R-Introduction using RStudio

S. Ravichandran, Ph.D

Jan 21, 2020; updated 01/2022

History of R

- R is a language
- No compilation
- Dynamic programming language

RStudio basics

- RStudio is a RGUI
- R is the main program

Helpful to start the R session with the following clean-up command. Warning the following command will remove everything in the working environment

```
rm(list=ls())
```

Let us treat R as a simple calculator

```
2 * 3  #[1] 6
```

```
## [1] 6
```

Note if R doesnt recognize a code, it displays a + sign and waits for you

```
# Uncomment the following line and run using the green arrow to the right
# 2 *
```

```
# uncomment the following line and run it
 # 10 % 5
```

Objects in R

Note that in R everything is treated as an Object.

Let us start with some basic commands.

```
# non-standard dont use the assignment operator <-
a = 1

a <- 1
```

Note that now object a contains number 1. Also note that RStudio environment pane now displays the variable

Let us now talk about what are acceptable and not acceptable variable names.

```
var1 <- 2
var1 <- 100 #overwrite
```

```
ls() # to see what variables you have so far
```

```
## [1] "a"    "var1"
```

```
list.files()  # to see the list of files
```

```
##  [1] "BasicCOLAB_Tutorial.ipynb"           "Bioinformatics.md"
##  [3] "ComputerLab-PevsnerBook"             "data"
##  [5] "Img"                                 "LICENSE"
##  [7] "Linux.ipynb"                         "mtcars.csv"
##  [9] "NCBI_EDIRECT.ipynb"                  "R_tutorial_Bioconductor_Colab.ipynb"
## [11] "R-Intro.html"                        "R-Intro.pdf"
## [13] "R-Intro.Rmd"                         "README.md"
```

```
currdir <- getwd() # to see where you are
setwd(currdir) # not going anywhere
getwd()   # check
```

```
## [1] "/Users/sakaravi/Documents/Github/bioinformatics"
```

Basic operations in R

Let us do some data analysis with a die?

```
die <- 1:6  # 6 sided die

# let us do some math with die
# notice the vector math

die + 2
```

```
## [1] 3 4 5 6 7 8
```

```
die/2
```

```
## [1] 0.5 1.0 1.5 2.0 2.5 3.0
```

```
die * die
```

```
## [1]  1  4  9 16 25 36
```

```r
# recycling

die + 1:2
```

```
## [1] 2 4 4 6 6 8
```

```r
# function
sum(die)
```

```
## [1] 21
```

```r
# Help

# ?sum  # quick way to explore the function arguments

mean(die)
```

```
## [1] 3.5
```

Constants in R

```r
# stored constants
pi
```

```
## [1] 3.141593
```

```r
# do sume calculations with the constants
# let us round it

round(pi, 2)
```

```
## [1] 3.14
```

```r
# let us find the arguments of funtion round
args(round)
```

```
## function (x, digits = 0)
## NULL
```

```r
args(plot)
```

```
## function (x, y, ...)
## NULL
```

Drawing samples is an important activity in Statistics

```
args(sample)
```

```
## function (x, size, replace = FALSE, prob = NULL)
## NULL
```

```
# sampling function in R

# let us sample a die by rolling 3 times

# uncomment (remove the #) and run the following line
# sample(die, size = 7)

# What was the problem?

# Read the help page or args function on sample
# to figure out

#  ?sample
```

```
p <- c(rep(0.01,5), 0.95)
p
```

```
## [1] 0.01 0.01 0.01 0.01 0.01 0.95
```

```
sample(x = die, size = 10, replace = TRUE, prob = p)
```

```
##  [1] 6 6 6 6 6 6 6 6 6 6
```

```
# no argument name
sample(die, 10, replace = TRUE)
```

