R Programming Swirl Exercises

Hariharan

5/11/2020

This PDF contains the code and output generated for swirl exercises in the course Find the R programming.Rmd in the same folder as this file to interact with the code and make changes for a better learninng experience

1. Basic Building Blocks

Assignment operator

```
x<-5+7
x
```

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

```
y<-x-3
y
```

```
## [1] 9
```

Concatenate/combine function

```
z<-c(1.1 , 9,3.14)
z
```

```
## [1] 1.10 9.00 3.14
```

```
c(z,555,z)
```

```
## [1]   1.10   9.00   3.14 555.00   1.10   9.00   3.14
```

Using numeric vectors in arithmetic expressions

```r
z*2+100
```

```
## [1] 102.20 118.00 106.28
```

```r
#using sqrt function
my_sqrt<-sqrt(z-1)
my_sqrt
```

```
## [1] 0.3162278 2.8284271 1.4628739
```

```r
my_div<-z/my_sqrt
my_div
```

```
## [1] 3.478505 3.181981 2.146460
```

Vector Recycling

```r
c(1,2,3,4)+c(0,10)
```

```
## [1]  1 12  3 14
```

```r
c(1,2,3,4) + c(0,10,100)
```

```
## Warning in c(1, 2, 3, 4) + c(0, 10, 100): longer object length is not a multiple
## of shorter object length
```

```
## [1]   1  12 103   4
```

2. Work space and files

N/A

3. Sequence of Numbers

using :

```r
pi:10
```

```
## [1] 3.141593 4.141593 5.141593 6.141593 7.141593 8.141593 9.141593
```

```r
15:1
```

```
##  [1] 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

using seq()

```
seq(1,20)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
seq(0,10,by=0.5)
```

```
##  [1]  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0
## [16]  7.5  8.0  8.5  9.0  9.5 10.0
```

```
my_seq<-seq(5,10,length=30)
length(my_seq)
```

```
## [1] 30
```

```
seq(along.with = my_seq)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30
```

```
seq_along(my_seq)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30
```

Repeating Sequences

```
rep(0, times = 40)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [39] 0 0
```

```
rep(c(0,1,2),times=10)
```

```
##  [1] 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
```

```
rep(c(0,1,2),each=10)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

4. Vectors

Basic operations

```
num_vect<-c(0.5,55,-10,6)
tf<-num_vect <1
tf
```

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

```
num_vect>=6
```

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

Character vectors

```
my_char <- c("My", "name", "is")
my_char
```

```
## [1] "My"   "name" "is"
```

```
#to join the vector
paste(my_char, collapse = " ")
```

```
## [1] "My name is"
```

```
#concatenation
my_name <- c(my_char, "Swirl")
my_name
```

```
## [1] "My"    "name"  "is"    "Swirl"
```

```
paste(my_name, collapse = " ")
```

```
## [1] "My name is Swirl"
```

```
paste("Hello", "world!", sep = " ")
```

```
## [1] "Hello world!"
```

```
paste(1:3, c("X", "Y", "Z"), sep = "")
```

```
## [1] "1X" "2Y" "3Z"
```

```
paste(LETTERS, 1:4, sep = "-")
```

```
##  [1] "A-1" "B-2" "C-3" "D-4" "E-1" "F-2" "G-3" "H-4" "I-1" "J-2" "K-3" "L-4"
## [13] "M-1" "N-2" "O-3" "P-4" "Q-1" "R-2" "S-3" "T-4" "U-1" "V-2" "W-3" "X-4"
## [25] "Y-1" "Z-2"
```

5. Missing Values

finding NA

```
x<-c(44,NA,5,NA)
x*3
```

```
## [1] 132  NA  15  NA
```

```
# creating a dataset with random NA's
y<-rnorm(1000)
z<-rep(NA,1000)
my_data<-sample(c(y,z),100)
sum(my_data)
```

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

```
#my_na contains bool values representing if that element is na or not
my_na<-is.na(my_data)
```

finding NaN

```
# instances where Nan is generated
0/0
```

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

```
Inf-Inf
```

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

6. Subsetting Vectors

forming random data set

```
x<-sample(c(rnorm(20),rep(NA,20)),40)
x[1:10]
```

```
##  [1]  0.61879074  0.33412379          NA  0.07020499 -2.56149886          NA
##  [7]          NA          NA          NA          NA
```

handling NA

```r
# extracting all NA
x[is.na(x)]
```

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

```r
#filtering missing values
y<-x[!is.na(x)]
y
```

```
##  [1]  0.61879074  0.33412379  0.07020499 -2.56149886 -0.87431377 -0.26736246
##  [7]  1.28425457 -0.06049454  0.05469658  0.07495401  2.52789082  0.86821367
## [13] -0.57356699 -0.25424920  0.13726985  0.03298453  0.96344735  0.08932958
## [19]  1.31660283 -0.20420837
```

more subsetting

```r
y[y>0]
```

```
##  [1] 0.61879074 0.33412379 0.07020499 1.28425457 0.05469658 0.07495401
##  [7] 2.52789082 0.86821367 0.13726985 0.03298453 0.96344735 0.08932958
## [13] 1.31660283
```

```r
x[!is.na(x) & x > 0]
```

```
##  [1] 0.61879074 0.33412379 0.07020499 1.28425457 0.05469658 0.07495401
##  [7] 2.52789082 0.86821367 0.13726985 0.03298453 0.96344735 0.08932958
## [13] 1.31660283
```

```r
x[c(3, 5, 7)]
```

```
## [1]        NA -2.561499        NA
```

```r
x[c(-2, -10)]
```

```
##  [1]  0.61879074          NA  0.07020499 -2.56149886          NA          NA
##  [7]          NA          NA          NA          NA -0.87431377          NA
## [13]          NA          NA          NA -0.26736246          NA          NA
## [19]  1.28425457 -0.06049454  0.05469658  0.07495401  2.52789082          NA
## [25]  0.86821367 -0.57356699          NA -0.25424920          NA  0.13726985
## [31]  0.03298453  0.96344735          NA          NA  0.08932958  1.31660283
## [37] -0.20420837          NA
```

```r
 x[-c(2, 10)]
```

```
##  [1]  0.61879074          NA  0.07020499 -2.56149886          NA          NA
##  [7]          NA          NA          NA          NA -0.87431377          NA
## [13]          NA          NA          NA -0.26736246          NA          NA
## [19]  1.28425457 -0.06049454  0.05469658  0.07495401  2.52789082          NA
## [25]  0.86821367 -0.57356699          NA -0.25424920          NA  0.13726985
## [31]  0.03298453  0.96344735          NA          NA  0.08932958  1.31660283
## [37] -0.20420837          NA
```

