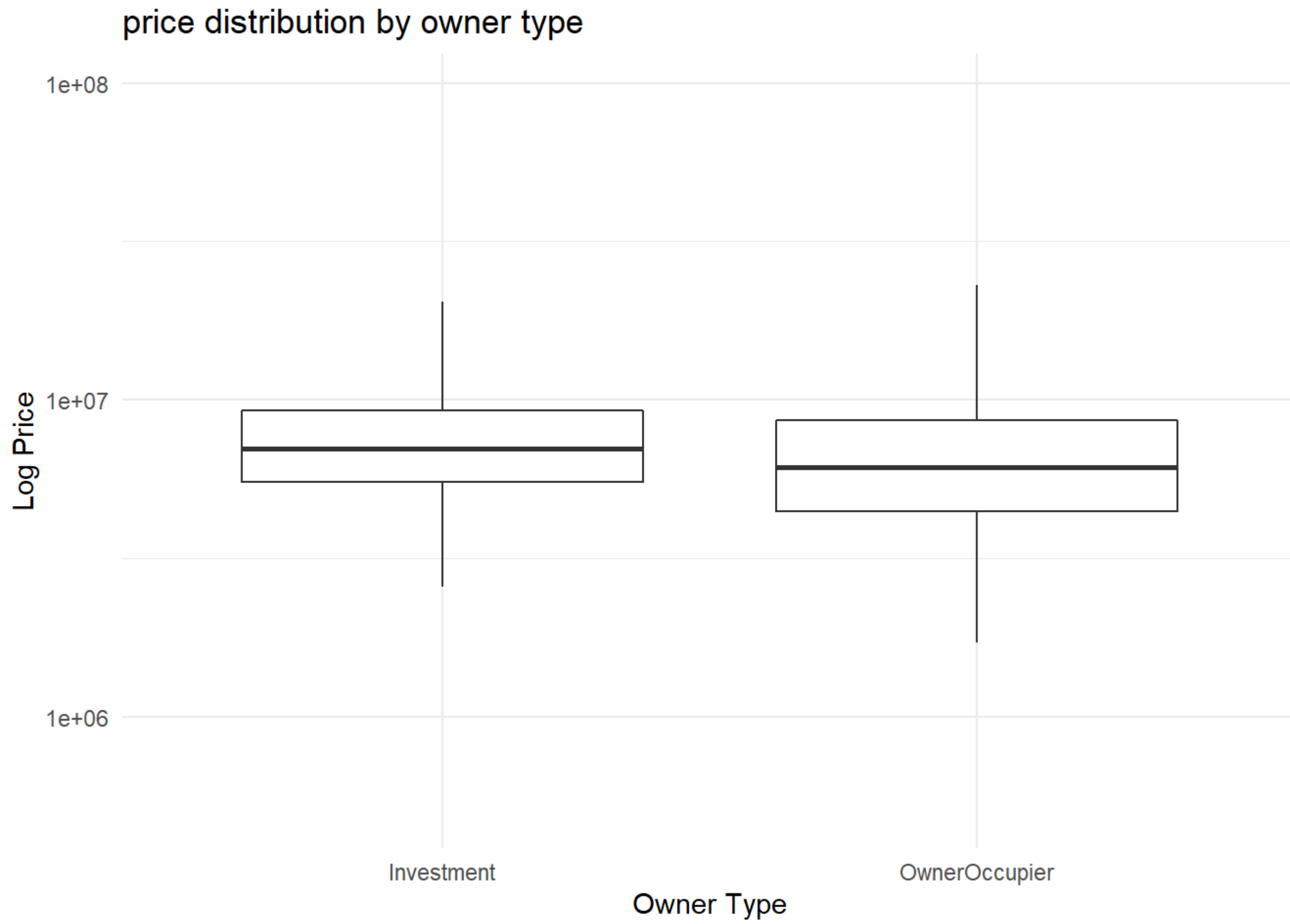
4.1.9 product_type

We investigate the property area against owner-occupier purchase or investment. Occupier are buying bigger houses which can be justified by the fact that they are getting both the utility of living in the property and also having it as a investment.

```
train %>%
  na.omit() %>%
  ggplot(aes(y=full_sq ,x=as.factor(product_type))) +
  geom_boxplot(outlier.shape = NA) +
  scale_y_log10()+
  labs(x='Owner Type', y='Property area', title='price distribution by owner type')+
    theme_minimal()
```

Here we have property value by owner against investor. Investors are buying bigger properties.

```
train %>%
  na.omit() %>%
  ggplot(aes(y=price_doc ,x=as.factor(product_type))) +
  geom_boxplot(outlier.shape = NA) +
  scale_y_log10()+
  labs(x='Owner Type', y='Log Price', title='price distribution by owner type')+
    theme_minimal()
```



4.2 Macro data

Among the columns of the Macro data, we have picked the most interesting ones.

4.2.1 usdrub

The graph is a proxy measurement of the Russia's economy. Inverting the Rubl to dollar conversion rate will give a better result, as we want to see how the value of Rubl is changing by time.

