5420_project

Zijun Ma_T00711782

2023-04-04

```r
# load the packages
pkg_list <- c("tidyverse","MASS", "dplyr", "caret","ModelMetrics",
              "ggplot2", "corrplot","glmnet","corrplot","RColorBrewer",
              "gridExtra","class",
              "readxl","knitr","ipred","rpart","vip","ranger","gridExtra")


# Install packages if needed
for (pkg in pkg_list)
{
  # Try loading the library.
  if ( ! library(pkg, logical.return=TRUE, character.only=TRUE) )
    {
         # If the library cannot be loaded, install it; then load.
        install.packages(pkg)
        library(pkg, character.only=TRUE)
  }
}
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   1.0.1
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.5.0
## v readr   2.1.2      v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
##
## Attaching package: 'MASS'
##
##
## The following object is masked from 'package:dplyr':
##
##     select
##
##
## Loading required package: lattice
##
##
## Attaching package: 'caret'
##
##
```

```
## The following object is masked from 'package:purrr':
##
##     lift
##
##
##
## Attaching package: 'ModelMetrics'
##
##
## The following objects are masked from 'package:caret':
##
##     confusionMatrix, precision, recall, sensitivity, specificity
##
##
## The following object is masked from 'package:base':
##
##     kappa
##
##
## corrplot 0.92 loaded
##
## Loading required package: Matrix
##
##
## Attaching package: 'Matrix'
##
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
##
## Loaded glmnet 4.1-6
##
##
## Attaching package: 'gridExtra'
##
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
##
##
## Attaching package: 'vip'
##
##
## The following object is masked from 'package:utils':
##
##     vi
```

I. Loading Data

```
# df <- read.csv("~/Desktop/5420project/SaYoPillow.csv")
df <- read.csv("~/Desktop/5420project/used_device_data.csv")
orign.df <- df
```

II. Missing Data

```
# deal with missing value
# checking missing data
sum(is.na(df))
```

```
## [1] 202
```

```
df <- na.omit(df)

str(df)
```

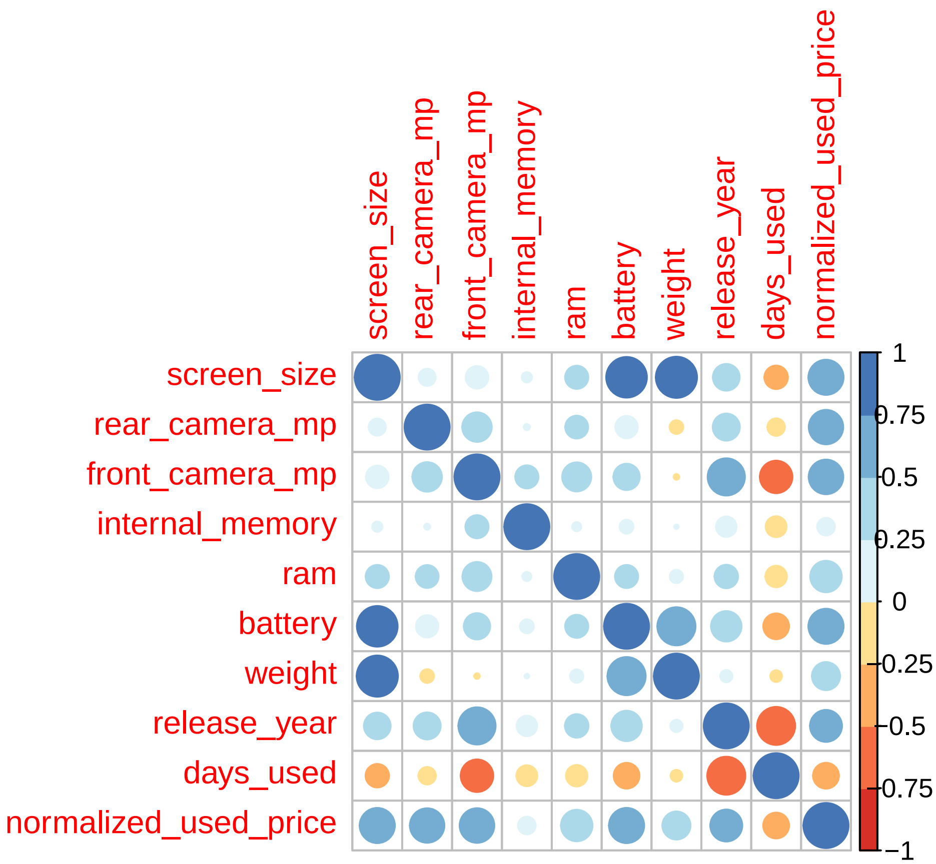
```
## 'data.frame':    3253 obs. of  15 variables:
##  $ device_brand         : chr  "Honor" "Honor" "Honor" "Honor" ...
##  $ os                   : chr  "Android" "Android" "Android" "Android" ...
##  $ screen_size          : num  14.5 17.3 16.7 25.5 15.3 ...
##  $ X4g                  : chr  "yes" "yes" "yes" "yes" ...
##  $ X5g                  : chr  "no" "yes" "yes" "yes" ...
##  $ rear_camera_mp       : num  13 13 13 13 13 13 8 13 13 13 ...
##  $ front_camera_mp      : num  5 16 8 8 8 8 5 8 16 8 ...
##  $ internal_memory      : num  64 128 128 64 64 64 32 64 128 128 ...
##  $ ram                  : num  3 8 8 6 3 4 2 4 6 6 ...
##  $ battery              : num  3020 4300 4200 7250 5000 4000 3020 3400 4000 4000 ...
##  $ weight               : num  146 213 213 480 185 176 144 164 165 176 ...
##  $ release_year         : int  2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 ...
##  $ days_used            : int  127 325 162 345 293 223 234 219 161 327 ...
##  $ normalized_used_price: num  4.31 5.16 5.11 5.14 4.39 ...
##  $ normalized_new_price : num  4.72 5.52 5.88 5.63 4.95 ...
##  - attr(*, "na.action")= 'omit' Named int [1:201] 60 61 62 63 64 65 98 99 100 101 ...
##   ..- attr(*, "names")= chr [1:201] "60" "61" "62" "63" ...
```

```
df$normalized_new_price <- NULL
```

III.EDA

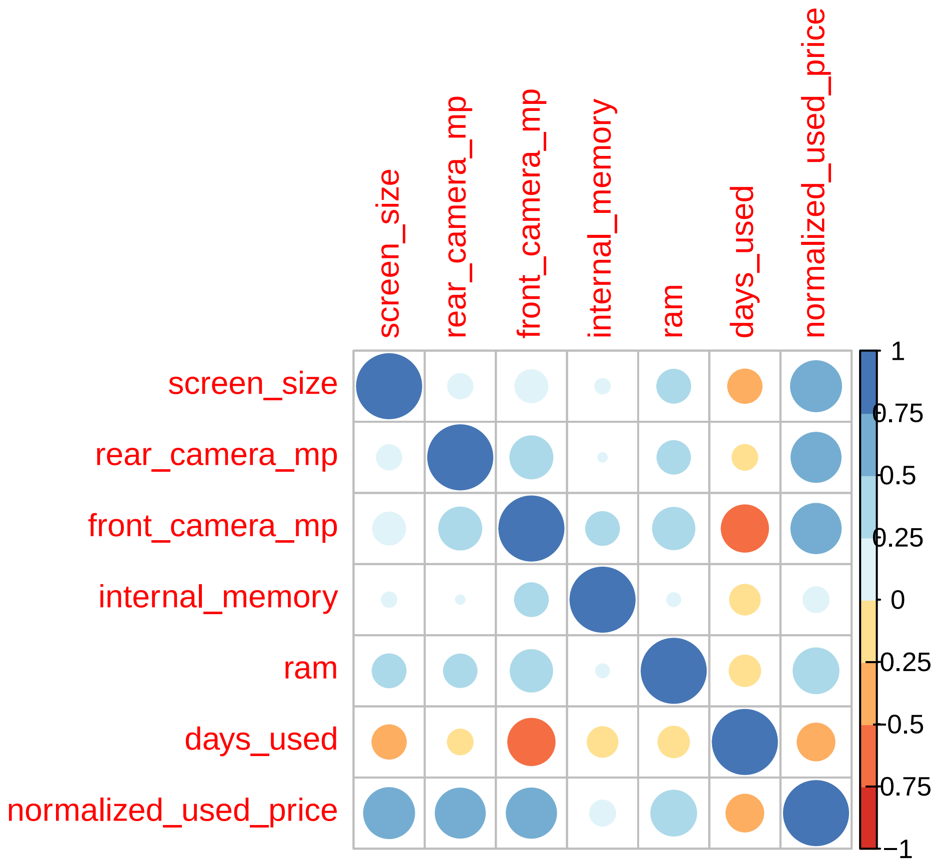
EDA for Numerical Variables.

```
par(mfrow = c(1,1))
num_names <- c("screen_size","rear_camera_mp","front_camera_mp",
               "internal_memory","ram", "battery","weight","release_year","days_used","normalized_used_
df.num <- df[,num_names]

cor_matrix <- cor(df.num)
corrplot(cor_matrix,col = brewer.pal(n=8, name="RdYlBu"))
```

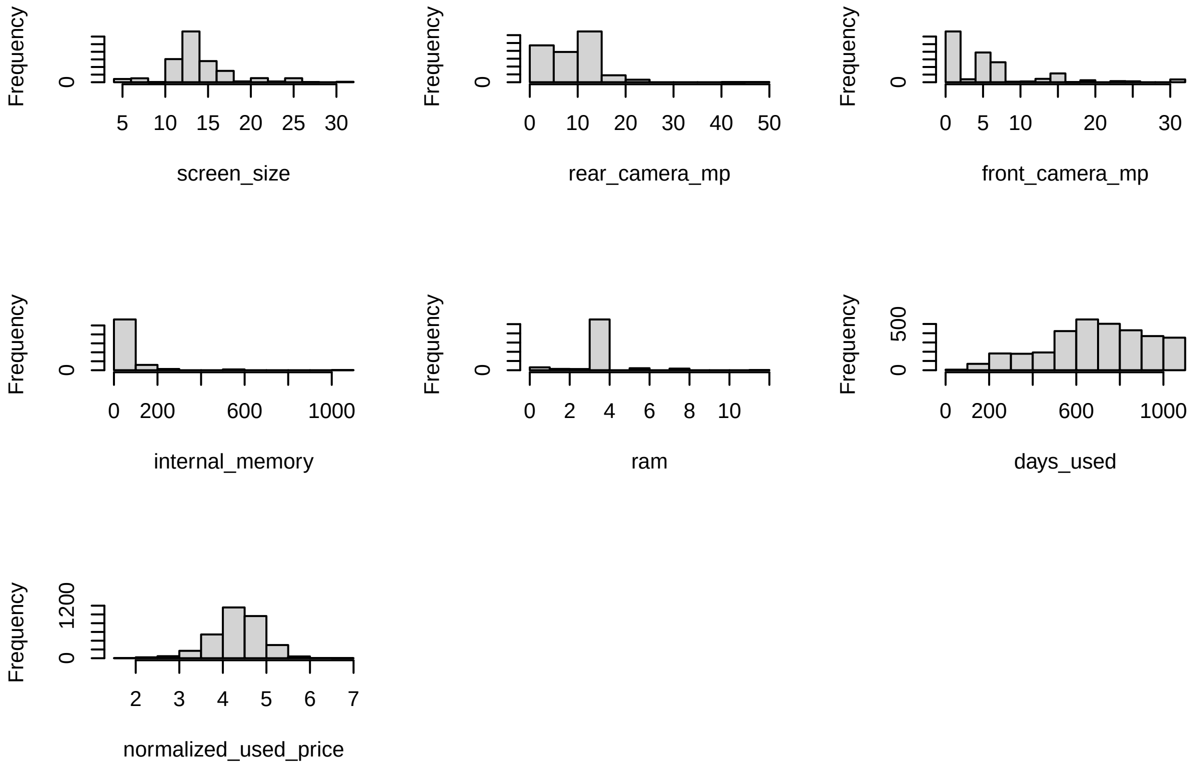


From the correlation plot, I can see there are some high correlation between battery and screen_size, weight and screen_size, release_year and days_used. Thus, I decide to remove battery, weight, and release_year from data set.

```
df <- subset(df, select = -c(battery,weight,release_year) )
df.num <- subset(df.num, select = -c(battery,weight,release_year) )
cor_matrix <- cor(df.num)
corrplot(cor_matrix,col = brewer.pal(n=8, name="RdYlBu"))
```



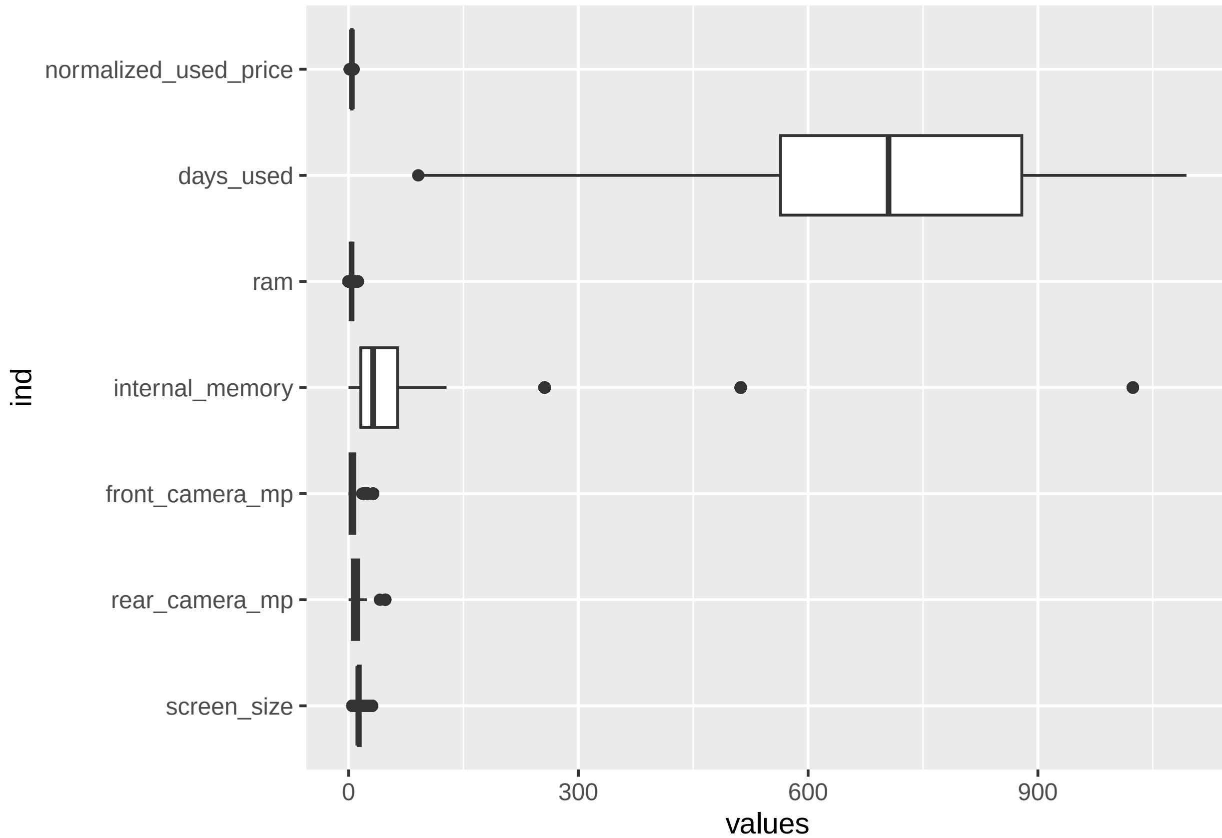

```r
# checking the histogram for numerical variables
par(mfrow = c(3,3))

for(i in names(df.num)){
hist(df[,i],main="", xlab = i)
}
```