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

```
sample(die, 10, TRUE) #  define them explicitely
```

```
##  [1] 1 3 5 1 3 3 5 1 2 2
```

```
sample(x = die, size = 6, replace = TRUE)
```

```
## [1] 3 1 4 4 1 4
```

```
# multiple rolls
# First roll
die
```

```
## [1] 1 2 3 4 5 6
```

```
sample(x = die, size = 2, replace = TRUE) # def prob
```

```
## [1] 2 3
```

```
# one more roll
sample(x = die, size = 2, replace = TRUE) # def prob
```

```
## [1] 2 3
```

Random numbers

Let us see how to generate random numbers.

```
unif_rand <- runif(10); unif_rand
```

```
##  [1] 0.177229857 0.302308446 0.179603071 0.053692748 0.601490391 0.773414086
##  [7] 0.727386957 0.136901115 0.182475724 0.005943073
```

```
args(round)
```

```
## function (x, digits = 0)
## NULL
```

```
#### Note the digits = 0; it is set to zero and optional
round(unif_rand, digits = 2)
```

```
##  [1] 0.18 0.30 0.18 0.05 0.60 0.77 0.73 0.14 0.18 0.01
```

Functions (like subroutines in Fortran)

Function constructor (like C++) has three parts

- name
- body (of code) () arguments

The function structure looks like the following: function() { ### }

```
dist <- function(a, b) {
  distsq <- sum( (b - a)^2 )
  sqrt(distsq)
}

a <- c(1,2)
b <- c(3,3)

dist(a,b)
```

```
## [1] 2.236068
```

We can use RStudio to turn the following line into a function and call it dist2 We can create is just using a few click in RStudio.

Let us do the following steps: 1) Select 253-254 lines by highlighting 2) After selecting the lines, do the following, Code –> "Extract Function" and give it a name, my_dist 3) Execute the code 4) test it with a <- c(1,1,2) b <- c(2,2,3) my_dist(a,b)

```
    distsq <- sum( (b - a)^2 )
    sqrt(distsq)
```

```
## [1] 2.236068
```

R-packages

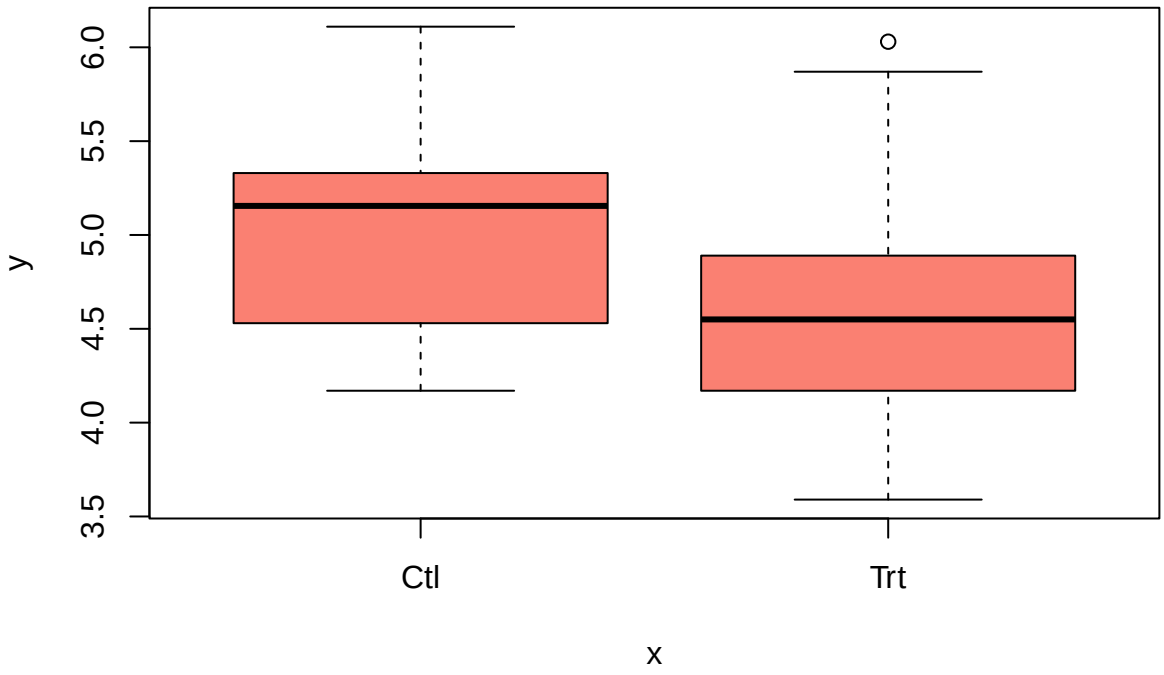
```
#install.packages("randomForest")
#library(randomForest)
# update.packages(c("randomForest","tidyverse"))
# update R after update.packages
```

