R Basics for Beginners

Sha Cao (adapted from online resources)

09-10 June 2021

Introduction

Get started with R

- To get started, download and install R (https://cran.r-project.org/). You should also download and install RStudio (https://www.rstudio.com/).
- RStudio bundles code editor, console, command history, debugging, documentation and visualization in a single install.
- Within RStudio, you can check the current version of R by typing the command `version` .
- Across the R ecosystem, software is delivered as packages.
- RStudio comes with the `base` package and more. Additional packages can be installed. You can also list currently installed packages. Here are some examples:

```
##print
print("Hello, world!")

# Install package data.table
install.packages("data.table")

# List all installed packages with details
installed.packages()

# List all installed packages
library()

# Get help on base package
library(help = "base")

# Update or remove package
update.packages("data.table")
remove.packages("data.table")

# Use a package
library("data.table")

# Find the version
packageVersion("data.table")
```

Within RStudio, these shortcuts are useful:

- Tab: Autocomplete a command on the console.
- Up/Down Arrow: Navigate the command history to reuse commands.

Resources

Here are some ways to get help:

```
# Two ways of getting help on function str
?str
help(str)

# Get help on only the arguments or examples of function dim
args(dim)
example(dim)

# Search all help pages for a given phrase
help.search("linear regression")
```

Here are some simple function calls for you to try:

```
ls()                          # list all R objects in current environment
dir()                         # display content of current directory
getwd()                       # display path of current working directory
setwd("example1/data")        # change the working directory with relative path
source("main.R")              # execute the named R script
```

Script files

- While you can enter commands directly into the R console, it is **highly** recommended that you use a script file for your R code. As noted above, these script files have a “.R” extension.
- They are essentially plain text files that R interprets and runs. The code in the script file is passed to the R console. Script files allow you to save and annotate your code for future use.

*In R, comments are indicated with `#`. Anything after `#` is commented out and not run.

Setting your working directory

Setting your working directory makes saving and uploading script files, data sets, and figures easier. To set your working directory to the desktop on a Mac, type

```
setwd("~/Desktop")
```

On a PC, it would be something like this:

```
setwd("C:\Users\yourname\Desktop")
```

You can get your current working directory with

```
setwd("/Users/shacao 1/errands/basicR")
getwd()
```

```
## [1] "/Users/shacao 1/errands/basicR"
```

