

R

Insight 2018

Xavier Thibert-Plante ¹

October 23, 2018

¹xavier@thibert-plante.com

Introduction

What you won't learn today

- ▶ Statistics
- ▶ Experimental design

What you will learn today

- ▶ Basic usage of high level programming
- ▶ Tool to learn more on your own

Introduction

What R is not going to do for you

- ▶ Not useful to enter data
- ▶ Will not tell you if you are using the right statistical test

What R is going to do for you

- ▶ Perform statistical test
- ▶ Plot figures

Introduction

Installing R

▶ Install Anaconda

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

▶ `> sudo R`

▶ Inside R

▶ `> update.packages()`

▶ `> install.packages(c('repr', 'IRdisplay',
'evaluate', 'crayon', 'pbdZMQ', 'devtools',
'uuid', 'digest'))`

▶ `>`

`devtools::install_github('IRkernel/IRkernel')`

▶ `> q()`

This page can take up to 30 minutes of computation

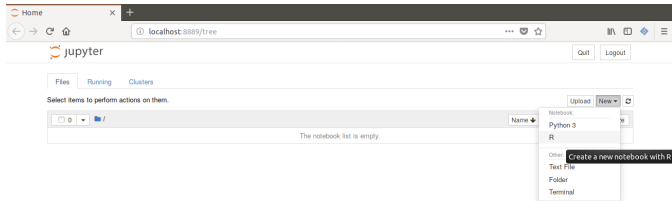
Introduction

Installing R

- ▶ Start R without the sudo from a terminal, then inside R
 - ▶ `> IRkernel::installspec()`
 - ▶ `> q()`
- ▶ You can now start you jupyter notebook (in a virtual environment if you like)

Introduction

Installing R



localhost:8889/tree#

Introduction

Quit

```
> q()
```

or

```
> quit()
```

You can save your session (variables and function and continue later)

Introduction

R graphical interface

- ▶ R is a command line interface, your mouse is useless here.
 - ▶ Advantage: batch files
 - ▶ Inconvenient: when you don't know what to type you feel pretty lonely

Variables

- ▶ One value

```
> a <- 1
```

equivalent to `a=1`

```
> 1 -> a
```

- ▶ A vector

```
> b <- c(1, 2, 3)
```

Variables

Empty

► Vector

```
> a <- array(NA,dim=10)
> a[4] <- 5
```

► Matrix

```
> b <- matrix(NA,ncol=10,nrow=30)
> b[30,3]<-1
```

Variables

Help

RTFM

```
> help(array)
> help(matrix)
```

Variables

Generating vector and matrix

- ▶ Sequence vector

```
> a<-array(seq(1,10,2))
```

- ▶ Random vector

```
> a<-array(rnorm(10,mean=15,sd=3))
```

- ▶ Sequence matrix

```
> b<-matrix(seq(1,20),ncol=2,nrow=10)
```

- ▶ Random matrix

```
> b<-matrix(runif(21),ncol=3,nrow=7)
```

Variables

Simple arithmetic operations

> $a+a$

> $a+5$

> $1+b$

> $5*a$

> $a*a$

> $a*b$

> $b*b$

> $a-a$

> $a-5$

> $a/2$

Variables

Structure

- ▶ `x` is a matrix with column and lines
 - `x[,1]` # refers to the first column
 - `x[,2]` # refers to the second column
 - `x[1,]` # refers to the first line
- ▶ `x[line,column]`

Data

Read the data

Go to my github

- ▶ https://github.com/xavierthibertplante/r_crash_course
- ▶ Pull

Data

Modifying the database

- ▶ Remove special character (#\$%&?+=-)
- ▶ Make sure that the first line is the title of the column without space (“colOne” vs “col one”)
- ▶ Save as csv (Coma Separated Variable)

Data

Entering your data

- ▶ Use excel spreadsheet
- ▶ Save as csv (coma separated variable)
- ▶ Look at the csv file in a text editor, such as Notepad
- ▶ One column must have only one type of cell: number, except for the first one, sometime.

