An introduction to R

Contents

An introduction to R

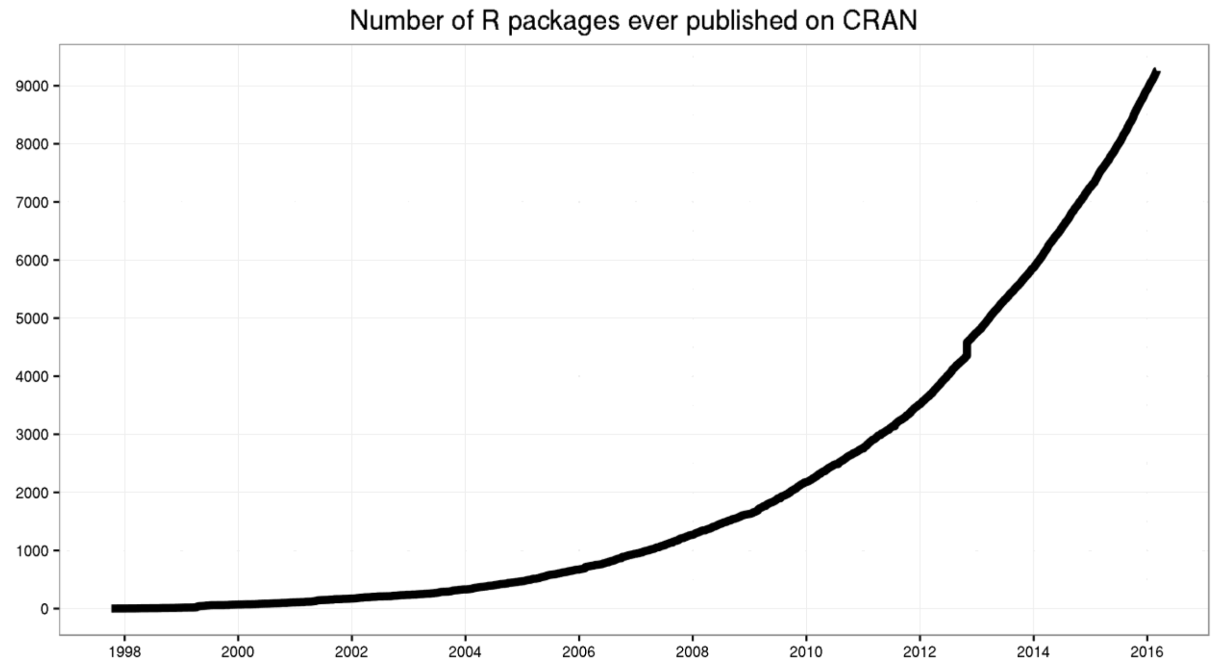
What is R?

- R is a language and environment for statistical computing, data manipulation and graphics.

- R is an interpreted computer language.
  - Similar to matlab /python.
  - Can interact with C, C++ languages for efficiency.
  - System commands can be called from within R.

- R is made up of:
  - Operators (+ - <- * %*% …) for calculations on arrays & matrices.
  - Large, coherent, integrated collection of functions.
  - **User written functions & sets of functions (packages).**

An introduction to R
Packages!



R packages for data analysis

- **ggplot2**: Visualization and graphics.
- **dplyr and tidyr**: Data tranformation.
- **Lubridate**: Date management.
- **Stringr**: Wrappers for common string operations.
- **MASS, h2o, kernlab, glmnet, caret, rpart, randomforest, gbm:** Machine Learning packages.
- **Shiny**: Create user interfaces.

Advantages and disadvantages

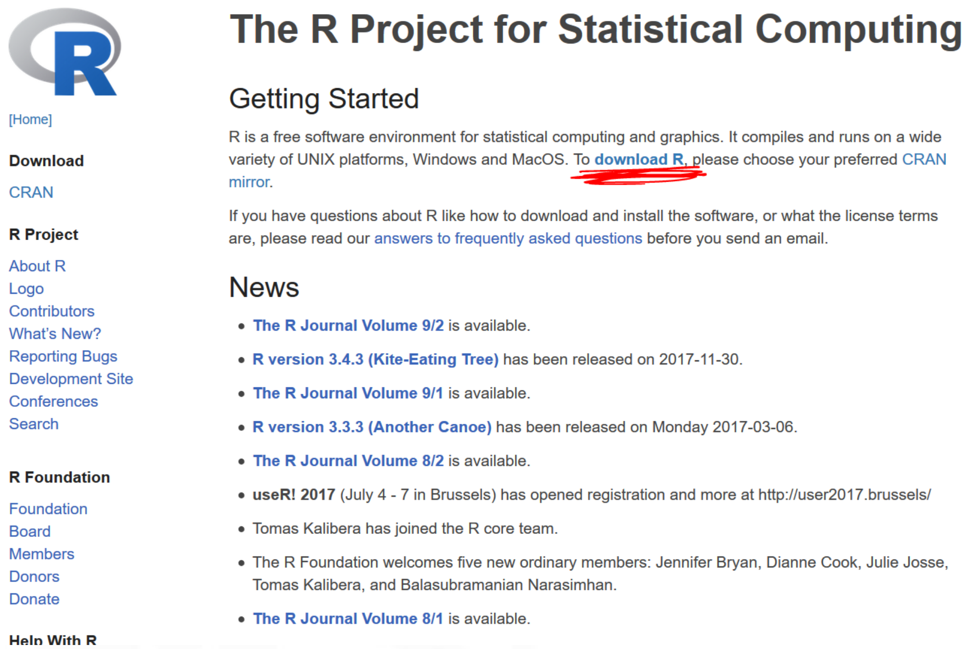
- **Advantages**
  - Free and open source.
  - State of the art: Statistical researchers provide their methods as R packages.
  - Active user community.

- **Disadvantages**
  - No commercial support; figuring out correct methods or how to use a function on your own can be frustrating.
  - Easy to make mistakes and not know.
  - As the number of packages grows, it becomes difficult to choose the best package for your needs.

An introduction to R

R installation

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



[Home]

Download

CRAN

R Project

About R
Logo
Contributors
What's New?
Reporting Bugs
Development Site
Conferences
Search

R Foundation

Foundation
Board
Members
Donors
Donate

Help With R

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred CRAN mirror.

If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

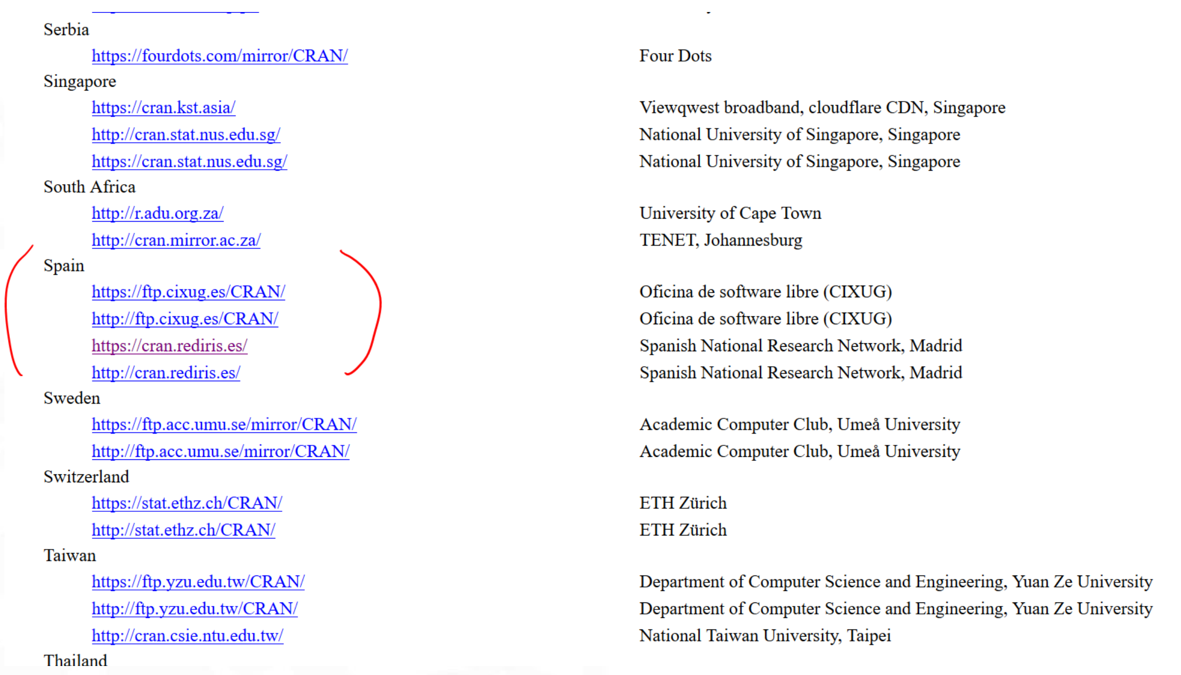
News

- **The R Journal Volume 9/2** is available.
- **R version 3.4.3 (Kite-Eating Tree)** has been released on 2017-11-30.
- **The R Journal Volume 9/1** is available.
- **R version 3.3.3 (Another Canoe)** has been released on Monday 2017-03-06.
- **The R Journal Volume 8/2** is available.
- **useR! 2017** (July 4 - 7 in Brussels) has opened registration and more at http://user2017.brussels/
- Tomas Kalibera has joined the R core team.
- The R Foundation welcomes five new ordinary members: Jennifer Bryan, Dianne Cook, Julie Josse, Tomas Kalibera, and Balasubramanian Narasimhan.
- **The R Journal Volume 8/1** is available.