From these plots, we can see as the quality (screen_size, rear_camera_mp, front_camera_mp, internal_memory, ram) increase, the count for phones become less. And normalized_used_price follows the normal distribution.

boxplot for numerical variables(6+1)

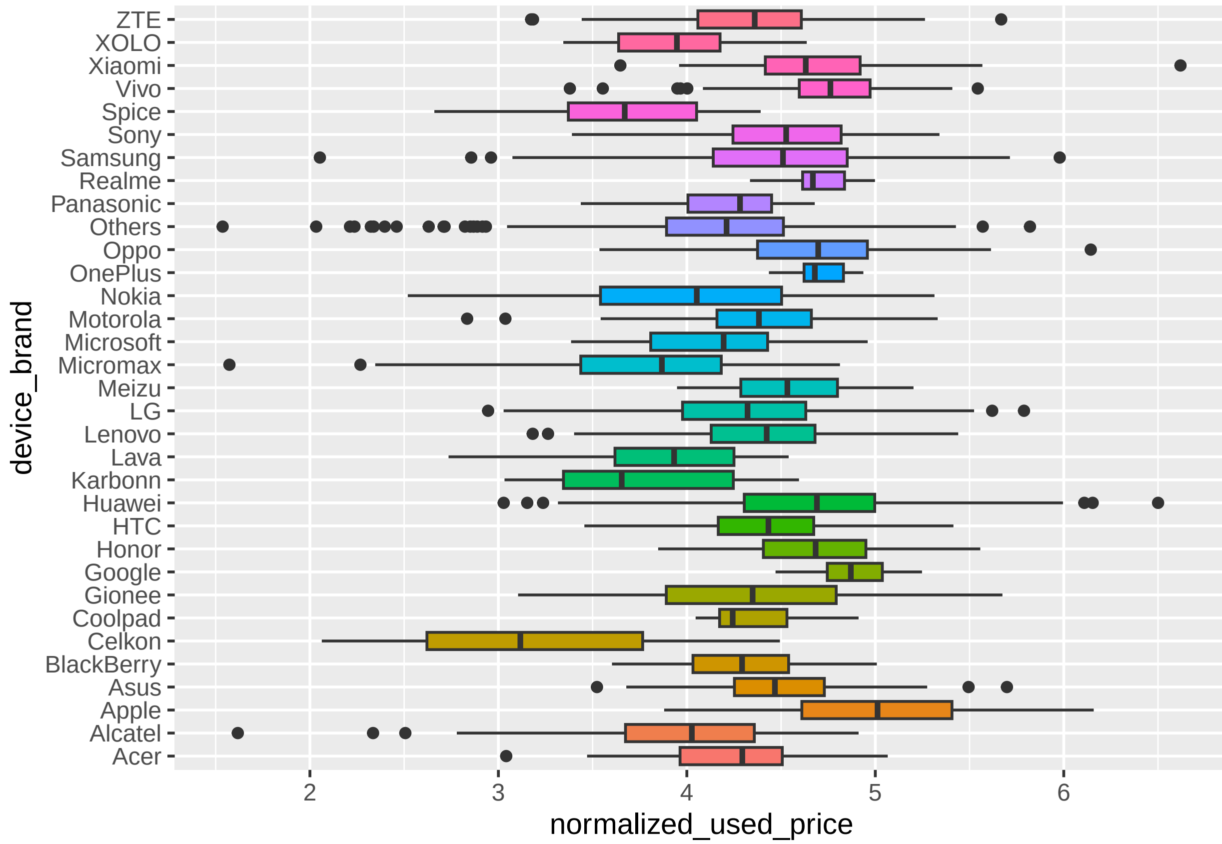
```
par(mfrow = c(1,1))

ggplot(stack(df.num), aes(x = ind, y = values)) +
  geom_boxplot() +
  coord_flip()
```



From the boxplot of our numerical variables, we can see that we have some outliers.

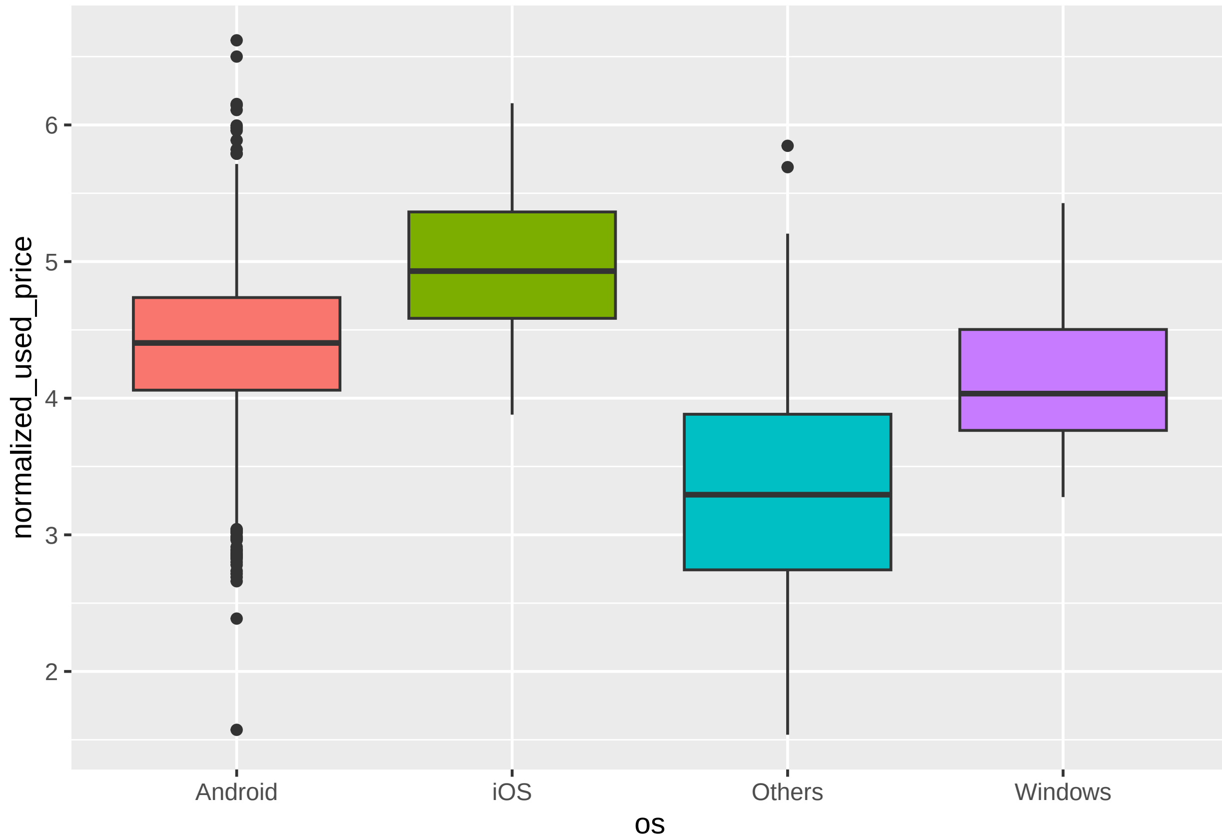
Bivariate Analysis

```
device_brand.plot <- ggplot(df, aes(x = device_brand, y = normalized_used_price, fill=device_brand)) +
  geom_boxplot() +
  theme(legend.position="none") +
  coord_flip()

device_brand.plot
```



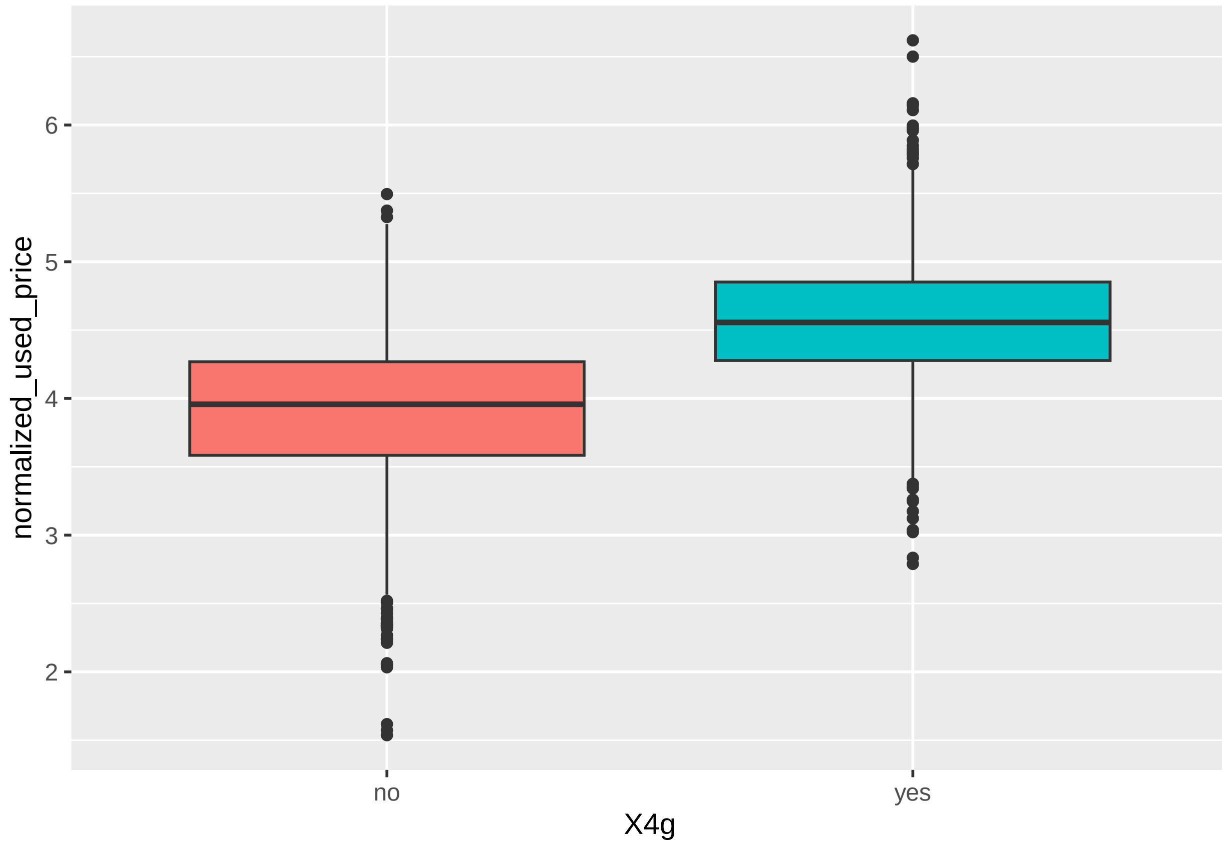
By comparing the device_brand variable with our target variable, we can see that we have some brands(Apple) that are more expensive.

```
os.plot <- ggplot(df, aes(x = os, y = normalized_used_price, fill=os)) +
  geom_boxplot() +
  theme(legend.position="none")
os.plot
```



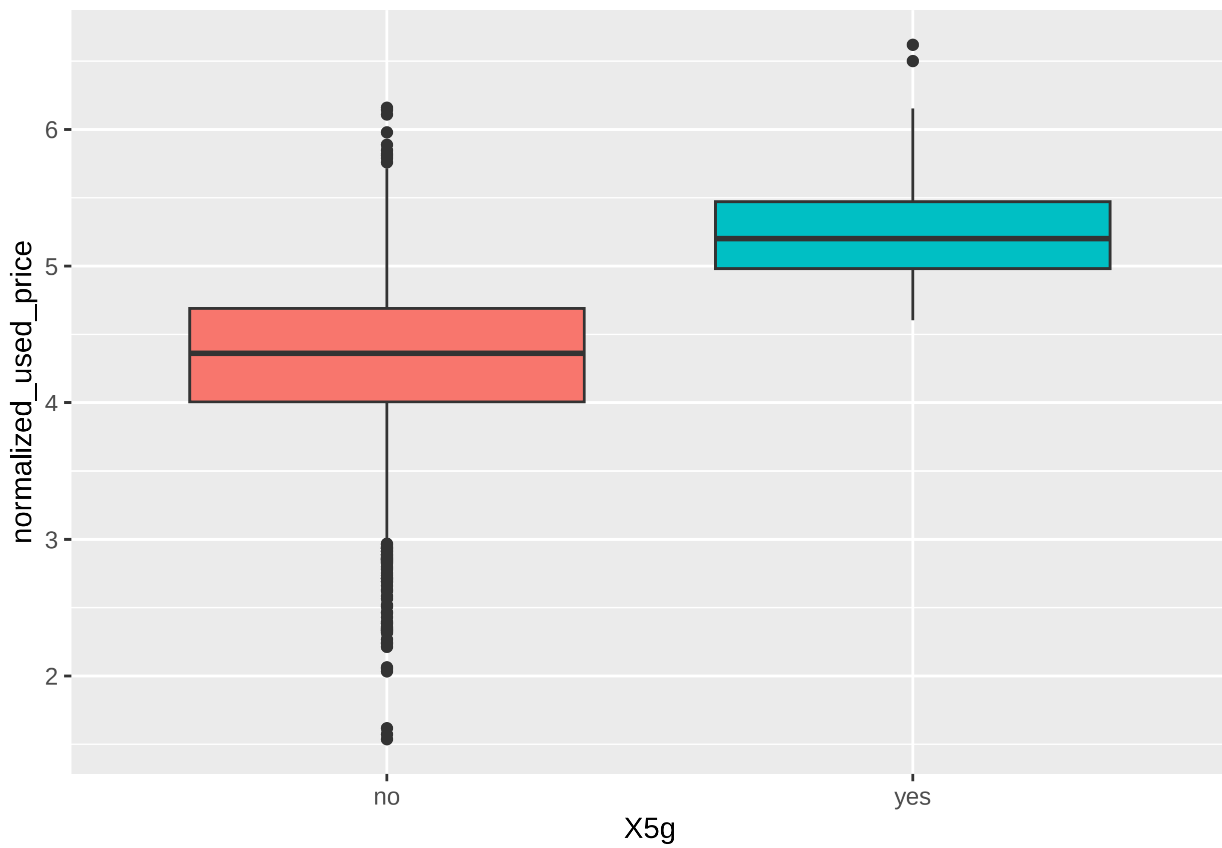
From the boxplot, we can see phones with iOS are more expensive.

```
four.g.plot<- ggplot(df, aes(x = X4g, y = normalized_used_price, fill=X4g)) +
  geom_boxplot() +
  theme(legend.position="none")
four.g.plot
```