Data

Load the data

- ▶ Open R

```
> x<-read.csv("hendryEtAl.csv")
```

- ▶ We can now play with the database with the variable x

Data

Structure

- ▶ `x` is a matrix with column and lines
 - `x[,1]` # refers to the first column
 - `x[,2]` # refers to the second column
 - `x[1,]` # refers to the first line
- ▶ `x[line,column]`

Data

Sanity check

- ▶ Number of line in the data before and after loading
`> length(x[,1])`
- ▶ Number of column:
`> length(x[1,])`

Data

Column name

- ▶ Syntax: `<variableName>$<columnName>`
`> x$Years`
instead of
`> x[,18]`
- ▶ Note that it is case sensitive:
`> x$years`
will not work

Data

Your data set in R

- ▶ First line: column name (no space or special character in the name)
- ▶ Each column is of one type
- ▶ Save as “csv”
- ▶ Look at your file in a text editor (note the separation ";" or "," and the decimal point "." or ",")
- ▶ Adapt the option of `read.csv` function

```
> help(read.csv)
```
- ▶ Load your data

```
yourName<-read.csv("fileName.csv")
```
- ▶ Test length and names of columns

One variable manipulations

Simple plot

- ▶ Reload the database

```
> x<-read.csv("hendryEtAl.csv")
```

- ▶ Histogram of the Haldanes

```
> hist(x$Haldanes)
```

- ▶ Change number of bars

```
> hist(x$Haldanes,breaks=100)
```

- ▶ Customize the position of the bars

```
> hist(...,  
breaks=seq(from=-1.2,to=0.8,by=0.1))
```

One variable manipulations

Simple plot

- ▶ Change the color of the bars
`> hist(..., col=2)`
- ▶ Change the color of the borders
`> hist(..., border=3)`

One variable manipulations

Test of normality

- ▶ Shapiro-Wilk normality test

```
> shapiro.test(x$Haldanes)
```

- ▶ Kolmogorov-Smirnov test

```
> ks.test(x$Haldanes, "pnorm",  
          mean=mean(x$Haldanes, na.rm=T),  
          sd=sd(x$Haldanes, na.rm=T))
```

- ▶ Give numbers, not its meaning

Tables

- ▶ Import a new database

```
> mydata<-read.csv("fruits.csv")
```

- ▶ Extract the information

```
> str(mydata)
```

```
> help(aggregate)
```

```
> aggregate(mydata$Fruit,  
list(mydata$State), mean)
```

```
> aggregate(mydata$Fruit,  
list(State=mydata$State), mean)
```

```
> aggregate(x=mydata$Fruit,  
by=list(State=mydata$State), FUN=mean)
```

NOT

```
> aggregate(mydata$Fruit, mydata$State,  
mean)
```

```
> help(aggregate)
```

Tables

Summary

► Data

```
> n.fruit <- aggregate(mydata$Fruit,  
  list(State=mydata$State), length)  
> mean.fruit <- aggregate(mydata$Fruit,  
  list(State=mydata$State), mean)  
> sd.fruit <- aggregate(mydata$Fruit,  
  list(State=mydata$State), sd)
```

► Built the table

```
> summary.table <- cbind(n.fruit[,2],  
  mean.fruit[,2], sd.fruit[,2])  
> summary.table
```

► Add names

```
> dimnames(summary.table) <-  
  list(n.fruit[,1], c("n", "mean", "SD"))
```

Tables

More complex

- ▶ Import a new database

```
> mydata<-read.csv("gain.csv")
```

- ▶ Extract the information

```
> str(mydata)
```

```
> aggregate(mydata$growth,  
list(mydata$experiment,mydata$food), mean)
```

- ▶ Another table format

```
> mean.growth<-tapply(mydata$growth,  
list(mydata$experiment,mydata$food), mean)
```

Barplot

- ▶ Import a new database

```
> mydata<-read.csv("fruits.csv")
```

- ▶ Extract the information

- ▶ Data

```
> n.fruit <- tapply(mydata$Fruit,  
list(State=mydata$State), length)  
> mean.fruit <- tapply(mydata$Fruit,  
list(State=mydata$State), mean)  
> sd.fruit <- tapply(mydata$Fruit,  
list(State=mydata$State), sd)
```

- ▶ Barplot > barplot(mean.fruit)

- ▶ May be better

```
> help(barplot)
```

Barplot

- ▶ Add names on the axes

```
> barplot(mean.fruit,  
xlab="Treatement",  
ylab="Fruit production",  
ylim=c(0,100))
```

