R Note for Statistics Learning and Computing

Daniel Cheung

2026-03-24

Table of contents

Preface

This is a R Markdown based on Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

Welcome to my R Note for Statistics Learning and Computing. Any mistake or suggestion please send to `DanielHCheung [at] outlook [dot] com`.

This document will be irregually updated. You're welcomed to provided any new ideas about it!

I personally recommand to use both RStudio and Positron for R development. RStudio is very traditional and familiar to those who get used to R programming for many years, and Positron is more vscode-like one. Positron connects both Python and R, so it's really interesting. But in this note, I will only share about the knowledge with R.

```
cat("Hello, World!")
```

```
Hello, World!
```

1 R Basics

1.1 Generate Random Sample

set random seed to get same result for repeating

```
set.seed(42)
```

a random vector size of 20, follows to Normal Distribution

```
rnorm(20)
```

```
 [1]  1.37095845 -0.56469817  0.36312841  0.63286260  0.40426832 -0.10612452
 [7]  1.51152200 -0.09465904  2.01842371 -0.06271410  1.30486965  2.28664539
[13] -1.38886070 -0.27878877 -0.13332134  0.63595040 -0.28425292 -2.65645542
[19] -2.44046693  1.32011335
```

```
rnorm(20, mean = 10, sd = 2)
```

```
 [1]  9.386723  6.437383  9.656165 12.429349 13.790387  9.139062  9.485461
 [8]  6.473674 10.920195  8.720010 10.910900 11.409675 12.070207  8.782147
[15] 11.009910  6.565983  8.431082  8.298185  5.171585 10.072245
```

a random vector size of 10, follows to Uniform Distribution

```
runif(20)
```

```
 [1] 0.5816040025 0.1579052082 0.3590283059 0.6456318784 0.7758233626
 [6] 0.5636468416 0.2337033986 0.0899805163 0.0856120649 0.3052183695
[11] 0.6674265147 0.0002388966 0.2085699569 0.9330341273 0.9256447486
[16] 0.7340943010 0.3330719834 0.5150633298 0.7439746463 0.6191592400
```

```r
runif(20, min = 0, max = 1)
```

```
 [1] 0.626245345 0.217157698 0.216567311 0.388945029 0.942455692 0.962608014
 [7] 0.739855279 0.733245906 0.535761290 0.002272966 0.608937453 0.836801559
[13] 0.751522563 0.452731573 0.535789994 0.537376695 0.001380844 0.355665954
[19] 0.612133090 0.828942131
```

```r
runif(20, 0, 1)
```

```
 [1] 0.3567220 0.4106351 0.5734759 0.5896783 0.7196573 0.3949730 0.9192039
 [8] 0.9625703 0.2335235 0.7244976 0.9036345 0.6034741 0.6315073 0.9373858
[15] 0.8504828 0.5798209 0.8214039 0.1137186 0.7645078 0.6236135
```

```r
prod(40:36)
```

```
[1] 78960960
```

```r
prod(5:1)/prod(40:36)
```

```
[1] 1.519738e-06
```

```r
1/choose(40,5)
```

```
[1] 1.519738e-06
```

```r
pvec <- seq(0,1,0.05)
pvec
```

```
 [1] 0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70
[16] 0.75 0.80 0.85 0.90 0.95 1.00
```

```r
x <- rnorm(60000)
quantile(x, pvec)
```

```
          0%           5%          10%          15%          20%          25%
-4.043276349 -1.656275635 -1.286941232 -1.045470141 -0.852443729 -0.684532015
         30%          35%          40%          45%          50%          55%
-0.529470949 -0.387580521 -0.257069669 -0.124515346  0.002733696  0.126948433
         60%          65%          70%          75%          80%          85%
 0.251286891  0.381425011  0.524315026  0.675289348  0.843406262  1.033822369
         90%          95%         100%
 1.282235340  1.649455790  4.328091274
```

```
quantile(x)
```

```
          0%          25%          50%          75%         100%
-4.043276349 -0.684532015  0.002733696  0.675289348  4.328091274
```

```
x.1 <- rnorm(20)
x.2 <- rnorm(20) + runif(20)
x.3 <- x.1 * 20 + x.2 * 4 - rnorm(20)

df <- data.frame(x1 = x.1, x2 = x.2, x3 = x.3)
```

1.2 Matrix

```
A <- cor(df)
A
```

```
          x1        x2        x3
x1 1.0000000 0.1598481 0.9841209
x2 0.1598481 1.0000000 0.3275044
x3 0.9841209 0.3275044 1.0000000
```

```
svd(A)
```

```
$d
[1] 2.0934081396 0.9056768657 0.0009149947

$u
           [,1]       [,2]       [,3]
[1,] -0.6617282  0.3025436 -0.6859906
```

```
[2,] -0.3022505 -0.9449705 -0.1252015
[3,] -0.6861198  0.1244917  0.7167576

$v
           [,1]       [,2]       [,3]
[1,] -0.6617282  0.3025436 -0.6859906
[2,] -0.3022505 -0.9449705 -0.1252015
[3,] -0.6861198  0.1244917  0.7167576
```

```
iA <- solve(A)

cA <- chol(A)

t(cA)  %*% cA
```

```
          x1        x2        x3
x1 1.0000000 0.1598481 0.9841209
x2 0.1598481 1.0000000 0.3275044
x3 0.9841209 0.3275044 1.0000000
```

```
solve(cA) %*% t(solve(cA))
```

```
           x1        x2         x3
x1  514.61173  93.64606 -537.10967
x2   93.64606  18.16131  -98.10696
x3 -537.10967 -98.10696  561.71133
```

```
solve(A)
```

```
           x1        x2         x3
x1  514.61173  93.64606 -537.10967
x2   93.64606  18.16131  -98.10696
x3 -537.10967 -98.10696  561.71133
```

1.3 string

```
astr <- as.character(12345)
bstr <- "NULL"
cstr <- "NOT LOWER, not upper"
c(astr, bstr, cstr)
```

```
[1] "12345"                "NULL"                 "NOT LOWER, not upper"
```

1.3.1 length of a string

```
nchar(astr)
```

```
[1] 5
```

```
c(toupper(cstr), tolower(cstr))
```

```
[1] "NOT LOWER, NOT UPPER" "not lower, not upper"
```

1.3.2 substring

both `substr` and `substring` can extract substring, and `substring` defined with default value for the `last = 1000000L`. Also, `substring` can exact multiple substrings while `substr` can't.

```
substr("HELLO", 2,3)
```

```
[1] "EL"
```

```
substr("HELLO", c(1,2,3),c(3,4,5))
```

```
[1] "HEL"
```

```
substring("HELLO", c(1,2,3),c(3,4,5))
```

```
[1] "HEL" "ELL" "LLO"
```

```
a = "12345678"
substr(a,2,2)
```

```
[1] "2"
```

```r
substr(a,2,4)
```

```
[1] "234"
```

```r
# substr(a,2)
# Error in substr(a, 2) : argument "stop" is missing, with no default

substring(a,2)
```

```
[1] "2345678"
```

```r
substr(a,3,4) <- "Toyota"
a
```

```
[1] "12To5678"
```

```r
substring(a,3) <- "Toyata"
a
```

```
[1] "12Toyata"
```

1.3.3 combine strings

```r
a = "12345"

b = "6789"

paste(a,b)
```

```
[1] "12345 6789"
```

```r
paste(a,b,sep="")
```

```
[1] "123456789"
```

```r
paste(a,b,sep="-")
```

```
[1] "12345-6789"
```

```r
paste(a,b,sep = "toyota")
```

```
[1] "12345toyota6789"
```

```r
a <- c('Taipei', "Tokyo", 'Singapore', 'Hong Kong', "Johor Bahru", "Kuala Lumpur")
paste(a)
```

```
[1] "Taipei"       "Tokyo"        "Singapore"    "Hong Kong"    "Johor Bahru"
[6] "Kuala Lumpur"
```

```r
paste(a, "?", sep = "")
```

```
[1] "Taipei?"       "Tokyo?"        "Singapore?"    "Hong Kong?"
[5] "Johor Bahru?"  "Kuala Lumpur?"
```

```r
paste(a, "is where", sep = " ")
```

```
[1] "Taipei is where"       "Tokyo is where"        "Singapore is where"
[4] "Hong Kong is where"    "Johor Bahru is where"  "Kuala Lumpur is where"
```

```r
paste(a, "is where", sep = " ", collapse = ", and ")
```

```
[1] "Taipei is where, and Tokyo is where, and Singapore is where, and Hong Kong is where, an
```

```r
paste(c("a","b"), c("1","2","3"))
```

```
[1] "a 1" "b 2" "a 3"
```

```r
paste(c("a","b","c","d"), c("1","2","3"))
```

```
[1] "a 1" "b 2" "c 3" "d 1"
```

1.3.4 split strings

```
strsplit(",,a,bb,", split = ",")
```

```
[[1]]
[1] ""   ""   "a"  "bb"
```

```
strsplit(",,a,,", split = ",")
```

```
[[1]]
[1] ""  ""  "a" ""
```

```
strsplit(",,a,", split = ",")
```

```
[[1]]
[1] ""  ""  "a"
```

1.3.5 replace

replace by vector

```
s <- "ccccaaaassssaaa"
chartr("s", "A", s)
```

```
[1] "ccccaaaaAAAAaaa"
```

```
chartr("sc", "AI", s)
```

```
[1] "IIIIaaaaAAAAaaa"
```

```
sub("a", "b", s)
```

```
[1] "ccccbaaassssaaa"
```

```
gsub("a", "b", s)
```

```
[1] "ccccbbbbssssbbb"
```

Note that `chartr` is vector-behavior function, which changes char-by-char; as `gsub` replaces the whole target substring globally, `sub` only changes the first occurance. ### Regular Expression try: type `?grep` in your console. Then see the help page "Pattern Matching and Replacement"

```
# grep()
```

1.4 todo

```
c("Hello")
```

```
[1] "Hello"
```

```
c("Hello")
```

```
[1] "Hello"
```

```
c("Hello")
```

```
[1] "Hello"
```

```
c("Hello")
```

```
[1] "Hello"
```

```
c("Hello")
```

```
[1] "Hello"
```

```
c("Hello")
```

```
[1] "Hello"
```

```r
c("Hello")
```

```
[1] "Hello"
```

```r
c("Hello")
```

```
[1] "Hello"
```

```r
c("Hello")
```

```
[1] "Hello"
```

```r
c("Hello")
```

```
[1] "Hello"
```

```r
c("Hello")
```

```
[1] "Hello"
```

```r
c("Hello")
```

```
[1] "Hello"
```

```r
c("Hello")
```

```
[1] "Hello"
```

```r
c("Hello")
```

```
[1] "Hello"
```

```r
c("Hello")
```

```
[1] "Hello"
```

```r
c("Hello")
```

```
[1] "Hello"
```

1.5 tricks

1.5.1 deal with potential exception

pass a string of the variable name to check if it's defined, return NULL if not found.

```
x0 <- "demo"
get0("x0")
```

```
[1] "demo"
```

```
get0("xa_nondefined")
```

```
NULL
```

1.5.2 RMarkdown features

When you write {r} after three ‘ signs you can give some features:

You can leave {r} by default and get very annoying outputs

```
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
library(tidyverse)
```

```
Warning: package 'ggplot2' was built under R version 4.5.2
```

```
Warning: package 'readr' was built under R version 4.5.2

-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v forcats   1.0.1     v readr     2.1.6
v ggplot2   4.0.1     v stringr   1.5.2
v lubridate 1.9.4     v tibble    3.3.0
v purrr     1.1.0     v tidyr     1.3.1

-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

you can write `{r, warning=FALSE, message=FALSE}` means no suppresses warnings and no suppresses messages.

```r
library(dplyr)
library(tidyverse)
```

`echo = FALSE` means to hide the code trunk

`include=FALSE` means to hide the code trunk and the result

`eval = FALSE` show but no executing

`results = 'hide'` will hide the code

`error = TRUE` continue to knit even there's error

1.5.3 package management

install if not exists then load the package.

```r
if (!requireNamespace("tidyverse")) install.packages('tidyverse')
library(tidyverse)
```

This can also be made with package `pacman`.

```r
library(pacman)
pacman::p_load(mblm)
```

```r
c("Hello")
```

```
[1] "Hello"
```

2 R Programming for Data Analysis

```
library(dplyr)
```

2.1 Data Cleaning

```
iris <- iris %>%
  mutate(Petal.Ratio = Petal.Length / Petal.Width)
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Petal.Ratio
1          5.1         3.5          1.4         0.2  setosa        7.00
2          4.9         3.0          1.4         0.2  setosa        7.00
3          4.7         3.2          1.3         0.2  setosa        6.50
4          4.6         3.1          1.5         0.2  setosa        7.50
5          5.0         3.6          1.4         0.2  setosa        7.00
6          5.4         3.9          1.7         0.4  setosa        4.25
```

```
iris %>%
  group_by(Species) %>%
  summarise(mean_Petal_Ratio = mean(Petal.Ratio))
```

```
# A tibble: 3 x 2
  Species    mean_Petal_Ratio
  <fct>                 <dbl>
1 setosa                 6.91
2 versicolor             3.24
3 virginica              2.78
```

2.2 Split Dataset

There's how you can get random sample.

```
iris %>% sample_n(10)
```

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species Petal.Ratio
1           7.9         3.8          6.4         2.0  virginica    3.200000
2           6.4         3.2          5.3         2.3  virginica    2.304348
3           5.7         2.6          3.5         1.0 versicolor    3.500000
4           5.4         3.7          1.5         0.2     setosa    7.500000
5           5.5         2.4          3.7         1.0 versicolor    3.700000
6           6.2         2.2          4.5         1.5 versicolor    3.000000
7           6.6         2.9          4.6         1.3 versicolor    3.538462
8           6.3         2.9          5.6         1.8  virginica    3.111111
9           5.1         3.8          1.6         0.2     setosa    8.000000
10          6.0         3.0          4.8         1.8  virginica    2.666667
```

```
iris[sample(nrow(iris), 10), ]
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species Petal.Ratio
109          6.7         2.5          5.8         1.8  virginica    3.222222
102          5.8         2.7          5.1         1.9  virginica    2.684211
148          6.5         3.0          5.2         2.0  virginica    2.600000
84           6.0         2.7          5.1         1.6 versicolor    3.187500
71           5.9         3.2          4.8         1.8 versicolor    2.666667
63           6.0         2.2          4.0         1.0 versicolor    4.000000
25           4.8         3.4          1.9         0.2     setosa    9.500000
150          5.9         3.0          5.1         1.8  virginica    2.833333
137          6.3         3.4          5.6         2.4  virginica    2.333333
59           6.6         2.9          4.6         1.3 versicolor    3.538462
```

Please use `y` (target variable) as the variable into `createDataPartition`, rather than any other variables. p is the portition of training set.

```
library(caret)
```

```
training.samples = iris$Species %>%
  createDataPartition(p = 0.70, list = FALSE)

traindf = iris[training.samples,]
testdf = iris[-training.samples,]
```

3 Datasets to Practice

3.1 Environment Setup

We use the tidyverse suite for data manipulation and visualization.

```
library(tidyverse)
library(scales) # For professional axis labels
```

3.1.1 The Tidyverse Collection (Modern Data Science)

The `tidyverse` is a suite of packages designed for a seamless data workflow. Its datasets are primarily used to master **Data Wrangling**, **Tidy Data principles**, and **Advanced Visualization**.

Dataset	Package	Description	Primary Analysis Use
`mpg`	`ggplot2`	Fuel economy data for 38 car models (1999–2008).	**Exploratory Data Analysis (EDA)**: Practicing scatterplots, smoothing lines, and correlation between engine size and efficiency.
`diamonds`	`ggplot2`	Prices and attributes of ~54,000 diamonds.	**Predictive Modeling**: Ideal for training linear regression models or random forests due to its large sample size.
`starwars`	`dplyr`	Characteristics of characters (height, mass, species).	**Data Cleaning**: Best for practicing how to handle `NA` (missing values) and complex list-columns.

Dataset	Package	Description	Primary Analysis Use
`table1-5`	`tidyr`	WHO Tuberculosis cases in different formats.	**Data Reshaping**: Specifically used to learn `pivot_longer()` and `pivot_wider()` to reach a "tidy" state.
`storms`	`dplyr`	Wind and pressure data for tropical cyclones.	**Time Series & Mapping**: Visualizing storm tracks over time and geographical coordinates.