An introduction to R

R installation

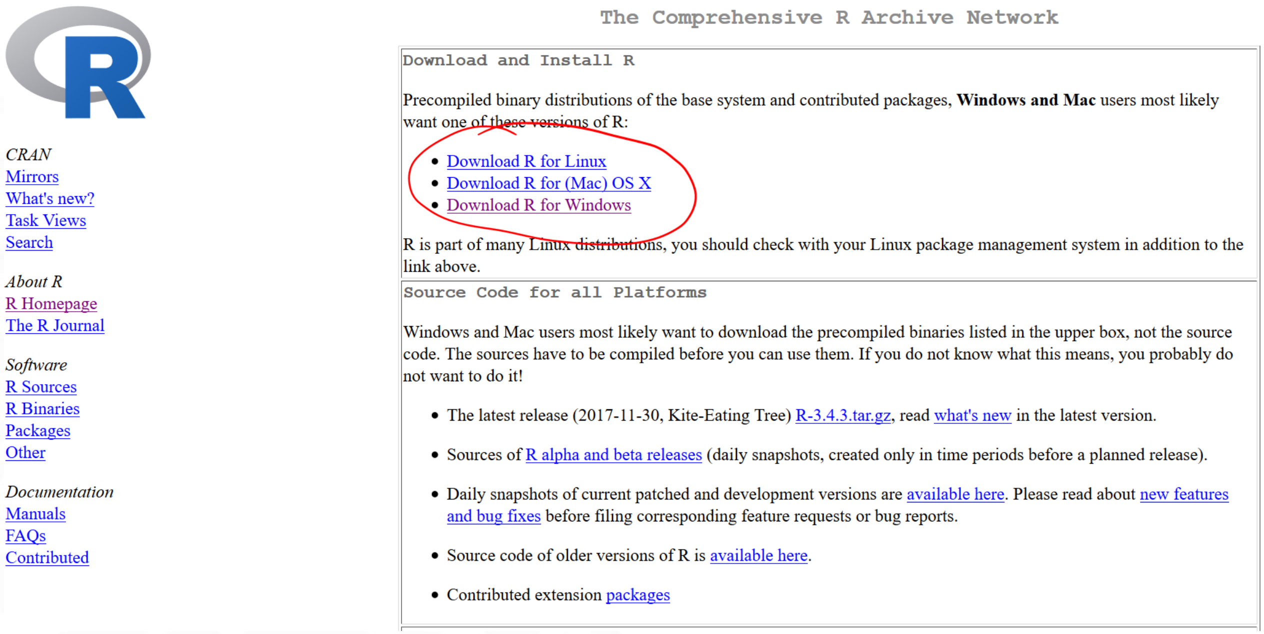
- Select Mirror (Download is“faster” from Spain)



Serbia	
https://fourdots.com/mirror/CRAN/	Four Dots
Singapore	
https://cran.kst.asia/	Viewqwest broadband, cloudflare CDN, Singapore
http://cran.stat.nus.edu.sg/	National University of Singapore, Singapore
https://cran.stat.nus.edu.sg/	National University of Singapore, Singapore
South Africa	
http://r.adu.org.za/	University of Cape Town
http://cran.mirror.ac.za/	TENET, Johannesburg
Spain	
https://ftp.cixug.es/CRAN/	Oficina de software libre (CIXUG)
http://ftp.cixug.es/CRAN/	Oficina de software libre (CIXUG)
https://cran.rediris.es/	Spanish National Research Network, Madrid
http://cran.rediris.es/	Spanish National Research Network, Madrid
Sweden	
https://ftp.acc.umu.se/mirror/CRAN/	Academic Computer Club, Umeå University
http://ftp.acc.umu.se/mirror/CRAN/	Academic Computer Club, Umeå University
Switzerland	
https://stat.ethz.ch/CRAN/	ETH Zürich
http://stat.ethz.ch/CRAN/	ETH Zürich
Taiwan	
https://ftp.yzu.edu.tw/CRAN/	Department of Computer Science and Engineering, Yuan Ze University
http://ftp.yzu.edu.tw/CRAN/	Department of Computer Science and Engineering, Yuan Ze University
http://cran.csie.ntu.edu.tw/	National Taiwan University, Taipei
Thailand	

R installation

- Choose the software:



CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

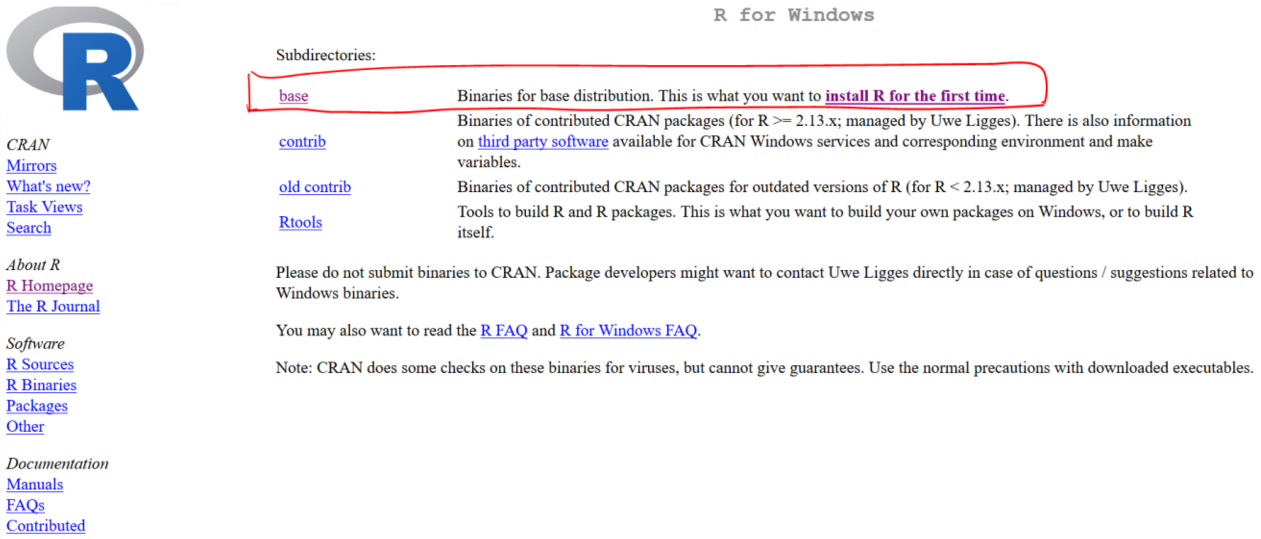
Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2017-11-30, Kite-Eating Tree) R-3.4.3.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

An introduction to R
R installation

- Base installation is what you want:



R for Windows

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Subdirectories:

base	Binaries for base distribution. This is what you want to **install R for the first time**.
contrib	Binaries of contributed CRAN packages (for R >= 2.13.x; managed by Uwe Ligges). There is also information on third party software available for CRAN Windows services and corresponding environment and make variables.
old contrib	Binaries of contributed CRAN packages for outdated versions of R (for R < 2.13.x; managed by Uwe Ligges).
Rtools	Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

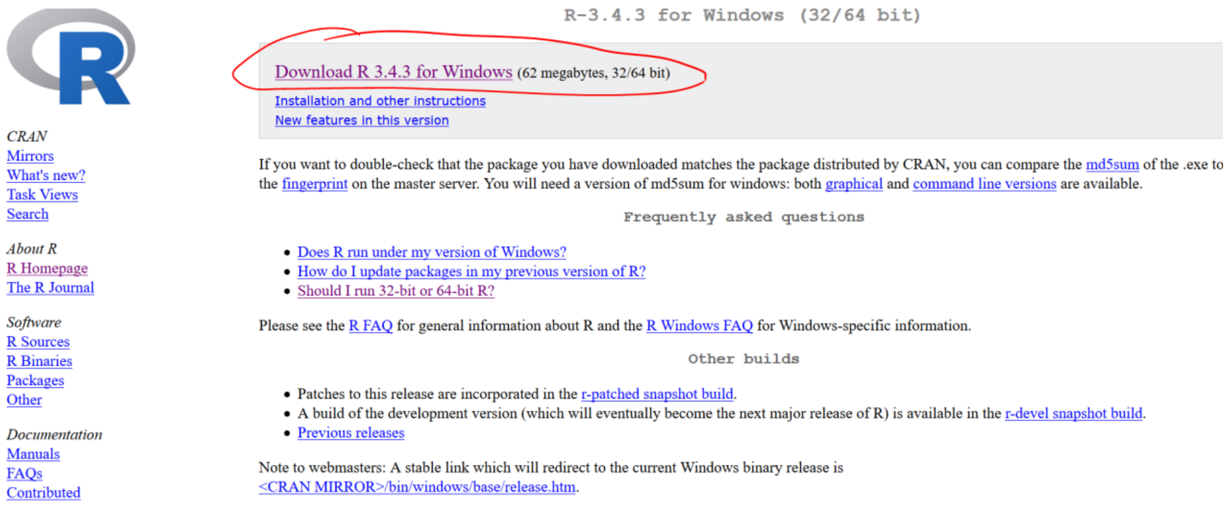
You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

An introduction to R

R installation

- Download executable file and install.



CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

R-3.4.3 for Windows (32/64 bit)

Download R 3.4.3 for Windows (62 megabytes, 32/64 bit)
Installation and other instructions
New features in this version

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master server. You will need a version of md5sum for windows: both graphical and command line versions are available.

Frequently asked questions

- Does R run under my version of Windows?
- How do I update packages in my previous version of R?
- Should I run 32-bit or 64-bit R?

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

Other builds

- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

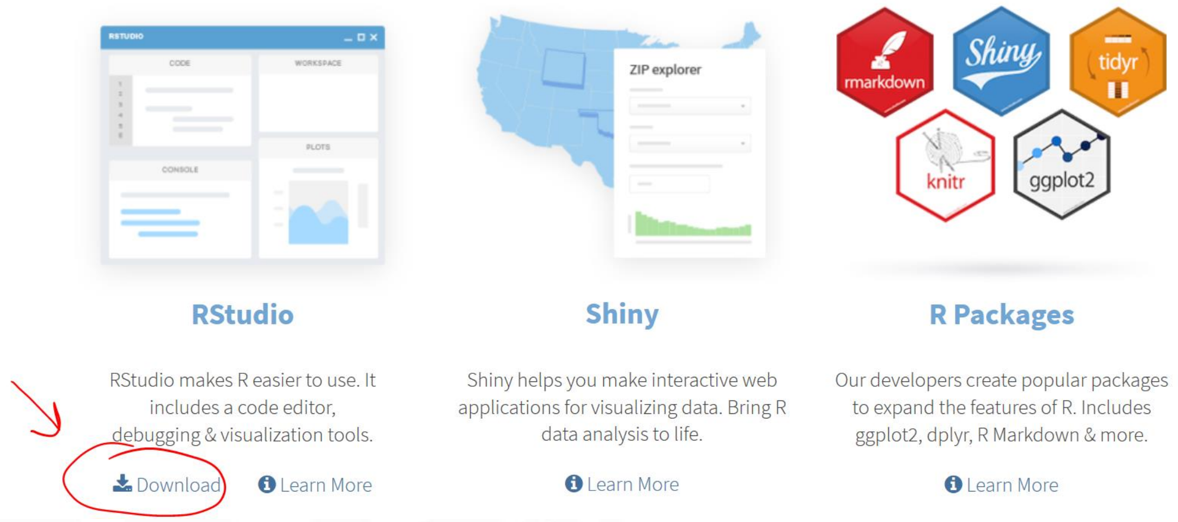
Note to webmasters: A stable link which will redirect to the current Windows binary release is
<CRAN MIRROR>/bin/windows/base/release.htm.