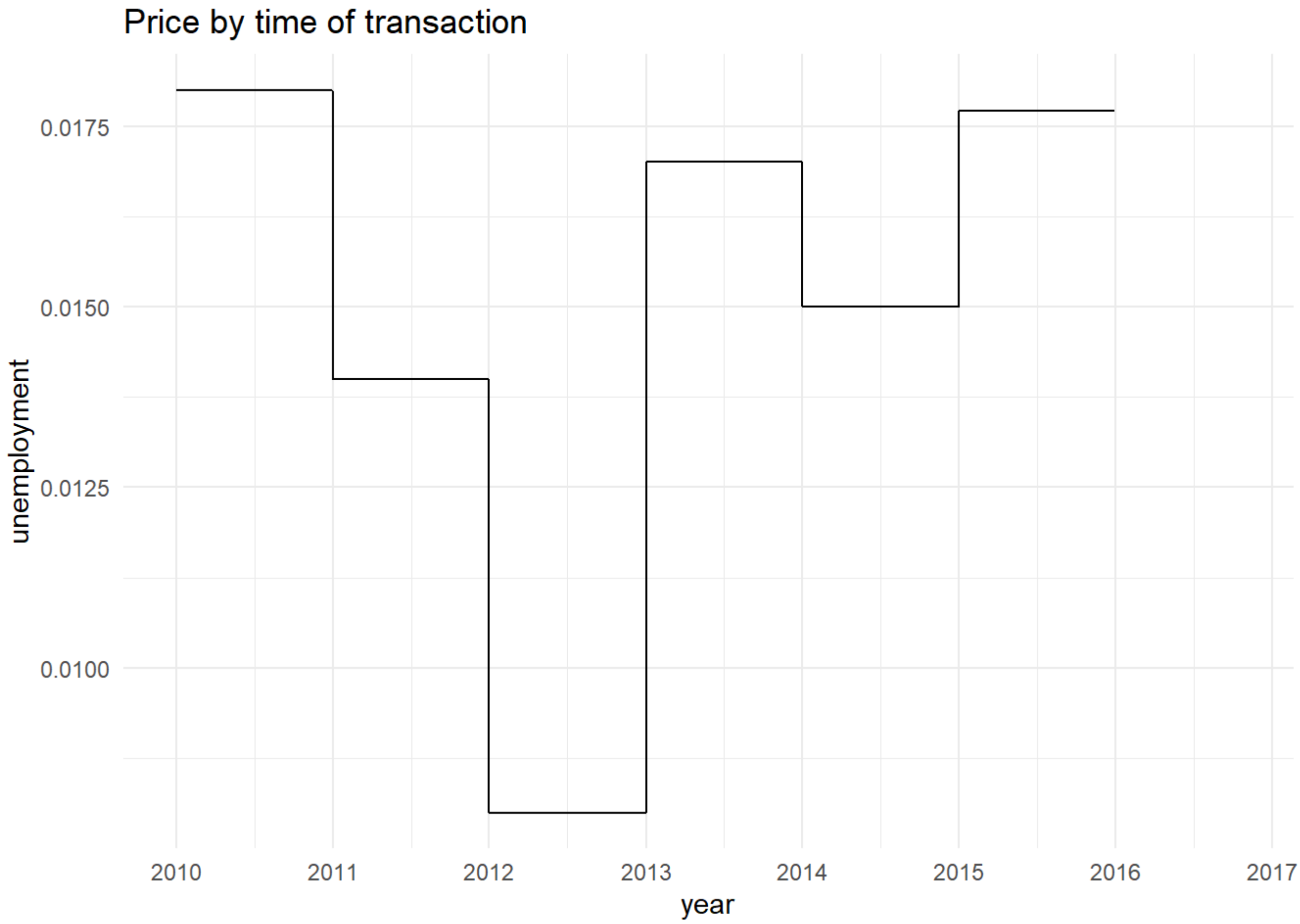
```
don <- xts(x = (1/ macro$usdrub), order.by = macro$timestamp)

dygraph(don) %>%
  dyOptions(labelsUTC = TRUE, fillGraph=TRUE, fillAlpha=0.1, drawGrid = TRUE, colors="dodgerblue
2") %>%
  dyRangeSelector() %>%
  dyCrosshair(direction = "vertical") %>%
  dyHighlight(highlightCircleSize = 5, highlightSeriesBackgroundAlpha = 0.2, hideOnMouseOut = FA
LSE)  %>%
  dyRoller(rollPeriod = 1)
```

4.2.2 unemployment

Unemplyment is another important factor

```
macro %>%
  ggplot(aes(y=unemployment , x= (timestamp) )) +
  geom_line()+
  scale_x_date(date_breaks = "years" , date_labels = "%Y") +
  labs(x='year', y='unemployment', title='Price by time of transaction')+
    theme_minimal()
```



4.2.2.1 unbalanced data and sample selection

Heckman sample selection bias and unbalanced pannel data Now we left-merge the main dataset with the macro data.

now we have to clean the Test data, with the rules used on the train datasets.

```
test[,"full_sq"][test[,"full_sq"] == 0] <- NA
test[,"life_sq"][test[,"life_sq"]>test[,"full_sq"]] <- NA
test[,"kitch_sq"][test$kitch_sq>test$full_sq] <- NA
test$max_floor[test$max_floor<test$floor] <- NA
test$build_year[test$build_year<1860 |test$build_year> 2018 ] <- NA
test$state[test$state == 33] <- 3
```

5 Data type

We transform character vectors to factor.

```
# First we convert the train dataset characters to factor
train[sapply(train, is.character)] <- lapply(train[sapply(train, is.character)], as.factor)

# now we have to do the same for the Test, however using the factors that has been used in train
only
test$product_type  <- factor(test$product_type, levels = levels(train$product_type))
sapply(train,class)
```

```
##    timestamp      full_sq      life_sq        floor    max_floor   build_year
##       "Date"    "integer"    "integer"    "integer"    "integer"    "integer"
##     num_room     kitch_sq        state     material product_type     full_all
##    "integer"    "integer"    "numeric"    "integer"     "factor"    "integer"
##    price_doc       usdrub unemployment
##    "integer"    "numeric"    "numeric"
```

```
sapply(test,class)
```

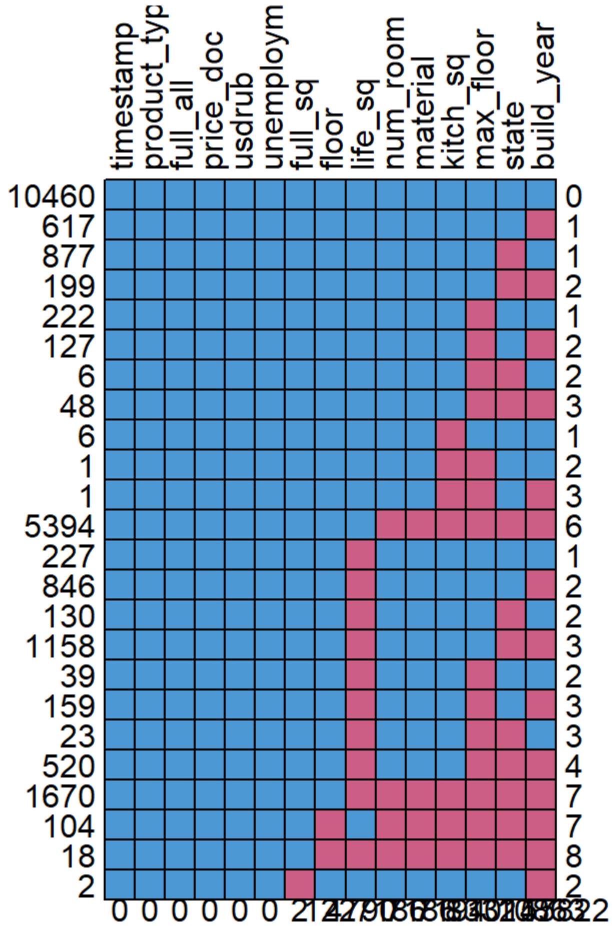
```
##    timestamp      full_sq      life_sq        floor    max_floor   build_year
##       "Date"    "integer"    "integer"    "integer"    "integer"    "integer"
##     num_room     kitch_sq        state     material product_type     full_all
##    "integer"    "integer"    "numeric"    "integer"     "factor"    "integer"
##    price_doc       usdrub unemployment
##    "integer"    "numeric"    "numeric"
```