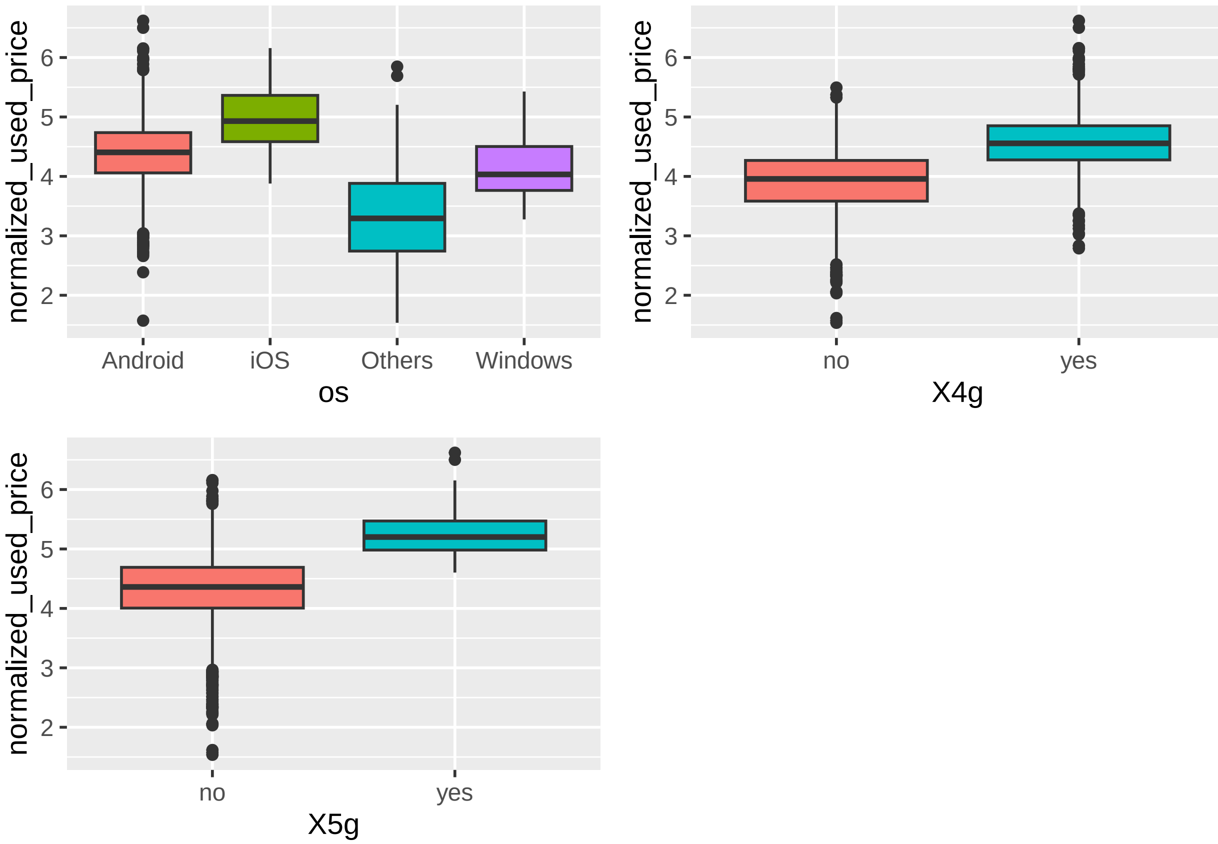
From the boxplot, we can see, when phones with 4G function, the price are more expensive.

```
five.g.plot <- ggplot(df, aes(x = X5g, y = normalized_used_price, fill=X5g)) +
  geom_boxplot() +
  theme(legend.position="none")
five.g.plot
```

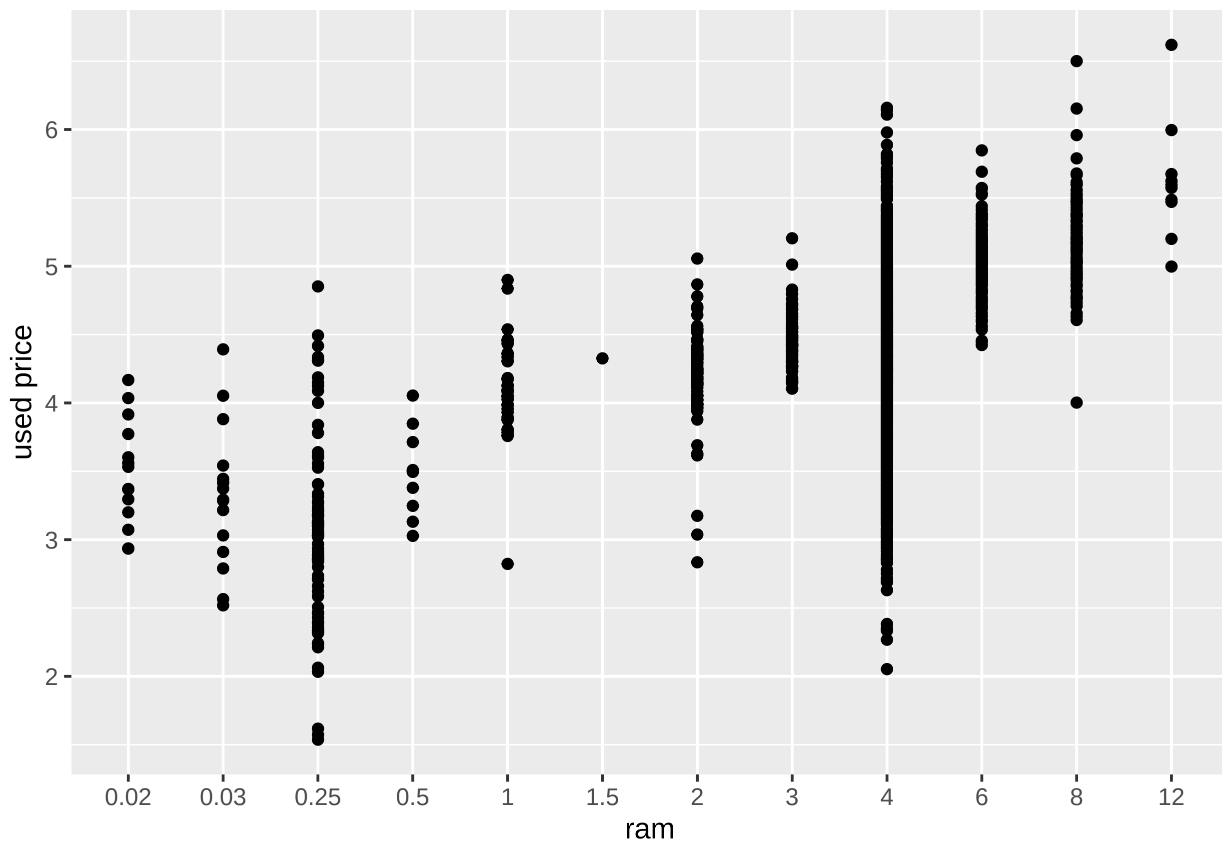


From the boxplot, we can see, when phones with 5G function, the price are more expensive.

```
grid.arrange(os.plot, four.g.plot, five.g.plot, ncol = 2)
```

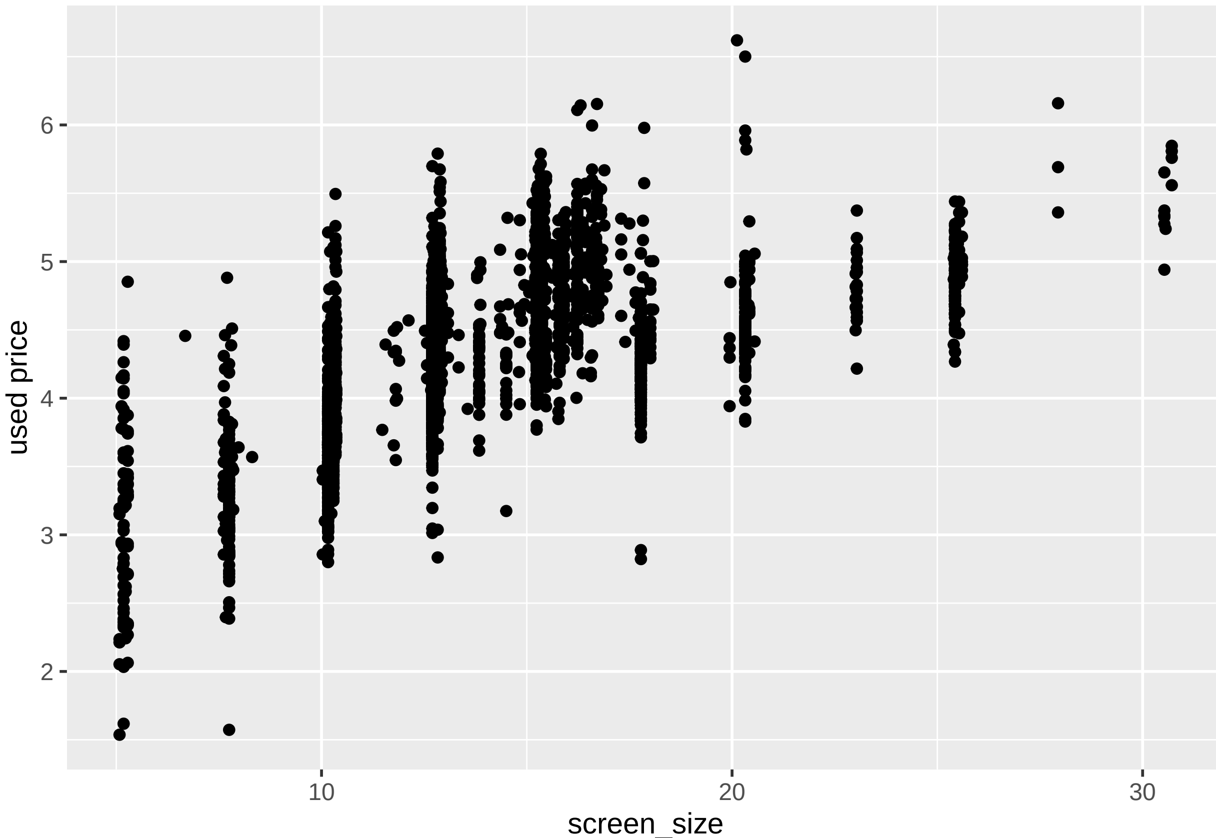



```
df.tep <- df
df.tep$ram <- factor(df.tep$ram)

ram.plot <- ggplot(df.tep, aes(x = ram, y = normalized_used_price, fill=ram)) +
  geom_point() +
  theme(legend.position="none")+
  labs(y = "used price")
ram.plot
```



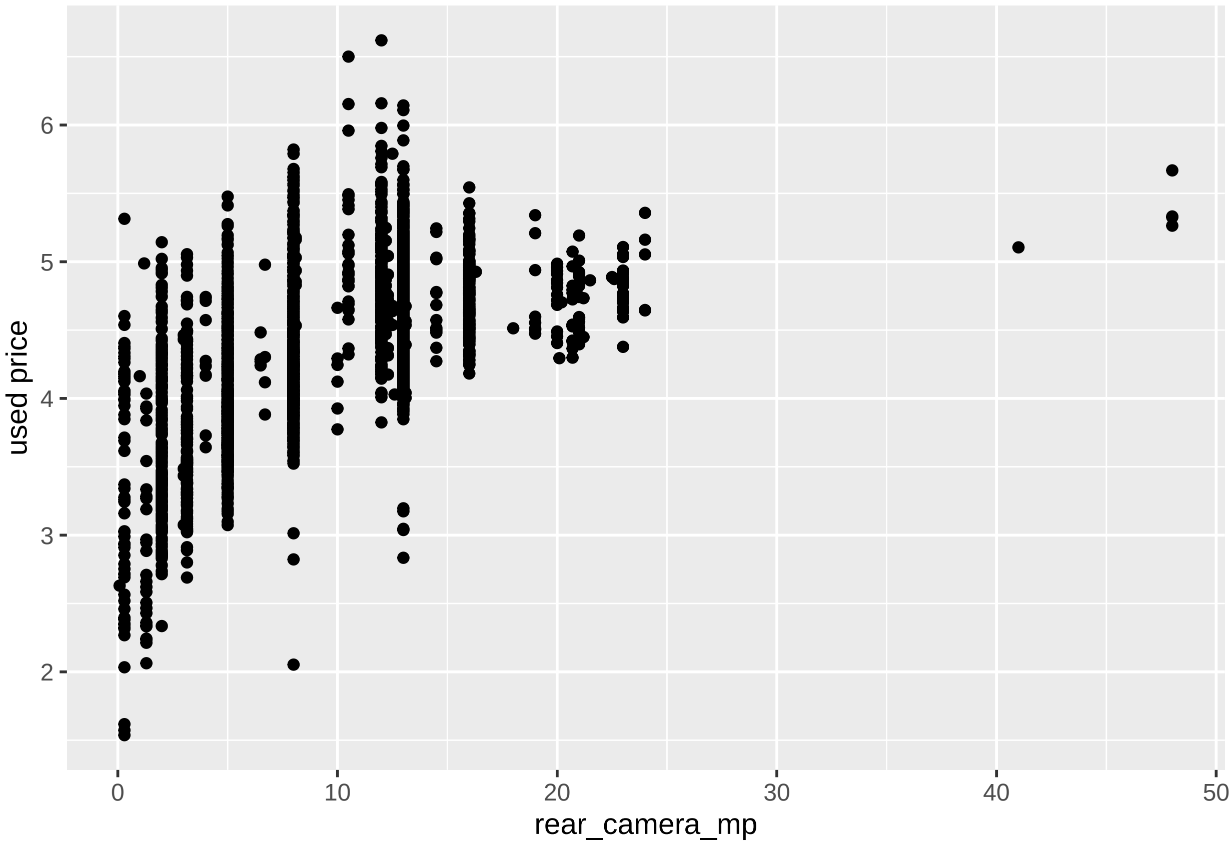
From the scatter plot we can see, the greater the ram power, the more likely the cell phone have higher price.

```
screen_size.plot <- ggplot(df, aes(x = screen_size, y = normalized_used_price)) +
  geom_point()+
  labs(y = "used price")
screen_size.plot
```



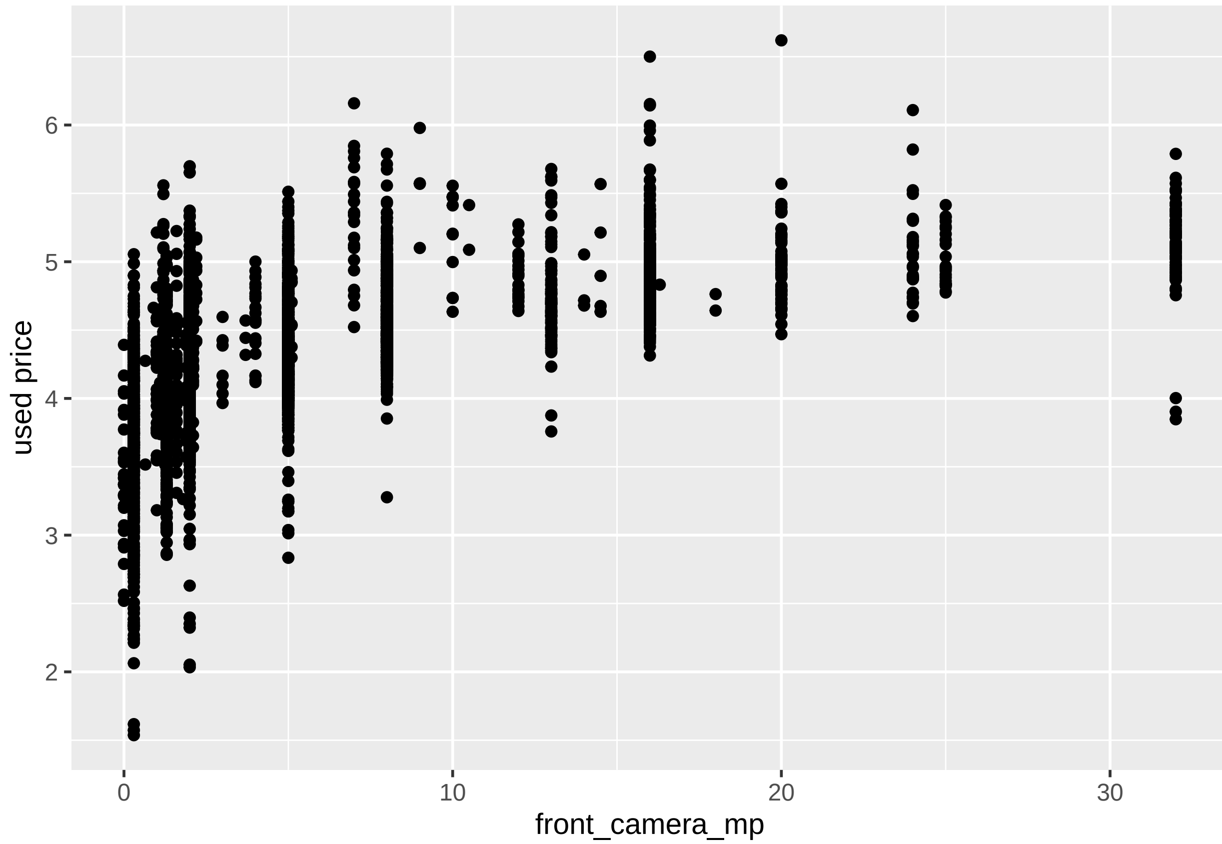
From the scatter plot, we can see the larger the screen size, the higher the phones' price.

```
rear_camera_mp.plot <- ggplot(df, aes(x = rear_camera_mp,
                                      y = normalized_used_price)) +
  geom_point()+
  labs(y = "used price")
rear_camera_mp.plot
```