- **Note**: 32bit vs 64bit depends on:
  - Your computer's processor
  - The amount of data that can be managed (32bit restricted to 4GB of RAM and objects of maximum 1GB)

R-Studio installation

https://www.rstudio.com/

- Scroll down to download link.



RStudio

RStudio makes R easier to use. It includes a code editor, debugging & visualization tools.

Download Learn More

Shiny

Shiny helps you make interactive web applications for visualizing data. Bring R data analysis to life.

Learn More

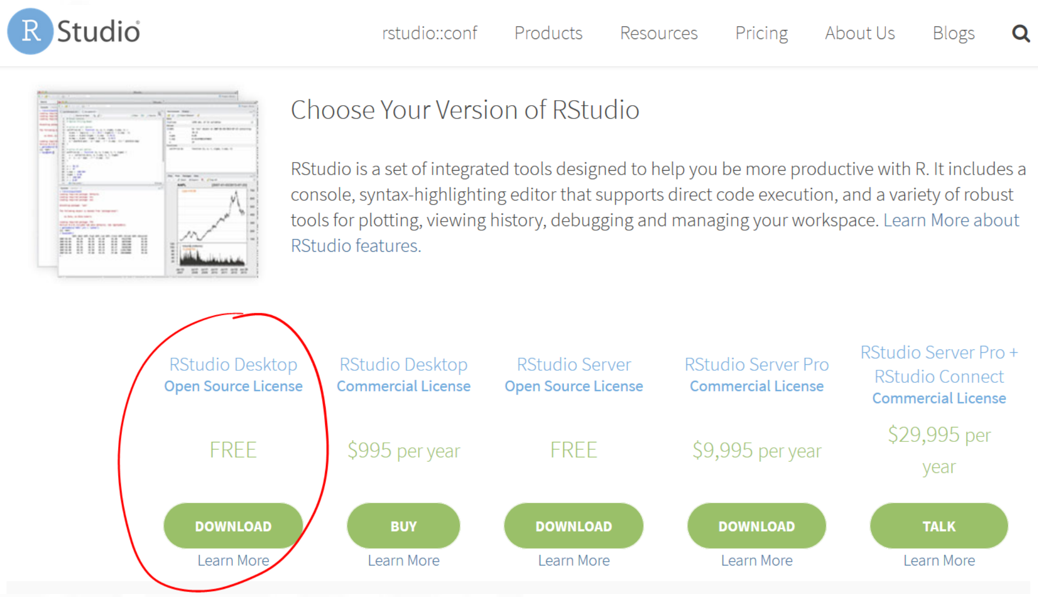
R Packages

Our developers create popular packages to expand the features of R. Includes ggplot2, dplyr, R Markdown & more.

Learn More

R-Studio installation

- Install free version.



Choose Your Version of RStudio

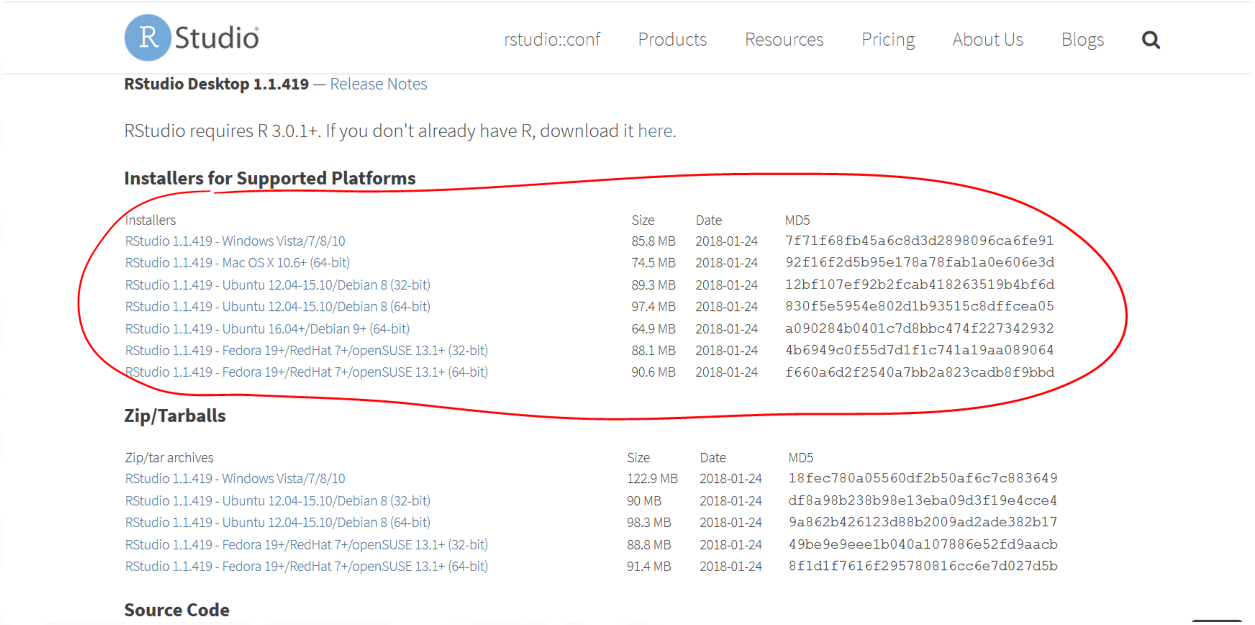
RStudio is a set of integrated tools designed to help you be more productive with R. It includes a console, syntax-highlighting editor that supports direct code execution, and a variety of robust tools for plotting, viewing history, debugging and managing your workspace. Learn More about RStudio features.

RStudio Desktop Open Source License	RStudio Desktop Commercial License	RStudio Server Open Source License	RStudio Server Pro Commercial License	RStudio Server Pro + RStudio Connect Commercial License
FREE	$995 per year	FREE	$9,995 per year	$29,995 per year
DOWNLOAD	BUY	DOWNLOAD	DOWNLOAD	TALK
Learn More	Learn More	Learn More	Learn More	Learn More

An introduction to R

R-Studio installation

- Choose executable file for your system.



RStudio Desktop 1.1.419 — Release Notes

RStudio requires R 3.0.1+. If you don't already have R, download it here.

Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.1.419 - Windows Vista/7/8/10	85.8 MB	2018-01-24	7f71f68fb45a6c8d3d2898096ca6fe91
RStudio 1.1.419 - Mac OS X 10.6+ (64-bit)	74.5 MB	2018-01-24	92f16f2d5b95e178a78fab1a0e606e3d
RStudio 1.1.419 - Ubuntu 12.04-15.10/Debian 8 (32-bit)	89.3 MB	2018-01-24	12bf107ef92b2fcab418263519b4bf6d
RStudio 1.1.419 - Ubuntu 12.04-15.10/Debian 8 (64-bit)	97.4 MB	2018-01-24	830f5e5954e802d1b93515c8dffcea05
RStudio 1.1.419 - Ubuntu 16.04+/Debian 9+ (64-bit)	64.9 MB	2018-01-24	a090284b0401c7d8bbc474f227342932
RStudio 1.1.419 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	88.1 MB	2018-01-24	4b6949c0f55d7d1f1c741a19aa089064
RStudio 1.1.419 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	90.6 MB	2018-01-24	f660a6d2f2540a7bb2a823cadb8f9bbd

Zip/Tarballs

Zip/tar archives	Size	Date	MD5
RStudio 1.1.419 - Windows Vista/7/8/10	122.9 MB	2018-01-24	18fec780a05560df2b50af6c7c883649
RStudio 1.1.419 - Ubuntu 12.04-15.10/Debian 8 (32-bit)	90 MB	2018-01-24	df8a98b238b98e13eba09d3f19e4cce4
RStudio 1.1.419 - Ubuntu 12.04-15.10/Debian 8 (64-bit)	98.3 MB	2018-01-24	9a862b426123d88b2009ad2ade382b17
RStudio 1.1.419 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	88.8 MB	2018-01-24	49be9e9eee1b040a107886e52fd9aacb
RStudio 1.1.419 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	91.4 MB	2018-01-24	8f1d1f7616f295780816cc6e7d027d5b

