R—Tutorial

Halil Bisgin, PhD

Installation

- Rstudio—Nice IDE
  - *https://www.rstudio.com/products/rstudio/download2/*
- R
  - *https://cran.r-project.org/src/base/R-3/*

Nuts & Bolts

- Entering Input
  - *x <- 1*
  - *print(x)*
  - *msg <- "hello”*
  - *print(msg)*
- Evaluation
  - *x<-5*
  - *x*

R Objects

- Data Types
  - *character*
  - *numeric (real numbers)*
  - *integer*
  - *complex*
  - *logical (True/False)*
- The most basic type of R object is a vector.
- Empty vectors can be created with the vector() function
- You can always check the data type with: class() function.

Attributes

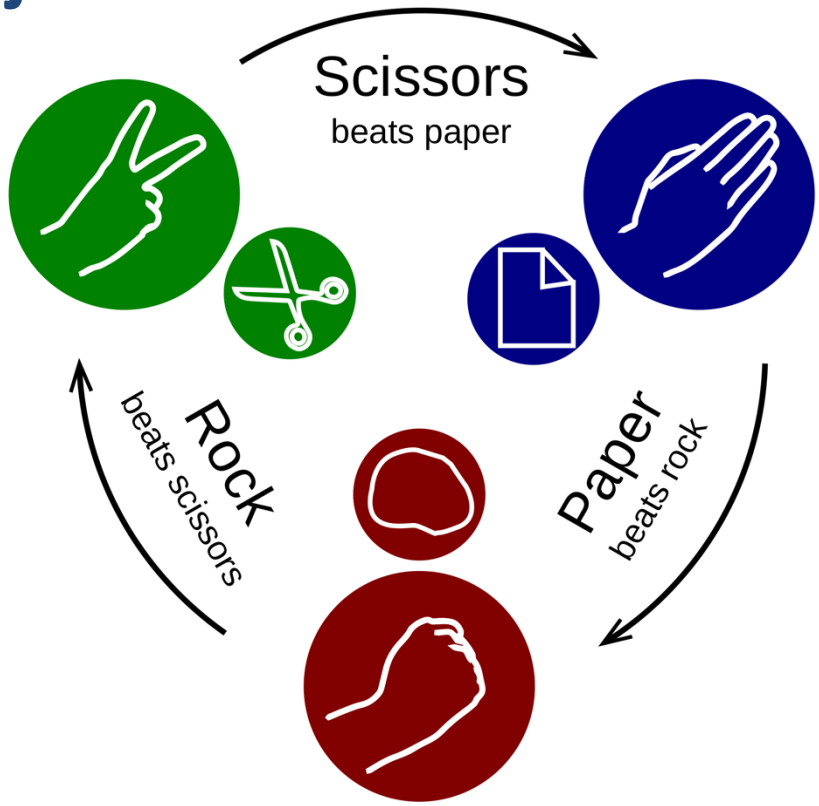
- R objects can have attributes, which are like metadata for the object:
  - *names, dimnames*
  - *dimensions (e.g. matrices, arrays)*
  - *class (e.g. integer, numeric)*
  - *length*
  - *other user-defined attributes/metadata*

Creating Vectors

```
> x <- c(0.5, 0.6) ## numeric
> x <- c(TRUE, FALSE) ## logical
> x <- c(T, F) ## logical
> x <- c("a", "b", "c") ## character
> x <- 9:29 ## integer
> x <- c(1+0i, 2+4i) ## complex
```

Mixing Objects

```
> y <- c(1.7, "a") ## character
> y <- c(TRUE, 2) ## numeric
> y <- c("a", TRUE) ## character
```



Explicit Coercion

```
> 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"
```

Matrices

```
> m <- matrix(nrow = 2, ncol = 3)
> m
[,1] [,2] [,3]
[1,] NA NA NA
[2,] NA NA NA
> dim(m)
[1] 2 3
> attributes(m)
$dim
[1] 2 3
```

Matrices

```
> m <- matrix(1:6, nrow
= 2, ncol = 3)
> m
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6
```

```
> m <- 1:10
> m
[1] 1 2 3 4 5 6 7 8 9 10
> dim(m) <- c(2, 5)
> m
[,1] [,2] [,3] [,4] [,5]
[1,] 1 3 5 7 9
[2,] 2 4 6 8 10
```

