

Analysing the Real-World Economic Statistics

Week 1 Introductions and discussions

Son-Kien Nguyen

kienns.research@gmail.com


```
mirror_mod = modifier_ob.  
set mirror object to mirror.  
mirror_mod.mirror_object =  
operation == "MIRROR_X":  
mirror_mod.use_x = True  
mirror_mod.use_y = False  
mirror_mod.use_z = False  
operation == "MIRROR_Y":  
mirror_mod.use_x = False  
mirror_mod.use_y = True  
mirror_mod.use_z = False  
operation == "MIRROR_Z":  
mirror_mod.use_x = False  
mirror_mod.use_y = False  
mirror_mod.use_z = True  
  
selection at the end -add  
mirror_ob.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob.name))  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
  
print("please select exactly  
-- OPERATOR CLASSES ----  
  
types.Operator):  
on X mirror to the selected  
object.mirror_mirror_x"  
mirror X"  
  
context):  
context.active_object is not
```

Please enter the following
code :

Attendance:

Outline for section 1

- 1 Introduction on R
- 2 Working with Vectors and Data Frames
- 3 Plotting
- 4 Simple Statistics Tests
- 5 Linear Regressions
- 6 Output Results

What is R?

- R is a programming language for statistical computing and graphics created by Ross Ihaka and Robert Gentleman¹.
- R is freely distributed under the terms of the GNU(General Public Licence).
 - ★ Its development and distribution are carried out by several statisticians known as the *R Development Core Team*.
- It is open-source and cross-platform (Window/Mac OS/Linux).
- R
 - is a command line programming language.
 - has many functions for statistical analysis and graphics.
 - has a very rich and active community.
- There are a large number of free packages available on [CRAN](https://cran.r-project.org/) ².

¹ Ihaka R. & Gentleman R. (1996) R: a language for data analysis and graphics. *Journal of Computational and Graphical Statistics* 5: 299-314.

² <https://cran.r-project.org/>

What is RStudio?

- RStudio is a free and open-source integrated development environment (IDE) for R.
- RStudio makes it easy for anyone to analyse data with R.
- RStudio is currently most widely used and enterprise-ready professional IDE for R.

Install R

- <https://www.stats.bris.ac.uk/R/>
 - ★ The link provides R installation for Window/Mac OS/Linux.
- Installing R is just like installing any other piece of software.
- Choose [Download R for Windows or OS X or Linux](#) → [base](#) → [Download R 4.0.5 for Windows](#)

Install RStudio

- <https://www.rstudio.com/products/rstudio/>
 - ★ Desktop version (Personal License) RStudio is enough for doctoral research.
- Create the folder RWorkShop in your “U” or “C” drive
 - ie U:\RWorkShop or C:\RWorkShop
- Return to RStudio and change the working directory to
 - *U:\RWorkShop* or *C:\RWorkShop* by one of two ways
 - select **Session** from the menu bar and then select **Set Working**
 - at the command prompt > `setwd("U:/ RWorkShop")` or `setwd("C:/ RWorkShop")`
 - press **Enter**

What is R Package

- R packages are the fundamental units of reproducible R code.
- They include reusable R functions, the documentation that describes how to use them, and sample data.
- There are over 6,000 packages available on the **Comprehensive R Archive Network**¹ (CRAN).
- This huge variety of packages is one of the reasons that R is so successful
 - ★ the chances are that someone has already solved a problem that you are working on, and you can benefit from their work by downloading their package.

¹<https://cran.r-project.org/>

click on panel "session => set working directory => choose directory"

How to Install and USE R Package

- You install them from CRAN with `install.packages('x')` ¹
- You use them in R with `library('x')`
- You get help on them with `help(package = 'x')`

¹You can also install package by clicking the “Tools” then select “install” in RStudio.

To start RStudio type this name in the magnifying glass located on the toolbar

Now have your rodent select Rstudio

RStudio Interface

The screenshot displays the RStudio interface with four main panes:

- Editor:** Contains R code for creating and analyzing a vector `x`. The code includes comments and functions like `length()`, `x[68]`, `x[12:18]`, `mean()`, and `sd()`.
- Environment:** Shows the current environment with a variable `x` of type `num` (numeric) with 100 elements.
- History:** Displays a list of executed commands, including `x <- c(18,11,19,12,20,19,13,14,17,17,14,13,15,18,19,13,18,13,18,14,12,19,12,13,19,11,14,16,12,17,11,11,20,15,19,19,12,13,19,13,13,19,15,16,19,19,19,15,17,17,13,18,16,13,18,19,13,18,11,19,17,13,13,15,14,20,11,14,19,11,14,12,13,18,11,18,16,14,11,18,16,14,19,11,18,17,18,19,11)`.
- Console:** Shows the output of the code executed in the Editor, including the length of `x` (100), the value of the 68th element (13), the values of elements 12 to 18, the mean of the data, and the standard deviation of the data.

Additional panes visible include **Files/Plots/Packages/Help** and **Environment**.

