DATA ANALYTICS AND MACHINE LEARNING WITH R

# ESSENTIALS OF R PROGRAMMING

LUIS GUSTAVO NARDIN
INTERNET TECHNOLOGY
BRANDENBURG UNIVERSITY OF TECHNOLOGY

# R

**https://www.r-project.org**

- R environment is an integrated suite of software facilities for data manipulation, calculation and graphical display

# R

**https://www.r-project.org**

- Integrated suite of software facilities for data manipulation, calculation and graphical display
- Scripting language and environment for statistical computing

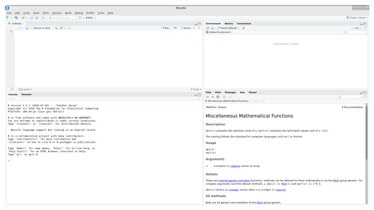
## R ENVIRONMENT

- an effective data handling and storage facility
- a suite of operators for calculations on arrays, in particular matrices
- a large, coherent, integrated collection of intermediate tools for data analysis
- graphical facilities for data analysis and display either on-screen or on hardcopy
- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities

## ADVANTAGES

- *de facto* standard among professional statisticians
- Comparable, and often superior, in power to commercial products
- Multi-platform (Windows, Mac, Linux and Unix)
- Open source software
- General-purpose programming language that can be easily extended with new functions via *packages*
- Incorporate features of object-oriented and functional programming

# RSTUDIO IDE



```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

MathFun {base} R Documentation

Miscellaneous Mathematical Functions

Description

abs(x) computes the absolute value of x, sqrt(x) computes the (principal) square root of x, √{x}.

The naming follows the standard for computer languages such as C or Fortran.

Usage

```
abs(x)
sqrt(x)
```

Arguments

x a numeric or complex vector or array.

Details

These are internal generic primitive functions: methods can be defined for them individually or via the Math group generic. For complex arguments (and the default method), z, abs(z) == Mod(z) and sqrt(z) == z^0.5.

abs(x) returns an integer vector when x is integer or logical.

S4 methods

Both are S4 generic and members of the Math group generic.

RSTUDIO IDE

- **R Script**: Editor to write code or documentation. To run the commands, simply select the line(s) of code and press Ctrl + Enter. Alternatively, you can click on little 'Run' or 'Source' buttons located at the top right corner of R Script panel.
- **R Console**: R instance where the commands are executed. Commands can be directly write in the console.
- **R Environment**: Display the objects available in the R instance. This includes data set, variables, vectors, functions, etc.
- **R Output**: Display graphs created during data analysis. Additionally, provide access to the file structure of the computer 'Files', available packages 'Packages' and R's official documentation 'Help'.

# BASICS

- Interactive Mode

```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Compu
Platform: x86_64-pc-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditio
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
```

- Batch Mode

```
$ R CMD BATCH script.R
```

## BASICS

### HELP FUNCTION

```
> help(seq)
```

```
> ?seq
```

```
> ?"<"
```

```
> ?"for"
```

```
example(sqrt)
```

## PACKAGES

R has a rich set of base functions; however, its power lies in its extensibility through packages available at
**CRAN R Packages**

Install packages

```
install.packages([package name])
```

Example

```
install.packages("data.table")
```

```
install.packages(c("data.table", "ggplot2"))
```

# PACKAGES

Update packages

```
update.packages()
```

List installed packages

```
installed.packages()
```

Load installed package

```
library(data.table)
```

List loaded packages

```
search()
```

# R LANGUAGE

Everything in R is an object

**Types of Objects**

| **typeof()** | **mode()** |
|---|---|
| `logical` | `logical` |
| `integer` | `numeric` |
| `double` | `numeric` |
| `complex` | `complex` |
| `character` | `character` |
| `raw` | `raw` |

## R LANGUAGE

- Objects may have attributes
- List object's attributes