Let us look at a linear regression example

```
## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## Page 9: Plant Weight Data.
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2, 10, 20, labels = c("Ctl","Trt"))
weight <- c(ctl, trt)
lm.D9 <- lm(weight ~ group)
lm.D90 <- lm(weight ~ group - 1) # omitting intercept
lm.D9
```

```
##
## Call:
## lm(formula = weight ~ group)
##
## Coefficients:
## (Intercept)     groupTrt
##       5.032       -0.371
```

```
plot(group,weight, col = "salmon")
```



Export an object

Save an object to a file

```
mtcars
```

```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4          21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag      21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive     21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant            18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D          24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280           19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE         16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC        15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4  8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial  14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## 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
```

```
getwd()
```

```
## [1] "/Users/sakaravi/Documents/Github/bioinformatics"
```

```
write.csv(mtcars, file = "mtcars.csv")
```

Debugging basics

Debugging example. Let us define a function called mysum.

```
mysum <- function(a,b) {
  sum( (bb - a)^2 )
}

dist <- function(a, b) {
  distsq <- mysum(a,b)
  sqrt(distsq)
}

# use of RStudio bebug or traceback function
# dist(a,b)
```

use of browser function in debug use the control buttons to step through the function

```
dist2 <- function(a, b) {
  browser()
  distsq <- mysum(a,b)
  sqrt(distsq)
}

## Uncomment and run the following two commands to understand
## how to debug codes

# dist2(a,b)
#
# debug(dist2(a,b))
```

Data types

- six basic types of atomic vectors:
- doubles, integers, characters, logicals, complex, and raw
- last two are not very important (see one example below) and we will not discuss further

```
complex(1,2)
```

```
## [1] 2+0i
```

```
raw(length = 2)
```

```
## [1] 00 00
```

Let us explore the data types with examples

```
ia <- 1L
ia1 <- 1
class(ia)
```

```
## [1] "integer"
```

```
typeof(ia) # internal type or storage mode of any object
```

```
## [1] "integer"
```

```
class(ia1)
```

```
## [1] "numeric"
```

```
typeof(ia1)
```

```
## [1] "double"
```

```
ra <- runif(10)
typeof(ra)
```

```
## [1] "double"
```

```
class(ra)
```

```
## [1] "numeric"
```

```
ca <- c("Tom", "cat")
typeof(ca)
```

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

```
class(ca)
```

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

What is an attribute?

```r
x <- cbind(a = 1:3, pi = pi) # simple matrix with dimnames
x
```

```
##      a       pi
## [1,] 1 3.141593
## [2,] 2 3.141593
## [3,] 3 3.141593
```

```r
attributes(x)
```

```
## $dim
## [1] 3 2
##
## $dimnames
## $dimnames[[1]]
## NULL
##
## $dimnames[[2]]
## [1] "a"  "pi"
```

More on standard R functions

```r
matrix(1:6, 3, 2)
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```r
args(matrix)
```

```
## function (data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
## NULL
```

```r
# Density function
# density function at 5 for a unit normal distribution
dnorm(1,0,5)
```

```
## [1] 0.07820854
```

```r
dnorm(2, mean = 2, sd = 3)
```

```
## [1] 0.1329808
```

```r
dnorm(0, mean = 0, sd = 1)
```

