Getting staRted with **R** PRogRamming

Raphael Karanja and Yong Yoon

Strathmore University

July 18, 2014

Outline

- **R** Ecosystem
  Data structures in **R**
- Importing and Exporting Data
  Data import and manipulation
  Exporting data
- Basics of **R** programming
  Simulations
  Loops
  IF ... THEN Conditions
  Writing functions
- Exploratory Data Analysis
  Graphics
  Basic Statistics
- Going Further

Why **R**?

- **R** is a very popular statistical language surpassing SAS, SPSS, Stata, ...
  - More than 2 million users
  - More than 3500 packages (with about 80 added each week)
- Why is **R** so popular?
  - Free and open source. Get **R** from www.r-project.org
  - Packages for nearly everything (statistics, finance, geology, graphics, ...)
  - Easy to add/publish packages
  - Multiple R interfaces available

R 101

```
> 3+3
[1] 6
> x<-3; y = 3
> x<-y<-3
> x^2 # square
[1] 9
> x<-c(1,2,3,4) # or x=c(1:4)
> x
[1] 1 2 3 4
> x+10 # R is vectorized
[1] 11 12 13 14
> y<-c(1,2)
> x+y # recycling rule
[1] 2 4 4 6
> x[4] # show 4th element
[1] 4
```

```
> x>2
[1] FALSE FALSE  TRUE  TRUE
> x[x>2]
[1] 3 4
> x[1:3]
[1] 1 2 3
> is.na(x)
[1] FALSE FALSE FALSE FALSE
> cumsum(x)
[1]  1  3  6 10
> seq(1,5)
[1] 1 2 3 4 5
> rep(1,5)
[1] 1 1 1 1 1
```

Basic Mathematical and Statistical Functions

R has a wide range of useful mathematical and statistical functions available through packages.

```
> x <- 1:10
> mean(x)
[1] 5.5
> median(x)
[1] 5.5
> sd(x)
[1] 3.02765
> sum(x)
[1] 55
> sqrt(x)
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490
 [7] 2.645751 2.828427 3.000000 3.162278
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.25    5.50    5.50    7.75   10.00
```

R online and local help

A good place to start learning **R** is the online documentation.
Launch **R** and type

```
> help.start()
```

The inline help is accessed by

```
> ?mean
```

A nice (fun) book:
http://www.burns-stat.com/documents/books/the-r-inferno/
There are many introductory (many free) sites, for example:
https://www.codecademy.com/learn/learn-r

R Packages

You can find many packages through CRAN.
Installing packages

```
> install.packages("ggplot2")
```

To update all your packages

```
> update.packages(checkBuilt = TRUE, ask = FALSE)
```

To know the libraries already installed in your machine

```
> search()
> library()
```

Data Structure in **R**

There are four main types of data structures in **R**

- vectors
- matrix
- data.frame
- lists

Vectors

Vectors are the basic unit for data storage in **R**.
Vectors can store one or more values of the same type (numeric or character strings).

```
> # Vector of strings
> x <- c("Germany", "Argentina", "Netherlands")
> # Replacing the second element of the vector
> x[2] <- NA
> x
[1] "Germany"     NA              "Netherlands"
> x[!is.na(x)]
[1] "Germany"     "Netherlands"
```

Vector Classes

```
> # character vectors
> x <- c("Germany", "Argentina", "Netherlands"); class(x)
[1] "character"
> x <- c(1.1, 2.5); class(x)
[1] "numeric"
> x <- 1:4; class(x)
[1] "integer"
> x <- x > 2; class(x)
[1] "logical"
```

Matrices

There are many ways to define a matrix.