Data Types

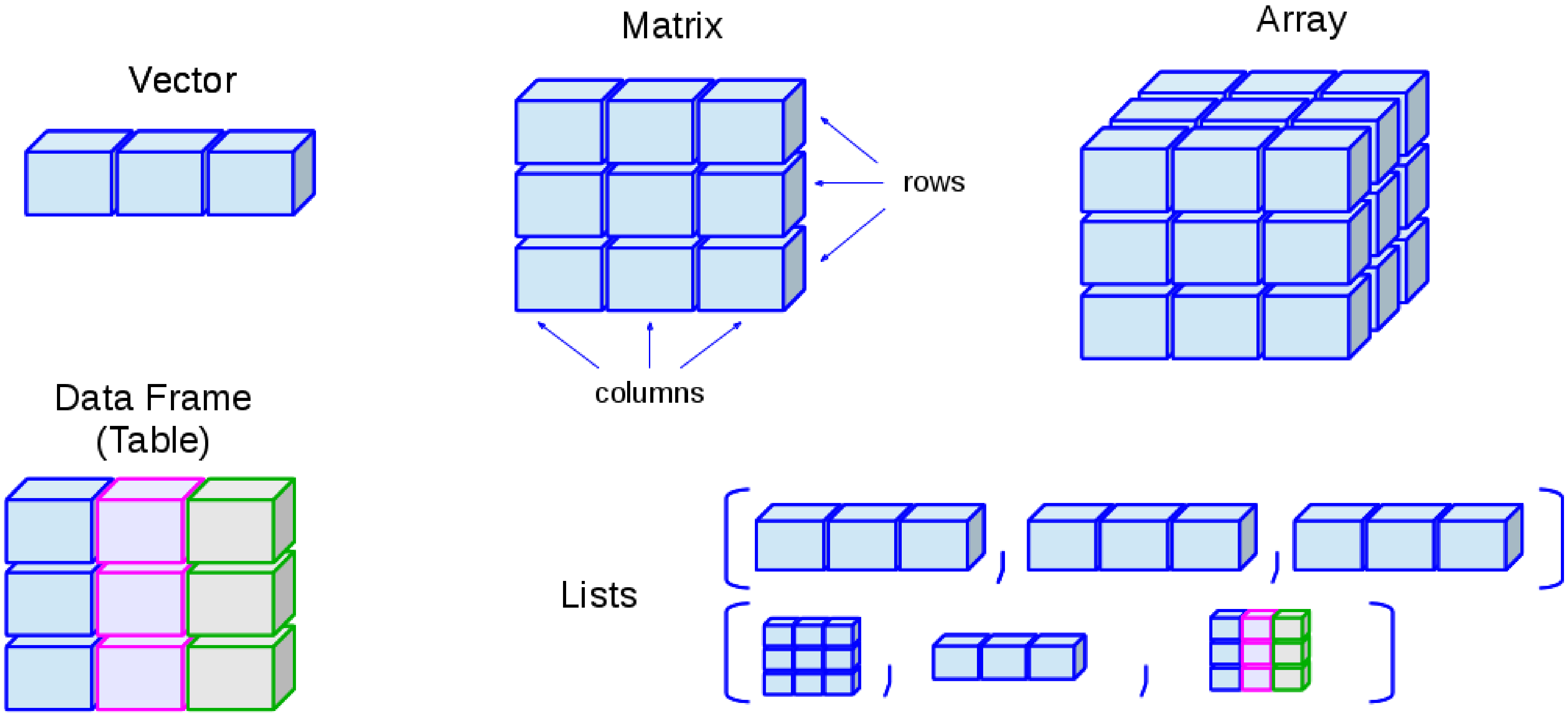
You can store anything you do in R as an object with a name, then look at that object, run another command on it later, overwrite it, or delete it as you see fit.

R has the following basic types that are called atomic types:

- `logical` : Can take value TRUE or FALSE.
- `integer` : Specified with suffix ‘L’. Eg. 23L, -4L
- `numeric` : Specified as a number without suffix. Eg. 23, -4, 2.12
- `complex` : Complex numbers. Eg. 2+3i, -3-5i
- `character` : Strings are specified as a sequence of characters.
- `raw` : Raw bytes.

Among the data types are the following:

- `vector` : Contains a sequence of items of the same type. Type is also called a mode in the context of vectors. This is most basic type. Items of a vector can be accessed using [].
- `scalar` : A special form or `vector` which is of length 1.
- `list` : Represented as a vector but can contain items of different types. Different columns can contain different lengths. Items of a list can be accessed using [[]]. This is a recursive data type: lists can contain other lists.
- `array` : Vectors with attributes dim and dimnames.
- `matrix` : A two-dimensional array.
- `data.frame` : While all columns of a matrix have same mode, with data frames different columns can have different modes. This can be considered a type of list where all columns have same length.
- `factor` : Dactor represents a finite set of values. We may also call factors as categories or enumerated types. It’s also possible to specify an order for factors.



