Introduction to R programming

Emily Haeuser

Eva Malecore

Ecology Konstanz

TReND

Institute for Health Metrics and Evaluation

The Company of Biologists

International Max Planck Research School for Organismal Biology

Overview

Topics
1 – Introduction to R programming
2 – More advanced R
3 – Data exploration

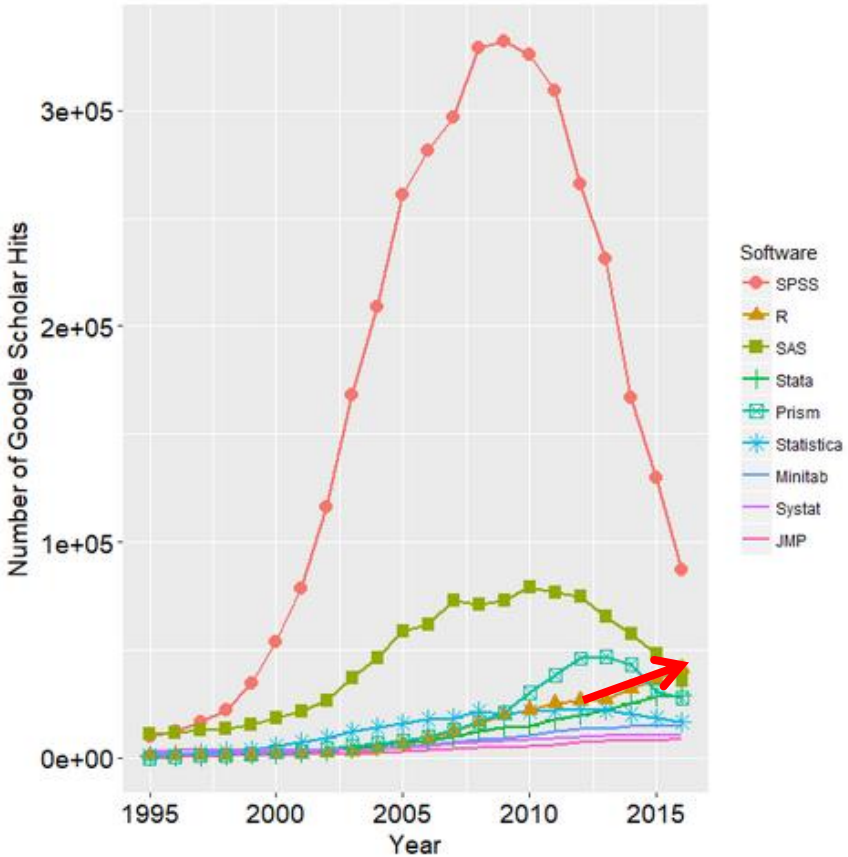
Why use R for statistics?

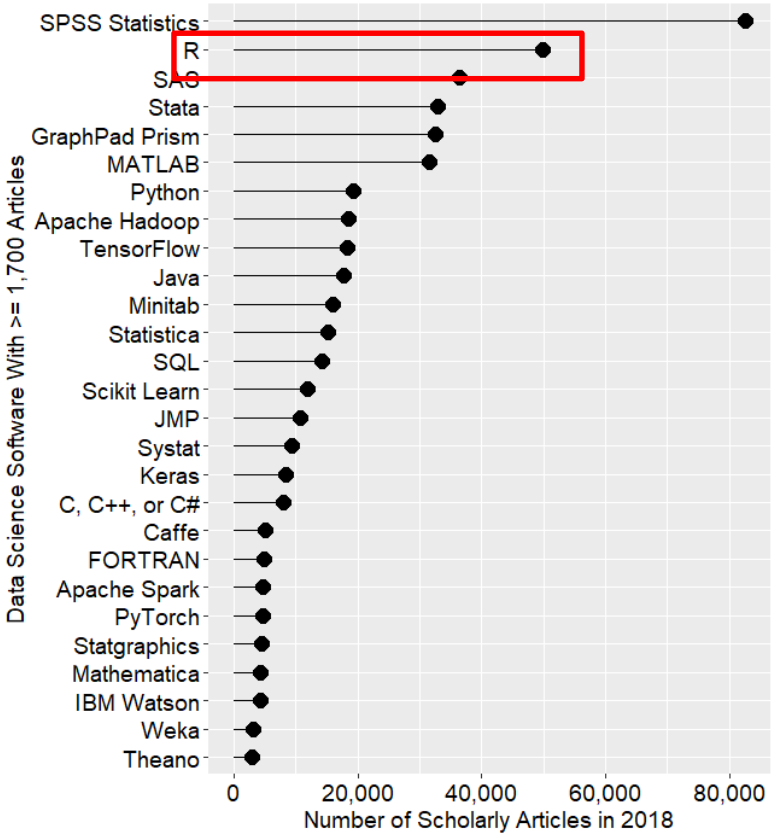
- R combines advantages:

	Limited possibilities	Great possibilities
Expensive		
Free		

Why use R for statistics?

- R has become one of the leading data science software for research articles: http://r4stats.com/articles/popularity/





What is R?

"R is a **language** and **environment** for statistical computing and graphics."

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

- **Programming language**
- Implementation of the **S language** (created by John Chambers, Bell Labs)
- Note: another implementation is S-PLUS, a commercial product
- R was created by **R**oss Ihaka and **R**obert Gentleman, Uni. Auckland, NZ
- Since 1997: developed by the ***R Development Core Team***

Why "R"??

- Because of the first names of the first two creators
- As a play on the name of S, which it derives from

- Interpreted language, i.e. instructions are executed directly, without compiling
- Multi-paradigm: **array** programming, **object-oriented** programming, **procedural** programming…

What is RStudio?