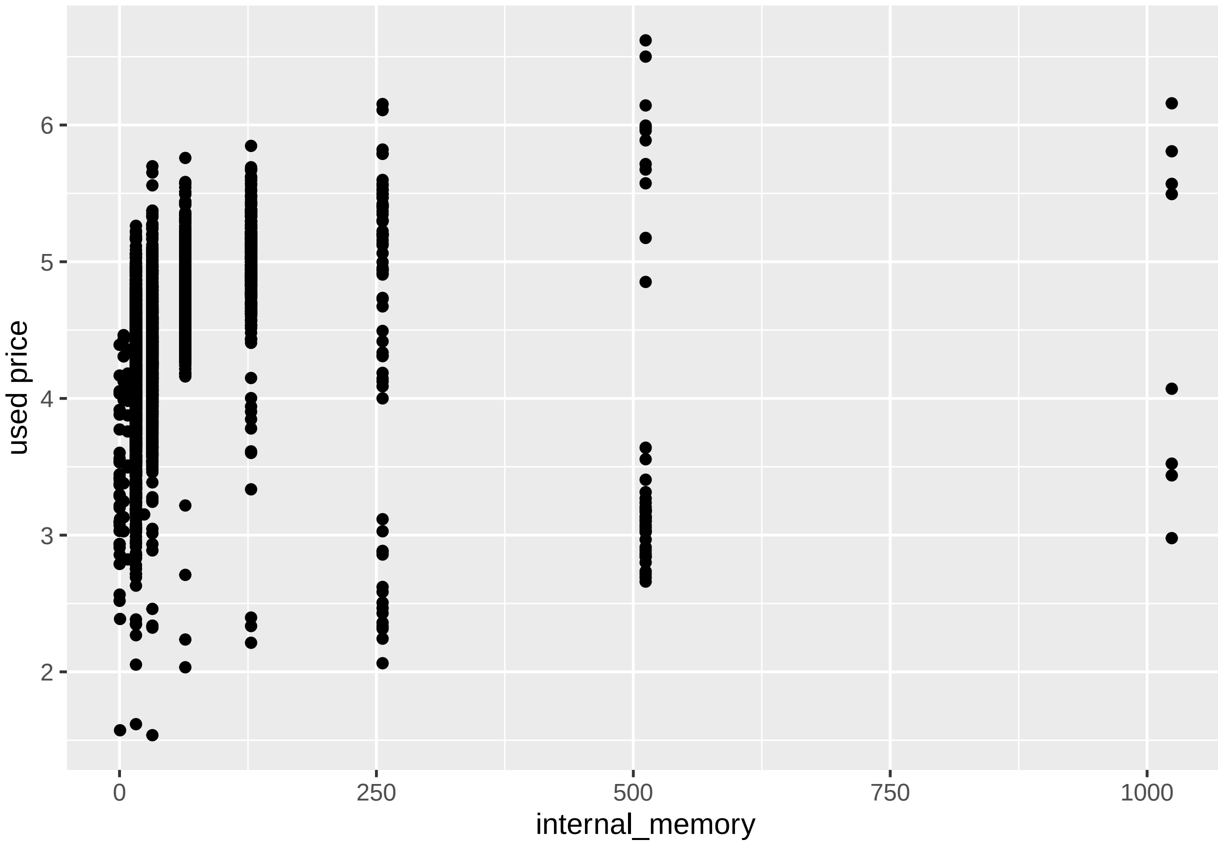


From the scatter plot, we can see the larger the rear_camera_mp, the higher the phones' price.

```
front_camera_mp.plot <- ggplot(df, aes(x = front_camera_mp, y = normalized_used_price)) +
  geom_point()+
  labs(y = "used price")

front_camera_mp.plot
```

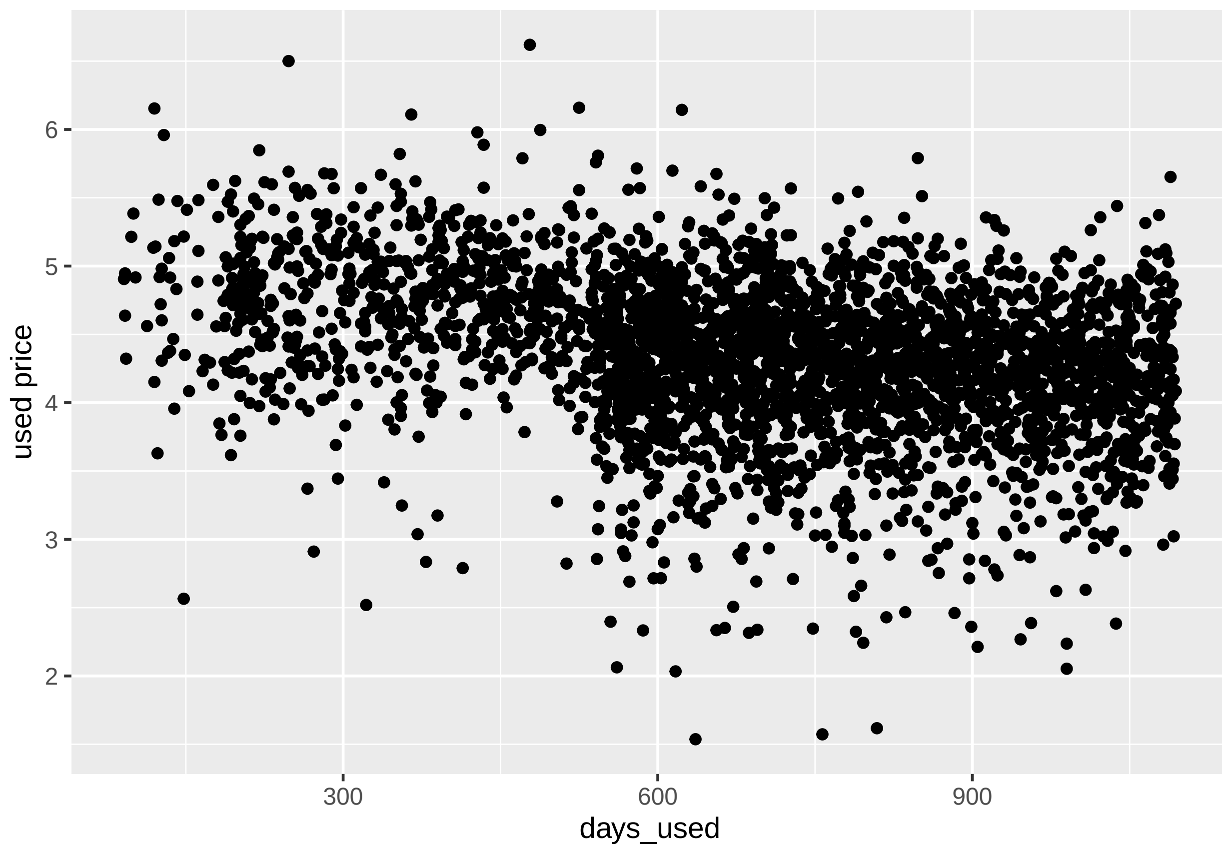



```
internal_memory.plot <- ggplot(df, aes(x = internal_memory, y = normalized_used_price)) +
  geom_point()+
  labs(y = "used price")

internal_memory.plot
```



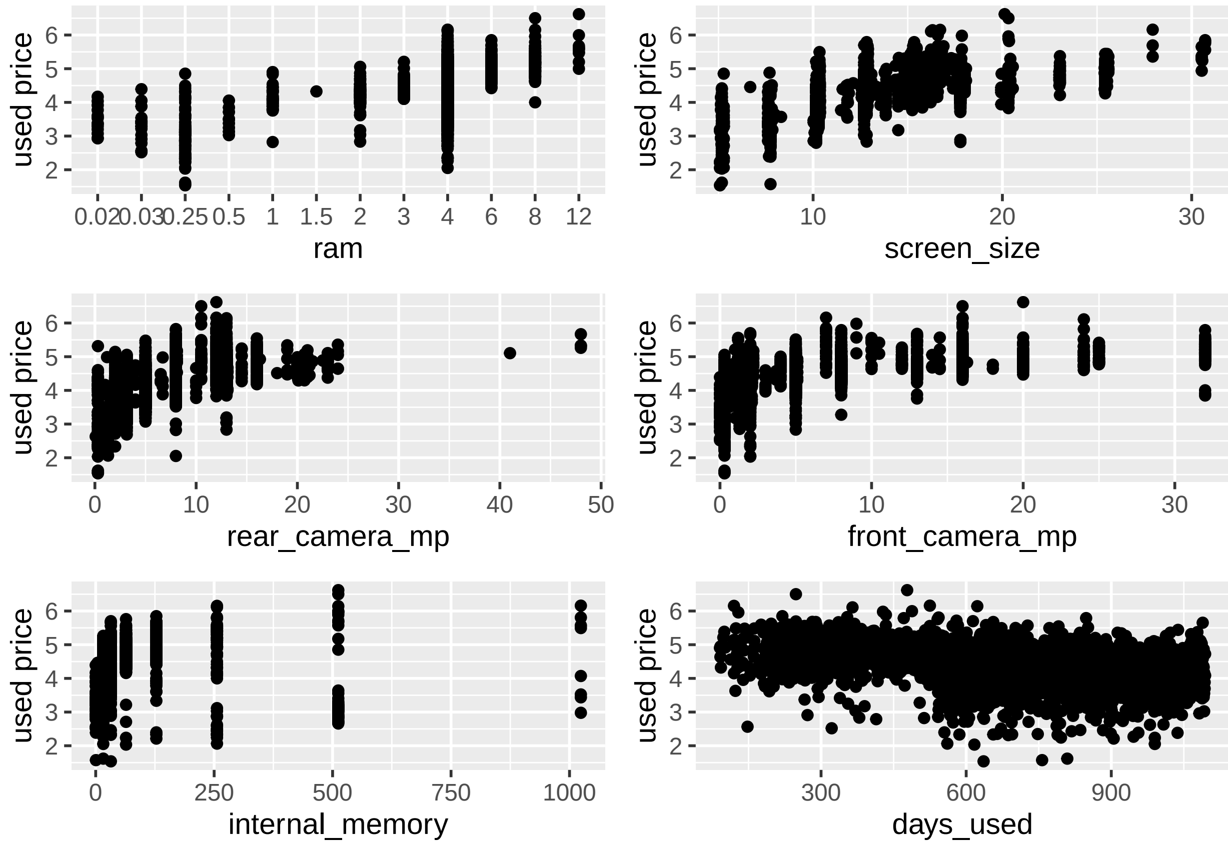
From the previous correlation plot, we already know the correlation between internal_memory and normalized_used_price is very weak, thus from plot, we can see when internal_memory value from lower to higher, the price has the same variation.

```
days_used.plot <- ggplot(df, aes(x = days_used, y = normalized_used_price)) +
  geom_point()+
  labs(y = "used price")
days_used.plot
```



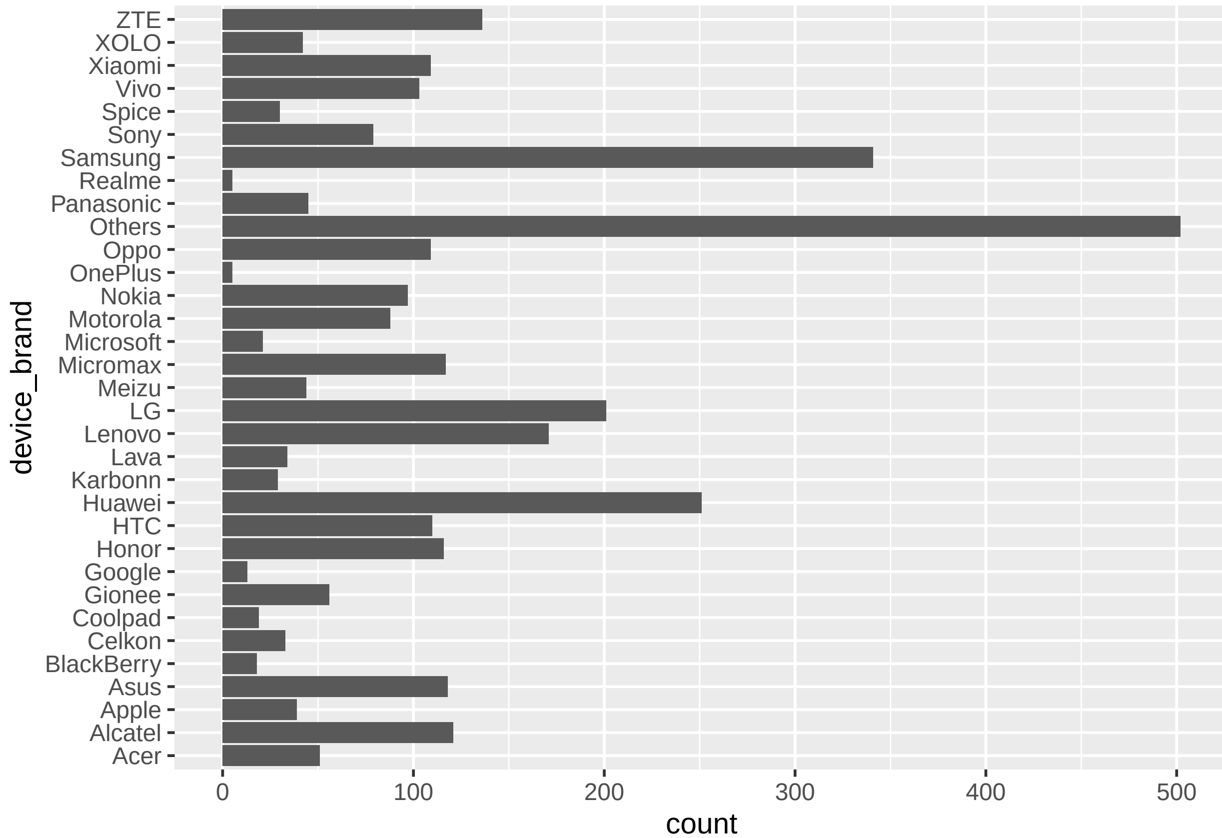
From the scatter plot, we can see the older the phones, the lower the prices.

```
grid.arrange(ram.plot, screen_size.plot, rear_camera_mp.plot,
             front_camera_mp.plot,
             internal_memory.plot, days_used.plot, ncol = 2)
```