3.1.2 ISwR (Introductory Statistics with R)

The `ISwR` package (by Peter Dalgaard) focuses on **Biostatistics** and **Inference**. These datasets are smaller but are "textbook cases" for specific mathematical tests.

Dataset	Description	Primary Analysis Use
`thuesen`	Ventricular shortening velocity and blood glucose in diabetics.	**Simple Linear Regression**: Testing the linear relationship between two continuous clinical variables.
`energy`	Energy expenditure in lean vs. obese women.	**Hypothesis Testing**: Perfect for practicing the Two-sample t-test and checking for equal variance.
`juul`	IGF-I (Growth Hormone) levels across different ages.	**ANOVA & Polynomial Regression**: Analyzing how a biological marker changes non-linearly across puberty stages.
`melanom`	Survival data of patients with malignant melanoma.	**Survival Analysis**: The gold standard for practicing Kaplan-Meier survival curves and Cox Proportional Hazards.

Dataset	Description	Primary Analysis Use
`stroke`	Clinical outcomes of stroke patients in Estonia.	**Logistic Regression**: Predicting binary outcomes (e.g., dead/alive) based on age, sex, and clinical history.

3.2 Loading the dataset

```
data("diamonds")
```

4 Hypothesis Test

4.1 mean inference (z,t-test)

4.1.1 Z-test

Note that there's no built-in Z-test in R. Use third party package `BSDA` (the code above written by Gemini)

```
# install.packages("BSDA")
library(BSDA)
```

```
Loading required package: lattice
```

```
Attaching package: 'BSDA'
```

```
The following object is masked from 'package:datasets':

    Orange
```

```
x <- c(1,1,1,1,1,4,3,2,1,1,1,1,1,11,1,1)
z.test(x, sigma.x = 0.5, conf.level = 0.95)
```

```

    One-sample z-Test

data:  x
z = 16, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 1.755005 2.244995
sample estimates:
mean of x
        2
```

You can also make implement from the formular: reference this passage: https://cran.r-project.org/web/packages/distributions3/vignettes/one-sample-z-test.html

4.1.2 t-test

```
x <- c(1,1,1,1,1,4,3,2,1,1,1,1,1,11,1,1)
y <- c(1,1,1,1,1,4,3,2,1,1,1,1,1,11,1,1)
t.test(x)
```

```
    One Sample t-test

data:  x
t = 3.1298, df = 15, p-value = 0.006884
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.6379832 3.3620168
sample estimates:
mean of x
        2
```

```
t.test(x,y)
```

```
    Welch Two Sample t-test

data:  x and y
t = 0, df = 30, p-value = 1
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.845594  1.845594
sample estimates:
mean of x mean of y
        2         2
```

```
t.test(x,y, mu = 2, alter = "two.sided", var.equal = T)
```

```
    Two Sample t-test
```

```
data:  x and y
t = -2.2131, df = 30, p-value = 0.03464
alternative hypothesis: true difference in means is not equal to 2
95 percent confidence interval:
 -1.845594  1.845594
sample estimates:
mean of x mean of y
        2         2
```

4.2 variance inference

Try to type `?var.test` to get the help document.

```
var.test(x,y)
```

```
    F test to compare two variances

data:  x and y
F = 1, num df = 15, denom df = 15, p-value = 1
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.3493947 2.8620925
sample estimates:
ratio of variances
                 1
```

4.3 goodness of fitness test

Test whether a sample data fit a distribution or not.

4.3.1 Normal Test

4.4 correlation

```r
x <- c(1,1,1,1,1,4,3,2,1,1,1,1,1,11,1,1)
y <- c(1,1,1,1,1,4,3,2,1,1,1,1,1,11,1,1)
cor.test(x, y)
```

```
    Pearson's product-moment correlation

data:  x and y
t = Inf, df = 14, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 1 1
sample estimates:
cor
  1
```

The default method, Pearson linear correlation is widely used.

4.5 median inference (non-para test)

Although it's non-parametric test based on median, the function also require to pass parameter as `mu`.

```r
x <- c(1,1,1,1,1,4,3,2,1,1,1,1,1,11,1,1)
y <- c(1,1,1,1,1,4,3,2,1,1,1,1,1,11,1,1)
wilcox.test(x)
```

```
Warning in wilcox.test.default(x): cannot compute exact p-value with ties
```

```
    Wilcoxon signed rank test with continuity correction

data:  x
V = 136, p-value = 0.0002424
alternative hypothesis: true location is not equal to 0
```

```r
wilcox.test(x,y)
```

```
Warning in wilcox.test.default(x, y): cannot compute exact p-value with ties
```

```
    Wilcoxon rank sum test with continuity correction

data:  x and y
W = 128, p-value = 1
alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(x,y, mu = 2, alter = "two.sided", var.equal = T)
```

```
Warning in wilcox.test.default(x, y, mu = 2, alter = "two.sided", var.equal =
T): cannot compute exact p-value with ties
```

```
    Wilcoxon rank sum test with continuity correction

data:  x and y
W = 33.5, p-value = 0.0001599
alternative hypothesis: true location shift is not equal to 2
```

5 Linear Regression

Coverage: Linear Regression, General Linear Regression, Lasso, Ridge, Elastic Net Regression

5.1 Theoritical Knowledge

The multivariate linear Regression $y = X\beta$ with loss function $Q = \|(y - X\beta)\|^2 = (y - X\beta)^T(y - X\beta)$

$$\frac{\partial Q}{\partial \beta} = -2X^T y + 2X^T X\beta = 0 \rightarrow \hat{\beta} = (X^T X)^{-1} X^T y$$

H is called Hat Matrix where: $\hat{y} = X\hat{\beta} = X(X^T X)^{-1} X^T y = Hy$

Normal Equation: $(X^T X)\hat{\beta} = X^T y \rightarrow X^T(y - X\hat{\beta}) = 0$

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

$$\text{Adjusted-}R^2 = \frac{SSR/k}{SST/(n-1)} = 1 - \frac{SSE/(n-k-1)}{SST/(n-1)}$$

Notice that here SSE means Sum Square Error, and SSR means Sum Square Regression. Sometimes you might see ESS which is Explained Sum Square which equals to SSR here, and RSS is Residual Sum Square.

In R, use `lm()` to fit a linear regression model.

5.2 Example: `diamonds` dataset

```
library(tidyverse)
library(caret)
library(car)
library(glmnet)
```

```
data(diamonds)
head(diamonds)
```

```
# A tibble: 6 x 10
  carat cut       color clarity depth table price     x     y     z
  <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

```
sum(is.na(diamonds))
```

```
[1] 0
```

```
str(diamonds)
```

```
tibble [53,940 x 10] (S3: tbl_df/tbl/data.frame)
 $ carat  : num [1:53940] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
 $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
 $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
 $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
 $ depth  : num [1:53940] 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
 $ table  : num [1:53940] 55 61 65 58 58 57 57 55 61 61 ...
 $ price  : int [1:53940] 326 326 327 334 335 336 336 337 337 338 ...
 $ x      : num [1:53940] 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
 $ y      : num [1:53940] 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
 $ z      : num [1:53940] 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

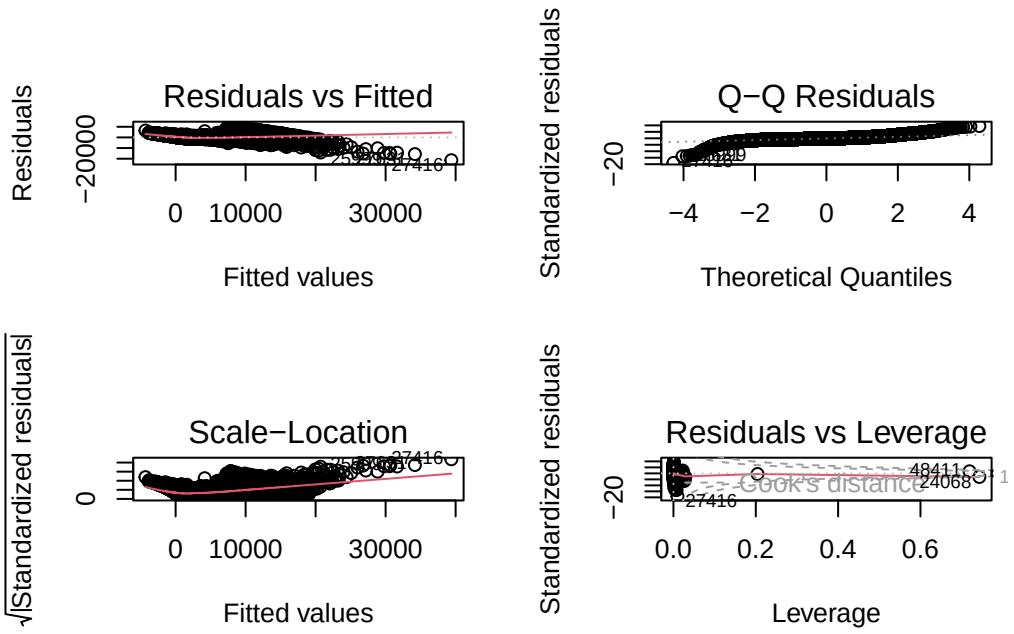
5.2.1 Without preprocessing

```
fit1 <- lm(price~., diamonds)
summary(fit1)
```

```
Call:
lm(formula = price ~ ., data = diamonds)

Residuals:
     Min       1Q   Median       3Q      Max
-21376.0   -592.4   -183.5    376.4  10694.2

Coefficients:
             Estimate Std. Error  t value Pr(>|t|)
(Intercept)  5753.762    396.630   14.507  < 2e-16 ***
carat       11256.978     48.628  231.494  < 2e-16 ***
cut.L         584.457     22.478   26.001  < 2e-16 ***
cut.Q        -301.908     17.994  -16.778  < 2e-16 ***
cut.C         148.035     15.483    9.561  < 2e-16 ***
cut^4         -20.794     12.377   -1.680  0.09294 .
color.L     -1952.160     17.342 -112.570  < 2e-16 ***
color.Q      -672.054     15.777  -42.597  < 2e-16 ***
color.C      -165.283     14.725  -11.225  < 2e-16 ***
color^4        38.195     13.527    2.824  0.00475 **
color^5       -95.793     12.776   -7.498 6.59e-14 ***
color^6       -48.466     11.614   -4.173 3.01e-05 ***
clarity.L    4097.431     30.259  135.414  < 2e-16 ***
clarity.Q   -1925.004     28.227  -68.197  < 2e-16 ***
clarity.C     982.205     24.152   40.668  < 2e-16 ***
clarity^4    -364.918     19.285  -18.922  < 2e-16 ***
clarity^5     233.563     15.752   14.828  < 2e-16 ***
clarity^6       6.883     13.715    0.502  0.61575
clarity^7      90.640     12.103    7.489 7.06e-14 ***
depth         -63.806      4.535  -14.071  < 2e-16 ***
table         -26.474      2.912   -9.092  < 2e-16 ***
x           -1008.261     32.898  -30.648  < 2e-16 ***
y               9.609     19.333    0.497  0.61918
z             -50.119     33.486   -1.497  0.13448
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 1130 on 53916 degrees of freedom
Multiple R-squared:  0.9198,    Adjusted R-squared:  0.9198
F-statistic: 2.688e+04 on 23 and 53916 DF,  p-value: < 2.2e-16
```

```r
par(mfrow = c(2, 2))
plot(fit1)
```




```r
vif(fit1)
```

```
            GVIF Df GVIF^(1/(2*Df))
carat   22.439582  1        4.737044
cut      1.945645  4        1.086758
color    1.178326  6        1.013768
clarity  1.347474  7        1.021531
depth    1.782401  1        1.335066
table    1.787765  1        1.337073
x       57.518327  1        7.584084
y       20.592160  1        4.537859
z       23.585582  1        4.856499
```

```r
fit1a <- lm(price~.-x-z-y, diamonds)
vif(fit1a)
```

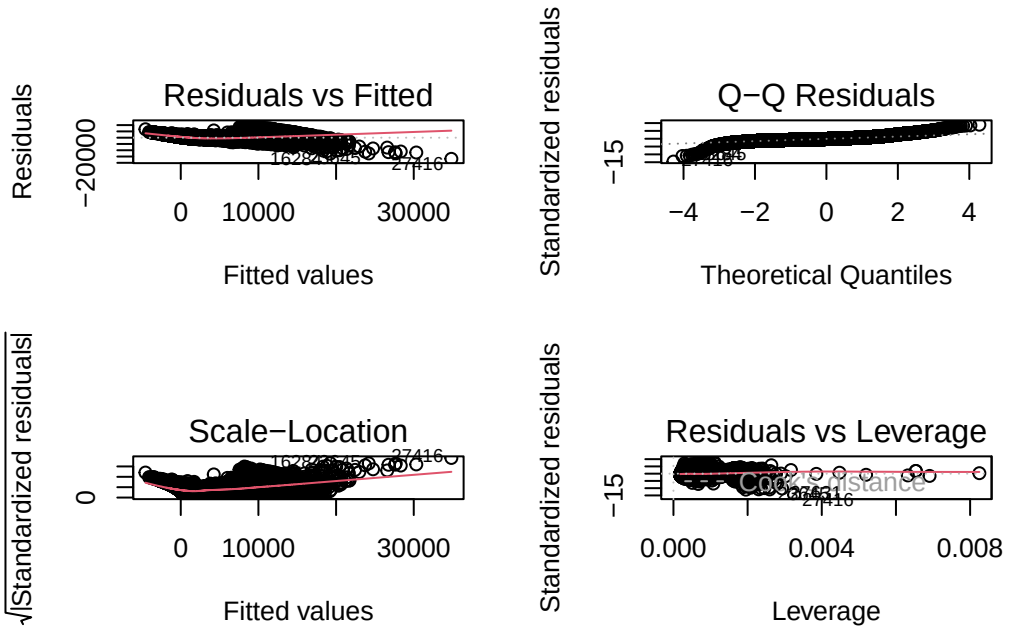
```
            GVIF Df GVIF^(1/(2*Df))
carat   1.323098  1        1.150260
cut     1.926004  4        1.085381
color   1.169317  6        1.013120
clarity 1.303877  7        1.019134
depth   1.378257  1        1.173992
table   1.786714  1        1.336680
```

```r
summary(fit1a)
```

```
Call:
lm(formula = price ~ . - x - z - y, data = diamonds)

Residuals:
     Min       1Q   Median       3Q      Max
-16828.8   -678.7   -199.4    464.6  10341.2

Coefficients:
             Estimate Std. Error  t value Pr(>|t|)
(Intercept)  -969.661    360.432   -2.690  0.00714 **
carat        8895.194     12.079  736.390  < 2e-16 ***
cut.L         615.613     22.985   26.784  < 2e-16 ***
cut.Q        -326.638     18.390  -17.762  < 2e-16 ***
cut.C         156.333     15.814    9.886  < 2e-16 ***
cut^4         -15.975     12.648   -1.263  0.20657
color.L     -1908.010     17.718 -107.689  < 2e-16 ***
color.Q      -626.087     16.112  -38.858  < 2e-16 ***
color.C      -172.056     15.063  -11.423  < 2e-16 ***
color^4        20.319     13.833    1.469  0.14187
color^5       -85.245     13.068   -6.523 6.95e-11 ***
color^6       -50.085     11.881   -4.216 2.50e-05 ***
clarity.L    4206.854     30.867  136.290  < 2e-16 ***
clarity.Q   -1831.804     28.811  -63.580  < 2e-16 ***
clarity.C     919.725     24.672   37.278  < 2e-16 ***
clarity^4    -361.609     19.728  -18.330  < 2e-16 ***
clarity^5     213.910     16.108   13.280  < 2e-16 ***
clarity^6       2.986     14.030    0.213  0.83148
```

```
clarity^7     110.147     12.375    8.901  < 2e-16 ***
depth         -21.024      4.079   -5.154 2.56e-07 ***
table         -24.803      2.978   -8.329  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1156 on 53919 degrees of freedom
Multiple R-squared:  0.9161,    Adjusted R-squared:  0.916
F-statistic: 2.942e+04 on 20 and 53919 DF,  p-value: < 2.2e-16
```

```r
par(mfrow = c(2, 2))
plot(fit1a)
```