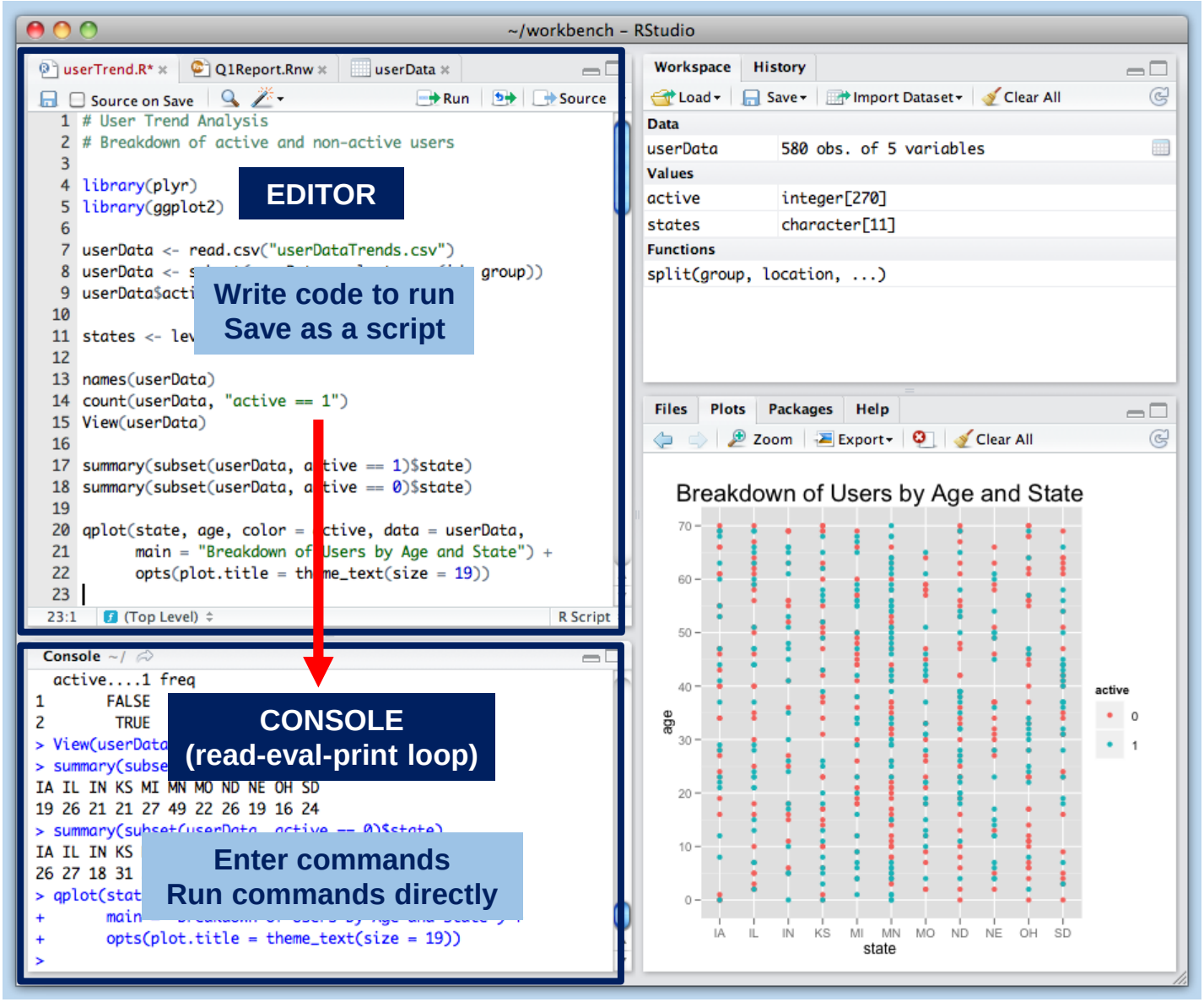
https://www.rstudio.com

- An **integrated development environment** (IDE) built for R
- **Free, open source**
- More user-friendly than the standard R GUI that you get when installing R
- Available as **RStudio Desktop** (regular desktop app) and RStudio Server (using a web browser while running on a remote Linux server)
- Overview: https://www.rstudio.com/products/RStudio/

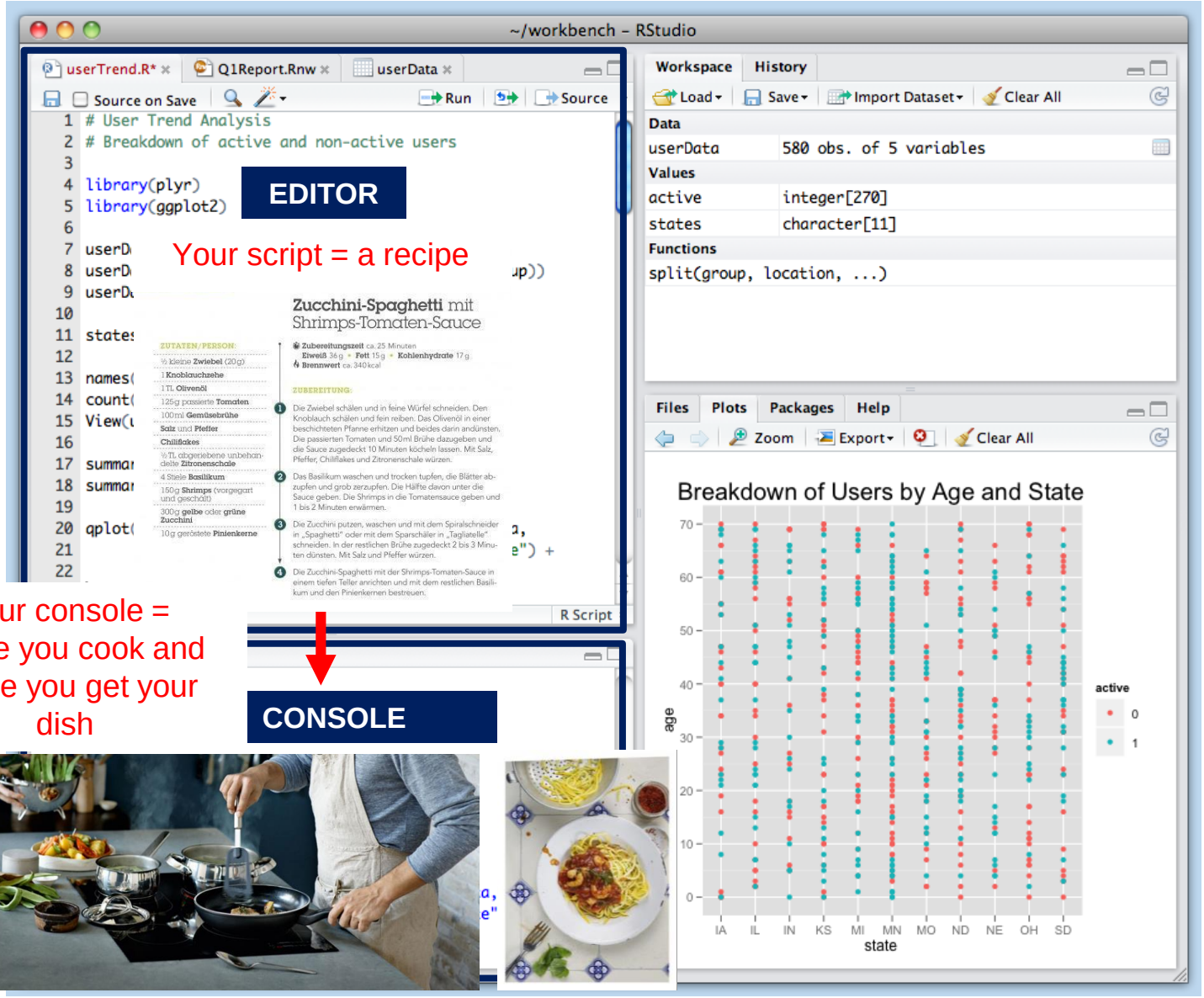
- Note: if you don't want to use RStudio, you may still want to use a text editor, which offers **syntax highlighting** (e.g. Notepad ++, Tinn-R)

- Other R-IDE'S: Rattle GUI, R Commander, and RKWard.