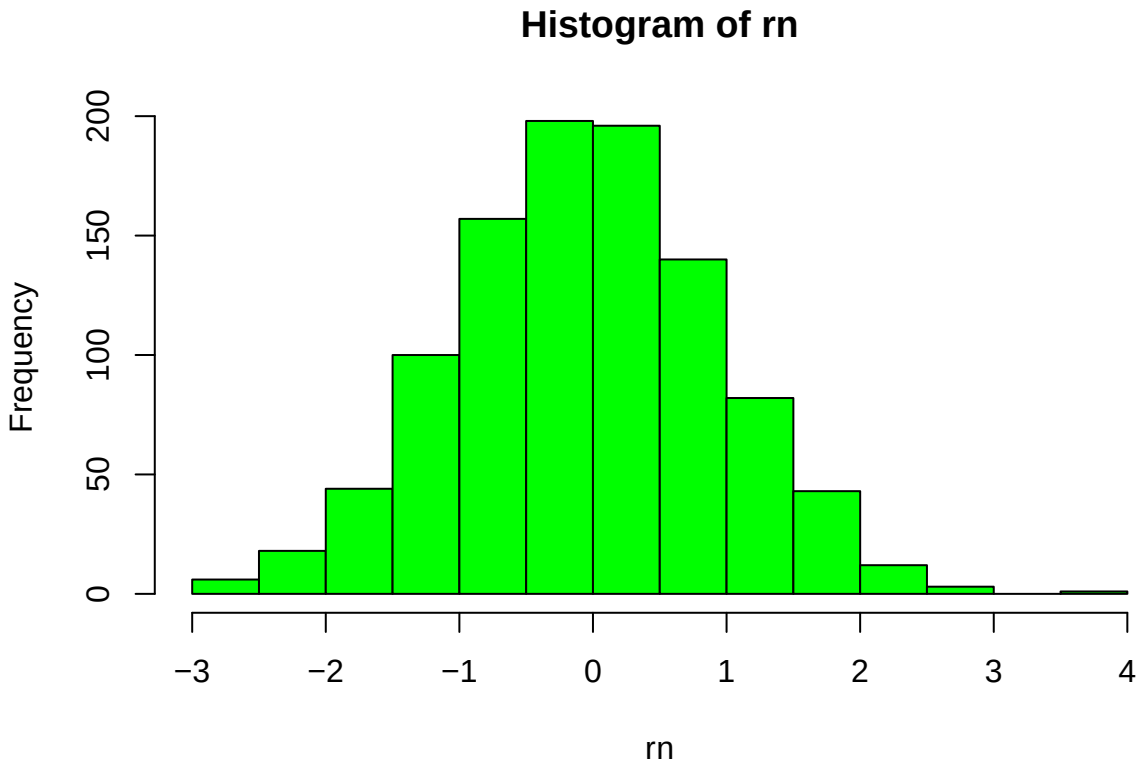
```
## [1] 0.3989423
```

```
pnorm(0, mean = 0, sd = 1)
```

```
## [1] 0.5
```

```
# Data from UsingR package
# to download do the following
# install.packages("UsingR")
# data(father.son,package="UsingR")

father.son <- read.csv("https://raw.githubusercontent.com/ravichas/bioinformatics/main/data/father.son_
rn <- rnorm(1000,0,1)
hist( rn, col = "green" )
```




```
x <- father.son$fheight
mean(x)
```

```
## [1] 67.6871
```