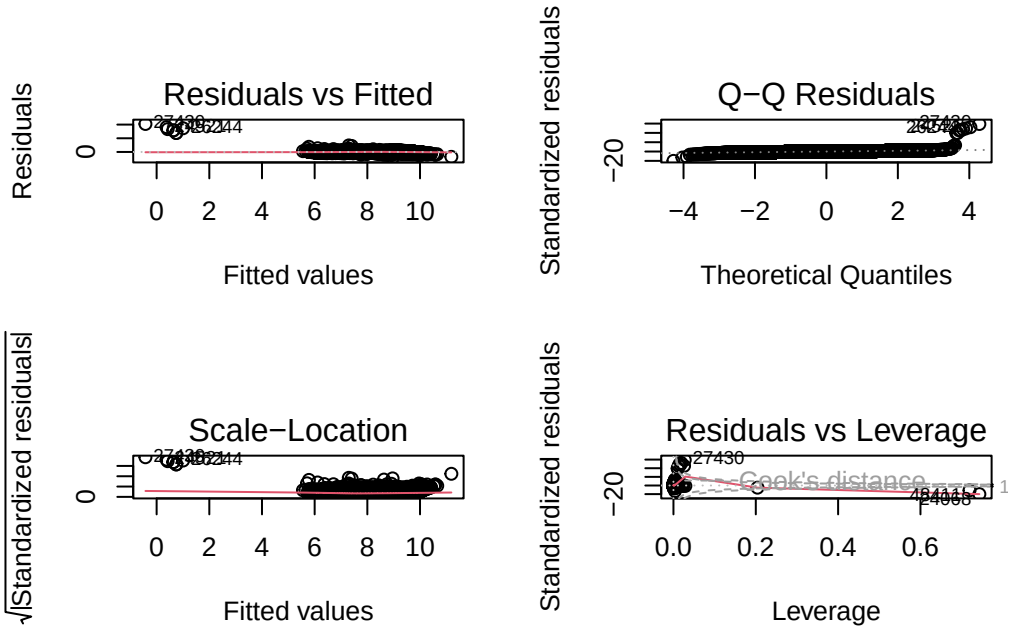



```r
fit2 <- lm(log(price)~., diamonds)
summary(fit2)
```

```
Call:
lm(formula = log(price) ~ ., data = diamonds)

Residuals:
    Min      1Q  Median      3Q     Max
-2.2544 -0.0911  0.0015  0.0902 10.2241
```

```
Coefficients:
             Estimate Std. Error  t value Pr(>|t|)
(Intercept) -2.5530834  0.0616229  -41.431  < 2e-16 ***
carat       -0.6120540  0.0075551  -81.012  < 2e-16 ***
cut.L        0.1045411  0.0034923   29.934  < 2e-16 ***
cut.Q       -0.0370562  0.0027956  -13.255  < 2e-16 ***
cut.C        0.0371749  0.0024056   15.454  < 2e-16 ***
cut^4        0.0115784  0.0019229    6.021 1.74e-09 ***
color.L     -0.4526278  0.0026943 -167.993  < 2e-16 ***
color.Q     -0.1035542  0.0024512  -42.246  < 2e-16 ***
color.C     -0.0118522  0.0022878   -5.181 2.22e-07 ***
color^4      0.0188754  0.0021016    8.982  < 2e-16 ***
color^5     -0.0068063  0.0019850   -3.429 0.000606 ***
color^6      0.0021343  0.0018044    1.183 0.236892
clarity.L    0.8907095  0.0047012  189.466  < 2e-16 ***
clarity.Q   -0.2599052  0.0043856  -59.264  < 2e-16 ***
clarity.C    0.1435463  0.0037523   38.255  < 2e-16 ***
clarity^4   -0.0662975  0.0029962  -22.127  < 2e-16 ***
clarity^5    0.0285591  0.0024473   11.670  < 2e-16 ***
clarity^6   -0.0039048  0.0021309   -1.833 0.066881 .
clarity^7    0.0290474  0.0018805   15.447  < 2e-16 ***
depth        0.0521543  0.0007045   74.028  < 2e-16 ***
table        0.0089978  0.0004524   19.890  < 2e-16 ***
x            1.1646945  0.0051112  227.871  < 2e-16 ***
y            0.0325386  0.0030037   10.833  < 2e-16 ***
z            0.0427049  0.0052026    8.208 2.29e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1756 on 53916 degrees of freedom
Multiple R-squared:  0.9701,    Adjusted R-squared:  0.9701
F-statistic: 7.597e+04 on 23 and 53916 DF,  p-value: < 2.2e-16
```

```r
par(mfrow = c(2, 2))
plot(fit2)
```




```
vif(fit2)
```

```
             GVIF Df GVIF^(1/(2*Df))
carat   22.439582  1        4.737044
cut      1.945645  4        1.086758
color    1.178326  6        1.013768
clarity  1.347474  7        1.021531
depth    1.782401  1        1.335066
table    1.787765  1        1.337073
x       57.518327  1        7.584084
y       20.592160  1        4.537859
z       23.585582  1        4.856499
```

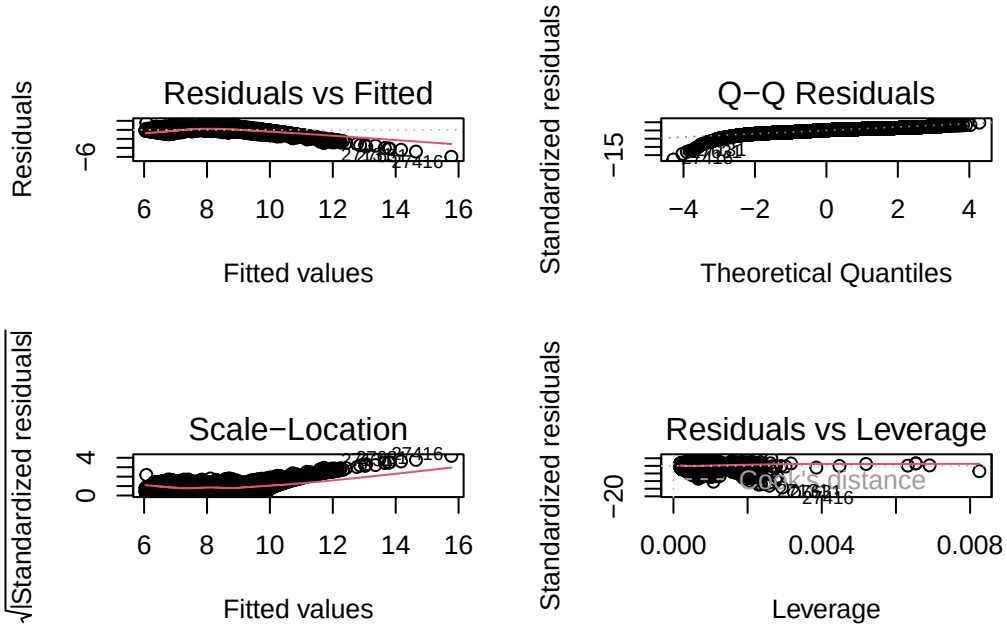
```
fit2a <- lm(log(price)~.-x-z-y, diamonds)
vif(fit2a)
```

```
            GVIF Df GVIF^(1/(2*Df))
carat   1.323098  1        1.150260
cut     1.926004  4        1.085381
color   1.169317  6        1.013120
clarity 1.303877  7        1.019134
depth   1.378257  1        1.173992
table   1.786714  1        1.336680
```

```r
summary(fit2a)
```

```
Call:
lm(formula = log(price) ~ . - x - z - y, data = diamonds)

Residuals:
    Min      1Q  Median      3Q     Max
-5.9717 -0.2181  0.0572  0.2479  1.6083

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.5193818  0.1055868  52.273  < 2e-16 ***
carat        2.1946870  0.0035386 620.211  < 2e-16 ***
cut.L        0.0682284  0.0067332  10.133  < 2e-16 ***
cut.Q       -0.0090647  0.0053872  -1.683   0.0925 .
cut.C        0.0291522  0.0046325   6.293 3.14e-10 ***
cut^4        0.0065684  0.0037051   1.773   0.0763 .
color.L     -0.5051100  0.0051903 -97.318  < 2e-16 ***
color.Q     -0.1582296  0.0047199 -33.524  < 2e-16 ***
color.C     -0.0038111  0.0044125  -0.864   0.3877
color^4      0.0400952  0.0040522   9.895  < 2e-16 ***
color^5     -0.0192706  0.0038283  -5.034 4.82e-07 ***
color^6      0.0041225  0.0034805   1.184   0.2362
clarity.L    0.7615612  0.0090423  84.222  < 2e-16 ***
clarity.Q   -0.3710533  0.0084401 -43.963  < 2e-16 ***
clarity.C    0.2181958  0.0072276  30.189  < 2e-16 ***
clarity^4   -0.0704719  0.0057792 -12.194  < 2e-16 ***
clarity^5    0.0521882  0.0047187  11.060  < 2e-16 ***
clarity^6    0.0006772  0.0041100   0.165   0.8691
clarity^7    0.0058294  0.0036253   1.608   0.1078
depth        0.0000854  0.0011950   0.071   0.9430
table        0.0068963  0.0008723   7.906 2.71e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3387 on 53919 degrees of freedom
Multiple R-squared:  0.8886,    Adjusted R-squared:  0.8886
F-statistic: 2.151e+04 on 20 and 53919 DF,  p-value: < 2.2e-16
```

```
par(mfrow = c(2, 2))
plot(fit2a)
```



The second model looks better.

5.2.2 With preprocessing

```
new_diamonds <- diamonds %>%
    predict(dummyVars(price ~ ., data = ., sep = "_", levelsOnly = FALSE), newdata = .) %>%
    as.data.frame() %>%
    mutate(logprice = log(diamonds$price))
colnames(new_diamonds) <- gsub("\\^", "_", colnames(new_diamonds))

set.seed(42)
training.idx <- new_diamonds$logprice %>%
    createDataPartition(p = 0.75, list = F)
trainset <- new_diamonds[training.idx, ]
testset <- new_diamonds[-training.idx, ]

prepproc <- preProcess(trainset, method = c("center", "scale"))
trainset.scaled <- predict(prepproc, trainset)
testset.scaled <- predict(prepproc, testset)
```

```r
mu <- prepproc$mean["logprice"]
sigma <- prepproc$std["logprice"]

train.x <- trainset.scaled %>% select(-logprice)
test.x <- testset.scaled %>% select(-logprice)
```

```r
fit3 <- lm(logprice~.-x-z-y, trainset.scaled)
summary(fit3)
```

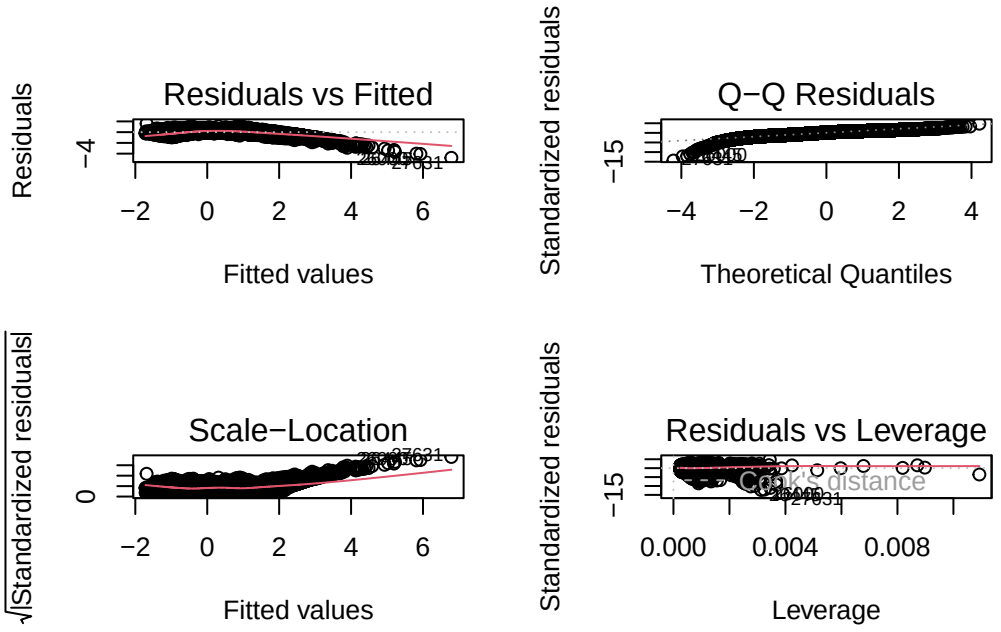
```
Call:
lm(formula = logprice ~ . - x - z - y, data = trainset.scaled)

Residuals:
    Min      1Q  Median      3Q     Max
-4.7768 -0.2137  0.0560  0.2444  1.5795

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.959e-14  1.653e-03   0.000   1.0000
carat        1.025e+00  1.899e-03 540.026  < 2e-16 ***
cut.L        2.457e-02  2.677e-03   9.179  < 2e-16 ***
cut.Q       -4.757e-03  2.791e-03  -1.705   0.0883 .
cut.C        1.268e-02  2.237e-03   5.667 1.46e-08 ***
cut_4        2.137e-03  1.869e-03   1.143   0.2530
color.L     -1.603e-01  1.888e-03 -84.918  < 2e-16 ***
color.Q     -5.285e-02  1.831e-03 -28.871  < 2e-16 ***
color.C      2.460e-05  1.810e-03   0.014   0.9892
color_4      1.505e-02  1.782e-03   8.448  < 2e-16 ***
color_5     -6.681e-03  1.702e-03  -3.926 8.65e-05 ***
color_6      5.532e-04  1.681e-03   0.329   0.7420
clarity.L    1.891e-01  2.585e-03  73.170  < 2e-16 ***
clarity.Q   -8.582e-02  2.271e-03 -37.790  < 2e-16 ***
clarity.C    6.390e-02  2.479e-03  25.779  < 2e-16 ***
clarity_4   -2.354e-02  2.295e-03 -10.257  < 2e-16 ***
clarity_5    1.698e-02  2.012e-03   8.440  < 2e-16 ***
clarity_6   -1.291e-03  1.878e-03  -0.687   0.4920
clarity_7    1.763e-03  1.755e-03   1.005   0.3151
depth       -3.771e-05  1.929e-03  -0.020   0.9844
table        1.549e-02  2.215e-03   6.990 2.80e-12 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3325 on 40436 degrees of freedom
Multiple R-squared:  0.8895,    Adjusted R-squared:  0.8895
F-statistic: 1.628e+04 on 20 and 40436 DF,  p-value: < 2.2e-16
```

```r
vif(fit3)
```

```
    carat     cut.L     cut.Q     cut.C     cut_4   color.L   color.Q   color.C
 1.319348  2.621717  2.850233  1.832031  1.278920  1.304379  1.226508  1.199441
  color_4   color_5   color_6 clarity.L clarity.Q clarity.C clarity_4 clarity_5
 1.161859  1.059918  1.033804  2.445376  1.887271  2.248751  1.926817  1.481558
clarity_6 clarity_7     depth     table
 1.291236  1.126544  1.362239  1.796295
```

```r
par(mfrow = c(2, 2))
plot(fit3)
```




```r
ytr.true <- exp(trainset$logprice)
yte.true <- exp(testset$logprice)
ytr.pred3 <- exp(predict(fit3, train.x) * sigma + mu)
yte.pred3 <- exp(predict(fit3, test.x) * sigma + mu)
```

```r
pf.lm <- data.frame(
    model = "lm",
    train.mse = mean((ytr.pred3-ytr.true)^2),
    test.mse = mean((yte.pred3-yte.true)^2)
)
pf.lm
```

```
  model train.mse   test.mse
1    lm 229599179 4206268380
```

```r
pf.linearRegression <- postResample(
    pred = yte.pred3,
    obs = yte.true
)
```

5.3 Stepwise regression and best subset

There're two Information Criterions:

Akaike Information Criterion $\text{AIC} = -2\log L + 2k$

Bayesian Information Criterion (aka Schwarz IC) $\text{BIC} = -2\log L + k\log N$

Here L denotes the maximum likelihood, k denotes number of parameters in the model, and N denotes the number of observations.

When sample size $N > 7 (e^2 \approx 7.389056)$, BIC is more likely to drop variables compare with AIC.

$$\text{Criterion} = \text{Penalty for Error} + \text{Penalty for Complexity}$$

The smaller, the better.

- **Stepwise Regression**: Backward (start from full model) or Forward (start from null model) then drop or add variables based on greedy strategy.
- **Best Subset Regression**: Iterate all subsets of full variables, then find the best subset.

5.3.1 Stepwise Regression

```r
library(MASS)
```

```
Attaching package: 'MASS'
```

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

    select
```

5.3.1.1 Backward (start from full model)

set `k = 2` for AIC, `k = log(nrow(train.data))` for BIC.

When you set `trace = 1`, every step will be printed.