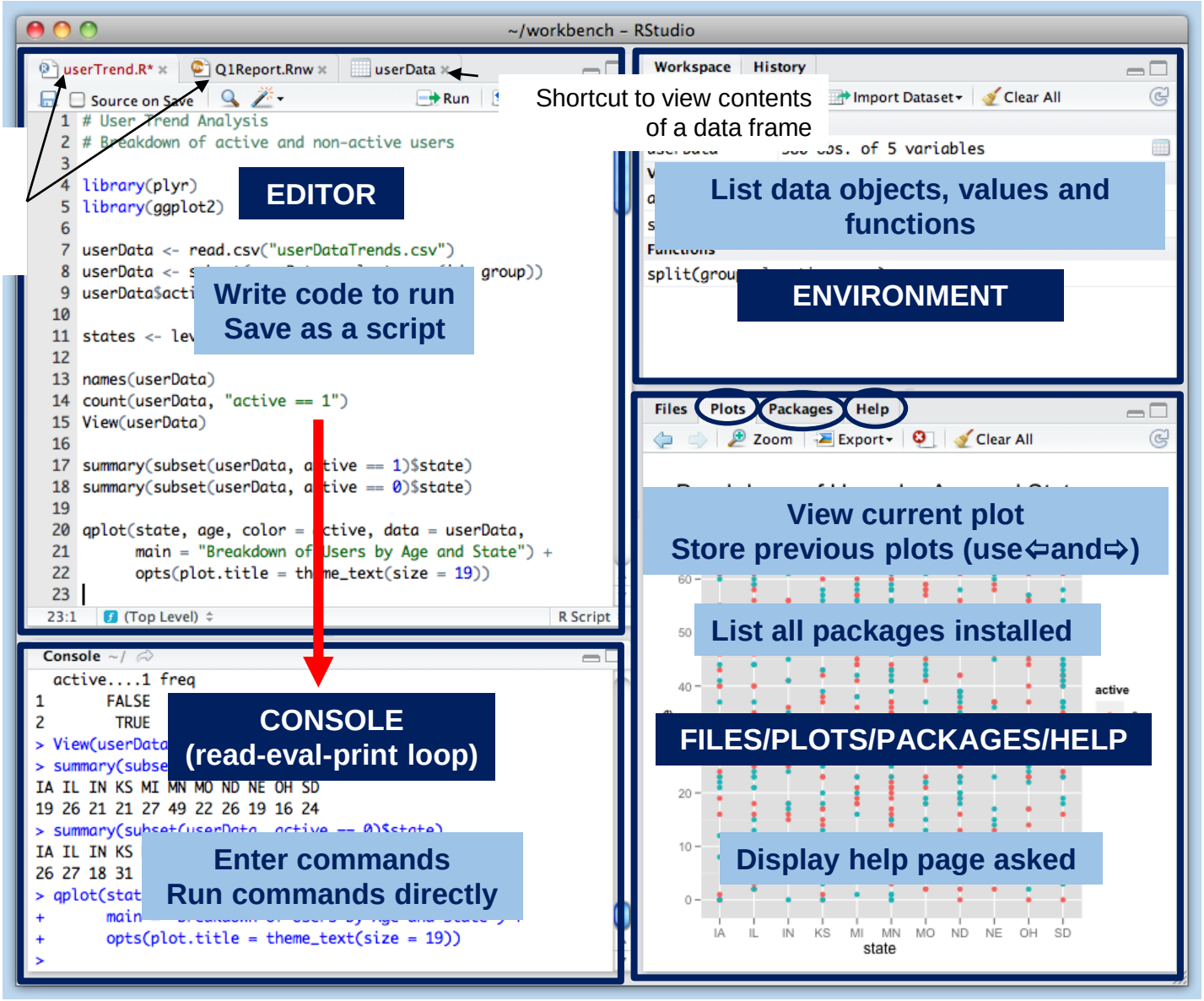
What is RStudio?



What is RStudio?



What is RStudio?



Let's get started!

1. Directories

What is your **working directory**? → The directory where by default

- data will be imported from and exported to.
- scripts, graphs and files will be saved to

You can:

- Check what is your current working directory: *getwd()*
- Set what you want to be your working directory: *setwd("C:/PathToMyWorkingDirectory")*

What if I want to refer to **another directory** for a specific task?

You can:

- Import data from any alternative directory
  e.g. *read.table("C:/PathToAnotherDirectory/MyDataFile.txt")*
- Save data and figures to any alternative directory
  e.g. *write.table(MyData,"C:/PathToAnotherDirectory/MyDataFile.txt")*
  or *pdf(MyData,"C:/PathToAnotherDirectory/MyDataFile.pdf")*

Let’s get started!

2. Packages

Set core of packages

- You can see in the ‘Global Environment’ the list of base packages that are automatically loaded when starting R
- If you select one of these base packages, you can even see the list of functions that are already available in this package

Additional packages

- You can check which packages are already loaded by going through the **list in the ‘Packages’ tab in the bottom right panel** (loaded packages have a tick)

Two steps:

- **Install** a package (that you do not have yet on your computer):
  either through Tools > Install Packages (a window pops up)
  or with the code *install.packages(“NameOfThePackage”)*
- **Load** a package when you start a **new session**
  library(NameOfThePackage)

  or by ticking the box for the respective package in the ‘Packages’ tab

Let’s get started!

3. Expressions, assignments and functions

Expressions

An expression is **evaluated, printed**, and **the value is lost** (i.e. you see it printed in the console, but it is not stored), e.g. *5+3*

Assignments

An assignment also **evaluates** an expression, but it **passes the value to a variable**, and the result is **not automatically printed**, e.g. *x<-5+3* (or *x=5+3*)

On the **left side** is the **name of the variable** to which a value is assigned

“**<-**” is the **assignment operator** interchangable with “=”

On the **right side** is the **value to be assigned** to x

Functions

Functions perform some specific actions, they can be used as expressions or in assignments, e.g. *mean(1:10)* or *x<-mean(1:10)*

Let's get started!

4. Objects and data classes

Vectors: collections of elements of the same type

Vectors are usually created with the *c()* (concatenate) function

e.g. *x<-c("A","B","C")*

- There are six basic ('atomic') vector types
  1. **Character**: character strings
     e.g. *This is a character string"*
  2. **Numeric double**: real numeric values
     e.g. *3.14*
  3. **Numeric integer**: integer numeric values
     e.g. *12*
  4. **Complex**: complex numbers (not so relevant)
     e.g. *2+3i*
  5. **Logical**: true or false
     TRUE (also *T*) or *FALSE* (also *F*)
  6. **Raw:** raw bytes (mainly for advanced programming)

Lists: they differ from vectors in that they may contain objects of different types

e.g. *x<-list("This is a character string", 3.14,12,2+3i,FALSE)*

Let's get started!

The **class** is an attribute of an R object. There are different **data classes** in R:

- **Factors**: vectors which take on a limited number of different values (categorical variables)

  e.g. *vec1<-factor(c("A", "B", "A", "C", "B", "B", "A", "C"))*

  You can check the list of factor levels using *levels(x)*

- **Matrices**: multi-dimensional vector (i.e. contains elements of the same type)
  Their dimension determines their form, i.e. *dim(mat1)<-c(2,3)* ≠ *dim(mat1)<-c(3,2)*
  Remember: **always *Rows,Columns***!

  e.g. mat1<- matrix(c(2, 4, 3, 1, 5, 7), nrow=3, ncol=2)

  You can access elements of a matrix using [], *mat1[2,2]*

- **Data frames**: tables with **rows** and **columns** (list type)
  This is typically what you create when you import a dataset!
  Columns have headers, and each **column** can be access separately by using the symbol **$** followed by the respective column name or header

  e.g. *df$HeaderOfMyFirstColumn*

  e.g. *df<-as.data.fram(mat1)*

Let's get started!

You can **check the class**

- of an object using *class(MyObject)*
- of all columns of a data frame at a time using *str(MyDataFrame)*

You can use their **position** to access **single elements**

- of a vector, e.g. *x[2]*
- of a list, e.g. *x[[2]][1]* (first element of the second object of the list)
- of a data frame or a matrix, e.g. *x[2,3]*
  Standard matrix notation: if you want all columns for the second row: *x[2,]*
  Similarly, if you want all rows for the third column: *x[,3]*

You can **coerce** an object into another type (**when it makes sense!**)

- When numbers have been input as character strings, use *x<-as.numeric(x)*
- When you want character strings to be factor levels, use *x<-as.factor(x)*

Missing values (whenever we don't have a measure of a variable):

- Missing values are stored as NA
- Is a NA (Not Available) value truly a NA or the character string "NA"?