vectors with named elements

```
vect <- c(foo = 11, bar = 2, norf = NA)
vect
```

```
##  foo  bar norf
##   11    2   NA
```

```
names(vect)
```

```
## [1] "foo"  "bar"  "norf"
```

```
#adding names
vect2 <- c(11, 2, NA)
names(vect2) <- c("foo", "bar", "norf")
identical(vect, vect2)
```

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

```
#filtering
vect["bar"]
```

```
## bar
##   2
```

```
vect[c("foo", "bar")]
```

```
## foo bar
##  11   2
```

7. Matrices and Data Frames

```
my_vector <- 1:20
dim(my_vector)
```

```
## NULL
```

```
dim(my_vector) <- c(4, 5)
dim(my_vector) <- c(4, 5)
 my_vector
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20
```

```
 my_matrix <- my_vector
```

Using the matrix function

```
my_matrix2 <- matrix(1:20, nrow=4, ncol=5)
```

Labeling

```
patients <- c("Bill", "Gina", "Kelly", "Sean")
cbind(patients, my_matrix)
```

```
##      patients
## [1,] "Bill"   "1" "5" "9"  "13" "17"
## [2,] "Gina"   "2" "6" "10" "14" "18"
## [3,] "Kelly"  "3" "7" "11" "15" "19"
## [4,] "Sean"   "4" "8" "12" "16" "20"
```

Constructing a data frame

```
my_data <- data.frame(patients, my_matrix)
my_data
```

```
##   patients X1 X2 X3 X4 X5
## 1     Bill  1  5  9 13 17
## 2     Gina  2  6 10 14 18
## 3    Kelly  3  7 11 15 19
## 4     Sean  4  8 12 16 20
```

```
cnames <- c("patient", "age", "weight", "bp", "rating", "test")
colnames(my_data) <- cnames
my_data
```

```
##   patient age weight bp rating test
## 1    Bill   1      5  9     13   17
## 2    Gina   2      6 10     14   18
## 3   Kelly   3      7 11     15   19
## 4    Sean   4      8 12     16   20
```

8. Logic

Some basic operations

```
TRUE == TRUE
```

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

```
 7 == 9
```

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

```
 7 > 9
```

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

```
 7 < 9
```

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

```
 7>=9
```

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

```
 7!=9
```

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

You can use the `&` operator to evaluate AND across a vector. The `&&` version of AND only evaluates the first member of a vector.

```
TRUE && c(TRUE, FALSE, FALSE)
```

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

```
TRUE & c(TRUE, FALSE, FALSE)
```

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

```
TRUE | c(TRUE, FALSE, FALSE)
```

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

```
TRUE || c(TRUE, FALSE, FALSE)
```

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

Logical functions in R

```r
isTRUE(6 > 4)
```

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

```r
xor(5 == 6, !FALSE)
```

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

```r
ints <- sample(10)
ints > 5
```

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

```r
which(ints > 7)
```

```
## [1] 1 3 8
```

```r
any(ints < 0)
```

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

9. Functions

N/A

10. lapply() and sapply()

code to download the data set for this section

```r
y<- "https://archive.ics.uci.edu/ml/machine-learning-databases/flags/flag.data"

flags <- read.table(y, header = FALSE, sep = ",")
names(flags) <- c("country", "landmass", "zone", "area", "population", "language",
              "religion", "bars", "stripes", "colors", "red", "green", "blue",
              "gold", "white", "black", "orange","mainhue", "circles", "crosses",
              "saltires", "quarters", "sunstars", "crescents", "triangle", "icon",
              "animate", "text", "topleft", "botright")
```

```r
head(flags)
```

```
##          country landmass zone area population language religion bars stripes
## 1    Afghanistan        5    1  648         16       10        2    0       3
## 2        Albania        3    1   29          3        6        6    0       0
## 3        Algeria        4    1 2388         20        8        2    2       0
```

```
## 4 American-Samoa        6    3    0          0        1        1    0       0
## 5        Andorra        3    1    0          0        6        0    3       0
## 6         Angola        4    2 1247          7       10        5    0       2
##   colors red green blue gold white black orange mainhue circles crosses
## 1      5   1     1    0    1     1     1      0   green       0       0
## 2      3   1     0    0    1     0     1      0     red       0       0
## 3      3   1     1    0    0     1     0      0   green       0       0
## 4      5   1     0    1    1     1     0      1    blue       0       0
## 5      3   1     0    1    1     0     0      0    gold       0       0
## 6      3   1     0    0    1     0     1      0     red       0       0
##   saltires quarters sunstars crescents triangle icon animate text topleft
## 1        0        0        1         0        0    1       0    0   black
## 2        0        0        1         0        0    0       1    0     red
## 3        0        0        1         1        0    0       0    0   green
## 4        0        0        0         0        1    1       1    0    blue
## 5        0        0        0         0        0    0       0    0    blue
## 6        0        0        1         0        0    1       0    0     red
##   botright
## 1    green
## 2      red
## 3    white
## 4      red
## 5      red
## 6    black
```

```
dim(flags)
```

```
## [1] 194  30
```

```
class(flags)
```