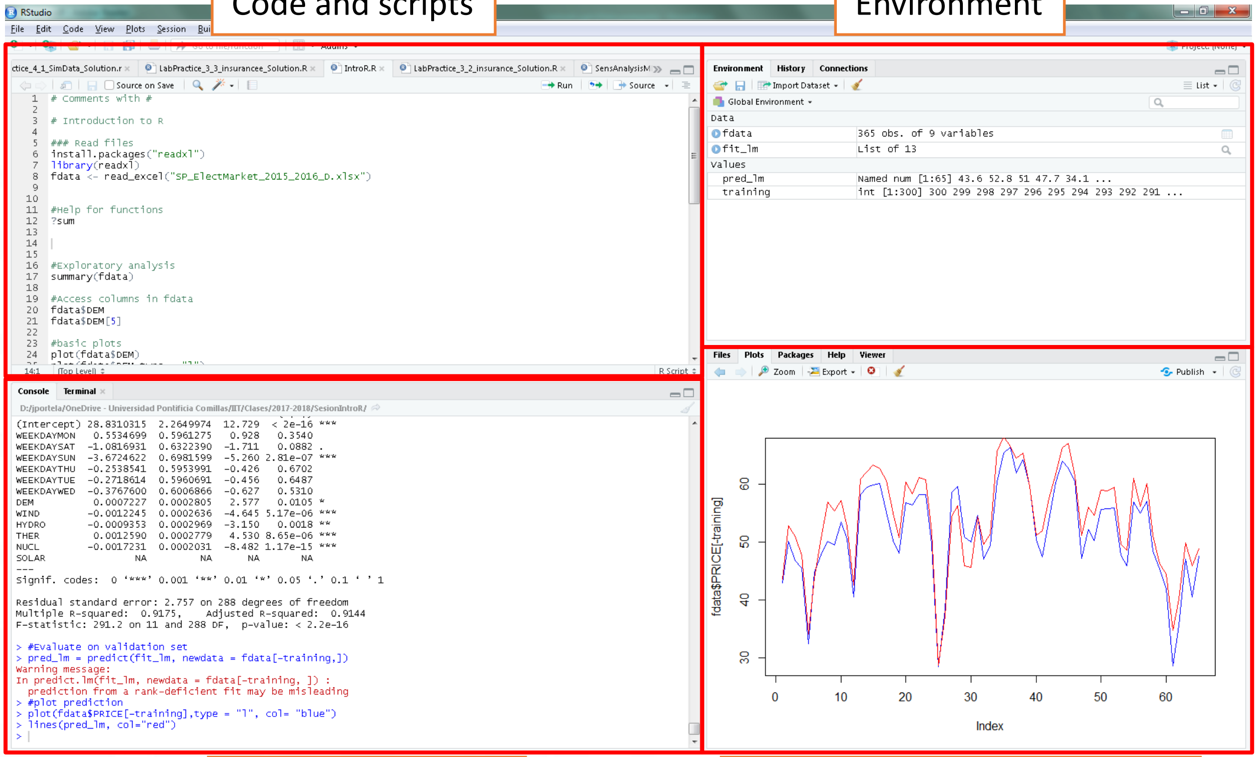
Source Code

An introduction to R

RStudio

Code and scripts

Environment



```r
# Comments with #

# Introduction to R

### Read files
install.packages("readxl")
library(readxl)
fdata <- read_excel("SP_ElectMarket_2015_2016_D.xlsx")


#Help for functions
?sum



#Exploratory analysis
summary(fdata)

#Access columns in fdata
fdata$DEM
fdata$DEM[5]

#basic plots
plot(fdata$DEM)
```

```
Data
fdata        365 obs. of 9 variables
fit_lm       List of 13
Values
pred_lm      Named num [1:65] 43.6 52.8 51 47.7 34.1 ...
training     int [1:300] 300 299 298 297 296 295 294 293 292 291 ...
```

```
D:/jportela/OneDrive - Universidad Pontificia Comillas/IIT/Clases/2017-2018/SesionIntroR/
(Intercept) 28.8310315  2.2649974  12.729  < 2e-16 ***
WEEKDAYMON   0.5534699  0.5961275   0.928   0.3540
WEEKDAYSAT  -1.0816931  0.6322390  -1.711   0.0882 .
WEEKDAYSUN  -3.6724622  0.6981599  -5.260 2.81e-07 ***
WEEKDAYTHU  -0.2538541  0.5953991  -0.426   0.6702
WEEKDAYTUE  -0.2718614  0.5960691  -0.456   0.6487
WEEKDAYWED  -0.3767600  0.6006866  -0.627   0.5310
DEM          0.0007227  0.0002805   2.577   0.0105 *
WIND        -0.0012245  0.0002636  -4.645 5.17e-06 ***
HYDRO       -0.0009353  0.0002969  -3.150   0.0018 **
THER         0.0012590  0.0002779   4.530 8.65e-06 ***
NUCL        -0.0017231  0.0002031  -8.482 1.17e-15 ***
SOLAR               NA         NA      NA       NA
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.757 on 288 degrees of freedom
Multiple R-squared:  0.9175,    Adjusted R-squared:  0.9144
F-statistic: 291.2 on 11 and 288 DF,  p-value: < 2.2e-16

> #Evaluate on validation set
> pred_lm = predict(fit_lm, newdata = fdata[-training,])
Warning message:
In predict.lm(fit_lm, newdata = fdata[-training, ]) :
  prediction from a rank-deficient fit may be misleading
> #plot prediction
> plot(fdata$PRICE[-training],type = "l", col= "blue")
> lines(pred_lm, col="red")
> 
```

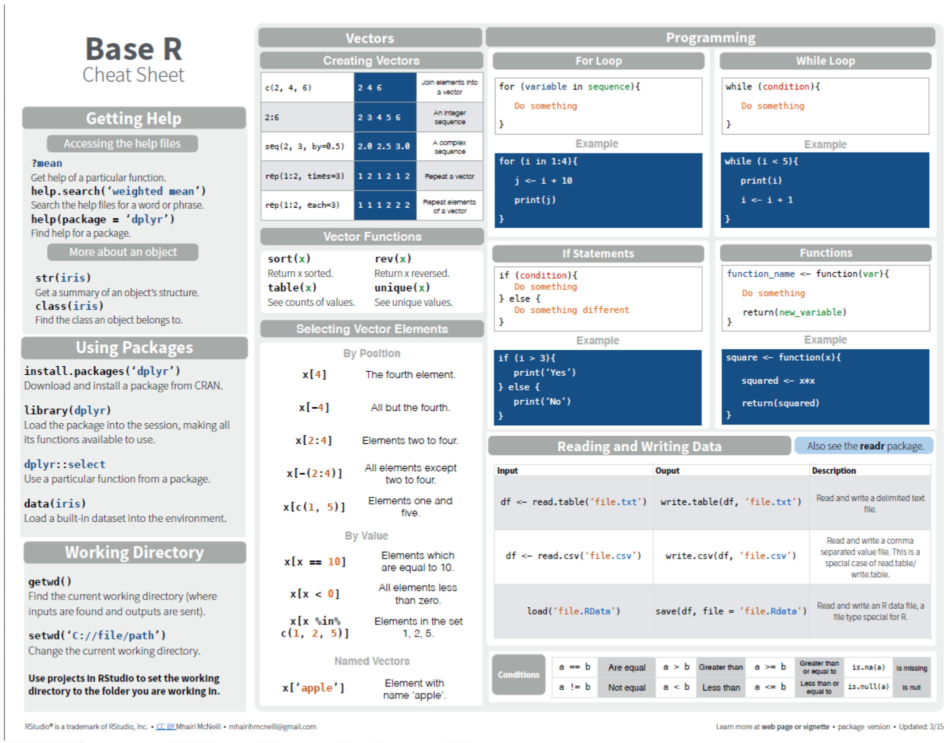
Command window

Graphics, files, packages, help

An introduction to R

Cheat sheets

https://www.rstudio.com/resources/cheatsheets



Base R
Cheat Sheet

Getting Help

Accessing the help files

`?mean`
Get help of a particular function.
`help.search('weighted mean')`
Search the help files for a word or phrase.
`help(package = 'dplyr')`
Find help for a package.

More about an object

`str(iris)`
Get a summary of an object's structure.
`class(iris)`
Find the class an object belongs to.

Using Packages

`install.packages('dplyr')`
Download and install a package from CRAN.

`library(dplyr)`
Load the package into the session, making all its functions available to use.

`dplyr::select`
Use a particular function from a package.

`data(iris)`
Load a built-in dataset into the environment.

Working Directory

`getwd()`
Find the current working directory (where inputs are found and outputs are sent).

`setwd('C://file/path')`
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

`c(2, 4, 6)`	2 4 6	Join elements into a vector
`2:6`	2 3 4 5 6	An integer sequence
`seq(2, 3, by=0.5)`	2.0 2.5 3.0	A complex sequence
`rep(1:2, times=3)`	1 2 1 2 1 2	Repeat a vector
`rep(1:2, each=3)`	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

`sort(x)` Return x sorted.	`rev(x)` Return x reversed.
`table(x)` See counts of values.	`unique(x)` See unique values.

Selecting Vector Elements

By Position

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

Named Vectors

`x['apple']`	Element with name 'apple'.

Programming

For Loop

```
for (variable in sequence){
  Do something
}
```

Example

```
for (i in 1:4){
  j <- i + 10
  print(j)
}
```

While Loop

```
while (condition){
  Do something
}
```

Example

```
while (i < 5){
  print(i)
  i <- i + 1
}
```

If Statements

```
if (condition){
  Do something
} else {
  Do something different
}
```

Example

```
if (i > 3){
  print('Yes')
} else {
  print('No')
}
```

Functions

```
function_name <- function(var){
  Do something
  return(new_variable)
}
```

Example

```
square <- function(x){
  squared <- x*x
  return(squared)
}
```

Reading and Writing Data

Also see the **readr** package.

Input	Ouput	Description
`df <- read.table('file.txt')`	`write.table(df, 'file.txt')`	Read and write a delimited text file.
`df <- read.csv('file.csv')`	`write.csv(df, 'file.csv')`	Read and write a comma separated value file. This is a special case of read.table/ write.table.
`load('file.RData')`	`save(df, file = 'file.Rdata')`	Read and write an R data file, a file type special for R.

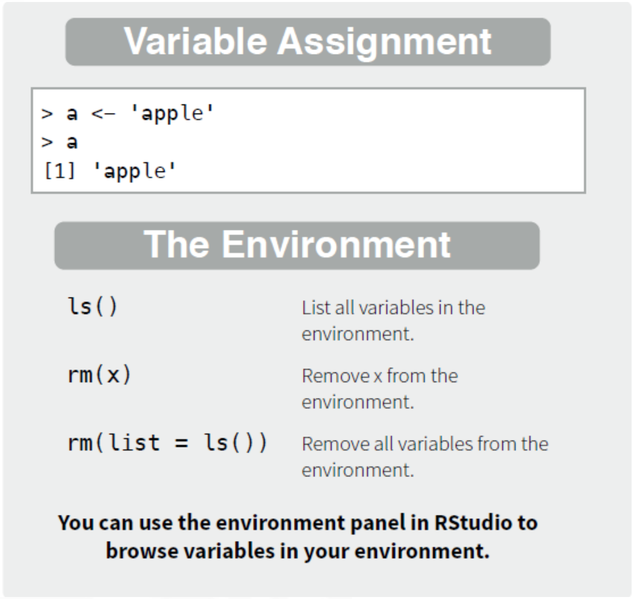
Conditions

`a == b`	Are equal	`a > b`	Greater than	`a >= b`	Greater than or equal to	`is.na(a)`	Is missing
`a != b`	Not equal	`a < b`	Less than	`a <= b`	Less than or equal to	`is.null(a)`	Is null

Learn more at **web page or vignette** • package version • Updated: 3/15

Base R
Starting commands

- Creating and managing variables



Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

`ls()`	List all variables in the environment.
`rm(x)`	Remove x from the environment.
`rm(list = ls())`	Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Base R

Essentials

Getting Help

Accessing the help files

`?mean`

Get help of a particular function.

`help.search('weighted mean')`

Search the help files for a word or phrase.

`help(package = 'dplyr')`

Find help for a package.

More about an object

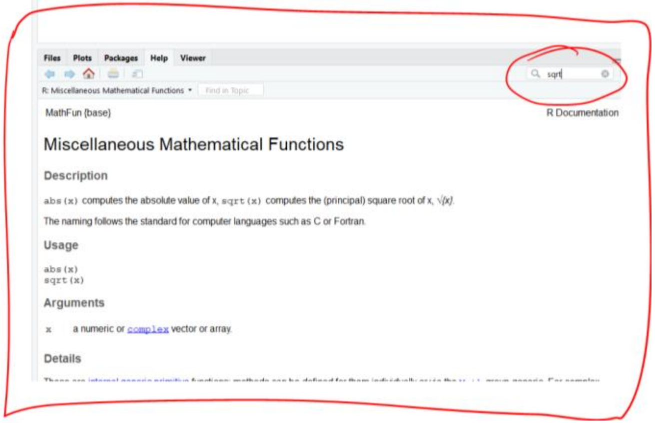
`str(iris)`

Get a summary of an object's structure.