Editor: create R script files that have extension .R

Console: submit commands after the >

Environment: displays the objects create during your R session

Files: list the files in the working directory

Plots: displays plots you create and how to export them

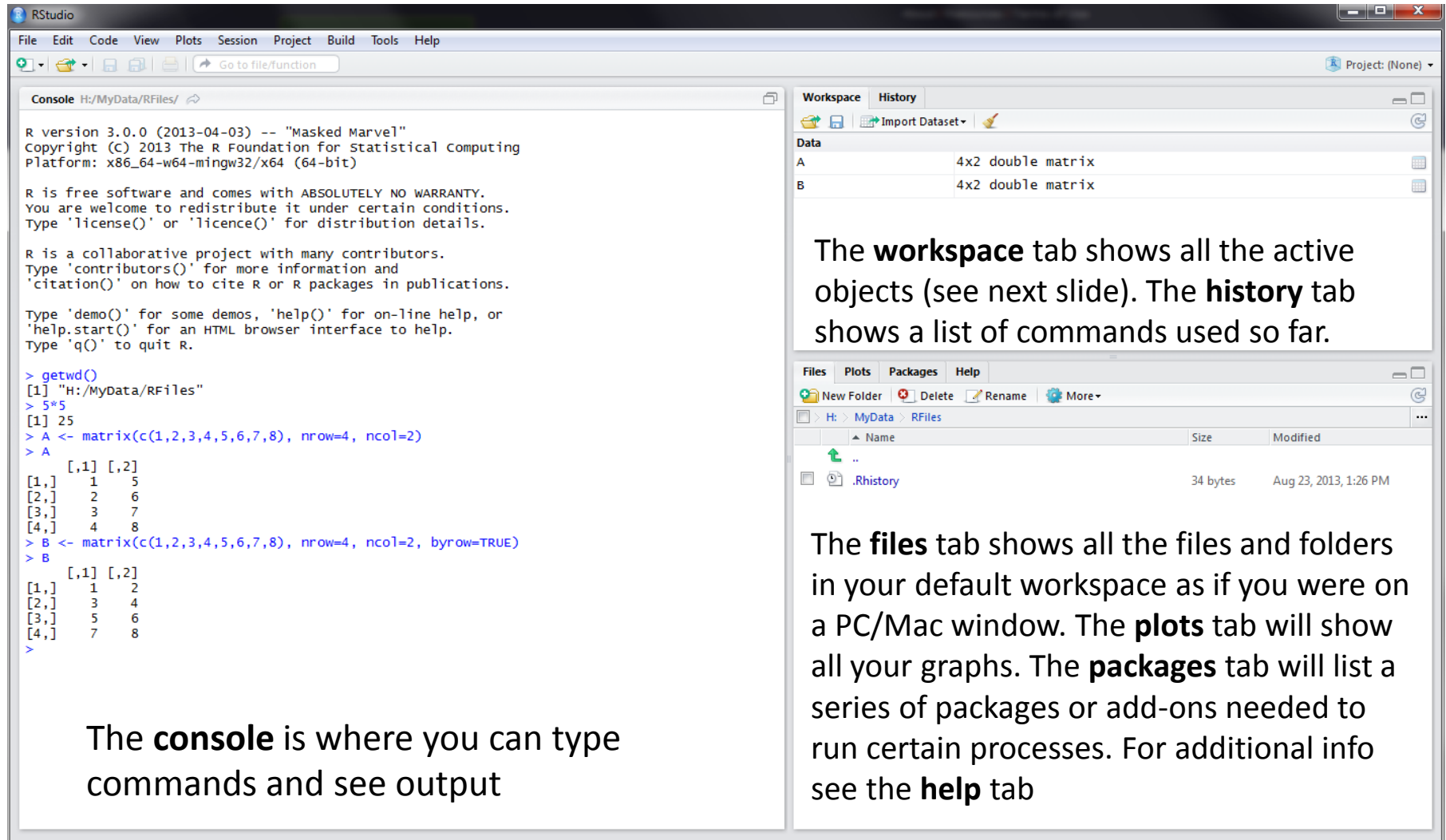
Packages: lists R packages that are available

Help: displays help when you type *?functionname* at the command prompt;

example(functionname) - provides examples how the function can be used

functionname - refers to the function you want more information on

RStudio screen



The screenshot displays the RStudio application window. The top menu bar includes File, Edit, Code, View, Plots, Session, Project, Build, Tools, and Help. Below the menu is a toolbar with icons for file operations and a search bar labeled 'Go to file/function'. The main interface is divided into four panes. The left pane is the Console, showing the R version (3.0.0), copyright information, and the output of several R commands. The top-right pane is the Workspace, displaying a table of active objects. The bottom-right pane is the Files tab, showing a file explorer view of the current workspace.

Console H:/MyData/RFiles/

```
R version 3.0.0 (2013-04-03) -- "Masked Marvel"
Copyright (C) 2013 The R Foundation for Statistical computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> getwd()
[1] "H:/MyData/RFiles"
> 5*5
[1] 25
> A <- matrix(c(1,2,3,4,5,6,7,8), nrow=4, ncol=2)
> A
      [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
> B <- matrix(c(1,2,3,4,5,6,7,8), nrow=4, ncol=2, byrow=TRUE)
> B
      [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
>
```

Workspace History

Data	
A	4x2 double matrix
B	4x2 double matrix

Files Plots Packages Help

New Folder Delete Rename More

H: > MyData > RFiles

Name	Size	Modified
..		
.Rhistory	34 bytes	Aug 23, 2013, 1:26 PM

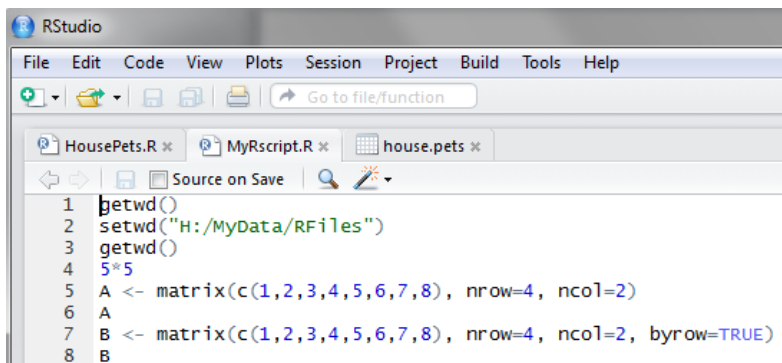
The **console** is where you can type commands and see output

The **workspace** tab shows all the active objects (see next slide). The **history** tab shows a list of commands used so far.

