Mastering Data Analysis & Visualization Using R

Osama Mahmoud

Website: ***http://osmahmoud.com***

E-mail: ***o.mahmoud@bristol.ac.uk***

MA902: Research Methods - R Courses

R courses: 19th October & 2nd November 2019

Structure

The R courses include two 'days' structered as follows:

Day 1: 19^{th} Oct. 2019 - Introduction to R.

Day 2: 2^{nd} Nov. 2019 - Data visualisation & dynamic reports.

Introduction to R

Osama Mahmoud

website: http://osmahmoud.com

E-mail: o.mahmoud@bristol.ac.uk

MA902: Research Methods - R Courses, 19^{th} October 2019

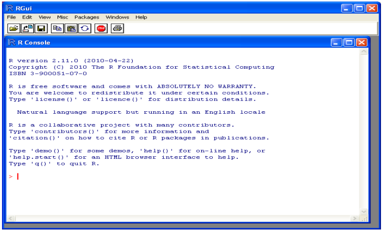
Contents

1. Basic Concepts
2. Data Structures
    - Simple operations & assign values
    - Vectors & Matrices
    - Objects
    - Data types
    - Data structures
    - Importing and exporting data
3. Basic Programming
    - Functions
    - Control structures
    - Conditional execution
    - Loops
4. Getting More of R

What is R?

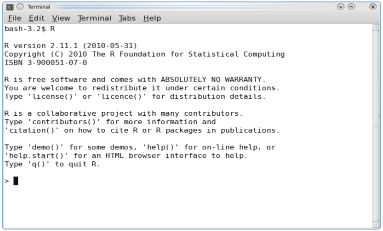
- R is a language (environment) for data analysis.
- Command-line oriantated available for Windows, Linux and Mac OS X.

Windows



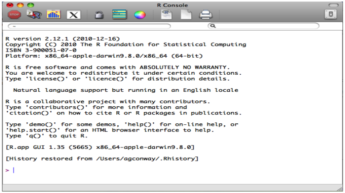
```
R version 2.11.0 (2010-04-22)
Copyright (C) 2010 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

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.

> |
```

Linux



```
bash-3.2$ R

R version 2.11.1 (2010-05-31)
Copyright (C) 2010 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

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.

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.

> █
```

Mac OS X



```
R version 2.12.1 (2010-12-16)
Copyright (C) 2010 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-apple-darwin9.8.0/x86_64 (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.

[R.app GUI 1.35 (5665) x86_64-apple-darwin9.8.0]

[History restored from /Users/agconway/.Rhistory]

> |
```

Why R?

- R is free, and available on every major platform (www.cran.r-project.org).
- Massive set of packages for statistical modelling, visulaisation, AI, ML, and data science.
- Powerful tools for communicating results:
  - RMarkdown - turns your results into HTML, PDF, Word documents, PowerPoint presentations, and more.
  - Shiny - makes beautiful interactive apps without any knowledge of HTML or Javascript.
- Many IDE, e.g. RStudio which tailored to the needs of data science and statistical programming.
- Cutting edge tools - Researchers do often publish an R package with their technical articles.
- Strongly supported open-source communities (e.g. over 50% of software engineers at RStudio work on open source projects)

Why R?

- R is free, and available on every major platform (www.cran.r-project.org).
- Massive set of packages for statistical modelling, visulaisation, AI, ML, and data science.
- Powerful tools for communicating results:
  - RMarkdown - turns your results into HTML, PDF, Word documents, PowerPoint presentations, and more.
  - Shiny - makes beautiful interactive apps without any knowledge of HTML or Javascript.
- Many IDE, e.g. RStudio which tailored to the needs of data science and statistical programming.
- Cutting edge tools - Researchers do often publish an R package with their technical articles.
- Strongly supported open-source communities (e.g. over 50% of software engineers at RStudio work on open source projects)

Why R?