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

```
> r1=c(1,2)
> r2=c(3,4)
> A = rbind(r1, r2)
> A
   [,1] [,2]
r1    1    2
r2    3    4
> t(A) #transpose
     r1 r2
[1,]  1  3
[2,]  2  4
> det(A) #determinant
[1] -2
> solve(A) #inverse
       r1   r2
[1,] -2.0  1.0
[2,]  1.5 -0.5
```

Data Frames

Think of data frams as specialized type of lists that can do more than usual matrices.

```
>  df <- data.frame(
    patientID = c("001", "002", "003"),
    treatment = c("drug", "placebo", "drug"),
    age = c(20, 30, 24)
  )
>  df
  patientID treatment age
1       001      drug  20
2       002   placebo  30
3       003      drug  24
>  subset(df, treatment=="drug")
  patientID treatment age
1       001      drug  20
3       003      drug  24
```

Lists

Lists are powerful and versatile data structures. You can store any data structure in lists, even lists.

```
>  my.list <- list(
    fruits = c("oranges", "bananas", "apples"),
    mat = matrix(1:10, nrow=2)
  )
>  my.list
$fruits
[1] "oranges" "bananas" "apples"

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

Importing Data

It is easy to import data into **R**. Simply invoke import and export **R** functions, which exist for many formats including

- text or ASCII files
- proprietary formats (e.g. SAS, Stata, SPSS, Excel, etc)
- databases (Oracle, MySQL, couchDB, SQLite, etc)
- **R** data and S
- XML, HTML tables
- and many more

Reading ASCII/Text files

A quick way to access a csv file is

```
> read.csv(file.choose(), header=TRUE)
```

The most common **R** command to import data from text/csv datafiles is `read.table` and `read.csv`, e.g.

```
> URL <- "http://www.strathmore.edu/~yy/scorer2014.csv"
> scorer <- read.table(URL, sep=',', header=TRUE)
> scorer$name
[1]  Rodriguez  Muller     Neymer     Messi
Levels:  Messi  Muller  Neymer  Rodriguez
> class(scorer)
[1] "data.frame"
```

To check your current directory type `getwd()`, which you can change, for example, by `setwd("C:/Users/YY/Documents/R")`

Reading Foreign Files

The package `foreign` enables you to import/export various data file types:

```
> help(package="foreign")
> library(foreign)
> # Read from SPSS datafile
> mydata = read.spss("myfile", to.data.frame=TRUE)
> # Read from Minitab datafile
> mydata = read.mtp("mydata.mtp")
```

The `gdata` and `read.xls` function (or more recently the `xlsx`) packages for Excel files

```
> library(gdata)                    # load gdata package
> mydata = read.xls("mydata.xls")   # read from first sheet
```

Exporting Data

To save as **R** data

```
> save(scorer, file="scorer2014.Rda")
> load("scorer2014.Rda") # to reload your saved data
```

If you want to export your data to a csv file you may use `write.csv` or to a text file use `write.table`

```
> write.csv(mydata, "scorer2014.csv")
> write.table(mydata, file="scorer2014.csv", sep = ',')
```

To export data into other formats (Stata, SAS, etc.) use `foreign` or `SASxport` package

```
> library(foreign)
> ?write.dta
> library(SASxport)
> ?write.xport
```

Useful Things to Know

```
# ls() to list objects in the working environment
# rm() to remove objects
# data entry with data editor
 mydata <- data.frame(age=numeric(0), gender=character(0), weight=numer
 mydata <- edit(mydata)
# list the variables in mydata
 names(mydata)
# list the structure of mydata
 str(mydata)
# class of an object (numeric, matrix, data frame, etc)
 class(age); class(gender)
# print mydata
 mydata
# print first 2 rows of mydata
 head(mydata, n=2)
# print last 2 rows of mydata
 tail(mydata, n=2)
# Ctrl-L to clear console screen
# Esc to interrupt currently executing command
```

Programming Basics

- Simulations
- Looping
- IF ... ELSE conditions
- Writing Functions

Simulations

R is really nice for doing simulations and monte carlo experiments:

```
> set.seed(10101)
> # drawing from normal distribution
> a <- rnorm(1000, mean=0, sd=1)
> hist(a)
```

You can also make random draws from other distributions:

```
> # drawing from poisson
> rpois(10,2)
```

You can also sample randomly from a specified set of objects:

```
> sample(1:10, 4)
[1] 7 6 1 5
> sample(1:10, 4, replace = T)
[1]  8  8  8 10
> sample(letters, 4)
```

Loops

Loops are useful (but not always an efficient way) to do repetitive tasks.

