Introduction to **R**

What is **R**?

As stated in the manual for beginners found at the **C**omprehensive **R** **A**rchive **N**etwork, **R** is

- An effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matices
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either directly at the computer or on hard-copy,
- a well developed simple and effective programming language (called `S`) which includes conditionals, loops, user defined recursive functions and input and output facilities

What is **R**?

- It is **free** and available for all major OS.
- `R` is an **`interpreted`** language.
- `R` itself is a rather 'modern' **GNU** implementation of `S` language, which in turn was developed by *Bell Laboratories*. The first stable beta version was released around 2000.
- Part of `R` functions are written in `R` itself and make it easy to understand, part are written in `C`, `FORTRAN` and other languages for the sake of the performance (mostly computational parts). There is a toolchain for building `R` ↔ `C` interoperable libraries.
- `R` can be extended through *packages* that can be installed from different centralized repositories (`CRAN`, `MRAN` and mirrors), directly from directories on local machine, from *`git` repos and from many other sources.

Basic concepts

- There are several base types in `R`, `integer`, `double`, `character`, `complex`, `factor` and `logical`. It is safe to say that everything in `R` is a first-class object
- The simplest collection is a `vector` - a group of objects of the same type. Any standalone primitive object (like number 2.0) is itself a `vector` of size 1.
- Matricies are arrays of higher order (rectangular), lists are general collections of any objects of any types, data frames are data-base like tables with columns of equal length but different type. Another type, part of the `tidyverse` package, `tibble` extends functions of data frames.
- Higher order types can be defined using class systems (I am aware of several, `S3`, `S4`, `Reference classes` and `S6`)

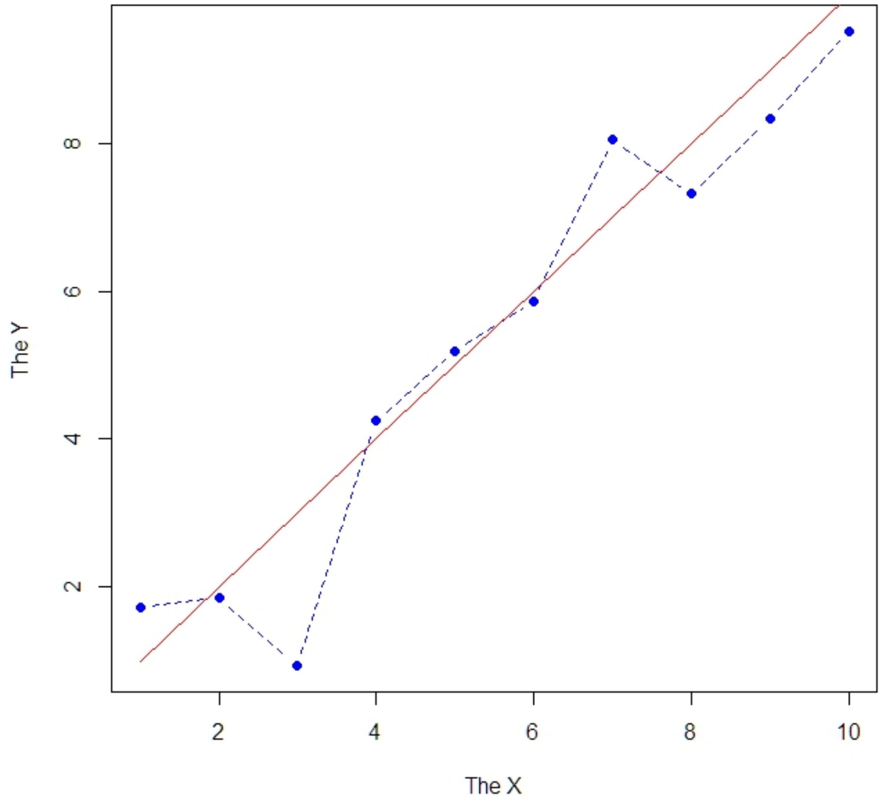
Examples: Run interactively

```
> x <- 5 # Assigns 5 (double) to x
> x      # Prints x, equivalent to print(x)
# [1] 5
> x[1]   # Accesses 1st element of a vector
# [1] 5
> y <- list(x, "Hello world") # A list
> y
# [[1]]
# [1] 5
#
# [[2]]
# [1] "Hello world"
> a <- 1:7 # Integer range from 1 to 7
> a ^ 2    # Vectorized power operation
# [1] 1 4 9 16 25 36 49
> mean(a ^ 2) # Calls built-in stat. mean
# [1] 38.5
> sd (a ^ 2)  # Standard deviation
# [1] 34.17358
```

```
> sprintf("Compare %3.2f to %3.2e", 0.0123, 0.0123)
# [1] "Compare 0.01 to 1.23e-02"
> quantile(rnorm(n = 100, mean = 0, sd = 3),
+ probs = c(0.16, 0.84))
#       16%       84%
# -3.191771  2.949907
> f <- function(x) {
+ mn <- mean(x)
+ s <- sd(x)
+ rng <- range(x)
+ return(c(mn, s, rng))
+}
> f(c(10, 20, 30, 123, 0.06, -100))
# [1]   13.84333   71.24490 -100.00000  123.00000
> z <- 0
> for (i in 1:100) z <- z + i; print(z)
# [1] 5050
> sum(1:100)
# [1] 5050
```