```r
fit_s1a <- lm(logprice~., trainset.scaled)
fit_step_backward <- stepAIC(fit_s1a, k = log(nrow(trainset.scaled)), trace = 0) # MASS pkg
summary(fit_step_backward)
```

```
Call:
lm(formula = logprice ~ carat + cut.L + cut.Q + cut.C + cut_4 +
    color.L + color.Q + color.C + color_4 + clarity.L + clarity.Q +
    clarity.C + clarity_4 + clarity_5 + clarity_7 + depth + table +
    x + y + z, data = trainset.scaled)

Residuals:
    Min      1Q  Median      3Q     Max
-2.2146 -0.0910  0.0021  0.0900  9.8143

Coefficients:
              Estimate Std. Error  t value Pr(>|t|)
(Intercept) -1.795e-14  8.879e-04    0.000  1.00000
carat       -2.533e-01  4.176e-03  -60.651  < 2e-16 ***
cut.L        3.546e-02  1.438e-03   24.663  < 2e-16 ***
cut.Q       -1.589e-02  1.500e-03  -10.593  < 2e-16 ***
cut.C        1.484e-02  1.203e-03   12.338  < 2e-16 ***
cut_4        5.114e-03  1.005e-03    5.090 3.60e-07 ***
color.L     -1.442e-01  1.015e-03 -142.104  < 2e-16 ***
color.Q     -3.530e-02  9.829e-04  -35.917  < 2e-16 ***
```

```
color.C     -3.310e-03  9.605e-04   -3.446  0.00057 ***
color_4      8.162e-03  9.336e-04    8.743  < 2e-16 ***
clarity.L    2.220e-01  1.386e-03  160.173  < 2e-16 ***
clarity.Q   -6.169e-02  1.210e-03  -50.991  < 2e-16 ***
clarity.C    4.353e-02  1.322e-03   32.916  < 2e-16 ***
clarity_4   -2.260e-02  1.190e-03  -18.988  < 2e-16 ***
clarity_5    1.105e-02  1.053e-03   10.498  < 2e-16 ***
clarity_7    1.260e-02  9.034e-04   13.949  < 2e-16 ***
depth        7.160e-02  1.159e-03   61.777  < 2e-16 ***
table        2.002e-02  1.190e-03   16.821  < 2e-16 ***
x            1.276e+00  6.303e-03  202.424  < 2e-16 ***
y            1.456e-02  3.555e-03    4.096 4.21e-05 ***
z            2.997e-02  3.932e-03    7.622 2.55e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1786 on 40436 degrees of freedom
Multiple R-squared:  0.9681,    Adjusted R-squared:  0.9681
F-statistic: 6.14e+04 on 20 and 40436 DF,  p-value: < 2.2e-16
```

```r
ytr.pred.sb <- exp(fit_step_backward %>% predict(train.x) * sigma + mu)
yte.pred.sb <- exp(fit_step_backward %>% predict(test.x) * sigma + mu)



pf.lm.sb <- data.frame(
    model = "Stepwise-Backward",
    train.mse = mean((ytr.pred.sb-ytr.true)^2),
    test.mse = mean((yte.pred.sb-yte.true)^2)
)
pf.lm.sb
```

```
              model train.mse test.mse
1 Stepwise-Backward   1200659  1243213
```

```r
pf.Stepwise.Backward <- postResample(
    pred = yte.pred.sb,
    obs = yte.true
)
```

5.3.1.2 Forward (start from null model)

```
fit_s2a <- lm(logprice~1, trainset.scaled)
fit_s2b <- lm(logprice~., trainset.scaled)
fit_step_forward <- stepAIC(fit_s2a, scope = list(lower = fit_s2a, upper = fit_s2b),
                        direction = "forward", k = log(nrow(trainset.scaled)), trace = 0) # l
summary(fit_step_forward)
```

```
Call:
lm(formula = logprice ~ x + clarity.L + color.L + carat + depth +
    clarity.Q + color.Q + clarity.C + clarity_4 + cut.L + table +
    clarity_7 + cut_4 + clarity_5 + color_4 + z + cut.C + cut.Q +
    y + color.C, data = trainset.scaled)

Residuals:
    Min      1Q  Median      3Q     Max
-2.2146 -0.0910  0.0021  0.0900  9.8143

Coefficients:
              Estimate Std. Error  t value Pr(>|t|)
(Intercept) -1.795e-14  8.879e-04    0.000  1.00000
x            1.276e+00  6.303e-03  202.424  < 2e-16 ***
clarity.L    2.220e-01  1.386e-03  160.173  < 2e-16 ***
color.L     -1.442e-01  1.015e-03 -142.104  < 2e-16 ***
carat       -2.533e-01  4.176e-03  -60.651  < 2e-16 ***
depth        7.160e-02  1.159e-03   61.777  < 2e-16 ***
clarity.Q   -6.169e-02  1.210e-03  -50.991  < 2e-16 ***
color.Q     -3.530e-02  9.829e-04  -35.917  < 2e-16 ***
clarity.C    4.353e-02  1.322e-03   32.916  < 2e-16 ***
clarity_4   -2.260e-02  1.190e-03  -18.988  < 2e-16 ***
cut.L        3.546e-02  1.438e-03   24.663  < 2e-16 ***
table        2.002e-02  1.190e-03   16.821  < 2e-16 ***
clarity_7    1.260e-02  9.034e-04   13.949  < 2e-16 ***
cut_4        5.114e-03  1.005e-03    5.090 3.60e-07 ***
clarity_5    1.105e-02  1.053e-03   10.498  < 2e-16 ***
color_4      8.162e-03  9.336e-04    8.743  < 2e-16 ***
z            2.997e-02  3.932e-03    7.622 2.55e-14 ***
cut.C        1.484e-02  1.203e-03   12.338  < 2e-16 ***
cut.Q       -1.589e-02  1.500e-03  -10.593  < 2e-16 ***
y            1.456e-02  3.555e-03    4.096 4.21e-05 ***
```

```
color.C     -3.310e-03  9.605e-04   -3.446  0.00057 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1786 on 40436 degrees of freedom
Multiple R-squared:  0.9681,    Adjusted R-squared:  0.9681
F-statistic: 6.14e+04 on 20 and 40436 DF,  p-value: < 2.2e-16
```

```
ytr.pred.sf <- exp(fit_step_forward %>% predict(train.x) * sigma + mu)
yte.pred.sf <- exp(fit_step_forward %>% predict(test.x) * sigma + mu)



pf.lm.sf <- data.frame(
    model = "Stepwise-Forward",
    train.mse = mean((ytr.pred.sf-ytr.true)^2),
    test.mse = mean((yte.pred.sf-yte.true)^2)
)
pf.lm.sb
```

```
              model train.mse test.mse
1 Stepwise-Backward   1200659  1243213
```

```
pf.Stepwise.Forward <- postResample(
    pred = yte.pred.sf,
    obs = yte.true
)
```

5.3.2 Best Subset Regression

```
library(leaps)
```

```
fit_bs <- regsubsets(logprice~., trainset.scaled, nvmax = 30)
# nvmax = 30: sets the maximum number of predictors to consider.
summary(fit_bs)
```

```
Subset selection object
Call: regsubsets.formula(logprice ~ ., trainset.scaled, nvmax = 30)
23 Variables  (and intercept)
```

```
          Forced in Forced out
carat         FALSE      FALSE
cut.L         FALSE      FALSE
cut.Q         FALSE      FALSE
cut.C         FALSE      FALSE
cut_4         FALSE      FALSE
color.L       FALSE      FALSE
color.Q       FALSE      FALSE
color.C       FALSE      FALSE
color_4       FALSE      FALSE
color_5       FALSE      FALSE
color_6       FALSE      FALSE
clarity.L     FALSE      FALSE
clarity.Q     FALSE      FALSE
clarity.C     FALSE      FALSE
clarity_4     FALSE      FALSE
clarity_5     FALSE      FALSE
clarity_6     FALSE      FALSE
clarity_7     FALSE      FALSE
depth         FALSE      FALSE
table         FALSE      FALSE
x             FALSE      FALSE
y             FALSE      FALSE
z             FALSE      FALSE
1 subsets of each size up to 23
Selection Algorithm: exhaustive
          carat cut.L cut.Q cut.C cut_4 color.L color.Q color.C color_4 color_5
1  ( 1 )  " "   " "   " "   " "   " "   " "     " "     " "     " "     " "
2  ( 1 )  " "   " "   " "   " "   " "   " "     " "     " "     " "     " "
3  ( 1 )  " "   " "   " "   " "   " "   "*"     " "     " "     " "     " "
4  ( 1 )  "*"   " "   " "   " "   " "   "*"     " "     " "     " "     " "
5  ( 1 )  "*"   " "   " "   " "   " "   "*"     " "     " "     " "     " "
6  ( 1 )  "*"   " "   " "   " "   " "   "*"     " "     " "     " "     " "
7  ( 1 )  "*"   " "   " "   " "   " "   "*"     "*"     " "     " "     " "
8  ( 1 )  "*"   " "   " "   " "   " "   "*"     "*"     " "     " "     " "
9  ( 1 )  "*"   " "   " "   " "   " "   "*"     "*"     " "     " "     " "
10  ( 1 ) "*"   "*"   " "   " "   " "   "*"     "*"     " "     " "     " "
11  ( 1 ) "*"   "*"   " "   " "   " "   "*"     "*"     " "     " "     " "
12  ( 1 ) "*"   "*"   " "   " "   " "   "*"     "*"     " "     " "     " "
13  ( 1 ) "*"   "*"   " "   " "   "*"   "*"     "*"     " "     " "     " "
14  ( 1 ) "*"   "*"   "*"   "*"   " "   "*"     "*"     " "     " "     " "
15  ( 1 ) "*"   "*"   "*"   "*"   " "   "*"     "*"     " "     " "     " "
16  ( 1 ) "*"   "*"   "*"   "*"   " "   "*"     "*"     " "     "*"     " "
```

```
17  ( 1 ) "*"   "*"   "*"   "*"   " "   "*"     "*"     " "     "*"     " "
18  ( 1 ) "*"   "*"   "*"   "*"   "*"   "*"     "*"     " "     "*"     " "
19  ( 1 ) "*"   "*"   "*"   "*"   "*"   "*"     "*"     " "     "*"     " "
20  ( 1 ) "*"   "*"   "*"   "*"   "*"   "*"     "*"     "*"     "*"     " "
21  ( 1 ) "*"   "*"   "*"   "*"   "*"   "*"     "*"     "*"     "*"     " "
22  ( 1 ) "*"   "*"   "*"   "*"   "*"   "*"     "*"     "*"     "*"     "*"
23  ( 1 ) "*"   "*"   "*"   "*"   "*"   "*"     "*"     "*"     "*"     "*"
          color_6 clarity.L clarity.Q clarity.C clarity_4 clarity_5 clarity_6
1   ( 1 )  " "     " "       " "       " "       " "       " "       " "
2   ( 1 )  " "     "*"       " "       " "       " "       " "       " "
3   ( 1 )  " "     "*"       " "       " "       " "       " "       " "
4   ( 1 )  " "     "*"       " "       " "       " "       " "       " "
5   ( 1 )  " "     "*"       " "       " "       " "       " "       " "
6   ( 1 )  " "     "*"       "*"       " "       " "       " "       " "
7   ( 1 )  " "     "*"       "*"       " "       " "       " "       " "
8   ( 1 )  " "     "*"       "*"       "*"       " "       " "       " "
9   ( 1 )  " "     "*"       "*"       "*"       "*"       " "       " "
10  ( 1 ) " "     "*"       "*"       "*"       "*"       " "       " "
11  ( 1 ) " "     "*"       "*"       "*"       "*"       " "       " "
12  ( 1 ) " "     "*"       "*"       "*"       "*"       " "       " "
13  ( 1 ) " "     "*"       "*"       "*"       "*"       " "       " "
14  ( 1 ) " "     "*"       "*"       "*"       "*"       " "       " "
15  ( 1 ) " "     "*"       "*"       "*"       "*"       "*"       " "
16  ( 1 ) " "     "*"       "*"       "*"       "*"       "*"       " "
17  ( 1 ) " "     "*"       "*"       "*"       "*"       "*"       " "
18  ( 1 ) " "     "*"       "*"       "*"       "*"       "*"       " "
19  ( 1 ) " "     "*"       "*"       "*"       "*"       "*"       " "
20  ( 1 ) " "     "*"       "*"       "*"       "*"       "*"       " "
21  ( 1 ) " "     "*"       "*"       "*"       "*"       "*"       "*"
22  ( 1 ) " "     "*"       "*"       "*"       "*"       "*"       "*"
23  ( 1 ) "*"     "*"       "*"       "*"       "*"       "*"       "*"
          clarity_7 depth table x   y   z
1   ( 1 )  " "       " "   " "   "*" " " " "
2   ( 1 )  " "       " "   " "   "*" " " " "
3   ( 1 )  " "       " "   " "   "*" " " " "
4   ( 1 )  " "       " "   " "   "*" " " " "
5   ( 1 )  " "       "*"   " "   "*" " " " "
6   ( 1 )  " "       "*"   " "   "*" " " " "
7   ( 1 )  " "       "*"   " "   "*" " " " "
8   ( 1 )  " "       "*"   " "   "*" " " " "
9   ( 1 )  " "       "*"   " "   "*" " " " "
10  ( 1 ) " "       "*"   " "   "*" " " " "
11  ( 1 ) " "       "*"   "*"   "*" " " " "
```

```
12  ( 1 ) "*"       "*"   "*"   "*" " " " " "
13  ( 1 ) "*"       "*"   "*"   "*" " " " " "
14  ( 1 ) "*"       "*"   "*"   "*" " " " " "
15  ( 1 ) "*"       "*"   "*"   "*" " " " " "
16  ( 1 ) "*"       "*"   "*"   "*" " " " " "
17  ( 1 ) "*"       "*"   "*"   "*" " " " " "*"
18  ( 1 ) "*"       "*"   "*"   "*" " " " " "*"
19  ( 1 ) "*"       "*"   "*"   "*" "*" " " "*"
20  ( 1 ) "*"       "*"   "*"   "*" "*" " " "*"
21  ( 1 ) "*"       "*"   "*"   "*" "*" " " "*"
22  ( 1 ) "*"       "*"   "*"   "*" "*" " " "*"
23  ( 1 ) "*"       "*"   "*"   "*" "*" " " "*"
```

```
result <- summary(fit_bs)
result$which[which.min(result$bic),]
```

```
(Intercept)       carat       cut.L       cut.Q       cut.C       cut_4
       TRUE        TRUE        TRUE        TRUE        TRUE        TRUE
    color.L     color.Q     color.C     color_4     color_5     color_6
       TRUE        TRUE        TRUE        TRUE       FALSE       FALSE
  clarity.L   clarity.Q   clarity.C   clarity_4   clarity_5   clarity_6
       TRUE        TRUE        TRUE        TRUE        TRUE       FALSE
  clarity_7       depth       table           x           y           z
       TRUE        TRUE        TRUE        TRUE        TRUE        TRUE
```

```
n <- nrow(trainset.scaled)  # Sample size
p <- 1:(length(result$rss))  # Model sizes (number of predictors + intercept)
aic_values <- n * log(result$rss / n) + 2 * p  # Compute AIC for each model
which.min(aic_values)  # Get the model size with the lowest AIC
```

```
[1] 22
```

```
result$which[12,]
```

```
(Intercept)       carat       cut.L       cut.Q       cut.C       cut_4
       TRUE        TRUE        TRUE       FALSE       FALSE       FALSE
    color.L     color.Q     color.C     color_4     color_5     color_6
       TRUE        TRUE       FALSE       FALSE       FALSE       FALSE
  clarity.L   clarity.Q   clarity.C   clarity_4   clarity_5   clarity_6
       TRUE        TRUE        TRUE        TRUE       FALSE       FALSE
  clarity_7       depth       table           x           y           z
       TRUE        TRUE        TRUE        TRUE       FALSE       FALSE
```