```
for (i in 1:5){
 print("Hello")
}

[1] "Hello"
[1] "Hello"
[1] "Hello"
[1] "Hello"
[1] "Hello"
```

```
> # Law of Large Numbers

> store<-matrix(NA,1000,1)
> for (i in 1:1000){
 a<-rnorm(i)
 store[i]<-mean(a)
 }

> plot(store, type="o")
```

Apply family of functions

R has a series of functions that can simulate small loops called apply.

```
> mat<-matrix(1:10, nrow=2, byrow=T)
> mat
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10

> # sum the columns
> apply(mat,2,sum)
[1]  7  9 11 13 15

> #sum the rows
> apply(mat,1,sum)
[1] 15 40
> rowSums(mat) # gives same results
```

IF ... ELSE Condition

```
x <- 1

if(x==1){
   print("x is equal 1")
 } else{
 print("x is NOT equal to 1")
 }

[1] "x is equal 1"
```

Shortcut

```
ifelse(x==1, "x is equal 1", "x is NOT equal to 1")

[1] "x is equal 1"
```

Exercise

Let's use the loop and IF ... ELSE conditional statement to produce the first 50 Fibanocci numbers.

```
fib <- rep(0,50)

for(i in 1:50){
if (i <= 2){
     fib[i]<-1
      }
      else{
       fib[i]<-fib[i-1]+fib[i-2]
      }
}

fib
```

Functions

We can define functions in **R**, which uses the following basic syntax:

```
> function.name<-function(INPUTS){ARGUMENTS}
```

Let's take a simple example:

```
mr.euclide<-function(a,b){
 dist<-sqrt(sum((b-a)^2))
 return(dist)
}
```

To execute the mr.euclide function:

```
> a<-c(1,1)
> b<-c(2,2)
> mr.euclide(a,b)
[1] 1.414214
```

Exercise

Simulating a dice:

```
dice <- function(n) {
k <- sample(1:6, size= n, replace = T)
return(k)
}

a <- dice(100)
hist(a)
```

Time series from Cauchy distribution:

```
crazy <- function(num) {
 x <- rep(NA , num)
 for (n in 1:num) x[n] <- mean(rcauchy(n))
 plot(x, type = "l", xlab = "sample size", ylab = "sample mean")
}

crazy(100)
```

Challenge

Now incorporate functions to produce say the first 20 Fibanocci numbers

```
fib <- function(n=20){
  if(n<3){
  return(c(1,1))
  } else{
    fib <- rep(0, n)
    for(i in 1:n){
      if(i <= 2){
        fib[i] <- 1
      } else{
        fib[i] <- fib[i-1] + fib[i-2]
      }
    }
  }
  return(fib)
}
fib(20)
```

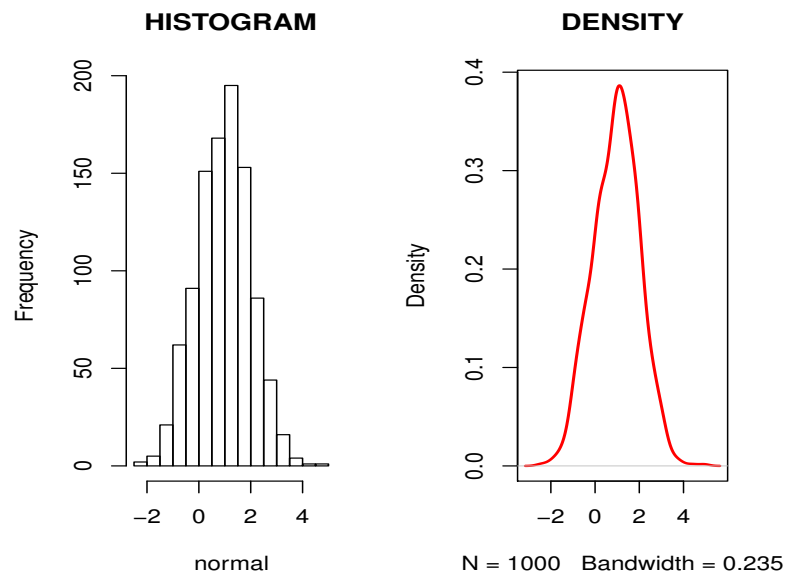
Exploratory Data Analysis

R is a statistical programming language with many useful and powerful graphical libraries.
Let's look at some of the functions available with the `base` package. For example, you can get the values from statistical tables:

```
> pnorm(1.96,0,1) # cumulative distribution
[1] 0.9750021
> pnorm(1.96,0,1, lower.tail = F)
[1] 0.0249979
> dnorm(1.96,0,1) # density
[1] 0.05844094
> dchisq(9,3)
[1] 0.01329555
> 1-pf(4,43,3.6) # p-value
[1] 0.1080504
```

Histograms

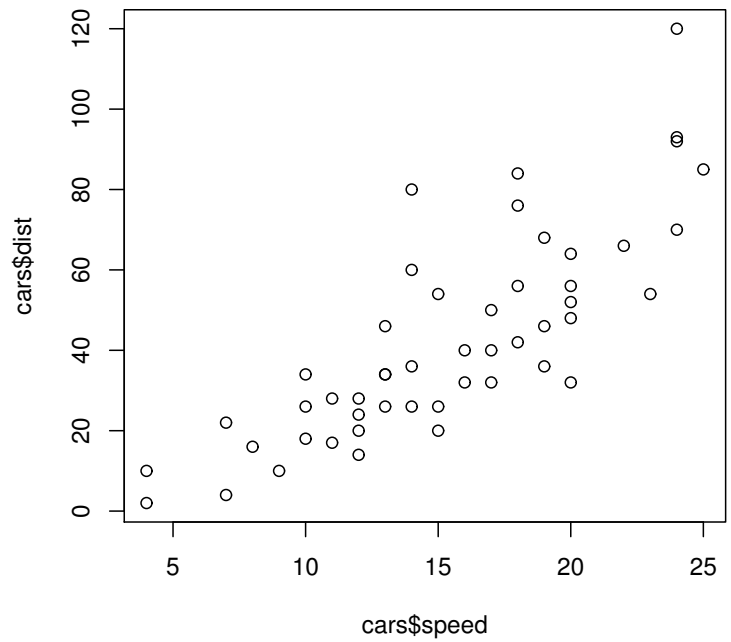
```
> normal <- rnorm(1000, mean=1, sd=1)
> par(mfrow=c(1,2))
> hist(normal, main="HISTOGRAM")
> plot(density(normal), col='red', lwd=2, main="DENSITY")
```



Scatter Plots

R can generate high quality publication ready graphics.