- R is free, and available on every major platform (www.cran.r-project.org).
- Massive set of packages for statistical modelling, visulaisation, AI, ML, and data science.
- Powerful tools for communicating results:
  - RMarkdown - turns your results into HTML, PDF, Word documents, PowerPoint presentations, and more.
  - Shiny - makes beautiful interactive apps without any knowledge of HTML or Javascript.
- Many IDE, e.g. RStudio which tailored to the needs of data science and statistical programming.
- Cutting edge tools - Researchers do often publish an R package with their technical articles.
- Strongly supported open-source communities (e.g. over 50% of software engineers at RStudio work on open source projects)

Why R?

- R is free, and available on every major platform (www.cran.r-project.org).
- Massive set of packages for statistical modelling, visulaisation, AI, ML, and data science.
- Powerful tools for communicating results:
  - RMarkdown - turns your results into HTML, PDF, Word documents, PowerPoint presentations, and more.
  - Shiny - makes beautiful interactive apps without any knowledge of HTML or Javascript.
- Many IDE, e.g. RStudio which tailored to the needs of data science and statistical programming.
- Cutting edge tools - Researchers do often publish an R package with their technical articles.
- Strongly supported open-source communities (e.g. over 50% of software engineers at RStudio work on open source projects)

Why R?

- R is free, and available on every major platform (www.cran.r-project.org).
- Massive set of packages for statistical modelling, visulaisation, AI, ML, and data science.
- Powerful tools for communicating results:
  - RMarkdown - turns your results into HTML, PDF, Word documents, PowerPoint presentations, and more.
  - Shiny - makes beautiful interactive apps without any knowledge of HTML or Javascript.
- Many IDE, e.g. RStudio which tailored to the needs of data science and statistical programming.
- Cutting edge tools - Researchers do often publish an R package with their technical articles.
- Strongly supported open-source communities (e.g. over 50% of software engineers at RStudio work on open source projects)

Why R?

- R is free, and available on every major platform (www.cran.r-project.org).
- Massive set of packages for statistical modelling, visulaisation, AI, ML, and data science.
- Powerful tools for communicating results:
  - RMarkdown - turns your results into HTML, PDF, Word documents, PowerPoint presentations, and more.
  - Shiny - makes beautiful interactive apps without any knowledge of HTML or Javascript.
- Many IDE, e.g. RStudio which tailored to the needs of data science and statistical programming.
- Cutting edge tools - Researchers do often publish an R package with their technical articles.
- Strongly supported open-source communities (e.g. over 50% of software engineers at RStudio work on open source projects)

Why R?

- R is free, and available on every major platform (www.cran.r-project.org).
- Massive set of packages for statistical modelling, visulaisation, AI, ML, and data science.
- Powerful tools for communicating results:
    - RMarkdown - turns your results into HTML, PDF, Word documents, PowerPoint presentations, and more.
    - Shiny - makes beautiful interactive apps without any knowledge of HTML or Javascript.
- Many IDE, e.g. RStudio which tailored to the needs of data science and statistical programming.
- Cutting edge tools - Researchers do often publish an R package with their technical articles.
- Strongly supported open-source communities (e.g. over 50% of software engineers at RStudio work on open source projects)

Editors

Functionality of editor softwares

- Easy tools for writing, managing and organizing R scripts
- offers syntax Auto-completion
- An effective interface to communicate with R

Examples of Editors

- **Windows**

✓ RStudio

✓ RWinEdt

- **Linux**

✓ RStudio

✓ JGR

- **Mac OS X**

✓ RStudio

RStudio is highly recommended

- Multi-platform IDE (Download: www.rstudio.com)
- Provides excellent handy features and professional products.

Concepts of R

- In an interactive session, R works with Read-Eval-Print mode:
    - Input: 5 + 2
    - Evaluation: R evaluates the input expression 5 + 2 = 7
    - Output: R prints the resulted output(s) [1] 7

- Comments are identifyed by #
- Input prompt: >
- Prompt turns to '+' when an expression needs to be completed.