Let’s get started!

5. Input data

Reading in data from a file (typically after you entered your data in Excel

- From a .csv file:
  MyData<-read.csv("MyDataFile.csv")
- From a .txt file:
  MyData<-read.table("MyDataFile.txt")
  Default arguments that are useful to know and can be modified:
  header=TRUE, sep="", dec=".", na.strings="NA"
- Use RStudio import wizard
- For other datatypes, use specific import function
  e.g. *library("ape"); MyTree=read.tree(" MyTree.tre ")*

Let’s get started!

Creating objects directly in R

- A **vector**: *x<-c(1:10)* or *x<-c("A", "B", “C")*
- A **list**: *x<-list(“A”,1:10)*
- A **matrix**: *x<-matrix(nrow=3,ncol=2,1:6)*
  The number of values to feed the matrix must be equal to the number of matrix elements!
- A **data frame**: *x<-data.frame(length=c(25:35),nstol=c(0:10))*
  When we do that, we give a name to the columns while creating the data frame.
  We can check column names using *names(x)* or *colnames(x)*.
- A **factor**: *x<-factor(c("A", "B", "A", "C", "B", "B", "A", "C"))*
- Referring to **parts of another object**, in particular from a data frame
  e.g. *y<-names(x)* (y is then a vector containing the headers of the data frame x)
  or *y<-x$HeaderOfTheFirstColumn* (y is then a vector containing the whole first column of the data frame x)

Let’s get started!

6. Managing your workspace

Your workspace contains **all objects, values and functions** that you have **stored** so far.

You can

- Check the **whole list of objects** that are in your workspace: *ls()*
- **Remove one particular object** from the workspace: *rm(MyObject)*
- **Remove several objects** at once, but not all of them: *rm(list=c(Object1,Object2))*
- Clear the workspace, i.e. **remove all objects** at once: *rm(list=ls())*
- You can also use the Global Environment for all the previous functions

Let's get started!

7. Handling data in R: useful functions

Arithmetic operators:

- Addition *+*, e.g *5+2* returns *7*
- Subtraction *–*, e.g *5-2* returns *3*
- Multiplication ***, e.g. *5*2* returns *10*
- Division */*, e.g. *5/2* reuturns *2.5*
- Exponentiaion ^ or ****, e.g. *5^2* returns *25*
- Modulus *%%*, e.g. *5%%2* returns *1*
- Integer division *%/%*, e.g. *5%/%2* returns *2*

Let’s get started!

7. Handling data in R: useful functions

Logical operators:

- Less than, less than or equal to *<, <=*
- Greater than, greater than or equal to *>, >=*
- Exactly equal to *==*
- Not equal to *!=*
- Not *!*
- Or *|*
- And *&*
- isTRUE *isTRUE ()*

Let’s get started!

7. Handling data in R: useful functions

View part of the data

- The first rows of a data frame: *head(MyDataFrame,20)*
- The last rows of a data frame: *tail(MyDataFrame,20)*

View part of the data

- Using the **function subset()**
  e.g. *y<-subset(x,ColumnUsedToSubset== "FactorLevelB")*
  or *y<-subset(x,ColumnUsedToSubset>0)*
- Using **square brackets**
  e.g. *y<-x[ColumnUsedToSubset== "FactorLevelB",]*
  or *y<-x[ColumnUsedToSubset>0,]*

Let’s get started!

7. Handling data in R: useful functions

Explore and check if there are issues

- Structure of data frame *str(x)*
- Are there NA values in the data frame: *any(is.na(x))*
- Are there NA values in a particular column of the data frame: *any(is.na(x$HeaderOfTheColumn))*

Summarize data

- Summarize a numeric/integer variable: *summary(MyVariable)*
  Note that it will also tell you if you have NA values, and how many
- Summarize a factor variable: *table(MyVariable)*
- Apply a function (e.g. mean) to a variable grouped by another variable: *aggregate(x$VariableToApplyTheFunction,x$VariableUsedToGroupBy,function)*
- Count the number of observations in one part of a data frame: *length(x$HeaderOfTheColumn)*
  Note that this can be combined with subsetting: *length(x$HeaderOfTheColumn[x$HeaderOfTheColumn>0])*

Let’s get started!

8. Saving your work when closing an R session

When you want to **quit R session**, you are asked whether you want to **save workspace image**:

- If yes, then it will create a **.RData file**, which contains **all the objects, values and functions** currently stored in your workspace, and that you can load as such when starting a new R session.
- At the same time, it will create a **.Rhistory file**, which contains **all the command lines** that you have used during the session you want to close.

RStudio keeps the edits in your script even when you do not save the session, but it is good practice **to save your R script inbetween**, not only when closing the R session!

Let's get started!

9. Simple plots (basics)

Use the *plot()* function

e.g. *x=seq(1:12); y=seq(4:15); plot(x,y)*

Arguments:

- *type*: points, lines etc.
- *xlab,ylab* and *main* : the labels on the x and y axis, plot title
- *col*: the color of the plotted data
- *pch*: point character, the shape of the points in the plot
- *cex, cex.lab*: character expansion, the size of the points and labels
- **TIP**: *ifelse()* function when plotting more than one color or *pch*

  ifelse(condition, value if condition is true, value if condition is false)

  e.g. *pch=ifelse(x==1,1,2)*

Use *points ()* to add points to an existing plot

Use *abline()* and *lines()* to add lines to an existing plot

Some tricks ;-)

- When you **read in data**, **copy-paste the name of the file** instead of typing it in, you save time and avoid mistakes!

- In the **editor**, you can create **headings in your script** by adding "**####**" after the text that you want to be your heading. It helps **navigate through** long scripts!

- In the **console**, you can use ↑ **and** ↓ to **rerun or edit lines of code** you have recently used (to avoid typing just the same over and over again).

- You can **clear the console** at any time using **ctrl+L**, your command lines are still in the 'History' tab in the top right panel.

- In the **'Files' tab** of the bottom right panel, you can see the contents of your **working directory**, create there a new folder, delete or rename some files, navigate and set a new working directory, etc.

Useful resources

- A guide to R and Rstudio: https://www.sitepoint.com/introduction-r-rstudio/
- **R manuals**: https://cran.r-project.org/manuals.html
- Crawley, M. J. (2013) ***The R Book***, 2nd edition. Wiley, UK
- Various online tutorials:
  - http://www.cyclismo.org/tutorial/R/
  - http://www.r-tutor.com/r-introduction
  - http://data.princeton.edu/R
  - https://www.datacamp.com/
  - Youtube tutorial videos
  - …
- **Interactive** tutorials:
  - http://tryr.codeschool.com/ (online)
  - http://swirlstats.com/ (in the R console)
  - …
- Search functions: http://rseek.org/

How to cite?