6 Imputing the missing data

The followings are several useful links that have been used for this project. This is a book on imputation by the developer of the package mice https://stefvanbuuren.name/fimd/ch-introduction.html (https://stefvanbuuren.name/fimd/ch-introduction.html) The following is a tutorial which explains how to implement the discussed ideas. https://amices.org/Winnipeg/ (https://amices.org/Winnipeg/) The following is a series of vignettes that covers the mice packages impelementation. https://www.gerkovink.com/miceVignettes/ (https://www.gerkovink.com/miceVignettes/)

Here we check the pattern of missing data, as we can see we have a case of multivariate missing values. In the graph, on the left we have the frequency of each pattern and on the right side the number of missing values.

```
md.pattern(train, rotate.names = TRUE)
```


timestamp
product_typ
full_all
price_doc
usdrub
unemploym
full_sq
floor
life_sq
num_room
material
kitch_sq
max_floor
state
build_year
10460
617
877
199
222
127
6
48
6
1
1
5394
227
846
130
1158
39
159
23
520
1670
104
18
2

```
##       timestamp product_type full_all price_doc usdrub unemployment full_sq
## 10460         1            1        1         1      1            1       1
## 617           1            1        1         1      1            1       1
## 877           1            1        1         1      1            1       1
## 199           1            1        1         1      1            1       1
## 222           1            1        1         1      1            1       1
## 127           1            1        1         1      1            1       1
## 6             1            1        1         1      1            1       1
## 48            1            1        1         1      1            1       1
## 6             1            1        1         1      1            1       1
## 1             1            1        1         1      1            1       1
## 1             1            1        1         1      1            1       1
## 5394          1            1        1         1      1            1       1
## 227           1            1        1         1      1            1       1
## 846           1            1        1         1      1            1       1
## 130           1            1        1         1      1            1       1
## 1158          1            1        1         1      1            1       1
## 39            1            1        1         1      1            1       1
## 159           1            1        1         1      1            1       1
## 23            1            1        1         1      1            1       1
## 520           1            1        1         1      1            1       1
## 1670          1            1        1         1      1            1       1
## 104           1            1        1         1      1            1       1
## 18            1            1        1         1      1            1       1
## 2             1            1        1         1      1            1       0
##               0            0        0         0      0            0       2
##       floor life_sq num_room material kitch_sq max_floor state build_year
## 10460     1       1        1        1        1         1     1          1     0
## 617       1       1        1        1        1         1     1          0     1
## 877       1       1        1        1        1         1     0          1     1
## 199       1       1        1        1        1         1     0          0     2
## 222       1       1        1        1        1         0     1          1     1
## 127       1       1        1        1        1         0     1          0     2
## 6         1       1        1        1        1         0     0          1     2
## 48        1       1        1        1        1         0     0          0     3
## 6         1       1        1        1        0         1     1          1     1
## 1         1       1        1        1        0         0     1          1     2
## 1         1       1        1        1        0         0     1          0     3
## 5394      1       1        0        0        0         0     0          0     6
## 227       1       0        1        1        1         1     1          1     1
## 846       1       0        1        1        1         1     1          0     2
## 130       1       0        1        1        1         1     0          1     2
## 1158      1       0        1        1        1         1     0          0     3
## 39        1       0        1        1        1         0     1          1     2
## 159       1       0        1        1        1         0     1          0     3
## 23        1       0        1        1        1         0     0          1     3
## 520       1       0        1        1        1         0     0          0     4
## 1670      1       0        0        0        0         0     0          0     7
## 104       0       1        0        0        0         0     0          0     7
## 18        0       0        0        0        0         0     0          0     8
## 2         1       1        1        1        1         1     1          0     2
##         122    4790     7186     7186     7194      8332 10147      10863 55822
```

Now we start the imputing the missing variables using “Multivariate Imputation by Chained Equations”.

First we set the prediction matrix.

```
pred <- imp$predictorMatrix
```

We also have to consider that the column subarea and area population have perfect correlation and we should use only one of them in our analysis. We also skip the column timestamp as it is not a numerical variable. We also won’t use the column price_doc as it is our target variable and we should not leak information.

```
pred[ ,"timestamp"] <- 0
pred[ ,"full_all"] <- 0
pred[ ,"price_doc"] <- 0
pred
```

```
##              timestamp full_sq life_sq floor max_floor build_year num_room
## timestamp            0       1       1     1         1          1        1
## full_sq              0       0       1     1         1          1        1
## life_sq              0       1       0     1         1          1        1
## floor                0       1       1     0         1          1        1
## max_floor            0       1       1     1         0          1        1
## build_year           0       1       1     1         1          0        1
## num_room             0       1       1     1         1          1        0
## kitch_sq             0       1       1     1         1          1        1
## state                0       1       1     1         1          1        1
## material             0       1       1     1         1          1        1
## product_type         0       1       1     1         1          1        1
## full_all             0       1       1     1         1          1        1
## price_doc            0       1       1     1         1          1        1
## usdrub               0       1       1     1         1          1        1
## unemployment         0       1       1     1         1          1        1
##              kitch_sq state material product_type full_all price_doc usdrub
## timestamp           1     1        1            1        0         0      1
## full_sq             1     1        1            1        0         0      1
## life_sq             1     1        1            1        0         0      1
## floor               1     1        1            1        0         0      1
## max_floor           1     1        1            1        0         0      1
## build_year          1     1        1            1        0         0      1
## num_room            1     1        1            1        0         0      1
## kitch_sq            0     1        1            1        0         0      1
## state               1     0        1            1        0         0      1
## material            1     1        0            1        0         0      1
## product_type        1     1        1            0        0         0      1
## full_all            1     1        1            1        0         0      1
## price_doc           1     1        1            1        0         0      1
## usdrub              1     1        1            1        0         0      0
## unemployment        1     1        1            1        0         0      1
##              unemployment
## timestamp               1
## full_sq                 1
## life_sq                 1
## floor                   1
## max_floor               1
## build_year              1
## num_room                1
## kitch_sq                1
## state                   1
## material                1
## product_type            1
## full_all                1
## price_doc               1
## usdrub                  1
## unemployment            0
```

