

R COURSE

VITOR SOUSA, LAURENT EXCOFFIER

Day 1 Introduction to R and R studio, basic syntax

Goals for first day

- Know what R can do
- Be comfortable with R studio interface and R in general
- Find relevant information with help
- Know some basic syntax and data types
- Manipulate vectors and matrices
- Be able to use R as a calculator

Some references

Online tutorials and courses

Swirl Intro tutorials

<http://swirlstats.com/>

Other online resources:

Butler, M.A. (2009) Getting started in R for Biologists

<http://www2.hawaii.edu/~mbutler/Rquickstart/simpleR.pdf>

Paradis, E. (2005) R for Beginners.

http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf

R Development Core Team: The R Manuals. <http://www.r-project.org/>

Venables, WN et al (2013) An Introduction to R

<http://cran.r-project.org/doc/manuals/R-intro.pdf>

Vernazzi, J. (2002) simpleR – Using R for introductory statistics

<http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf>

Yakir, B (2011) Introduction to Statistical Thinking (With R, Without Calculus)

<http://pluto.huji.ac.il/~msby/StatThink/>

More advanced resources

Burns, P. (2011) R inferno

http://www.burns-stat.com/pages/Tutor/R_inferno.pdf

Kabacoff, R. I. (2012) Quick R – Accessing the power of R

<http://www.statmethods.net/>

Schluter, D and Veen, T (2013) R workshops

<https://www.zoology.ubc.ca/~bio501/R/workshops/>

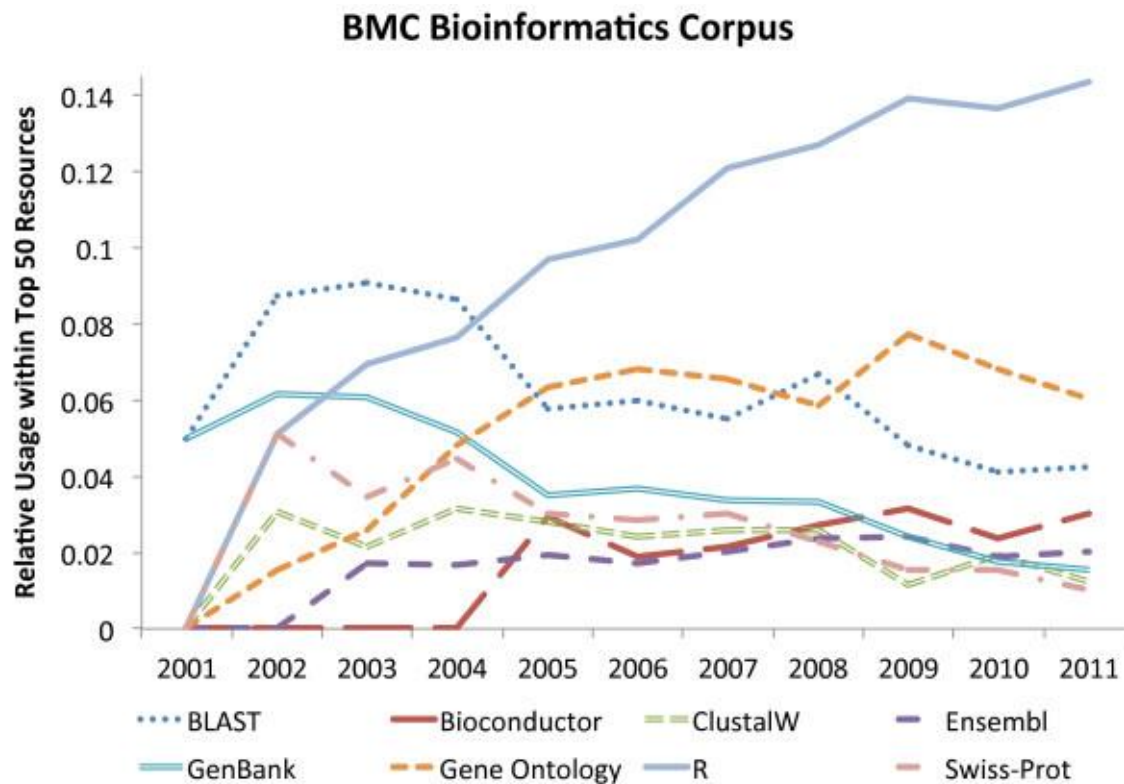
Why learning and using R?

- R is free, open source, updated regularly
- Good help system
- Excellent graphing capabilities
- Built-in powerful statistical analyses
- R can be extended by user, highly customizable
- R has a programming language
- R can handle large data sets (physics, biology,...)
- Invested work can be reused for later analyses (custom functions and scripts)
- R increasingly used in research
- Results are stored, not necessarily shown, and can be reused (plots ...)

R drawbacks

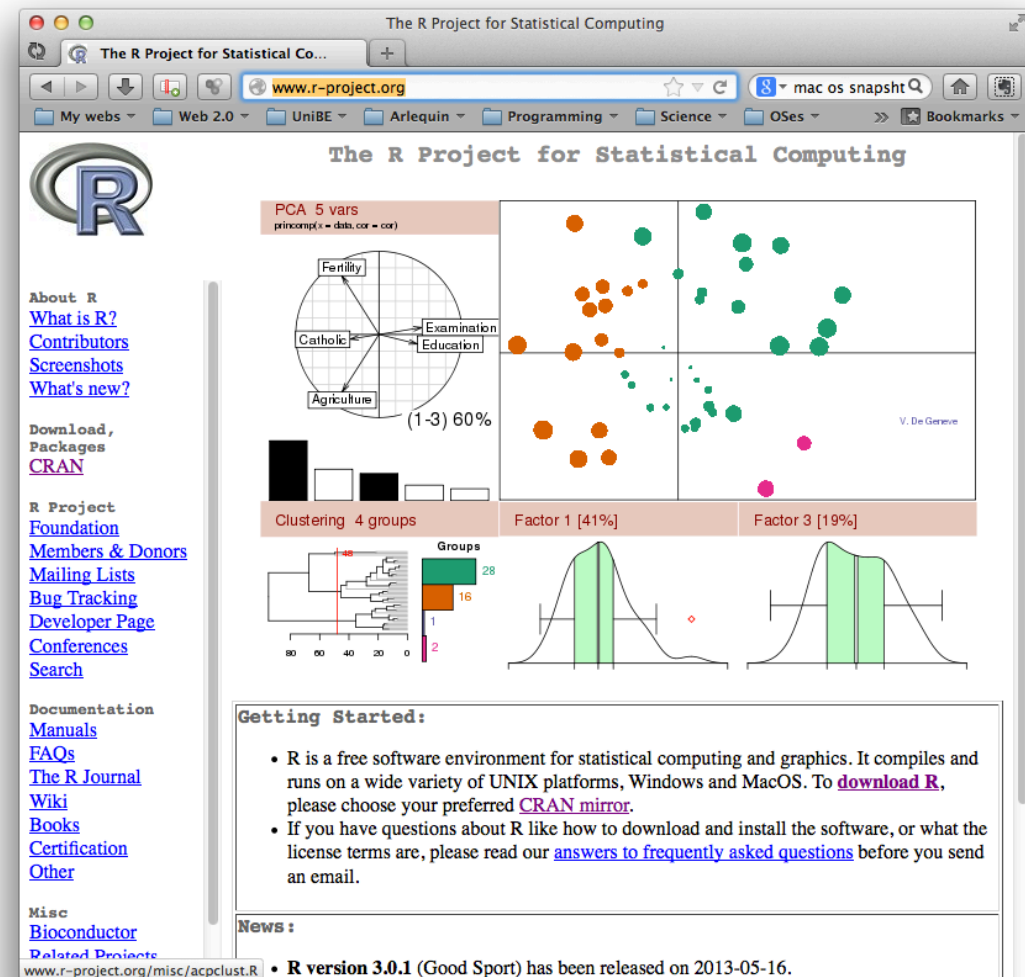
- Need to learn R syntax and programming language
- No commercial support
- No graphical interface

But...



