R tutorial

Econometrics 322

Hang Miao

0. Assignment and Basics

```
n <- 15
n
```

```
## [1] 15
```

```
a = 12
a
```

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

```
24 -> z
z
```

```
## [1] 24
```

Variables must start with a letter, but may also contain numbers and periods. R is case sensitive.

```
N <- 26.42
N
```

```
## [1] 26.42
```

```
n
```

```
## [1] 15
```

To see a list of your objects, use ls( ). The ( ) is required, even though there are no arguments.

```
ls()
```

```
## [1] "a" "n" "N" "z"
```

Use rm to delete objects you no longer need.

```
rm(n)
```

```
ls()
```

```
## [1] "a" "N" "z"
```

You may see online help about a function using the help command or a question mark.

```
?ls
```

```
help(rm)
```

Several commands are available to help find a command whose name you don't know. Note that anything after a pound sign (#) is a comment and will not have any effect on R.

```
help.search("help") # "help" in name or summary; note quotes!
```

```
help.start() # also remember the R Commands web page
```

```
## starting httpd help server ... done
```

```
## If the browser launched by '/usr/bin/open' is already running, it
##     is *not* restarted, and you must switch to its window.
## Otherwise, be patient ...
```

Other data types are available. You do not need to declare these; they will be assigned automatically.

name <- "Mike" # Character data name

```
q1 <- TRUE # Logical data
q1
```

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

```
q2 <- F
q2
```

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

1. Simple calculation

R may be used for simple calculation, using the standard arithmetic symbols +, -, *, /, as well as parentheses and ^ (exponentiation).

```
a <- 12+14
a
```

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

```
3*5
```

```
## [1] 15
```

```
(20-4)/2
```

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

```
7^2
```

```
## [1] 49
```

Standard mathematical functions are available.

```
exp(2)
```

```
## [1] 7.389056
```

```
log(10) # Natural log
```

```
## [1] 2.302585
```

```
log10(10) # Base 10
```

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

```
log2(64) # Base 2
```

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

```
pi
```

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

```
cos(pi)
```

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

```
sqrt(100) # square root
```

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

2. Vectors

Vectors may be created using the c command, separating your elements with commas.

```
a <- c(1, 7, 32, 16)
a
```

```
## [1]  1  7 32 16
```

Sequences of integers may be created using a colon (:).

```
b <- 1:10
b
```

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

```
c <- 20:15
c
```

```
## [1] 20 19 18 17 16 15
```

Other regular vectors may be created using the seq (sequence) and rep (repeat) commands.

```
d <- seq(1, 5, by=0.5)
d
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
e <- seq(0, 10, length=5)
e
```

```
## [1]  0.0  2.5  5.0  7.5 10.0
```

```
f <- rep(0, 5)
f
```

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

```
g <- rep(1:3, 4)
g
```

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

```
h <- rep(4:6, 1:3)
h
```

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

Random vectors can be created with a set of functions that start with r, such as rnorm (normal) or runif (uniform).

```
x <- rnorm(5) # Standard normal random variables
x
```

```
## [1] -0.3285511  0.5440755  1.1188503  1.2524622  0.5721741
```

```
y <- rnorm(7, 10, 3) # Normal r.v.s with mu = 10, sigma = 3
y
```

```
## [1]  5.429975 10.097136 14.642965  9.096982 11.271690 11.585727 12.916732
```

```
z <- runif(10) # Uniform(0, 1) random variables
z
```

```
##  [1] 0.9056515 0.7958397 0.7443220 0.7062697 0.7093662 0.7533793 0.3966831
##  [8] 0.1803170 0.1504333 0.4823195
```

If a vector is passed to an arithmetic calculation, it will be computed element-by-element.

```
c(1, 2, 3) + c(4, 5, 6)
```

```
## [1] 5 7 9
```

```
sqrt(c(100, 225, 400))
```

```
## [1] 10 15 20
```

If the vectors involved are of different lengths, the shorter one will be repeated until it is the same length as the longer.

```
c(1, 2, 3, 4) + c(10, 20)
```

```
## [1] 11 22 13 24
```

```
c(1, 2, 3) + c(10, 20)
```

```
## Warning in c(1, 2, 3) + c(10, 20): longer object length is not a multiple
## of shorter object length
```

```
## [1] 11 22 13
```