Sources for R help

Embeded Help System

- Search help for a topic `help.search("sum")`
- Search help for a function `help("sum")` or `?sum`
- Search help for usages of a function `example("sum")` or `demo()`

Integrated Help System

- HTML help system via your web browser `help.start()`

Online Sources & Communities

- rweekly: www.rweekly.org
- R seek: http://www.rseek.org
- Twitter: #rstats twitter community

Simple operations

- Arithmetic operations

```
> 5 + 3        > 13 - 7       > 3 * 4        > 20 / 4       > 2 ^ 3
[1] 8          [1] 6          [1] 12         [1] 5          [1] 8
```

- Boolean operations

```
> 3 == 5       > !(2 > 3)     > TRUE | FALSE        > T & F
[1] FALSE      [1] TRUE       [1] TRUE              [1] FALSE
```

- Simple functions

```
> exp(1)          > log(100, base = 10)        > sqrt(16)
[1] 2.71828       [1] 2                        [1] 4
```

Assign values

Example: BMI Calculation

Calculation of the Body Mass Index for a person whose weight in kilograms and height in meters are w and h respectively.

$$BMI = \frac{w}{h^2}$$

```
> w <- 100              # assignment of weight value
> h <- 1.75             # assignment of height value
> BMI <- w/(h^2)        # calculation of BMI
> BMI
[1] 32.65306
```

Vectors

- Combinations of elements from the same data type

```
> x <- c(2,4,6)
> x
[1] 2 4 6
```

```
> days <- c("Monday", "Thursday")
> days
[1] "Monday" "Thursday"
```

- Sequences

```
> x <- 3:5
> x
[1] 3 4 5
```

```
> x <- seq(from=10, to=20, by=2)
> x
[1] 10 12 14 16 18 20
```

- Replications

```
> x <- rep(5, times=3)
> x
[1] 5 5 5
```

```
> x <- rep(c(1,2,3), each=2)
> x
[1] 1 1 2 2 3 3
```

Arithmetic operations using vectors

- Arithmetics are defined in a component-wise fashion

```
> x <- c(5, 10, 15); y <- 1:3
> A <- sqrt(7)*((x^2)+y)
> A
[1] 68.78953 269.86663 603.23130
```

```
> round(A, digits = 2)
[1] 68.79 269.87 603.23
```

- Length of a vector

```
> length(x)
[1] 3
```

Indexing elements of a vector

- Vector elements are selected with a set of indices locating in square brackets:

✓ *Regular indexing*

```
> x <- 11 :  20
> x [3]
[1] 13
```

```
> x <- seq(0, 1, by=0.1)
> x [c(1, 2, 5)]
[1] 0.0 0.1 0.4
```

✓ *Reverse indexing*

```
> x <- 1 :  5
> x [-2]
[1] 1 3 4 5
```

```
> x <- c(1, 3, 5, 7, 11, 13)
> x [-(1 :  3)]
[1] 7 11 13
```

Arithmetics for vectors with different lengths

Prior to operation, the elements of the shorter vector are repeated in a cycled way until the length of the longer one is reached.

```
# equivalent to c(1,2,1,2) + c(1,2,3,4)
> c(1,2) + c(1,2,3,4)
```

```
[1] 2 4 4 6
```

If the length of the longer vector is not a multiple of the length of the shorter one, a warning is printed.

```
# equivalent to c(1,2,1,2,1) + c(1,2,3,4,5)
> c(1,2) + c(1,2,3,4,5)
```

```
[1] 2 4 4 6 6
Warning message:
In c(1, 2) + c(1, 2, 3, 4, 5) :  longer object length is
not a multiple of shorter object length
```

Practical Intro-1

Logical values

A boolean expression, e.g. a <= b, returns logical value: TRUE or FALSE. R treats TRUE as 1 and FALSE as 0 when they are used in an arithmetic operation.

```
> 3 <= 5
[1] TRUE
```

```
> TRUE + TRUE + FALSE
[1] 2
```

```
> x <- c(1,2,3,4,5); x < 4
[1] TRUE TRUE TRUE FALSE FALSE
sum(x < 4)
[1] 3
```