- ▶ Add error bars

```
> mids<-barplot(mean.fruit,  
xlab="Treatement",  
ylab="Fruit production",  
ylim=c(0,100))  
> arrows(mids,mean.fruit+sd.fruit,  
mids, mean.fruit - sd.fruit)
```

- ▶ Almost

```
> help(arrows)
```

Barplot

```
> mids<-barplot(mean.fruit,  
xlab="Treatement",  
ylab="Fruit production",  
ylim=c(0,100))  
> arrows(mids,mean.fruit+sd.fruit,  
mids, mean.fruit - sd.fruit,  
angle=90, code=3)  
> text(mids,5, paste("N = ", n.fruit))
```

Barplot

More complex

```
> myData<-read.csv("gain.csv")
> mean.growth<-tapply(myData$growth,
list(myData$experiment,myData$food),
mean)
> sd.growth<-tapply(myData$growth,
list(myData$experiment,myData$food),
sd)
> n.growth<-tapply(myData$growth,
list(myData$experiment,myData$food),
length)
> barplot(mean.growth)
```

Almost

Barplot

Plus complexe

```
> mids<- barplot(mean.growth,beside=T,  
xlab="Food type",  
ylab="Gain",  
ylim=c(0,35),  
col=grey(c(0,0.3,0.6,1)))  
> arrows(mids, mean.growth+sd.growth,  
mids, mean.growth-sd.growth,  
angle=90, code=3, length=0.1)  
> text(mids, 2, paste(n.growth),  
col=c("white", rep("black",3)))  
> legend("topleft",  
legend=rownames(mean.growth),  
fill=grey(c(0,0.3,0.6,1)))
```

Two variables manipulations

Simple graph

```
> plot(mydata$Root, mydata$Fruit)
> plot(mydata$Root, mydata$Fruit,
      xlab="Root size",
      ylab="Fruit production")
> plot(mydata$Root, mydata$Fruit,
      xlab="Root size",
      ylab="Fruit production",
      pch=21, bg="grey", cex=2.0)
```

Intact versus eaten

Two variables manipulations

Simple graph

```
> clr<-ifelse(mydata$State == "Eaten",  
"Green","Blue")  
> plot(mydata$Root, mydata$Fruit,  
      xlab="Root size",  
      ylab="Fruit production",  
      pch=21,bg=clr,cex=2.0)  
> legend("topleft",  
      legend=c("Eaten", "Intact"), pch=21,  
      pt.bg=c("Green", "Blue"), pt.cex=2.0)
```

Two variables manipulations

Simple graph

Modifications possibles sur un graphique.

