R Programming

Coursera Course by John Hopkins University

INSTRUCTORS: Dr. Jeff Leek, Dr. Roger D. Peng, Dr. Brian Caffo

---

Overview of R, R data types and objects, reading and writing data

Installing R & RStuido

- This was covered in the previous course.

Swirl

- swirl teaches you R programming and data science interactively, at your own pace, and right in the R console.

- Start swirl
  - install the package "swirl" if you haven't yet

  - Everytime you want to run swirl execute:
    * library("swirl")

    * swirl()

  - You'll then be prompted to install a course

  - Help page for swirl

History of S and R programming

- What is S?
  - R is a dialect of S

  - S was developed by John Chambers and others at Bell Labs

  - Initiated in 1976 as an internal statistical analysis environment, implemented as Fortran libraries
    * Early versions did not contain functions for statistical modeling

  - Version 3 was released in 1988, which was rewritten in C and began to resemble the system that we have today.

  - Version 4 was released in 1998 and is the version we use today.

    - This version is documented in *Programming with Data* by John Chambers (the green book)
  - Insightful sells its implementation of the S language under the name *S-PLUS*, which includes a number of fancy features, mostly GUIs.
  - S won the Association for Computing Machinery's Software System Award in '98

  - (More about S)[https://web.archive.org/web/20181014111802/ect.bell-labs.com/sl/S/]
- What is R?
  - R was developed by Ross Ihaka and Robert Gentleman, they documented thier experience in a (1996 JCGS paper)[https://amstat.tandfonline.com/doi/abs/10.1080/10618600.1996.10474713].

  - In 1995, R become free software after Martin Machler convinced Ross & Robert to use the GNU (General Public License)

  - Versions
    - R version 1.0.0 was released in 2000

    - R version 3.0.2 is released in Dec. 2013

  - Syntax is similar to S, making it easy for S-PLUS users to switch over

  - Runs on almost any standard computing platform/OS (even on the PS3)

  - Frequent releases; active development and communities

  - Funtionality is divided into modular packages as to keep it "lean"

  - It's free!

  - What is free about Free Software?
    - Freedom 0: freedom to run the program, for any purpose

    - Freedom 1: freedom to study how the program works, and adapt it to one's needs. Which implies access to the source code
    - Freedom 2: freedom to redistribute copies
    - Freedom 3: freedom to improve the program, and release your improvements to the public, or to sell them.
    - These are outlined by the (Free Software Foundation)[https://www.fsf.org/]
- Drawbacks of R
  - Essentially based on 40 year old technology,the original S language
  - Little build support for dynamic or 3D graphics. Although there are packages for such
  - Functionality is based on consumer demand and use contributions, if a feature is not present you'll have to build it.

  - Objects that are manipulated in R have to be stored in the physical memory of the computer, as such if an object is bigger than the memory you'll be unable to load it into memory

  - Not ideal for all possible situations, such as calling to order pizza (but this is a drawback of all software packages)

*Design of the R System
+ "base" R system that can be downloaded from (CRAN)[http://cran.r-project.org] (krey-an) which…
- contains the packages: **utils, stats, datasets, graphics, grDevices, grid, methods, tools, parallel, compiler, splines, tcltk, stats4**.
- and "Recommends" the packages: **boot, class, cluster, codetools, foreign, KernSmooth, lattice,**

mgcv, nlme, rpart, survival, MASS, spatial, nnet, Matrix.
+ Packages are available all around the web, but packages on CRAN have to meet a certain level of quality.

- Some Useful Books on S/R
  - Chambers (2008). *Software for Data Analysis*, Springer.
  - Chambers (1998). *Programming with Data*, Springer.
  - Venables & Ripley (2002). *Modern Applied Statistics with S*, Springer.
  - Venables & Ripley (2000). *S Programming*, Springer.
  - Pinheiro & Bates (2000). *Mixed-Effects Models in S and S-Plus*, Springer.
  - Murrell (2005). *R Graphics*, Chapman & Hall/CRC Press.
  - (Additional Books)[http://www.r-project.org/doc/bib/R-books.html]

Review of getting help

- Covered in previous course

Input and Evaluation: Vocabulary/Syntax

- **Expressions -** The code that is typed into the R prompt.

- **Assignment Operator -** assigns a value to a symbol, Ex:
  `x <- 1`

- *Output a variable*:

```
x <- 36
print(x) ##explicit printing
```

```
## [1] 36
```

```
## or one can just type the variable
x ##auto-printing
```

```
## [1] 36
```

- *Comment:* Use a Hash(#) symbol to make a comment to the right of #

- *[1]* is indicating the following variable is the first element of the vector

```
x <- 1:30 ##Loads x with the numbers 1 to 30
print(x)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30
```

```
## here, [26] is telling you the next number is the 26th element of the vector
```

- `Inf` - represents infinity and can be used in ordinary calculations (Ex: 1 / `Inf` is 0)

- `Nan` - represents an undefined value ("not a number") (Ex: 0/0 is `NaN`).