Now we have to set the statistical method that we want to be used for prediction of each column. The mice package makes the best choices as predictive mean matching, logistic and polynomial based on data and we have to change that for variables that we think it is necessary. The columns that do not have a missing variable do not have a method.

```
meth <- imp$meth
meth
```

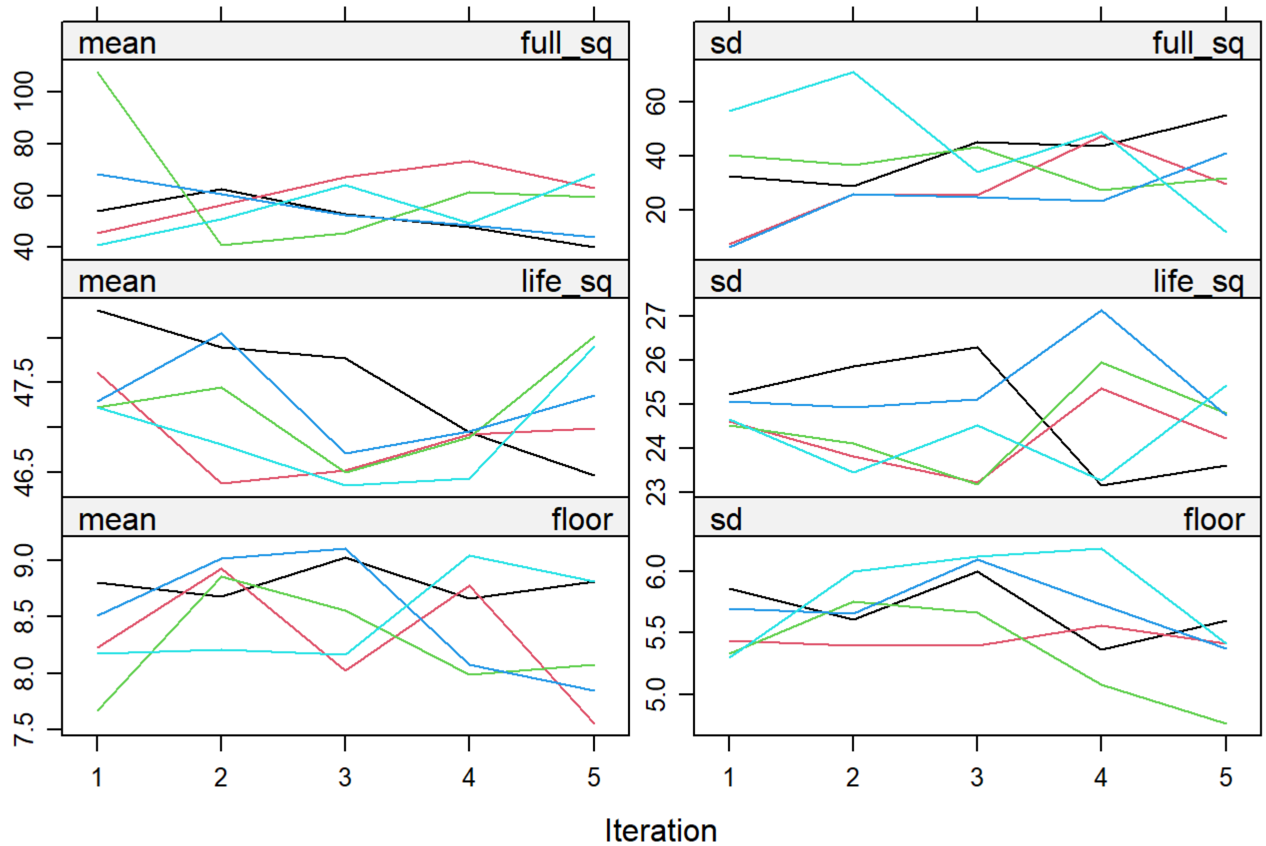
```
##    timestamp      full_sq      life_sq        floor    max_floor   build_year
##           ""        "pmm"        "pmm"        "pmm"        "pmm"        "pmm"
##     num_room     kitch_sq        state     material product_type     full_all
##        "pmm"        "pmm"        "pmm"        "pmm"           ""           ""
##    price_doc       usdrub unemployment
##           ""           ""           ""
```