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

The lapply() function takes a list as input, applies a function to each element of the list, then returns a list of the same length as the original one. Since a data frame is really just a list of vectors (you can see this with as.list(flags)), we can use lapply() to apply the class() function to each column of the flags dataset.

```
cls_list <- lapply(flags, class)
cls_list
```

```
## $country
## [1] "factor"
##
## $landmass
## [1] "integer"
##
## $zone
## [1] "integer"
##
## $area
## [1] "integer"
##
```

```
## $population
## [1] "integer"
##
## $language
## [1] "integer"
##
## $religion
## [1] "integer"
##
## $bars
## [1] "integer"
##
## $stripes
## [1] "integer"
##
## $colors
## [1] "integer"
##
## $red
## [1] "integer"
##
## $green
## [1] "integer"
##
## $blue
## [1] "integer"
##
## $gold
## [1] "integer"
##
## $white
## [1] "integer"
##
## $black
## [1] "integer"
##
## $orange
## [1] "integer"
##
## $mainhue
## [1] "factor"
##
## $circles
## [1] "integer"
##
## $crosses
## [1] "integer"
##
## $saltires
## [1] "integer"
##
## $quarters
## [1] "integer"
##
```

```
## $sunstars
## [1] "integer"
##
## $crescents
## [1] "integer"
##
## $triangle
## [1] "integer"
##
## $icon
## [1] "integer"
##
## $animate
## [1] "integer"
##
## $text
## [1] "integer"
##
## $topleft
## [1] "factor"
##
## $botright
## [1] "factor"
```

```
as.character(cls_list)
```

```
##  [1] "factor"  "integer" "integer" "integer" "integer" "integer" "integer"
##  [8] "integer" "integer" "integer" "integer" "integer" "integer" "integer"
## [15] "integer" "integer" "integer" "factor"  "integer" "integer" "integer"
## [22] "integer" "integer" "integer" "integer" "integer" "integer" "integer"
## [29] "factor"  "factor"
```

if we use saaply the result is simplified in this case list will be character vector automatically

```
cls_vect <- sapply(flags, class)
class(cls_vect)
```

```
## [1] "character"
```

To see how many flags have orange in them we use

```
sum(flags$orange)
```

```
## [1] 26
```

Now we want to repeat this operation for each of the colors recorded in the dataset.

First, use flag_colors <- flags[, 11:17] to extract the columns containing the color data and store them in a new data frame called flag_colors. (Note the comma before 11:17. This subsetting command tells R that we want all rows, but only columns 11 through 17.)

```
flag_colors <- flags[, 11:17]
head(flag_colors)
```

```
##   red green blue gold white black orange
## 1   1     1    0    1     1     1      0
## 2   1     0    0    1     0     1      0
## 3   1     1    0    0     1     0      0
## 4   1     0    1    1     1     0      1
## 5   1     0    1    1     0     0      0
## 6   1     0    0    1     0     1      0
```

```
lapply(flag_colors, sum)
```

```
## $red
## [1] 153
##
## $green
## [1] 91
##
## $blue
## [1] 99
##
## $gold
## [1] 91
##
## $white
## [1] 146
##
## $black
## [1] 52
##
## $orange
## [1] 26
```

```
sapply(flag_colors, sum)
```

```
##    red  green   blue   gold  white  black orange
##    153     91     99     91    146     52     26
```

To find mean use of each color

```
sapply(flag_colors, mean)
```

```
##       red     green      blue      gold     white     black    orange
## 0.7886598 0.4690722 0.5103093 0.4690722 0.7525773 0.2680412 0.1340206
```

```
flag_shapes <- flags[, 19:23]

lapply(flag_shapes, range)
```

```
## $circles
## [1] 0 4
##
## $crosses
## [1] 0 2
##
## $saltires
## [1] 0 1
##
## $quarters
## [1] 0 4
##
## $sunstars
## [1]  0 50
```

```
shape_mat <- sapply(flag_shapes, range)
 shape_mat
```

```
##      circles crosses saltires quarters sunstars
## [1,]       0       0        0        0        0
## [2,]       4       2        1        4       50
```