Source: Instituto de Física de Cantabria. http://venus.ifca.unican.es/Rintro/_images/dataStructuresNew.png (http://venus.ifca.unican.es/Rintro/_images/dataStructuresNew.png)

Objects in R

Scalars

Here is a basic example. I want to create an object called `x` that has the value 2. I use the assignment operator, `<-` , to do this:

```
x <- 2
x
```

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

```
# Scalar variables can be differentiated from vectors using str but not class
a <- 3
str(a)
```

```
##  num 3
```

```
class(a)
```

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

```
a <- c(3,4)
str(a)
```

```
##  num [1:2] 3 4
```

```
class(a)                         # same output as scalar
```

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

You can use R as a calculator. Storing various objects and calculations as you go for future use.

```
y <- log(10)
x + y
```

```
## [1] 4.302585
```

```
a <- x + y^2
a
```

```
## [1] 7.301898
```

The objects `x` , `y`, , and `a` are simple scalars. They only have one value. Next, we’ll explore some of the most common types of R objects: vectors, matrices, lists, and data frames.

Vectors

To create a vector of numbers, use the `c` function.

```
my_vector <- c(1, 2, 3, 4, 5)
my_vector
```

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

To create a sequence of numbers, use the colon `:` or the `seq` function. To repeat a pattern, use `rep` . This makes a sequence from 0 to 10, increasing by 1 each time.

```
my_seq <- 0:10
my_seq
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10
```

This makes a sequence from 0 to 10, increasing by 2 each time.

```
my_seq2 <- seq(0, 10, by = 2)
my_seq2
```

```
## [1]  0  2  4  6  8 10
```

This uses `c()` to repeat the pattern “1, 2, 3” 8 times.

```
my_rep <- rep(c(1, 2, 3), times = 8)
my_rep
```

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

To reference specific elements from a vector, use the square brackets `[]`. This displays the 4th element in `my_seq`

```
el_4 <- my_seq[4]
el_4
```

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

This uses `c()` to reference multiple elements.

```
el_4_and_6 <- my_seq[c(4, 6)]
el_4_and_6
```

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

This makes a sequence with all of the elements of `my_seq` except the 3rd element.

```
no_3rd_el <- my_seq[-3]
no_3rd_el
```

```
##  [1]  0  1  3  4  5  6  7  8  9 10
```

This replaces an element in your vector.

```
my_seq[c(1, 3)]<-c(22, 60)
my_seq
```

```
##  [1] 22  1 60  3  4  5  6  7  8  9 10
```

Matrices

To create a matrix, use the `matrix` function. There are several arguments available for this function. To learn about them, type `?matrix`.

This makes a 3 x 3 matrix with the numbers 1 through 9, entering by columns. Notice the use of `:` .

```
my_mat <- matrix(1:9, nrow = 3, ncol = 3)
my_mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

We still use square brackets to reference elements of a matrix, but this time in 2 dimensions. This references the element in row 2, column 2.

```
row2_col2 <- my_mat[2, 2]
row2_col2
```

```
## [1] 5
```

This references the entire third row.

```
row_3 <- my_mat[3, ]
row_3
```

```
## [1] 3 6 9
```

To make a matrix from previously defined vectors use `rbind` or `cbind` .

```
v1 <- c(1, 2, 3, 4, 5)
v2 <- c(1, 2, 1, 2, 1)
v3 <- c(.1, .2, .7, .4, .1)
data_mat <- cbind(v1, v2, v3)
data_mat
```

```
##      v1 v2  v3
## [1,]  1  1 0.1
## [2,]  2  2 0.2
## [3,]  3  1 0.7
## [4,]  4  2 0.4
## [5,]  5  1 0.1
```

Lists

A list is basically a vector that may contain multiple types of data (e.g. strings, numbers, or even matrices). You can probably guess the function to create a list.

```
my_list <- list(7, "Hello World", data_mat, a)
my_list
```

```
## [[1]]
## [1] 7
##
## [[2]]
## [1] "Hello World"
##
## [[3]]
##      v1 v2  v3
## [1,]  1  1 0.1
## [2,]  2  2 0.2
## [3,]  3  1 0.7
## [4,]  4  2 0.4
## [5,]  5  1 0.1
##
## [[4]]
## [1] 7.301898
```

You can reference an element of a list the same way you would a vector.