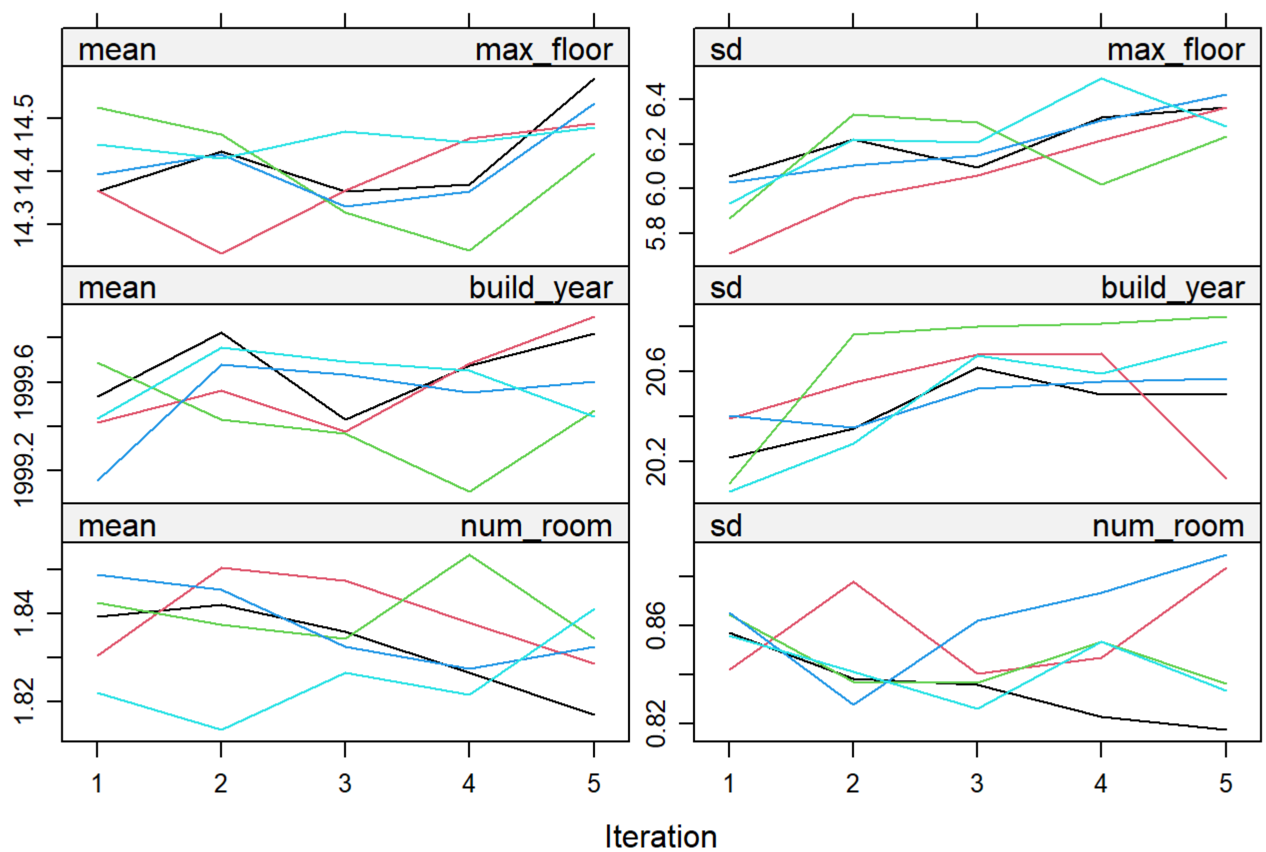
Now we can run the algorithm

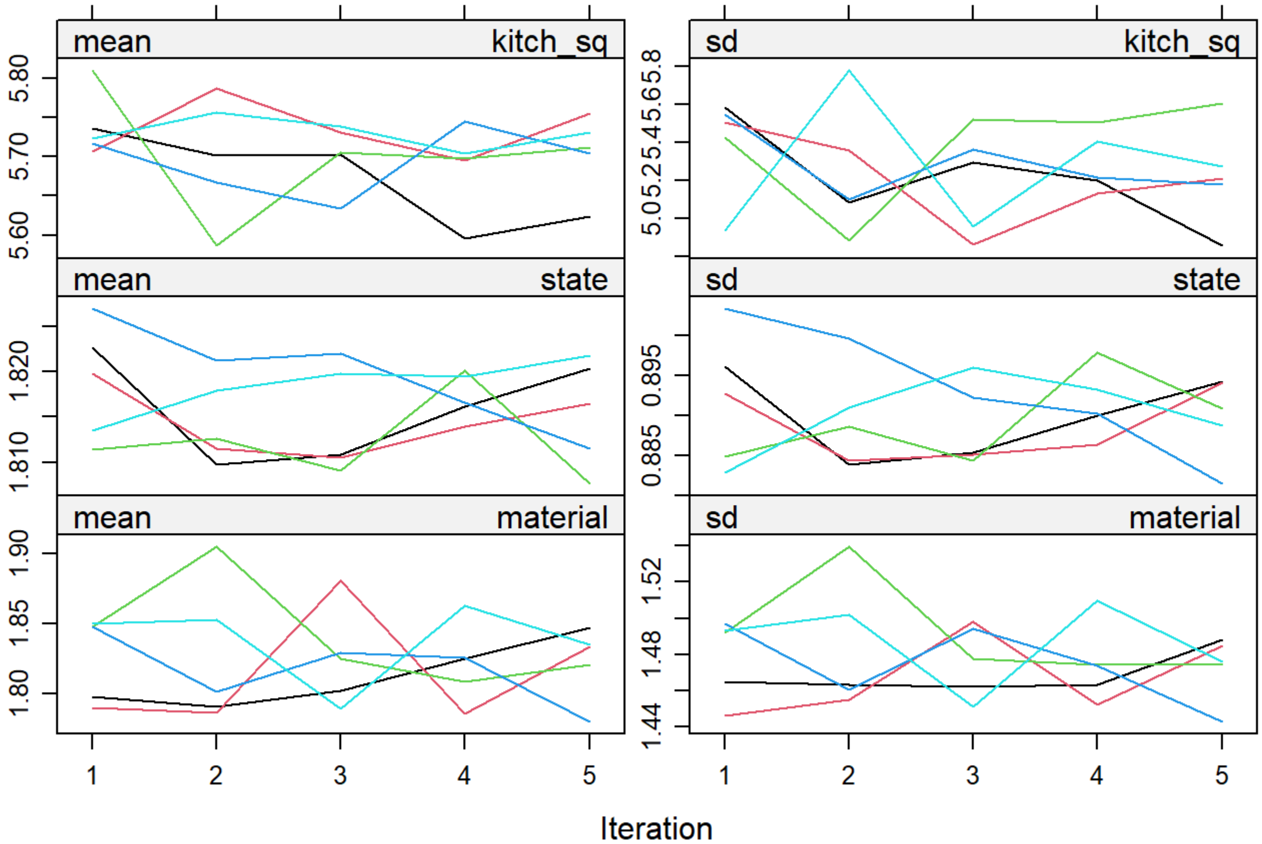
```
imp <- mice(train, meth = meth, pred = pred, maxit = 5 , seed = 1234 , print = FALSE)
```