    - Can also be thought of as a missing value

- **Attributes -** Some objects in R come with attributes. These attributes can be set or modified with the expression `attributes()`. They are:
    - names, dimnames (dimension names)
    - dimensions (e.g. matrices, arrays) - number of rows & cols, or more depending on dimensions of array
    - class - the data type of the object
    - length - number of elements
    - other user-defined attributes/metadata can be added

- **Coercion -** occurs so that every element of a vector is of the same class (Covered further in Vector section)

Differences between atomic data types

- R has five basic, or "atomic", classes of objects:
    - character
        * In `R` there is no `string` data type. It is also considered part of the `character` data type
    - numeric (real numbers)
        * R thinks as numbers as these by default

    - integer
        * Must be explicity declared with the L suffix; `x <- 1` assigns a numeric object, but `x <- 1L` explicitly assigns an integer

    - complex

    - logical (True/False)
- A vector can only contain objects of the same class
    - an empty vector can be created with `vector()`

- However, a **list** is represented as a vector but can contain objects of different classes (as such we usually use these)

Vectors, Lists, and Matrices

- The `c()` function (can be thought to stand for "concatenate")
    - Can be used to create vectors of objects

```
x <- c(0.5, 0.6) ## numeric
x <- c(TRUE, FALSE) ## logical
x <- c(T, F) ## logical
x <- c("a", "b", "c") ## character
x <- c(1+0i, 2+4i) ## complex
```

- The `vector()` function
    - Can also be used to create, you guessed it, vectors

```
x <- vector() ## Creates an empty vector
x ## Prints as code that evaluates as FALSE
```

```
## logical(0)
```

```
x <- vector(mode = "numeric", length = 10) ## Creates a vector with length "10" of
## numeric data type,default value is 0
x
```

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

```
x <- vector("numeric", 5) ##The parameter names are not required, but can easily clarify code
x
```

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

- When different objects are mixed in a vector, **coercion** occurs so all objects are of the same class.
  - R will implicitly create the "Least Common Denominator" of the mixed classes

```
y <- c(1.7, "a") ## character
y
```

```
## [1] "1.7" "a"
```

```
y <- c(TRUE, 2) ## numeric
y
```

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

```
y <- c("a", TRUE) ## character
y
```

```
## [1] "a"    "TRUE"
```

```
y[2] ## "TRUE" is a string stored as a "character" data type
```

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

```
y[3] ## The third element does not exist
```

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

- Objects can be **explicitly coerced** from onne class to another using the `as.*` functions, if available.
  - Nonsensical coercion reasults in `NA`s

```
x <- 0:6
class(x)
```

```
## [1] "integer"
```

```
as.numeric(x)
```

```
## [1] 0 1 2 3 4 5 6
```

```
as.logical(x)
```

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

```
as.character(x)
```

```
## [1] "0" "1" "2" "3" "4" "5" "6"
```

```
as.complex(x)
```

```
## [1] 0+0i 1+0i 2+0i 3+0i 4+0i 5+0i 6+0i
```

```
x
```

```
## [1] 0 1 2 3 4 5 6
```

```r
y <- as.character(x)
y
```

```
## [1] "0" "1" "2" "3" "4" "5" "6"
```

```r
x <- c("a", "b", "c")
as.numeric(x) ##Nonsensical coercion will also show a warning
```

```
## Warning: NAs introduced by coercion
```

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

```r
as.logical(x)
```

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

```r
as.complex(x)
```

```
## Warning: NAs introduced by coercion
```

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

- Lists (Important data type in R that you should get to know well)
  - Lists are a type of vector that can contain elements of different classes.
  - Doesn't print like a vector because every element is different
    - prints index of element with double brackets bordering it: `[[1]]`

```r
x <- list(1, "a", TRUE, 1 + 4i, 16 +18i)
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+4i
##
## [[5]]
## [1] 16+18i
```

- **Matrices -** a type of vector with a *dimension* attribute.
  - The *dimension* attribute is itself an integer vector of length 2 (numRows, numCols)
  - Constructed *column-wise*, so entries can be thought of starting in the "upper left" corner, then running down the columns

  - Matrices can also be created by adding a *dimension* attribute to an existing vector

```r
m <- matrix(nrow = 2, ncol = 3)
m
```