```
my_list[2]
```

```
## [[1]]
## [1] "Hello World"
```

Lists are generally not as widely used as other objects, but as data sets and analysis become more complex, their ability to hold multiple types of data becomes very powerful.

Data frames

Data frames are arguably the most important type of object in R. They are made up of other objects, like vectors and lists. Data frames are similar to data sets that are used in Stata. They have rows and columns and can be visualized like a spreadsheet.

Let’s create a simple data frame.

```
my_data_frame <- as.data.frame(data_mat)
my_data_frame
```

```
##   v1 v2  v3
## 1  1  1 0.1
## 2  2  2 0.2
## 3  3  1 0.7
## 4  4  2 0.4
## 5  5  1 0.1
```

Notice that the column names are preserved from the matrix we created earlier out of pre-defined vectors. We can change the column names to something more descriptive. This is a bit convoluted. Let’s change “v2” to “Sex”.

```
colnames(my_data_frame)[colnames(my_data_frame)=="v2"] <- "Sex"
colnames(my_data_frame)
```

```
## [1] "v1"  "Sex" "v3"
```

We can also change the contents of the data frame itself. Instead of numbers for sex, we can use words. This syntax for recoding variables isn’t the most intuitive or compact. Later on, we’ll cover a simpler way of recoding.

```
my_data_frame$Sex[my_data_frame$Sex==1] <- "Female"
my_data_frame$Sex[my_data_frame$Sex==2] <- "Male"
my_data_frame
```

```
##   v1    Sex  v3
## 1  1 Female 0.1
## 2  2   Male 0.2
## 3  3 Female 0.7
## 4  4   Male 0.4
## 5  5 Female 0.1
```

```
class(my_data_frame)
```

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

```
str(my_data_frame)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ v1 : num  1 2 3 4 5
##  $ Sex: chr  "Female" "Male" "Female" "Male" ...
##  $ v3 : num  0.1 0.2 0.7 0.4 0.1
```

In general, the `$` operator is how you refer to specific variables within a data frame.

We can use `table` to describe the breakdown of sex within our data frame.

```
table(my_data_frame$Sex)
```

```
## 
## Female   Male 
##      3      2
```

Operators, Control Structures and Functions

Here are some basic operations we can do on data:

```
v1 <- c(1:5)

# Some examples showing arithmetic operations
v1 + 2                   # value 2 is recycled for the entire vector
```

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

```
v1 + v1                  # element-wise addition
```

```
## [1]  2  4  6  8 10
```

```
v1 - 2                   # element-wise subraction
```

```
## [1] -1  0  1  2  3
```

```
v1 / 2                   # element-wise division
```

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

```
v1 %/% 2                 # integer division, retain only integer part
```

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

```
v1 / v1                  # element-wise division
```

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

```
v1 * 2                   # element-wise multiplication
```

```
## [1]  2  4  6  8 10
```

```
v1 ^ 2                   # exponential operator, can alternatively use **
```

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

```
v1 %% 2                  # modulo operator
```

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

```
m1 <- matrix(1:6, nrow=2, ncol=3, byrow=T)
m1 + m1                  # matrix element-wise addition
```

```
##      [,1] [,2] [,3]
## [1,]    2    4    6
## [2,]    8   10   12
```

```
m1 * m1                  # matrix element-wise multiplication
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    9
## [2,]   16   25   36
```

```
m1 %*% t(m1)             # matrix multiplication of m1 with its transpose
```

```
##      [,1] [,2]
## [1,]   14   32
## [2,]   32   77
```

```
m1 <- matrix(c(1:4, 11:15), nrow=3)
m1inv <- solve(m1)       # get inverse of matrix
m1 %*% m1inv             # result is an identity matrix
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```
# Some examples showing logical operations
v1 > 2
```

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

```
v1 <= 2
```

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

```
!(v1 <= 2)
```

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

```
v1 == 2
```

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

```
v1 != 2
```

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

```
v1 > 2 | v1 < 4
```

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

```
v1 > 2 & v1 < 4
```

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

For control structures, we have `if-else` , `for` loops, `while` loops, `repeat` loops. Also available are `ifelse` and `switch` . Here are some examples:

```
# Example of a for loop and if-else
for (val in 1:10) {
    if (val %% 2 == 0) {
        next
        print(paste(val, "is an even number.")) # this will not be printed
    }
    else if (val > 5) {
        print(paste(val, "is an odd number greater than 5."))
        break
    }
    else {
        print(paste(val, "is an odd number less than or equal to 5."))
    }
}
```

```
## [1] "1 is an odd number less than or equal to 5."
## [1] "3 is an odd number less than or equal to 5."
## [1] "5 is an odd number less than or equal to 5."
## [1] "7 is an odd number greater than 5."
```

```
# Example of a while loop
names <- c("Manoj", "Anjali", "Poonam", "Kumar", "Gautam")
needle <- "Poonam"
i = 1
foundAt = 0
while (foundAt == 0) {
    if (names[i]==needle) {
        foundAt = i
    }
    i = i + 1
}
print(paste("Found at index", foundAt))
```

```
## [1] "Found at index 3"
```

```
# Repeat is an infinite loop used with a break
i = 1
repeat {
    if (names[i]==needle) {
        foundAt = i
        break
    }
    i = i + 1
}
print(paste("Found at index", foundAt))
```

```
## [1] "Found at index 3"
```

```
# Example showing the use of ifelse
grades <- c(55, 65, 43, 67, 22, 83)
ifelse(grades >= 50, "pass", "fail")
```

```
## [1] "pass" "pass" "fail" "pass" "fail" "pass"
```

```r
# Without ifelse, this would be the long way
results <- grades >= 50                        # get a logical vector
results[results] <- "pass"                     # also coerces to character vector
results[results=="FALSE"] <- "fail"

# Example showing the use of switch
# C: 0-24, B: 25-49, A: 50:74, A+: 75-100
for (g in ceiling(grades/25)) {
    print(switch(g, "C", "B", "A", "A+"))
}
```

```
## [1] "A"
## [1] "A"
## [1] "B"
## [1] "A"
## [1] "C"
## [1] "A+"
```

Working with Data

R offers many ways to read data from files in many formats. It’s common to read data into `data.frame` type. Reading data into `data.table` type is more suitable for large datasets. Here are a couple of examples:

```
# Download file from URL if not already downloaded
if (!file.exists("atheletes.csv"))
    download.file("https://raw.githubusercontent.com/flother/rio2016/master/athletes.csv",
                  "atheletes.csv", method="curl", quiet=T)

# Read from .csv file into a data.frame
dat <- read.csv("atheletes.csv",header=TRUE, sep=",", skip=0L)

# Read from .csv file into a data.table (assuming that data.table package is installed)
library("data.table")
dat <- fread("atheletes.csv",sep = ",", header = TRUE, stringsAsFactors = TRUE, )
dat <- fread("atheletes.csv",sep = ",", header = TRUE, stringsAsFactors = TRUE, select=c("name","nationality"))

# Read from .xlsx file into a data.table (assuming that readxl package is installed)
library("readxl")
dat_expr_1 <- read_excel("test-expr.xlsx",sheet=1,col_names = TRUE,  na = "", skip = 0)
dat_expr_2 <- read_excel("test-expr.xlsx",sheet=2,col_names = TRUE,  na = "", skip = 0)
dat_expr_11 <- data.frame(dat_expr_1)

# Read from .txt file into a data.table
dat <- read.table("test.txt",sep=",",header=T,skip=0)

# Save R objects into file and restore them later
x <- 1:10
y <- LETTERS[1:10]
save(x, y, file = "backup.RData")
rm(x)
rm(y)
load("backup.RData")
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
y
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
```

Basic Data Analysis

R comes with pre-loaded datasets that can be handy for beginners learning the language. In the following examples, some operations may not make sense from the point of analysis but they are shown only to illustrate the features of R. Here are a few things to try:

```
data()                                            # display a list of pre-loaded datasets

# Basic information about the data
help(mtcars)                                      # display help on pre-loaded dataset mtcars
```

```
str(mtcars)                                       # mtcars is of type data.frame
```

```
## '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 ...
```

```
dim(mtcars)                                       # display the dimensions
```

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

```
colnames(mtcars)                                  # display the column names
```

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

```
mtcars$mpg                                        # display column mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

```
object.size(mtcars)                               # size in bytes
```

```
## 7208 bytes
```

```
format(object.size(mtcars), units="KB")           # size in kilobytes
```

```
## [1] "7 Kb"
```

```
# Peeking into the data
head(mtcars)                                      # display only the first 6 rows
```

```
##                    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
```

```
head(mtcars, 10)                                  # display only the first 10 rows
```

```
##                    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
```

```
tail(mtcars)                                      # display only the last 6 rows
```

```
##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

```
tail(mtcars[c("mpg","hp")], 10)                   # display last 10 rows of only two columns
```

```
##                   mpg  hp
## AMC Javelin      15.2 150
## Camaro Z28       13.3 245
## Pontiac Firebird 19.2 175
## Fiat X1-9        27.3  66
## Porsche 914-2    26.0  91
## Lotus Europa     30.4 113
## Ford Pantera L   15.8 264
## Ferrari Dino     19.7 175
## Maserati Bora    15.0 335
## Volvo 142E       21.4 109
```

```
# Getting a subset of data
subset(mtcars, mpg>20, c("mpg", "hp"))
```

```
##                 mpg  hp
## Mazda RX4      21.0 110
## Mazda RX4 Wag  21.0 110
## Datsun 710     22.8  93
## Hornet 4 Drive 21.4 110
## Merc 240D      24.4  62
## Merc 230       22.8  95
## Fiat 128       32.4  66
## Honda Civic    30.4  52
## Toyota Corolla 33.9  65
## Toyota Corona  21.5  97
## Fiat X1-9      27.3  66
## Porsche 914-2  26.0  91
## Lotus Europa   30.4 113
## Volvo 142E     21.4 109
```

```
subset(mtcars, mpg==max(mpg))
```

```
##                 mpg cyl disp hp drat    wt qsec vs am gear carb
## Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.9  1  1    4    1
```

```
subset(mtcars, mpg==max(mpg), mpg)
```

```
##                 mpg
## Toyota Corolla 33.9
```

```
# Basic analysis
summary(mtcars)                                   # calculate min, max, mean, etc. by columns
```

```
##       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  
```

```
colMeans(mtcars)                                  # calculate mean by columns
```

```
##        mpg        cyl       disp         hp       drat         wt       qsec 
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250  17.848750 
##         vs         am       gear       carb 
##   0.437500   0.406250   3.687500   2.812500 
```

```
colSums(mtcars)                                   # calculate sum by columns
```

```
##      mpg      cyl     disp       hp     drat       wt     qsec       vs 
##  642.900  198.000 7383.100 4694.000  115.090  102.952  571.160   14.000 
##       am     gear     carb 
##   13.000  118.000   90.000 
```

```
rowSums(mtcars)                                   # calculate sum by rows
```

```
##           Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive 
##             328.980             329.795             259.580             426.135 
##   Hornet Sportabout             Valiant          Duster 360           Merc 240D 
##             590.310             385.540             656.920             270.980 
##            Merc 230            Merc 280           Merc 280C          Merc 450SE 
##             299.570             350.460             349.660             510.740 
##          Merc 450SL         Merc 450SLC  Cadillac Fleetwood Lincoln Continental 
##             511.500             509.850             728.560             726.644 
##   Chrysler Imperial            Fiat 128         Honda Civic      Toyota Corolla 
##             725.695             213.850             195.165             206.955 
##       Toyota Corona    Dodge Challenger         AMC Javelin          Camaro Z28 
##             273.775             519.650             506.085             646.280 
##    Pontiac Firebird           Fiat X1-9       Porsche 914-2        Lotus Europa 
##             631.175             208.215             272.570             273.683 
##      Ford Pantera L        Ferrari Dino       Maserati Bora          Volvo 142E 
##             670.690             379.590             694.710             288.890
```