```
attributes([object])
```

- Get or set specific attributes of an object

```
attr(x, which)
attr(x, which) <- value
```

R LANGUAGE

Assignment

```
> x <- 10
> y <- 10 + 4
> c(1, 2, 3) -> z
```

# R LANGUAGE - OPERATORS

| | |
|---|---|
| - | Minus, can be unary or binary |
| + | Plus, can be unary or binary |
| ! | Unary not |
| ~ | Tilde, used for model formulae, can be either unary or binary |
| : | Sequence, binary |
| * | Multiplication, binary |
| / | Division, binary |
| ^ | Exponentiation, binary |

| | |
|---|---|
| %x% | Special binary operators, x can be replaced by any valid name |
| %% | Modulus, binary |
| %/% | Integer divide, binary |
| %*% | Matrix product, binary |
| %o% | Outer product, binary |
| %x% | Kronecker product, binary |
| %in% | contains, binary |
| is.element(x,y) | contains, function |

# R LANGUAGE - OPERATORS

| | |
|---|---|
| & | And, binary, vectorized |
| && | And, binary, not vectorized |
| \| | Or, binary, vectorized |
| \|\| | Or, binary, not vectorized |
| <- | Left assignment, binary |
| -> | Right assignment, binary |
| $ | List subset, binary |

| | |
|---|---|
| == | Equal to, binary |
| != | Not equal to, binary |
| < | Less than, binary |
| > | Greater than, binary |
| >= | Greater than or equal to, binary |
| <= | Less than or equal to, binary |
| any() | Expression TRUE for at least one element in vector |
| all() | Expression TRUE for all elements in vector |

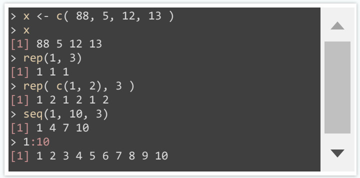
## DATA STRUCTURES

- Vector
- Matrix
- Array
- List
- Data Frame

## VECTORS

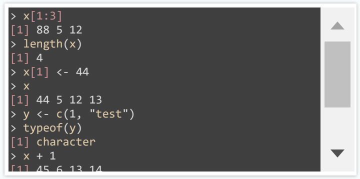
- A series of elements
- Created with
    - `c()` to concatenate elements or sub-vectors
    - `rep()` to repeat elements or patterns
    - `seq` or `m:n` to generate sequences
    - `vector(length, mode)` to create an vector object
- Most mathematical functions and operators can be applied to vectors **without loops!**
- Vector can contain a single mode of element

VECTORS



```
> x <- c( 88, 5, 12, 13 )
> x
[1] 88 5 12 13
> rep(1, 3)
[1] 1 1 1
> rep( c(1, 2), 3 )
[1] 1 2 1 2 1 2
> seq(1, 10, 3)
[1] 1 4 7 10
> 1:10
[1] 1 2 3 4 5 6 7 8 9 10
```

## VECTORS



```
> x[1:3]
[1] 88 5 12
> length(x)
[1] 4
> x[1] <- 44
> x
[1] 44 5 12 13
> y <- c(1, "test")
> typeof(y)
[1] character
> x + 1
[1] 45 6 13 14
```

# VECTORS