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

```r
dim(m)##reports num of rows then cols
```

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

```r
attributes(m) ## dim is an attribute of the vector
```

```
## $dim
## [1] 2 3
```

```r
m <- matrix(1:6, 2, 3) ## Demonstrating column-wise filling of matrix
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```r
m <- 1:10 ## m is now just a vector
m
```

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

```r
dim(m) <- c(2,5) ## adding the dimension attribute
m
```

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

- Creating a matrix with **cbind** and **rbind**
  - cbind fills the columns with the elements of the vectors that are passed as the respective parameters
  - likewise, rbind fills the rows with the elements of the respective parameters

```r
x <- 1:3
y <- 10:12
cbind(x,y)
```

```
##      x  y
## [1,] 1 10
## [2,] 2 11
## [3,] 3 12
```

```r
rbind(x,y)
```

```
##   [,1] [,2] [,3]
## x    1    2    3
## y   10   11   12
```

Other data types

- Factors
  - Used to represent categorical data

  - can be unordered or ordered
  - Kinda like enumerated data, where it's an integer at heart, and each integer has a *label*
  - Using factors with labels is *better* than using integers because factors are self-describing
    - consider "Male" and "Female" as opposed to just the values 1 and 2
  - Prints differently than a character value, does not include quotations and displays *Levels*

```r
x <- factor(c("yes", "yes", "no", "yes", "no"))
x
```

```
## [1] yes yes no  yes no
## Levels: no yes
```

```r
table(x) ## displays a frequency table of the factors
```

```
## x
##  no yes
##   2   3
```

```r
unclass(x) ## strips out the class and displays the underlying integer vector
```

```
## [1] 2 2 1 2 1
## attr(,"levels")
## [1] "no"  "yes"
```

- The order of the levels can be set with the `levels` argument to `factor()`
  - This can be important in linear modelling because the first level is sued as the baseline level.
  - default levels are based alphabetically

```r
x <- factor(
        c("yes", "yes", "no", "yes", "no"),
        levels = c("yes", "no")
        )
x
```

```
## [1] yes yes no  yes no
## Levels: yes no
```

- Missing Values (`NA` or `NaN`)
  - `NaN` is for undefined mathematical operations
  - `is.na()` and `is.nan()` are logical tests for the respective missing values
  - `NA` values have a class also, so there are integer `NA`, character `NA`, etc.
  - a `NaN` is also a `NA`, however the converse is not true

```r
x <- c(1, 2, NA, 10, 3)
is.na(x)
```

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

```r
is.nan(x)
```

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

```r
x <- c(1, 2, NaN, NA, 4)
is.na(x)
```

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

```r
is.nan(x)
```

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

- Data Frames
  - Used to store tabular data
  - Special type of list where every element has to have the same length
  - Each element is like a column and the length of each element is the number of rows
  - like lists, Data Frames can store different classes in each column
  - Attribute: `row.names`

        * Useful for annotating data
        * However, often the row names are not interesting and we use "1, 2, 3..."
    - Usually created by calling `read.table()` or `read.csv()`
    - Can be converted to a matrix with `data.matrix()`
        * Forces each object to be coerced

```
x <- data.frame(foo = 1:4, bar = c(T, T, F, F))## cols are named here
x
```

```
##   foo   bar
## 1   1  TRUE
## 2   2  TRUE
## 3   3 FALSE
## 4   4 FALSE
```

```
nrow(x)
```

```
## [1] 4
```

```
ncol(x)
```

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

```
row.names(x)
```

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

- Names Attribute, useful for writing readable code and self-describing objects
    - Any R object can have names

```
x <- 1:3
names(x)## by default there are no names
```

```
## NULL
```

```
names(x) <- c("foo", "bar", "norf")
x
```

```
##  foo  bar norf
##    1    2    3
```

```
names(x)
```

```
## [1] "foo"  "bar"  "norf"
```

```
##Lists can also have names
x <- list(a=1, b=2, c=3) ## here, names are assigned as list is established
x
```

```
## $a
## [1] 1
##
## $b
## [1] 2
##
## $c
## [1] 3
```

```
## Matrices can also have names, called dimnames
m <- matrix(1:4, nrow = 2, ncol = 2)
m
```

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

```
dimnames(m) <- list(c("a", "b"), c("c", "d")) ##First vector is rownames, second is colnames
m
```

```
##   c d
## a 1 3
## b 2 4
```

Basic Arithmetic operations

Subset R objects using the "[", "[[", and "$" operators and logical vectors

The explicit coercion feature of R

Removing missing (NA) values from a vector

Control structures, functions, scoping rules, dates and times

Loop functions, debugging tools

Simulation, code profiling