R package

- <http://www.r-project.org/>
- Ver 3.5.1
- Downloads
- Information
- Examples
- Tutorials
- Help
- Graphics examples
- Books
- FAQs



RStudio: Powerful IDE for R

<http://www.rstudio.com/>



RStudio

D:/Users/Laurent/Dropbox/CMPG/Courses/R course 2013/material Lutz Dumbgen course 2012 - RStudio

File Edit Code View Plots Session Project Build Tools Help

Go to file/function

Monday1.R x Monday2.R x

Source on Save Run Source

```

5  ## (and some plots)  ##
6  #####
7
8
9  # There are various possibilities to create
10 # one-dimensional arrays (vectors):
11 #
12 # If x is an array, x[j] is its j-th component.
13 # length(x) is the number of its components.
14
15 # The simplest command to generate an array
16 # the command c(ombine)(...):
17
18 x <- c(1.8, 6, 7.5)
19 x
20
21 x[1]
22 x[2]
23 x[3]
24
25 length(x)
26
27 y <- c(x, 30, 1000)
28 y
29 y[3:5]
30
31 length(y)
32
33
34 # Creation of arrays with regular patterns.
35
36 # For integers a <= b,
37 # x <- a:b
38 # generates an array x with components
39 # a, a+1, a+2, ..., b
16:6 (Untitled)

```

Scripts & view files
you can type your commands here and save them into a file

Workspace History

Values

C	integer [51]
n	53
x	numeric [39901]
y	numeric [39901]
z	numeric [16]

Workspace
Shows all active objects in memory

Files Plots Packages Help

Zoom Export Clear All

A strange function

Plots
Shows plots

Console D:/Users/Laurent/Dropbox/CMPG/Courses/R course 2013/material Lutz Dumbgen course 2012/

```

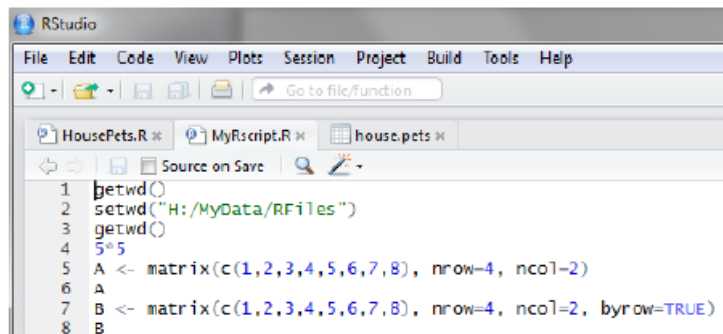
> x <- seq(0.001,1, ... [TRUNCATED]
> y <- x*sin(2*pi/x)
> plot(x,y,type="l")
> plot(x,y,type="l",
+ xlab="x",ylab="f(x)")
> # Trial 2:
> # Take a fine and non-uniform grid of points,
> # making sure that 1/x takes on many of the
> # special points 1, 2, 3, ... :
> x <- 1/ .... [TRUNCATED]
> y <- x*sin(2*pi/x)
> plot(x,y,type="l",
+ xlab="x",ylab="f(x)")
> plot(c(0,x),c(0,y),type="l",
+ xlab="x",ylab="f(x)",
+ main="A strange function")
>

```

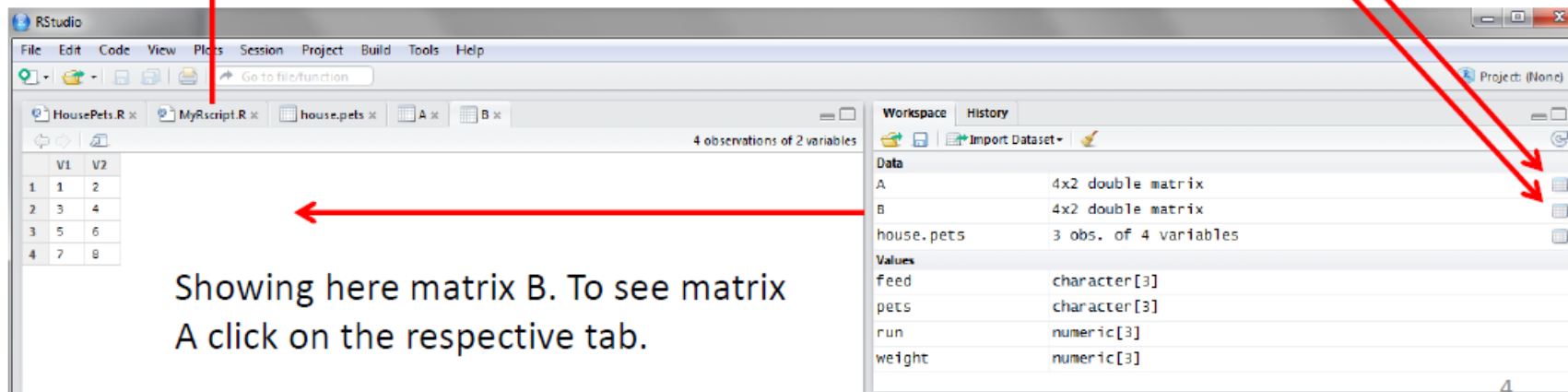
Console is where you run (or type) the commands and see the output

Workspace tab (1)

The workspace tab stores any object, value, function or anything you create during your R session. In the example below, if you click on the dotted squares you can see the data on a screen to the left.



```
1 getwd()
2 setwd("H:/MyData/RFiles")
3 getwd()
4 5*5
5 A <- matrix(c(1,2,3,4,5,6,7,8), nrow=4, ncol=2)
6 A
7 B <- matrix(c(1,2,3,4,5,6,7,8), nrow=4, ncol=2, byrow=TRUE)
8 B
```



The screenshot shows the RStudio interface with the Workspace tab selected. The Workspace pane on the right lists objects: A (4x2 double matrix), B (4x2 double matrix), and house.pets (3 obs. of 4 variables). Below these are 'Values' for feed, pets, run, and weight. On the left, a preview of matrix B is shown as a table with 4 rows and 2 columns (V1, V2).

	V1	V2
1	1	2
2	3	4
3	5	6
4	7	8

Showing here matrix B. To see matrix A click on the respective tab.

from Oscar Torres Reyna (Introduction to RStudio <http://dss.princeton.edu/training/>)

Why using RStudio?

- Excellent Integrated Development Interface
- Easy installation, platform independence (Mac, Windows, Linux)
- Finds latest version of R
- Command history
- Command completion (ctrl + space)
- Integrated help system
- Plot history
- Projects
- Easy installation of new packages and libraries
- Highly customizable interface (colours, windows)
- Data structures can be browsed in tabs

Getting help

- ?help
 - “?” General help on keyword
- help(help)
 - Like ?help
- help.search('plot')
- help.search("plot")
- ??plot
 - Help on topic by content



help {utils}

R Documentation

Documentation

Description

help is the primary interface to the help systems.

Usage

```
help(topic, package = NULL, lib.loc = NULL,  
      verbose = getOption("verbose"),  
      try.all.packages = getOption("help.try.all.packages"),  
      help_type = getOption("help_type"))
```

Arguments

topic	usually, a name or character string specifying the topic for which help is sought. A character string (enclosed in explicit single or double quotes) is always taken as naming a topic. If the value of topic is a length-one character vector the topic is taken to be the value of the only element. Otherwise topic must be a name or a reserved word (if syntactically valid) or character string. See 'Details' for what happens if this is omitted.
package	a name or character vector giving the packages to look into for

Getting help

- ?help
 - “?” General help on keyword
- help(help)
 - Like ?help
- help.search('plot')
- help.search("plot")
- ??plot
 - Help on topic by content
- Highlight and F1
- Ctrl+space
 - Help on syntax
- rseek.org
 - Google for R
- Google



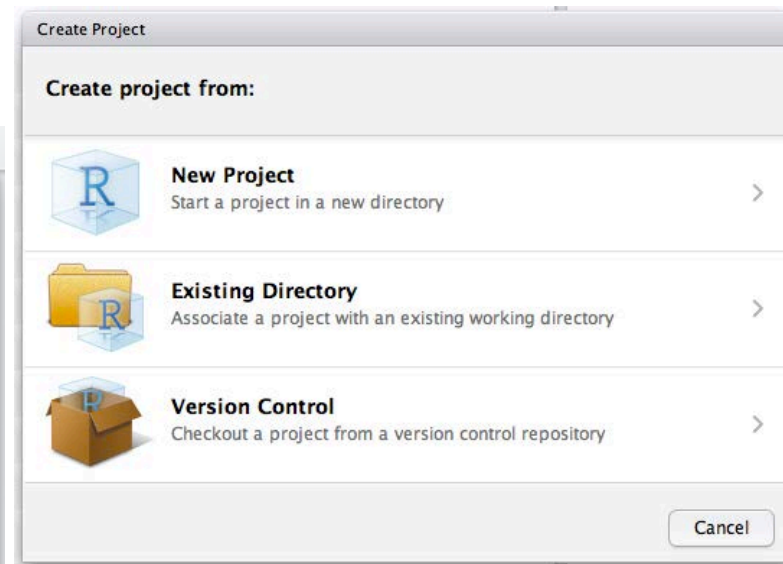
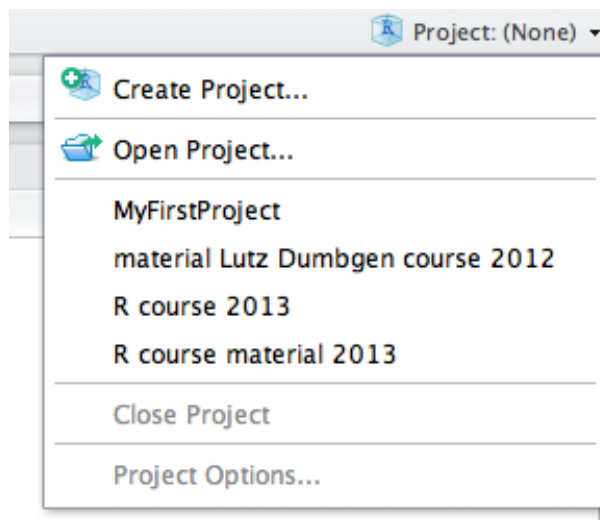
```
> help.request {utils} help.request(...)
> help.search {utils}
> help.start {utils} Prompts the user to check they have done all that is expected of
> them before sending a post to the R-help mailing list, provides a
> template for the post with session information included and
> optionally sends the email (on Unix systems).
f
U
<
< Press F1 for additional help
> help.
```