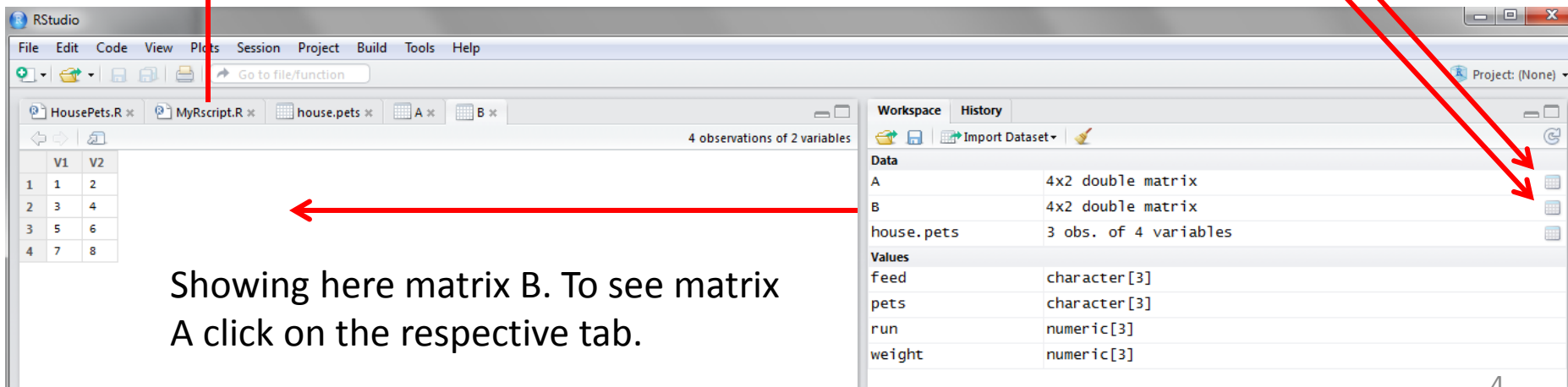
The **files** tab shows all the files and folders in your default workspace as if you were on a PC/Mac window. The **plots** tab will show all your graphs. The **packages** tab will list a series of packages or add-ons needed to run certain processes. For additional info see the **help** tab

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 betwd()
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
```



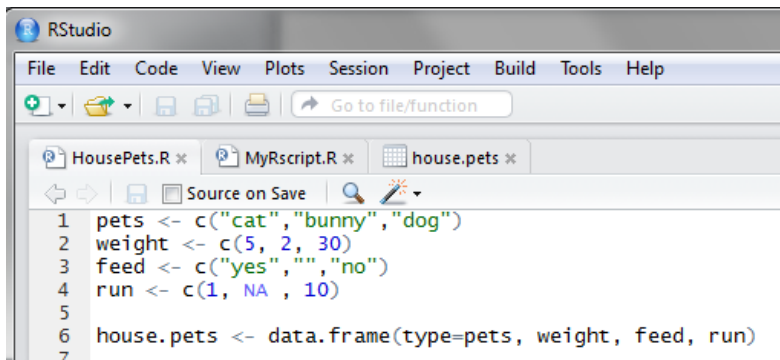
The screenshot shows the RStudio interface with the workspace and data viewer. The workspace tab is active, showing the objects A, B, and house.pets. The data viewer shows the data for house.pets, which has 3 observations of 4 variables.

	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.

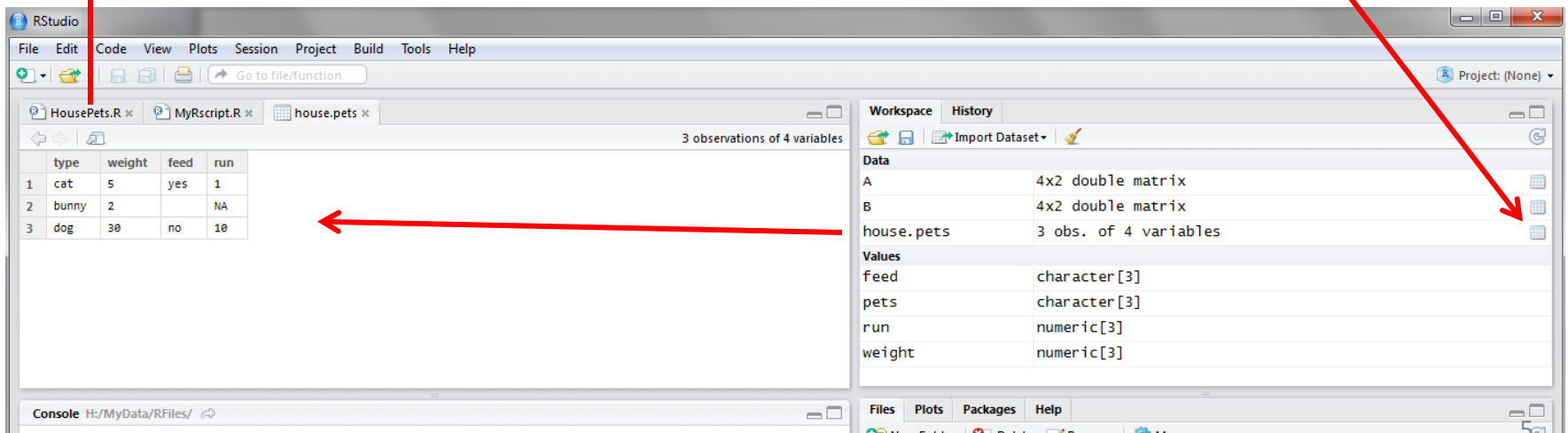
Workspace tab (2)

Here is another example on how the workspace looks like when more objects are added. Notice that the data frame `house.pets` is formed from different individual values or vectors.