We check whether there is a trend in imputation, and the data seems fine.

```
plot(imp)
```


mean
full_sq
sd
full_sq
mean
life_sq
sd
life_sq
mean
floor
sd
floor
Iteration


mean
max_floor
sd
max_floor
mean
build_year
sd
build_year
mean
num_room
sd
num_room
Iteration



We make a long dataframe, stacking iterations of imputation over each other, since we are using the data for prediction, it is fine to do so.

```
train_stack <- complete(imp, "long")
dim(train_stack)
```

```
## [1] 114270     17
```

Now we need to impute the test data.

```
imp1 <- mice(test, maxit=0)
```

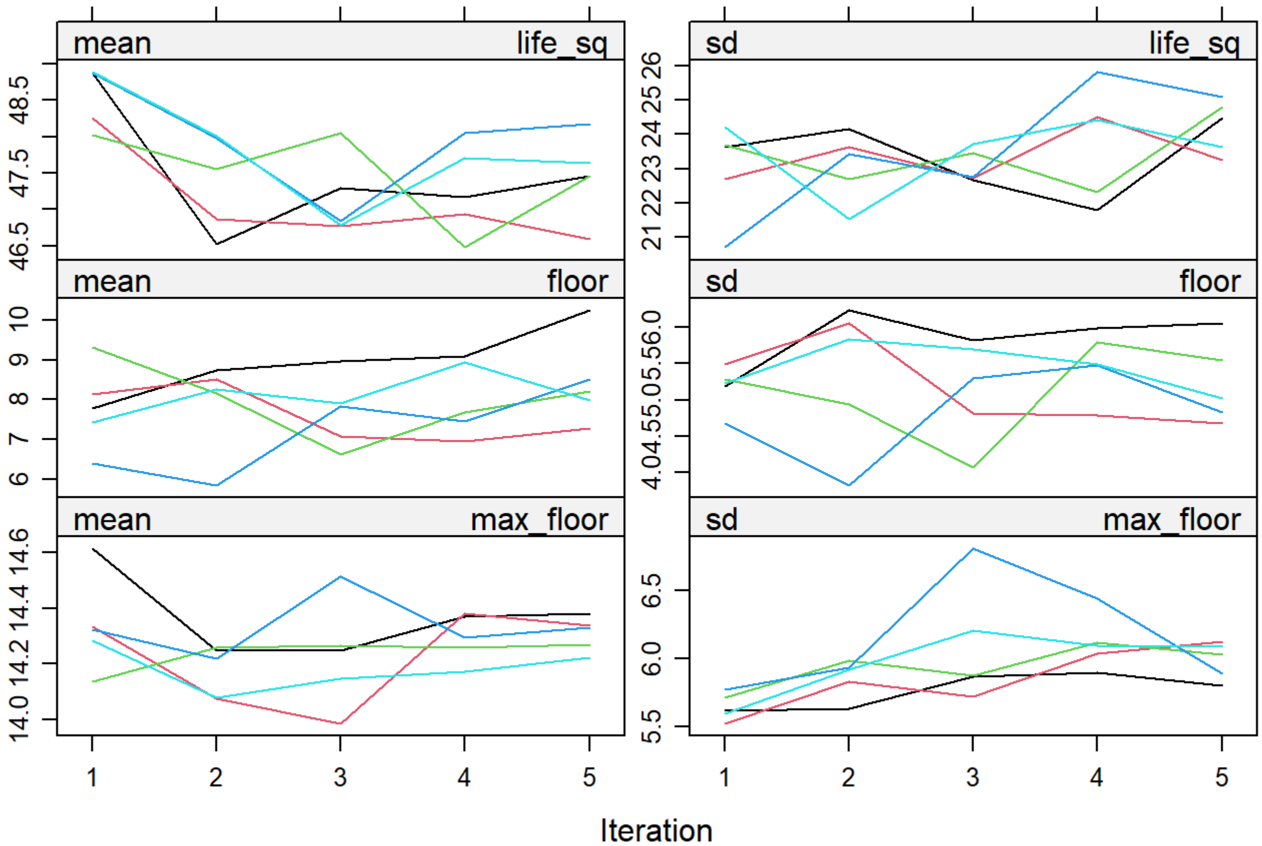
```
pred1 <- imp1$predictorMatrix
```

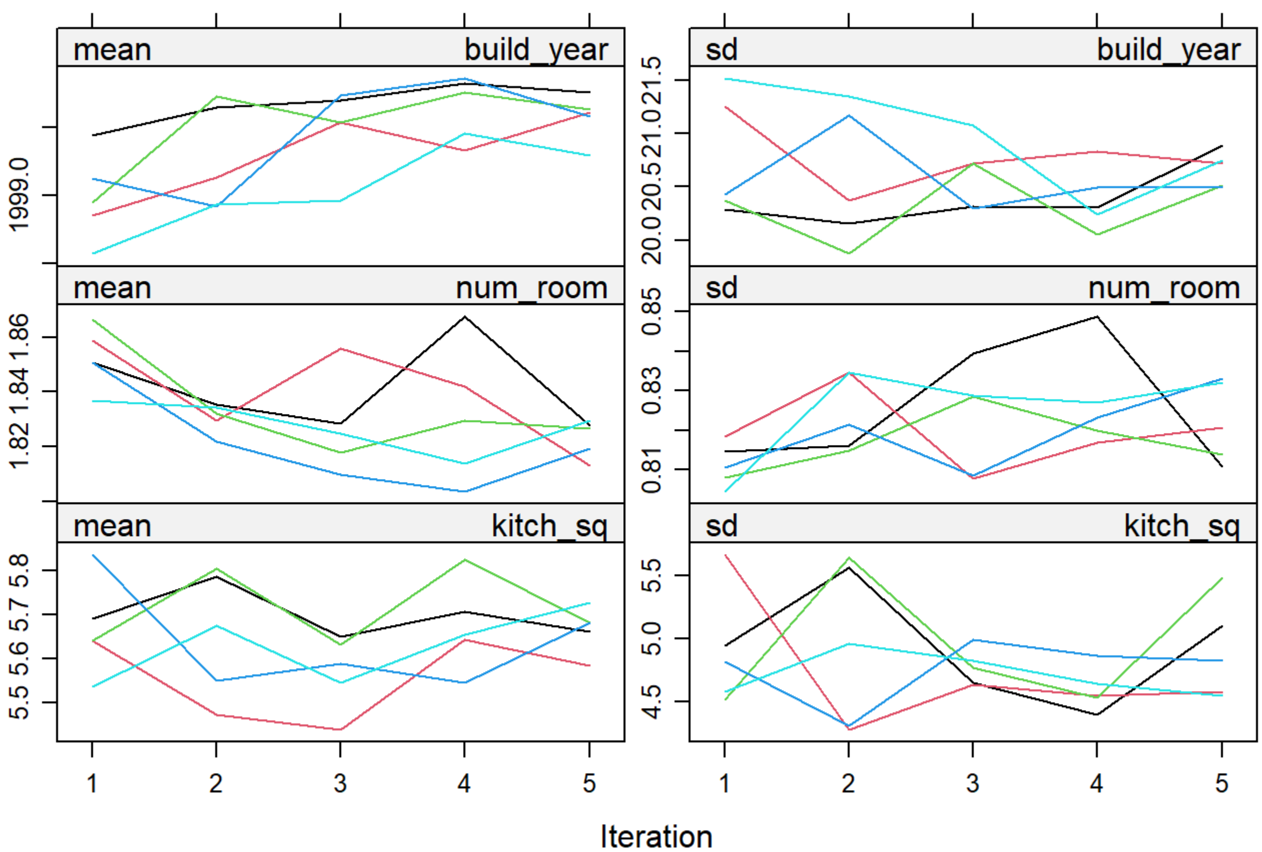
```
pred1[ ,"timestamp"] <- 0
pred1[ ,"full_all"] <- 0
pred1[ ,"price_doc"] <- 0
```