```
which.max(result$adjr2)
```

```
[1] 23
```

```
result$which[1,]
```

```
(Intercept)       carat       cut.L       cut.Q       cut.C       cut_4
       TRUE       FALSE       FALSE       FALSE       FALSE       FALSE
    color.L     color.Q     color.C     color_4     color_5     color_6
      FALSE       FALSE       FALSE       FALSE       FALSE       FALSE
  clarity.L   clarity.Q   clarity.C   clarity_4   clarity_5   clarity_6
      FALSE       FALSE       FALSE       FALSE       FALSE       FALSE
  clarity_7       depth       table           x           y           z
      FALSE       FALSE       FALSE        TRUE       FALSE       FALSE
```

5.4 Lasso, Ridge and Elastic Net Regression

A general idea of penealized regression (i.e., Lasso, Ridge and Elastic Net Regression) is to add some restriction to minimize.

5.4.1 Lasso

The defination of Lasso is based on the loss function: $Q = \|(y - X\beta)\|^2$ minimized subject to $|\beta|_1 < t$

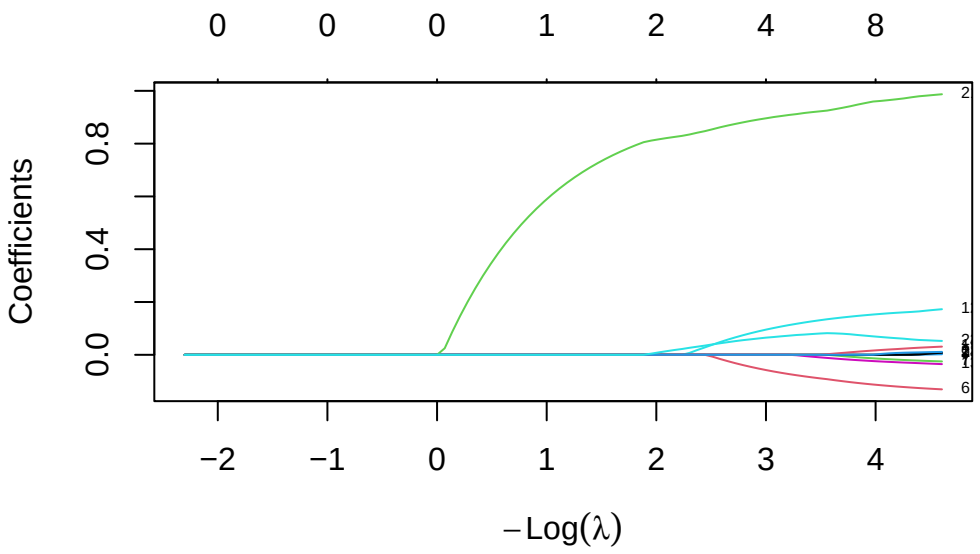
It can also be written:

$$\min_{\beta \in \mathbb{R}^p} (\frac{1}{N} \|(y - X\beta)\|^2 + \lambda |\beta|_1)$$

Lasso will let coeffections of the model to 0, which is useful for feature selection.

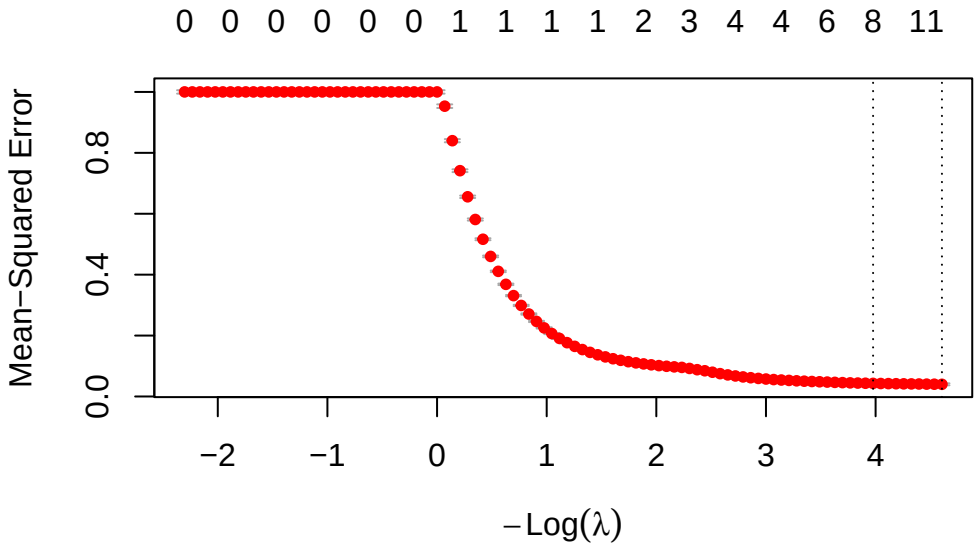
set `alpha = 1` for function `glmnet`.

```
lambda <- 10^seq(1,-2,length = 100)

fit.lasso <- glmnet(train.x, trainset.scaled$logprice, alpha = 1, lambda = lambda)

plot(fit.lasso, xvar = "lambda", label = TRUE)
```




```
cv.lasso <- cv.glmnet(as.matrix(train.x), trainset.scaled$logprice,alpha = 1, lambda = lambda
```

```
plot(cv.lasso)
```




```
cv.lasso
```

```
Call:  cv.glmnet(x = as.matrix(train.x), y = trainset.scaled$logprice,      lambda = lambda,

Measure: Mean-Squared Error
```

```
     Lambda Index Measure       SE Nonzero
min 0.01000   100 0.03992 0.003268      11
1se 0.01874    91 0.04297 0.003157       8
```

```r
best_lam <- cv.lasso$lambda.min
lasso_coef <- predict(cv.lasso, s = best_lam, type = "coefficients")

import_coefs <- as.matrix(lasso_coef)
active_vars <- data.frame(
  Variable = rownames(import_coefs),
  Coefficient = as.numeric(import_coefs)
) %>% filter(Coefficient != 0 & Variable != "(Intercept)")


active_vars
```

```
    Variable  Coefficient
1      cut.L  0.006033271
2    color.L -0.130907441
3    color.Q -0.025134709
4    color_4  0.002078327
5  clarity.L  0.172731638
6  clarity.Q -0.035257206
7  clarity.C  0.007961533
8      depth  0.030957031
9          x  0.987147413
10         y  0.010949118
11         z  0.052448464
```

```r
cat("Dropped Variables: ", rownames(import_coefs)[import_coefs[,1] == 0])
```

```
Dropped Variables:  carat cut.Q cut.C cut_4 color.C color_5 color_6 clarity_4 clarity_5 clari
```

```r
ytr.pred.lasso <- exp(predict(cv.lasso,s =best_lam, newx=as.matrix(train.x)) * sigma + mu)
yte.pred.lasso <- exp(predict(cv.lasso,s = best_lam,newx= as.matrix(test.x)) * sigma + mu)


pf.lasso <- data.frame(
    model = "lasso",
    train.mse = mean((ytr.pred.lasso-ytr.true)^2),
```

```
    test.mse = mean((yte.pred.lasso-yte.true)^2)
)
pf.lasso
```

```
  model train.mse test.mse
1 lasso   3193370  4703154
```

```
pf.lasso2 <- postResample(
    pred = yte.pred.lasso,
    obs = yte.true
)
```

5.4.2 Ridge

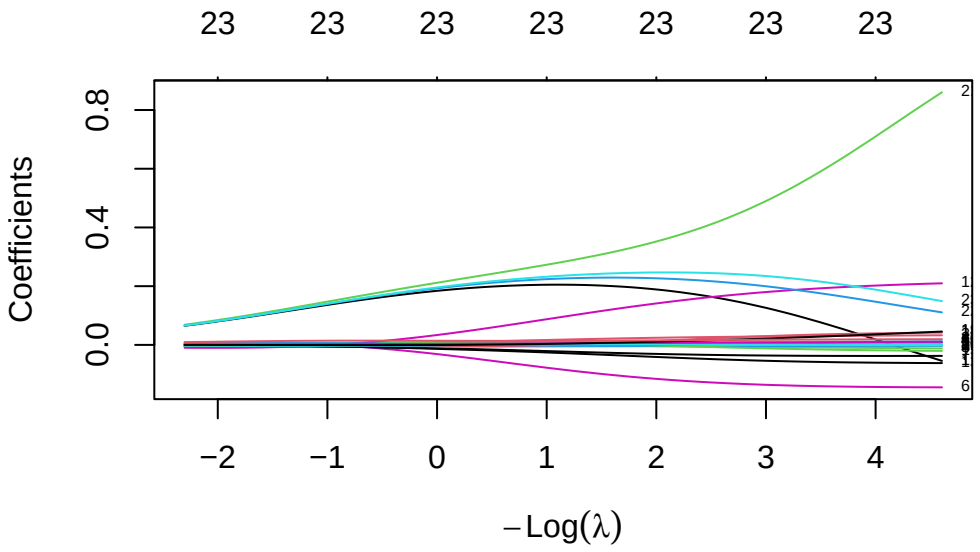
loss function $Q = (y - X\beta)^T(y - X\beta) + \beta^T\beta$

Ridge works best in situation that OLS has high variance.

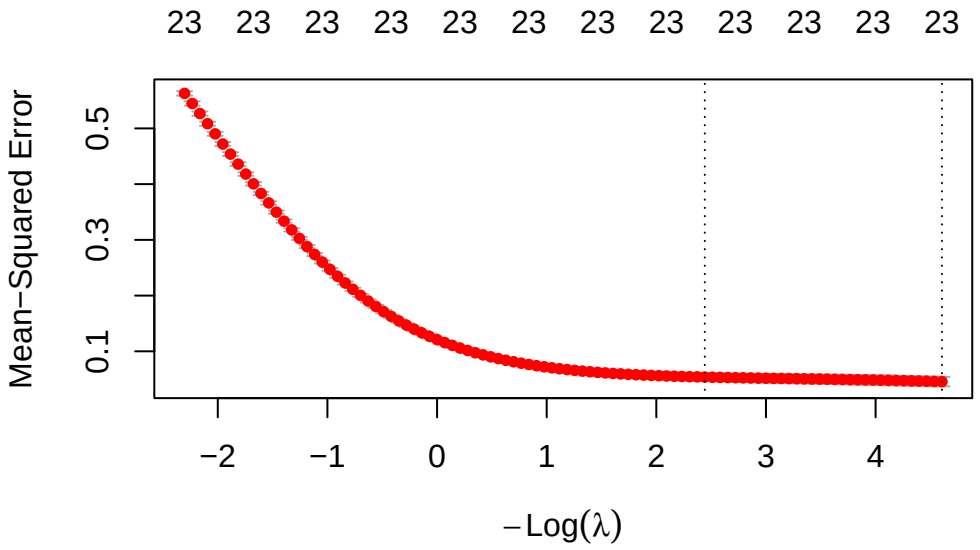
$$\frac{\partial Q}{\partial \beta} = -2X^T y + 2X^T X\beta + 2\lambda\beta = 0 \rightarrow \hat{\beta} = (X^T X + \lambda I)^{-1} X^T y$$

set `alpha = 0` for function `glmnet`.

```
lambda <- 10^seq(1,-2,length = 100)

fit.ridge <- glmnet(train.x, trainset.scaled$logprice, alpha = 0, lambda = lambda)

plot(fit.ridge, xvar = "lambda", label = TRUE)
```




```
cv.ridge <- cv.glmnet(as.matrix(train.x), trainset.scaled$logprice,alpha = 0, lambda = lambda

plot(cv.ridge)
```




```
cv.ridge
```

```
Call:  cv.glmnet(x = as.matrix(train.x), y = trainset.scaled$logprice,      lambda = lambda,

Measure: Mean-Squared Error
```

```
     Lambda Index Measure       SE Nonzero
min 0.01000   100 0.04562 0.008444      23
1se 0.08697    69 0.05379 0.006803      23
```

```r
best_lam <- cv.ridge$lambda.min
ridge_coef <- predict(cv.ridge, s = best_lam, type = "coefficients")

ridge_coef
```

```
24 x 1 sparse Matrix of class "dgCMatrix"
                    s=0.01
(Intercept) -1.564928e-14
carat       -5.448247e-02
cut.L        3.267201e-02
cut.Q       -1.197818e-02
cut.C        1.219518e-02
cut_4        2.998444e-03
color.L     -1.446473e-01
color.Q     -3.744780e-02
color.C     -2.738590e-03
color_4      9.394073e-03
color_5     -3.019684e-03
color_6      7.682292e-04
clarity.L    2.097124e-01
clarity.Q   -6.202300e-02
clarity.C    4.186375e-02
clarity_4   -2.048792e-02
clarity_5    9.100290e-03
clarity_6   -1.519407e-03
clarity_7    1.037701e-02
depth        4.537965e-02
table        1.932575e-02
x            8.605183e-01
y            1.107601e-01
z            1.491461e-01
```

```r
ytr.pred.ridge <- exp(predict(cv.ridge,s =best_lam, newx=as.matrix(train.x)) * sigma + mu)
yte.pred.ridge <- exp(predict(cv.ridge,s = best_lam,newx= as.matrix(test.x)) * sigma + mu)
```

```
pf.ridge <- data.frame(
    model = "ridge",
    train.mse = mean((ytr.pred.ridge-ytr.true)^2),
    test.mse = mean((yte.pred.ridge-yte.true)^2)
)
pf.ridge
```

```
  model train.mse test.mse
1 ridge 391270522  2773539
```

```
pf.ridge2 <- postResample(
    pred = yte.pred.ridge,
    obs = yte.true
)
```

5.5 Summary

```
rbind(
    pf.lm,
    pf.lm.sb,
    pf.lm.sf,
    pf.lasso,
    pf.ridge
)
```

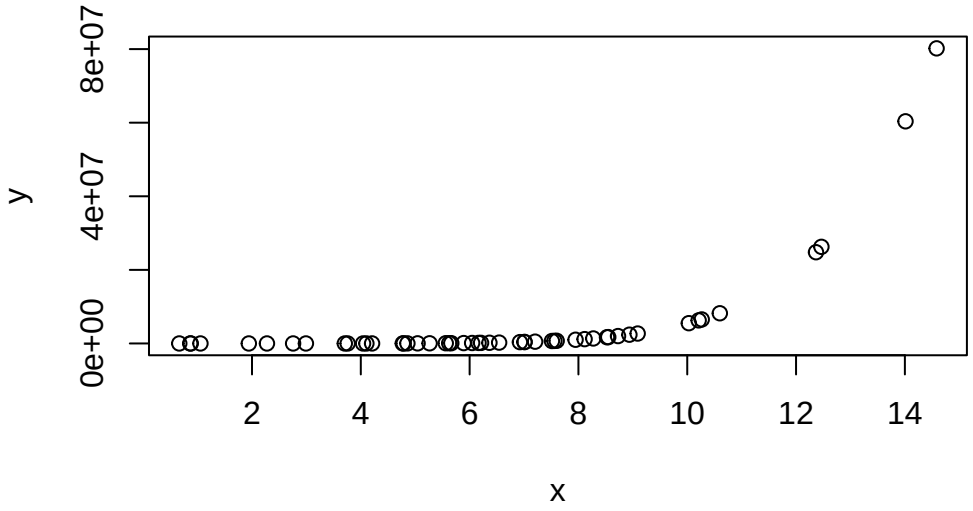
```
              model train.mse   test.mse
1                lm 229599179 4206268380
2 Stepwise-Backward   1200659    1243213
3  Stepwise-Forward   1200659    1243213
4             lasso   3193370    4703154
5             ridge 391270522    2773539
```

```
rbind(
    pf.linearRegression,
    pf.Stepwise.Backward,
    pf.Stepwise.Forward,
    pf.lasso2,
    pf.ridge2
)
```

```
                          RMSE   Rsquared       MAE
pf.linearRegression  64855.751 0.01863255 2350.7410
pf.Stepwise.Backward  1114.994 0.93432609  486.5987
pf.Stepwise.Forward   1114.994 0.93432609  486.5987
pf.lasso2             2168.676 0.81577988  664.7580
pf.ridge2             1665.395 0.88162500  587.0151
```

5.6 Bad Example (Random Numbers)

```
x <- rnorm(50, 6,3)
y <- 10+ 2*x - 3 *x^3 -6*x^5 + 0.6 * x^7 + rnorm(50,10,8)

plot(x,y)
```




```
lm(y~x) %>%
    summary()
```

```
Call:
lm(formula = y ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-9551661 -7168900 -2518269  3203266 51954823
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -14210959    3541874  -4.012  0.00021 ***
x             2911495     486816   5.981 2.68e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11180000 on 48 degrees of freedom
Multiple R-squared:  0.427, Adjusted R-squared:  0.4151
F-statistic: 35.77 on 1 and 48 DF,  p-value: 2.683e-07
```

```
lm(y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5)+I(x^6)+I(x^7)) %>%
    summary()
```