```
1 pets <- c("cat","bunny","dog")
2 weight <- c(5, 2, 30)
3 feed <- c("yes","", "no")
4 run <- c(1, NA, 10)
5
6 house.pets <- data.frame(type=pets, weight, feed, run)
7
```

Click on the dotted square to look at the dataset in a spreadsheet form.



The RStudio interface shows the `house.pets` data frame in the Workspace tab. The data is displayed in a spreadsheet format with 3 observations of 4 variables.

	type	weight	feed	run
1	cat	5	yes	1
2	bunny	2	NA	NA
3	dog	30	no	10

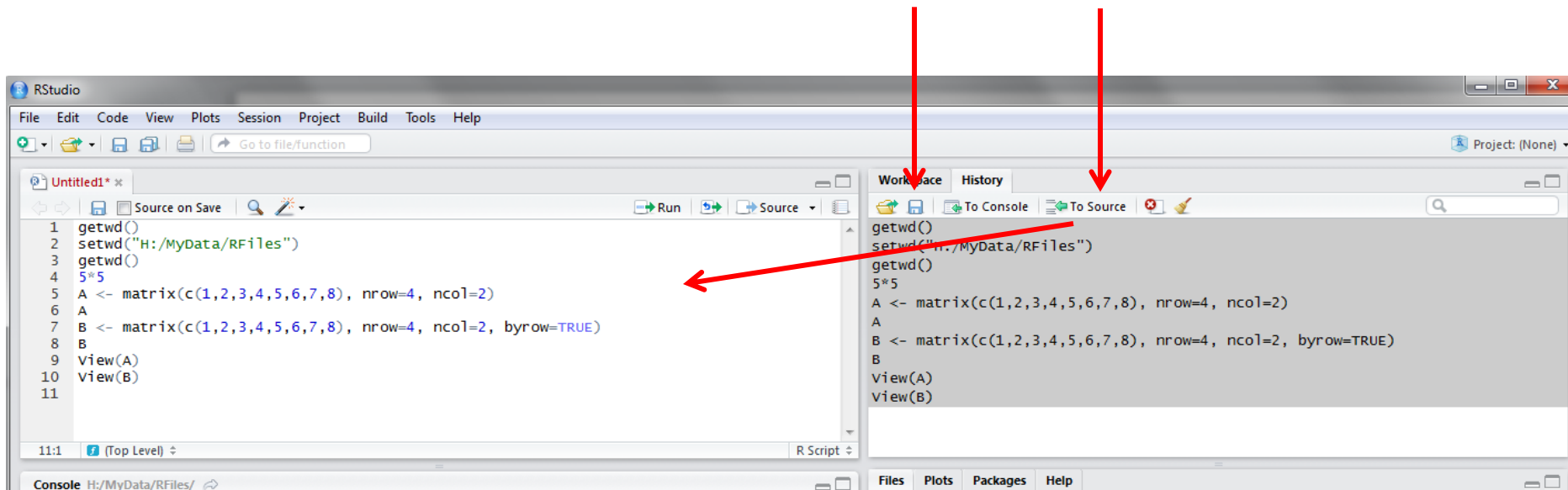
The Environment pane on the right shows the following objects:

- Data
 - A: 4x2 double matrix
 - B: 4x2 double matrix
 - house.pets: 3 obs. of 4 variables
- Values
 - feed: character[3]
 - pets: character[3]
 - run: numeric[3]
 - weight: numeric[3]

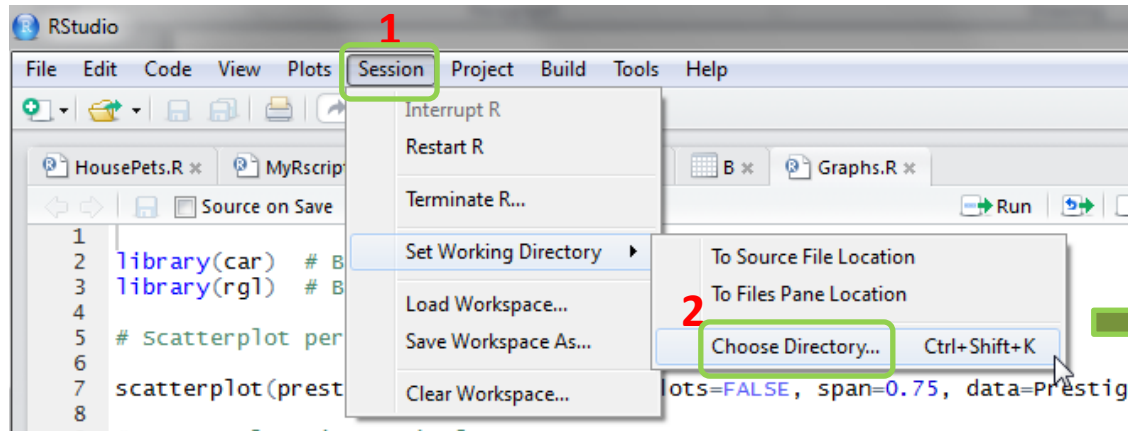
History tab

The history tab keeps a record of all previous commands. It helps when testing and running processes. Here you can either **save** the whole list or you can **select** the commands you want and send them to an R script to keep track of your work.

In this example, we select all and click on the “To Source” icon, a window on the left will open with the list of commands. Make sure to save the ‘untitled1’ file as an *.R script.



Changing the working directory



If you have different projects you can change the working directory for that session, see above. Or you can type:

```
# Shows the working directory (wd)
```

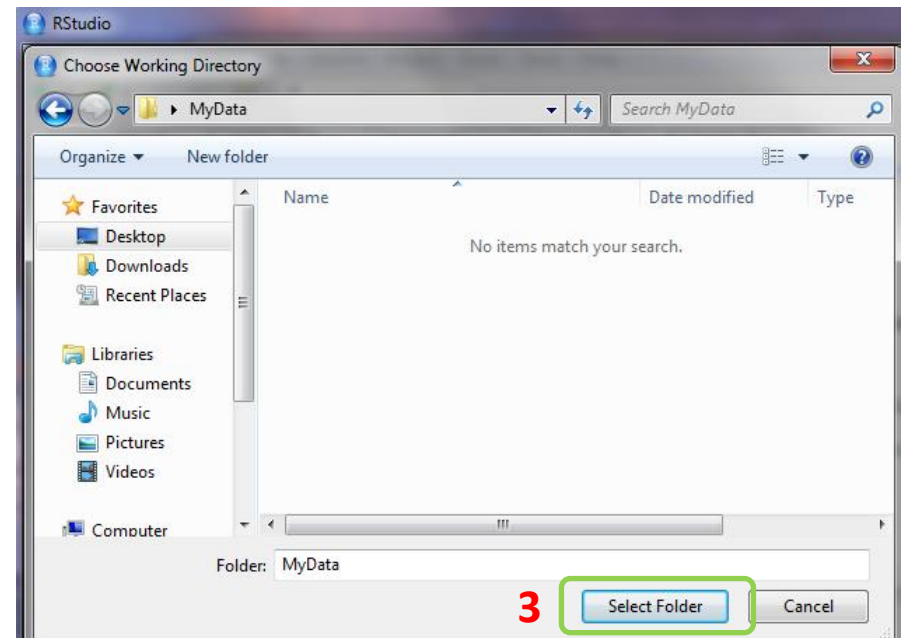
```
getwd()
```