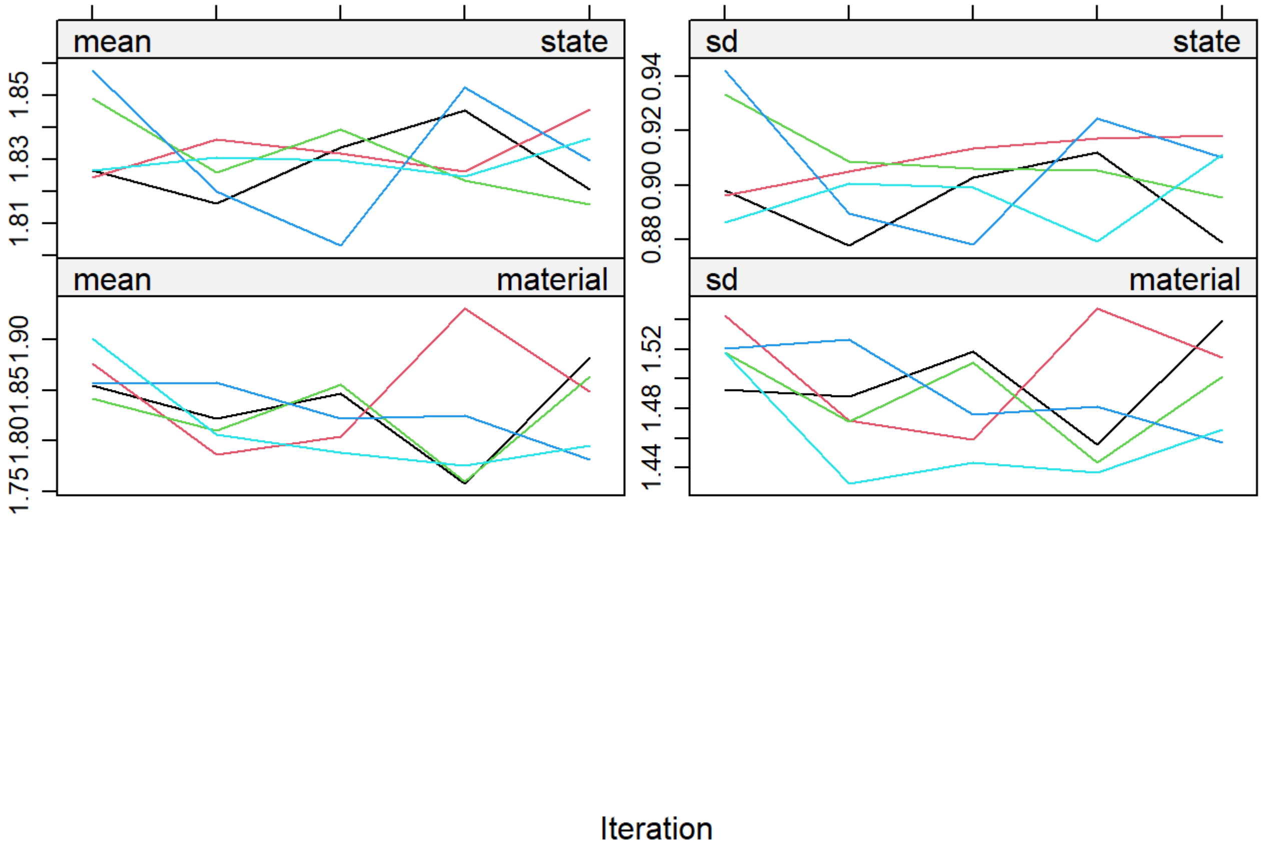
```
meth1 <- imp1$meth
```

```
imp1 <- mice(test, meth = meth1, pred = pred1, maxit = 5 , seed = 1234 , print = FALSE)
```

```
plot(imp1)
```


mean life_sq
sd life_sq
mean floor
sd floor
mean max_floor
sd max_floor
Iteration


mean build_year
sd build_year
mean num_room
sd num_room
mean kitch_sq
sd kitch_sq
Iteration




```
test_stack <- complete(imp1, "long")
dim(test_stack)
```

```
## [1] 38085    17
```

7 Model Fit

For modeling we use XGBoost regressor. It is fast, and has been shown to outperform most competitors.