Matrices

- Matrices can be created by column-binding or row-binding with the cbind() and rbind() functions.

```
> x <- 1:3
> y <- 10:12
> cbind(x, y)
x y
[1,] 1 10
[2,] 2 11
[3,] 3 12

> rbind(x, y)
[,1] [,2] [,3]
x 1 2 3
y 10 11 12
```

Lists

- Lists are a special type of vector that can contain elements of different classes.

  x <- list(1, "a", TRUE, 1 + 4i)

- We can also create an empty list of a pre-specified length with the vector() function

  x <- vector("list", length = 5)

Factors

- Factors are used to represent categorical data and can be unordered or ordered.
- One can think of a factor as an integer vector where each integer has a label.
- Factors are important in statistical modeling and are treated specially by modelling functions like *lm()*.
- Often factors will be automatically created for you when you read a dataset in using a function like read.table().

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

Missing Values

- Missing values are denoted by NA or NaN for q undefined mathematical operations.
  - *is.na() is used to test objects if they are NA*
  - *is.nan() is used to test for NaN*
  - *NA values have a class also, so there are integer NA, character NA, etc.*
  - *A NaN value is also NA but the converse is not true*

```
>x <- c(1, 2, NA, 10, 3)
> is.na(x)
[1] FALSE FALSE TRUE FALSE FALSE
> is.nan(x)
[1] FALSE FALSE FALSE FALSE FALSE
```

Data Frames

- Data frames are used to store tabular data in R.
- Data frames are represented as a special type of list where every element of the list has to have the same length
- Unlike matrices, data frames can store different classes of objects in each column
- Data frames have a special attribute called `row.names`
- Data frames can be converted to a matrix by calling `data.matrix()`.(`as.matrix()` also works)

Data Frames

```
> var1 <- 1:5
> var2 <- (1:5)/10
```

- var3 <- c("R", "and", "Data Mining", "Examples", "Case Studies")
- df1 <- data.frame(col1=var1, col2=var2, col3=var3)
- names(df1) <- c("VarInt", "VarReal", "VarChar")

Data Frame

```
> m <- matrix(1:4, nrow = 2, ncol = 2)
> dimnames(m) <- list(c("a", "b"), c("c", "d"))
> m
c d
a 1 3
b 2 4

> colnames(m) <- c("h", "f")
> rownames(m) <- c("x", "z")
```

FOR LOOPS

- Not much different.
- Need to indicate the index and its range, though

```
for(j in 3:13) {
  fib[j] <- fib[j-1] + fib[j-2]
}
fib
```

Other flow control

- R has a *while* function as well as a for function. This allows iterations of flexible length.

- while(expression)
  {
  code block
  }

- should be read as

  – *while the expression is true, iteratively execute the code block and re-evaluate the expression based on the result.*

Conditional execution in R

- The syntax of the if function in R is
- if(expression) { code block}
- which is executed as “if the expression is TRUE, execute the code block, otherwise skip it”.
- This is handy when the code block can only be executed when some condition holds.
- There is also an if else combination.
- if(expression) { code block}
- else {other code block}

Functions

- R has a rich collection of functions for performing calculations necessary for doing statistics.
- Much of learning R comes down to acquiring a large enough vocabulary of functions to solve your problems.
- The actual structure of the programs may be quite simple, but you still need to learn the names of the functions that do the required work.
- However, even for everyday users need to write their own functions.

Functions

- Functions are written in a library file with a .R extension and the files are loaded into the R Markdown document doing an analysis with the source command.
- Putting the code into its own function saves retyping the commands and ensures reproducibility by forcing exactly the same code to be run when it is used.