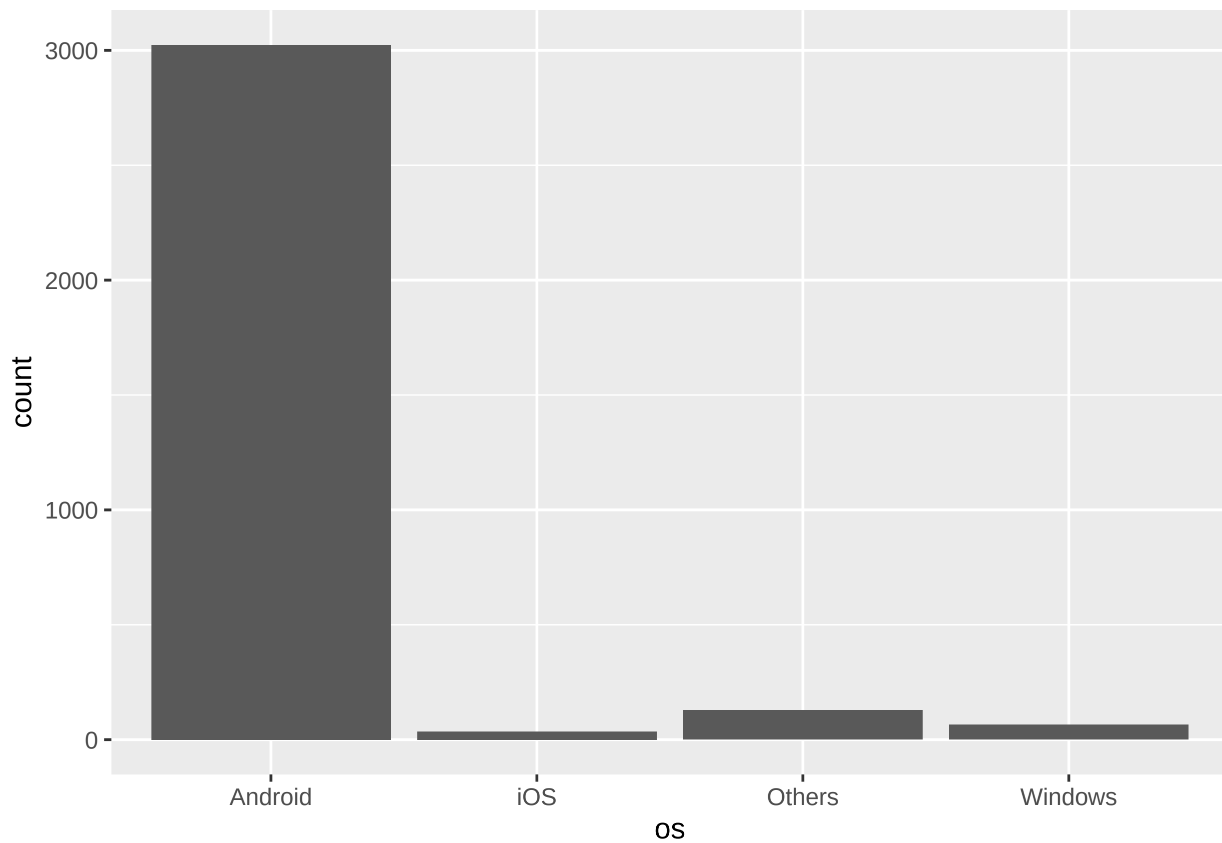
EDA for Categorical Variables(4)

```
cat_names <- c("device_brand","os","X4g","X5g")
df.cat <- df[,cat_names]

ggplot(data=df.cat, aes(x=device_brand)) +
  geom_bar() +
  coord_flip()
```



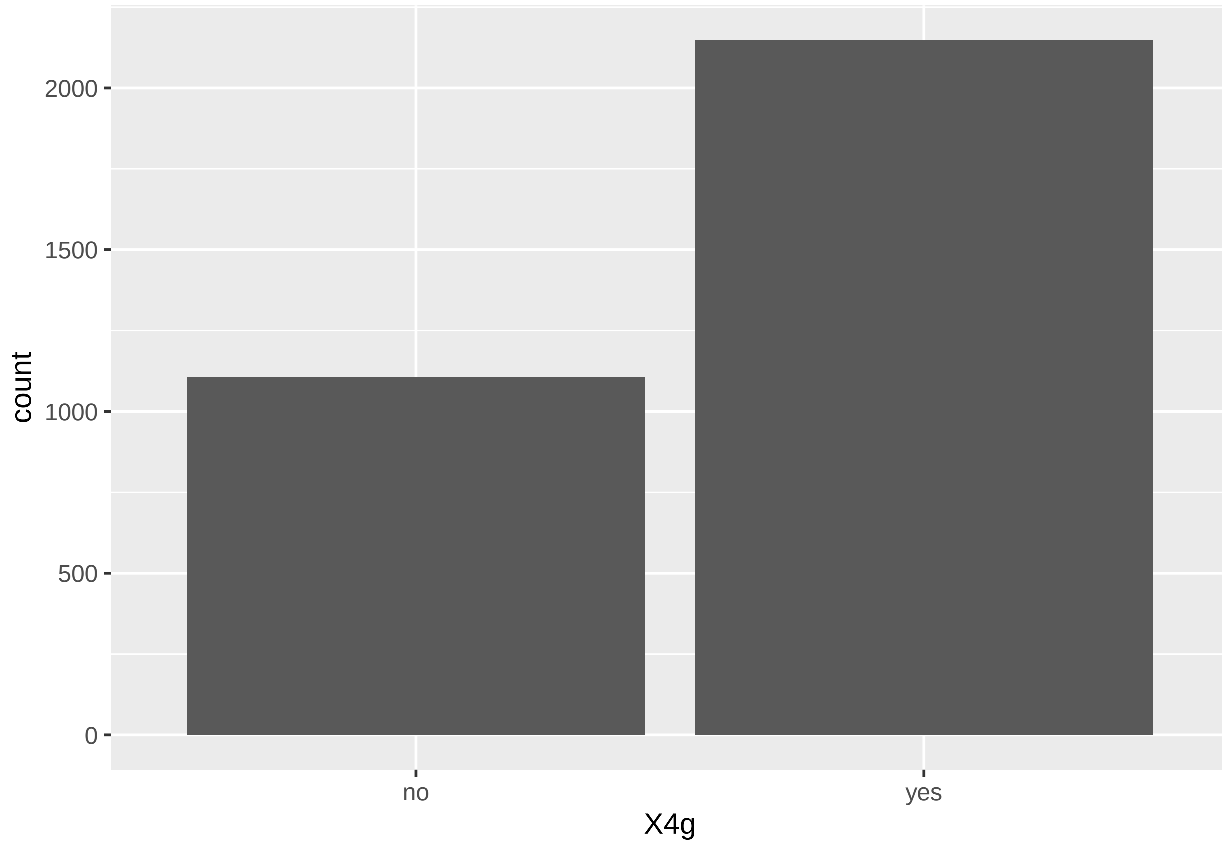
Here we can see the distribution of the cell phone brand, we can conclude that we have several cell phone brands and we have predominance in some like Samsung.

```
ggplot(data=df.cat, aes(x=os)) +
  geom_bar()
```



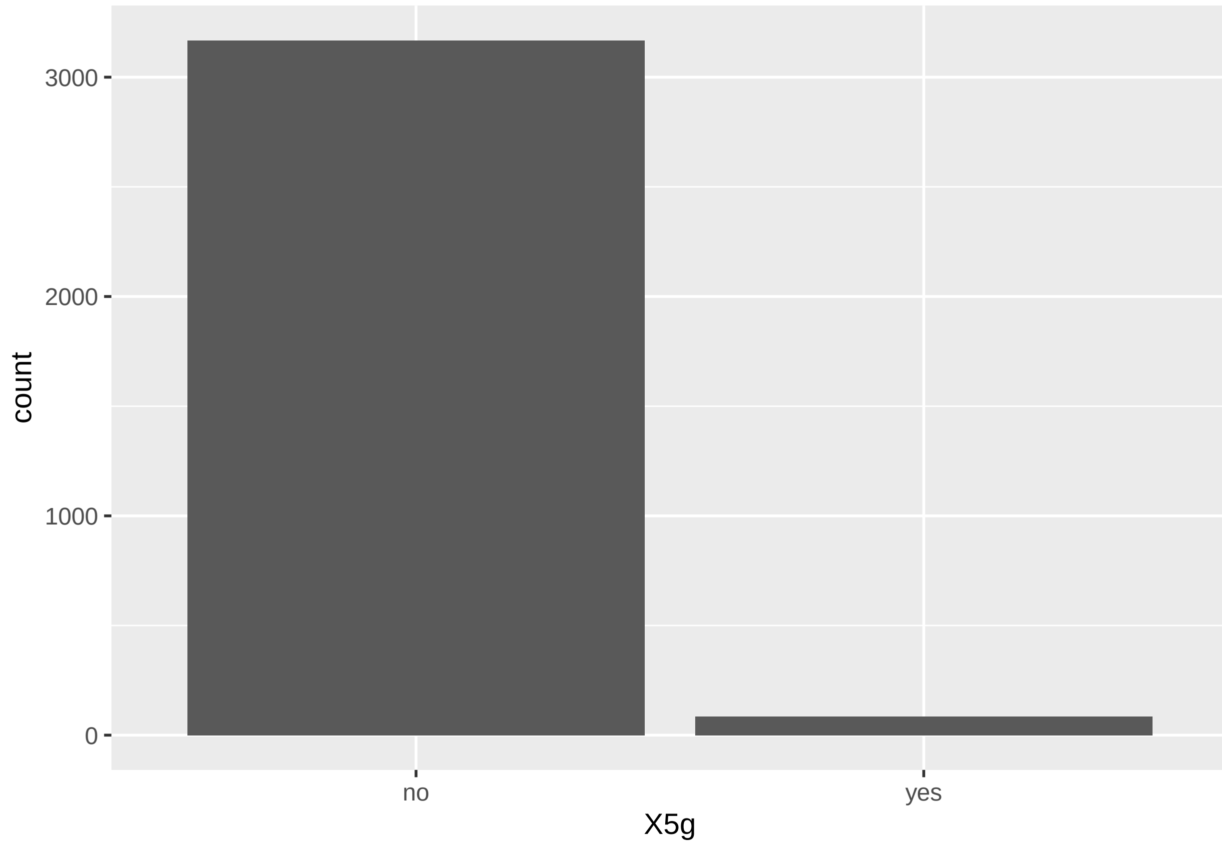
Android accounts for the majority of all OS

```
ggplot(data=df.cat, aes(x=X4g)) +
  geom_bar()
```



Lots of phones have 4G function.

```
ggplot(data=df.cat, aes(x=X5g)) +
  geom_bar()
```



Only few phones have 5G function.

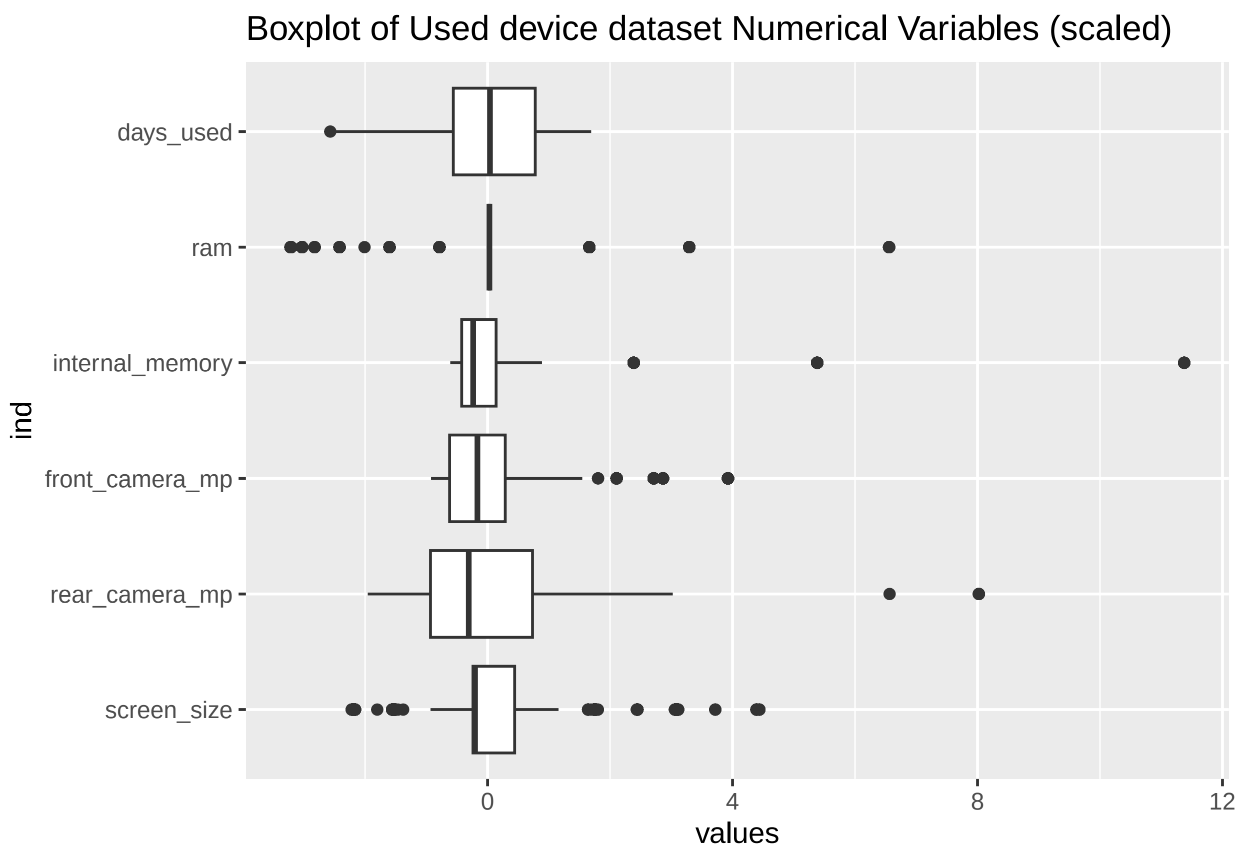
IV. data pre-processing

```r
table(df$device_brand)
```