`class(iris)`

Find the class an object belongs to.

Rstudio interface



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

Base R
Essentials

Using Packages

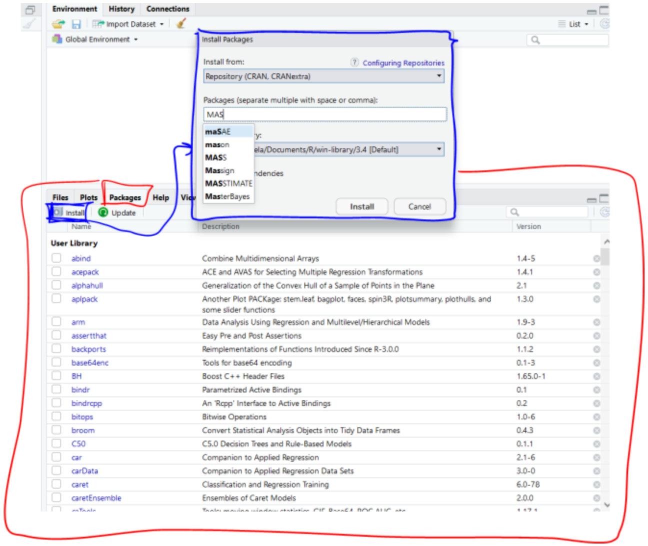
`install.packages('dplyr')`
Download and install a package from CRAN.

`library(dplyr)`
Load the package into the session, making all its functions available to use.

`dplyr::select`
Use a particular function from a package.

`data(iris)`
Load a built-in dataset into the environment.

<u>Rstudio interface</u>



Install Packages

Install from: Configuring Repositories
Repository (CRAN, CRANextra)

Packages (separate multiple with space or comma):
MAS

maSAE
mason
MASS
Massign
MASSTIMATE
MasterBayes

ela/Documents/R/win-library/3.4 [Default]

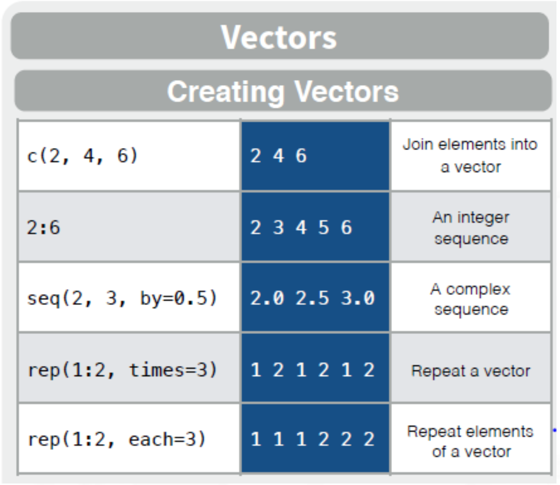
Install | Cancel

	Name	Description	Version
	User Library		
☐	abind	Combine Multidimensional Arrays	1.4-5
☐	acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
☐	alphahull	Generalization of the Convex Hull of a Sample of Points in the Plane	2.1
☐	aplpack	Another Plot PACKage: stem.leaf, bagplot, faces, spin3R, plotsummary, plothulls, and some slider functions	1.3.0
☐	arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.9-3
☐	assertthat	Easy Pre and Post Assertions	0.2.0
☐	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.2
☐	base64enc	Tools for base64 encoding	0.1-3
☐	BH	Boost C++ Header Files	1.65.0-1
☐	bindr	Parametrized Active Bindings	0.1
☐	bindrcpp	An 'Rcpp' Interface to Active Bindings	0.2
☐	bitops	Bitwise Operations	1.0-6
☐	broom	Convert Statistical Analysis Objects into Tidy Data Frames	0.4.3
☐	C50	C5.0 Decision Trees and Rule-Based Models	0.1.1
☐	car	Companion to Applied Regression	2.1-6
☐	carData	Companion to Applied Regression Data Sets	3.0-0
☐	caret	Classification and Regression Training	6.0-78
☐	caretEnsemble	Ensembles of Caret Models	2.0.0

NOTE: Package *swirl* is a good tutorial for learning R.

Base R

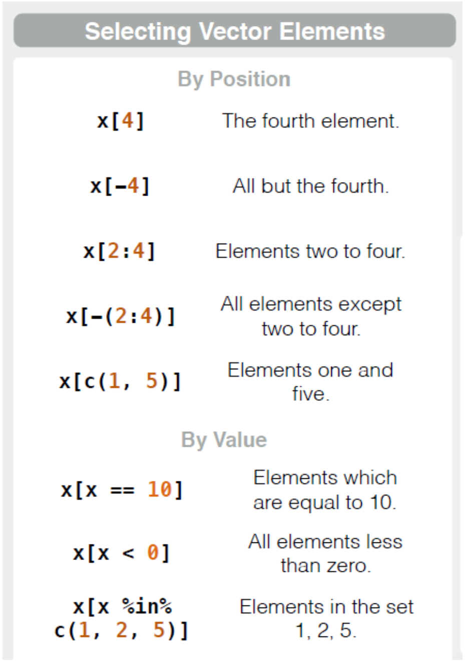
Vectors and indexing



Vectors

Creating Vectors

`c(2, 4, 6)`	`2 4 6`	Join elements into a vector
`2:6`	`2 3 4 5 6`	An integer sequence
`seq(2, 3, by=0.5)`	`2.0 2.5 3.0`	A complex sequence
`rep(1:2, times=3)`	`1 2 1 2 1 2`	Repeat a vector
`rep(1:2, each=3)`	`1 1 1 2 2 2`	Repeat elements of a vector



Selecting Vector Elements

By Position

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

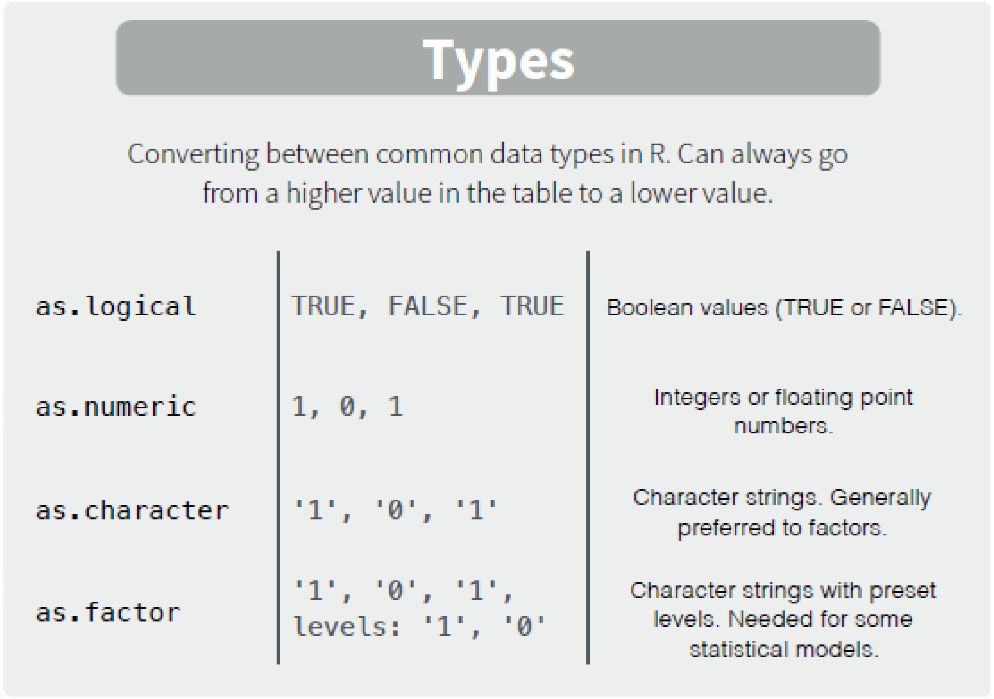
By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

You can also *combine* vectors and numbers with c()

Base R

Data types



Types

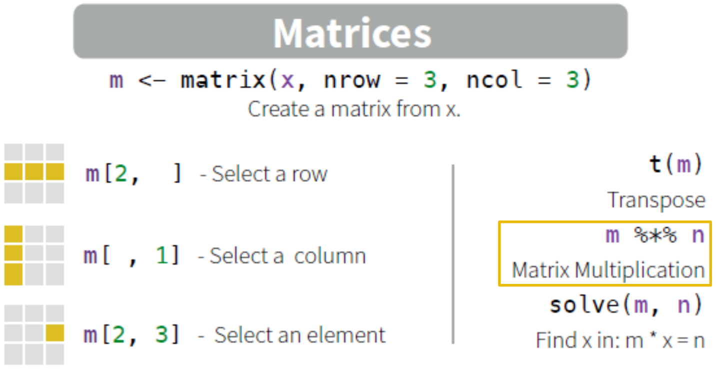
Converting between common data types in R. Can always go from a higher value in the table to a lower value.

`as.logical`	`TRUE, FALSE, TRUE`	Boolean values (TRUE or FALSE).
`as.numeric`	`1, 0, 1`	Integers or floating point numbers.
`as.character`	`'1', '0', '1'`	Character strings. Generally preferred to factors.
`as.factor`	`'1', '0', '1', levels: '1', '0'`	Character strings with preset levels. Needed for some statistical models.

NOTE: The dot . in R acts as another character for naming variables and functions

Base R

Matrices and lists

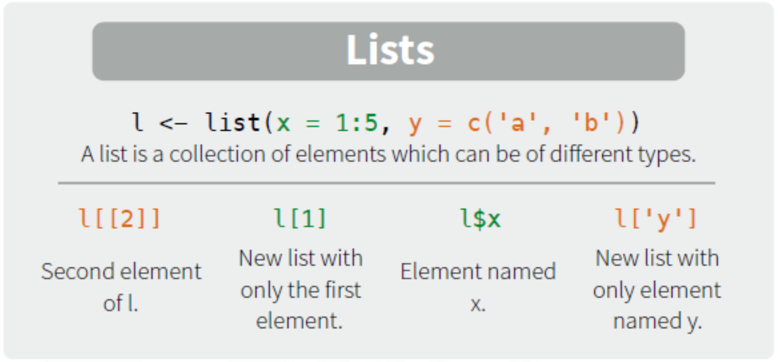


Matrices

```
m <- matrix(x, nrow = 3, ncol = 3)
```
Create a matrix from x.