```
> head(cars, 3)
  speed dist
1     4    2
2     4   10
3     7    4
> plot(cars$speed, cars$dist)
```



OLS

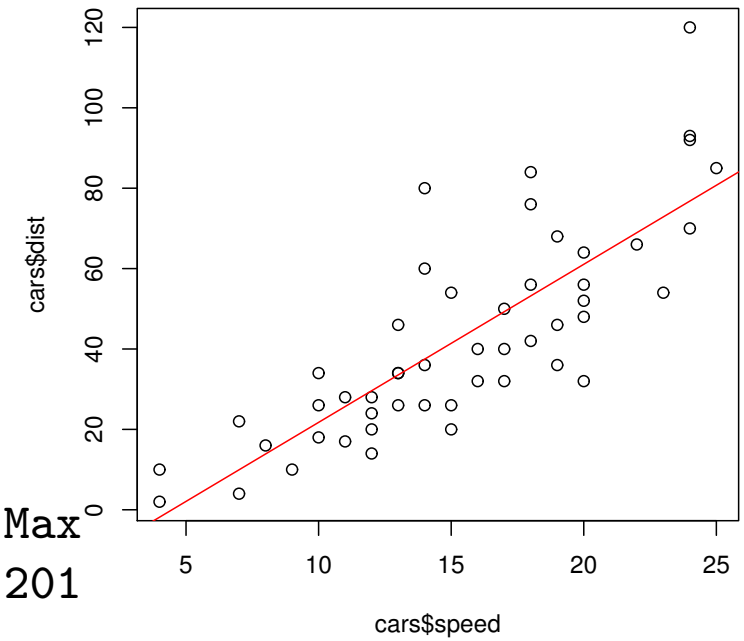
Scatter plots with linear regression line.

```
> plot(cars$speed, cars$dist)
> fit <- lm(cars$dist ~ cars$speed)
> abline(fit, col='red')
> summary(fit)

Call:
lm(formula = cars$dist ~ cars$speed)

Residuals:
    Min      1Q  Median      3Q     Max
-29.069  -9.525  -2.272   9.215  43.201

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -17.5791     6.7584  -2.601   0.0123 *
cars$speed    3.9324     0.4155   9.464 1.49e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



Exercise

Simulate a linear model:

$$y = \alpha + \beta x + \epsilon$$

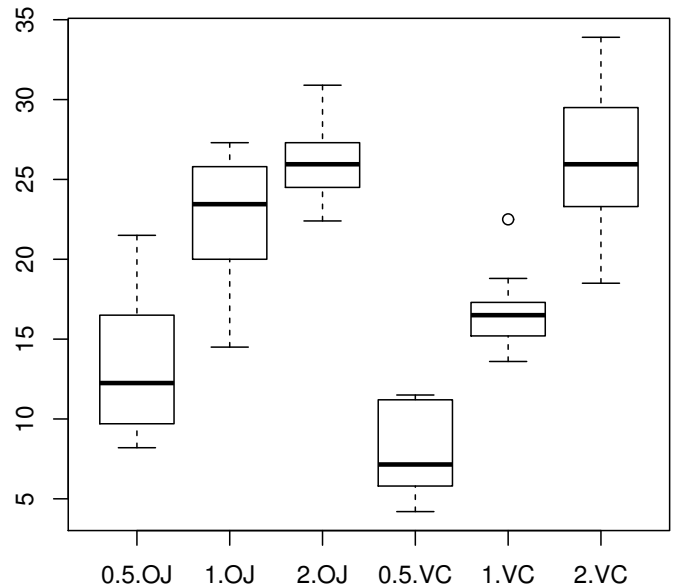
where $\epsilon \sim N(0, 2^2), x \sim X(0, 1^2), \alpha = \frac{1}{2}$ and $\beta = 2$

```
> set.seed(10101)
> x<-rnorm(100)
> e<-rnorm(100,0,2)
> y<-.5+2*x+e
> summary(y)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-9.3680 -1.1200  0.5318  0.6110  2.7240  7.9840
> plot(x,y)
```

Try also fitting a line and find parameters by OLS.

Boxplots

```
> boxplot(len ~ dose * supp,  data=ToothGrowth)
```



Saving Pictures

To save your figures use `pdf()` or `savePlot()`

```
> pdf(file='scatter.pdf', height=5, width=5)
> plot(cars$speed, cars$dist)
> dev.off()
```

There is of course much more to discover in **R**.
Some YouTubes and sites worth visiting are:
http://www.youtube.com/watch?v=tvv4IA8PEzw
https://www.youtube.com/watch?v=jWjqLW-u3hc
⋮
https://www.r-project.org/doc/bib/R-books.html
And there are many, many books as well.
Good luck and remember to have fun with R.

- END -