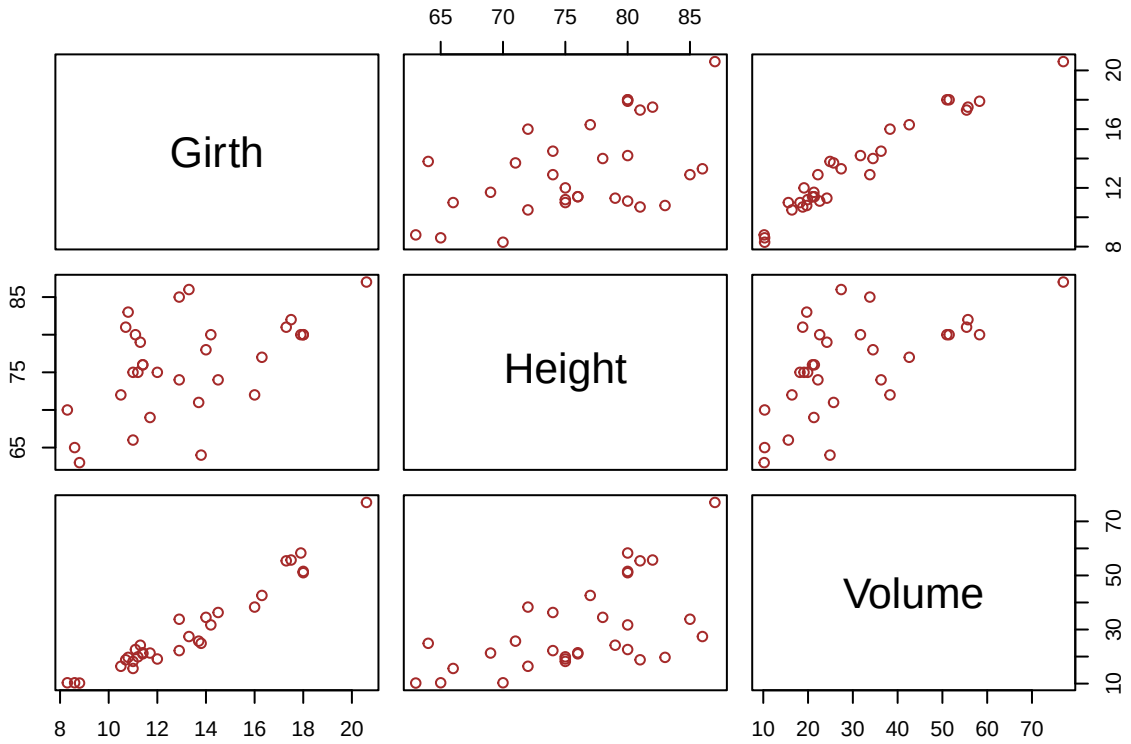
```
sd(x)
```

```
## [1] 2.744868
```

```
 hist(x,xlab="Height (in inches)",main="Adult men heights", col = "pink")
  abline(v = 75, col = "blue",lwd = 2)
```


```
# if trees object is a matrix then then
# figure from pairs can be imagined to a graphical view of a
# matrix

#take advantage of system functions
pairs(trees, col = "brown")
```




```
# let us roll a die and sum up the numbers
# first time
results <- sample(x = 1:6, 3)
results
```

```
## [1] 5 2 3
```

```
sum(results)
```

```
## [1] 10
```

```
# second time
results <- sample(x = 1:6, 3)
sum(results)
```

```
## [1] 6
```

```
# We do not want to repeat the same commands all the time
# let us put them in a function

roll <- function() {
  die <- 1:6
  outcome <- sample(x = die, size = 2, replace = TRUE )
#  print(outcome)
  sum(outcome) # returns when you run the function
}

roll()
```

```
## [1] 10
```

```
roll()
```