Examples: Get help & make a plot

```
> f # Type the name of previously defined func
# function(x) {
# mn <- mean(x)
# s <- sd(x)
# rng <- range(x)
# return(c(mn, s, rng))
# }
> ?mean  # Opens local web page with manual
> data <- data.frame(x = 1:10,
+ y = 1:10 + rnorm(10))
> plot(data$x, data$y, type = "b",
+ pch = 19, lty = 2, col = "blue",
+ xlab = "The X", ylab = "The Y")
> lines(1:10, 1:10, lty = 1, col = "#FF0000")
> data[1:2, ]
#   x        y
# 1 1 1.721106
# 2 2 1.848209
> quit("no")
```



Examples: Install JAGS and get some packages

- JAGS can be found here
- We need rjags to run JAGS from R, tidyverse (optional) to use fancy data manipulation facilities.

```
> install.packages(c("rjags", "tidyverse")) # Installs two packages
> library(rjags)      # Loads rjags, can be used without quotes
> library(tidyverse) # All useful data processing tools
```

- Now check out code examples (demo)

More useful examples

Assignment operators:

```
<- and -> are default operators
= is used with named func. arguments
<<- and ->> are global assignment ops.
> getwd()  # Returns current working directory
# [1] "\path\to\current\dir"
> setwd("\path\to\new\dir") # Sets work dir
```

Loop alternatives

```
apply, lapply, sapply, vapply
> sapply(1:5, function(x) (x + 1) ^ 2)
# [1]  4  9 16 25 36
```

```
> library(foreach)
> foreach(i = 1:2) %do% { i ^ 2}
# [[1]]
# [1] 1
#
# [[2]]
# [1] 4
> library(parallel)
> library(doSNOW)
> cl <- makeCluster(2, "SOCK")
> registerDoSNOW(cl)
> foreach(...) %dopar% {...}
> stopCluster(cl)
```

```
> file.path("dir1", "dir2")
# "dir1/dir2"
```

More useful examples

```
> paste("d", c("x", "y", "z"), sep = "_")
# [1] "d_x" "d_y" "d_z"
> substring("Hello World!", 7, 11)
# [1] "World"
> grepl("[W|w]o.*!$",
+ c("World!", "world!", "world"))
# [1]  TRUE  TRUE FALSE
> c(5 / 2, 5 %% 2, 5 %/% 2)
# [1] 2.5 1.0 2.0
> c(1, 5, 11) %in% (1:10)
# [1]  TRUE  TRUE FALSE
> setNames(c(1, 2, 3), c("a", "b", "c"))
# a b c
# 1 2 3
> order(c(20, 50, 10, 90, 30))
# [1] 3 1 5 2 4
```

```
> sink("filename.dat") # Output goes to file
> print("Hello World!")
# Nothing in terminal, line appears in file
> sink() # Closes last sink
> pdf("file.pdf", width = 7, height = 5)
> plot(1:5, 5:1) # Plot goes into pdf file
> dev.off() # Closes current device !important!
> tryCatch(stop("Sample error"),
+ error = function(e) print("Err handled"),
+ finally = print("Cleanup"))
# [1] "Cleanup"
# [1] "Err handled"
```

Thank you!