But first lets do a simple regression.

```
regression <- lm(price_doc ~ . , data = train_stack)
regression_pred <- predict(regression, newdata = test_stack)
reg_r2 <- sum((regression_pred - test_stack$price_doc)^2)/nrow(test_stack)
reg_r2
```

```
## [1] 1.418832e+13
```

```
train_df <- data.table(train_stack[,4:17])
test_df  <- data.table(test_stack[,4:17])
train_df$product_type <- as.numeric(train_df$product_type)
test_df$product_type <- as.numeric(test_df$product_type)
```

Setting the validation dataset for XGBoost.

```
train_id <- sample(1:nrow(train_df), size = floor(0.8 * nrow(train)), replace=FALSE)
# Split in training and validation (80/20)
training <- train_df[train_id,]
validation <- train_df[-train_id,]
```

One hot encoding and setting the target variable

```
new_tr <- model.matrix(~.+0,data = training[,-c("price_doc"),with=F])
new_val<- model.matrix(~.+0,data = validation[,-c("price_doc"),with=F])
new_ts <- model.matrix(~.+0,data = test_df[,-c("price_doc"),with=F])
train_traget <- training$price_doc
val_traget <- validation$price_doc
test_target <- test_df$price_doc
```

preparing XGBoost matrix.

```
dtrain <- xgb.DMatrix(data = new_tr,label = train_traget)
dval   <- xgb.DMatrix(data = new_val,label = val_traget)
dtest  <- xgb.DMatrix(data = new_ts,label = test_target)
```

Setting default default parameters for the first run.

```
params <- list(booster = "gbtree", objective = "reg:squarederror",
               eta=0.3, gamma=0, max_depth=6, min_child_weight=1,
               subsample=1, colsample_bytree=1)
```

Running the first run

```
set.seed(1234)
xgb_base <- xgb.train (params = params,
                       data = dtrain,
                       nrounds =1000,
                       print_every_n = 200,
                       eval_metric = 'rmse',
                       early_stopping_rounds = 50,
                       watchlist = list(train= dtrain, val= dval))
```

Now we run a random parameter search with 1000 iteration

```
# strt time
start.time <- Sys.time()

# empty lists
lowest_error_list = list()
parameters_list = list()

# 1000 rows with random hyperparameters
set.seed(1234)
for (iter in 1:1000){
  param <- list(booster = "gbtree",
                objective = "reg:squarederror",
                max_depth = sample(3:10, 1),
                eta = runif(1, .01, .3),
                subsample = runif(1, .7, 1),
                colsample_bytree = runif(1, .6, 1),
                min_child_weight = sample(0:10, 1)
  )
  parameters <- as.data.frame(param)
  parameters_list[[iter]] <- parameters
}

# object that contains all randomly created hyperparameters
parameters_df = do.call(rbind, parameters_list)

# using randomly created parameters to create 1000 XGBoost-models
for (row in 1:nrow(parameters_df)){
  set.seed(20)
  mdcv <- xgb.train(data=dtrain,
                    booster = "gbtree",
                    objective = "reg:squarederror",
                    max_depth = parameters_df$max_depth[row],
                    eta = parameters_df$eta[row],
                    subsample = parameters_df$subsample[row],
                    colsample_bytree = parameters_df$colsample_bytree[row],
                    min_child_weight = parameters_df$min_child_weight[row],
                    nrounds= 300,
                    eval_metric = "rmse",
                    early_stopping_rounds= 30,
                    watchlist = list(train= dtrain, val= dval)
  )
  lowest_error <- as.data.frame(1 - min(mdcv$evaluation_log$val_error))
  lowest_error_list[[row]] <- lowest_error
}

# object that contains all accuracy's
lowest_error_df = do.call(rbind, lowest_error_list)

# binding columns of accuracy values and random hyperparameter values
randomsearch = cbind(lowest_error_df, parameters_df)

# end time
```

```
end.time <- Sys.time()
time.taken <- end.time - start.time
time.taken
```

```
time.taken
```

```
## Time difference of 1.790259 hours
```

Here we have a table of our random search results

```
randomsearch <- as.data.frame(randomsearch) %>%
  rename(val_acc = `1 - min(mdcv$evaluation_log$val_error)`) %>%
  arrange(-val_acc)
```

We calculate the error of the best model on the validation set.

```
# Tuned-XGBoost model
set.seed(1234)
params <- list(booster = "gbtree",
               objective = "reg:squarederror",
               max_depth = randomsearch[1,]$max_depth,
               eta = randomsearch[1,]$eta,
               subsample = randomsearch[1,]$subsample,
               colsample_bytree = randomsearch[1,]$colsample_bytree,
               min_child_weight = randomsearch[1,]$min_child_weight)
xgb_tuned <- xgb.train(params = params,
                       data = dtrain,
                       nrounds =1000,
                       print_every_n = 100,
                       eval_metric = "rmse",
                       early_stopping_rounds = 30,
                       watchlist = list(train= dtrain, val= dval))

# Make prediction on dvalid
validation$pred_survived_tuned <- predict(xgb_tuned, dval)

val_r2 = sum((validation$price_doc - validation$pred_survived_tuned) ^ 2 ) / nrow(validation)
val_r2
```

```
val_r2
```

```
## [1] 5.080351e+12
```

And finally here we have error on the test set.

```
set.seed(1234)
params <- list(booster = "gbtree",
               objective = "reg:squarederror",
               max_depth = randomsearch[1,]$max_depth,
               eta = randomsearch[1,]$eta,
               subsample = randomsearch[1,]$subsample,
               colsample_bytree = randomsearch[1,]$colsample_bytree,
               min_child_weight = randomsearch[1,]$min_child_weight)
xgb_tuned <- xgb.train(params = params,
                       data = dtrain,
                       nrounds =1000,
                       eval_metric = "rmse",
                       early_stopping_rounds = 30,
                       watchlist = list(train= dtrain, val= dtest))
# Make prediction on dvalid
test_df$pred_price_tuned <- predict(xgb_tuned, dtest)

test_r2 = sum((test_df$price_doc - test_df$pred_price_tuned) ^ 2 ) / nrow(test_df)
test_r2
```

```
test_r2
```

```
## [1] 7.966814e+12
```

As one would expect, a randomly tuned XGBoost, drastically outperforms simple regression

```
round(test_r2/reg_r2,2)
```

```
## [1] 0.56
```