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

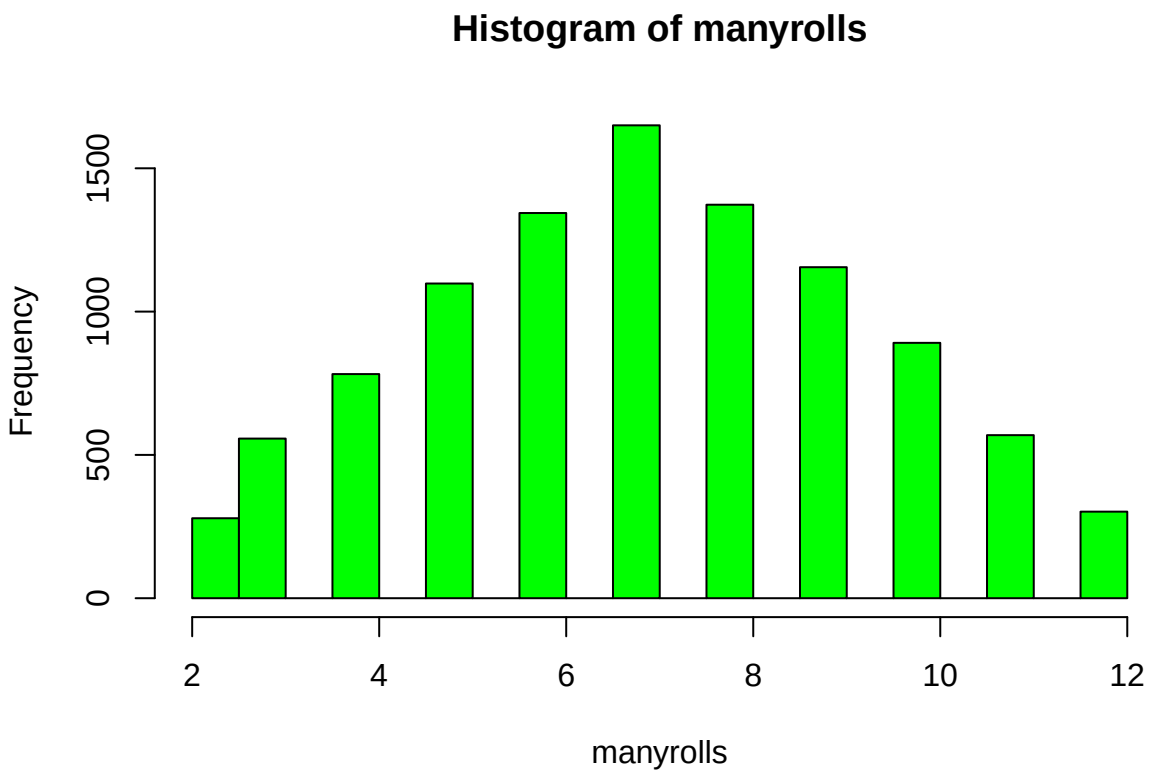
```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1
```