```
> x <- c( 88, 5, 12, 13 )
> x > 10
[1] TRUE FALSE TRUE TRUE
> x[ x > 10 ]
[1] 88 12 13
> x[ x < 10 ] <- 100
> x
[1] 88 100 12 13
> subset( x, x > 50)
[1] 88 100
> which( x < 50 )
[1] 3 4
```

## NA AND NULL

- `NA` represents missing values
- `NULL` means that the value does not exist

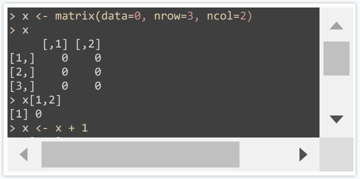
```
> x <- c( 88, NA, 12, 168, 13 )
> x
[1] 88 NA 12 168 13
> mean(x)
[1] NA
> mean(x, na.rm=TRUE)
[1] 70.25
> x <- c( 88, NULL, 12, 168, 13 )
> mean(x)
[1] 70.25
> x
[1] 88 12 168 13
```

## MATRIX

- A rectangular array of elements
- Set of vectors
- Most mathematical functions and operators can be applied to matrices **without loops!**
- Matrix can contain a single type of element

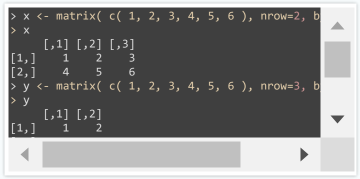
```
matrix(data=NA, nrow=1, ncol=1)
```

## MATRIX



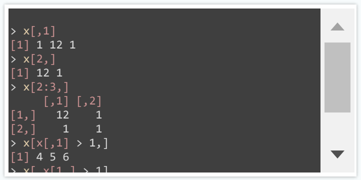
```
> x <- matrix(data=0, nrow=3, ncol=2)
> x
     [,1] [,2]
[1,]    0    0
[2,]    0    0
[3,]    0    0
> x[1,2]
[1] 0
> x <- x + 1
```

# MATRIX



```
> x <- matrix( c( 1, 2, 3, 4, 5, 6 ), nrow=2, b
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> y <- matrix( c( 1, 2, 3, 4, 5, 6 ), nrow=3, b
> y
     [,1] [,2]
[1,]    1    2
```

## MATRIX



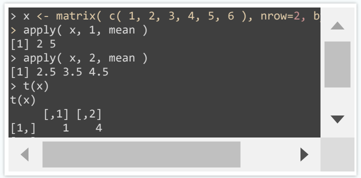
```
> x[,1]
[1] 1 12 1
> x[2,]
[1] 12 1
> x[2:3,]
     [,1] [,2]
[1,]   12    1
[2,]    1    1
> x[x[,1] > 1,]
[1] 4 5 6
> x[ x[1,] > 1]
```

# MATRIX

**Operations**

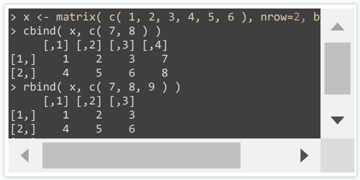
- `t(m)` transpose matrix `m`
- `apply(m, dimcode, f, fargs)` applies a function to the matrix values
- `cbind` appends column
- `rbind` appends row
- `colnames(m)` get/set names to columns
- `rownames(m)` get/set names to rows

# MATRIX



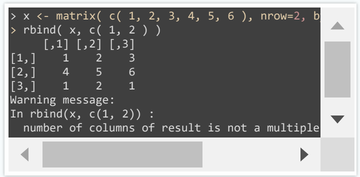
```
> x <- matrix( c( 1, 2, 3, 4, 5, 6 ), nrow=2, b
> apply( x, 1, mean )
[1] 2 5
> apply( x, 2, mean )
[1] 2.5 3.5 4.5
> t(x)
t(x)
     [,1] [,2]
[1,]    1    4
```

## MATRIX



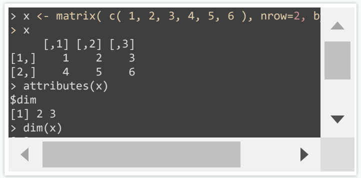
```
> x <- matrix( c( 1, 2, 3, 4, 5, 6 ), nrow=2, b
> cbind( x, c( 7, 8 ) )
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    7
[2,]    4    5    6    8
> rbind( x, c( 7, 8, 9 ) )
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

## MATRIX



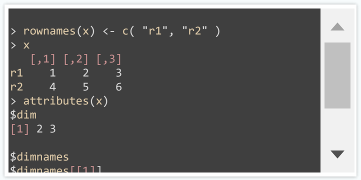
```
> x <- matrix( c( 1, 2, 3, 4, 5, 6 ), nrow=2, b
> rbind( x, c( 1, 2 ) )
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    1    2    1
Warning message:
In rbind(x, c(1, 2)) :
  number of columns of result is not a multiple
```