- ▶ Title of the graph

```
> plot(..., main="Title")
```

- ▶ Axes name

```
> plot(..., xlab="nameX", ylab="nameY")
```

- ▶ Size of the points

```
> plot(..., cex=2.0)
```

- ▶ Axes size

```
> plot(..., cex.lab=2.0)
```

- ▶ Axes legend size

```
> plot(..., cex.axis=2.0)
```

- ▶ Axes range

```
> plot(..., xlim=c(0,100), ylim=c(0,2))
```

Two variables manipulations

Simple graph

Add points to a graph

- ▶ One point

```
> points(x=50,y=0)
```

- ▶ Type of point

```
> points(...,pch=2)
```

- ▶ Color of point

```
> points(..., col=2)
```

- ▶ Size of point

```
> points(..., cex=2.0)
```

- ▶ Many points

```
> points(x=c(1,2,3),y=c(1,2,3))
```

Two variables manipulations

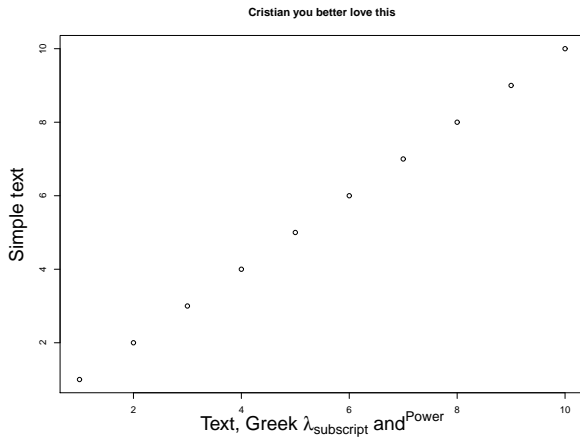
Simple graph

Add line to a graph

- ▶ One line between (x1,y1) and (x2,y2)
`> lines(x=c(x1,x2),y=c(y1,y2))`
- ▶ Line type
`> lines(..., lty=2)`
- ▶ Ligne color
`> lines(..., col=2)`
- ▶ Ligne size
`> lines(..., lwd=2.0)`
- ▶ Many lines together
`> lines(x=c(x1,x2,x3),y=c(y1,y2,y3))`

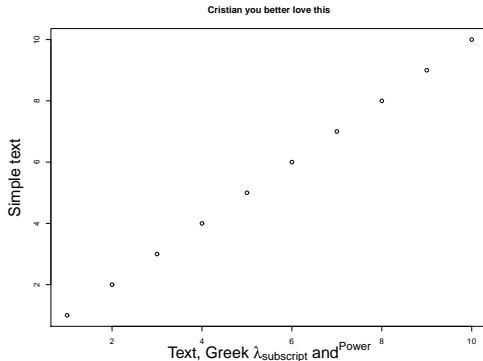
Two variables manipulations

Simple graph



Two variables manipulations

Simple graph



```
> plot(1:10, xlab=expression(paste(
  "Text, Greek ", lambda[subscript], " ",
  and^{Power})), ylab="Simple text",
  main="Cristian you better love this")
```


Save information

Figures

- ▶ **Lazy** : click file-> save as -> jpeg
- ▶ **More efficient**
 - > `jpeg("fileName.jpg")`
 - > `plot(...)`
 - > `dev.off()`
- ▶ **More option with command line**
 - > `help(jpeg)`

Save information

Figures

- ▶ Format

- ▶ postscript
- ▶ pdf
- ▶ jpeg
- ▶ png
- ▶ bmp
- ▶ tiff

- ▶ Options

- ▶ Size (width, height)
- ▶ Compression (quality)
- ▶ Pointsize (pointsize)

Save information

Variables

```
> help(save)
> save(x, y, z, file="saveXYZ.RData")
```

The whole workspace

```
> save.image(file="workspace.RData")
```

Save information

Get the information back

```
> load("saveXYZ.RData")  
> load("workspace.RData")
```

What was loaded

```
> ls()
```

Linear models

- ▶ Haldane function of generation length?
- ▶ Syntax: `x$Haldanes ~x$GLength`
`> lm(x$Haldanes ~x$GLength)`
or
`> my.lm<-lm(x$Haldanes ~x$GLength)`
- ▶ More information:
`> summary(lm(x$Haldanes ~x$GLength))`
or
`> summary(my.lm)`

Linear regression

- ▶ Extract the information from the model

```
> attributes(my.lm)
```

- ▶ Residuals

```
> my.lm$residuals
```

- ▶ Predicted values

```
> my.lm$coefficients
```

Linear Regression

- ▶ Extract more information from the model

```
> my.summ.lm<-summary(my.lm)  
> attributes(my.summ.lm)
```

- ▶ R^2

```
> my.summ.lm$r.squared
```

- ▶ F-statistics

```
> my.summ.lm$fstatistic
```

Linear models

How it looks like?

```
> plot(x$Haldanes ~x$GLength)
```

This is equivalent to

```
> plot(x$GLength, x$Haldanes)
```


Linear models

- ▶ Absolute values of Haldanes function of generation length?

```
> lm(x$HaldanesAbs ~x$GLength)
```

- ▶ More information:

```
> summary(lm(x$HaldanesAbs ~x$GLength))
```

Linear models

More factor

- ▶ Two factor

```
> m1<- x$HaldanesAbs ~x$GLength+x$Years
```

- ▶ Interaction term

```
> m2<- x$HaldanesAbs ~x$GLength:x$Years
```

- ▶ Two factor + interaction term:

```
> m3<- x$HaldanesAbs ~x$GLength*x$Years
```

equivalent to:

```
> m3<- x$HaldanesAbs ~x$GLength + x$Years +  
x$GLength:x$Years
```