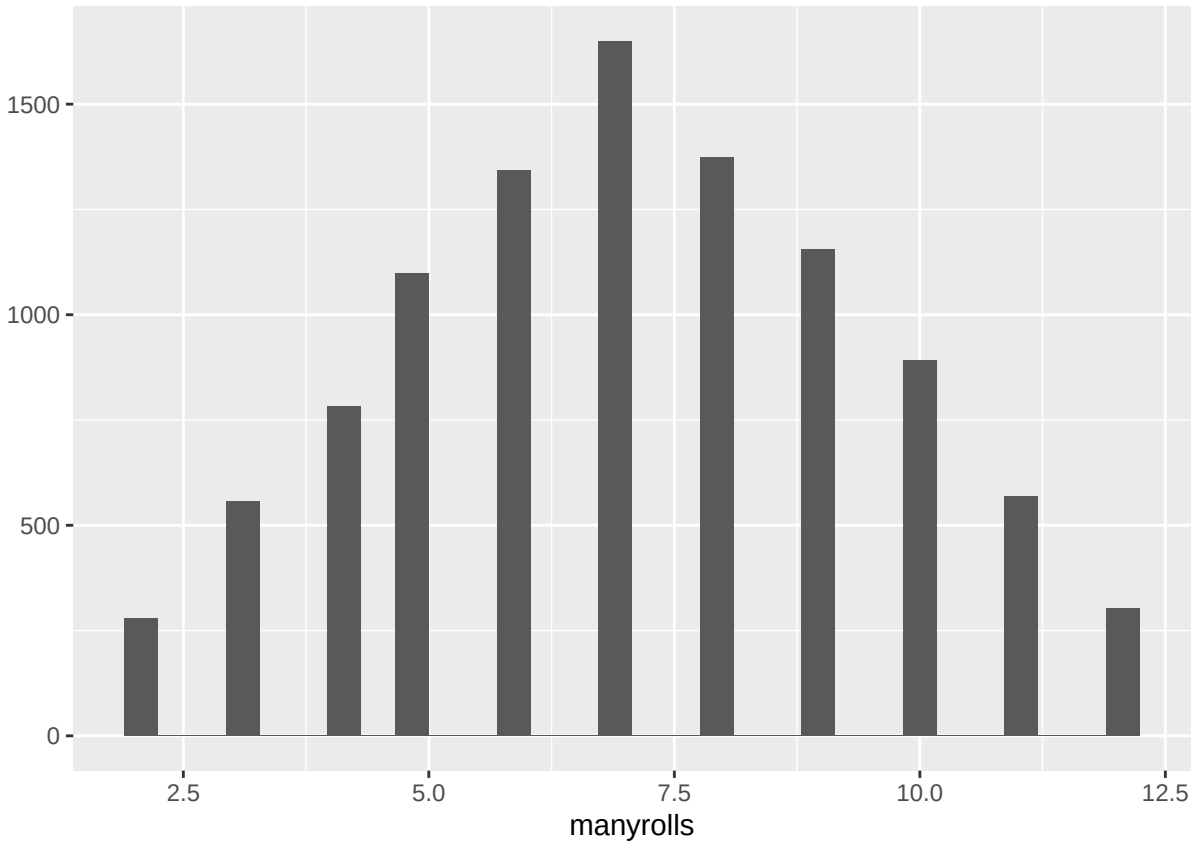
```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
manyrolls <- replicate(10000, roll())
hist(manyrolls, col = "green")
```




```
qplot(manyrolls, geom = "histogram")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
roll <- function() {
   die <- 1:6
   dice <- sample(die, size = 2, replace = TRUE)
   sum(dice)
}

roll1 <- function( s = 2) {
   die <= 1:6
   dice <- sample(x = die, size = s, replace = TRUE) # def prob
  sum(dice)
}
```

How to let RStudio create

```
# Turn the following lines into a function in RStudio
# call this a function, myroll. Here are the steps:
#  1) select lines, 450-451 and uncomment them using code --> uncomment Lines
#  2) use ctrl-Alt-X to create a function (works for windows)
#  3) give a function name, myroll

# after step 3, you should see something like the following
#

 # myroll <- function(die, s) {
 #   dice <- sample(x = die, size = s, replace = TRUE) # def prob
```

```
 #   sum(dice)
 # }
```

```
# dice <- sample(x = die, size = s, replace = TRUE)
# sum(dice)

die <- 1:6
s <- 3
# myroll(die, s )

random <- sample(1:52, size = 52)
random
```

```
##  [1] 29 34 33 51 10 43 45 38 40  5  7 19  8 22 50 11 47 23 25 28  3 14  4 48 27
## [26] 35 24  9 49 41 18 16 12 26 36 21 20 39 30 52 31 32 42  2 13 17 44  6 46 37
## [51] 15  1
```

```
get_symbols <- function() {
  wheel <- c("DD", "7", "BBB", "BB", "B", "C", "0")
  sample(wheel, size = 3, replace = TRUE,
         prob = c(0.03, 0.03, 0.06, 0.1, 0.25, 0.01, 0.52))
}
get_symbols()
```

```
## [1] "0" "0" "0"
```