Creating an RStudio project

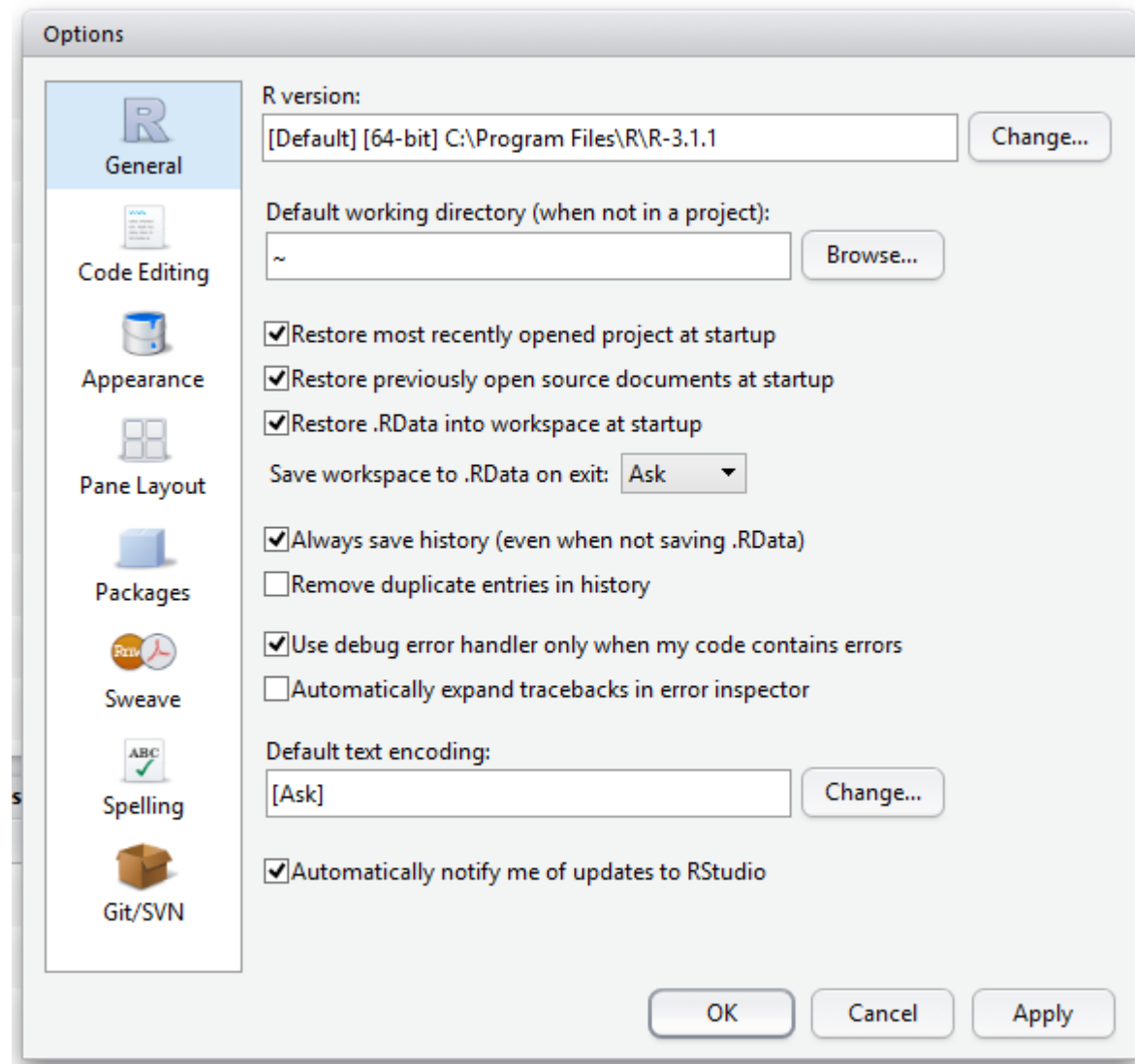
- Rstudio projects are stored in a new or existing directory
 - This is your working directory
- Keeps track of
 - your command history
 - your open scripts
 - your workspace (data structures, results of previous analyses)
- You can use the directory to store
 - R scripts
 - results
 - pdfs,
 - graphs,
 - tutorials, etc...

Creating an Rstudio project

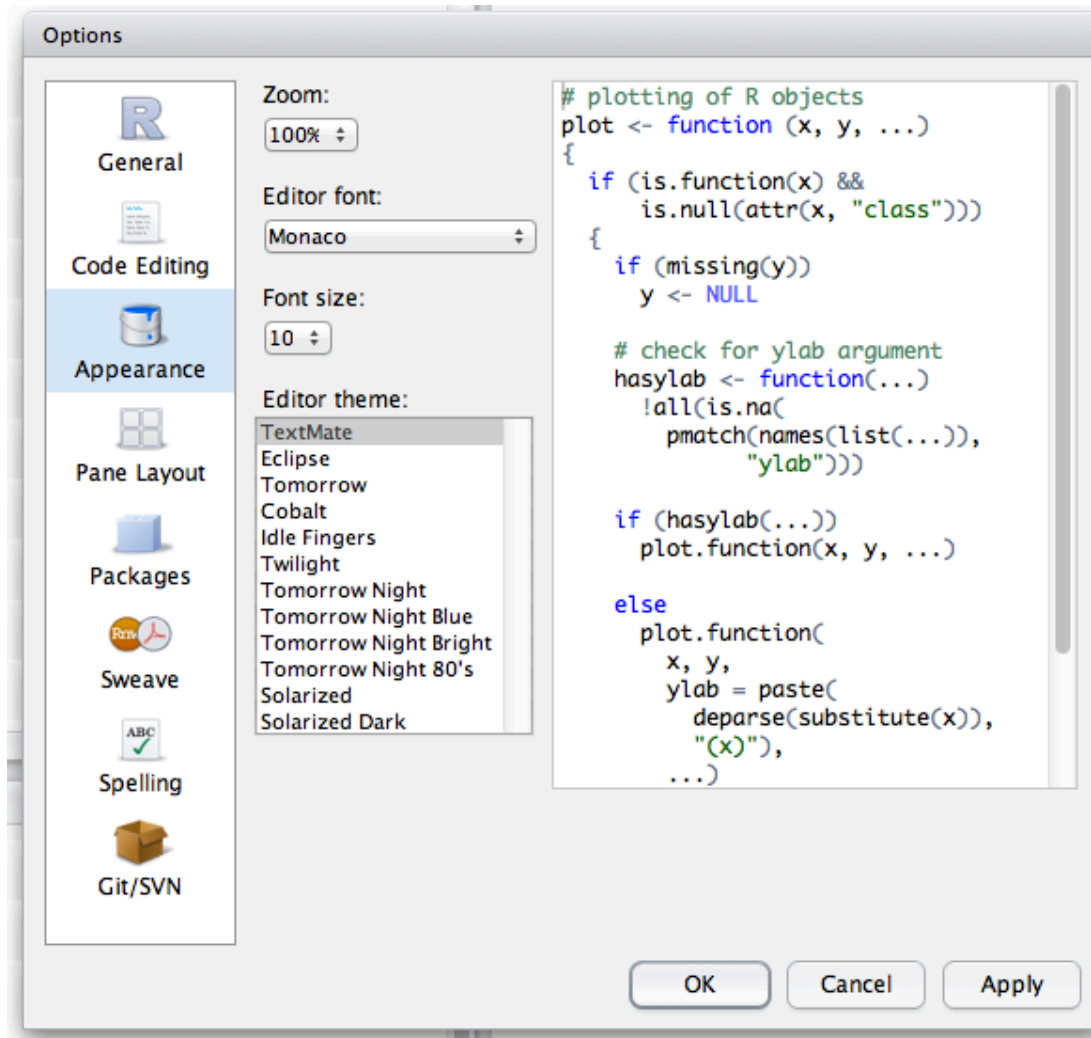
- Rstudio projects are stored in a new or existing directory
 - This is your **working directory** (where R is expecting your input files to be)
- Keeps track of
 - your command history
 - your open scripts
 - your workspace (data structures, results of previous analyses)
- You can use the directory to store
 - R scripts
 - results
 - pdfs
 - graphs
 - tutorials, etc...
- How to:
 - Go to upper right corner
 - Click on Project (none)
 - Create Project...