using unique()

```
unique(c(3, 4, 5, 5, 5, 6, 6))
```

```
## [1] 3 4 5 6
```

```
unique_vals <- lapply(flags, unique)
unique_vals
```

```
## $country
##   [1] Afghanistan             Albania                 Algeria
##   [4] American-Samoa          Andorra                 Angola
##   [7] Anguilla                Antigua-Barbuda         Argentina
##  [10] Argentine               Australia               Austria
##  [13] Bahamas                 Bahrain                 Bangladesh
##  [16] Barbados                Belgium                 Belize
##  [19] Benin                   Bermuda                 Bhutan
##  [22] Bolivia                 Botswana                Brazil
##  [25] British-Virgin-Isles    Brunei                  Bulgaria
##  [28] Burkina                 Burma                   Burundi
##  [31] Cameroon                Canada                  Cape-Verde-Islands
##  [34] Cayman-Islands          Central-African-Republic Chad
##  [37] Chile                   China                   Colombia
##  [40] Comorro-Islands         Congo                   Cook-Islands
##  [43] Costa-Rica              Cuba                    Cyprus
##  [46] Czechoslovakia          Denmark                 Djibouti
##  [49] Dominica                Dominican-Republic      Ecuador
##  [52] Egypt                   El-Salvador             Equatorial-Guinea
##  [55] Ethiopia                Faeroes                 Falklands-Malvinas
##  [58] Fiji                    Finland                 France
```

```
##  [61] French-Guiana           French-Polynesia        Gabon
##  [64] Gambia                  Germany-DDR             Germany-FRG
##  [67] Ghana                   Gibraltar               Greece
##  [70] Greenland               Grenada                 Guam
##  [73] Guatemala               Guinea                  Guinea-Bissau
##  [76] Guyana                  Haiti                   Honduras
##  [79] Hong-Kong               Hungary                 Iceland
##  [82] India                   Indonesia               Iran
##  [85] Iraq                    Ireland                 Israel
##  [88] Italy                   Ivory-Coast             Jamaica
##  [91] Japan                   Jordan                  Kampuchea
##  [94] Kenya                   Kiribati                Kuwait
##  [97] Laos                    Lebanon                 Lesotho
## [100] Liberia                 Libya                   Liechtenstein
## [103] Luxembourg              Malagasy                Malawi
## [106] Malaysia                Maldive-Islands         Mali
## [109] Malta                   Marianas                Mauritania
## [112] Mauritius               Mexico                  Micronesia
## [115] Monaco                  Mongolia                Montserrat
## [118] Morocco                 Mozambique              Nauru
## [121] Nepal                   Netherlands             Netherlands-Antilles
## [124] New-Zealand             Nicaragua               Niger
## [127] Nigeria                 Niue                    North-Korea
## [130] North-Yemen             Norway                  Oman
## [133] Pakistan                Panama                  Papua-New-Guinea
## [136] Parguay                 Peru                    Philippines
## [139] Poland                  Portugal                Puerto-Rico
## [142] Qatar                   Romania                 Rwanda
## [145] San-Marino              Sao-Tome                Saudi-Arabia
## [148] Senegal                 Seychelles              Sierra-Leone
## [151] Singapore               Soloman-Islands         Somalia
## [154] South-Africa            South-Korea             South-Yemen
## [157] Spain                   Sri-Lanka               St-Helena
## [160] St-Kitts-Nevis          St-Lucia                St-Vincent
## [163] Sudan                   Surinam                 Swaziland
## [166] Sweden                  Switzerland             Syria
## [169] Taiwan                  Tanzania                Thailand
## [172] Togo                    Tonga                   Trinidad-Tobago
## [175] Tunisia                 Turkey                  Turks-Cocos-Islands
## [178] Tuvalu                  UAE                     Uganda
## [181] UK                      Uruguay                 US-Virgin-Isles
## [184] USA                     USSR                    Vanuatu
## [187] Vatican-City            Venezuela               Vietnam
## [190] Western-Samoa           Yugoslavia              Zaire
## [193] Zambia                  Zimbabwe
## 194 Levels: Afghanistan Albania Algeria American-Samoa Andorra ... Zimbabwe
##
## $landmass
## [1] 5 3 4 6 1 2
##
## $zone
## [1] 1 3 2 4
##
## $area
```

```
##   [1]   648    29  2388     0  1247  2777  7690    84    19     1   143    31
##  [13]    23   113    47  1099   600  8512     6   111   274   678    28   474
##  [25]  9976     4   623  1284   757  9561  1139     2   342    51   115     9
##  [37]   128    43    22    49   284  1001    21  1222    12    18   337   547
##  [49]    91   268    10   108   249   239   132  2176   109   246    36   215
##  [61]   112    93   103  3268  1904  1648   435    70   301   323    11   372
##  [73]    98   181   583   236    30  1760     3   587   118   333  1240  1031
##  [85]  1973  1566   447   783   140    41  1267   925   121   195   324   212
##  [97]   804    76   463   407  1285   300   313    92   237    26  2150   196
## [109]    72   637  1221    99   288   505    66  2506    63    17   450   185
## [121]   945   514    57     5   164   781   245   178  9363 22402    15   912
## [133]   256   905   753   391
##
## $population
##  [1]   16    3   20    0    7   28   15    8   90   10    1    6  119    9   35
## [16]    4   24    2   11 1008    5   47   31   54   17   61   14  684  157   39
## [31]   57  118   13   77   12   56   18   84   48   36   22   29   38   49   45
## [46]  231  274   60
##
## $language
##  [1] 10  6  8  1  2  4  3  5  7  9
##
## $religion
## [1] 2 6 1 0 5 3 4 7
##
## $bars
## [1] 0 2 3 1 5
##
## $stripes
##  [1]  3  0  2  1  5  9 11 14  4  6 13  7
##
## $colors
## [1] 5 3 2 8 6 4 7 1
##
## $red
## [1] 1 0
##
## $green
## [1] 1 0
##
## $blue
## [1] 0 1
##
## $gold
## [1] 1 0
##
## $white
## [1] 1 0
##
## $black
## [1] 1 0
##
## $orange
## [1] 0 1
```

```
##
## $mainhue
## [1] green  red    blue   gold   white  orange black  brown
## Levels: black blue brown gold green orange red white
##
## $circles
## [1] 0 1 4 2
##
## $crosses
## [1] 0 1 2
##
## $saltires
## [1] 0 1
##
## $quarters
## [1] 0 1 4
##
## $sunstars
##  [1]  1  0  6 22 14  3  4  5 15 10  7  2  9 50
##
## $crescents
## [1] 0 1
##
## $triangle
## [1] 0 1
##
## $icon
## [1] 1 0
##
## $animate
## [1] 0 1
##
## $text
## [1] 0 1
##
## $topleft
## [1] black  red    green  blue   white  orange gold
## Levels: black blue gold green orange red white
##
## $botright
## [1] green  red    white  black  blue   gold   orange brown
## Levels: black blue brown gold green orange red white
```

```
sapply(unique_vals, length)
```

```
##    country   landmass       zone       area population   language   religion
##        194          6          4        136         48         10          8
##       bars    stripes     colors        red      green       blue       gold
##          5         12          8          2          2          2          2
##      white      black     orange    mainhue    circles    crosses   saltires
##          2          2          2          8          4          3          2
##   quarters   sunstars  crescents   triangle       icon    animate       text
##          3         14          2          2          2          2          2
##    topleft   botright
```

```
##         7         8
```

11. vapply and tapply

in vapply we can specify the output type instead of letting R guess the format

```
vapply(flags, class, character(1))
```

```
##    country   landmass       zone       area population   language   religion
##   "factor"  "integer"  "integer"  "integer"  "integer"  "integer"  "integer"
##       bars    stripes     colors        red      green       blue       gold
##  "integer"  "integer"  "integer"  "integer"  "integer"  "integer"  "integer"
##      white      black     orange    mainhue    circles    crosses   saltires
##  "integer"  "integer"  "integer"   "factor"  "integer"  "integer"  "integer"
##   quarters   sunstars  crescents   triangle       icon    animate       text
##  "integer"  "integer"  "integer"  "integer"  "integer"  "integer"  "integer"
##    topleft   botright
##   "factor"   "factor"
```

using tapply()

```
table(flags$landmass)
```

```
##
##  1  2  3  4  5  6
## 31 17 35 52 39 20
```

```
table(flags$animate)
```

```
##
##   0   1
## 155  39
```