Function definition

```
myfun <- function (a, b, c)
{
  do something ….
  do another thing …
  output #this line will be without <- or =
}
```

Function example

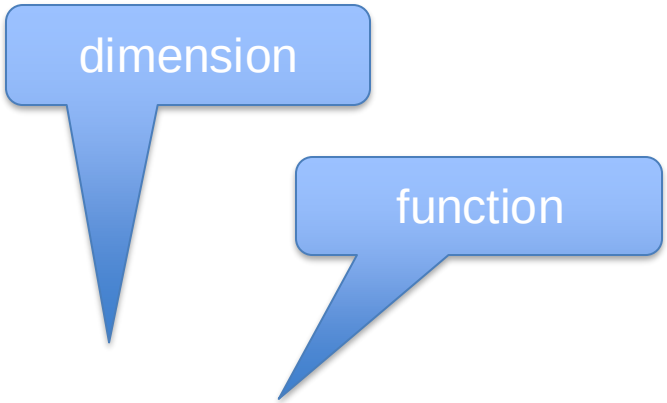
```
mymean <-function (x,y)
{
  output = (x+y)/2
  output
}
```

Default arguments

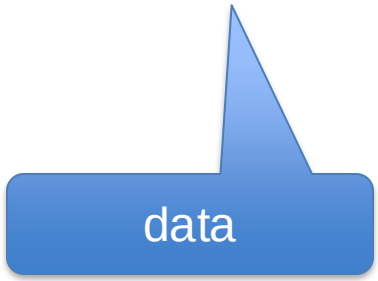
```
mymean <-function (x=0,y=0)
{
  output = (x+y)/2
  output
}
```

Using built-in functions for rows/columns

```
set.seed(123)
v <- sample(x = -10:10, size = 12)
M1 <- matrix(v, nrow=4)
M1
```



- apply
  - *Example: sum_stats <- apply(expr_mat, 1, summary)*



Getting Data In and Out

- Read data from and write data to
  - *R native formats (incl. Rdata and RDS)*
  - *CSV files*
  - *EXCEL files*
  - *ODBC databases*
  - *SAS databases*

Getting Data In and Out

```
a <- 1:10
save(a, file = "./data/dumData.Rdata")
rm(a)
a

## Error in eval(expr, envir, enclos):  object 'a' not found

load("./data/dumData.Rdata")
a

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

Getting Data In and Out

```
library(xlsx)
xlsx.file <- "./data/dummmyData.xlsx"
write.xlsx(df2, xlsx.file, sheetName = "sheet1", row.names = F)
df3 <- read.xlsx(xlsx.file, sheetName = "sheet1")
df3
```

```
##   VarInt VarReal      VarChar
## 1      1     0.1            R
## 2      2     0.2          and
## 3      3     0.3  Data Mining
## 4      4     0.4     Examples
## 5      5     0.5 Case Studies
```

Getting Data In and Out

```
read.delim(file, header = TRUE, sep = "\t")

read.csv(file, header = TRUE, sep = ",")

read.table(file, header = FALSE, sep = "")
```

- The `read.table()` function is one of the most commonly used functions for reading data
- file: filename in quotation marks (may need to include path)
- header: if your file has headers (column names)
- sep: separator; each function has its default separator
- There are further options in these functions if you need
- Built in datasets: mtcars, iris

```
data= read.delim("~/Documents/TEACHING/Data_Science/R/data/Su_raw_matrix.txt")
```

Getting Data In and Out

- *`file`, the name of a file, or a connection*
- *`header`, logical indicating if the file has a header line*
- *`sep`, a string indicating how the columns are separated*
- *`colClasses`, a character vector indicating the class of each column in the dataset*
- *`nrows`, the number of rows in the dataset. By default `read.table()` reads an entire file.*

- Built in datasets: mtcars, iris
- write.table

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE)
```

Working Directory

- > getwd() # get current working directory
  - *You can select a different working directory with the function setwd(), and thus avoid entering the full path of the data files.*

- > setwd("<new path>") # set working directory
  - *Note that the forward slash should be used as the path separator even on Windows platform.*
  - *> setwd("C:/MyDoc")*

Install and import a library

- install.packages("ggplot2")
- library(ggplot2)

Exploring Data

- dim(iris)
- str(iris)
- names(iris)
- attributes(iris)
- head(iris, 3)
- tail(iris, 3)
- iris[1:10, "Sepal.Length"]
- summary(iris)
- library(Hmisc)
  - *describe(iris[, c(1, 5)])*
- range(iris$Sepal.Length)
- quantile(iris$Sepal.Length, c(0.1, 0.3, 0.65))