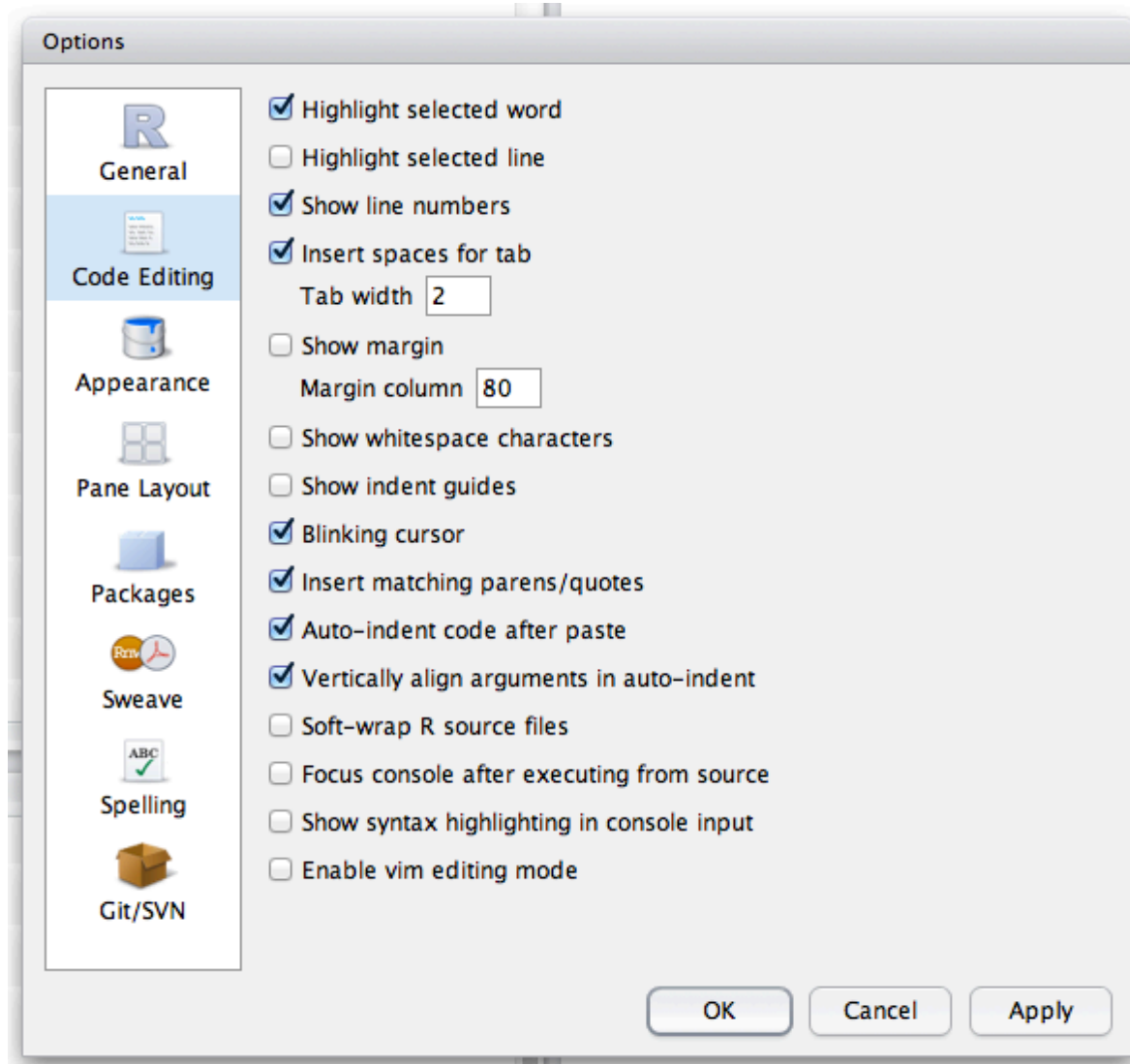
RStudio options



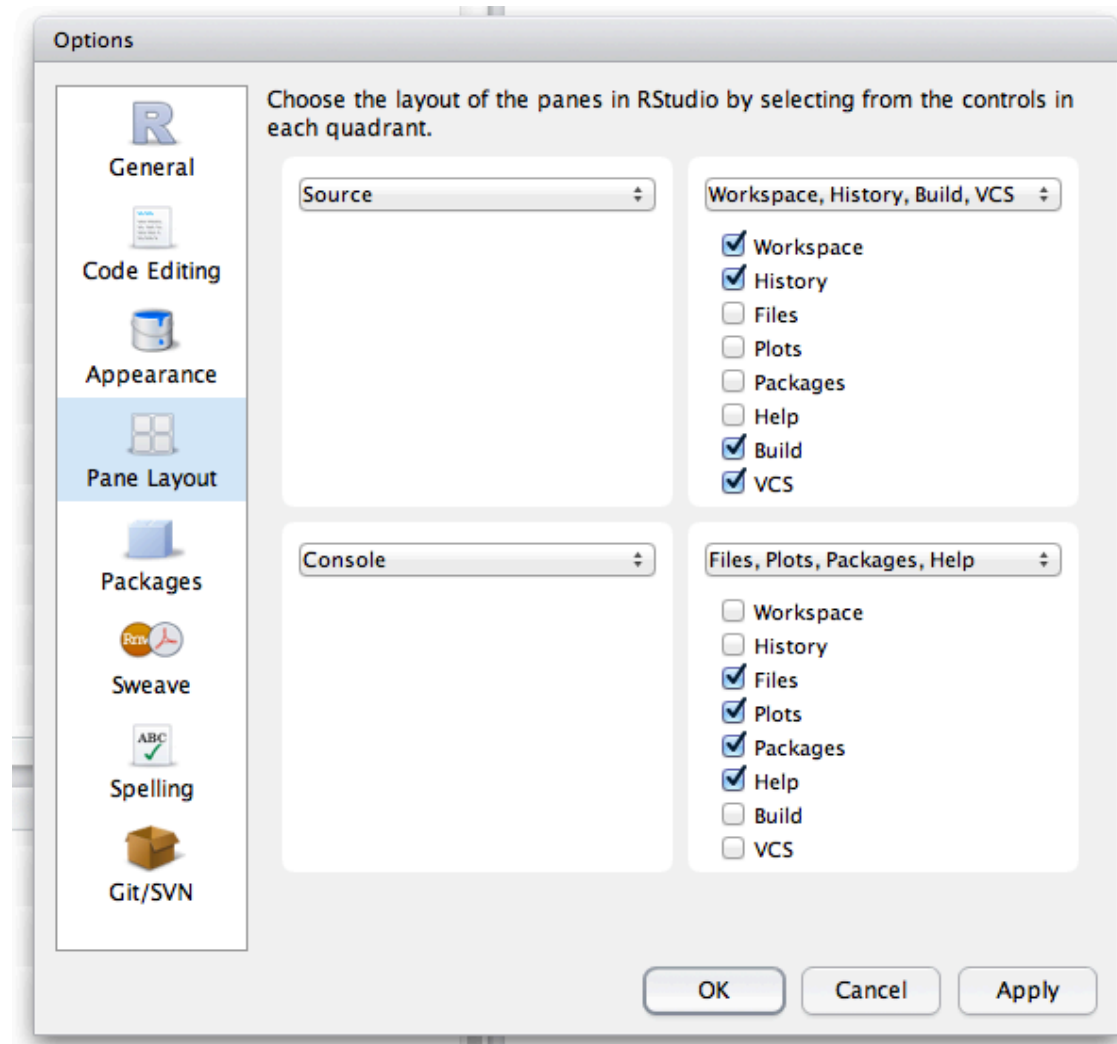
RStudio options



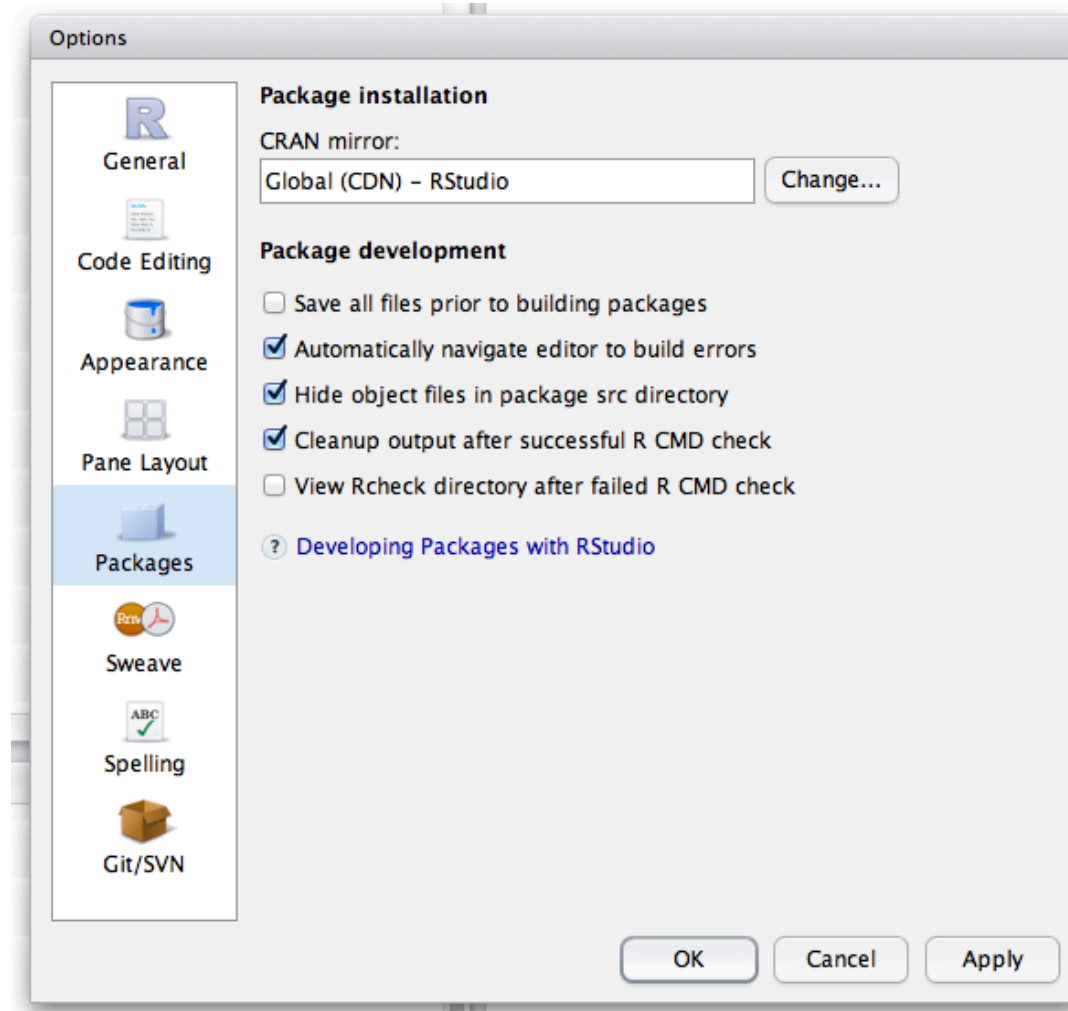
RStudio options



RStudio options



RStudio options



R philosophy

- R is an **interpreted** command line language
- R is using a precise syntax (case sensitive)
myData != Mydata
- R is using objects (object oriented language)
Data objects
Result objects
- Objects have members (data, functions)
accessible with \$: e.g. x\$data
listed with str(x)
- Data are stored internally as vectors (numbers, characters, etc...)
- R allows you to manipulate these objects:
read them
plot them
analyse them