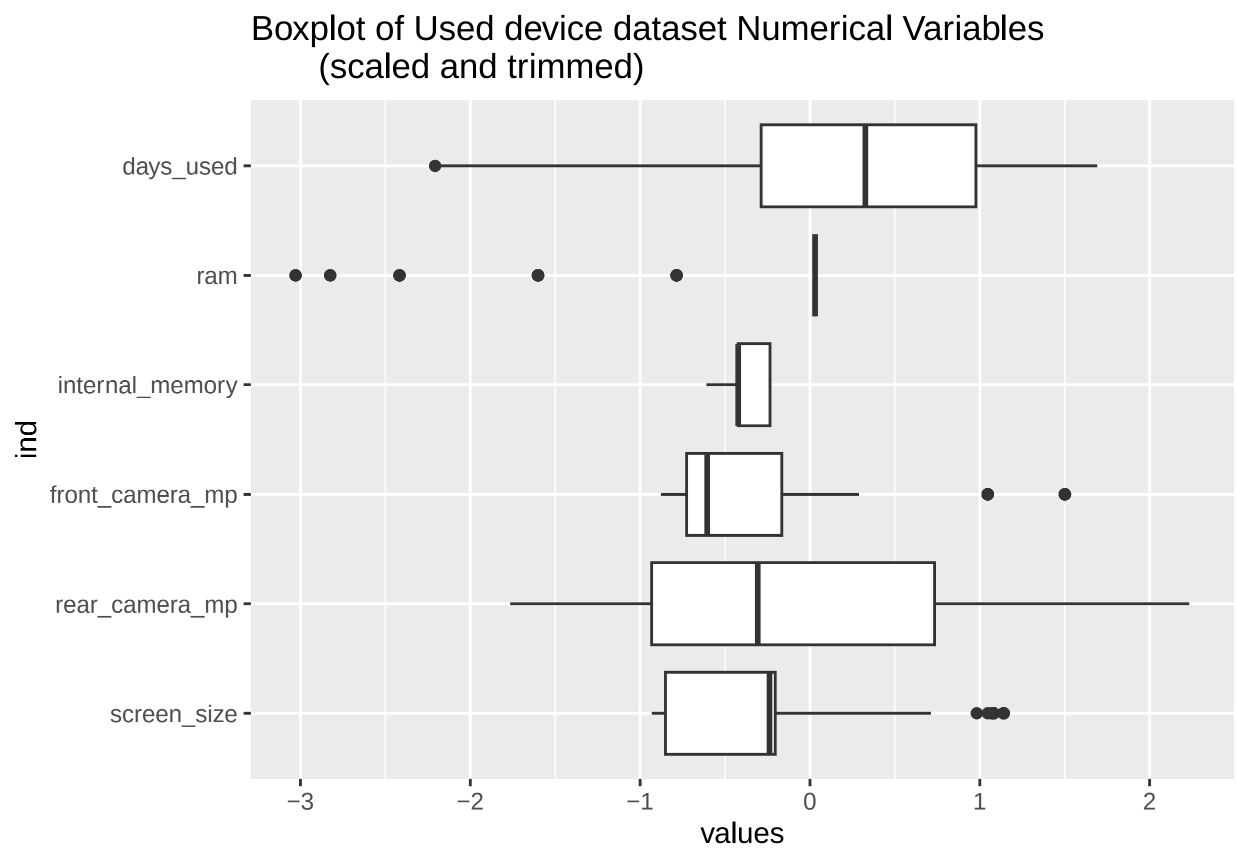
```
##
##       Acer    Alcatel      Apple       Asus BlackBerry     Celkon    Coolpad
##         51        121         39        118         18         33         19
##     Gionee     Google      Honor        HTC     Huawei    Karbonn       Lava
##         56         13        116        110        251         29         34
##     Lenovo         LG      Meizu   Micromax  Microsoft   Motorola      Nokia
##        171        201         44        117         21         88         97
##    OnePlus       Oppo     Others  Panasonic     Realme    Samsung       Sony
##          5        109        502         45          5        341         79
##      Spice       Vivo     Xiaomi       XOLO        ZTE
##         30        103        109         42        136
```

```r
## Grouping device_brand variable
df.2 <- df
df.2$brand_average <- round(
  ave(df.2$normalized_used_price, df.2$device_brand, FUN = mean),
  2
```

```r
)

for (i in 1:nrow(df.2)) {
  if (df.2[[i,"brand_average"]] < 3.7){
    df.2[[i,"brand_average"]] <- "A"
  }else if (df.2[[i,"brand_average"]]>=3.7 & df.2[[i,"brand_average"]]<4.0){
    df.2[[i,"brand_average"]] <- "B"
  }else if (df.2[[i,"brand_average"]]>=4.0 & df.2[[i,"brand_average"]]<4.3){
    df.2[[i,"brand_average"]] <- "C"
  }else if (df.2[[i,"brand_average"]]>=4.3 & df.2[[i,"brand_average"]]<4.7){
    df.2[[i,"brand_average"]] <- "D"
  }else if (df.2[[i,"brand_average"]]>=4.7 ){
    df.2[[i,"brand_average"]] <- "E"
  }
}
df.2$device_brand <- NULL
names(df.2)[names(df.2)=="brand_average"] <- "device_brand"

#scale df.num
scale.df.num <- as.data.frame(cbind(normalized_used_price =
                                      df.2$normalized_used_price,
                                scale(subset(df.num, select
                                             = -c(normalized_used_price)))))

ggplot(stack(subset(scale.df.num, select = -c(normalized_used_price))),
       aes(x = ind, y = values)) +
  geom_boxplot() +
  coord_flip() +
  labs(title = "Boxplot of Used device dataset Numerical Variables (scaled)")
```




```
# add the scaled columns back to the original data frame
df_scaled <- cbind(df.2 %>% select_if(is.character), scale.df.num)

## remove the outliers
par(mfrow = c(1,1))
# remove the outliers for all variables except response variable.
trimmed.df = df_scaled
for (col in names(df.num)[-c(5,7)]) {
# Identify potential outliers
outliers <- boxplot(trimmed.df[,col], plot = FALSE)$out
trimmed.df <- subset(trimmed.df, !trimmed.df[,col] %in% outliers)
}

ggplot(stack(trimmed.df[,c(6:11)]),
       aes(x = ind, y = values)) +
  geom_boxplot() +
  coord_flip()+
  labs(title = "Boxplot of Used device dataset Numerical Variables
       (scaled and trimmed)")
```




```r
df.full <- trimmed.df
table(df.full$X5g)
```

```
##
##   no
## 1958
```

```r
table(df.full$ram)
```

```
##
##   -3.02877558993269   -2.82484292508587   -2.41697759539223   -1.60124693600494
##                   5                   4                  22                  49
## -0.785516276617663 0.0302143827696191
##                 37               1841
```

```r
df.full$X5g <- NULL
cat_names <- c("device_brand","os","X4g")

## factor categorical variable
for (i in cat_names){
df.full[,i] <- factor(df.full[,i])
}

# Split the data into training and testing
```

```r
set.seed(5420)
sample <- sample(c(TRUE, FALSE), nrow(df.full), replace=TRUE,prob=c(0.80,0.20))
# training data
train.df <- df.full[sample, ]
train.x <- subset(train.df, select = -c(normalized_used_price))
train.y <- train.df$normalized_used_price
# testing data
test.df <- df.full[!sample, ]
test.x <- subset(test.df, select = -c(normalized_used_price))
test.y <- test.df$normalized_used_price
```

Note that the boxplot() function identifies outliers using the interquartile range (IQR) method, which defines an outlier as a value that is more than 1.5 times the IQR below the first quartile (Q1) or above the third quartile (Q3) of the data set.

V. Method

linear regression model

```r
linmod <- lm(normalized_used_price~., data = train.df)
summary(linmod)
```

```
##
## Call:
## lm(formula = normalized_used_price ~ ., data = train.df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.36397 -0.16610 -0.00773  0.15076  1.29135
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      4.254769   0.054470  78.113  < 2e-16 ***
## osiOS            0.219217   0.087778   2.497 0.012615 *
## osOthers        -0.103436   0.070333  -1.471 0.141584
## osWindows        0.026632   0.037806   0.704 0.481253
## X4gyes           0.060997   0.016459   3.706 0.000218 ***
## device_brandB    0.039882   0.048997   0.814 0.415796
## device_brandC    0.079948   0.048500   1.648 0.099471 .
## device_brandD    0.124532   0.047999   2.594 0.009564 **
## device_brandE    0.135564   0.067967   1.995 0.046267 *
## screen_size      0.225357   0.013359  16.870  < 2e-16 ***
## rear_camera_mp   0.226308   0.010485  21.584  < 2e-16 ***
## front_camera_mp  0.087396   0.019237   4.543 5.97e-06 ***
## internal_memory  0.310321   0.071192   4.359 1.39e-05 ***
## ram              0.106661   0.016408   6.501 1.08e-10 ***
## days_used        0.033853   0.008542   3.963 7.74e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2509 on 1543 degrees of freedom
## Multiple R-squared:  0.5885, Adjusted R-squared:  0.5847
## F-statistic: 157.6 on 14 and 1543 DF,  p-value: < 2.2e-16
```

```r
confint(linmod)
```

```
##                         2.5 %     97.5 %
## (Intercept)       4.147927030 4.36161138
## osiOS             0.047039113 0.39139403
## osOthers         -0.241393406 0.03452143
## osWindows        -0.047523282 0.10078822
## X4gyes            0.028711970 0.09328230
## device_brandB    -0.056226878 0.13599047
## device_brandC    -0.015184537 0.17507989
## device_brandD     0.030380954 0.21868278
## device_brandE     0.002246263 0.26888225
## screen_size       0.199154555 0.25156005
## rear_camera_mp    0.205742004 0.24687384
## front_camera_mp   0.049663430 0.12512861
## internal_memory   0.170678482 0.44996363
## ram               0.074476956 0.13884429
## days_used         0.017097023 0.05060894
```

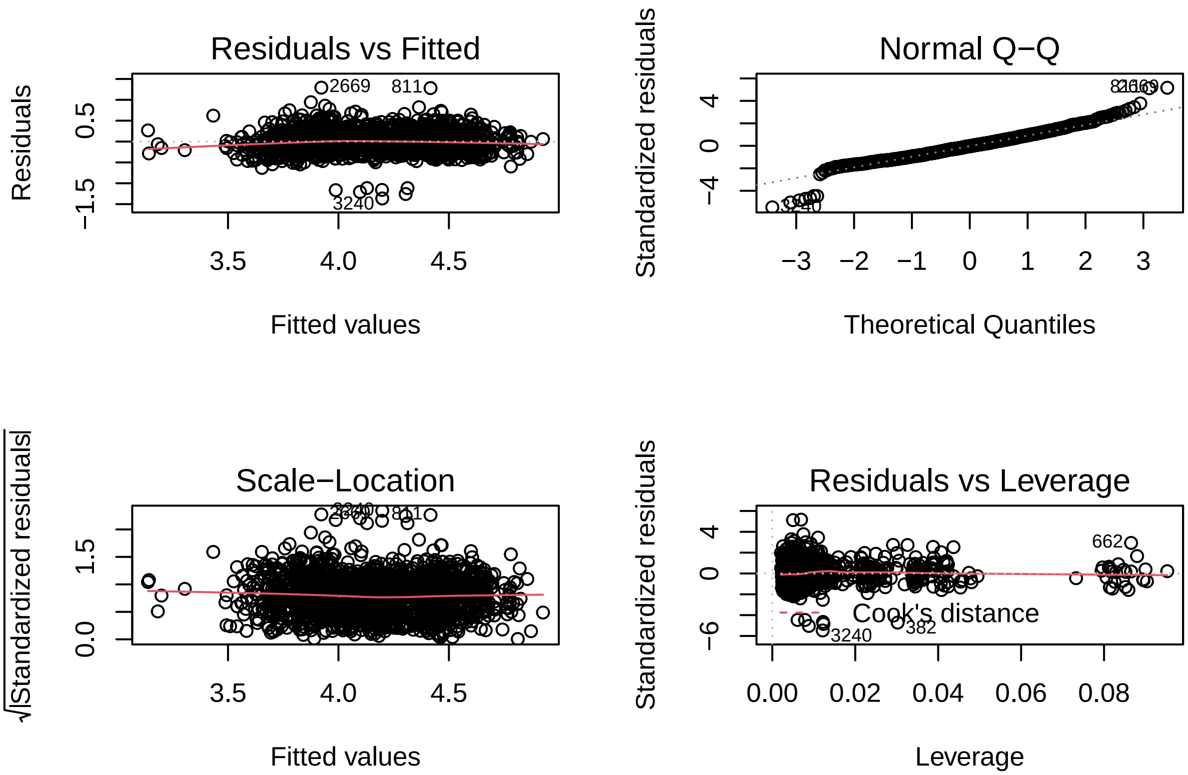
```r
# Make predictions on the testing set
predictions <- predict(linmod, newdata = test.df)
# Calculate the MSE
mse.lm <- mean((test.y - predictions)^2)
mse.lm
```

```
## [1] 0.06495327
```

```r
adjr2 <- summary(linmod)$adj.r.squared
adjr2
```

```
## [1] 0.5847265
```

```r
par(mfrow=c(2,2))
plot(linmod)
```


Residuals vs Fitted
Residuals
Fitted values
Normal Q−Q
Standardized residuals
Theoretical Quantiles
Scale−Location
√|Standardized residuals|
Fitted values
Residuals vs Leverage
Standardized residuals
Cook's distance
Leverage

stepwise regression (both)

```r
set.seed(5420)
# Train the model
step.model <- train(normalized_used_price~., data = train.df,
                    method = "lmStepAIC",
                    trControl = trainControl(method = "cv", number = 10),
                    trace = FALSE
                    )
# Model accuracy
step.model$results
```

```
##   parameter      RMSE  Rsquared       MAE     RMSESD RsquaredSD      MAESD
## 1      none 0.2523114 0.5815378 0.1931709 0.01975808  0.0442446 0.01116546
```

```r
# Final model coefficients
step.model$finalModel
```

```
##
## Call:
## lm(formula = .outcome ~ osiOS + osOthers + X4gyes + device_brandC +
##     device_brandD + device_brandE + screen_size + rear_camera_mp +
##     front_camera_mp + internal_memory + ram + days_used, data = dat)
##
## Coefficients:
##     (Intercept)            osiOS         osOthers           X4gyes
##         4.29325          0.21889         -0.10480          0.06178
##   device_brandC    device_brandD    device_brandE      screen_size
##         0.04667          0.08846          0.09972          0.22482
##  rear_camera_mp  front_camera_mp  internal_memory              ram
##         0.22680          0.08590          0.32090          0.10586
##       days_used
##         0.03401
```

```r
# Summary of the model
summary(step.model$finalModel)
```

```
##
## Call:
## lm(formula = .outcome ~ osiOS + osOthers + X4gyes + device_brandC +
##     device_brandD + device_brandE + screen_size + rear_camera_mp +
##     front_camera_mp + internal_memory + ram + days_used, data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.36448 -0.16702 -0.00788  0.15146  1.28840
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      4.293247   0.031323 137.066  < 2e-16 ***
## osiOS            0.218893   0.087754   2.494 0.012721 *
## osOthers        -0.104804   0.070253  -1.492 0.135952
## X4gyes           0.061780   0.016428   3.761 0.000176 ***
## device_brandC    0.046670   0.020547   2.271 0.023260 *
## device_brandD    0.088459   0.019005   4.655 3.52e-06 ***
## device_brandE    0.099722   0.051636   1.931 0.053634 .
## screen_size      0.224824   0.013295  16.910  < 2e-16 ***
## rear_camera_mp   0.226798   0.010472  21.657  < 2e-16 ***
## front_camera_mp  0.085899   0.019172   4.480 8.00e-06 ***
## internal_memory  0.320903   0.070277   4.566 5.36e-06 ***
## ram              0.105857   0.016386   6.460 1.40e-10 ***
## days_used        0.034009   0.008534   3.985 7.06e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2508 on 1545 degrees of freedom
## Multiple R-squared:  0.5881, Adjusted R-squared:  0.5849
## F-statistic: 183.9 on 12 and 1545 DF,  p-value: < 2.2e-16
```

```r
# Make predictions on the testing set
```

```r
predictions <- predict(step.model, newdata = test.df)
# Calculate the MSE
mse.both <- mean((test.y - predictions)^2)
mse.both
```

```
## [1] 0.0650784
```