Logical functions: any(), all(), which()

Missing values

NA (Not Available) represents missing values

```
> x <- c(2, -1, NA, 5, 0); any(is.na(x))
[1] TRUE
```

```
> x[10]
[1] NA
```

Proceses with missing values result in NA unless the answer is clear.

```
> sum(x)
[1] NA
```

```
> NA & TRUE
[1] NA
```

```
> NA | TRUE
[1] TRUE
```

However, some functions can exclude missing values from calculations. Users can extract data without missing values using the function `na.omit()`

```
> sum(x, na.rm = TRUE)
[1] 6
```

```
> na.omit(x)
[1] 2 -1 5 0
attr(,"na.action")
[1] 3
attr(,"class")
[1] "omit"
```

Matrices

Matrices can be produced using the function matrix()

Function Parameters

nrow number of rows

ncol number of columns

byrow logical. If FALSE (the default), the matrix is filled by column

Examples

```
> (M <- matrix(1:6, ncol=3, byrow=FALSE))
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> (M <- matrix(c(1,2,3,4,5,6), ncol=3, byrow=TRUE))
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Matrix arithmetics

```
> (M1 <- matrix(c(2,1,5,-1),ncol=2));(M2 <- matrix(1:4,2))
```

```
     [,1] [,2]
[1,]   2    5
[2,]   1   -1
```

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

```
> 3 * M
     [,1] [,2]
[1,]   6   15
[2,]   3   -3
```

```
> M1 + M2
     [,1] [,2]
[1,]   3    8
[2,]   3    3
```

```
> M1 %*% M2
     [,1] [,2]
[1,]  12   26
[2,]  -1   -1
```

Practical Example

Instead of a matrix multiplication, use an ordinary multiplication (M1*M2). What is the difference between both operations?

Indexing elements of a matrix

Matrix elements can be indexed using row, column or cell. Reverse indexing is feasible only using rows or columns.

```
> (M <- matrix(1:10, ncol=5, byrow=TRUE))
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
```

Practical Example

Extract the following sub-matrix from M:

```
     [,1] [,2] [,3]
[1,]    2    3    4
[2,]    7    8    9
```

Indexing Matrix

- Row:

```
> M [2,]
[1] 6 7 8 9 10
```

- Column:

```
> M [,4]
[1] 4 9
```

- Cell:

```
> M [1,5]
[1] 5
```

Break

Objects

The functions objects() and ls() show a list of assigned objects. Data type of an object can be shown using the function mode(ObjectName)

```
> objects()
[1] "A" "BMI" "days" "h" "M" "M1" "M2" "w" "x" "y"
```

```
> mode(M); mode(days); mode (c(TRUE, FALSE, FALSE))
[1] "numeric"
[1] "character"
[1] "logical"
```

Objects' further attributes (if any) can be shown using attributes(). The structure of an object can be shown using str().

```
> attributes(M)
$dim
[1] 2 5
```

```
> str(M)
int [1:2, 1:5] 1 6 2 7 3 8 4 9 5 10
```

Atomic data types

- `numeric`: integer and real (defined as double) numbers.
- `complex`: complex numbers
- `character`: strings
- `logical`: logical values
- `NULL`: An empty object

Data types can be shown using the `typeof()` function. They can be tested using `is.numeric()`, `is.complex()`, ... , `is.null()`.

```
> x <- c(1,2); y <- c(2i, 3); gender <- c("m", "f")
> typeof(x); typeof(x==2)
 [1] "double"
 [1] "logical"
```

```
> is.numeric(x); is.numeric(y)
[1] TRUE
[1] FALSE
```

Factor class

The factor data type is used for categorical data. It can be produced by the function factor().