If you take the arithmetic mean of a bunch of 0s and 1s, you get the proportion of 1s. Use tapply(flags$animate, flags$landmass, mean) to apply the mean function to the 'animate' variable separately for each of the six landmass groups, thus giving us the proportion of flags containing an animate image WITHIN each landmass group.

```
tapply(flags$animate, flags$landmass, mean)
```

```
##         1         2         3         4         5         6
## 0.4193548 0.1764706 0.1142857 0.1346154 0.1538462 0.3000000
```

Similarly, we can look at a summary of population values (in round millions) for countries with and without the color red on their flag with tapply(flags$population, flags$red, summary).

```
tapply(flags$population, flags$red, summary)
```

```
## $`0`
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    0.00    3.00   27.63    9.00  684.00
##
## $`1`
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0     0.0     4.0    22.1    15.0  1008.0
```

12. Looking at data

loading a sample dataset for this lesson

```
data("mtcars")
```

Some functions to look at data

```
dim(mtcars)
```

```
## [1] 32 11
```

```
nrow(mtcars)
```

```
## [1] 32
```

```
ncol(mtcars)
```

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

```
object.size(mtcars)
```

```
## 7208 bytes
```

```
names(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```r
tail(mtcars,15)
```

```
##                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Fiat 128         32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic      30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla   33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona    21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger 15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin      15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28       13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9        27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2    26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa     30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L   15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino     19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora    15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E       21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```r
summary(mtcars)
```

```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

```r
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
```

```
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

13. Simulation

Let's simulate rolling four six-sided dice:

```
sample(1:6, 4, replace = TRUE)
```

```
## [1] 4 4 2 4
```

```
sample(1:20, 10)
```

```
##  [1] 17 18  9 14 19 16 13  8  5 12
```

```
LETTERS
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

Let the value 0 represent tails and the value 1 represent heads. Use sample() to draw a sample of size 100 from the vector c(0,1), with replacement. Since the coin is unfair, we must attach specific probabilities to the values 0 (tails) and 1 (heads) with a fourth argument, prob = c(0.3, 0.7). Assign the result to a new variable called flips.

```
flips <- sample(c(0,1), 100, replace = TRUE, prob = c(0.3, 0.7))
flips
```

```
##   [1] 1 1 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1
##  [38] 1 0 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0
##  [75] 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 0 0 1 1 0 1 1 1 1 1
```

```
sum(flips)
```

```
## [1] 76
```

A binomial random variable represents the number of 'successes' (heads) in a given number of independent 'trials' (coin flips). Therefore, we can generate a single random variable that represents the number of heads in 100 flips of our unfair coin using rbinom(1, size = 100, prob = 0.7). Note that you only specify the probability of 'success' (heads) and NOT the probability of 'failure' (tails).

```
rbinom(1, size = 100, prob = 0.7)
```

```
## [1] 72
```

```
flips2 <- rbinom(100, size = 1, prob = 0.7)
flips2
```

```
##   [1] 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1
##  [38] 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 1
##  [75] 1 1 1 1 0 1 0 1 0 1 1 0 1 1 0 0 1 1 1 1 1 0 1 1 1 1
```

```
sum(flips2)
```

```
## [1] 77
```

The standard normal distribution has mean 0 and standard deviation 1. As you can see under the 'Usage' section in the documentation, the default values for the 'mean' and 'sd' arguments to rnorm() are 0 and 1, respectively. Thus,rnorm(10) will generate 10 random numbers from a standard normal distribution.

```
rnorm(10)
```

```
##  [1] -0.4596784  1.2874490 -1.4331321 -1.0792075 -0.7617486  1.5838202
##  [7]  0.2242924 -0.1231211 -1.1694740  1.1752922
```

```
# with a mean of 100 and a standard deviation of 25
rnorm(10, 100, 25)
```

```
##  [1] 135.53789  73.05536 103.87626 134.09271  99.68665 103.70117  63.98229
##  [8] 140.61743 137.63932  91.11030
```

what if we want to simulate 100 *groups* of random numbers, each containing 5 values generated from a Poisson distribution with mean 10? Let's start with one group of 5 numbers, then I'll show you how to repeat the operation 100 times in a convenient and compact way.

```
rpois(5, 10)
```

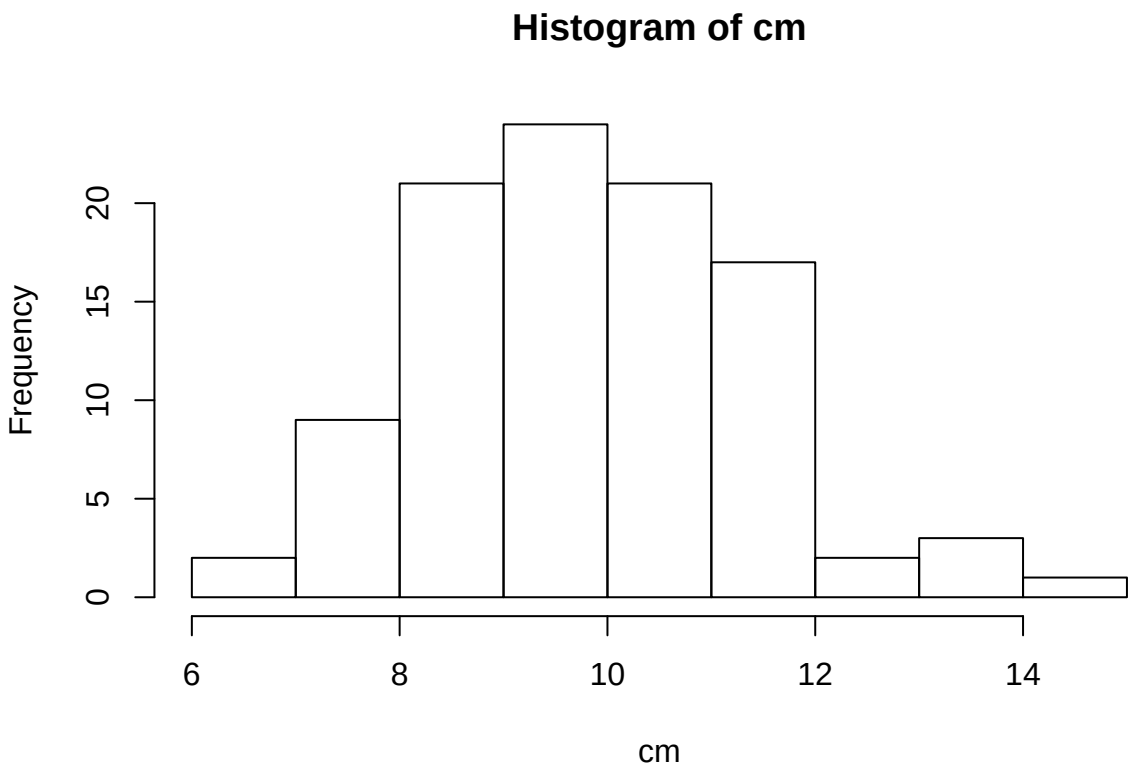
```
## [1] 12  6  6 11  6
```