```
Call:
lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) +
    I(x^7))

Residuals:
     Min       1Q   Median       3Q      Max
-14.0609  -5.1669  -0.8895   4.8241  16.1121

Coefficients:
              Estimate Std. Error  t value Pr(>|t|)
(Intercept)  5.714e+01  2.726e+01    2.096   0.0421 *
x           -7.012e+01  5.435e+01   -1.290   0.2040
I(x^2)       4.399e+01  3.591e+01    1.225   0.2274
I(x^3)      -1.544e+01  1.123e+01   -1.376   0.1762
I(x^4)       1.855e+00  1.873e+00    0.990   0.3278
I(x^5)      -6.150e+00  1.712e-01  -35.933   <2e-16 ***
I(x^6)       6.295e-03  8.064e-03    0.781   0.4394
I(x^7)       5.999e-01  1.529e-04 3922.666   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.238 on 42 degrees of freedom
Multiple R-squared:      1, Adjusted R-squared:      1
F-statistic: 2.854e+13 on 7 and 42 DF,  p-value: < 2.2e-16
```

6 ANalysis of VAriance (ANOVA)

ANalysis of VAriance (ANOVA) is a method to make multiple group mean inference with F statistics based on variance.

$$H_0 : \mu_1 = \mu_2 = \mu_3 = ... \mu_p \quad H_a : \text{At least one of them different from others}$$

ANOVA tries to calculate the variance with in groups and variance for all groups, and have a F-test.

7 Logistics Regression

7.1 Theoritical Knowledge

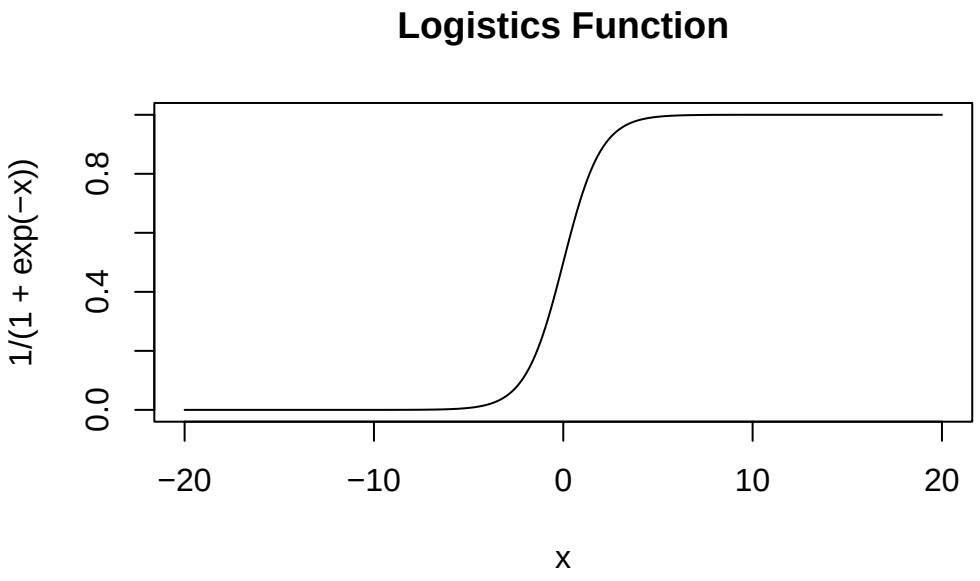
Assume you're predicting binary categorical data with OLS:

$$Y = \alpha + \beta X + \varepsilon$$

The problem is, $Y \sim Bin(p)$ rather than $N(\mu, \sigma^2)$. The target should be from $[0, 1]$, indicating the probability which category this sample likely belongs to. However, the predicated value may greater than 1 or less than 0, which cause a problem.

But you can have a function to map $(-\infty, \infty)$ to $(0, 1)$ which is $y = \frac{1}{1+e^{-x}}$ (logistic function, aka. sigmoid function as activation function in neural network)

```
x <- seq(-20,20,0.1)
plot(x, 1/(1+exp(-x)), 'l')
title('Logistics Function')
```



Let $p = \frac{1}{1+e^{-\eta}}$. Thus the model is,

$$\eta = \log \frac{p}{1-p} = \alpha + \beta X + \varepsilon$$

Here we define Odds ($\eta = \log Odds$) helps explain the model.

$$Odds = \frac{p}{1-p}$$

We use MLE to estimate, and solve by iteration. In R, we use `glm` function with `family = binomial`.

MLE function is given by

$$L = \prod_{i=1}^{n} (1-p)^{1-y_i} p^{y_i} = (1-p)^{n-\sum y_i} p^{\sum y_i}$$

$$l = (n - \sum y_i) \log (1-p) + (\sum y_i) \log p$$

R utilize Iteratively Reweighted Least Squares (IRLS) to calculate the coefficients of the regression. Guess first, then iteration until likelihood function no longer increase.

For generalized model, p variables,

$$\log \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

$$P(Y=1) = \frac{e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p}}{1 + e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p}}$$

$$\text{Odds Ratio} = OR = \frac{Odds_1}{Odds_2}$$

$$\text{Relative Risk} = RR = \frac{p_1}{p_2} \to OR = RR \times \frac{1-p_1}{1-p_2}$$

Note that when p is small, $OR \approx RR$

to explain the model, we can write

$$\frac{p}{1-p} = e^{\beta_0} (e^{\beta_1})^{x_1} (e^{\beta_2})^{x_2} \ldots (e^{\beta_p})^{x_p}$$

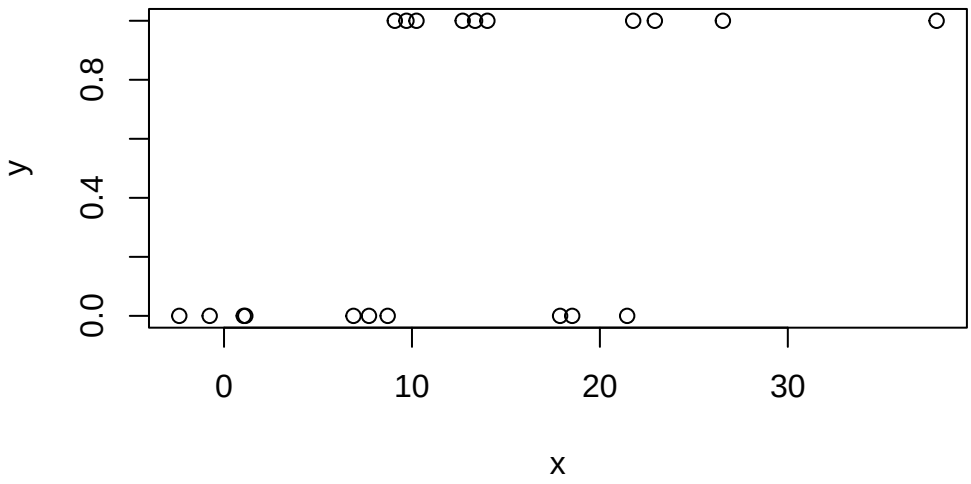
when reply X by $X+1$ then

$$\frac{e^{\beta_0+\beta_1(X+1)}}{e^{\beta_0+\beta_1 X}} = e^{\beta_1}$$

The odds ratio $e^{\beta} > 1$ indicates increasing odds, otherwise it's decreasing odds.

7.2 R code

```
set.seed(3306)

y <- sample(c(0,1), 20, T)
x <- y * 5 -y^2  + runif(20, 5, 10)+ rnorm(20, 5, 10)
plot(x,y)
```




```
model <- glm(y ~ x, family = binomial)
summary(model)
```

```
Call:
glm(formula = y ~ x, family = binomial)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.73353    1.00181   -1.73   0.0836 .
x            0.13851    0.07067    1.96   0.0500 *
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 27.726  on 19  degrees of freedom
Residual deviance: 21.951  on 18  degrees of freedom
AIC: 25.951

Number of Fisher Scoring iterations: 4
```

```r
log_or <- coef(model)["x"]
or <- exp(log_or)
list(log_or = log_or, or = or)
```

```
$log_or
        x
0.1385138

$or
       x
1.148566
```

```r
confint(model)
```

```
Waiting for profiling to be done...

                  2.5 %      97.5 %
(Intercept) -4.08999874 -0.00682785
x            0.02215079  0.30995357
```

7.3 Performance

For classfication models like logitics regression, **Confusion Matrix** is a good method to evaluate your model.

7.3.1 Binary Classification: Confusion Matrix

	Predicted: NO (0) *(Not to Reject)*	Predicted: YES (1) *(Reject Null Hypothesis)*	Total
Actual: NO (0) *(Alter is Wrong)*	True Negative (TN)	False Positive (FP) *(Type I Error)*	N
Actual: YES (1) *(Alter is Correct)*	False Negative (FN) *(Type II Error)*	True Positive (TP) *(Power)*	P
Total	N_{pred}	P_{pred}	Total (N)

You can see there's also a link between statistics inference and machine learning. Interesting! Note that in statistics we prefer writing H_0 (Null Hypothesis) is true or false rather than then H_1 (or H_a, alternative hypothesis).

The first letter T or F means whether this case is correctly identified; the second letter P or N means which sample the case belongs to.

```
prob <- predict(model, list(x), type = "response")
predicted.classes <- ifelse(prob > 0.5, "c1", "c0")
table(predicted.classes,y)
```

```
                y
predicted.classes 0 1
               c0 7 3
               c1 3 7
```

From the confusion matrix, we can tell $7 + 9 = 16$ samples correctly classificated, and 2 of Positive and 2 of Negative mistakenly classfied into Negative and Positive respectively.

So the TP = TN = 7, FP = FN = 3.

7.3.1.1 Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

7.3.1.2 Recall (or Sensitive):

$$\text{Recall} = \frac{TP}{TP + FN}$$

Recall is used to measure how many **Positive Samples** being correctly recognized. Assume you're predicating whether a product has an issue, and your target is to recall samples with an issue. Recall reports in what extend you find the real issue samples.

7.3.1.3 Specificity:

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Specificity is opposite to the Recall. The target is to find how many **Negative Samples** beging correctly recognized.

7.3.1.4 Precision

$$\text{Precision} = \frac{TP}{TP + FP}$$

Precision is different from either Recall and Specificity, since it measures how many samples with **Positive Results** really from **Positive Samples**. Assume you're predicating whether a product has an issue, Precision reports whether a sample you're recalling is really recall-needed one.

7.3.1.5 F-1 Score

Let's make an example. You're doing a model about fraud detection. The higher recall indicate you find almost most fraud, while you may interupt some non-fraud transcations; the higher Precision means you're not interupt most non-fraud transcations while may miss some fraud transcations.

F-1 Score is the **Harmonic Mean** of Recall and Precision

$$\text{F-1 Score} = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}} = 2\frac{Recall \times Precision}{Recall + Precision}$$

8 Generalized Linear Regression

> NOTE THAT GLM here is for **Generalized Linear Regression**, which is different from **General linear model**.
>
> Wikipedia says: Not to be confused with Multiple linear regression, General linear model or General linear methods.

Generalized Linear Regression (GLM) is linear regression that variables are categorical, and the response variable is exponential family

- variables are categorical, count data, etc
- A distibution for the regression
- A link function (here, g) connects the predictions to the mean of the distribution

$$\mathbb{E}[Y] = \mu = g^{-1}(X\beta)$$

Examples: Logistics Regression, Probit Regression, Poisson Regression

The logistic regression, $X\beta = \log \frac{\mu}{1-\mu}$

GLM uses likelihood function, Deviance: smaller deviance -> better model

GLM introduces the conception of **deviance**, smaller deviance means better model, which is defined:

$$D = -2 \log \frac{\text{Likelihood of Current Model}}{\text{Likelihood of Saturated Model}}$$

For large sample, the distribution of D is proved by **Wilks' Theorem**.

$$D \sim \chi^2 \quad df = k_{full} - k_{reduced}$$

```
glm(formular, family = ?)
```

8.1 Logitics Regression Revisited

use `family = binomial` for logistics regression.

```
no.yes <- c("No", "Yes")
smoking <- gl(2,1,8,no.yes)
obesity <- gl(2,2,8,no.yes)
snoring <- gl(2,4,8,no.yes)
n.tot <- c(60,17,8,2,187,85,51,23)
n.hyp <- c(5,2,1,0,35,13,15,8)

data.frame(smoking, obesity, snoring, n.tot, n.hyp)
```

```
  smoking obesity snoring n.tot n.hyp
1      No      No      No    60     5
2     Yes      No      No    17     2
3      No     Yes      No     8     1
4     Yes     Yes      No     2     0
5      No      No     Yes   187    35
6     Yes      No     Yes    85    13
7      No     Yes     Yes    51    15
8     Yes     Yes     Yes    23     8
```

```
hyp.tbl <- cbind(n.hyp, n.tot-n.hyp)
glm.hyp <- glm(hyp.tbl~smoking+obesity+snoring, family = binomial("logit"))
summary(glm.hyp)
```

```
Call:
glm(formula = hyp.tbl ~ smoking + obesity + snoring, family = binomial("logit"))

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.37766    0.38018  -6.254    4e-10 ***
smokingYes  -0.06777    0.27812  -0.244   0.8075
obesityYes   0.69531    0.28509   2.439   0.0147 *
snoringYes   0.87194    0.39757   2.193   0.0283 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)
```

```
    Null deviance: 14.1259  on 7  degrees of freedom
Residual deviance:  1.6184  on 4  degrees of freedom
AIC: 34.537

Number of Fisher Scoring iterations: 4
```

```r
anova(glm.hyp, test = "Chisq")
```

```
Analysis of Deviance Table

Model: binomial, link: logit

Response: hyp.tbl

Terms added sequentially (first to last)

        Df Deviance Resid. Df Resid. Dev Pr(>Chi)
NULL                        7    14.1259
smoking  1   0.0022         6    14.1237 0.962724
obesity  1   6.8274         5     7.2963 0.008977 **
snoring  1   5.6779         4     1.6184 0.017179 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

8.2 Poisson Regression

The response of Poisson Regression is count data.

- Aggregated counts over time
- Individual level data with event indicators

$$\log \lambda = X\beta$$

```r
library(ISwR)
data(eba1977)

str(eba1977)
```

```
'data.frame':   24 obs. of  4 variables:
 $ city : Factor w/ 4 levels "Fredericia","Horsens",..: 1 2 3 4 1 2 3 4 1 2 ...
 $ age  : Factor w/ 6 levels "40-54","55-59",..: 1 1 1 1 2 2 2 2 3 3 ...
 $ pop  : int  3059 2879 3142 2520 800 1083 1050 878 710 923 ...
 $ cases: int  11 13 4 5 11 6 8 7 11 15 ...
```

```r
fit.ps <- glm(cases~city+age, data = eba1977, family = poisson)
summary(fit.ps)
```

```
Call:
glm(formula = cases ~ city + age, family = poisson, data = eba1977)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  2.24374    0.20363  11.019   <2e-16 ***
cityHorsens -0.09844    0.18129  -0.543    0.587
cityKolding -0.22706    0.18770  -1.210    0.226
cityVejle   -0.22706    0.18770  -1.210    0.226
age55-59    -0.03077    0.24810  -0.124    0.901
age60-64     0.26469    0.23143   1.144    0.253
age65-69     0.31015    0.22918   1.353    0.176
age70-74     0.19237    0.23517   0.818    0.413
age75+      -0.06252    0.25012  -0.250    0.803
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 27.704  on 23  degrees of freedom
Residual deviance: 20.673  on 15  degrees of freedom
AIC: 135.06

Number of Fisher Scoring iterations: 5
```

9 Classification Models

9.1 Performance

Please see Logistics -> Performance

10 Neural Network

Note that dataset `diamonds` actually can be fit by simple models. We proved that in Lasso of Linear Regression is much better since lower MSE/MAE. While the linear regression is overfitted on training set, ridge is lessfitted, Lasso is the best regression on both training set and test set. Please check that.