# MATRIX



```
> x <- matrix( c( 1, 2, 3, 4, 5, 6 ), nrow=2, b
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> attributes(x)
$dim
[1] 2 3
> dim(x)
```

# MATRIX



```
> rownames(x) <- c( "r1", "r2" )
> x
   [,1] [,2] [,3]
r1    1    2    3
r2    4    5    6
> attributes(x)
$dim
[1] 2 3

$dimnames
$dimnames[[1]]
```

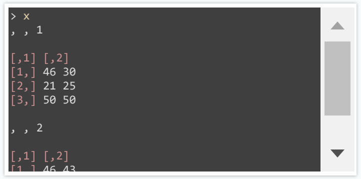
## ARRAY

- Arrays are high-dimensional data structures (more than 2 dimensions
- Created with `array(data=NA, dim=length(data)`

## ARRAY

```
> fmatrix <- matrix( c( 46, 30, 21, 25, 50, 50 ), nr
> smatrix <- matrix( c( 46, 43, 41, 35, 50, 50 ), nr
> x <- array( data=c( fmatrix, smatrix ), dim=c( 3, 
> attributes(x)
$dim
[1] 3 2 2
```

# ARRAY

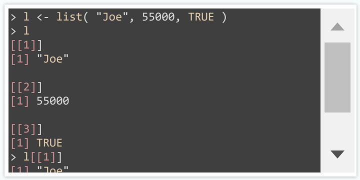


```
> x
, , 1

[,1] [,2]
[1,] 46 30
[2,] 21 25
[3,] 50 50

, , 2

[,1] [,2]
[1,] 46 43
```

## LIST

- Collection of related variables
- Allow value of different modes
- Created with
  `l <- list(x = 1, y = "1")`
- Access to component x
  - `l$x`
  - `l["x"]`
  - `l[1]`
  - `l[-2]`

# LIST



```
> l <- list( "Joe", 55000, TRUE )
> l
[[1]]
[1] "Joe"

[[2]]
[1] 55000

[[3]]
[1] TRUE
> l[[1]]
[1] "Joe"
```

# LIST

```
> names(l) <- c("name", "salary", "employee")
> l
$name
[1] "Joe"

$salary
[1] 55000

$employee
[1] TRUE
```

## DATA FRAME

- Two-dimensional rows-and-columns structure
- Different than matrix and vectors, it allows multiple data modes
- Most of the time, when data is loaded from storage (e.g., file), it will be organized in a data frame

## DATA FRAME

- Load from a text file using `read.table()`
  - Parameters `header`, `sep`, and `na.string` control useful options
  - `read.csv()` and `read.delim()` have useful defaults for comma or tab delimited files
- Create from scratch using `data.frame()`

## DATA FRAME

- `x <- data.frame( height=c( 150, 160 ), weight=c( 65, 72 ) )`
- Retrieving data
  - `x["weight"]`
  - `x[,2]`
  - `x$weight`

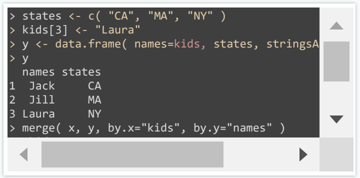
## DATA FRAME

```
> kids <- c( "Jack", "Jill" )
> ages <- c( 12, 10 )
> x <- data.frame( kids, ages, stringsAsFactors
> x
  kids ages
1 Jack   12
2 Jill   10
> str(x)
'data.frame':   2 obs. of  2 variables:
```

## DATA FRAME

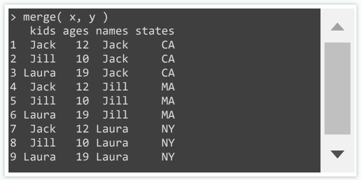
```
> x <- rbind( x, list( "Laura", 19 ) )
> x
   kids ages
1  Jack   12
2  Jill   10
3 Laura   19
> x[x$ages > 10,]
   kids ages
1  Jack   12
3 Laura   19
```