```
my_pois <- replicate(100, rpois(5, 10))
my_pois
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]    8    7    8   18   14    6   13    6   11    16     6     7    15    11
## [2,]    7    7   16   12   12   10   12   15   11    13     6     6    15     8
## [3,]    2   13    4    7    9    5   10   10    9     6    10    11     7    14
## [4,]    7    4   10    8    6    8   10    9   12    12     8     7     9     3
## [5,]   10   12   10    8   17    3   13    4    4    10    11     7    13    13
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
## [1,]     7     9     7    12     7     7     6     8     9    14     9     9
## [2,]    12    11     9    12    22    10    10    16     6     7    16     9
## [3,]     5    13    12     8    14     6     6    16    11    12     2     7
## [4,]    12     7     8    15     5    14    10    12    11    11    11     6
## [5,]    11     4     7     7     8     8    11     9    10     6    12    11
```

```
##      [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
## [1,]    12     8     9    11     6     8    12    15     5    11     5    10
## [2,]    11    13    11     5    11    15     9    13    17    12    10    10
## [3,]     5    13     8     9    15    10     8    16    10     7     6     6
## [4,]    16     8     7    11     9    15    10    12    12    12    10     5
## [5,]    12     7     8    11     7    10     8    10    11     7     7    13
##      [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]
## [1,]    10     6    11    12    13    15    18     5    12    13     8    11
## [2,]    14    10     9     8     7    14     4    14    11    11     8     7
## [3,]    11     9     9     5    16     7    13    12    12    13     7    14
## [4,]     6     5    11    17     6    14     5     6    12     5     7     7
## [5,]     9    13     8    10    10     6    13    10    10     5     9    12
##      [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62]
## [1,]    12    16     4     7     8    10    11     8    13    12    10     6
## [2,]    20    15     9     6     9    14     9     9     5    11     7    10
## [3,]     7    10    13    16     7    11    12    13     9    14    15    12
## [4,]     7    11    15    13     9     9     3     4    12     4     9    10
## [5,]     6    14    13     7     4    11     6    11     8    12    14     7
##      [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74]
## [1,]    11    11    11     8    12    16    15    12    13     8    14     7
## [2,]     8    14    11    18     9     6    12     9     8    10    12    11
## [3,]    13    10     9     6     9     6    10     9     8     9    11    16
## [4,]    10    16    10    17    13    15    10    14     7     7     9     7
## [5,]    10     8    12    16     4    10    12    10     8     6     8    12
##      [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84] [,85] [,86]
## [1,]    11     9    10    11     8    13    10    11     6     9    11     7
## [2,]    12    10     9    11     7    13    11    10     9    11     8    10
## [3,]    12     9    10     7    11     9    12     8     9    11    12    13
## [4,]    10     4    10     9    11    19     6     8     9     5     8     3
## [5,]    11     9    14     9    10    13    15    15    11     9     6     9
##      [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98]
## [1,]     7    10     2    11     4    13    10    14    10    17     7     7
## [2,]     8    16    11    12     6     7    15     8    11    16     7     9
## [3,]     7    12    11     9    12    14    13    13    10    15     6    11
## [4,]     7    14     6     9     8     9    13    12    15    11    11    14
## [5,]     9     7     8     9     9     5     7    12    12    15    14     7
##      [,99] [,100]
## [1,]     8      7
## [2,]    13      7
## [3,]    17     10
## [4,]     9      8
## [5,]    13      7
```

```
cm <- colMeans(my_pois)
hist(cm)
```



14. Dates and Times

Dates

```
d1 <- Sys.Date()
unclass(d1)
```

```
## [1] 18393
```

```
d1
```

```
## [1] "2020-05-11"
```

```
d2 <- as.Date("1969-01-01")
unclass(d2)
```

```
## [1] -365
```

Times

```
t1 <- Sys.time()
t1
```

```
## [1] "2020-05-11 21:25:35 IST"
```

```
class(t1)
```

```
## [1] "POSIXct" "POSIXt"
```

```
unclass(t1)
```

```
## [1] 1589212535
```

```
t2 <- as.POSIXlt(Sys.time())
t2
```

```
## [1] "2020-05-11 21:25:35 IST"
```

```
unclass(t2)
```

```
## $sec
## [1] 35.11376
##
## $min
## [1] 25
##
## $hour
## [1] 21
##
## $mday
## [1] 11
##
## $mon
## [1] 4
##
## $year
## [1] 120
##
## $wday
## [1] 1
##
## $yday
## [1] 131
##
## $isdst
## [1] 0
##
## $zone
## [1] "IST"
##
```

```
## $gmtoff
## [1] 19800
##
## attr(,"tzone")
## [1] ""      "IST"   "+0630"
```

```
str(unclass(t2))
```

```
## List of 11
##  $ sec   : num 35.1
##  $ min   : int 25
##  $ hour  : int 21
##  $ mday  : int 11
##  $ mon   : int 4
##  $ year  : int 120
##  $ wday  : int 1
##  $ yday  : int 131
##  $ isdst : int 0
##  $ zone  : chr "IST"
##  $ gmtoff: int 19800
##  - attr(*, "tzone")= chr [1:3] "" "IST" "+0630"
```