```
cor(mtcars)                                       # correlation of variables
```

```
##             mpg        cyl       disp         hp        drat         wt
## mpg   1.0000000 -0.8521620 -0.8475514 -0.7761684  0.68117191 -0.8676594
## cyl  -0.8521620  1.0000000  0.9020329  0.8324475 -0.69993811  0.7824958
## disp -0.8475514  0.9020329  1.0000000  0.7909486 -0.71021393  0.8879799
## hp   -0.7761684  0.8324475  0.7909486  1.0000000 -0.44875912  0.6587479
## drat  0.6811719 -0.6999381 -0.7102139 -0.4487591  1.00000000 -0.7124406
## wt   -0.8676594  0.7824958  0.8879799  0.6587479 -0.71244065  1.0000000
## qsec  0.4186840 -0.5912421 -0.4336979 -0.7082234  0.09120476 -0.1747159
## vs    0.6640389 -0.8108118 -0.7104159 -0.7230967  0.44027846 -0.5549157
## am    0.5998324 -0.5226070 -0.5912270 -0.2432043  0.71271113 -0.6924953
## gear  0.4802848 -0.4926866 -0.5555692 -0.1257043  0.69961013 -0.5832870
## carb -0.5509251  0.5269883  0.3949769  0.7498125 -0.09078980  0.4276059
##             qsec         vs          am       gear        carb
## mpg   0.41868403  0.6640389  0.59983243  0.4802848 -0.55092507
## cyl  -0.59124207 -0.8108118 -0.52260705 -0.4926866  0.52698829
## disp -0.43369788 -0.7104159 -0.59122704 -0.5555692  0.39497686
## hp   -0.70822339 -0.7230967 -0.24320426 -0.1257043  0.74981247
## drat  0.09120476  0.4402785  0.71271113  0.6996101 -0.09078980
## wt   -0.17471588 -0.5549157 -0.69249526 -0.5832870  0.42760594
## qsec  1.00000000  0.7445354 -0.22986086 -0.2126822 -0.65624923
## vs    0.74453544  1.0000000  0.16834512  0.2060233 -0.56960714
## am   -0.22986086  0.1683451  1.00000000  0.7940588  0.05753435
## gear -0.21268223  0.2060233  0.79405876  1.0000000  0.27407284
## carb -0.65624923 -0.5696071  0.05753435  0.2740728  1.00000000
```

```
# Histogram analysis
table(mtcars$cyl)                                 # get count of cars by no. of cylinders
```

```
## 
##  4  6  8 
## 11  7 14
```

```
table(mtcars$cyl, mtcars$gear)                    # get count of cars by no. of cylinders and no. of gears
```

```
##     
##       3  4  5
##   4   1  8  2
##   6   2  4  1
##   8  12  0  2
```

```
table(mtcars[c("cyl", "gear")])                   # a better syntax since column names are preserved
```

```
##    gear
## cyl  3  4  5
##   4  1  8  2
##   6  2  4  1
##   8 12  0  2
```

```
table(cyl=mtcars$cyl, gear=mtcars$gear)           # an alternative syntax
```

```
##    gear
## cyl  3  4  5
##   4  1  8  2
##   6  2  4  1
##   8 12  0  2
```

```
table(mtcars[c("cyl", "gear")], exclude=c("4"))   # exclude specified factors
```

```
##    gear
## cyl  3  5
##   6  2  1
##   8 12  2
```

```
# Quantile analysis
quantile(mtcars$mpg)                              # miles per gallon: get its 4 quartiles
```