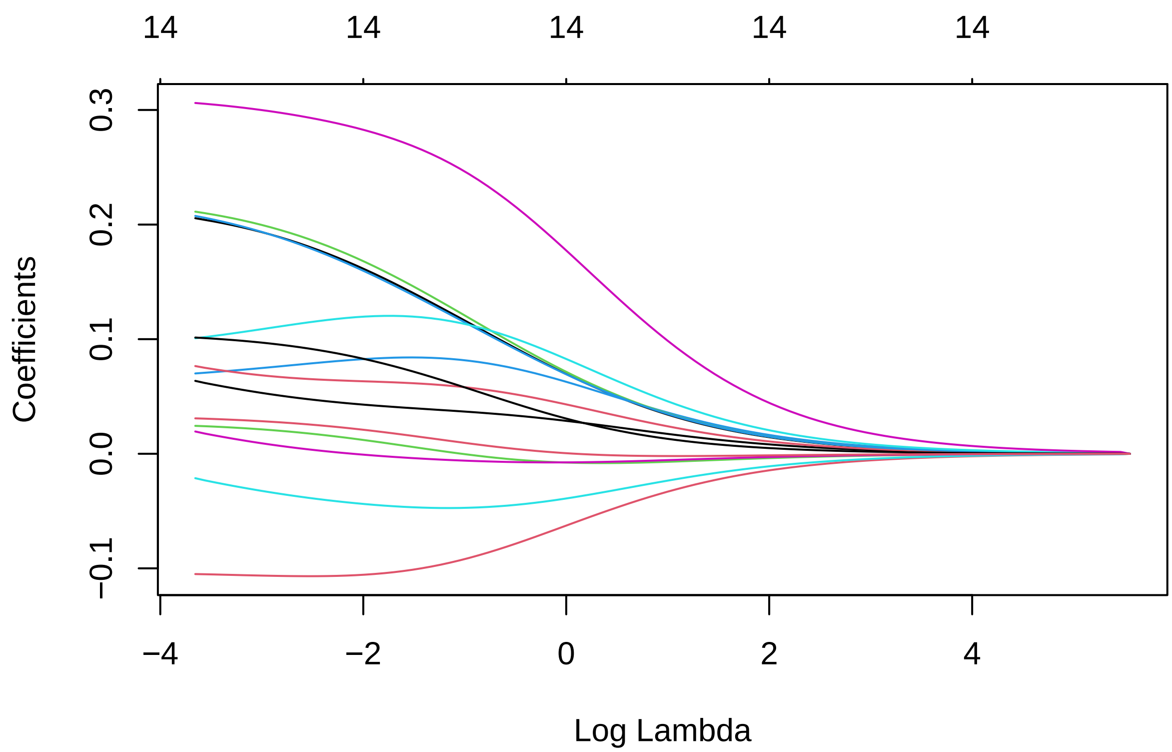
```r
adjr2.both <- summary(step.model)$adj.r.squared
adjr2.both
```

```
## [1] 0.5849493
```

ridge regression

```
train.X <- model.matrix(~ ., data = train.x)
train.X <- train.X[,-1]
test.X <- model.matrix(~ ., data = test.x)
test.X <- test.X[,-1]
```

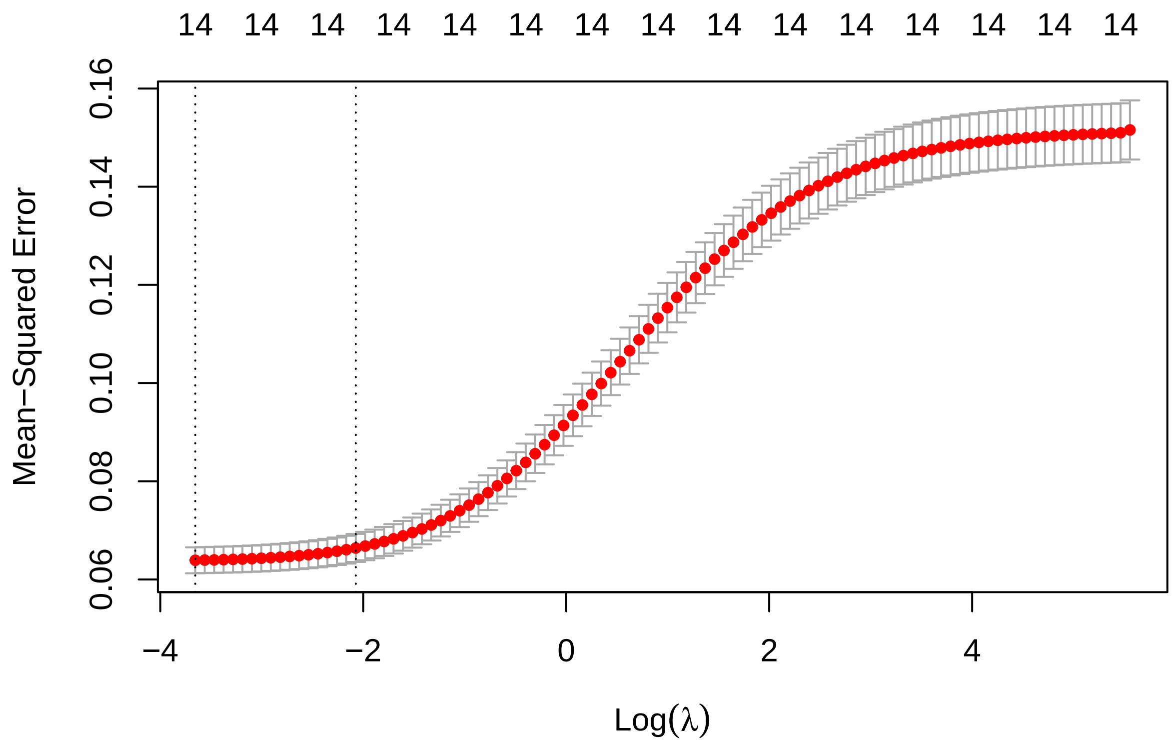
```
set.seed(5420)
par(mfrow = c(1,1))
ridge.model <- glmnet(train.X, train.y, alpha = 0)
plot(ridge.model, xvar = "lambda")
```




```
cv <- cv.glmnet(train.X, train.y, alpha=0)
best.lambda <- cv$lambda.min
best.lambda
```

```
## [1] 0.02584945
```

```
plot(cv)
```




```
ridge.best <- glmnet(train.X, train.y, alpha = 0, lambda = best.lambda)
ridge.best$beta
```

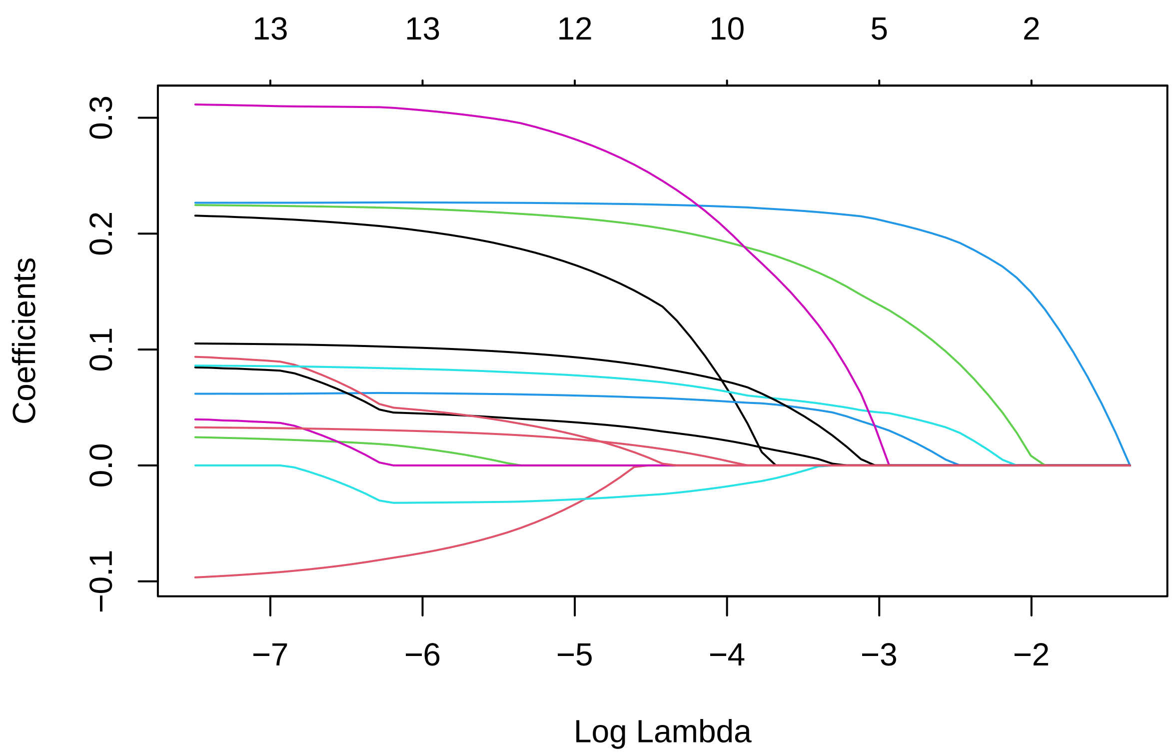
```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                          s0
## osiOS            0.20556523
## osOthers        -0.10497976
## osWindows        0.02435125
## X4gyes           0.07008456
## device_brandB   -0.02137162
## device_brandC    0.01937849
## device_brandD    0.06355957
## device_brandE    0.07648465
## screen_size      0.21126466
## rear_camera_mp   0.20754149
## front_camera_mp  0.10076652
## internal_memory  0.30609476
## ram              0.10141671
## days_used        0.03089745
```

```
predicted <- predict(ridge.best, newx = test.X, type = "response")
MSE.ridge <- mean((test.y - predicted)^2)
MSE.ridge
```

```
## [1] 0.06479813
```

lasso regression

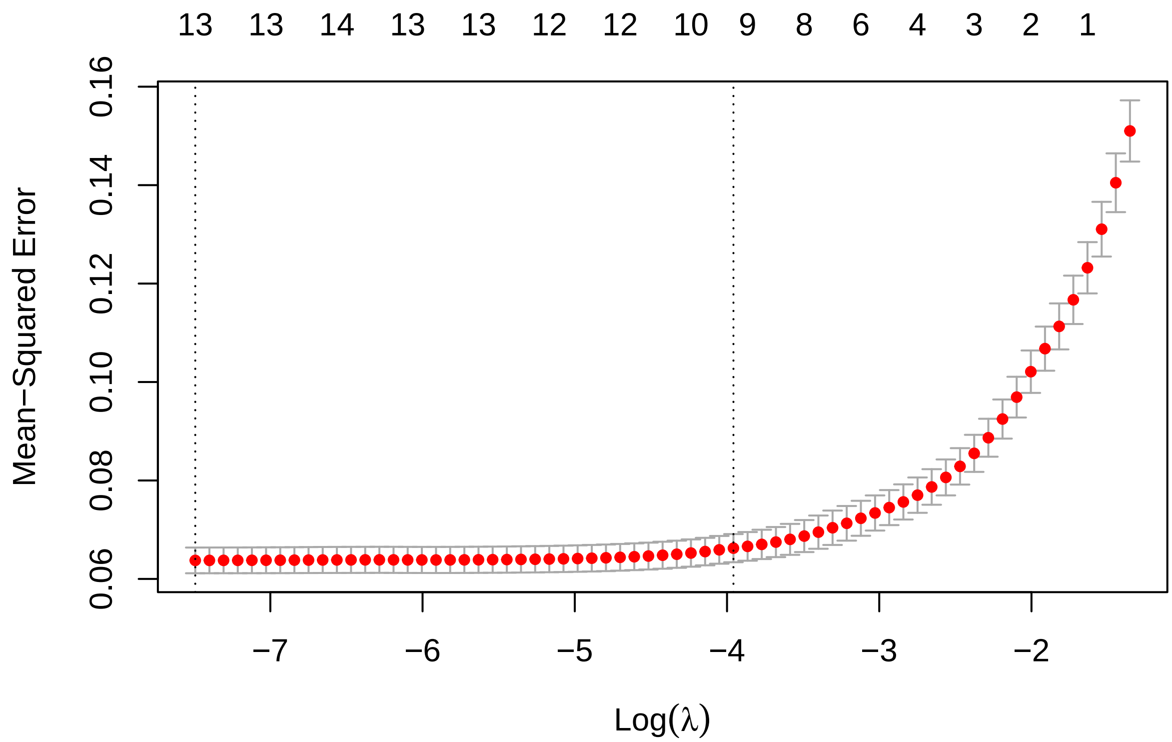
```
set.seed(5420)
par(mfrow = c(1,1))
lasso.model <- glmnet(train.X, train.y, alpha = 1)
plot(lasso.model, xvar = "lambda")
```




```
cv <- cv.glmnet(train.X, train.y, alpha=1)
best.lambda <- cv$lambda.min
best.lambda
```

```
## [1] 0.0005569095
```

```
plot(cv)
```




```r
lasso.best <- glmnet(train.X, train.y, alpha = 1, lambda = best.lambda)
lasso.best$beta
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                          s0
## osiOS             0.21533806
## osOthers         -0.09658768
## osWindows         0.02440836
## X4gyes            0.06182323
## device_brandB     .
## device_brandC     0.03992730
## device_brandD     0.08474524
## device_brandE     0.09389406
## screen_size       0.22466432
## rear_camera_mp    0.22662290
## front_camera_mp   0.08613265
## internal_memory   0.31134951
## ram               0.10520696
## days_used         0.03283846
```

```r
predicted <- predict(lasso.best, newx = test.X, type = "response")
MSE.lasso <- mean((test.y - predicted)^2)
MSE.lasso
```