check and change their types
- R has a set of predefined functions and data sets one can use
E.g.: c, vector, plot, print, read.table, anova, etc

R packages

- *Packages* are collections of R functions, data, and compiled code in a well-defined format.
- One can load additional functions by importing packages

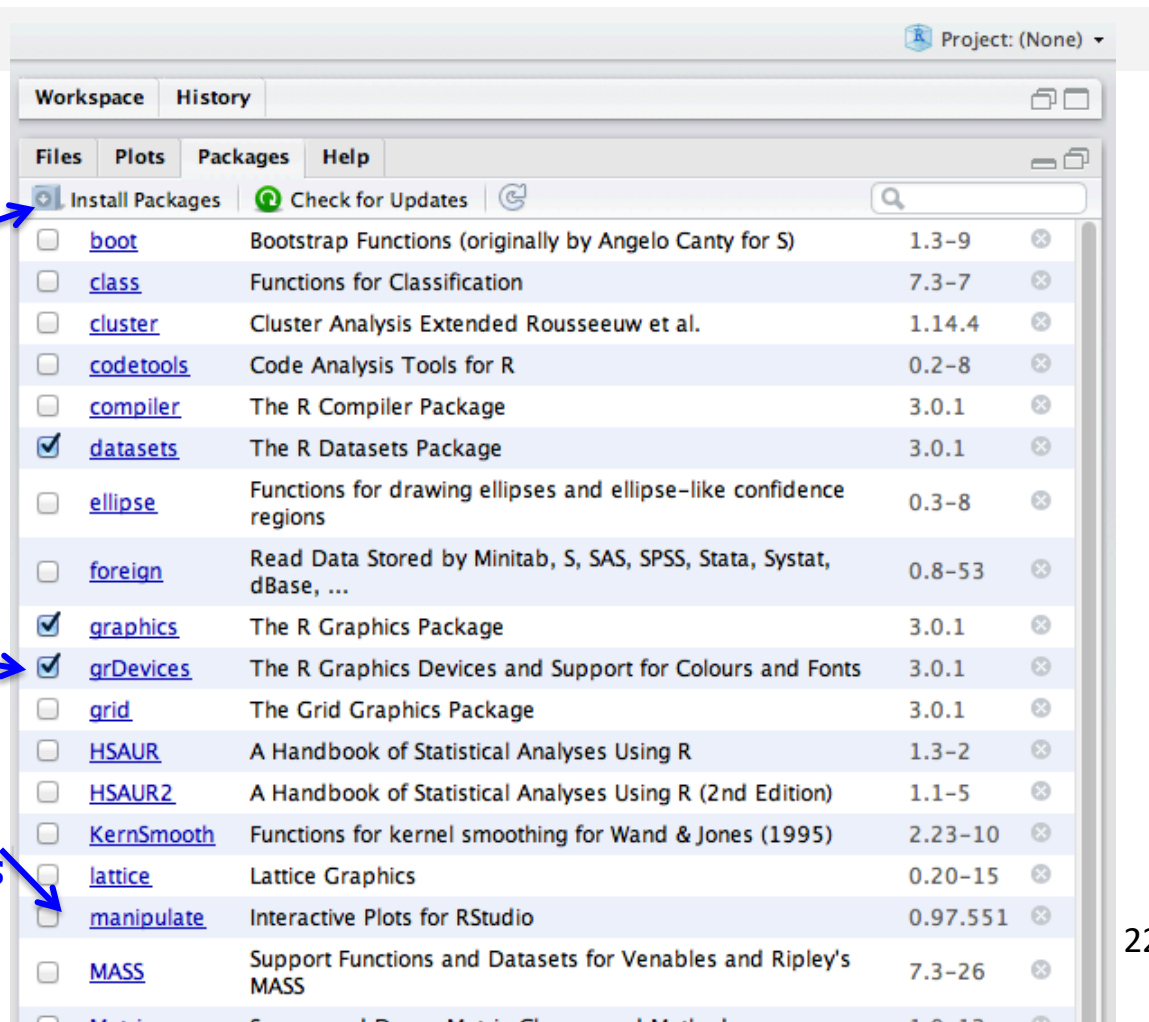
You can list all loaded packages with the command `search()`

```
search( )
```

Click to install new packages

Installed and loaded packages

Installed and not loaded packages



Libraries

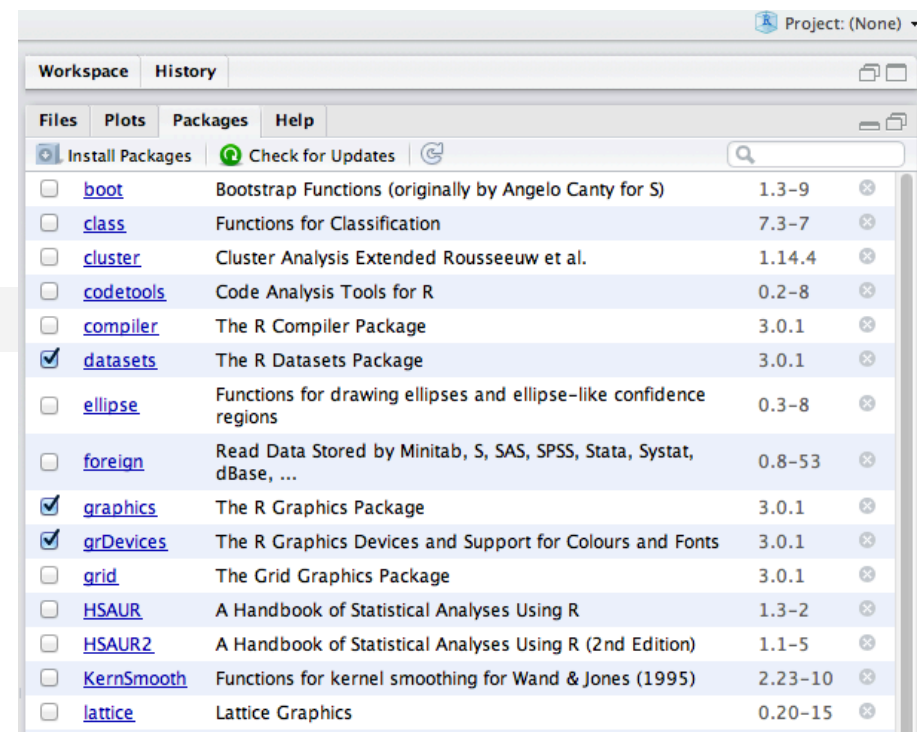
The directory where packages are stored is called the library.

