R Programming, Week 3

Denis Vrdoljak

Variables

```
> a <- ‘apple’
> a
 [1] ‘apple’
```

Numeric

- # R has all the functionality of a calculator, like most other languages
- > a = 3- 1
- > a
- [1] 2

- # The “<-” operator is the same as the “=“ operator
- > a <- 1 + 1
- >a
- [1] 2

- # operators are + (addition), - (subtraction), / (division), * (multiplication), and ^ (raise to a power).
- > a = 1 + 1
- >a
- [1] 2

Algebra

- > b = 1 + 2 / 3 - 2 * 6.5
- > b
- [1] -11.33333

- > b = 1 * (2 / (1 + 1))
- > b
- [1] 1

- > b = 1 * 2 / 1 + 1
- > b
- [1] 3

Math Functions

`log(x)`	Natural log.	`sum(x)`	Sum.
`exp(x)`	Exponential.	`mean(x)`	Mean.
`max(x)`	Largest element.	`median(x)`	Median.
`min(x)`	Smallest element.	`quantile(x)`	Percentage quantiles.
`round(x, n)`	Round to n decimal places.	`rank(x)`	Rank of elements.
`signif(x, n)`	Round to n significant figures.	`var(x)`	The variance.
`cor(x, y)`	Correlation.	`sd(x)`	The standard deviation

Example:

```
> sqrt(5)
[1] 2.236068

> sum (5, 6, 7, 8, 9)
> [1] 35

> sqrt (144)
> [1] 12
```

Statistics

`lm(x ~ y, data=df)`
Linear model.

`glm(x ~ y, data=df)`
Generalised linear model.

`summary`
Get more detailed information out a model.

`t.test(x, y)`
Preform a t-test for difference between means.

`pairwise.t.test`
Preform a t-test for paired data.

`prop.test`
Test for a difference between proportions.

`aov`
Analysis of variance.

Character

- # Anything contained in quotation marks I called a character string
- > a = "hello world"
- > a
- [1] "hello world"

- # number in quotation mark is considered a character, not an integer
- > b = "7"
- > b
- [1] "7"

- > c = “1 + 1”
- > c
- [1] 1 +1

Date

- > as.Date("2019-10-8")
- [1] "2019-10-08"

Converting

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

`as.logical`	`TRUE, FALSE, TRUE`	Boolean values (TRUE or FALSE).
`as.numeric`	`1, 0, 1`	Integers or floating point numbers.
`as.character`	`'1', '0', '1'`	Character strings. Generally preferred to factors.
`as.factor`	`'1', '0', '1', levels: '1', '0'`	Character strings with preset levels. Needed for some statistical models.

Example:

> > a = "5"

> a

> as.integer (a)

[1] 5

> b = 5

> as.character (b)

[1] "5"

> c = "2019-10-8"

> as.Date (c)

[1] "2019-10-08"

Vectors

- # vector is a string of numbers
- > vector1 <- 1:9
- > vector1
- [1] 1 2 3 4 5 6 7 8 9

- # colon operator creates a sequence of numbers from left to the right
- > a <- 1.2:9

[1] 1.2 2.2 3.2 4.2 5.2 6.2 7.2 8.2

- # specifically selected numbers
- > vector3 = 1:9
- > vector3
- [1] 1 2 3 4 5 6 7 8 9

- > vector2 = c(1, 3, 2, -8.1)
- > vector2
- [1] 1.0 3.0 2.0 –8.1

Creating Vectors

C(2,4,6)	2 4 6	Join elements into a vector
2:6	2 3 4 5 6	An interger sequence
Seq(2,3,by=0.5)	2.0 2.5 3.0	A complex sequence
Rep(1:2, times=3)	1 2 1 2 1 2	Repeat a vector
Rep(1:2, each=3)	1 1 1 2 2 2	Repeat elements of a vector

Selecting Vector Elements

By Position	
`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.
By Value	
`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.
Named Vectors	
`x['apple']`	Element with name 'apple'.

Vector Functions

sort(x)

return x sorted

rev(x)

return x reversed

table(x)

see counts of values

unique(x)

see unique values

Example:

```
vec <- 1:10
> vec
[1] 1 2 3 4 5 6 7 8 9 10
> vec_rev <- rev(vec)                    # Apply rev function to vector
> vec_rev                                # Print reversed example vector
[1] 10  9  8  7  6  5  4  3  2  1
```

Strings

`paste(x, y, sep = ' ')`	Join multiple vectors together.
`paste(x, collapse = ' ')`	Join elements of a vector together.
`grep(pattern, x)`	Find regular expression matches in x.
`gsub(pattern, replace, x)`	Replace matches in x with a string.
`toupper(x)`	Convert to uppercase.
`tolower(x)`	Convert to lowercase.
`nchar(x)`	Number of characters in a string.

Example:

> paste (“a” , “b” , sep = “-“)
[1] a-b

> x = 1:15
> x
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
> grep (5 , x)
[1] 5 15

Factors

`factor(x)`	`cut(x, breaks = 4)`
Turn a vector into a factor. Can set the levels of the factor and the order.	Turn a numeric vector into a factor but 'cutting' into sections.