## DATA FRAME



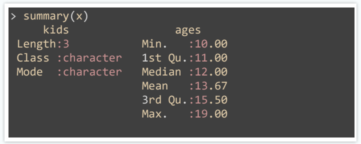
```
> states <- c( "CA", "MA", "NY" )
> kids[3] <- "Laura"
> y <- data.frame( names=kids, states, stringsA
> y
  names states
1  Jack     CA
2  Jill     MA
3 Laura     NY
> merge( x, y, by.x="kids", by.y="names" )
```

## DATA FRAME



```
> merge( x, y )
   kids ages names states
1  Jack   12  Jack     CA
2  Jill   10  Jack     CA
3 Laura   19  Jack     CA
4  Jack   12  Jill     MA
5  Jill   10  Jill     MA
6 Laura   19  Jill     MA
7  Jack   12 Laura     NY
8  Jill   10 Laura     NY
9 Laura   19 Laura     NY
```

## DATA FRAME



```
> summary(x)
     kids               ages      
 Length:3           Min.   :10.00  
 Class :character   1st Qu.:11.00  
 Mode  :character   Median :12.00  
                    Mean   :13.67  
                    3rd Qu.:15.50  
                    Max.   :19.00  
```

# DATA FRAME

Form row and column sums and means for numeric arrays (or data frames).

- `colSums`
- `rowSums`
- `colMeans`
- `rowMeans`

## EXERCISE

Upload the file `bp.txt`

- `HEIGHT` (cm)
- `WEIGHT` (cm)
- `WAIST` (cm)
- `HIP` (cm)
- `BPSYS` (Systolic pressure)
- `BPDIA` (Diastolic pressure)

## EXERCISE

- Check the structure of the data table loaded
- Summarize the data table content
- Average all columns
- Include a new record to the data frame
- Ratio of Waist and Hip
- Filter records with BPSYS greater than 120 and BPDIA greater than 80

## PROGRAMMING STRUCTURES

- `ifelse`
- `if-then-else`
- `for` loops
- `repeat` loops
- `while` loops
- `next`, `break` statements

## GROUP EXPRESSIONS

```
{ expr_1; expr_2; ... }
```

- Valid wherever single expression could be used
- Return the result of last expression evaluated

## `IFELSE` FUNCTION

- Vectorized form of `if-then-else` control statement
- Potentially much faster
- `ifelse( expr_1, expr_2, expr_3 )` evaluates `expr_1` and evaluates `expr_2` in case of `expr_1` is TRUE, otherwise evaluates `expr_3`

```
> x <- c( 3, 2, 2, 3, 1, 2, 3, 3 )
> ifelse( x >= 3, x / 2, x * 2 )
[1] 1.5 4.0 4.0 1.5 2.0 4.0 1.5 1.5
> ifelse( x > 3, x / 2, x * 2 )
[1] 6 4 4 6 2 4 6 6
```

## IF-THEN-ELSE

**if (** `expr_1` **)** `expr_2` **else** `expr_3`

- `expr_1` should return a single logical value
    - Operators && or `||` may be used
- Conditional execution of code

## IF-THEN-ELSE

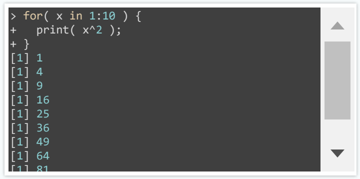
```
>if ( any( x >= 3 ) ) {
+   x / 2;
+ } else {
+   x * 2;
+ }
[1] 1.5 1.0 1.0 1.5 0.5 1.0 1.5 1.5

> if ( all( x >= 3 ) ) {
+   x / 2;
+ } else {
+   x * 2;
+ }
```

## `for` LOOPS

```
for ( name in expr_1 ) expr_2
```

- `name` is the loop variable
- `expr_1` is often a sequence
- `expr_2` is evaluated for each value in `expr_1`

FOR LOOPS



```
> for( x in 1:10 ) {
+   print( x^2 );
+ }
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
[1] 49
[1] 64
[1] 81
```