```
> Groups <- factor(c("Treatment", "Control", "Treatment",
+ "Control", "Control"))
> Groups
```

```
[1] Treatment Control Treatment Control Control
Levels:  Control Treatment
```

Remark

The factor data is coded with numbers. Note, mode() leads to numeric. Nevertheless, factors can be identified by class()

Practical Example

- Check the mode and class of the Groups vector.
- Convert the Groups vector to a numerical form.

Types of data structures

Structures of R's data can be organised by their dimensionality and whether they're homogeneous or heterogeneous:

dim.	Homogeneous	Heterogeneous
1d	`vector`	`data frame`
2d	`matrix`	`list`
nd	`array`	

```
> ?array
> ?data.frame
```

Concatenating of structures

The functions rbind() and cbind() concatenate data structures by row and by column respectively.

```
> Patient <- c(102, 105); gender <- factor(c("F", "M"))
> Heart.R <- c(83, 78)
> (Rates <- cbind(Patient, Heart.R))

     Patient Heart.R
[1,]     102      83
[2,]     105      78

> class(Rates)
[1] "matrix"

> HeartData <- as.data.frame(Rates)
> class(HeartData)
[1] "data.frame"
```

Practical Example

Add the vector gender to the matrix Rates and to the data frame HeartData. What do you observe?

Data frames

Data frames consist of vectors (of equal sizes) which, unlike to matrices, may represent different types of data (heterogeneous). The function data.frame() can generate them.

```
> participant <- c(1:4); group <- c("T", "C", "C", "T")
> age <- c(22, 18, 33, 45); BMI <- c(25, 18, 32, 36)
> (MyData <- data.frame(participant, group, age, BMI))

   participant  group  age  BMI
1            1      T   22   25
2            2      C   18   18
3            3      C   33   32
4            4      T   45   36
```

Practical Example

Show the structure and the attributes of the created object MyData.

Data frames

- Edit elements, row and/or column names

```
> MyData$participant <- MyData$participant + 100
> MyData
   participant  group  age  BMI
 1         101      T   22   25
 2         102      C   18   18
 3         103      C   33   32
 4         104      T   45   36
> colnames(MyData) <- c("Patient", "Treat.", "Age", "BMI")
# Similarly, use rownames() for row names
> MyData[1:2,]

   Patient  Treat.  Age  BMI
 1     101       T   22   25
 2     102       C   18   18
```

Merging of data frames

```
> merge(MyData, HeartData)
   Patient  Treat.  Age  BMI  Heart.R
1      102       C   18   18       83

> merge(MyData, HeartData, all = TRUE)
   Patient  Treat.  Age  BMI  Heart.R
1      101       T   22   25       NA
2      102       C   18   18       83
3      103       C   33   32       NA
4      104       T   45   36       NA
5      105      NA   NA   NA       78
```

Practical Example

Merge the data frames MyData and HeartData together: (a) with the parameter all.x=TRUE; (b) with the parameter all.y=TRUE. What do you observe ?

Import data sets

Data sets that internally stored with R do not need to be loaded

```
> data(iris); head(iris)
```

- view specific rows, e.g. rows number 7 and 11

```
> iris[c(7,11),]
```

- view specific columns, e.g. columns number 3, 4 and 5

```
> iris[,3:5]
```

To import and/or export data from/to an external file - on your local drive - you need to specify the path to the folder containing/that will contain the file.

- Show current working directory

```
> getwd()
```

- Change working directory

```
> setwd(‘‘C:/R-courses/RIntro’’)
```

Import and export data sets

- List all files in a given path

```
> list.files(getwd())
```

- export data sets to csv or txt file formats can be done using:

```
> write.csv(iris, "Mydata.csv", row.names = FALSE)
> write.table(iris, "Mydata.txt", row.names = FALSE)
```

- import data from csv or txt files:

```
> Im.data1 = read.csv(file = "Mydata.csv", header = TRUE)
> Im.data2 = read.table("Mydata.txt", header = TRUE)
```

- Get more information on data set

```
> attributes(iris); str(iris)
```

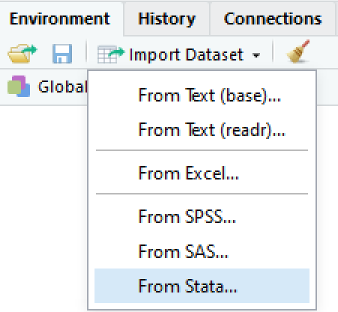
- Show class of each variable