`m[2, ]` - Select a row	`t(m)` Transpose
`m[ , 1]` - Select a column	`m %*% n` Matrix Multiplication
`m[2, 3]` - Select an element	`solve(m, n)` Find x in: m * x = n

Operator * multiplies element by element



Lists

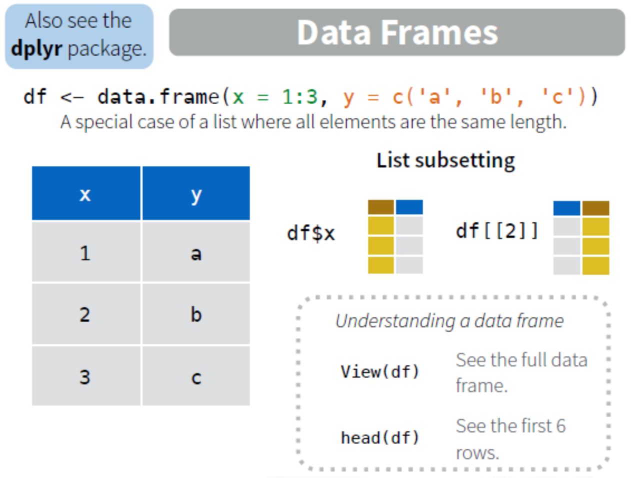
```
l <- list(x = 1:5, y = c('a', 'b'))
```
A list is a collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Base R

Data Frames

- Data Frames are the most common type of variable used.



Also see the **dplyr** package.

Data Frames

```r
df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))
```

A special case of a list where all elements are the same length.

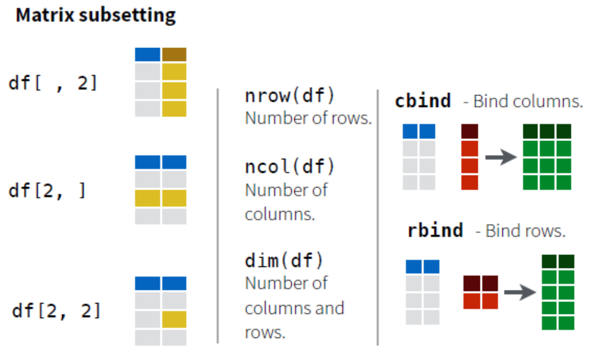
x	y
1	a
2	b
3	c

List subsetting

`df$x` `df[[2]]`

Understanding a data frame

`View(df)`	See the full data frame.
`head(df)`	See the first 6 rows.



Matrix subsetting

`df[ , 2]`

`df[2, ]`

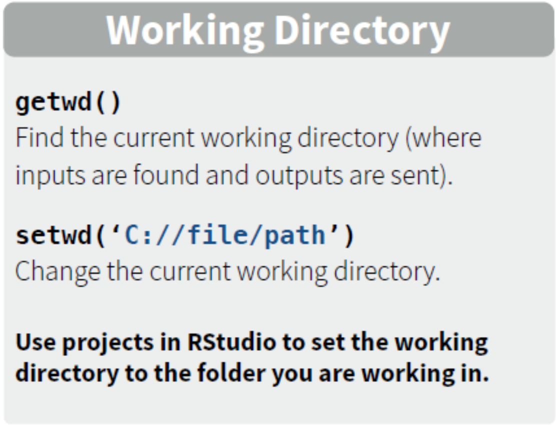
`df[2, 2]`

`nrow(df)`	Number of rows.
`ncol(df)`	Number of columns.
`dim(df)`	Number of columns and rows.

`cbind` - Bind columns.

`rbind` - Bind rows.

Base R
Working directory



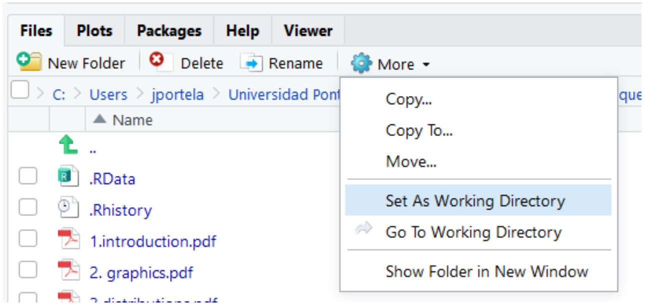
Working Directory

`getwd()`
Find the current working directory (where inputs are found and outputs are sent).

`setwd('C://file/path')`
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Rstudio interface

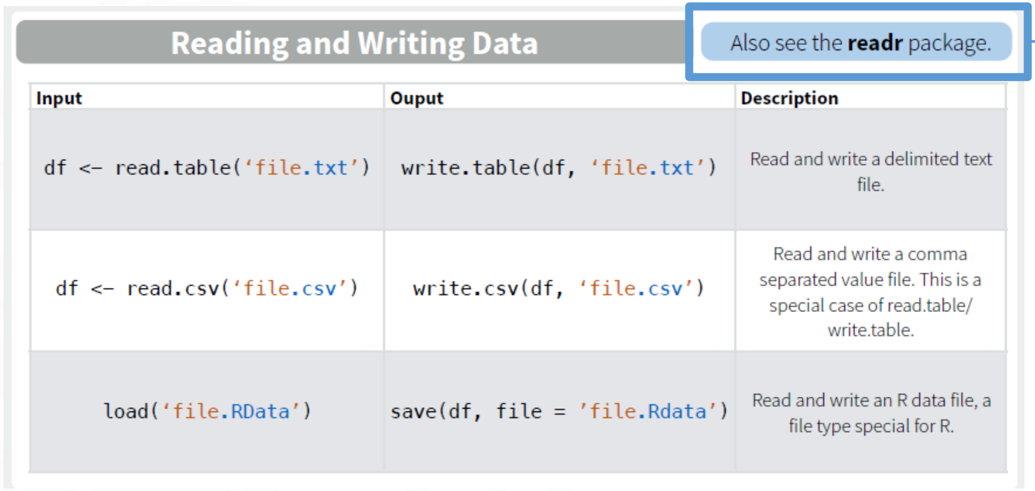


¡Important for loading files!

Base R

Reading and writing data

- Reading functions return a data frame object containing the data.



Reading and Writing Data

Also see the **readr** package.

Input	Ouput	Description
`df <- read.table('file.txt')`	`write.table(df, 'file.txt')`	Read and write a delimited text file.
`df <- read.csv('file.csv')`	`write.csv(df, 'file.csv')`	Read and write a comma separated value file. This is a special case of read.table/ write.table.
`load('file.RData')`	`save(df, file = 'file.Rdata')`	Read and write an R data file, a file type special for R.

readr also allows reading Excel files

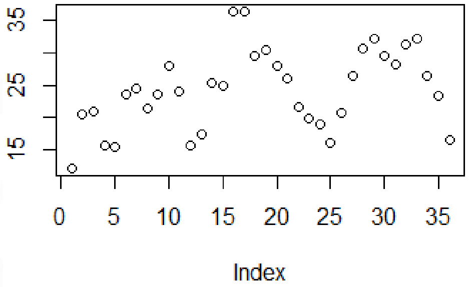
comillas.edu

Base R

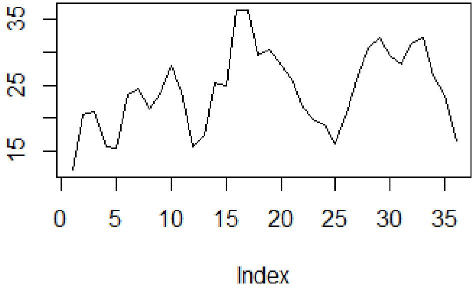
Basic plotting

- Dotplot and lines

`plot(x)`

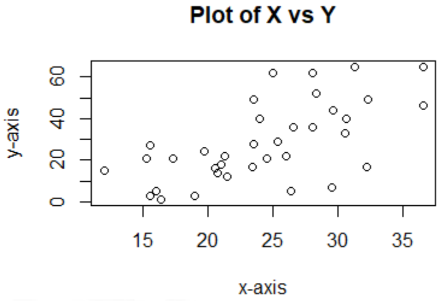


`plot(x, type = "l")`



- Scatterplots

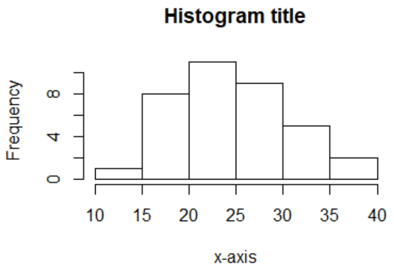
```
plot(x,y, xlab="x-axis", ylab="y-axis", main="Plot of X vs Y")
```



Base R
Basic plotting

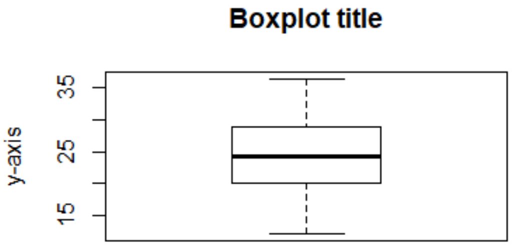
- Histograms

```
hist(x, main = "Histogram title",  xlab = "x-axis")
```



- Boxplots

```
boxplot(x, main="Boxplot title",  ylab="y-axis")
```



Base R

Useful hints

- Hit Ctrl+Enter to evaluate line of code or area selected from an R script.
- Remember to always set the working directory to the folder you are working in.
- Comments in the code with #
- Arguments passed to functions do not need to be in order. They can be passed naming the argument.

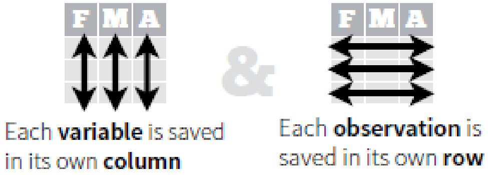
```
rep(1:2, times=3)
```

```
rep(1:2, each=3)
```

Preparation of data for machine learning

Data structure for machine learning

- Machine learning consists in extracting information from observed data. The data should be ordered in a meaningful way to perform analysis.
- We will work with data frames (tables) where:


F M A
&
F M A
Each **variable** is saved in its own **column**
Each **observation** is saved in its own **row**

- Useful tools for data wrangling:
  - *aggregate* and *rapply* functions.
  - *tidyr* and *dplyr* libraries.
- Type of variables:
  - Numeric: Continuous or discrete data.
  - Factors: Categorical variables.
  - Char: Variables containing text.