```
t2$min
```

```
## [1] 25
```

other functions to work with date and time

```
weekdays(d1)
```

```
## [1] "Monday"
```

```
months(t1)
```

```
## [1] "May"
```

```
quarters(t2)
```

```
## [1] "Q2"
```

Processing from string

```
t3 <- "October 17, 1986 08:24"
t4 <- strptime(t3, "%B %d, %Y %H:%M")
t4
```

```
## [1] "1986-10-17 08:24:00 IST"
```

```
Sys.time() > t1
```

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

```
Sys.time() - t1
```

```
## Time difference of 0.04490709 secs
```

```
difftime(Sys.time(), t1, units = 'days')
```

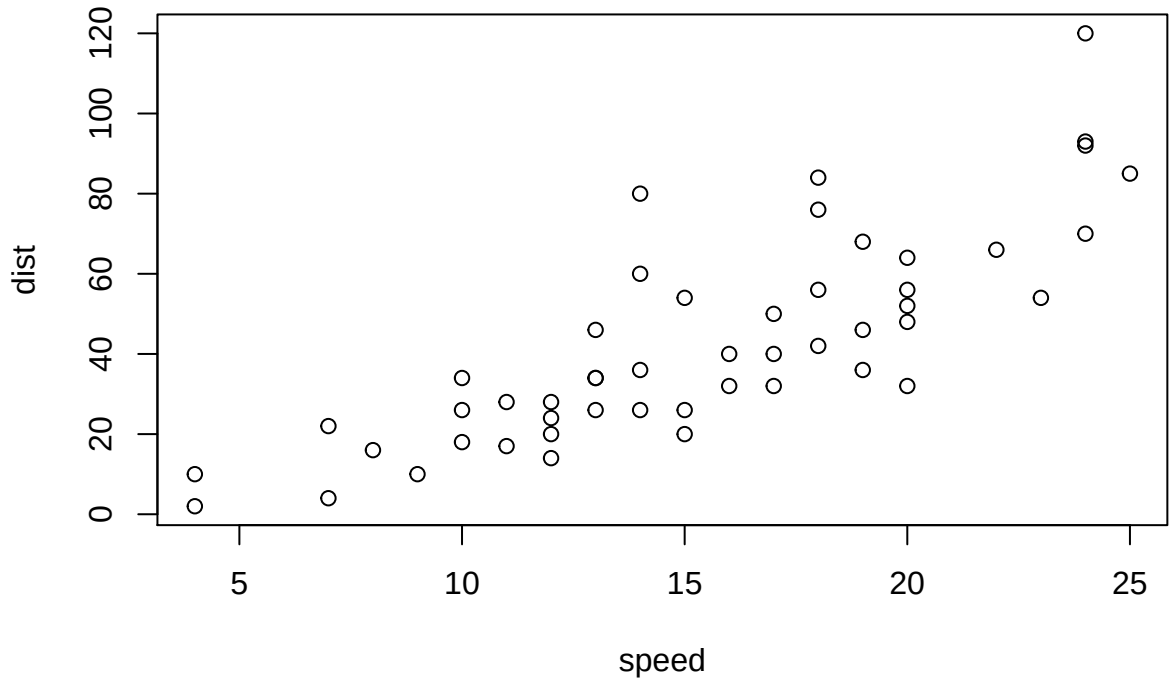
```
## Time difference of 5.425458e-07 days
```

15. Base Graphics

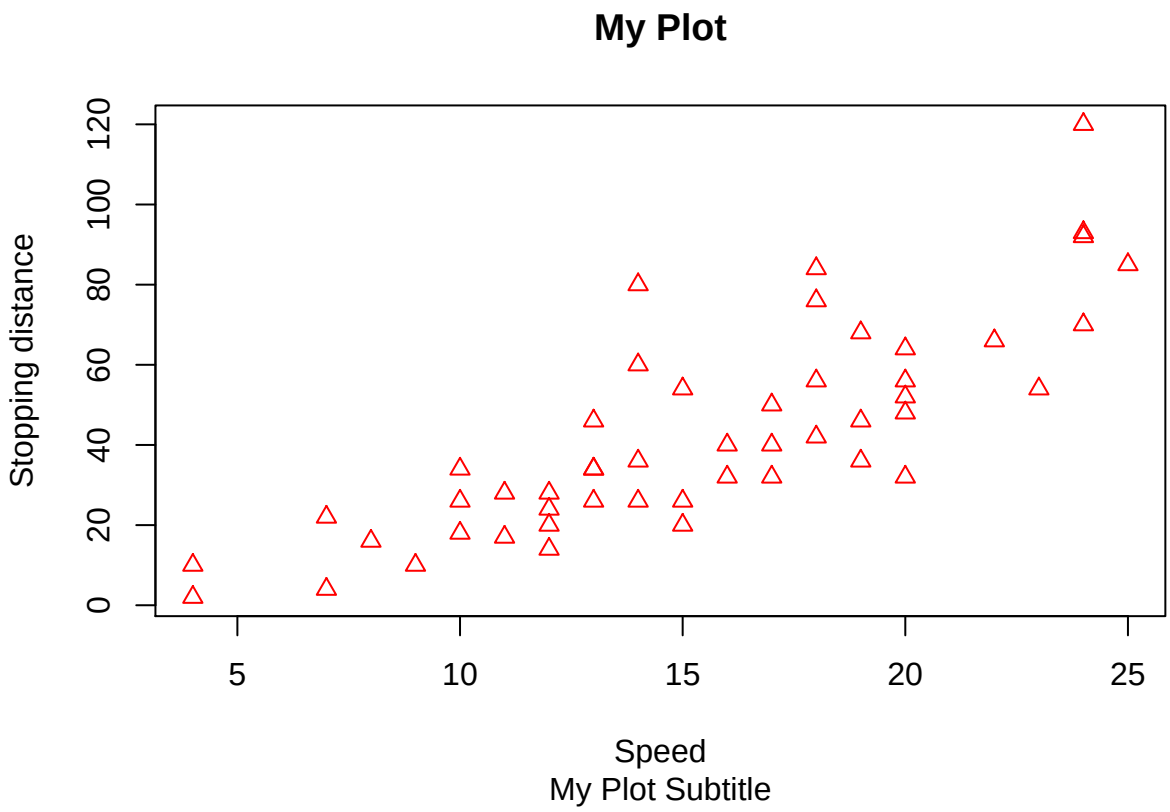
```
data(cars)
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