How to cite **R software**? >> In R, use ***citation()***.

```
> citation()

To cite R in publications use:

  R Core Team (2016). R: A language and environment for statistical computing. R Foundation for Statistical Computing,
  Vienna, Austria. URL https://www.R-project.org/.

A BibTeX entry for LaTeX users is

  @Manual{,
    title = {R: A Language and Environment for Statistical Computing},
    author = {{R Core Team}},
    organization = {R Foundation for Statistical Computing},
    address = {Vienna, Austria},
    year = {2016},
    url = {https://www.R-project.org/},
  }

We have invested a lot of time and effort in creating R, please cite it when using it for data analysis. See also
'citation("pkgname")' for citing R packages.
```

How to cite a **particular R package**? >> In R, use ***citation("MyPackage")***.

```
> citation("ggplot2")

To cite ggplot2 in publications, please use:

  H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2009.

A BibTeX entry for LaTeX users is

  @Book{,
    author = {Hadley Wickham},
    title = {ggplot2: Elegant Graphics for Data Analysis},
    publisher = {Springer-Verlag New York},
    year = {2009},
    isbn = {978-0-387-98140-6},
    url = {http://ggplot2.org},
  }
```

Note: the R package must be loaded
(use library(MyPackage) if it is not loaded yet).

Time for an exercise

Use the excel spreadsheet “Leaftraits.xlsx”. Now we will play around with this dataset in R.

(1) Save the data as a .txt (tab delimited) file

(2) Import the leaftraits.txt in R.

(3) How many observations are there in this dataset?

(4) Display the list of fields that the dataset contains. What is the datatype of each field?

(5) Check if there are NAs in the Leaf Mass per unit Area (LMA) field. How many are there?

(6) How many different types of growth form (GF) are there?

(7) Create a subset that contains only “Tree” (T) growth form.

(8) How many species have a Nitrogen per unit mass (Nmass) larger than 1.5%?

(9) Plot the Nitrogen per unit mass. Use red for Nmass>1.5%, and blue for Nmass <=1.5%.

(10) Plot the Nitrogen per unit mass. Use red for “Tree” species and green for all the others.

(+) Label the previous plot properly. Play around to make them nicer.

More advanced R

1. Control flow statements

for loop

A **for loop** allows us to repeat (loop) through the elements of a vector or list and run the same code for each loop

```
for( index in seq) {
    expr
}
```

The { } curly brackets wrap the code of the for loop

seq *= some vector or list*
expr *= some code*

e.g.

```
for (i in 1:5) {
    print(i^2)
}
```

The print() function prints out its arguments on the Console

```
x=c(1,7,23)
for (i in x) {
    print(i)
}
```

The index “i” will be replaced with the values of the vector x

More advanced R

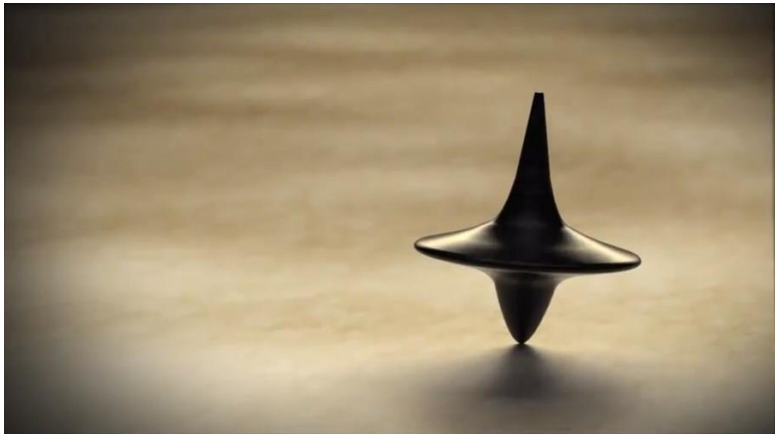
Nesting for loops

We can have a **for loop** within a **for loop**

e.g.

```
mat1 = matrix(nrow=30, ncol=30)
for(i in 1:dim(mat1)[1]) {
    for(j in 1:dim(mat1)[2]) {
                mat1[i,j] = i*j
    }
}
```

Or a **for loop** within a **for loop** within a **for loop**



More advanced R

If else statement

An if else statement is used to execute some code if a certain condition is TRUE, and some alternative code if the condition is FALSE. The else is not compulsory.

If (cond) {
expr1
} else {
expr2
}

cond *= condition, a logical condition returning TRUE or FALSE*
expr *= some code*

e.g.

```
x=sample(1:20,1)
if (x) <= 10) {
    print("x is less than 10")
} else {
    print("x is greater than 10")
}
```

The function sample(x,n) samples n number from x

ifelse(cond,value1,value2*)*

The function sample(x,n) samples n number from x

More advanced R

Other control flow statements

while loops

```
while(cond){
  expr
}
```

cond *= condition, a logical condition returning TRUE or FALSE*
expr *= some code*

repeat break

```
repeat {
    expr
    if (cond) {break}
}
```

break: interrupts a control flow statement

next: skips to the next loop

More advanced R

2. Functions

A **function** is used to repeat the same lines of code more than once

```
my.function = function(arg1, arg2,…) {
    expr
return(return.obj)
}
```

e.g.

```
my.function= function(a,b,c){
    result=a*b-c+a^2
    return(result)
}
```

my.function *= function name*
arg *= arguments passed to the function*
expr *= some code*
return.obj *= the object returned by the function*

More advanced R

3. Apply family

An alternative to for loops. Code is more compact, but execution time can be slower for big data.

apply(), lapply() , sapply(), vapply(), mapply(), rapply(), and tapply()

apply(x, margin, function)

e.g.

```
mat1 = matrix(nrow=30, ncol=30,
              data=c(1:30*30)))

lapply(mat1, 2, mean)
```

vs

```
mean=NULL
for (i in 1:ncol(mat1)) {
  mean[i]=mean(mat1[,i])
}
```

x= *a 2d array (e.g. matrix, dataframe)*

margin*=1 if you want to apply your function by rows, 2 if you want to apply your function by columns*

function*=the function you want to apply*

returns a vector

More advanced R

Function	Syntax	Arguments	Return
lapply()	*lapply(x, function)*	***x***= *a vector or an object (e.g. a list)* ***function***=*the function to apply to each element of X*	list
sapply()	*sapply(x, function)*	***x***= *a vector or an object (e.g. a list)* ***function***=*the function to apply to each element of X*	vector
vapply()	*vapply(x,function, output.type)*	***x***= *a vector or an object (e.g. a list)* ***function***=*the function to apply to each element of X* **output.type**=in which form we want the output	user defined
mapply()	*mapply(function,args)*	**function**=a function to apply multiple times **args**= the arguments of the function	vector
rapply()	*rapply(x,function,how)*	***x***= *a vector or an object (e.g. a list)* ***function***=*the function to apply to each element of X* **how**=output as vector or list	vector or list
tapply()	*tapply(x, index,function)*	***x***= *a vector or an object (e.g. a list)* ***index***=*a vector containing factors returns a vector*	vector

Time for an exercise

(1) Use a **for loop** to calculate the sum of the first 100 squares (i.e. 1+4+9+…+10000)

(2) Given the vector *x=c(3, 4, 253, 8, 11, 456, 23, 476, 46)*, write a **for-loop** counting the elements divisible by 23. Print out the result on the console. Make a vector containing the elements divisible by 23.

- TIP: use the operator %%

(3) Using a **for loop**, calculate the mean of the result of 1000 rolls of a die

- TIP: use sample()

Time for an exercise

(4) Write a **function** my.mean() to calculate the mean of a vector. Compare with mean(). Use system.time() to evaluate the functions. Is your function faster than mean()?