```
> sapply(iris, class); lapply(iris, class)
```

Import and export data sets

- Import from other data formats using haven function:
  - `> install.packages("haven")`
  - `> library(haven)`

- SPSS:
  - `> read_sav()`
- Excel sheets:
  - `> read_excel()`
- Stata:
  - `> read_stata()`



Practical Intro-2

What is an R function?

- A pre-defined set of actions\calculations that can be called by function's name:

```
> mean(x = c(2,5,8,11))
[1] 6.5
```

Function arguments

a set of input parameters associated with functions (e.g., `x` in the function `mean`).

- You can find out arguments of a function and their descriptions using R help: `?function-name`

```
> ?mean
```

Creating functions

Your own function is defined using `function()` as follows:

Syntax for creating a function

```
function-name <- function(argument1, argument1, ...){
                 # function body
                 }
```

```
> BMI <- function(h, w){
+ Index <- w/(h^2)
+ return(Index)
+ }
```

```
> BMI(h=1.75, w=100)
[1] 32.65306
```

```
> BMI(h=1.80, w=62)
[1] 19.1358
```

Usage of functions

Functions can be called either with or without explicit argument assignments. Without explicit assignment, parameters should be given such that the order of arguments is crucial.

Examples

```
> # with explicit assignment
> BMI(w=94, h=1.70)
[1] 32.52595

> # without explicit assignment
> BMI(1.70, 94)
[1] 32.52595
```

Conditional execution

General syntax of if-statement

```
if(condition){
    statement(s)
}
```

OR

```
if(condition){
    statement(s)
} else {
    statement(s)
  }
```

Example

The created function BMI produce sensable results for positive values. What action(s) do you want to take when h or w parameter has a non-positive value?

Conditional execution

BMI - conditional execution: the if block

```
> BMI <- function(h, w){
+   if (h <= 0 | w <= 0){
+     cat('warning:  non-positive height or weight\n')
+      h <- abs(h); w <- abs(w)
+   }
+   Index <- w/(h^2)
+   return(Index)
+ }

> BMI(-1.7,80)
warning:  non-positive height or weight
[1] 27.68166
```

Conditional execution

BMI - conditional branching: the if-else block

```
> BMI <- function(h, w){
+   if (h > 0 & w > 0){
+     Index <- w/(h^2)
+     return(Index)
+   } else {
+     stop(’non-positive height or weight\n’)
+     }
+ }

> BMI(-1.7,80)
Error in BMI(-1.7, 80) :  non-positive height or weight
```

Conditional execution

ifelse-function

```
ifelse(condition, statement if TRUE, statement if FALSE)
```

Remark

The condition in the `ifelse` function is allowed to be on a vector, a matrix or a data frame

```
> x <- 1:5
# Is the modulo (remainder of division) of 2 is zero?
> ifelse(x %% 2 == 0, "even", x)
[1] "1" "even" "3" "even" "5"
```

FOR-loop

Loops execute iteratively the programme steps - using a different index for each step.

General syntax of `for`-loop

```
for (i in a:b) {statement(s)}
```

In other language, for each element in a:b execute the statements.

```
> h <- c(1,3,5)
> for (i in h){
+ print(i)
+ }

[1] 1 [1] 3 [1] 5
```

```
> show = c()
> for (i in 1:3){
+ show[i] <- ifelse(i %% 2 == 0,
"E", "O") + }
> show
[1] "O" "E" "O"
```

REPEAT-loop

General syntax of repeat-loop

```
repeat{
    statement(s)
    if (condition){break}
}
```

Statement(s) is (are) repeated, at least once, till the condition is met.

Example

```
> show <- 0
> repeat{
+ show <- show + 1
+ if(show == 4){break}
+ }
> show
[1] 4
```

WHILE-loop

General syntax of while-loop