```
# Changes the wd
```

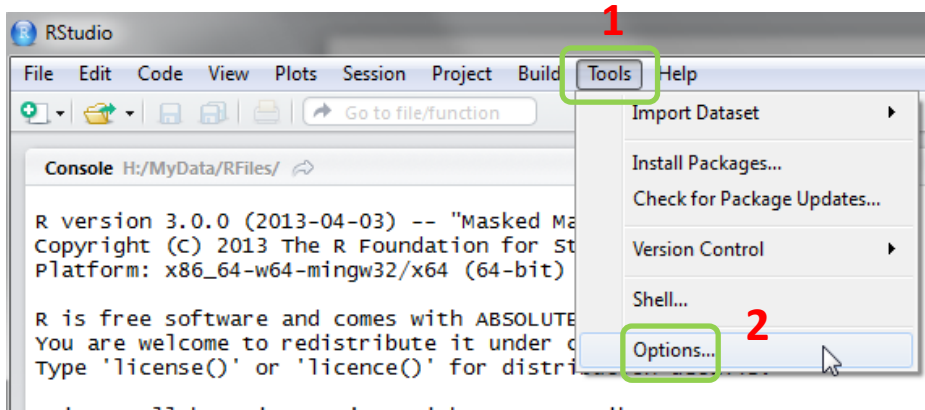
```
setwd("C:/myfolder/data")
```

More info see the following document:

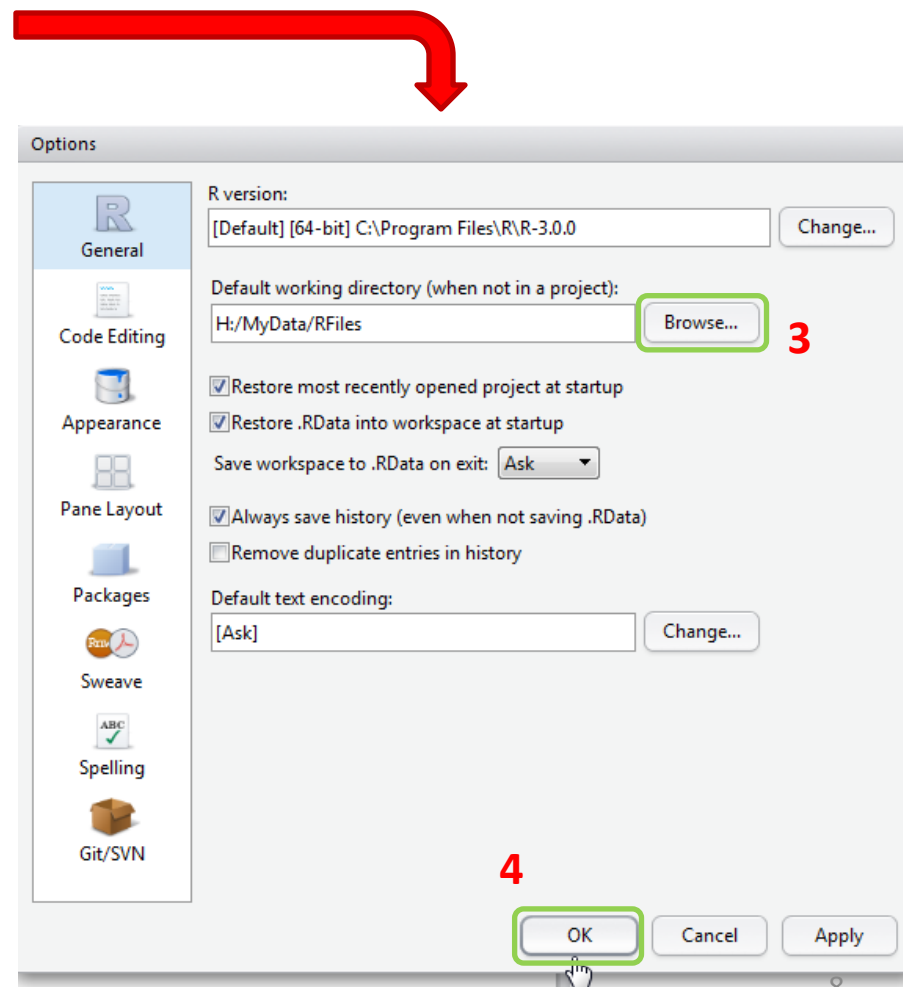
<http://dss.princeton.edu/training/RStata.pdf>



Setting a default working directory



Every time you open RStudio, it goes to a default directory. You can change the default to a folder where you have your datafiles so you do not have to do it every time. In the menu go to Tools->Options

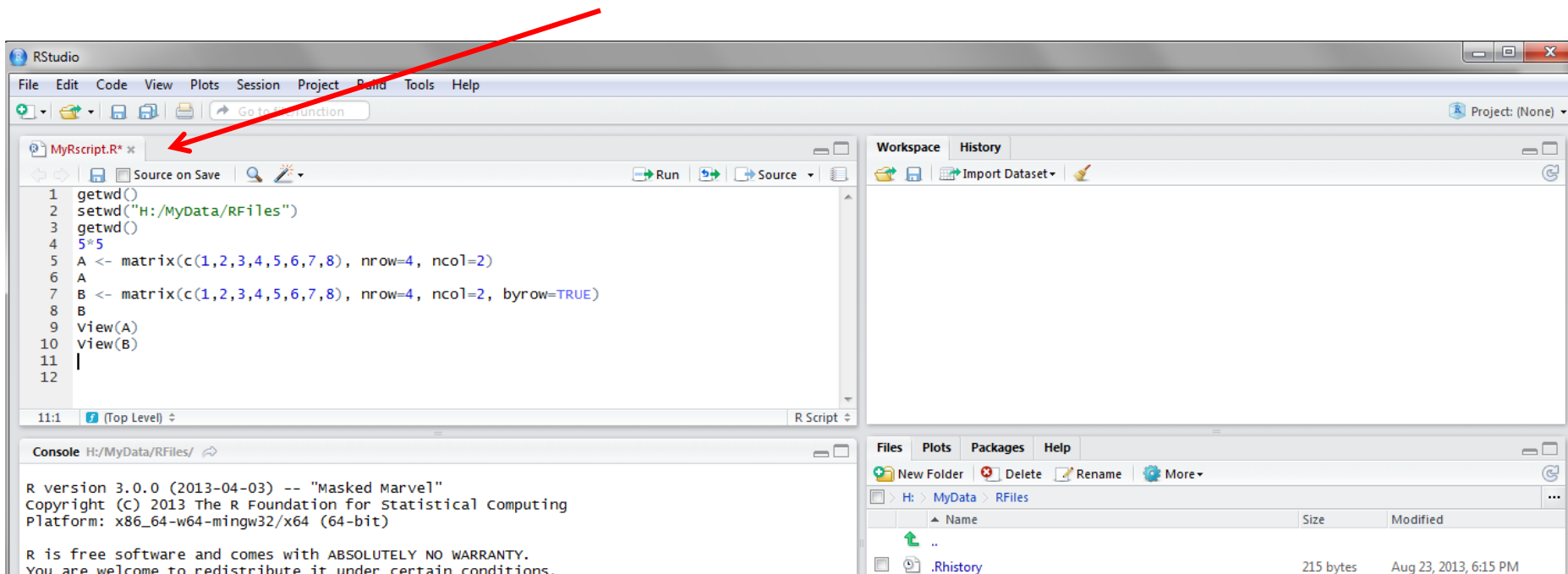


R script (1)

The usual Rstudio screen has four windows:

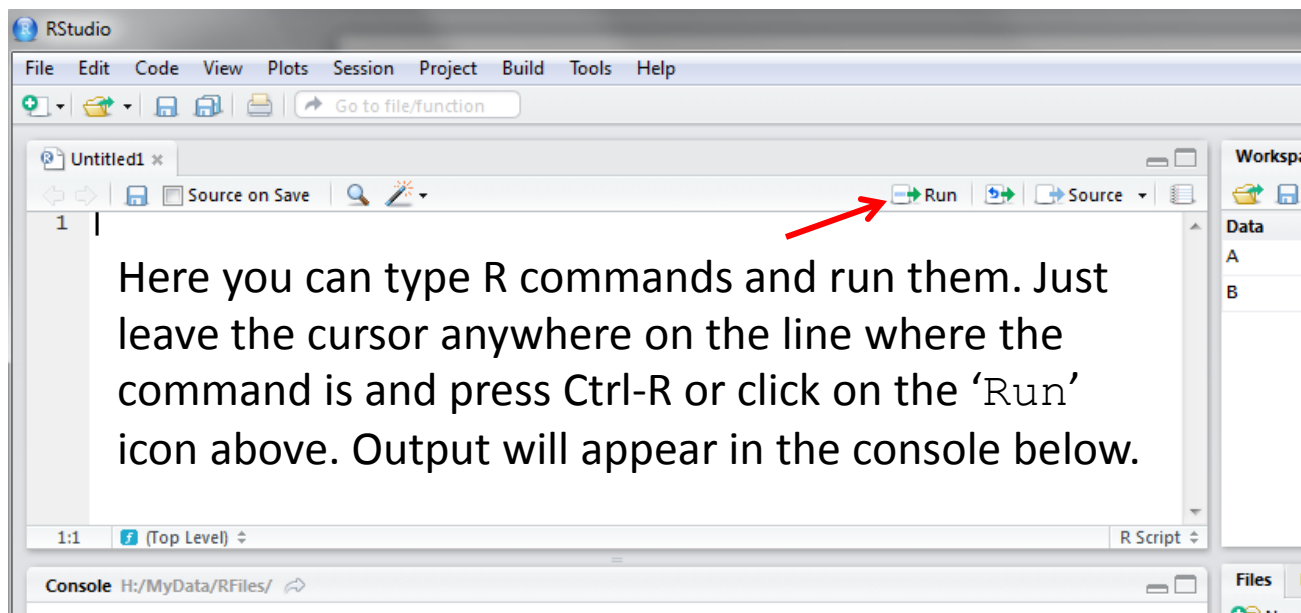
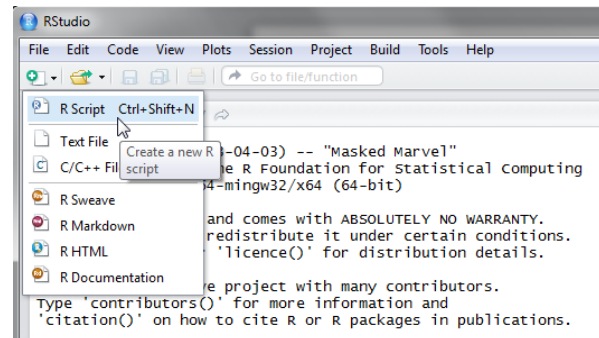
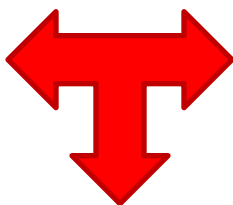
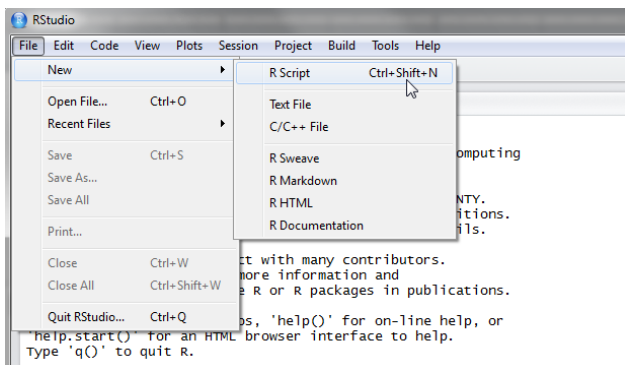
1. Console.
2. Workspace and history.
3. Files, plots, packages and help.
4. The R script(s) and data view.

The R script is where you keep a record of your work. For Stata users this would be like the do-file, for SPSS users is like the syntax and for SAS users the SAS program.



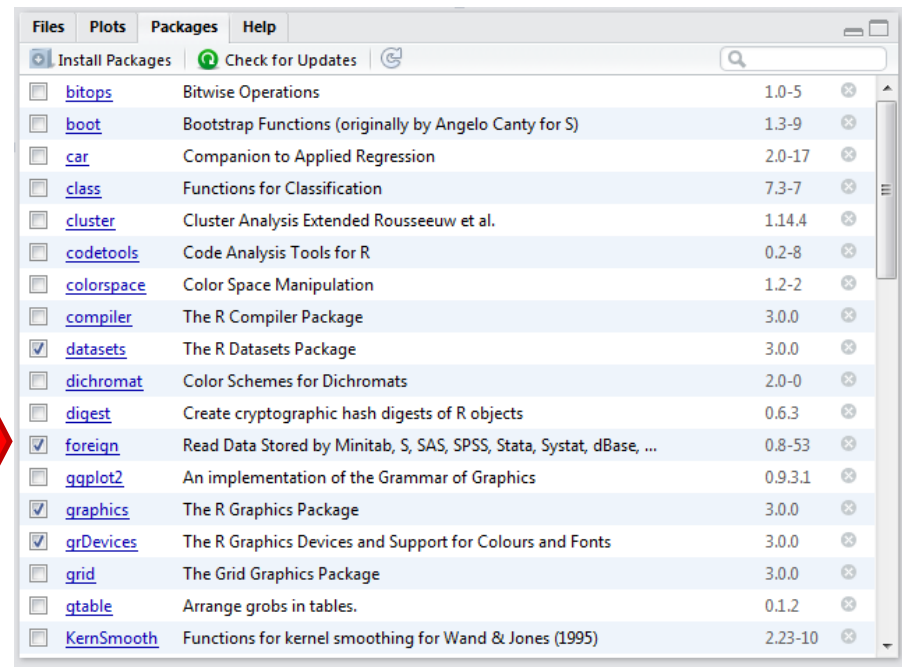
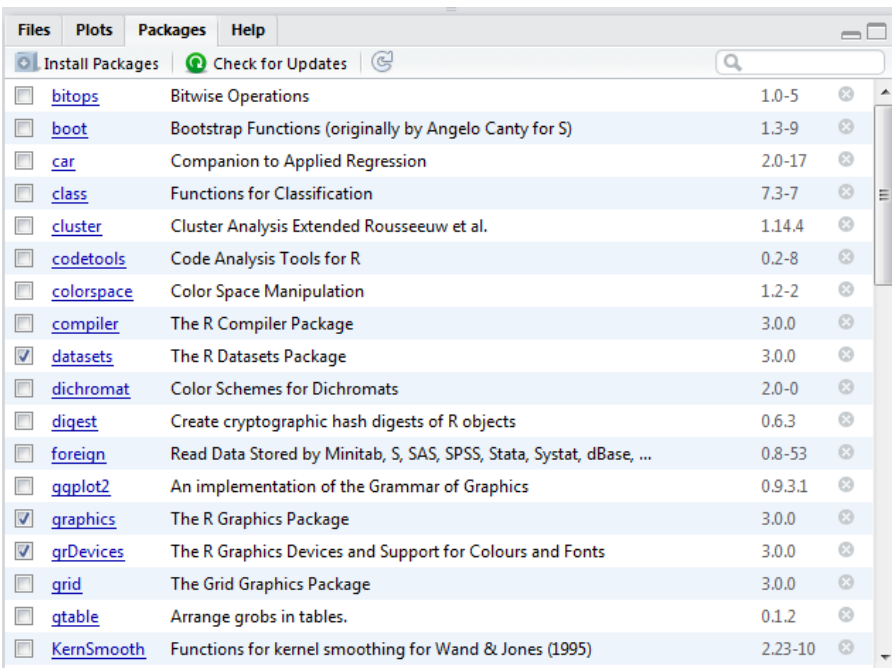
R script (2)

To create a new R script you can either go to `File -> New -> R Script`, or click on the icon with the “+” sign and select “R Script”, or simply press `Ctrl+Shift+N`. Make sure to save the script.



Packages tab

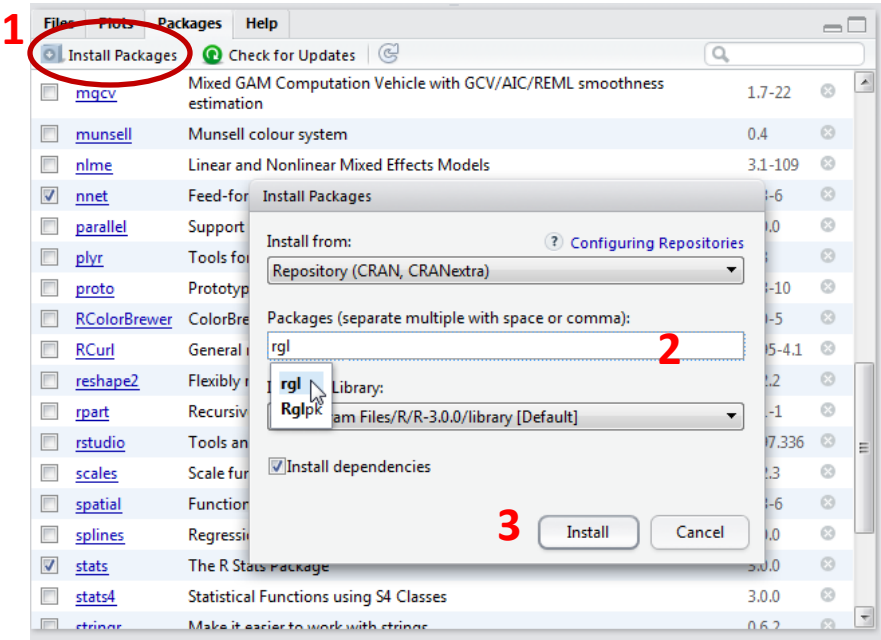