To select subsets of a vector, use square brackets ([ ]).

```
d
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
d[3]
```

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

```
d[5:7]
```

```
## [1] 3.0 3.5 4.0
```

A logical vector in the brackets will return the TRUE elements.

```
d > 2.8
```

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

```
d[d > 2.8]
```

```
## [1] 3.0 3.5 4.0 4.5 5.0
```

The number of elements in a vector can be found with the length command.

```
length(d)
```

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

```
length(d[d > 2.8])
```

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

3. Simple statistics

There are a variety of mathematical and statistical summaries which can be computed from a vector.

```
1:4
```

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

```
sum(1:4)
```

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

```r
prod(1:4) # product
```

```
## [1] 24
```

```r
24
```

```
## [1] 24
```

```r
max(1:10)
```

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

```r
min(1:10)
```

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

```r
range(1:10)
```

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

```r
X <- rnorm(10)
X
```

```
##  [1]  1.1309652 -0.5398528 -0.8574352  1.9165319  0.8138337 -1.2700377
##  [7]  0.5860495  0.3706055  0.4214494 -1.5509829
```

```r
mean(X)
```

```
## [1] 0.1021127
```

```r
sort(X)
```

```
##  [1] -1.5509829 -1.2700377 -0.8574352 -0.5398528  0.3706055  0.4214494
##  [7]  0.5860495  0.8138337  1.1309652  1.9165319
```

```r
median(X)
```

```
## [1] 0.3960275
```

```r
var(X)
```

```
## [1] 1.245982
```

```r
sd(X)
```

```
## [1] 1.116235
```

** 4. Matrices **

Matrices can be created with the matrix command, specifying all elements (column-by-column) as well as the number of rows and number of columns.

```r
A <- matrix(1:12, nr=3, nc=4)
A
```

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

You may also specify the rows (or columns) as vectors, and then combine them into a matrix using the rbind (cbind) command.

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

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

```
b <- c(10, 20, 30)
b
```

```
## [1] 10 20 30
```

```
c <- c(100, 200, 300)
c
```

```
## [1] 100 200 300
```

```
d <- c(1000, 2000, 3000)
d
```

```
## [1] 1000 2000 3000
```

```
B <- rbind(a, b, c, d)
B
```

```
##   [,1] [,2] [,3]
## a    1    2    3
## b   10   20   30
## c  100  200  300
## d 1000 2000 3000
```

```
C <- cbind(a, b, c, d)
C
```

```
##      a  b   c    d
## [1,] 1 10 100 1000
## [2,] 2 20 200 2000
## [3,] 3 30 300 3000
```

To select a subset of a matrix, use the square brackets and specify rows before the comma, and columns after.

```
C[1:2,]
```

```
##      a  b   c    d
## [1,] 1 10 100 1000
## [2,] 2 20 200 2000
```

```
C[,c(1,3)]
```

```
##      a   c
## [1,] 1 100
## [2,] 2 200
## [3,] 3 300
```

```
C[1:2,c(1,3)]
```

```
##      a   c
```

```
## [1,] 1 100
## [2,] 2 200
```

Matrix multiplication is performed with the operator %*%. Remember that order matters!

```
B%*%C
```

```
##       a      b       c       d
## a    14    140    1400 1.4e+04
## b   140   1400   14000 1.4e+05
## c  1400  14000  140000 1.4e+06
## d 14000 140000 1400000 1.4e+07
```

```
C%*%B
```

```
##         [,1]    [,2]    [,3]
## [1,] 1010101 2020202 3030303
## [2,] 2020202 4040404 6060606
## [3,] 3030303 6060606 9090909
```

You may apply a summary function to the rows or columns of a matrix using the apply function.

```
C
```

```
##      a  b   c    d
## [1,] 1 10 100 1000
## [2,] 2 20 200 2000
## [3,] 3 30 300 3000
```

```
sum(C)
```

```
## [1] 6666
```

```
apply(C, 1, sum) # apply sum function on each row
```

```
## [1] 1111 2222 3333
```

```
apply(C, 2, sum) # apply sum function on each column
```

```
##    a    b    c    d
##    6   60  600 6000
```

```
rowSums(C)
```

```
## [1] 1111 2222 3333
```

```
colSums(C)
```

```
##    a    b    c    d
##    6   60  600 6000
```