Lists

```
l <- list(x = 1:5, y = c('a', 'b'))
```

A list is collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Example:

```
> list_a <- list("Red", "Green", c(21,32,11), TRUE)
> list_a
[[1]]
[1] "Red"

[[2]]
[1] "Green"

[[3]]
[1] 21 32 11

[[4]]
[1] TRUE
```

Reading and Writing Data

Input	Ouput	Description
`df <- read.table('file.txt')`	`write.table(df, 'file.txt')`	Read and write a delimited text file.
`df <- read.csv('file.csv')`	`write.csv(df, 'file.csv')`	Read and write a comma separated value file. This is a special case of read.table/ write.table.
`load('file.RData')`	`save(df, file = 'file.Rdata')`	Read and write an R data file, a file type special for R.

Conditions								
	`a == b`	Are equal	`a > b`	Greater than	`a >= b`	Greater than or equal to	`is.na(a)`	Is missing
	`a != b`	Not equal	`a < b`	Less than	`a <= b`	Less than or equal to	`is.null(a)`	Is null

Matrixes

```
m <- matrix(x, nrow = 3, ncol = 3)
```
Create a matrix from x.

`m[2, ]` - Select a row

`m[ , 1]` - Select a column

`m[2, 3]` - Select an element

`t(m)`
Transpose

`m %*% n`
Matrix Multiplication

`solve(m, n)`
Find x in: m * x = n

Create a Matrix

- # matrix() function takes the data and the number of rows (nrow) and makes a matrix by filling down each column from the left to the right

- > matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3)
- [,1] [,2] [,3]
- [1,] 1 4 7
- [2,] 2 5 8
- [3,] 3 6 9

- > matrix(1:8, ncol = 2)
- [,1] [,2]
- [1,] 1 5
- [2,] 2 6
- [3,] 3 7
- [4,] 4 8

Manipulate a Matrix

- > matrix1
- [,1] [,2] [,3]
- [1,] 1 4 7
- [2,] 2 5 8
- [3,] 3 6 9
-

- > matrix1[1, 3]
- [1] 7

- > matrix1[2,]
- [1] 2 5 8

- # the result is returned as a mathematical vector

Change a Matrix

- > m = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3)
- > m
- [,1] [,2] [,3]
- [1,] 1 4 7
- [2,] 2 5 8
- [3,] 3 6 9

- # To change use operator “=”
- > m[,2] = 1
- > m
- [,1] [,2] [,3]
- [1,] 1 1 7
- [2,] 2 1 8
- [3,] 3 1 9

- # algebra
- > m + 2
- [,1] [,2] [,3]
- [1,] 3 3 9
- [2,] 4 3 10
- [3,] 5 3 11

- # To remove a column use operator “-”
- > m[,-2]
- [,1] [,2]
- [1,] 1 7
- [2,] 2 8
- [3,] 3 9

Data Frames

```
df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))
```

A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

`df$x` 

`df[[2]]` 

Understanding a data frame

`View(df)` See the full data frame.

`head(df)` See the first 6 rows.

Matrix subsetting

`df[ , 2]` 

`df[2, ]` 

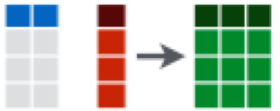
`df[2, 2]` 

`nrow(df)` Number of rows.

`ncol(df)` Number of columns.

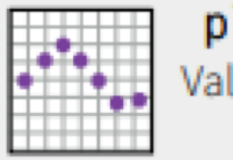
`dim(df)` Number of columns and rows.

`cbind` - Bind columns.

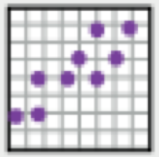


`rbind` - Bind rows.

Plotting



`plot(x)`
Values of x in order.



`plot(x, y)`
Values of x against y.

`hist(x)`
Histogram of x.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poison	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

For Loop

```
for (variable in sequence){
        Do Something

}
```

Example:

```
for i in 1:5 {
        print (i)
}
```

Output:

1
2
3
4
5

While Loop

```
while(condition){
        Do something
}
```

Example:

```
I = 0
While (I <= 5){
        print (i)
        I = I + 1}
```

Output:

1
2
3
4
5

IF Statement

```
if(condition){
        Do Something
} else {
        Do Something
}
```

Example:

```
var = 5
if(class(var)){
        printf (“A character string”)
} else {
        printf (“Not a character”)
}
```

Output:
Not a character

Functions

```
function_name <- function(var){
        Do something
        return(new_variable)
}
```

Example:

```
square_function <- function (x)
{
      # compute the square of x
      y = sqrt (x)
      return (y)
}

> square_function (144)
[1] 12
```

Environment

ls()	List all variable in environment
rm(x)	Remove x from the environment
rm(list=ls())	Remove all variables from environment

Example
> ls()
[1] “a”
[2] “b”
[3] “x”

>rm(b)
> ls()
[1] “a”
[3] “x”