ANOVA

Setup

- ▶ Look at the file `anova.txt` in a text editor

- ▶ Read a table

```
> z<-read.table("anova.txt")
```

- ▶ Give names to the column

```
> names(z) <- c("response", "category",  
"replicat", "coVar")
```

- ▶ Shortcut to column name

```
> attach(z)  
> response  
> detach(z)  
> response  
> attach(z)
```

ANOVA

Nominal term

- ▶ Everything is considered numeric as default
- ▶ Define the categorie RDexp: nominal

```
> category<-factor(category)
```

ANOVA

First steps

- ▶ **Write the model**

```
> mod1<-response ~category
```

- ▶ **Take a look at the model**

```
> boxplot(mod1)
```

- ▶ **Linear model of the data**

```
> mod1.lm<-lm(mod1)
```

- ▶ **Vizualize the model**

```
> plot(mod1.lm)
```

- ▶ **Get the information out of the model**

```
> summary(mod1.lm)
```

- ▶ **Perform the ANOVA**

```
> anova(mod1.lm)
```

ANCOVA

Models

- ▶ **Sanity check:**

```
> is.factor(category)
```

- ▶ **Look at the data:**

```
> plot(response  
~coVar, pch=as.numeric(category))
```

- ▶ **Everything in common:**

```
> ResE<-response ~coVar
```

- ▶ **Common slope, different intercept:**

```
> ResCD<-response ~category+ coVar
```

- ▶ **Full model:**

```
> ResFull<-response ~category+ coVar +  
category:coVar
```

ANCOVA

Models

► Linear model

```
> ResE.lm<-lm(ResE)
> ResCD.lm<-lm(ResCD)
> ResFull.lm<-lm(ResFull)
```

► Look at the models

```
> plot(ResE.lm)
> plot(ResCD.lm)
> plot(ResFull.lm)
```

ANCOVA

Analysis

- ▶ Get the information from the models:

```
> summary(ResE.lm)
> summary(ResCD.lm)
> summary(ResFull.lm)
```

- ▶ ANCOVA

```
> anova(ResE.lm, ResFull.lm)
> anova(ResE.lm, ResCD.lm, ResFull.lm)
```


ANCOVA

Useful command

- ▶ Verify the hypothesis of equal variance within group

```
> tapply(response, category, var,  
na.rm=TRUE)
```
- ▶ Verify the hypothesis of normality in a group

```
> tapply(response, category, function(x)  
shapiro.test(x))
```

Simple Programming

- ▶ Loops:
for
while
- ▶ Conditions:
if
- ▶ No help!

Simple Programming

Loops

```
> for (dummyVariable in array){  
>   operation  
> }
```

or

```
> while (condition){  
>   operation  
>   change the condition  
> }
```

Simple Programming

Loop example

Fibonacci

```
> x<- array(1,dim=10)
> for (i in seq(3,length(x))) {
>   x[i]<-x[i-1]+x[i-2]
> }
```

or

```
> x<- array(1,dim=10)
> i<-3
> while (i <= length(x)) {
>   x[i]<-x[i-1]+x[i-2]
>   i<-i+1
> }
```

Simple Programming

Condition

```
> if (condition) {  
>     operation 1  
> } else {  
>     operation 2  
> }
```

Simple Programming

List of conditions

>	Greater
>=	Greater or equal
<	Smaller
<=	Smaller or equal
==	Equal
!=	Not equal
&	And
	Or

Simple Programming

Condition example

```
> if (x>0) {  
>   x<-x+1  
> } else {  
>   x<-x-1  
> }
```

Simple Programming

Your first program: chaos

- ▶ Logistic equation: $x_{t+1} = r \times x_t(1 - x_t)$
- ▶ Run simulation of 1000 generations
- ▶ Set your initial population size
- ▶ Try different r values between 3 and 4
- ▶ Plot the time serie

Simple Programming

Chaos solution

```
> x<-array(dim=1000)
> x[1]=0.5
> r<-3.7
> for (t in 2:length(x)){
>     x[t]<-r*x[t-1]*(1-x[t-1])
> }
```