```
##     0%    25%    50%    75%   100% 
## 10.400 15.425 19.200 22.800 33.900
```

```
quantile(mtcars$mpg, probs=seq(0,1,length=4))     # split the data into 3 quantiles (4 points)
```

```
##        0% 33.33333% 66.66667%      100% 
##      10.4      16.7      21.4      33.9
```

```
groups <- cut(mtcars$mpg,
              breaks=quantile(mtcars$mpg),        # mark data to the quartile where it belongs
              include.lowest = T)                 # include the lowest point in 1st quartile
table(groups, mtcars$cyl)                         # analyze no. of cylinders by mpg quartiles
```

```
##              
## groups        4 6 8
##   [10.4,15.4] 0 0 8
##   (15.4,19.2] 0 3 6
##   (19.2,22.8] 4 4 0
##   (22.8,33.9] 7 0 0
```

```
# Aggregation
aggregate(mpg ~ cyl, data=mtcars, summary)        # summary of miles per gallon grouped by no. of cylinders
```

```
##   cyl mpg.Min. mpg.1st Qu. mpg.Median mpg.Mean mpg.3rd Qu. mpg.Max.
## 1   4 21.40000    22.80000   26.00000 26.66364    30.40000 33.90000
## 2   6 17.80000    18.65000   19.70000 19.74286    21.00000 21.40000
## 3   8 10.40000    14.40000   15.20000 15.10000    16.25000 19.20000
```

```
ag <- aggregate(hp ~ .,                           # calculate mean of horsepower with cyl and gear as axes of ana
lysis
        data=mtcars[,c("hp","cyl","gear")],
        mean)
xtabs(hp ~ ., data=ag)                            # display the contingency table
```

```
##    gear
## cyl        3        4        5
##   4  97.0000  76.0000 102.0000
##   6 107.5000 116.5000 175.0000
##   8 194.1667   0.0000 299.5000
```

```
# Using third-party libraries
library(psych)
describe(mtcars)                                  # get more advanced stats
```

```
##      vars  n   mean     sd median trimmed    mad   min    max  range  skew
## mpg     1 32  20.09   6.03  19.20   19.70   5.41 10.40  33.90  23.50  0.61
## cyl     2 32   6.19   1.79   6.00    6.23   2.97  4.00   8.00   4.00 -0.17
## disp    3 32 230.72 123.94 196.30  222.52 140.48 71.10 472.00 400.90  0.38
## hp      4 32 146.69  68.56 123.00  141.19  77.10 52.00 335.00 283.00  0.73
## drat    5 32   3.60   0.53   3.70    3.58   0.70  2.76   4.93   2.17  0.27
## wt      6 32   3.22   0.98   3.33    3.15   0.77  1.51   5.42   3.91  0.42
## qsec    7 32  17.85   1.79  17.71   17.83   1.42 14.50  22.90   8.40  0.37
## vs      8 32   0.44   0.50   0.00    0.42   0.00  0.00   1.00   1.00  0.24
## am      9 32   0.41   0.50   0.00    0.38   0.00  0.00   1.00   1.00  0.36
## gear   10 32   3.69   0.74   4.00    3.62   1.48  3.00   5.00   2.00  0.53
## carb   11 32   2.81   1.62   2.00    2.65   1.48  1.00   8.00   7.00  1.05
##      kurtosis    se
## mpg     -0.37  1.07
## cyl     -1.76  0.32
## disp    -1.21 21.91
## hp      -0.14 12.12
## drat    -0.71  0.09
## wt      -0.02  0.17
## qsec     0.34  0.32
## vs      -2.00  0.09
## am      -1.92  0.09
## gear    -1.07  0.13
## carb     1.26  0.29
```

```
library(modeest)
```

```
## Registered S3 method overwritten by 'rmutil':
##   method         from 
##   plot.residuals psych
```

```
mfv(mtcars$cyl)                                       # most frequent value
```

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