You can load a library by just clicking on one of the installed package

Alternatively, you can explicitly type

```
library("lattice")
```

With this R will be able to access functions and objects located in this package



Note that no object is loaded in R memory when loading the package

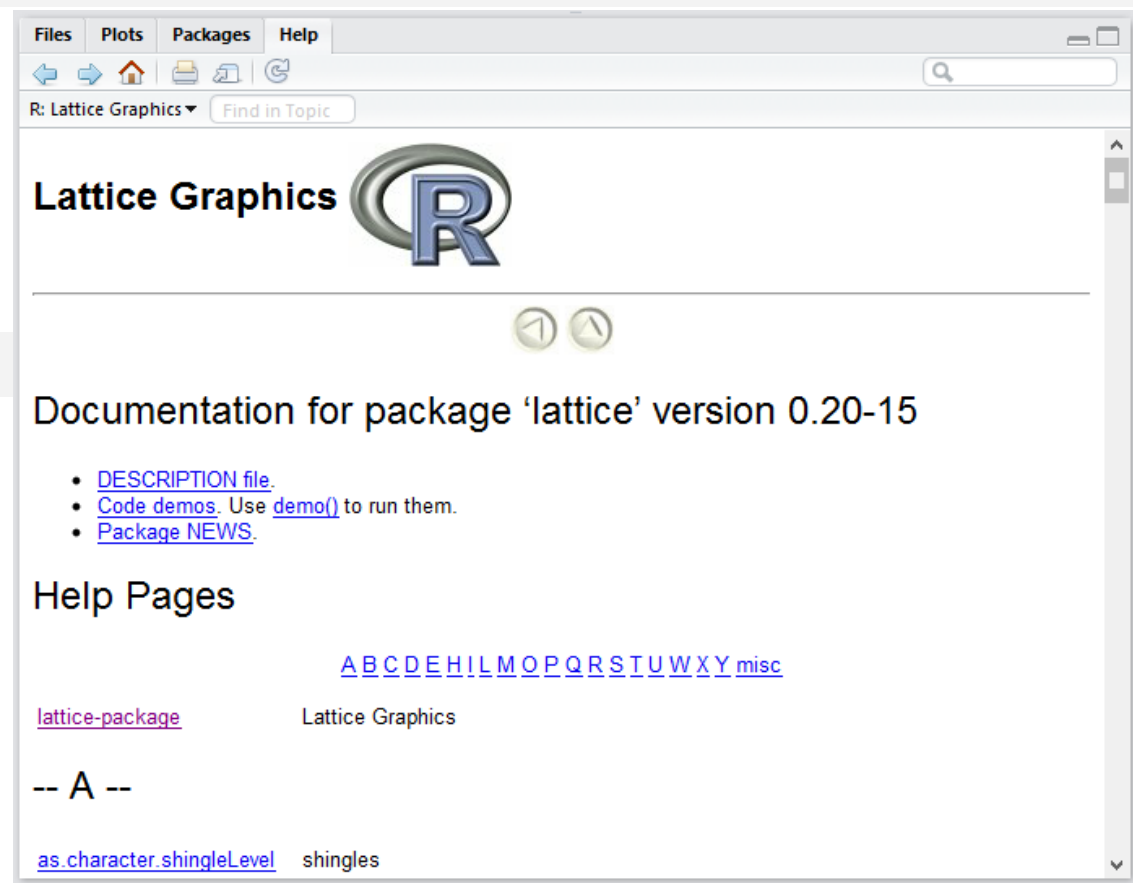
Libraries

You can visualize the content of the package by clicking on the package name, or type

```
help(package = lattice)
```

You can list all the object of the package with the **ls** command

```
ls(package:datasets)
```



Working directory

Rstudio takes the project directory as your working directory. It assumes that all your input files will be located here, and it will output any output file in this working directory

Know where your current working directory is:

```
> getwd()  
[1] "D:/Users/Laurent/Dropbox/CMPG/Courses/R course 2014"
```



Change the current working directory:

```
> setwd("D:/Users/Laurent/Dropbox/CMPG/Courses/R course 2014/R course students")  
> getwd()  
[1] "D:/Users/Laurent/Dropbox/CMPG/Courses/R course 2014/R course students"
```

R only accepts forward slash as in Linux or MacOs (even under windows), and not backslash, as standard in windows, but

```
> setwd("D:\\Users\\Laurent\\Dropbox\\CMPG\\Courses\\R course 2013\\R course students")
```

is okay

Some important notes about programming in R (I)

In R, you save into memory the “*values of stuff*” into variables, using the assignment operator “<-” (kind of arrow) or “=”.

NOTE: computers read from right to left). So, when you type

```
y <- 4
```

you are telling R to assign (save) the value 4 into memory at a variable called y.

What happens if you do this?

```
y <- 4  
x <- 5  
z <- x+y
```



Some important notes about programming in R (II)

In Excel, you see the values directly, but in R, the values of variables are saved into the memory and usually you do not see them, but they are there and you can see them if you type their names in the **console**, or in the **workspace** tab.

	A	B	C	D	E
1		y	x	y+x	
2			4	5	=B2+C2
3					
4					

To see the values stored in the variables you need to type their name

y
x
z

The screenshot shows an R IDE window titled 'Untitled1*'. The source editor contains the following code:

```
1 y <- 4
2 x <- 5
3 z <- y + x
4
```

The Environment pane on the right shows the 'Global Environment' with the following values:

values	
x	5
y	4
z	9

Some important notes about programming in R (III)

Difference between a **variable** and a **function**!

- **Variables** are accessed by its name or with [], e.g. **x** or **x[1]**
- **Functions** are called with **()**, e.g. **help()**. For **functions** you always need its name and brackets afterwards e.g. **plot()**, **c()**, **help()**. You can give variables as input for functions.

For instance

```
y <- 4  
x <- 5  
z <- c(x, y)  
w <- mean(z)
```

What are the **variables** and the **functions** in the above command line?

Some important notes about programming in R (III)

Difference between a **variable** and a **function**!

- **Variables** are accessed by its name or with [], e.g. **x** or **x[1]**
- **Functions** are called with (), e.g. **help()**. For **functions** you always need its name and brackets afterwards e.g. **plot()**, **c()**, **help()**. You can give variables as input for functions.

For instance

```
y <- -4  
x <- 5  
z <- c(x, y)  
w <- mean(z)
```

What are the **variables** and the **functions** in the above command line?

Variables: y, x, z, w

Functions: c() and mean()

A **variable** stores in memory “values of stuff”. “Stuff” can be numbers (like 5e-5, 3.14, 6, etc), names (like “vitor”, “treatment1”, “drugA”), or other complex object.

To access values of a variable we use:

- square brackets [] for vectors, matrices and data frames (e.g **x[1]**)
- double square brackets [[]] for lists, e.g. **x[[1]]**
- dollar sign \$ for data.frames and lists, **x\$variable1**

FUNCTIONS in R (I)

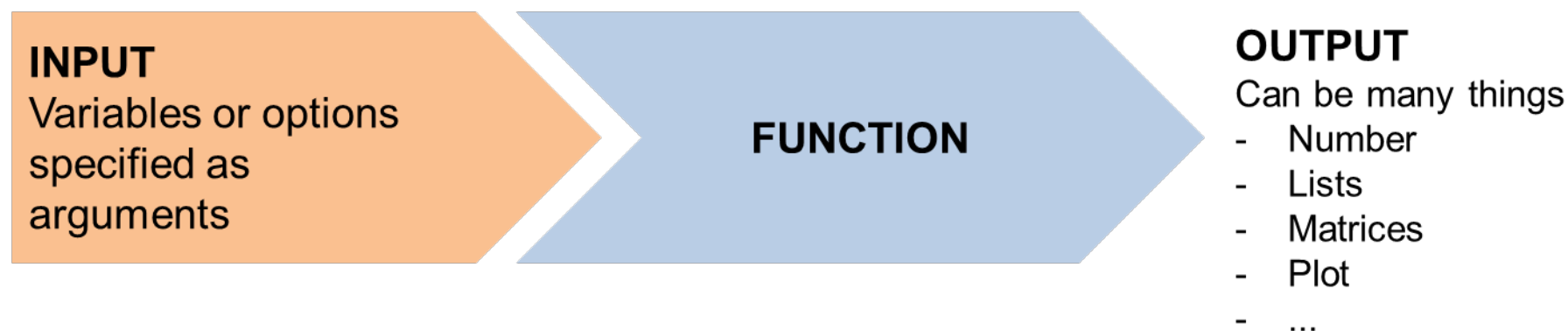
For instance, `mean()` is a function, `c()` is a function, but `mean` is a variable, and `c` is a variable.