```
while(condition){
    statement(s)
}
```

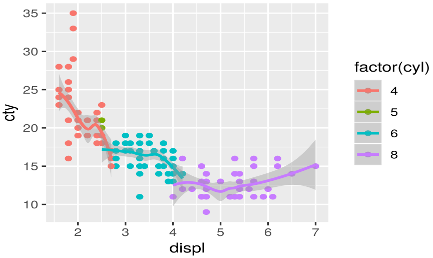
Statement(s) is (are) repeated iteratively as long as the condition is fulfilled.

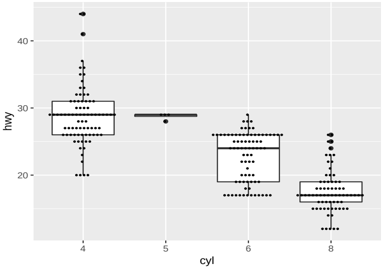
Example

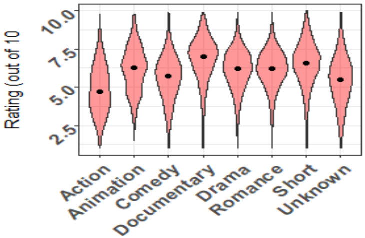
```
> set.seed(1234)    # to get as same outputs as here
> show <- c()
> while(length(show) < 3){
+ show[length(show)+1] <- rnorm(1)
+ }
> show
[1] -1.2070657 0.2774292 1.0844412
```

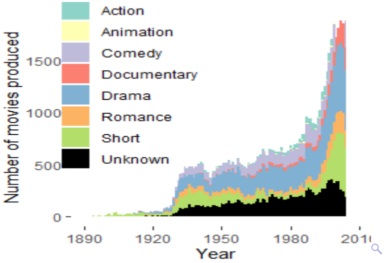
Getting More - High quality plots

ggplot2 allows you to create plots with publication quality



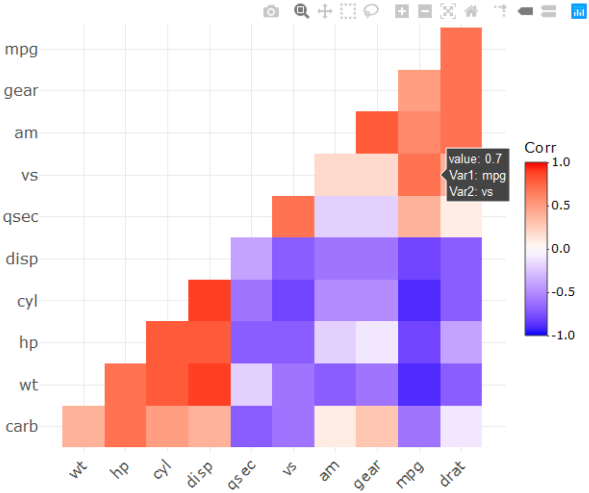






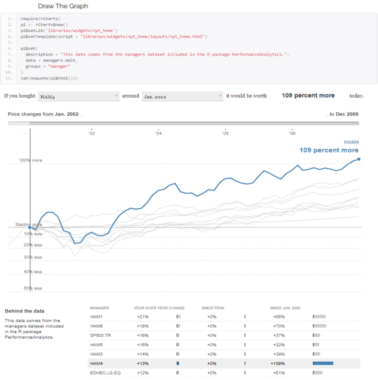
Getting More - Interactive plots

plotly allows you to create interactive plots



Getting More - Dynamic reports

R Markdown allows you to create: dynamic/interactive reports including narrative text, script, and output; dash, handout tutorials; presentations; dashboards; and more.



Draw The Graph