## REPEAT LOOPS

**`repeat`** `expr_1`

- Continually evaluates `expr_1`
- Loop must be terminated with a `break` statement

## REPEAT LOOPS

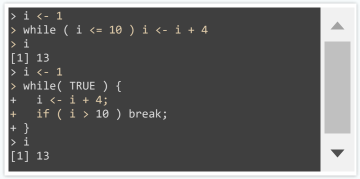
```
> i <- 1
> repeat {
+   i <- i + 4;
+   if ( i > 10 ) break;
+ }
> i
[1] 13
```

## `WHILE` LOOPS

**`while`** `( expr_1 ) expr_2`

- While `expr_1` is FALSE, repeatedly evaluates `expr_2`
- `next` and `break` statements can be used within the loop

# WHILE LOOPS



```
> i <- 1
> while ( i <= 10 ) i <- i + 4
> i
[1] 13
> i <- 1
> while( TRUE ) {
+   i <- i + 4;
+   if ( i > 10 ) break;
+ }
> i
[1] 13
```

## FUNCTION

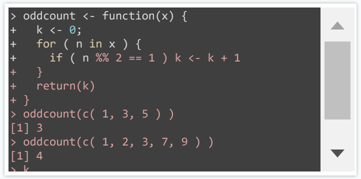
- A function is a group of instructions that takes inputs, uses them to compute other values, and returns a result
- So, as tasks become complex, it is a good idea to organize code into functions that perform defined tasks
- In R, functions are **first-class objects**

## FUNCTION

`name <- function( arg1, arg2, ... ) expression`

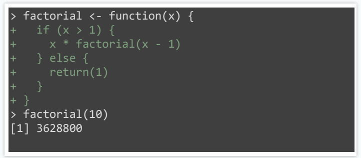
- Arguments can be assigned default values: `arg = expression`
- Return value is the last evaluated expression or can be set explicitly with `return()`
- Variables declared inside a function can be accessed only inside the function
- To change the value of global variables from inside a function use operator `<<-`

# FUNCTION



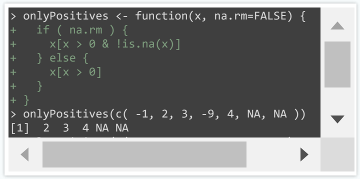
```
> oddcount <- function(x) {
+   k <- 0;
+   for ( n in x ) {
+     if ( n %% 2 == 1 ) k <- k + 1
+   }
+   return(k)
+ }
> oddcount(c( 1, 3, 5 ) )
[1] 3
> oddcount(c( 1, 2, 3, 7, 9 ) )
[1] 4
> k
```

FUNCTION



```
> factorial <- function(x) {
+   if (x > 1) {
+     x * factorial(x - 1)
+   } else {
+     return(1)
+   }
+ }
> factorial(10)
[1] 3628800
```

## FUNCTION



```
> onlyPositives <- function(x, na.rm=FALSE) {
+   if ( na.rm ) {
+     x[x > 0 & !is.na(x)]
+   } else {
+     x[x > 0]
+   }
+ }
> onlyPositives(c( -1, 2, 3, -9, 4, NA, NA ))
[1]  2  3  4 NA NA
```

# FUNCTION

```
> intsum <- function(from=1, to=10){
+   stopifnot(!is.integer(from), !is.integer(to
+   sum <- 0
+   for (i in from:to)
+     sum <- sum + i
+   return(sum)
+ }
> intsum()
[1] 55
```

## FUNCTION

Some notes on functions

- `args` Displays the argument names and corresponding default values of a function or primitive.
- `stopifnot` If any of the expressions TRUE produces an error message indicating what was not true.
- `match.arg` matches the argument against a table of candidate values as specified as default values.
- You can print the content of a function by typing only its name, without the `()`

## OTHER COMMANDS

- `ls() list all environment objects`
- `rm(object_name)` remove environment objects
  `rm(list=ls())` remove all environment objects
- `getwd(path)` display working directory
- `setwd(path)` set working directory
- `save(filename)` save workspace
- `load(filename)` restore workspace