Q: What is the difference between `mean()` and `mean`?

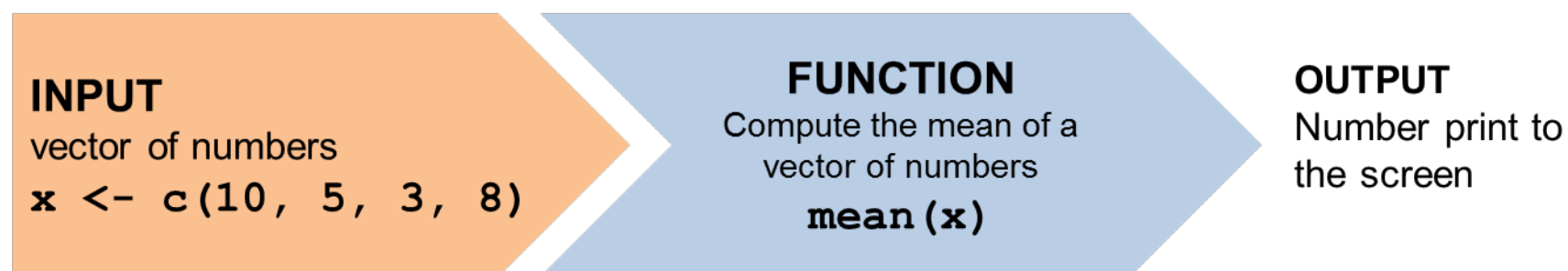
A: A function is identified by the brackets `()`. A variable does not have `()`.

Functions in R (II)

You need to give an input (*arguments* or *parameters*) and you will get an output (that can be *a number, a list, a matrix, a plot*)



Example: Compute the mean of a vector



The **argument** of the function **mean()** is a vector of numbers. When using **mean(x)** the variable **x** is the input.

A few commands to practice

Assign data to variable using '<-' or '=' (**save values into memory**)

```
x=3  
y<-4
```

- Enter data by using function **scan()** – you can type in the console numbers to add them to the variable “a” in the next command line.

```
a <- scan( )
```

- Combine data using the function **c()**

```
x <- c(1,2,3,4,5)
```

- Create a sequence of numbers using the function **seq()**

```
y <- seq(1, 5, by=1)
```

- Create a sequence of numbers using the function “:” – this is a special function that does not need the brackets to be called. As all languages, R has its exceptions, and the function “:” is one of them!

```
z <- 1:5
```

- Create a vector of numbers using the **rep()** function.

```
v <- rep(z,2)
```

- Sample data with or without replication from a vector using the function **sample()**

```
sample(1:12)
```

- Generate normally distributed random numbers distribution using the function **rnorm()**

```
rnorm(10)
```

- Plot using the function **plot()**

```
plot(x=c(1:10), y=rnorm(10))
```

NOTE: In R you can perform several operations in a single line, or go step by step.

For instance, we can do all in a single line

```
plot(c(1:10), log10(1:10))
```

or go step by step

```
# create the x axis - a sequence of numbers from 1 to 10 using the  
function c()  
x <- c(1:10)  
# create the y axis - the log10() of numbers 1 to 10  
y <- log10(1:10)  
# plot x and y  
plot(x, y)
```

For beginners it is always better to do things step by step!

Assign data to variable (save into memory)

If you do not assign the output of a function (or functions) into memory using "<-" or "=", then it is only output to the screen and it is not saved!

#numbers

```
x=3
```

```
y<-4
```

#string of characters

```
day= "Monday"
```

#booleans

```
a=TRUE
```

```
b=T
```

```
c=F
```



To print the values stored in memory, just print the variable name

```
x
```

```
y
```

Perform simple operations on variables

#Use ';' to execute several commands on the same line

```
a;b;c
```

#enter data from keyboard (numbers)

```
z=scan(n=3);z
```

lets assign some values into memory

```
x=2; y=3
```

#additions

```
x+y
```

#exponents

```
x^3
```

```
x^(y+1)
```

```
10^x
```

#logarithms

```
log(100)
```

```
log10(100)
```

```
log2(1024)
```

Two important functions: **c()** and **str()**

Combine data using the **c()** function

#combine numbers into a **vector** with c command

```
t=c(-2, 4, 8)
```



Check the data type stored in memory with function **str()**

```
str(t)
```

#combine numbers and a string into a vector

```
tt=c(1, 2, 'hello')
```

```
tt
```

#check the vector type

```
is.numeric(tt)
```

```
str(tt)
```

```
is.character(tt)
```

```
mode(tt)
```

Create a sequence of numbers

```
x <- c(1,2,3,4,5)
y <- 1:5
```

#Warning, previous value of z will be lost

```
z <- seq(1, 5, by=1)
rep(y,2)
```

IMPORTANT: Every time you assign a value to a variable you delete its previous value!

```
z <- c(1, 5, 4)
x <- z + 10
z <- x
z
x
```

What is the value of z saved in memory after the above operations?

LOGICAL OPERATORS

A special type of functions: they get as input two vectors and return a vector of Booleans (True and False). Note that a number is a vector of length 1.

Compare variables, logical operations

```
comp <- 2 == 3  
comp2 <- 2 != 3
```

Compare vectors, logical operations

```
a <- c(4,8,10)  
b <- c(2,8,9)  
comp <- a==b # what are the elements that are equal?  
comp2 <- a != b # what are the elements that are different?  
comp3 <- a < b # what are the elements of a smaller than b?  
comp4 <- a > b # what are the elements of a larger than b?
```



Logical operators

```
== equal  
!= not equal  
< smaller than  
> larger than  
<= smaller or equal  
>= larger or equal
```

Combining logical operators with AND , OR

```
vectorTrueFalse1 <- a>5 & b<10  
vectorTrueFalse2 <- a>5 | b <10  
isTRUE(vectorTrueFalse1)  
isTRUE(vectorTrueFalse2)
```

Arithmetic operators

Another special type of functions: they get as input two vectors and return a vector of Booleans (True and False). Note that a number is a vector of length 1.

```
+ add  
- subtract  
* multiply  
/ divide  
^ power  
** power  
%% modulus (find even and odd numbers)  
/% integer division
```

Logical and arithmetic operators

Arithmetic Operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/2 is 2

Logical Operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

Combining different types of data with paste

paste can be used to concatenate different types of data into a string (chain of characters)

```
a="I think"  
b="therefore"  
c="I am" #actually never use c as a variable name!!!! (it is an R command)  
paste(a,b,c)  
paste(c,b,a)
```

paste is quite powerful

```
paste("there are", 10, "types of people in the world:", "Those who  
understand binary, and those who don't...") #10 is converted into chars
```

```
paste("Today is", Sys.Date())
```

```
paste(letters[c(1,3,5)], 1:3, sep=".")
```

```
paste("[", LETTERS[1:5], "]", sep="") #example of recycling
```



Time to practice:

Exercises 1.1 to 1.3

While solving the exercises, identify what are the variables, and what are the functions.

Scalars and Vectors

Scalars: numeric values, e.g.: 5, 0.5, 3.14159

```
# create scalar data object x and assign value 5  
x <- 5
```

Vectors: a list of scalars, e.g.: (5,0.5,3.14159) - the whole list is a single object!

```
# create vector v by concatenating scalars  
v<-c(20,1,7.5,13)  
# create scalar vector v as sequence from 1 to 5  
v<-seq(1,5,by=1)  
V<-1:5  
# create vector of strings:  
v<-c("one","two","three")
```

The *i*th element of v can be **accessed** by:

```
v[i]
```

The *j*th element of v can be **filtered out by using negative indices**

```
v[-j]
```

Matrices

Matrices are table-like objects with rows and columns of scalars:

	Column 1	Column 2	Column 3
Row 1	4	5	8
Row 2	6	5	12
Row 3	14	3	9

Internally, elements are stored in a vector:

	Column 1	Column 2	Column 3
Row 1	Element 1	Element 4	Element 7
Row 2	Element 2	Element 5	Element 8
Row 3	Element 3	Element 6	Element 9

Matrices

Different ways of creating matrices in R:

```
mat <- matrix(c(1,2,3,4),nrow=2,ncol=2)
```

```
mat <- c(1,2,3,4); dim(mat) = c(2,2)
```

```
mat <- cbind(c(1,2,3),c(4,5,6)) # the vectors are treated as columns
```

1	4
2	5
3	6

```
mat <- rbind(c(1,2,3),c(4,5,6)) # the vectors are treated as rows
```

1	2	3
4	5	6

The element in row i and column j can be accessed by `mat[i,j]`

```
mat[2,2] <- 5
```

The i th row can be accessed by `mat[i,]` and the j th column by `mat[,j]`

```
mat[1,] <- c(5,6,7)
```