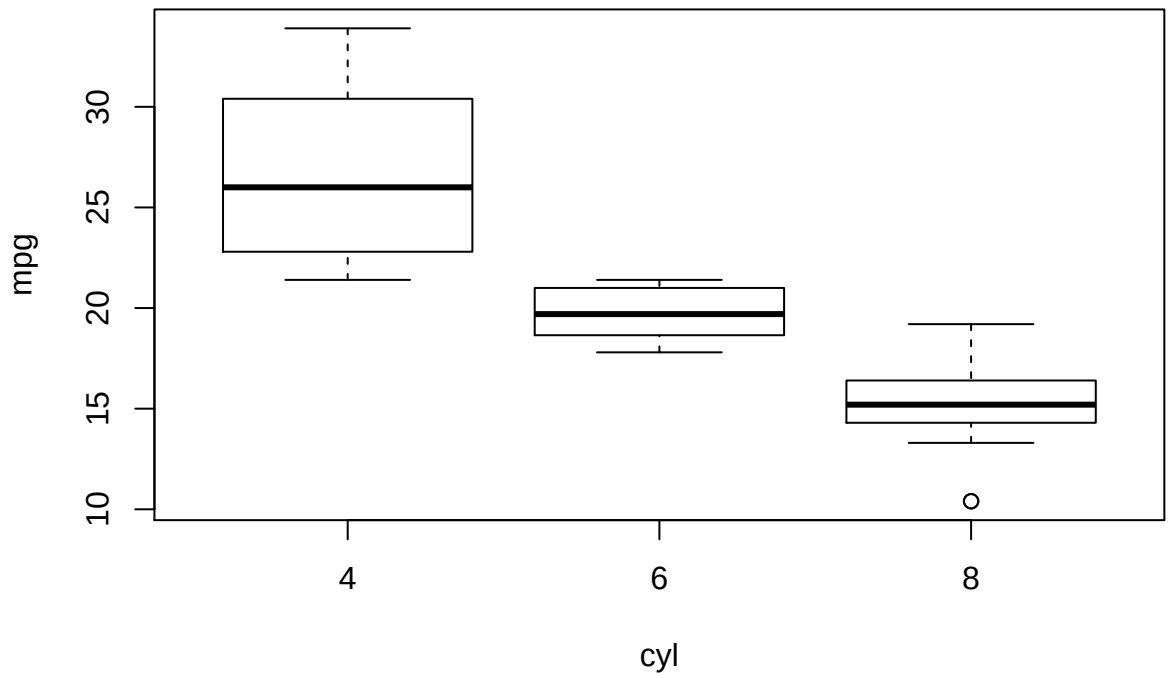
```
plot(cars)
```



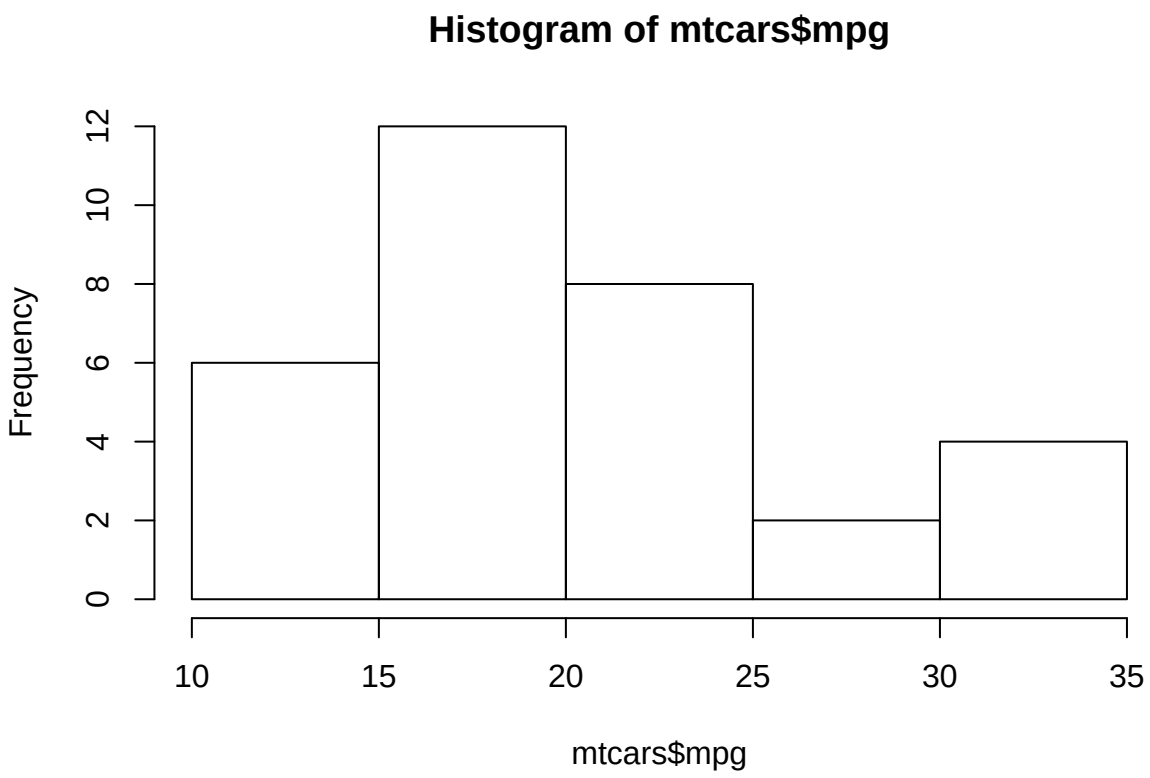

```
plot(x = cars$speed, y = cars$dist,xlab= "Speed" , ylab = "Stopping distance", main = "My Plot", sub =
```




```r
data(mtcars)
boxplot(formula = mpg ~ cyl, data = mtcars)
```




```
hist(mtcars$mpg)
```


Histogram of mtcars$mpg
Frequency
0
2
4
6
8
10
12
10
15
20
25
30
35
mtcars$mpg