10.1 neuralnet

```
library(tidyverse)
library(caret)
library(neuralnet)
```

```
data(diamonds)
head(diamonds)
```

```
# A tibble: 6 x 10
  carat cut       color clarity depth table price     x     y     z
  <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

```
sum(is.na(diamonds))
```

```
[1] 0
```

```
str(diamonds)
```

```
tibble [53,940 x 10] (S3: tbl_df/tbl/data.frame)
 $ carat  : num [1:53940] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
 $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
 $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
 $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
 $ depth  : num [1:53940] 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
 $ table  : num [1:53940] 55 61 65 58 58 57 57 55 61 61 ...
 $ price  : int [1:53940] 326 326 327 334 335 336 336 337 337 338 ...
 $ x      : num [1:53940] 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
 $ y      : num [1:53940] 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
 $ z      : num [1:53940] 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

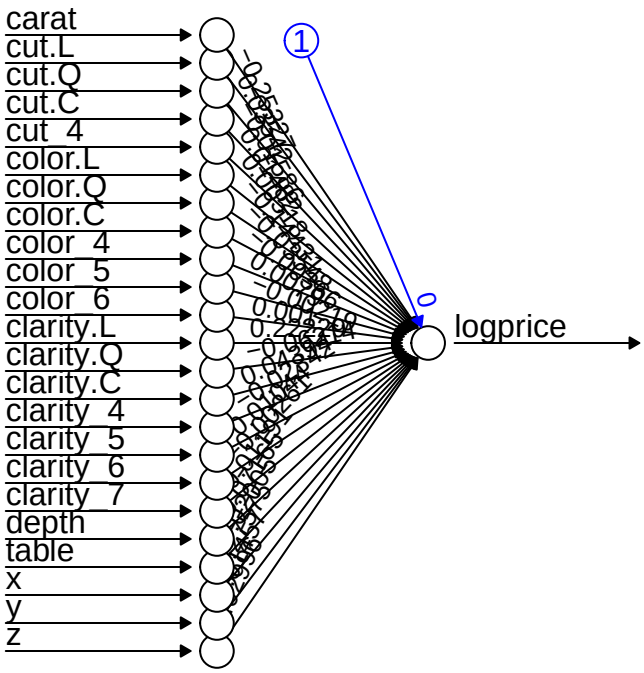
```
new_diamonds <- diamonds %>%
    predict(dummyVars(price ~ ., data = ., sep = "_", levelsOnly = FALSE), newdata = .) %>%
    as.data.frame() %>%
    mutate(logprice = log(diamonds$price))
```

```
set.seed(42)
colnames(new_diamonds) <- gsub("\\^", "_", colnames(new_diamonds))
training.idx <- new_diamonds$logprice %>%
    createDataPartition(p = 0.75, list = F)
trainset <- new_diamonds[training.idx, ]
testset <- new_diamonds[-training.idx, ]
```

```
prepproc <- preProcess(trainset[, -ncol(trainset)], method = c("center", "scale"))
train.x <- predict(prepproc, trainset[,-ncol(trainset)])
test.x <- predict(prepproc, testset[,-ncol(testset)])

trainset.scaled <- data.frame(train.x, logprice = trainset$logprice)
testset.scaled  <- data.frame(test.x, logprice = testset$logprice)
```

```
prepproc <- preProcess(trainset, method = c("center", "scale"))
trainset.scaled <- predict(prepproc, trainset)
testset.scaled <- predict(prepproc, testset)

mu <- prepproc$mean["logprice"]
sigma <- prepproc$std["logprice"]
```

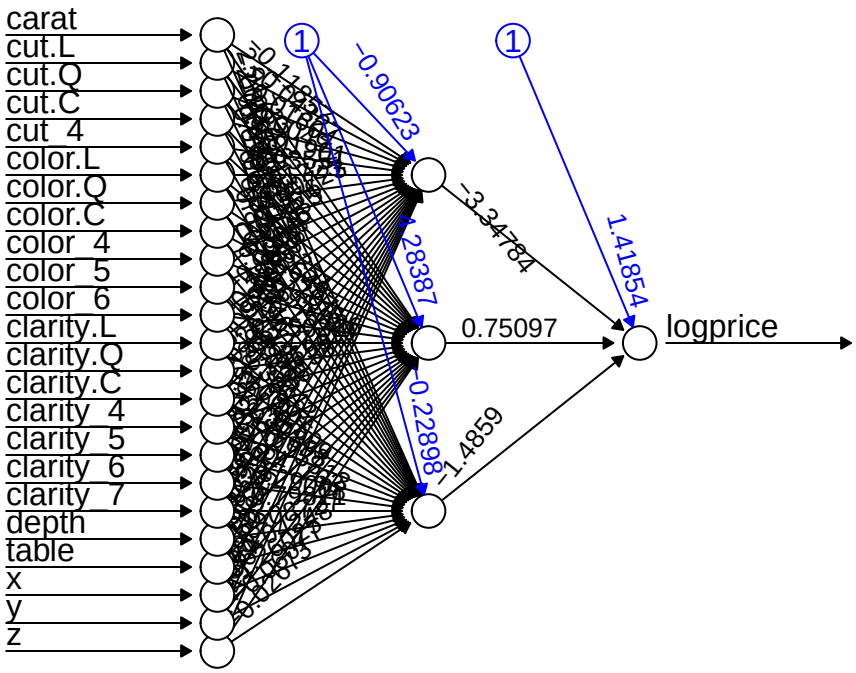
```
model <- neuralnet(logprice ~ ., data = trainset.scaled, hidden = 0, err.fct = "sse", linear
plot(model, rep = 'best')
```




```
pred_scaled <- compute(model, testset.scaled %>% select(-logprice))$net.result
pred_usd <- exp(pred_scaled * sigma + mu)

performance <- postResample(pred = pred_usd, obs = testset.scaled$logprice)
performance
```

```
        RMSE     Rsquared          MAE
5848.2315253    0.7159979 3955.0920002
```

```
init_weights <- runif(67, min = -1, max = 1)
model_w <- neuralnet(logprice~., data = trainset.scaled, hidden = 3, err.fct = "sse", linear
plot(model_w, rep = "best")
```




```
pred_w_scaled <- compute(model_w, testset.scaled %>% select(-logprice))$net.result
pred_w_usd <- exp(pred_w_scaled * sigma + mu)

performance_w <- postResample(pred = pred_w_usd, obs = testset.scaled$logprice)
performance_w
```

```
        RMSE     Rsquared          MAE
5551.4706951    0.7914741 3906.2047833
```

```
rbind(performance, performance_w)
```

```
                RMSE  Rsquared      MAE
performance   5848.232 0.7159979 3955.092
performance_w 5551.471 0.7914741 3906.205
```

10.2 torch

11 Decision Tree and Random Forests

12 Generate Random Variables

Reference: Rizzo (2019) Chapter 3

12.1 Remember to Set the Random Seed

```
set.seed(42)
```

If the seed is not set by a default value, each time I render the webpage, the R codes will run, and different results may given. In some documents, the text will require to change in order to match the code output. In your project, set it whatever you like, 42, 2026, 3306, 3389, 65535, 114514, ...

12.2 Direct approach in R

```
# Uniform (0,1)
runif(n = 10, min = 0, max = 1)
```

```
 [1] 0.9148060 0.9370754 0.2861395 0.8304476 0.6417455 0.5190959 0.7365883
 [8] 0.1346666 0.6569923 0.7050648
```

```
# Bin(30, 0.6)
rbinom(n = 10, size = 30, prob = 0.6)
```

```
 [1] 18 16 14 20 18 14 13 21 18 18
```

```
# Poisson(7)
rpois(n = 5, lambda = 7)
```

```
[1] 11  4 14 12  4
```

```r
# Beta(5,3)
rbeta(n = 10, shape1 = 5, shape2 = 3)
```

```
 [1] 0.6180147 0.3383139 0.4158620 0.4378221 0.5261561 0.4187009 0.7703246
 [8] 0.5679070 0.6560498 0.5524067
```

12.3 Inverse CDF

For any continuous random variable, the random variable defined by its Cumulative Distribution Function (CDF) evaluated at itself follows a U(0,1) distribution.

Thus you can get the random value from the inverse of CDF.

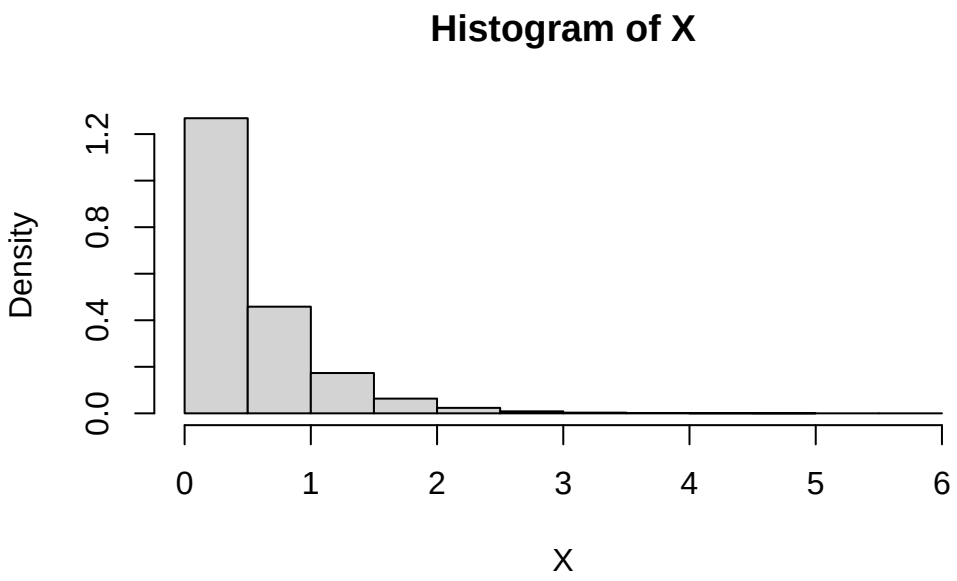
12.3.1 Example: Exp(2)

$$f(x) = 2e^{-2x} \quad F(x) = 1 - e^{-2x}$$

Thus, $U = F(x)$ we have $x = -\frac{1}{2}\log(1-U)$

- Generate $U \sim Uniform(0,1)$ (10,000) uniform random numbers
- Compute $X = -\frac{1}{2}\log U$

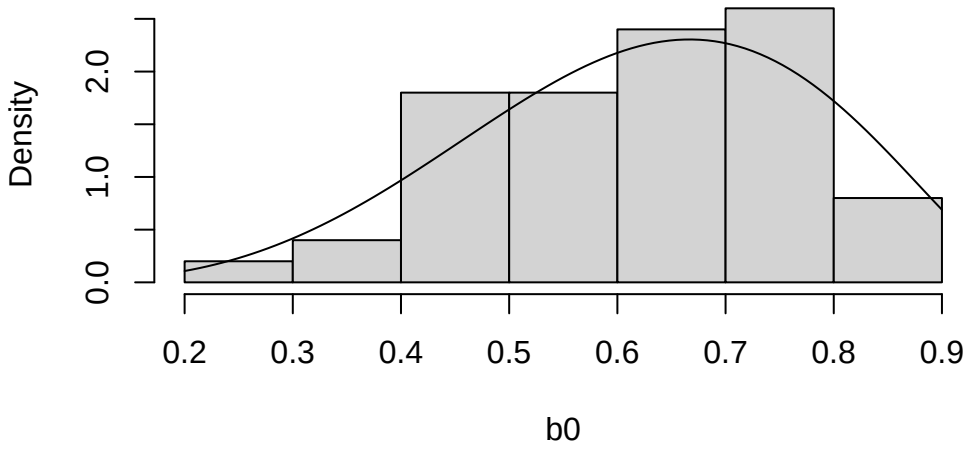
```r
m <- 100000
U <- runif(m)
X <- -1/2 * log(U)

hist(X, freq = F)
```



12.3.2 Example: Beta(5,3)

$$f(x) = \frac{1}{Beta(5,3)} x^4 (1-x)^2$$

```
ReverseCDF <- function(N) {
    return(qbeta(runif(N),5,3))
}

b0 <- ReverseCDF(50)
hist(b0, freq = F, ylim = c(0, 2.8))
curve(dbeta(x, 5, 3), add = T)
```



12.4 Acceptance-Rejection Method

You want to generate $X \sim f_X$ but it's difficult to do directly.

Assume you have $Y \sim g_Y$ which is easy to generate, and

$$\frac{f(t)}{g(t)} \leq c, \forall t \text{ s.t. } f(t) > 0$$

Then generate u from $U \sim U(0,1)$ and y from $Y \sim g_Y$, accept y if $u < \frac{f(t)}{cg(t)}$, otherwise drop it.

In some textbooks, $M = sup\frac{f(t)}{g(t)} < \infty \Rightarrow \frac{f(t)}{Mg(t)} \leq 1$. The M here is equal to the c mentioned earlier.

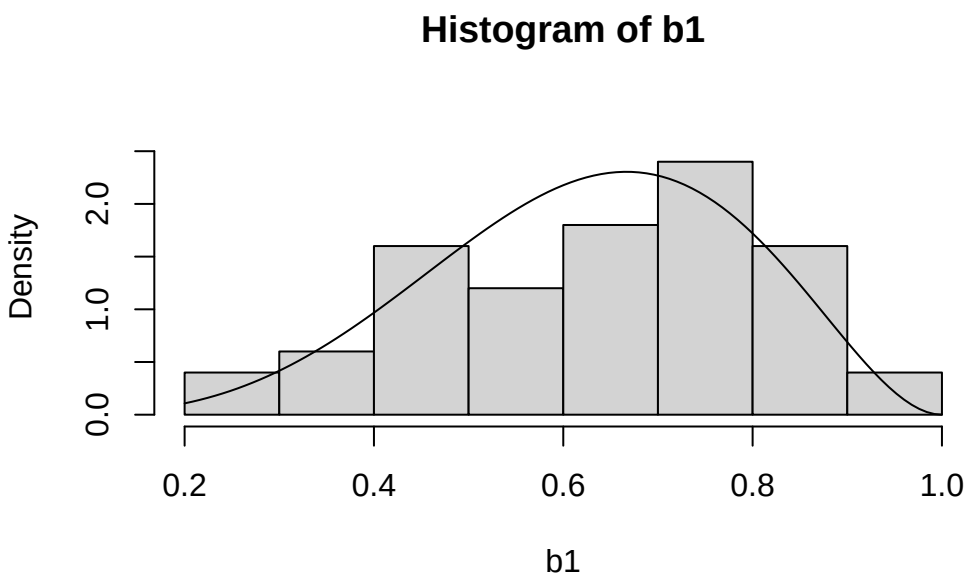
12.4.1 Example: Beta(5,3)

$$f(x) = \frac{1}{Beta(5,3)} x^4(1-x)^2$$

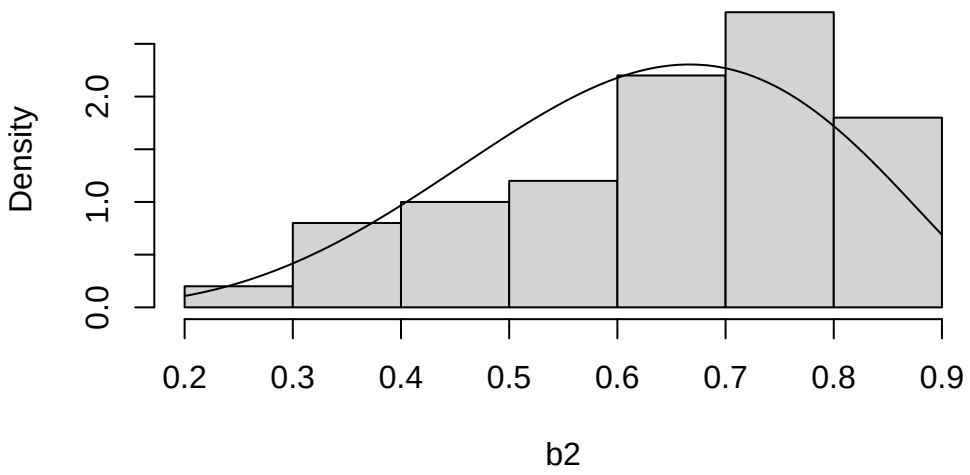
$$f'(x) = \frac{1}{Beta(5,3)}[4x^3(1-x)^2 - 2x^4(1-x)] = \frac{2x^3(1-x)(2-3x)}{Beta(5,3)}$$

Then we have $M = sup\frac{f(t)}{g(t)} = f(\frac{2}{3})$

```r
b1 <- rbeta(50, 5,3)
hist(b1, freq = F, ylim = c(0, 2.8))
curve(dbeta(x, 5, 3), add = T)
```




```r
b2 <- c()
M <- dbeta(2/3, 5,3)
while (length(b2) < 50){
    u1 <- runif(1)
    y <- runif(1)
    if(u1 < dbeta(y,5,3)/M){
        b2 <- c(b2,y)
    }
}
hist(b2, freq = F, ylim = c(0, 2.8))
curve(dbeta(x, 5, 3), add = T)
```