(5) Write a **function** that will return the sum of two integers.

(6) Write a **function** that given a vector $x=c(x_1, x_2, x_3, \ldots, x_n)$ will return a vector $x=c(x_1, x_2^2, x_3^3, \ldots, x_n^n)$

(7) Write a **function** that given a vector $x=c(x_1, x_2, x_3, \ldots, x_n)$ returns how many values of x are larger than the mean of x.

  (4) TIP: use which()

(8) Write a **function** my.factorial() to calculate the factorial of n. Use a for loop in the function. Write a second function my.factorial2() that uses a while loop. Calculate the factorial of 19. Check if your functions work, comparing the output of your function with the output of the already existing factorial() function.

- n!=1*2*3*…n

(9) Write a **function** that given an integer will calculate how many divisors it has (other than 1 and itself). Make the divisors appear by screen.

(10) Write a **function** simulating *n* throws of *m* dices. The function will have *n* and *m* as arguments, and return the sum of the result of all dice per throw. Use 1 die and 1000 throws as input. Plot a histogram of the output. Gradually increase the dice number in the input, keeping the throw number constant. Plot again a histogram of the output. What do you notice?

- TIP: use apply()

Time for an exercise

For the pros

Exercise1

(1) Open the file "ItRaSA_proExercise1.R".Compare the two chunks of code. What are the differences? Try to run both chunks of code. Do you notice anything?

- Tip: Use the function system.time()

(2) How could you improve the code?

(3) Which function would you use to avoid a **for loop**?

Exercise2 (try at home)

(1) You want to perform a species richness experiment. You decide to use 12 species, and you want to prepare 8 groups of 6 species. Each group shall contain each species only once. Also, given all groups together, each species shall be used *x* times. Write the code that given your species vector s=c(s_1, s_2, s_3, … s_{12}) returns you a matrix spec.group with 8 rows (one per group), and 6 columns (one per species per group), meeting the conditions described above.

(2) Write a function performing the same task having the number of species (n), the number of groups (g) and the number of species per group (s) as argument

(3) What relationship has to be there between n, g and s?

Data cleaning

- R doesn't like empty cells!- replace with "NA" [not with zero!]
- R likes no spaces in header row names, or in data cells
- The data has to be structured in columns and rows

Time for an exercise

Use the excel spreadsheet "Leaftraits NOT CLEAN.xlsx"

(1) Prepare the spreadsheet for data input in R.
(2) Save it as "LeaftraitsCleaned.txt"
(3) Read it into R and check if everything fits

The leaftraits dataset


The worldwide leaf economics spectrum

Ian J. Wright, Peter B. Reich, Mark Westoby, David D. Ackerly, Zdravko Baruch, Frans Bongers, Jeannine Cavender-Bares, Terry Chapin, Johannes H. C. Cornelissen, Matthias Diemer, Jaume Flexas, Eric Garnier, Philip K. Groom, Javier Gulias, Kouki Hikosaka, Byron B. Lamont, Tali Lee, William Lee, Jacek Oleksyn, Norifuki Osada, Hendrik Poorter, Christopher Lusk, Jeremy J. Midgley, Marie-Laure Navas, Ülo Niinemets, Pieter Poot, Lynda Prior, Vladimir I. Pyankov, Catherine Roumet, Sean C. Thomas, Mark G. Tjoelker, Erik J. Veneklaas & Rafael Villar

Wright et al. (2004) *Nature* 428: 821-827

- Plant leaves have a number of **covarying traits**, that are thought to represent a **spectrum in life histories**.

LOW LIGHT:
High leaf Mass per unit area
(Dry mass/Area)
Low N/P content
Long leaf lifespan
Low photosynthetic rate

HIGH LIGHT:
Low leaf Mass per unit area
(Dry mass/Area)
High N/P content
Short leaf lifespan
High photosynthetic rate

- Data were collected for several thousand plant species on multiple leaf traits.

The leaftraits dataset

GF= Growth Form
- **H**erb
- **T**ree
- **S**hrub
- **G**rass
- **F**ern
- **V**ine
- **Ep**iphyte

Needle_Broadlf
- **N**eedle
- **B**roadleaf
- NA

N2_fixer
- **Y**es
- **N**o

LMA= Leaf **M**ass per unit **A**rea (g m^{-2})

Pmass= Phosphorus per unit **mass** (%)

1	Species	GF	Decid_Evergreen	Needle_Broadlf	C3C4	N2_fixer	LL	LMA	Nmass	Pmass	Amass
2	Aa_sp	H	NA	NA	NA	N	NA	7.578429	1.524405	NA	NA
3	Abies_alba	T	E	N	C3	N	7.337513	9.665912	1.265577	0.428399	4.294091
4	Abies_lasiocarpa	T	E	N	C3	N	7.259212	11.08516	1.01292	NA	2.888386
5	Abronia_villosa	H	NA	B	NA	N	NA	6.384492	2.01168	NA	14.15839
6	Abutilon_theophrastii	H	NA	B	NA	N	NA	4.963246	1.81193	NA	12.69971
7	Acacia_auriculiformis	T	E	B	C3	Y	2.718282	7.555505	1.48357	0.398193	8.444699
8	Acacia_colletioides	S	E	N	C3	Y	4.829732	12.92482	1.286612	0.267467	4.090495
9	Acacia_dimidiata	S	E	B	NA	Y	NA	9.975801	1.396053	NA	NA
10	Acacia_doratoxylon	S	E	B	C3	Y	3.651128	10.38189	1.352874	0.339283	6.208478

Decid_Evergreen
- **E**vergreen
- **D**eciduous
- NA

C3C4= Photosynthetic metabolism
- **C3**
- **C4**
- NA

LL= Leaf Lifespan (months)

Nmass= Nitrogen per unit **mass** (%)

Amass= Photosynthetic capacity (nmol g^{-1} s^{-1})

Data exploration

How many data points do I have per growth form?

- table

  table(leaftraits$GF)

What are the maximum and the minimum values of Leaf Lifespan?

- *max()*

  max(leaftraits$LL)

- *min()*

  min(leaftraits$LL)

Data exploration

What are the mean values for Leaf Lifespan per Growth Form?

– *aggregate()*

Splits the data into subsets, computes summary statistics for each, and returns the result in table form.

aggregate(LL~GF, data=leaftraits, mean)

functions applied

~ (Tilda) means 'as a function of'
On Win: alt+126

```
  GF       LL
1  F 1.744211
2  G 1.727545
3  H 1.758920
4  S 3.042508
5  T 3.256022
6  V 2.396015
```

Data exploration

What are the mean values per Leaf Lifespan, for deciduous and evergreen leaf types, within Growtn Form?