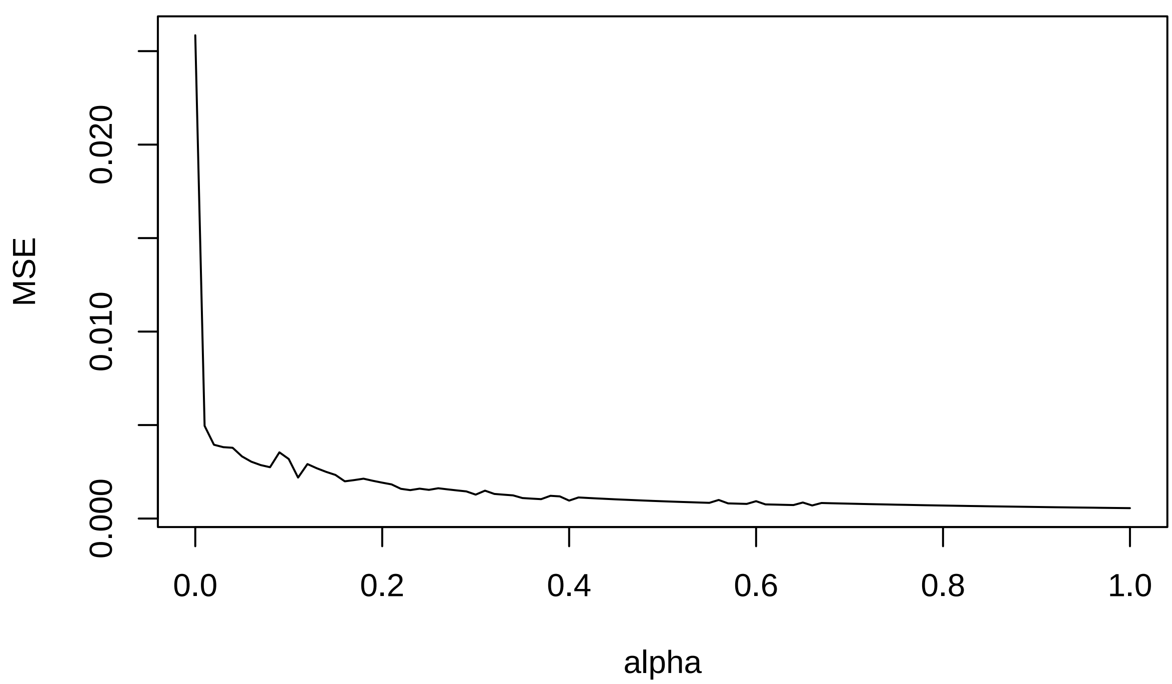
```
## [1] 0.06498742
```

elastic net regression

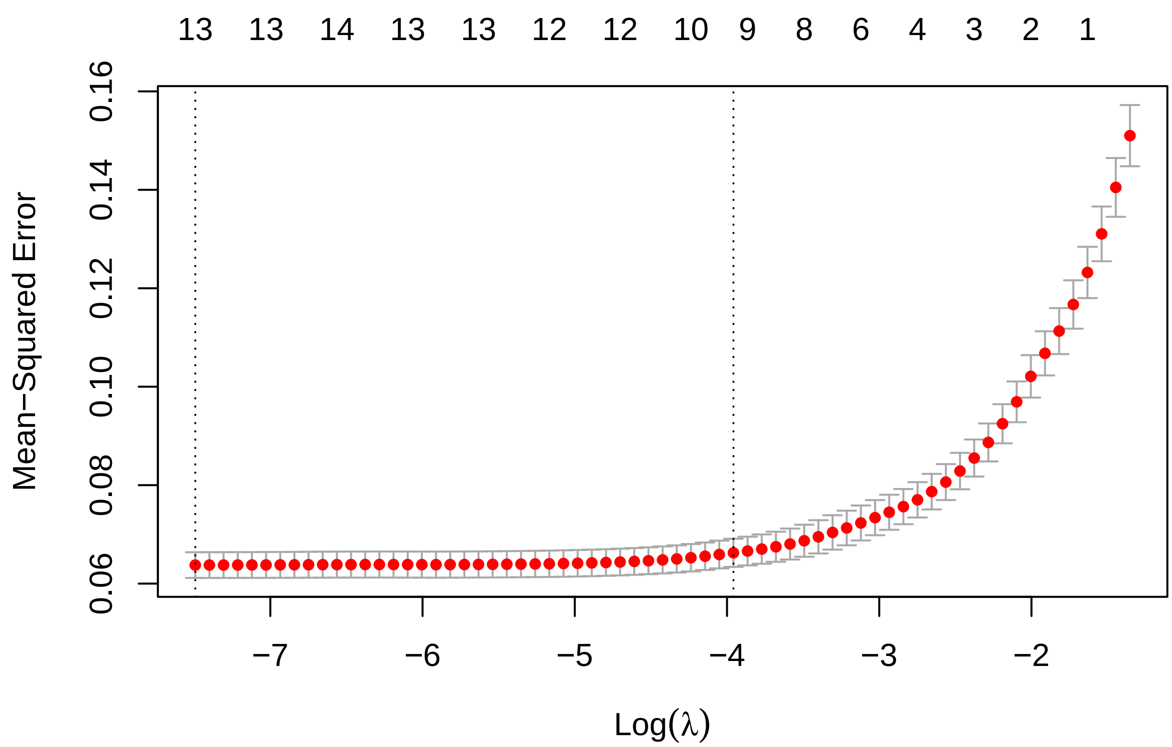
```
alpha.seq <- seq(0,1,0.01)
result.df <- matrix(0, nrow = length(alpha.seq), ncol = 2)

for (i in 1:length(alpha.seq)) {
  set.seed(5420)
  cv <- cv.glmnet(train.X, train.y, alpha = alpha.seq[i])
  lambda.index <- cv$index[1]
  mse <- cv$lambda[lambda.index]
  result.df[i,1] <- alpha.seq[i]
  result.df[i,2] <- mse
}
alpha.best.index <- which(result.df[,2]==min(result.df[,2]))
alpha.best <- result.df[alpha.best.index,1]
alpha.best
```

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

```
df <- as.data.frame(result.df)
plot(df,type="l", xlab = "alpha",
     ylab = "MSE")
```



```
set.seed(5420)
cv <- cv.glmnet(train.X, train.y, alpha = alpha.best)
plot(cv)
```




```
lambda.best <- cv$lambda.min
lambda.best
```

```
## [1] 0.0005569095
```

```
EN.best <- glmnet(train.X,train.y,alpha=alpha.best,lambda = lambda.best)
EN.best$beta
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                         s0
## osiOS            0.21533806
## osOthers        -0.09658768
## osWindows        0.02440836
## X4gyes           0.06182323
## device_brandB    .
## device_brandC    0.03992730
## device_brandD    0.08474524
## device_brandE    0.09389406
## screen_size      0.22466432
## rear_camera_mp   0.22662290
```

```
## front_camera_mp  0.08613265
## internal_memory  0.31134951
## ram              0.10520696
## days_used        0.03283846
```

```
predicted <- predict(EN.best, newx = test.X, type = "response")
MSE.EN <- mean((test.y - predicted)^2)
MSE.EN
```

```
## [1] 0.06498742
```

NN

```r
library(neuralnet)
```

```
##
## Attaching package: 'neuralnet'

## The following object is masked from 'package:dplyr':
##
##     compute
```

```r
# Build the neural network model
# Train the neural network
train.DF <- model.matrix(~., data = train.df)
train.DF <- model.matrix(~., data = train.df)[,-1]
nn.model <- neuralnet(normalized_used_price~., data=train.DF, hidden=c(5,3))
# plot(nn.model)

# Make predictions
test.X <- as.data.frame(test.X)
predicted <- compute(nn.model, test.X)$net.result

# Evaluate the model
MSE.NN <- mean((predicted - test.y)^2)
MSE.NN
```

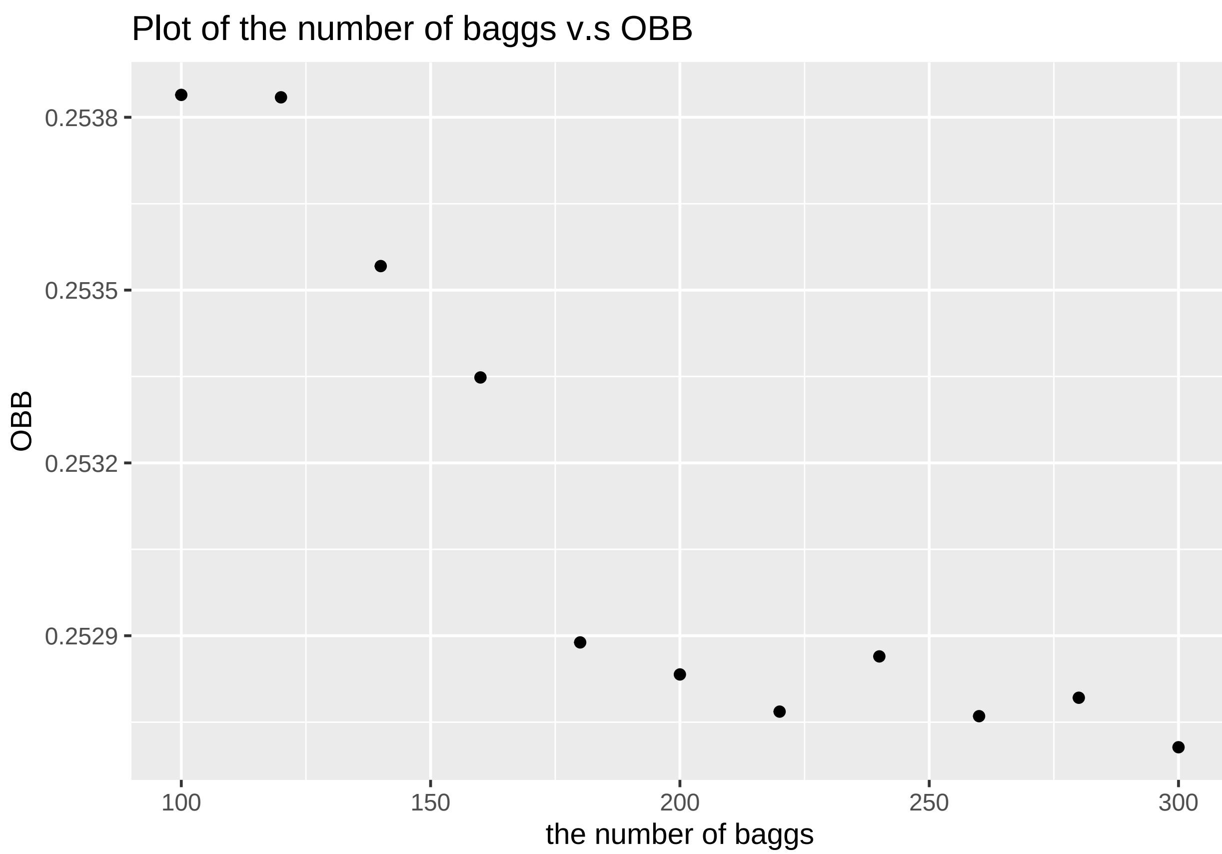
```
## [1] 0.06498484
```

VI. Tree Based Method

Bagging

```r
bagg.num <- seq(100,300,20)
MSE.bagg.df <- data.frame(numer_bag= bagg.num, OOB = rep(0,length(bagg.num)))
## loop to find best nbagg size
for (i in seq_len(length(bagg.num))) {

  set.seed(5420)
  bag <- bagging(
  formula = normalized_used_price~.,
  data=train.df,
  nbagg = MSE.bagg.df$numer_bag[i],
  coob = TRUE,
  control = rpart.control(minsplit = 20, cp = 0)
  )

  # Compute mean squared error of predictions
  MSE.bagg.df[i,2] <- bag$err
}
```

```r
ggplot(MSE.bagg.df, aes(x = numer_bag, y = OOB)) +
  geom_point() +
  labs(title = "Plot of the number of baggs v.s OBB",
       x = "the number of baggs", y = "OBB")
```



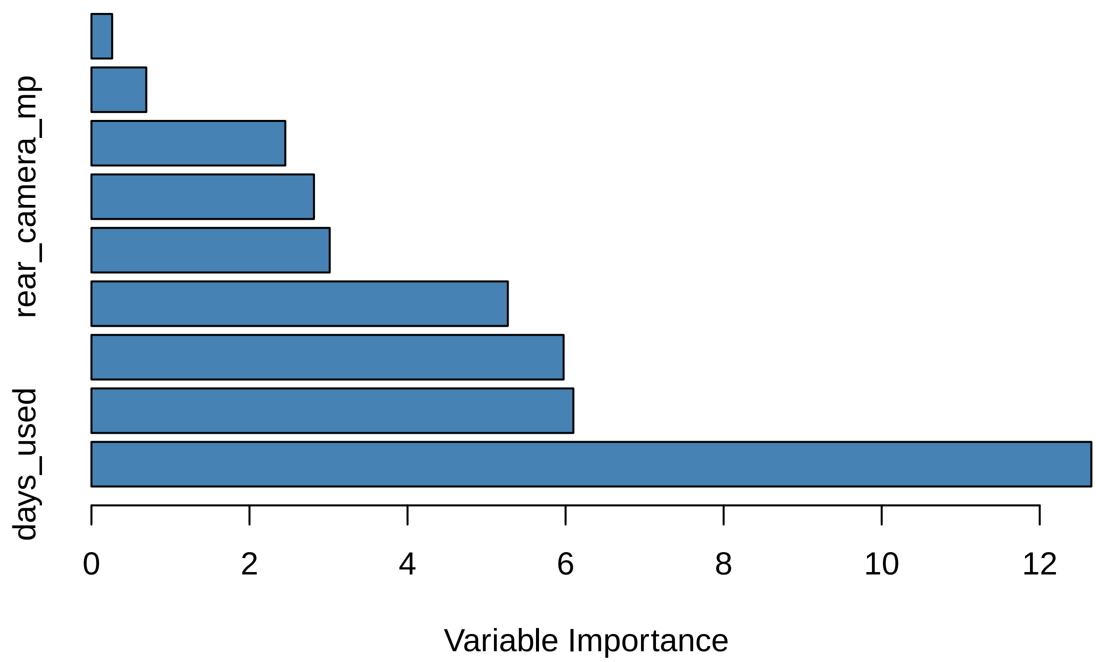
Error curve for bagging 100-300 deep, unpruned decision trees. The benefit of bagging is optimized at 300 trees although the majority of error reduction occurred within the first 220 trees.

```
best.nbagg <- MSE.bagg.df[which(MSE.bagg.df$OOB==min(MSE.bagg.df$OOB)),1]

bag_model <- bagging(formula = normalized_used_price ~.,
                     data=train.df,
                     nbagg = best.nbagg,
                     coob = TRUE,
                     control = rpart.control(minsplit = 20, cp = 0))


# Make predictions on test data
predictions <- predict(bag_model, newdata = test.df)

# Compute mean squared error of predictions
MSE.bagg <- mean((test.y - predictions)^2)
MSE.bagg
```

```
## [1] 0.06307225
```

```
#calculate variable importance
VI <- data.frame(var=names(train.df[,-4]), imp= varImp(bag_model))

#sort variable importance descending
VI_plot <- VI[order(VI$Overall, decreasing=TRUE),]
```

```r
#visualize variable importance with horizontal bar plot
barplot(VI_plot$Overall,
        names.arg=rownames(VI_plot),
        horiz=TRUE,
        col='steelblue',
        xlab='Variable Importance')
```



We can, however, visualize the importance of the predictor variables by calculating the total reduction in RSS (residual sum of squares) due to the split over a given predictor, averaged over all of the trees. The larger the value, the more important the predictor.

We can see that days_used is the most importance predictor variable in the model while os is the least important.

Random Forest

```
n_features <- length(names(test.df))-1

hyper_grid <- expand.grid(
  mtry = c(1,2,3),
  min.node.size = c(10,20),
  replace = c(TRUE, FALSE),
  sample.fraction = c(.5, .63, .8, 1),
  OOB = NA
)

# execute full cartesian grid search
for(i in seq_len(nrow(hyper_grid))) {
  # fit model for ith hyperparameter combination
  fit <- ranger(
    formula         = normalized_used_price ~ .,
    data            = train.df,
    num.trees       = n_features * 10,
    mtry            = hyper_grid$mtry[i],
    min.node.size   = hyper_grid$min.node.size[i],
    replace         = hyper_grid$replace[i],
    sample.fraction = hyper_grid$sample.fraction[i],
    verbose         = FALSE,
    seed            = 5420,
    respect.unordered.factors = 'order',
  )
  # export OOB error
  hyper_grid$OOB[i] <- sqrt(fit$prediction.error)
}

hyper_grid.ordered <- hyper_grid %>%
  arrange(OOB) %>%
  head(10)
hyper_grid.ordered
```

```
##    mtry min.node.size replace sample.fraction       OOB
## 1     3            20    TRUE            0.50 0.2474806
## 2     3            20   FALSE            0.50 0.2482197
## 3     3            20    TRUE            1.00 0.2485383
## 4     3            10    TRUE            0.50 0.2485693
## 5     3            20    TRUE            0.63 0.2488084
## 6     3            10   FALSE            0.50 0.2488370
## 7     3            20   FALSE            0.63 0.2490910
## 8     3            20    TRUE            0.80 0.2491526
## 9     3            10    TRUE            0.63 0.2494640
## 10    3            10    TRUE            0.80 0.2495774
```

```r
## test the best model
best.RF.model <- ranger(
    formula         = normalized_used_price ~ .,
    data            = train.df,
    num.trees       = n_features * 10,
    mtry            = hyper_grid.ordered[1,1],
    min.node.size   = hyper_grid.ordered[1,2],
    replace         = hyper_grid.ordered[1,3],
    sample.fraction = hyper_grid.ordered[1,4],
    verbose         = FALSE,
    seed            = 5420,
    respect.unordered.factors = 'order',
  )

# Make predictions on test data
predictions <- predict(best.RF.model, data = test.df)$predictions

# Compute mean squared error of predictions
MSE.RF <- mean((test.y - predictions)^2)
MSE.RF
```

```
## [1] 0.05887985
```

VII. Result

```r
df.result <- data.frame(
  Method = c("Linear","stepwise(both)","Ridge","Lasso","Elastic Net",
             "Neural Network","Bagging","Random Forest"),
  MSE = c(mse.lm,mse.both,MSE.ridge,MSE.lasso,MSE.EN,MSE.NN,MSE.bagg,MSE.RF)
)

# Format the data frame as a table using kable
kable(df.result, format = "markdown")
```

Method	MSE
Linear	0.0649533
stepwise(both)	0.0650784
Ridge	0.0647981
Lasso	0.0649874
Elastic Net	0.0649874
Neural Network	0.0649848
Bagging	0.0630723
Random Forest	0.0588799