UseRs! An introduction to R

Class 1 – R Overview

1. A few concepts before starting

Object-oriented language: variables, data, functions, results, etc, are stored in the active memory of the computer in the form of *object*, which have a *name*. The user can do actions on these objects with *operators* (arithmetic, logical and comparison), and with *functions* (which are themselves objects)

R uses an interpreted language, not a compiled one: commands typed on the keyboard are directly executed. The user executes functions via some commands and the results are displayed in the screen, stored in an object, or written to the disk. Don't be scared!! R language is very intuitive…

2. Creating objects in memory

2.1. What's an object?

The simplest ways to understand what an object is, is to type the name of an object to display its content.

Assign operator: arrow with a minus sign or equal sign. Either is OK.

```
> n = 15

> n

[1] 15
```

If the object already exists, its previous value is erased and replaced by the new one.

```
> n = 15 + 2

> n

[1] 17
```

Note that is R is case sensitive.

2.2. Objects

The four most frequently used types of data objects in R are vectors,

matrices, data frames, and lists.

The objects are characterized by their names, their content and their attributes, which specify the kind of data represented by an object.

2.2.1.Vector

A *vector* represents a set of elements of the same mode.

Create a vector with the function c() (concatenate), which binds elements together, whether they are of character form, numeric or logical. with the function seq() (sequence generation).

```
> x = c(4,8,16,9)
> x
[1] 4 8 16 9
> y= seq(1:10)
> y
[1] 1 2 3 4 5 6 7 8 9 10
> z=seq(from=2,to=2,by=2)
> z
[1] 2 4 6 8 10
```

The objects are characterized by their names, their content and their attributes, which specify the kind of data represented by an object.

All objects have two intrinsic attributes, the *length* and the *mode*.

➔ The length is the number of elements of the object.

```
> length(y)
> [1] 10
```

➔ The mode is the basic type of the elements of the object.

```
> mode(y)
> [1] numeric
```

There are four different modes: numeric, character, complex and logical.

```
> a =   Welcome to the UseRs community!   ; b=TRUE; z =1i
> mode (a); mode (b); mode(z)
```

```
[1]   character

[1]   logical

[1]   complex
```

(Characters must be always surrounded by #quotation marks)

NULL objects are empty objects with no assigned mode. They have a length of zero.

```
> nn=vector()

> nn

[1] NULL
```

➔ Other types of objects representing data.

2.2.2.Matrix

A *matrix* is a set of elements appearing in rows and columns where the elements are of the same mode whether they are logical, numeric (integer or double), complex or character.

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

The option byrow indicates whether the values given by data must fill successively the columns (the default) or the rows (if TRUE). The option dimnames allows us to give names to the rows and columns.

```
> matrix(data=5, nr=2, nc=2)
    [,1][,2]
[1,]  5       5
[2,]  5       5


> matrix(1:6, 2, 3)
    [,1] [,2] [,3]
[1,]  1   3    5
[2,]  2   4    6


> matrix(1:6, 2, 3, byrow=TRUE)
    [,1] [,2] [,3]
[1,]  1   2    3
[2,]  4   5    6
```

```
```

2.2.3.Data Frame

A *data frame* is similar to a matrix object but the columns can be of different modes.

You can create a data frame with the function `data.frame`. The vectors included in the data frame must be of the same length.

Create a data frame with two vectors of the same length.

It is possible to change the names of the columns with `data.frame(A1=x, A2=n)`. One can also give names to the rows with the option row.names which must be, of course, a vector of mode character and of length equal to the number of lines of the data frame. Finally, note that data frames have an attribute dim similarly to matrices.

2.2.4.Lists

A *list* is a generalization of a vector and represents a collection of data objects.
Lists can be created using the list function. Like data frames, they can incorporate a
mixture of modes into the one list and each component can be of a different length or size. For example, the following is an example of how we might create a list from scratch.

```
> L1 <- list(x = sample(1:5, 20, rep=T),
y = rep(letters[1:5], 4), z = rpois(20, 1))
```

3. Reading data in a file

The function `read.table()` has for effect to create a data frame, and is the main way to read data in tabular form.

Several ways to specify the path:

In a Mac

```
> mydata=read.table("~/Desktop/UseRs Seminar
EEB/class1/treedata.txt", h=T)
```

In a PC

```
> tree = read.table("C:/&.
```

To change the working directory use the command `setwd()`

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

or apple D

```
> mydata=read.table(  treedata.txt  ,h=T)
```

To know what is the directory the command `getwd()` can be used

`read.table()`creates a data.frame named mydata. h=T, means header=TRUE, that is, the first row of your data corresponds to the column names.

```
> head(mydata)
```

If you don't specify `h=T`, each variable will be named, by default V1, V2, V3, and can be accessed individually by

Row and column names are already assigned to a data frame but they may be changed using the `names` and `row.names` functions.

```
> names(mydata)

> row.names(mydata)

> col.names(mydata)
```

Other arguments in the function `read.table()`(See R for beginners, p 11.).

4. Accessing the values of an object: the indexing system

The indexing system is an efficient and flexible way to access selectively the elements of an object; it can be either *numeric* or *logical*.

```
> x <- sample(1:5, 20, rep=T)
> x
[1] 3 4 1 1 2 1 4 2 1 1 5 3 1 1 1 2 4 5 5 3

> x[1:5]
[1] 3 4 1 1 2

> x == 1
[1] FALSE FALSE TRUE TRUE FALSE TRUE FALSE FALSE TRUE
[10] TRUE FALSE FALSE TRUE TRUE TRUE FALSE FALSE FALSE
[19] FALSE FALSE

> x = 1

> x > 1
```

Replacing values:

```
> x[x==1] <- 0
> x
[1] 3 4 0 0 2 0 4 2 0 0 5 3 0 0 0 2 4 5 5 3
```

Replace the values > 1 by 0

If x is a matrix or a data frame, the value of the *i*th line and *j*th column is accessed with `x[i, j]`.

Select the columns 6 to 7 and the rows 100 to 105 from the dataset `mydata`

5. Generating data

Explore the rep and seq functions using `?rep` *and* `?seq`

Generate this vector using the rep, seq functions inside the concatenation function to create the vector

```
> x
[1]  5.0  5.0  5.0  5.0 20.0 20.2 20.4 20.6 20.8 21.0  5.0  5.0
6.0  6.0
```

Create a vector `y` *of length = 15, then a data frame,* `df`*, with* `x` *and* `y`.

```
> z=1+2*x+rnorm(x)

> df = data.frame(df, z=z)

> attach(rdata)


> lm1 = lm(y~x, data=df)
> summary(lm1)

> lm2 = lm(z~x, data=df)
> summary (lm2)
> plot(x,y)

> plot(x,z)
```