- *aggregate()*

aggregate(LL~Decid_Evergreen+GF, data=leaftraits, mean)

```
   Decid_Evergreen GF       LL
1                D  G 1.560855
2                E  G 5.253869
3                D  H 1.544702
4                E  H 3.979728
5                D  S 1.799041
6                E  S 3.515714
7                D  T 2.196100
8                E  T 4.078669
9                D  V 2.177443
10               E  V 2.833160
```

Data exploration

We can calculate other **summary statistics** instead of *mean()*

- **Standard deviation** *sd()*
- **Variance** *var()*
- **Minimum** *min()*
- **Maximum** *max()*
- **Median** *median()*
- **Quantile** *quantile()*

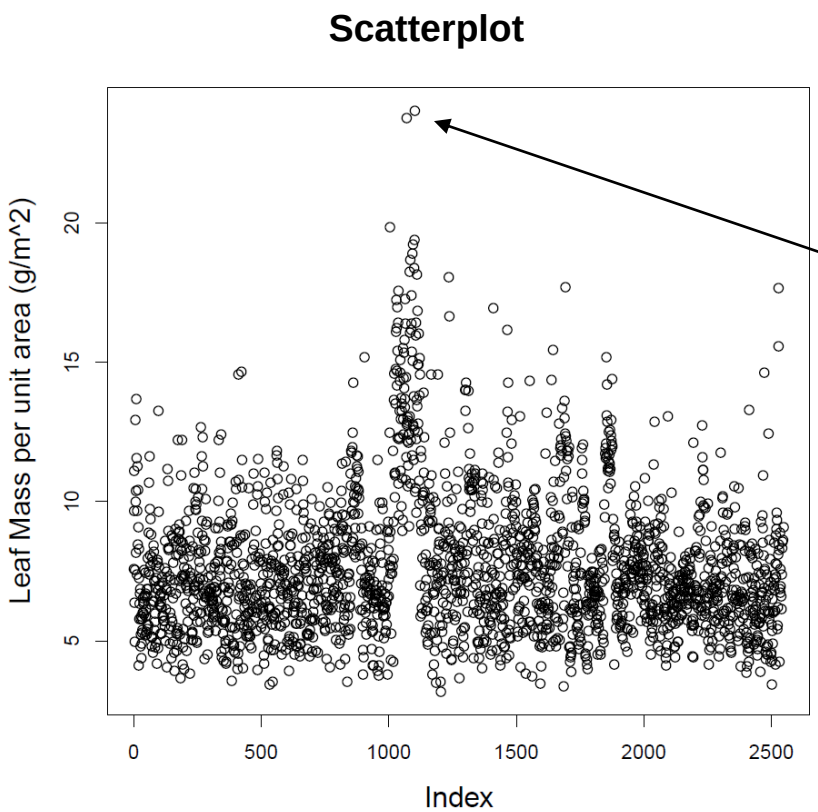
We can remove NAs subsetting, or use the argument *na.rm=TRUE*

Data exploration

What is the spread of the data like?

plot(leaftratis$LMA,ylab="Leaf Mass per unit area (g/m^2)", cex.lab=1.4)

cex = character expansion, size of label



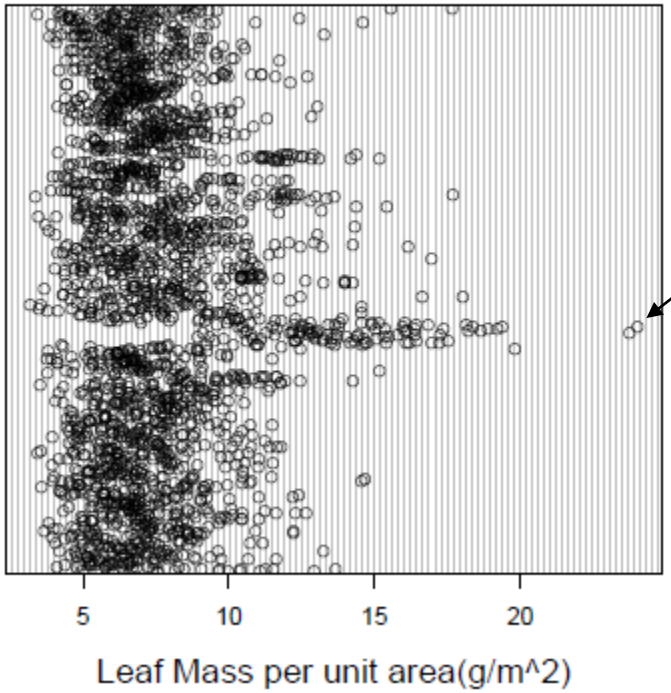
Useful for exposing outliers

Data exploration

What is the spread of the data like?

dotchart(leaftraits$LMA, ylab="Leaf Mass per unit area(g/m^2)", cex.lab=1.4)

Cleveland plot



Useful for exposing outliers

Data exploration

Looks of graphics can be easily changed:

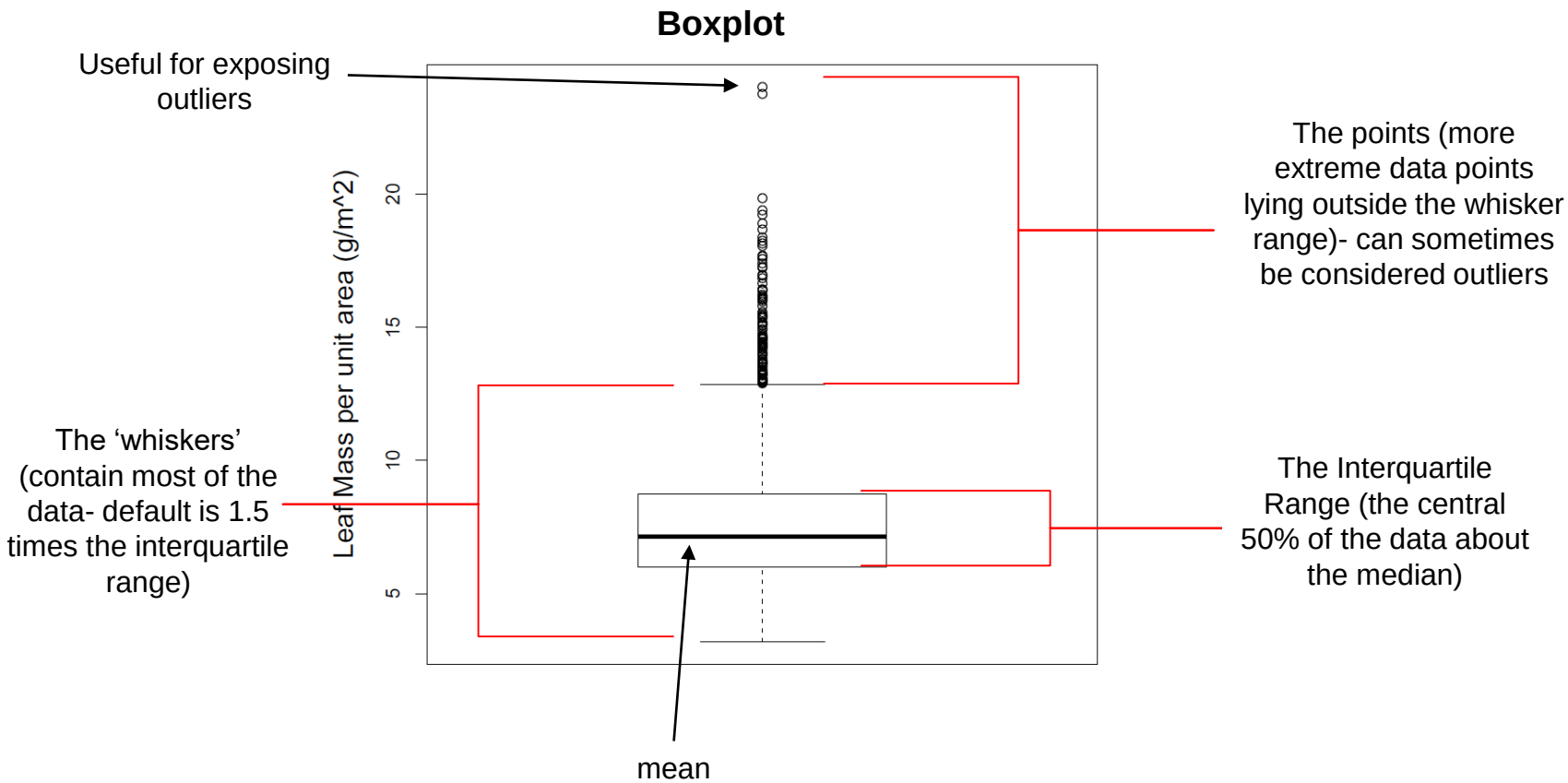
- **pch**= point character (default is 1)