```r
require(rCharts)
p2 <- rCharts$new()
p2$setLib('libraries/widgets/nyt_home')
p2$setTemplate(script = "libraries/widgets/nyt_home/layouts/nyt_home.html")

p2$set(
  description = "This data comes from the managers dataset included in the R package PerformanceAnalytics.",
  data = managers.melt,
  groups = "manager"
)
cat(noquote(p2$html()))
```

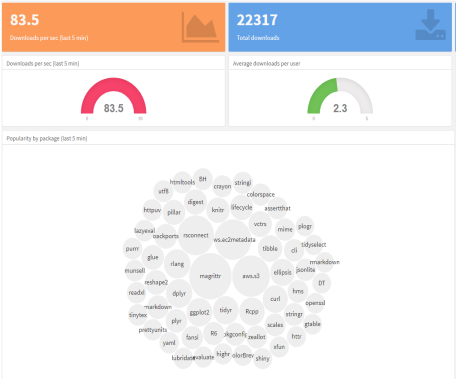
If you bought HAM4 around Jan. 2002 it would be worth **109 percent more** today.

Price changes from **Jan. 2002**... ...to **Dec 2006**

Behind the data

This data comes from the managers dataset included in the R package PerformanceAnalytics.

MANAGER	YEAR-OVER-YEAR CHANGE	SINCE PEAK	SINCE JAN. 2002
HAM1	+21%	+0%	+68%
HAM6	+18%	+0%	+70%
SP500.TR	+16%	+0%	+37%
HAM5	+16%	+0%	+32%
HAM3	+14%	+0%	+39%
HAM4	**+13%**	**+0%**	**+109%**
EDHEC.LS.EQ	+12%	+0%	+51%



83.5
Downloads per sec (last 5 min)

22317
Total downloads

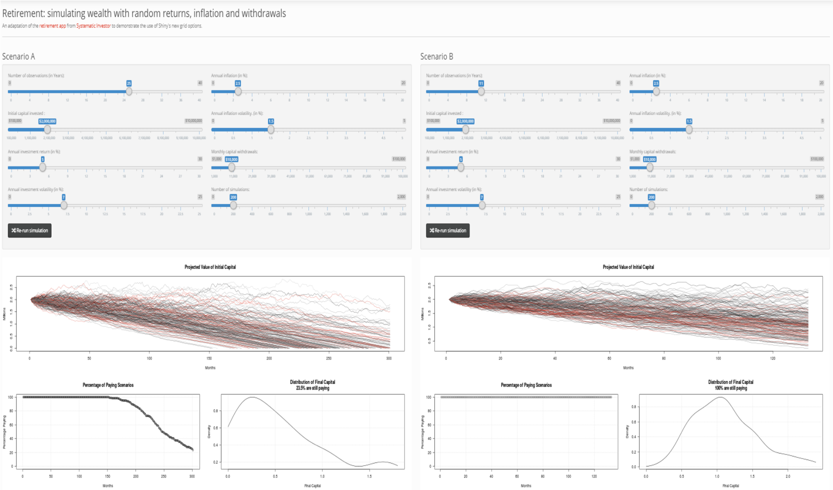
Downloads per sec (last 5 min)

Average downloads per user

Popularity by package (last 5 min)

Getting More - Web apps

Shiny allows you to create: interactive apps without any knowledge of HTML or Javascript!



Retirement: simulating wealth with random returns, inflation and withdrawals

An adaptation of the retirement app from Systematic Investor to demonstrate the use of Shiny's new grid options.

Scenario A

Number of observations (in Years): 25 (0–40)
Annual inflation (in %): 2.5 (0–20)
Initial capital invested : $2,000,000 ($100,000–$10,000,000)
Annual inflation volatility. (in %): 1.5 (0–5)
Annual investment return (in %): 5 (0–30)
Monthly capital withdrawals: $10,000 ($1,000–$100,000)
Annual investment volatility (in %): 7 (0–25)
Number of simulations: 200 (0–2,000)

Scenario B

Number of observations (in Years): 11 (0–40)
Annual inflation (in %): 2.5 (0–20)
Initial capital invested : $2,000,000 ($100,000–$10,000,000)
Annual inflation volatility. (in %): 1.5 (0–5)
Annual investment return (in %): 5 (0–30)
Monthly capital withdrawals: $10,000 ($1,000–$100,000)
Annual investment volatility (in %): 7 (0–25)
Number of simulations: 200 (0–2,000)

Practical Intro-3