COMP226: Slides 03

Functions in R; the apply family

Rahul Savani

rahul.savani@liverpool.ac.uk

Overview

- The **working directory**
- Defining **functions in R**
- **Matrices and data.frames**
- The **apply** family

Setting the working directory

If you want to source a file, **R needs to be able to find it**

Assume we start R in `/tmp` and want to source `/tmp/R/test.R`

```
> source('/tmp/R/test.R') # specify full path
```

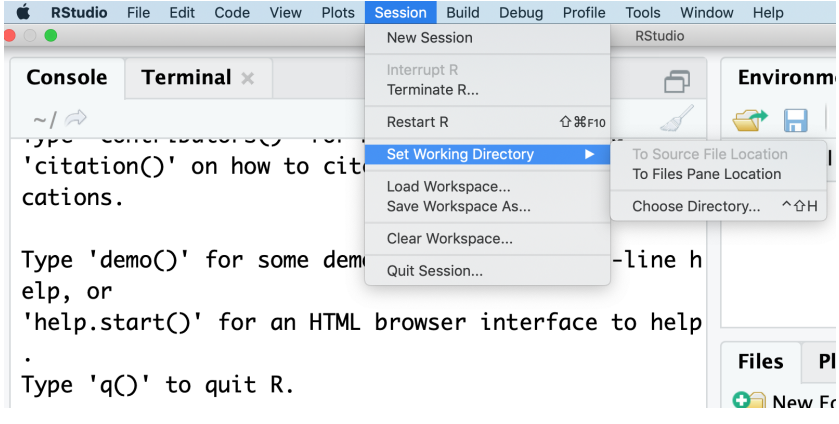
Additionally, R always has a **working directory** and looks there:

```
> getwd(); list.files(); list.dirs()
[1] "/tmp"
[1] "R"
[1] "."   "./R"

> setwd("/tmp/R"); list.files(); source("test.R")
[1] "test.R"
[1] "test.R sourced :-)"
```

Alternative in RStudio

In RStudio you can also do it via the menu in the gui:



Example of a function

The following is an example of a function that computes the mean of the components of a vector (or list)

```
> myMean <- function(input) {
    sum <- 0
    for (x in input)
        sum <- sum + x
    return(sum/length(input))
  }
> myMean(1:10)
[1] 5.5
> sum(1:10)/10 # R naturally has a built-in sum function!
[1] 5.5
> mean(1:10) # R naturally has a built-in mean function!
[1] 5.5
```

Example of a function

```
> myMean <- function(input) {
    sum <- 0
    for (x in input)
        sum <- sum + x
    return(sum/length(input))
  }
```

Note we **did not** specify the type of the argument input

Be careful to make sure you pass the expected arguments

```
> myMean(c("A","B","C"))
Error in sum + x : non-numeric argument to binary operator
```

Default arguments

We can specify a default for an argument as above

```
> myMean <- function(input=1:5) {
    sum <- 0
    for (x in input)
        sum <- sum + x
    return(sum/length(input))
}

> print(myMean()) # default for input is used
[1] 3

> print(myMean(1:10)) # override default
[1] 5.5
```

Matrices

matrices are 2-dimensional vectors:

```
m <- matrix(1:10, ncol=5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10

> m[1,2]; m[1,3:4]; m[1,3:4,drop=FALSE]
[1] 3
[1] 5 7
     [,1] [,2]
[1,]    5    7
```

Data frames

A *data.frame* shares the properties of matrices and lists; unlike a matrix a `data.frame` can have columns with different types of data

```
> let <- LETTERS[1:3]
> let
[1] "A" "B" "C"

> data.frame(x = 1, y = 1:3, z=let)
  x y z
1 1 1 A
2 1 2 B
3 1 3 C

> data.frame(1, 1:3, let, row.names=paste("Row",let))
      X1 X1.3 let
Row A  1    1   A
Row B  1    2   B
Row C  1    3   C
```

Reading a data.frame from a file

```
> df = read.csv('prices.csv',row.names=1)
> df
     Stock1 Stock2 Stock3
Day1  17.34   1.32    612
Day2  19.43   1.31    580
Day3  15.64   1.22    695
Day4  15.66     NA    690

> is.data.frame(df)
[1] TRUE

> df["Day1", 2:3]
     Stock2 Stock3
Day1   1.32    612
```

Apply family of functions in R

- the `apply` family serve as an **alternatives to loops** (similar to *list comprehension* in python; and *map* in functional programming)
- **apply functions to elements of data structures**:
    - lapply (for lists/vectors)
    - sapply (simplified version of lapply)
    - apply (for matrices)
    - mapply (for multiple lists/vectors/matrices)
- **It's important that you are comfortable with their use**

Example: lapply/sapply

`lapply` returns a list:

```
> lst <- list(c(1,2,3), c(2,3,4), c(0,10)); lapply(lst, mean)
[[1]]
[1] 2

[[2]]
[1] 3

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

The `s` in `sapply` stands for **simple**; `sapply` simplifies the output as much as possible, now returning a vector, not list:

```
> sapply(lst, mean)
[1] 2 3 5
```

Example: apply

`apply(X,MARGIN,FUN)` applies `FUN` to rows (`MARGIN=1`), columns (`MARGIN=2`), or both (`MARGIN=c(1,2)`) of a **matrix**/**data frame**

```
> df
     Stock1 Stock2 Stock3
Day1  17.34   1.32    612
Day2  19.43   1.31    580
Day3  15.64   1.22    695
Day4  15.66     NA    690

> apply(df, mean, MARGIN=2)
 Stock1   Stock2   Stock3
17.0175       NA 644.2500

> apply(df, mean, MARGIN=1)
    Day1     Day2     Day3     Day4
210.2200 200.2467 237.2867       NA
```

Example: mapply

mapply is multivariate apply:

```
> mapply(rep, 1:3, 3:1)
```

What do you think the result will be?

Example: mapply

mapply is multivariate apply:

```
> mapply(rep, 1:3, 3:1)
[[1]]
[1] 1 1 1

[[2]]
[1] 2 2

[[3]]
[1] 3

# equivalent to
# list(rep(1, 3), rep(2, 2), rep(3, 1))
```

Summary

Function	Objective	Input	Output
apply(x, MARGIN, FUN)	Apply FUN to rows/columns or both	data frame or matrix	vector, list, array
lapply(X, FUN)	Apply FUN to all elements of input	list, vector or data frame	list
sapply(X, FUN)	Apply FUN to all elements of input	list, vector or data frame	vector or matrix
mapply(FUN, ...)	Apply FUN to all elements of inputs	set comprising lists, vectors or data frame	vector or matrix

We will see **lots more examples throughout the module**...