- **xlab/ylab**= labels on the axes (default is variable column name)
- **cex**= point size (default is 1)
- **cex.lab**= size of axis names
- **cex.axis**=

Data exploration

What is the spread of the data like?

boxplot(leaftraits$LMA,ylab="Leaf Mass per unit area (g/m^2)", cex.lab=1.4)

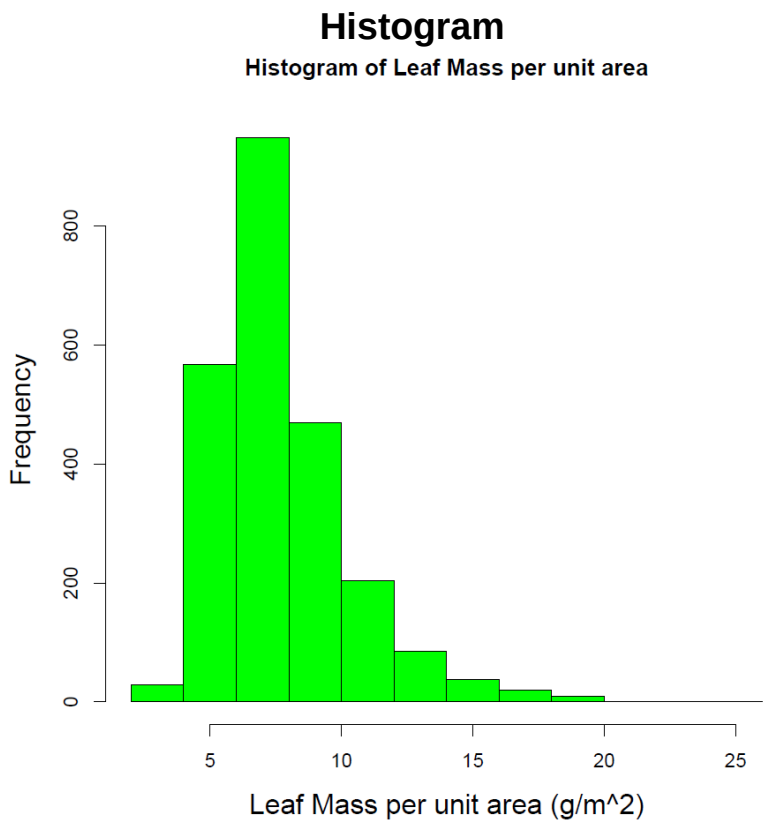


Data exploration

What is the spread of the data like?

hist(leaftraits$LMA,xlab="Leaf Mass per unit area(g/m^2)",

ylab="Frequency",cex.lab=1.4,col="green", main="Histogram of Leaf Mass per unit area")



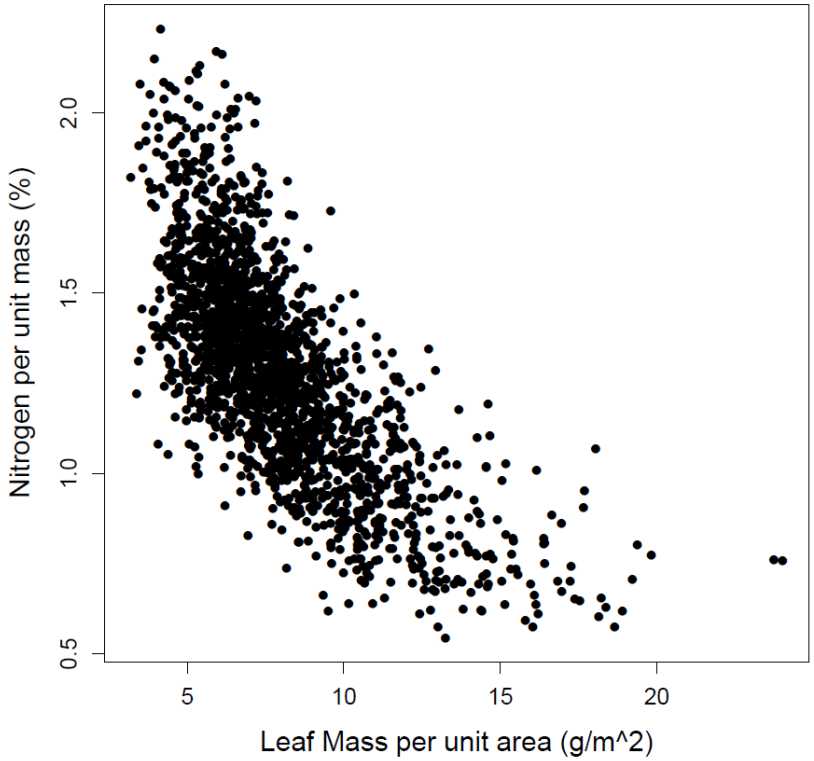
Help to identify skew (asymmetric distribution) and need for transformation

This data is positively skewed

Data exploration

What do relationships between variables look like?

plot(leaftraits$LMA, leaftraits$Nmass, xlab="Leaf Mass per unit area (g/m^2)", ylab="Nitrogen per unit mass (%) ", pch=16, cex.lab=1.4)



Time for an exercise 3

Use the text file “wingbowl.txt”. Now you will do some data exploration on this data

Some info about the data

The data contains wing length measurements of Barn owl nestlings that were either treated with a corticosterone or a placebo implant

Brood: brood id

Ring: individual id

Age1: age of the individual at the day it received the implant, in days

Implant: type of implant: C = corticosterone, P = placebo

Days: number of days after the implant

Age: age of the nestling at the day of the wing length measurement, in days

Wing: wing length measurement in mm

Time for an exercise 3

(1) Read in the data (wingbowl.txt) in R

(2) How many data points do I have per brood?

(3) What are the mean, the minimum and the maximum wing lengths?

(4) What are the mean values for wing length per implant type?

(5) What are the mean values for wing length, for brood per implant type?

(6) What are the variance and standard deviation of wing length?

(7) Look at the spread of wing length using a scatterplot. Are there any outliers?

(8) Look at the spread of the age when the implant was received using a Cleveland plot

(9) Look at the spread of wing length using a boxplot. Label the plot properly

(10) What does the relationship between the age when the implant was received and wing length look like? Does it make sense to look at this relationship? Why? What would you use instead?

(11) Look at the relationship between wing length and implant type. What type of plot do you expect?

Acknowledgements

People:

Noelie Maurel
Wayne Dawson
Fränzi Körner

International Max Planck Research School for Organismal Biology

Ecology Konstanz

Institute for Health Metrics and Evaluation

TReND

The Company of Biologists

Supported by

Development

Journal of Cell Science

Journal of Experimental Biology

Disease Models & Mechanisms

Biology Open

What is RStudio?