Graphs in R

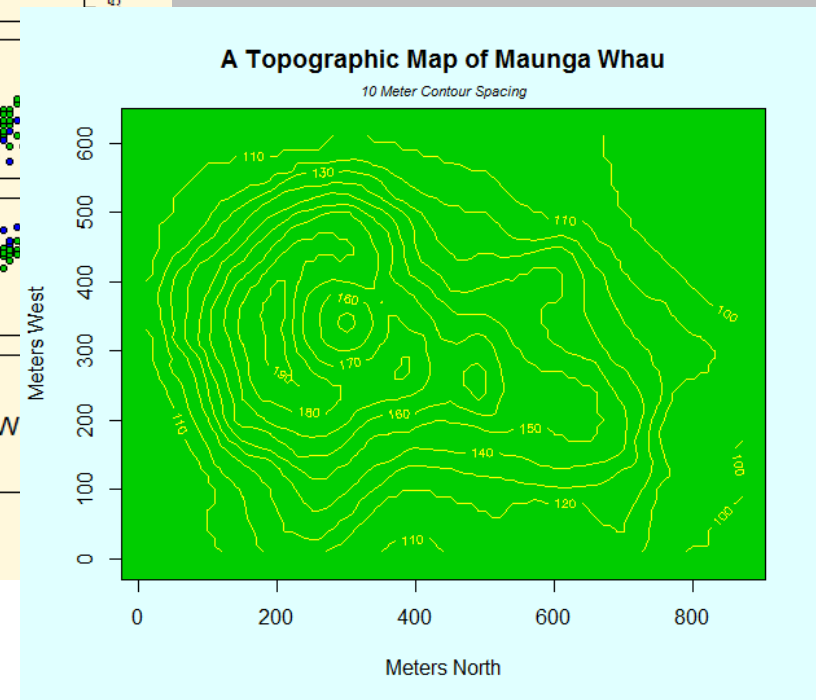
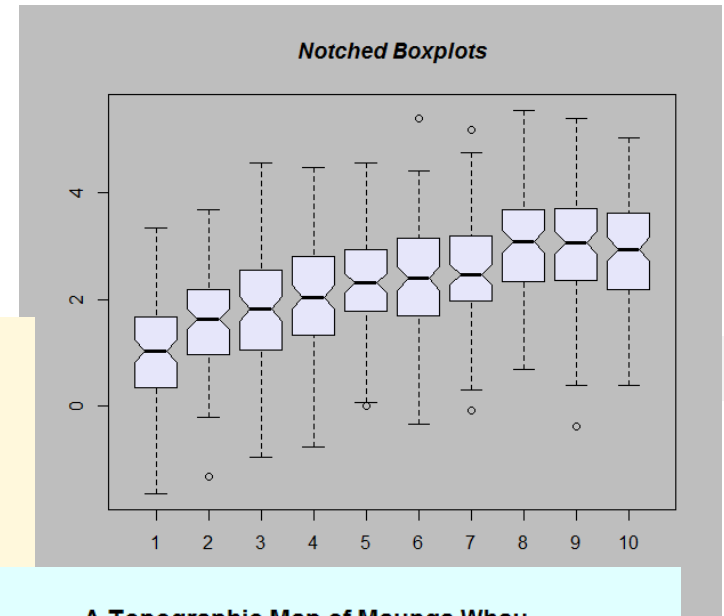
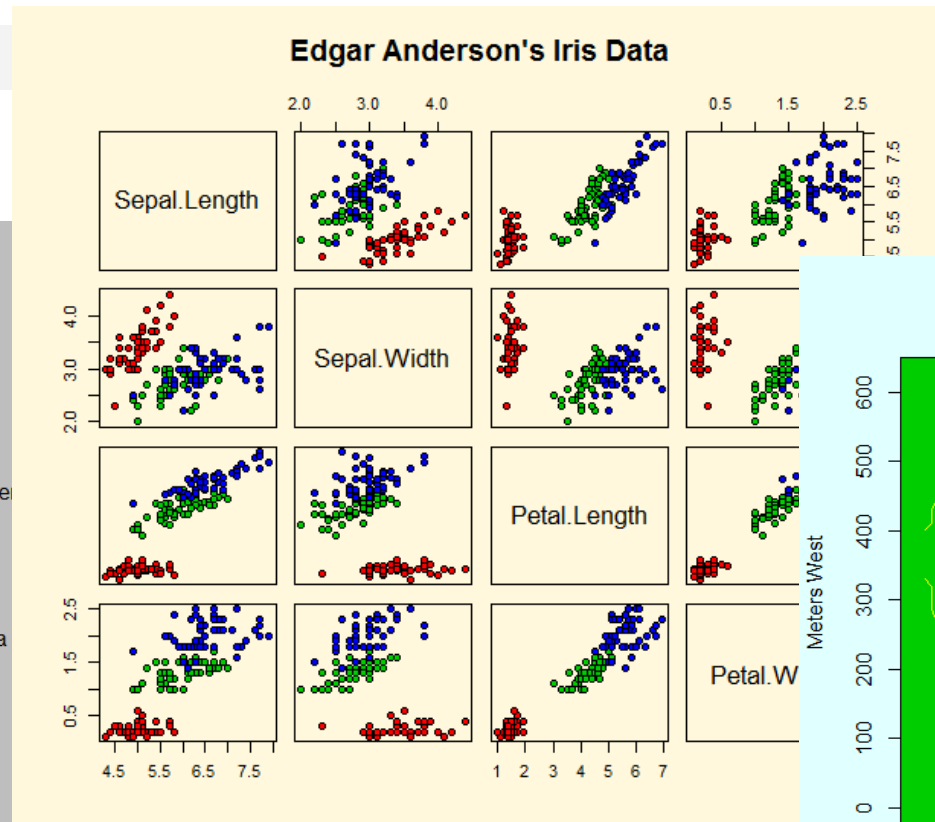
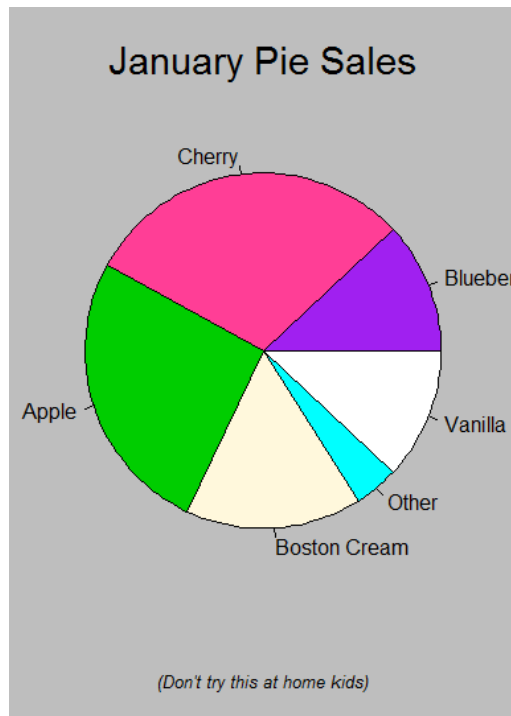
R can be used to produce quite complex graphs (to be seen on day 2)

```
demo(graphics)
```

Graphs in R

R can be used to produce quite complex graphs

```
demo(graphics)
```



Goals for first day

- Know what R can do
- Be comfortable with R studio interface and R in general
- Find relevant information with help
- Know some basic syntax and data types
- Manipulate vectors and matrices
- Be able to use R as a calculator

If there is time, only **after you solved all exercises Day 1**, you can practice **on your own** using interactive **swirl()**

R programming Tutorial

Swirl Intro tutorials

Swirl provides several interactive tutorials to learn how to use R. I recommend that you follow this before the course to get familiar with the R environment. Just follow the following steps

In RStudio console type:

```
install.packages("swirl")
```

For Linux users you might need to update a few libraries in your system. Install them using the following commands in your terminal.

```
sudo apt-get install libcurl4-openssl-dev  
sudo apt-get install libssl-dev
```

After installing the package swirl, you can start the tutorials.

```
library("swirl")  
swirl()
```

After asking for your name (anything you wish), it will print something like:

```
To begin, you must install a course. I can install a course for you from the internet, or I can send you to a web page (https://github.com/swirldev/swirl\_courses) which will provide course options and directions for installing courses yourself. (If you are not connected to the internet, type 0 to exit.)  
1: R Programming: The basics of programming in R  
2: Regression Models: The basics of regression modeling in R  
3: Statistical Inference: The basics of statistical inference in R  
4: Exploratory Data Analysis: The basics of exploring data in R  
5: Don't install anything for me. I'll do it myself.
```

Select the option 1: R Programming: The basics of programming in R.

Go through the tutorial, going through lessons 1 to 4.

1: Basic Building Blocks; 2: Workspace and Files; 3: Sequences of Numbers; 4: Vectors;

Exercises Part 1 day 1

All the exercises are found in the file ExercisesDay1_2018.r