R is a high level language and many functions "interpret" what the user wants.
This is very useful, but can lead to mistakes.

Hints:
- Identify each variable and set the appropriate type.
- Identify NA values that can contaminate the analysis.

Exploratory analysis

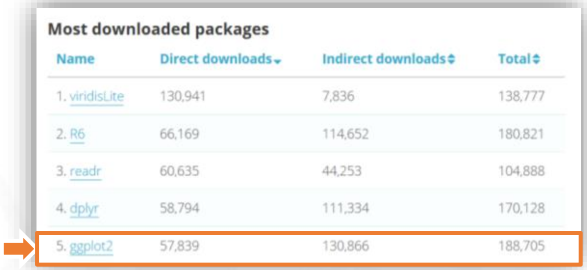
- Exploratory analysis of data is essential in machine learning.

- Objectives:
  - Identify outliers.
  - Relations between input and output variables.

- Some tools available:
  - Summary functions (mean, variance, quantiles…).
  - Plotting
    - Plot(), coplot(), pairs(), ...
    - Lattice library.
    - ggplot2 library.

ggplot2 library
Introduction

- Why ggplot2? → Top downloaded R packages



Most downloaded packages

Name	Direct downloads	Indirect downloads	Total
1. viridisLite	130,941	7,836	138,777
2. R6	66,169	114,652	180,821
3. readr	60,635	44,253	104,888
4. dplyr	58,794	111,334	170,128
5. ggplot2	57,839	130,866	188,705

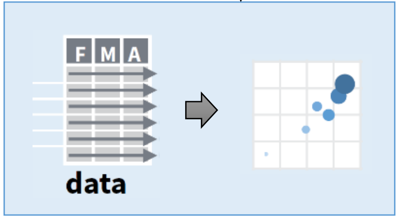
Source: https://www.rdocumentation.org/trends. Visited 29/10/2017

- ggplot2 implements the **grammar of graphics**, a coherent system for describing and building graphs.
- Some references:
  - Documentation: http://ggplot2.tidyverse.org/index.html
  - Introduction: http://r4ds.had.co.nz/data-visualisation.html
  - Book: H. Wickham (2016). *ggplot2: Elegant Graphics for Data Analysis*. 2nd Ed. Springer
  - Cheatsheet: https://github.com/rstudio/cheatsheets/raw/master/data-visualization-2.1.pdf
  - Top 50 visualizations: http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html

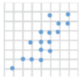
ggplot2 library
Basics

- **ggplot2** is based on the idea that you can build every graph from the same components: a **data set**, a **coordinate system**, and **geoms** (layers of graphics).
- The visual properties of the geoms are called **aesthetics.**
- Command example:

```
ggplot(data) + geom_point( aes(x = F, y = A, color = F, size = A))
```



- Description of syntax:
  - `ggplot(data)` → Begin a ggplot graphics using data as the dataset.
  - `geom_point()` → geom function to add a points to the graph.
    This function admits several aesthetic properties:



e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

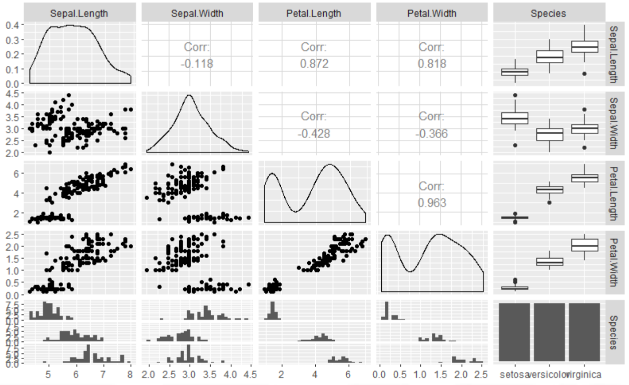
  - `aes()` → Function to specify aesthetic properties of the points depending on the values of variables:
    - `x = F` → The x coordinates of the points are given by variable F of data.
    - `y = A` → The y coordinates of the points are given by variable A of data.
    - `color = F` → The color of the points are given by the values of variable F of data.
    - `size = A` → The size of the points are given by the values of variable A of data.
- Open Example_ggplot.R for more details.

ggplot2 library

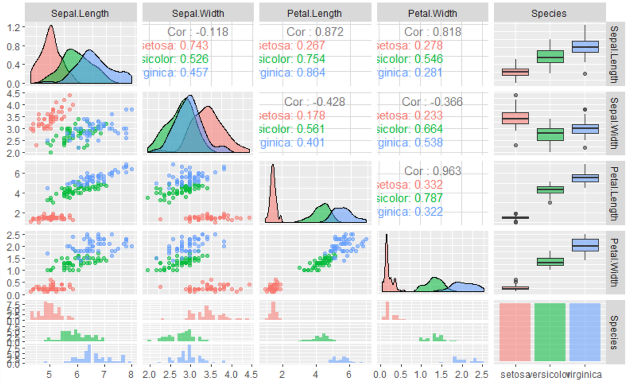
ggpairs

- `ggpairs()` function provides an extension of `pairs()` following ggplot philosophy.
- It is located in GGally package.
- Example with iris dataset:
  - Compute ggpairs plot for *iris* dataset.
  - Colour each plot in function of Species variable.

```
ggpairs(iris)
```



```
ggpairs(iris,
    aes(color = Species, alpha = 0.4))
```



Control flow

Conditionals

In R there are three main control flow tools:

- `if()`, `else if()` and `else`: standard conditionals
  - Condition in `if()` needs to give one `TRUE` or `FALSE` value
  - Note that `else` statement is optional
  - Single line actions don't need braces: `if(x>=0) x else -x`
  - `else if()` can be arbitrarily used many times
- `ifelse()`: conditional function that vectorizes nicely
  - Use this when a conditional shall be applied to each member of a vector: ifelse(vector < value_to_compare, value_if_true, value_if_false)
- `switch()`: handy for deciding between several options instead of using nested `else if()`, but almost never used.

Remember to use `&&` and `||` to evaluate several conditionals in an `if` clause and & and | for indexing.

for loop

A for() loop increments a counter variable along a vector. It repeatedly runs a code block, called the body of the loop, with the counter set at its current value, until it runs through the vector:

```
n = 10
log.vec = vector(length=n, mode="numeric")
for (i in 1:n) {
  log.vec[i] = log(i)
}
log.vec
```

```
##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595
1.9459101
##  [8] 2.0794415 2.1972246 2.3025851
```

In the example, `i` is the counter and the vector we are iterating over is `1:n`.

Breaking the for loop

To avoid executing the whole `for()` loop, we can break out of it using `break`

```
n = 10
log.vec = vector(length=n, mode="numeric")
for (i in 1:n) {
  if (log(i) > 2 {
    break
  }
  log.vec[i] = log(i)
}
log.vec
```

```
##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595
1.9459101
##  [8] 0.0000000 0.0000000 0.0000000
```

Iteration

Variations of for loops

Many different variations on standard `for()` are possible. Two common ones:

- Nonnumeric counters: counter variable always gets iterated over a vector, but it doesn't have to be numeric
- Nested loops: body of the `for()` loop can contain another `for()` loop (or several others)

```
for (str in c("Prof", "Ryan", "Tibs")) {
  cat(paste(str, "declined to comment\n"))
}
## Prof declined to comment
## Ryan declined to comment
## Tibs declined to comment

for (i in 1:4) {
  for (j in 1:i^2) {
    cat(paste(j,""))
  }
  cat("\n")
}
## 1
## 1 2 3 4
## 1 2 3 4 5 6 7 8 9
```

Iteration

while loop

A `while()` loop repeatedly runs a code block, again called the body, until some condition is no longer true.

```
i = 1
log.vec = c()
while (log(i) <= 2) {
  log.vec = c(log.vec, log(i))
  i = i+1
}
log.vec
## [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
```

`Repeat` can be used if we want an infinite loop as in `while(TRUE)`:

```
repeat {
  ans = readline("Who is the best Professor of Statistics at ICAI? ")
  if (ans == “Jaime Pizarroso" || ans == “Jaime") {
    cat("Yes! You get a 10.")
    break
  }
  else {
    cat("Wrong answer!\n")
  }
}
```

Iteration

for() versus while()

- `for()` is better when the number of times to repeat (values to iterate over) is clear in advance
- `while()` is better when you can recognize when to stop once you're there, even if you can't guess it to begin with
- `while()` is more general, in that every `for()` could be replaced with a `while()` (but not vice versa)
- WARNING: some people have a tendency to overuse `for` and `while` loops. They are not always needed, remember vectorization should be used whenever possible.

Iteration

apply()

R offers a family of apply functions, which allow you to apply a function across different chunks of data. Offers an alternative to explicit iteration using for() loop; can be simpler and faster, though not always. Summary of functions:

- `apply()`: apply a function to any dimension of a matrix, data.frame or array
- `lapply()`: apply a function to elements of a list or vector
- `sapply()`: same as the above, but simplify the output (if possible)
- `tapply()`: apply a function to levels of a factor vector

Iteration

apply()

The `apply()` function takes inputs of the following form:

- `apply(x, MARGIN=1, FUN=my.fun)`, to apply `my.fun()` across dimension n of object x

For example:

```
apply(state.x77, MARGIN=2, FUN=function(v) {
  q1 = quantile(v, prob=0.1)
  q2 = quantile(v, prob=0.9)
  return(mean(v[q1 <= v & v <= q2]))
})
##   Population      Income  Illiteracy    Life Exp      Murder   HS Grad
##   3384.27500  4430.07500     1.07381    70.91775      7.2975  53.33750
##        Frost        Area
##    104.68293 56575.72500
```

Iteration

apply()

If any more arguments shall be passed to the FUN argument, the named arguments shall be passed after the FUN argument:

- `apply(x, MARGIN=1, FUN=my.fun)`, to apply `my.fun()` across dimension n of object x

For example:

```
apply(state.x77, MARGIN=2, FUN=function(v, p1, p2) {
  q1 = quantile(v, prob=p1)
  q2 = quantile(v, prob=p2)
  return(mean(v[q1 <= v & v <= q2]))
}, p1=0.01, p2=0.99)
##   Population      Income  Illiteracy    Life Exp     Murder    HS Grad
##     3974.125   4424.5208      1.1367   70.882708   7.341667   53.13125
##        Frost        Area
##   104.895833  61860.6875
```

Iteration

apply()

The type of data that `apply()` would return depends on the function passed to the `FUN` argument:

- If `my.fun()` returns a single value, then `apply()` will return a vector
- If `my.fun()` returns k values, then `apply()` will return a matrix with k rows (note: this is true regardless of whether `MARGIN=1` or `MARGIN=2`)
- If `my.fun()` returns different length outputs for different inputs, then `apply()` will return a list
- If `my.fun()` returns a list, then `apply()` will return a list

Do not overuse the apply paradigm, there are some special functions optimized that are simpler and faster:

- `rowSums()`, `colSums()`: for computing row, column sums of a matrix
- `rowMeans()`, `colMeans()`: for computing row, column means of a matrix
- `max.col()`: for finding the maximum position in each row of a matrix

Combining these functions with logical indexing and vectorized operations will enable you to do quite a lot.