Functions

Your first function

You want to be able to change you parameter faster: write a function

Functions

Before

```
> x<-array(dim=1000)
> x[1]=0.5
> r<-3.7
> for (t in 2:length(x)){
>     x[t]<-r*x[t-1]*(1-x[t-1])
> }
```

Functions

After

```
> chaos <- function(time,r,popS) {  
>   x<-array(dim=time)  
>   x[1]=popS  
>   for (t in 2:length(x)) {  
>     x[t]<-r*x[t-1]*(1-x[t-1])  
>   }  
>   return(x)  
> }
```

Functions

Callign your function

```
> chaos(1000, 3.5, 0.5)
> c<-chaos(1000, 3.5, 0.5)
> plot(c)
> plot(chaos(1000, 3.5, 0.5))
```

Functions

General

All procedure that you will repeat more than once: do a function of it

Matrix operation

- ▶ Create a vector

```
> V1 <- as.vector(seq(1,10))  
> V2 <- as.vector(rnorm(10))
```

- ▶ Scalar product

```
> V3<- V1 * V2
```

- ▶ Vectorial product $((1 \times n) * (n \times 1))$

```
> s1<- t(V1) %*% V2
```

- ▶ External vectoriel product $((n \times 1) * (1 \times n))$

```
> M1<- V1 %*% t(V2)
```

Matrix operation

- ▶ Building a matrix from vectors

Line by line `M1<-rbind(V1,V2)`

Column by column `M2<-cbind(V1,V2)`

- ▶ Building matrix element by element (column major)

```
> M3 <-
```

```
matrix(c(1,2,3,4,5,6),ncol=3,nrow=2)
```

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

```
> M4 <-
```

```
matrix(c(1,2,3,4,5,6),ncol=2,nrow=3)
```

$$\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

Matrix operation

- ▶ Multiply matrix by a vector

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \begin{pmatrix} 7 \\ 8 \\ 9 \end{pmatrix}$$

- ▶

```
> M1 <- matrix(c(1,2,3,4,5,6), ncol=3,  
  nrow=2)  
> V1 <- as.vector(c(7,8,9))  
> M1 %*% V1  
NOT: > V1 %*% M1
```

Matrix operation

- ▶ Multiply matrix by matrix

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \begin{pmatrix} 7 & 10 \\ 8 & 11 \\ 9 & 12 \end{pmatrix}$$

- ▶

```
> M1 <- matrix(c(1,2,3,4,5,6), ncol=3,
nrow=2)
> M2 <- matrix(c(7,8,9,10,11,12), ncol=2,
nrow=3)
> M1 %*% M2
```

and not the opposite:

```
> M2 %*% M1
```

Matrix operation

Exercise

- ▶ Proportion of population at equilibrium (Leslie matrix)

$$T = \begin{pmatrix} 0 & 0.63 & 0.702 \\ 0.7 & 0 & 0 \\ 0 & 0.2 & 0 \end{pmatrix} P = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$



$$P_1 = TP$$

$$P_2 = TP_1$$

$$P_3 = TP_2$$

Matrix operation

- ▶ Eigenvalue and eigenvector

$$\begin{pmatrix} 0 & 0.63 & 0.702 \\ 0.7 & 0 & 0 \\ 0 & 0.2 & 0 \end{pmatrix}$$

- ▶

```
> M1 <-  
matrix(c(0,0.7,0,0.63,0,0.2,0.702,0,0),  
ncol=3, nrow=3)  
> eigen(M1)
```

Matrix operation

- ▶ Inverse, eigenvalue and diagonal

$$\begin{pmatrix} 0 & 0.63 & 0.702 \\ 0.7 & 0 & 0 \\ 0 & 0.2 & 0 \end{pmatrix}$$

- ▶

```
> M1 <-  
matrix(c(0,0.7,0,0.63,0,0.2,0.702,0,0),  
ncol=3, nrow=3)  
> solve(M1)  
> M1 %*% solve(M1)  
> det(M1)  
> diag(M1)
```

Conclusion

- ▶ RTFM

```
help(functionName)  and  
help.search("what you looking for")
```

- ▶ Text editors are your best companions

- ▶ Verify the format of your data csv and others
- ▶ To write down the command BEFORE you put them in the R console

- ▶ You will NEVER screw up your data in R if you load them from a file

Acknowledgement

► Thanks you!