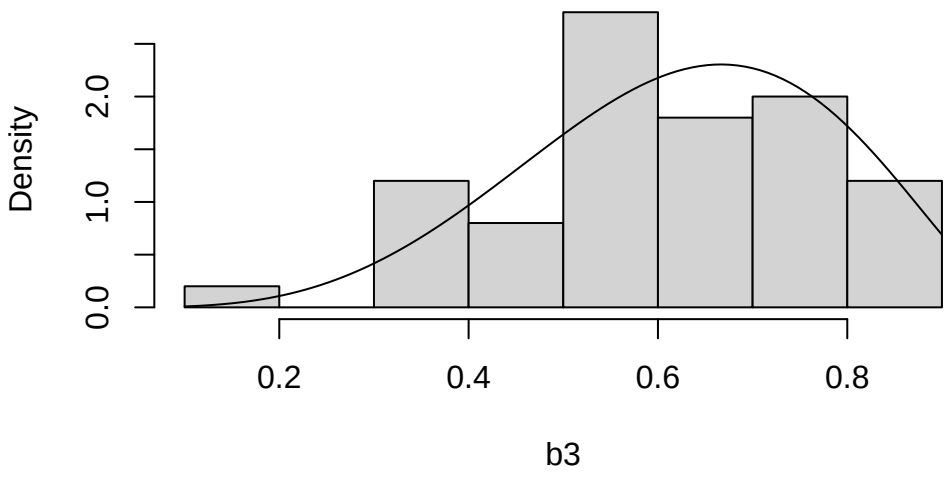
There're some performance weakness: - bad append operation `b2 <- c(b2,y)` - non-vectorized operation

Let's optimize!

```
N <- 50
M <- dbeta(2/3, 5,3)
N1 <- as.integer(N * M * 1.2)

u <- runif(N1)
y <- runif(N1)
b3 <- y[u < dbeta(y,5,3)/M]
print(paste(c("target is", N, ", and we generated", N1, "finally get", length(b3)), collapse
```

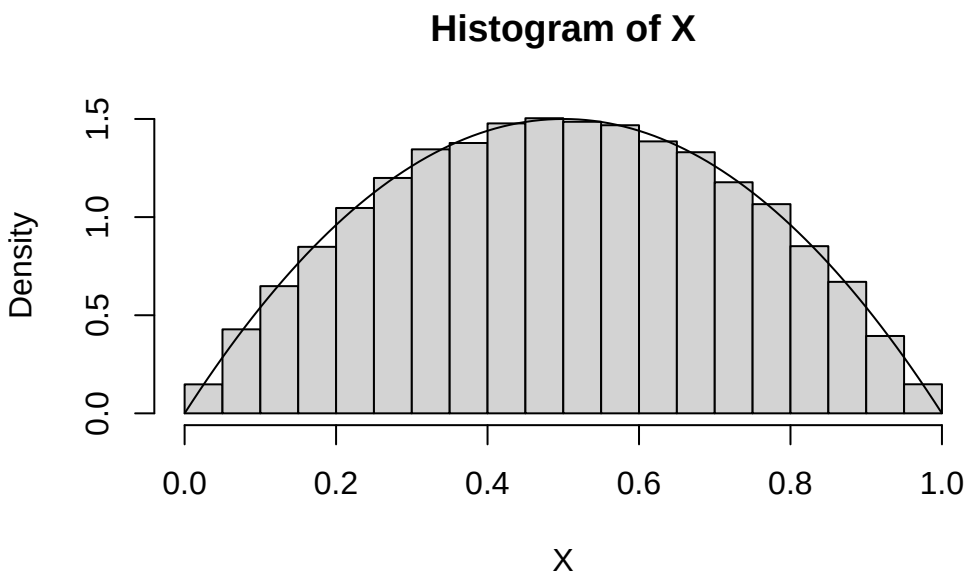
```
[1] "target is 50 , and we generated 138 finally get 54"
```

```
b3 <- b3[1:N]
hist(b3, freq = F, ylim = c(0, 2.8))
curve(dbeta(x, 5, 3), add = T)
```



12.4.2 Example: Beta(2,2)

```
M <- dbeta(0.5,2,2)
mb <- as.integer(m/M * 1.2)
U <- runif(mb)
Y <- runif(mb)

X <- Y[U < dbeta(Y,2,2)/M]

hist(X, freq=F)
curve(dbeta(x,2,2), add = T)
```



12.5 Transformation Method

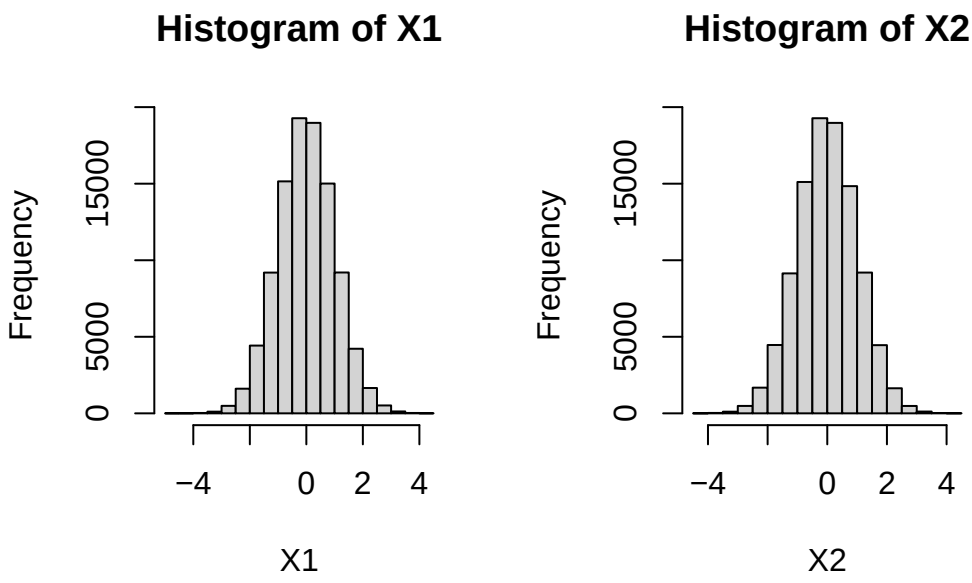
iid series $X_i \sim \Gamma(\alpha_i, \lambda)$
$\Longrightarrow \frac{\sum_{k=1}^{i} X_k}{\sum_{k=1}^{i+1} X_k} \sim Beta(\sum_{k=1}^{i} \alpha_k, \alpha_{i+1})$ and $\sum_{k=1}^{n} X_k \sim \Gamma(\sum_{i=1}^{n} \alpha_i, \lambda)$ independent

$$U, V \overset{iid}{\sim} U(0,1) \rightarrow \begin{cases} Z_1 &= \sqrt{-2\log U} cos 2\pi V \\ Z_2 &= \sqrt{-2\log V} cos 2\pi U \end{cases}, Z_1, Z_2 \overset{iid}{\sim} N(0,1)$$

```
m <- 100000
U1 <- runif(m)
U2 <- runif(m)
X1 <- sqrt(-2*log(U1)) * cos(2 * pi * U2)
X2 <- sqrt(-2*log(U1)) * sin(2 * pi * U2)

cor(X1, X2)
```

```
[1] 0.00217154
```

```
par(mfrow = c(1,2))
hist(X1)
hist(X2)
```




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

12.6 Benchmark

```
library(microbenchmark)

rejection_scalar <- function(N) {
    b2 <- numeric(N)
    M <- dbeta(2/3, 5, 3)
    count <- 0
    while (count < N) {
        u1 <- runif(1)
        y <- runif(1)
        if (u1 < dbeta(y, 5, 3) / M) {
            count <- count + 1
            b2[count] <- y
        }
    }
    return(b2)
}

rejection_vectorized <- function(N) {
    M <- dbeta(2/3, 5,3)
```

```
    N1 <- as.integer(N * M * 1.2)

    u <- runif(N1)
    y <- runif(N1)
    b3 <- y[u < dbeta(y,5,3)/M]
    return(b3[1:N])
}

results <- microbenchmark(
    Reverse = ReverseCDF(5000),
    Scalar_Loop = rejection_scalar(5000),
    Vectorized  = rejection_vectorized(5000),
    Built_in = rbeta(5000, 5, 3),
    times = 100
)
```

```
Warning in microbenchmark(Reverse = ReverseCDF(5000), Scalar_Loop =
rejection_scalar(5000), : less accurate nanosecond times to avoid potential
integer overflows
```

```
results
```

```
Unit: microseconds
        expr       min        lq      mean    median         uq       max neval
     Reverse  2033.887  2045.531  2070.634  2055.761  2064.5755  2575.825   100
 Scalar_Loop 11623.828 12087.927 12717.223 12335.855 12877.3210 32167.001   100
  Vectorized   713.523   726.315   754.318   735.950   747.4505  2378.533   100
    Built_in   246.820   252.888   255.510   254.405   257.5620   275.848   100
```

13 Monte Carlo

13.1 Monte Carlo Simulation of Intergation

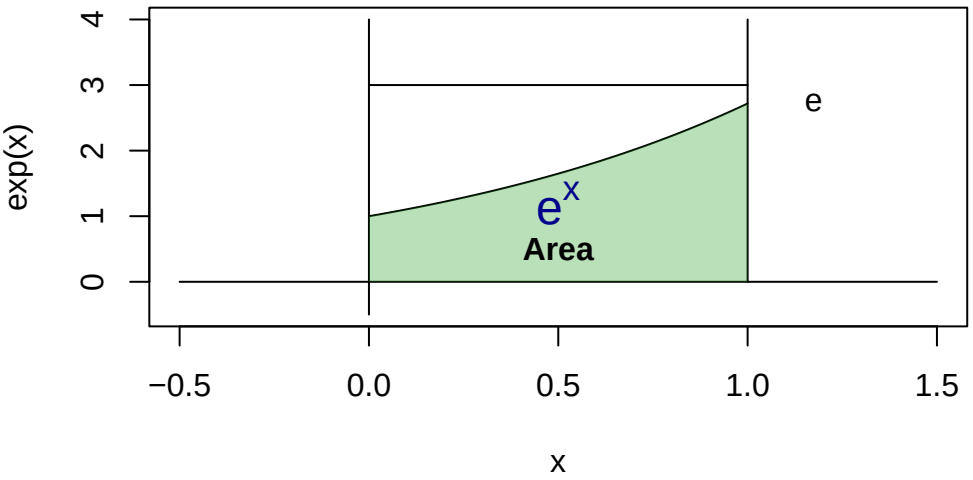
13.1.1 Example 1

Estimating this intergal by Monte Carlo:

$$\theta = \int_0^1 e^x dx = e - 1$$

The real integation is $e - 1$. But this is meaningless for those we can't integrate easily.

The idea of Monte Carlo is, assuming there's a area x from 0 to 1, y from 0 to M (a large number, greater than the maximum of function we're estimating)



We're calculting the green area in the plot, note that there's a probability of random point in the green area is equal to their ratio of area.

Find any M, i.e., here we let $M = 3 > e$

$$p = \frac{\text{Area of Green}}{\text{Area of Rectangular}} = \frac{S_G}{S_R}$$

Let X is a random variable with uniform distribution from 0 to 1, $Y = e^X$, then the Intergal $\hat{\theta} = S_G = pS_R = \mathbb{E}[\bar{Y}]$

```
m <- 1000
x <- runif(m)

cat(c("Monte Carlo", mean(exp(x)), '\n'))
```

```
Monte Carlo 1.71023613681694
```

```
cat(c("Real Intergation", exp(1)-1, '\n'))
```

```
Real Intergation 1.71828182845905
```

13.1.2 Example 2: Improper Integral

Estimating this intergal by Monte Carlo: (F(x) here is CDF of normal distribution)

$$\theta = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt = F(x)$$

Let's make transformation first:

$$\theta = (0.5 + \int_{0}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \int_{0}^{1} x e^{-\frac{(xy)^2}{2}} dy$$

```
y <- runif(m)
x <- 1
cat(c("Monte Carlo", 0.5 + x / sqrt(2 * pi) * mean(exp(-(x*y)^2/2)), '\n'))
```

```
Monte Carlo 0.840837027284758
```

```
cat(c("Real Intergation", pnorm(1), '\n'))
```

```
Real Intergation 0.841344746068543
```

Another idea is that, the sample EDF will converge to CDF. Thus, let's generate normal random variables.

```
y <- rnorm(m)
x <- 1

cat(c("Monte Carlo",  mean(y<=x), '\n'))
```

```
Monte Carlo 0.842
```

```
cat(c("Real Intergation", pnorm(1), '\n'))
```

```
Real Intergation 0.841344746068543
```

13.1.3 Variance

In the first example we use $Y = e^X$ to estimate $ = _{0}^{1} {e^x} dx $

As $X \sim U(0,1)$, it's unbiased estimation since $\mathbb{E}[Y] = \mathbb{E}[e^X] = e - 1 = \theta$.

The variance is $Var[Y] = \mathbb{E}[Y^2] - [\mathbb{E}[Y]]^2 = \frac{(e-1)(3-e)}{2} = 0.2420$

```
[1] "variance is, 0.242035607452765"
```

For the second example, we have two estimation:

$$X_1 \sim U(0,1) \Rightarrow Y_1 = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} x e^{-\frac{xX_1}{2}}$$

```
x_1 <- runif(m)
x <- 1
y_1 <- 1/2+ x / sqrt(2 * pi) * exp(-(x*x_1)^2/2)
var(y_1)
```

```
[1] 0.002347814
```

$$X_2 \sim N(0,1) \Rightarrow \frac{Y_2}{\sqrt{2\pi}} \sim B(1,p)$$

```r
x_2 <- rnorm(m)
x <- 1
y_2 <- (x_2<=x)

var(y_2)
```

```
[1] 0.129025
```

13.2 Variance Decrease

13.2.1 Antithetic Variates

$$U \sim U(0,1) \Rightarrow 1-U \sim U(0,1)$$

$$Z \sim N(0,1) \Rightarrow -Z \sim Z(0,1)$$

```r
x_1 <- runif(m/2)
x <- 1
y_1a <- 1/2+ x / sqrt(2 * pi)* ((exp(-(x*x_1)^2/2) + exp(-(x*(1-x_1))^2/2))/2)
var(y_1a)
```

```
[1] 8.304542e-05
```

```r
var(y_1)/var(y_1a)
```

```
[1] 28.27144
```

```r
x_2 <- rnorm(m/2)
x <- 1
y_2a <- ((x_2<=x) + (-x_2 <= x)/2)

var(y_2a)
```

```
[1] 0.1374138
```

```
var(y_2)/var(y_2a)
```

```
[1] 0.9389523
```

13.2.2 Control Variable

Estimating $g(U)$ with $Var[g(U)]$, optimizing by $h(U) = g(U) + c[f(U) - \mu]$

Then the variance become $Var[g[U]] + c^2 Var[f[U]] + 2cCOV(g(X), f(X))$, which is minimized when $c^* = -\frac{cov(g(X), f(X))}{Var[f(U)]}$

$$Var[h(U)] = Var[g(U)] - \frac{[COV(g(X), f(X))]^2}{[Var[f(U)]]^2}$$

Example: estimating $\theta = \int_0^1 e^x dx = e - 1$

```
m <- 1000
x <- runif(m)

y3a <- exp(x)
cat(c("Monte Carlo", mean(y3a), '\n'))
```

```
Monte Carlo 1.71288580562846
```

```
cat(c("Monte Carlo Variance", var(y3a), '\n'))
```

```
Monte Carlo Variance 0.255165094190046
```

We use $U \sim U(0, 1)$ and $g(U) = e^U$ as estimation. Let $f(U) = U$ be the controling variable:

```
gU <- exp(x)
fU <- x
c.star = - cov(gU, fU) / var(fU)

y3b <- gU + c.star * (fU - mean(fU))

cat(c("Control Variable", mean(y3b), '\n'))
```

```
Control Variable 1.71288580562846
```

```
cat(c("Control Variable Variance", var(y3b), '\n'))
```

```
Control Variable Variance 0.00395813289699567
```

```
cat(c("Variance Decrease Ratio", var(y3a)/var(y3b), '\n'))
```

```
Variance Decrease Ratio 64.4660249744829
```

13.2.3 Importance Sampling

References

Rizzo, Maria L. 2019. *Statistical Computing with r*. 2nd ed. CRC Press.