Iteration

lapply(), sapply() and tapply()

- The `lapply()` function takes inputs as in: `lapply(x, FUN=my.fun)`, to apply `my.fun()` across elements of a list or vector `x`. The output is always a list.

```
lapply(my.list, FUN=mean) # Returns a list
```

- The `sapply()` function works just like `lapply()`, but tries to simplify the return value whenever possible. E.g., most common is the conversion from a list to a vector

```
sapply(my.list, FUN=mean) # Returns a vector
```

- The function `tapply()` takes inputs as in: `tapply(x, INDEX=my.index, FUN=my.fun)`, to apply `my.fun()` to subsets of entries in x that share a common level in `my.index`

```
# Compute the mean and sd of the Frost variable, within each region
tapply(state.x77[,"Frost"], INDEX=state.region, FUN=mean)
##      Northeast         South North Central          West
##       132.7778       64.6250      138.8333      102.1538
```

Iteration
split()

- The function `split()` split up the rows of a data frame by levels of a factor, as in: `split(x, f=my.index)` to split a data.frame x according to levels of `my.index`. It returns a list containing each subset of the data.frame x. Each subset of the list is named after the level it was separated. A combination of `split()` and `lapply()` functions can be used to apply a custom function to a data.frame based on the level of a factor variable.

```
# Split up the state.x77 matrix according to region
state.by.reg = split(data.frame(state.x77), f=state.region)
class(state.by.reg) # The result is a list
## [1] "list"
names(state.by.reg) # This has 4 elements for the 4 regions
## [1] "Northeast"     "South"         "North Central" "West"
class(state.by.reg[[1]]) # Each element is a data frame
## [1] "data.frame“
# For each region, display the first 3 rows of the data frame
lapply(state.by.reg, FUN=head, 3)
```

Functions
Why?

- Data structures tie related values into one object
- Functions tie related commands into one object
- In both cases: easier to understand, easier to work with, easier to build into larger things

The structure of a function has three basic parts:

- Inputs (or argumets)
- Body (code that is executed)
- Output (or return value)

R does not let your function have multiple outputs, but a list containing multiple objects can be returned

Functions

Creating your own function

Call `function()` to create your own function. Document your function with comments using #

```r
# get.wordtab.king: get a word table from King's "I Have A Dream" speech
# Input: none
# Output: word table, i.e., vector with counts as entries and associated
#   words as names
get.wordtab.king = function() {
  lines = readLines("http://www.stat.cmu.edu/statcomp/data/king.txt")
  text = paste(lines, collapse=" ")
  words = strsplit(text, split="[[:space:]]|[[:punct:]]")[[1]]
  words = words[words != ""]
  wordtab = table(words)
  return(wordtab)
}
```

Creating your own function

Much better: create a word table function that takes a URL of web

```
# get.wordtab.king: get a word table from King's "I Have A Dream" speech
# Input:
# - str.url: string, specifying URL of a web page
# Output: word table, i.e., vector with counts as entries and associated
#   words as names
get.wordtab.from.url = function(str.url) {
  lines = readLines(str.url)
  text = paste(lines, collapse=" ")
  words = strsplit(text, split="[[:space:]]|[[:punct:]]")[[1]]
  words = words[words != ""]
  wordtab = table(words)
  return(wordtab)
}
```

Default return value

With no explicit `return()` statement, the default is just to return whatever is on the last line. So the following is equivalent to the previous slide:

```
# get.wordtab.king: get a word table from King's "I Have A Dream" speech
# Input:
# - str.url: string, specifying URL of a web page
# Output: word table, i.e., vector with counts as entries and associated
#   words as names
get.wordtab.from.url = function(str.url) {
  lines = readLines(str.url)
  text = paste(lines, collapse=" ")
  words = strsplit(text, split="[[:space:]]|[[:punct:]]")[[1]]
  words = words[words != ""]
  table(words)
}
```

Functions

Multiple inputs

```
# get.wordtab.from.url: get a word table from text on the web
# Inputs:
# - str.url: string, specifying URL of a web page
# - split: string, specifying what to split on
# Output: word table, i.e., vector with counts as entries and associated
#   words as names
get.wordtab.from.url = function(str.url, split) {
  lines = readLines(str.url)
  text = paste(lines, collapse=" ")
  words = strsplit(text, split=split)[[1]]
  words = words[words != ""]
  table(words)
}
```

Default inputs value

A function can also specify default values for the inputs (if the user doesn't specify an input in the function call, then the default value is used)

```
# get.wordtab.from.url: get a word table from text on the web
# Inputs:
# - str.url: string, specifying URL of a web page
# - split: string, specifying what to split on. Default is the regex pattern
#   "[[:space:]]|[[:punct:]]"
# - tolower: Boolean, TRUE if words should be converted to lower case before
#   the word table is computed. Default is TRUE
# Output: word table, i.e., vector with counts as entries and associated
#   words as names
get.wordtab.from.url = function(str.url, split="[[:space:]]|[[:punct:]]", tolower=TRUE) {
  lines = readLines(str.url)
  text = paste(lines, collapse=" ")
  words = strsplit(text, split=split)[[1]]
  words = words[words != ""]
  # Convert to lower case, if we're asked to
  if (tolower) words = tolower(words)
  table(words)
}
```

Functions

Examples of function calls

```
# Inputs can be called by name, or without names
king.wordtab1 = get.wordtab.from.url(
  str.url="http://www.stat.cmu.edu/~ryantibs/statcomp/data/king.txt",
  split="[[:space:]]|[[:punct:]]", tolower=TRUE)
king.wordtab2 = get.wordtab.from.url(
  "http://www.stat.cmu.edu/~ryantibs/statcomp/data/king.txt",
  "[[:space:]]|[[:punct:]]", TRUE)
# Inputs can be called by partial names (if uniquely identifying)
king.wordtab3 = get.wordtab.from.url(
  str="http://www.stat.cmu.edu/~ryantibs/statcomp/data/king.txt",
  spl="[[:space:]]|[[:punct:]]", tolower=TRUE)
# When inputs aren't specified, default values are used
king.wordtab4 = get.wordtab.from.url(
  str.url="http://www.stat.cmu.edu/~ryantibs/statcomp/data/king.txt",
  split="[[:space:]]|[[:punct:]]")
```

Examples of function calls

```
# Named inputs can go in any order
king.wordtab5 = get.wordtab.from.url(
  tolower=TRUE, split="[[:space:]]|[[:punct:]]",
  str.url="http://www.stat.cmu.edu/~ryantibs/statcomp/data/king.txt")
# Unnamed inputs cannot go in any order
king.wordtab6 = get.wordtab.from.url("[[:space:]]|[[:punct:]]",
  "http://www.stat.cmu.edu/~ryantibs/statcomp/data/king.txt",
  tolower=FALSE)
## Error in file(con, "r"): cannot open the connection
```

Functions

Return multiple values

```
get.wordtab.from.url = function(str.url,
split="[[:space:]]|[[:punct:]]", tolower=TRUE, keep.nums=FALSE) {
  lines = readLines(str.url)
  text = paste(lines, collapse=" ")
  words = strsplit(text, split=split)[[1]]
  words = words[words != ""]
  # Convert to lower case, if we're asked to
  if (tolower) words = tolower(words)
  # Get rid of words with numbers, if we're asked to
  if (!keep.nums) words = grep("[0-9]", words, inv=TRUE, val=TRUE)
  # Compute the word table
  wordtab = table(words)
  return(list(wordtab=wordtab,
              number.unique.words=length(wordtab),
              number.total.words=sum(wordtab),
              longest.word=words[which.max(nchar(words))]))
}
```

Functions
Side effects and evironments

A side effect of a function is something that happens as a result of the function's body, but is not returned. Examples:

- Printing something out to the console
- Plotting something on the display
- Saving an R data file, or a PDF, etc.

An environment in R can be thought of as a collection of objects (functions, variables, etc.)

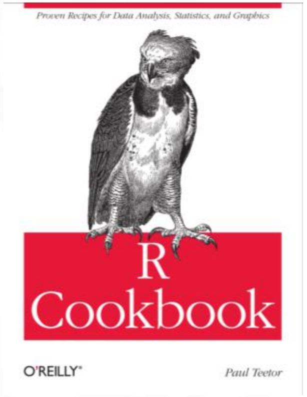
- Each function has its own environment
- Names here override names in the global environment
- Internal environment starts with the named arguments
- Assignments inside the function only change the internal environment
- Names undefined in the function are looked for in the global environment. However, this is not advisable unless you are using built-in constants like `pi`, `letters`, etc.

Top-down function design

1. Start with the big-picture view of the task
2. Break the task into a few big parts
3. Figure out how to fit the parts together
4. Repeat this for each part

Bibliography

- P. Teetor(2011). R Cookbook. O'Reilly
- W. Chang (2012). *R Graphics Cookbook*. O'Reilly.
- D. Teutonico (2015). *ggplot2 Essentials*. Packt Publishing.
- G. James, D. Witten, T. Hastie & R. Tibshirani (2013). An Introduction to Statistical Learning with Applications in R. Springer (see http://www-bcf.usc.edu/~gareth/ISL/)



Bibliography

- H. Wickham (2016). *ggplot2: Elegant Graphics for Data Analysis*. 2nd Ed. Springer.
- W. Chang (2012). *R Graphics Cookbook*. O'Reilly.
- D. Teutonico (2015). *ggplot2 Essentials*. Packt Publishing.
- L. Wilkinson (2003). *The Grammar of Graphics* (Statistics and Computing). 2nd Ed. Springer.
- R. Tibshirani (2021). *Statistical Computing Course: Fall 2001*. URL: http://www.stat.cmu.edu/~ryantibs/statcomp/

comillas.edu

Alberto Aguilera 23, E-28015 Madrid - Tel: +34 91 542 2800 - http://www.iit.comillas.edu