The package tab shows the list of add-ons included in the installation of RStudio. If checked, the package is loaded into R, if not, any command related to that package won't work, you will need select it. You can also install other add-ons by clicking on the 'Install Packages' icon. Another way to activate a package is by typing, for example, `library(foreign)`. This will automatically check the `--foreign` package (it helps bring data from proprietary formats like Stata, SAS or SPSS).



Installing a package

<input type="checkbox"/>	RCurl	General network (HTTP/FTP/...) client interface for R	1.95-4.1	✕
<input type="checkbox"/>	reshape2	Flexibly reshape data: a reboot of the reshape package.	1.2.2	✕
<input type="checkbox"/>	rpart	Recursive Partitioning	4.1-1	✕

Before



We are going to install the package – `rgl` (useful to plot 3D images). It does not come with the original R install.

Click on “Install Packages”, write the name in the pop-up window and click on “Install”.

After

<input type="checkbox"/>	RCurl	General network (HTTP/FTP/...) client interface for R	1.95-4.1	✕
<input type="checkbox"/>	reshape2	Flexibly reshape data: a reboot of the reshape package.	1.2.2	✕
<input type="checkbox"/>	rgl	3D visualization device system (OpenGL)	0.93.952	✕
<input type="checkbox"/>	rpart	Recursive Partitioning	4.1-1 12	✕

Plots tab (1)

RStudio interface showing the **Plots** tab. The script editor contains the following code:

```
1 library(car) # By John Fox and Sanford Weisberg
2 library(rgl) # By Daniel Adler and Duncan Murdoch
3
4 # Scatterplot per group
5
6 scatterplot(prestige ~ income|type, boxplots=FALSE, span=0.75, data=Prestige)
7
8 # Scatterplots in matrix form
9
10 scatterplotMatrix(~ prestige + income + education, span=0.7, data=Prestige)
11
12 # 3D graph, scatter3d is from the --car package. It will open in a separate window.
13
14 scatter3d(prestige ~ income + education, id.n=3, data=Duncan)
15
```

The console shows the execution of the command on line 7:

```
> scatterplot(prestige~income|type, boxplots=FALSE, span=0.75, data=Prestige)
>
```

The **Plots** tab displays a scatterplot of prestige (y-axis) versus income (x-axis), faceted by type (bc, prof, wc). The legend indicates:

- bc (black circles)
- prof (red triangles)
- wc (green plus signs)

A red arrow points from the text to the plot.

Plots tab (2)

The screenshot shows the RStudio interface with the **Plots** tab selected. The **Source** pane on the left contains R code for creating scatterplots and a 3D plot. The **Console** pane shows the execution of the first two commands. The **Plots** pane on the right displays a 3x3 grid of plots for the variables **prestige**, **income**, and **education**. Each plot shows a scatter of data points with a fitted curve and confidence intervals. A red arrow points from the console to the left arrow icon in the **Plots** tab toolbar, indicating how to navigate between plots.

```
1 library(car) # By John Fox and Sanford Weisberg
2 library(rgl) # By Daniel Adler and Duncan Murdoch
3
4 # Scatterplot per group
5
6 scatterplot(prestige ~ income|type, boxplots=FALSE, span=0.75, data=Prestige)
7
8 # Scatterplots in matrix form
9
10 scatterplotMatrix(~ prestige + income + education, span=0.7, data=Prestige)
11
12 # 3D graph, scatter3d is from the --car package. It will open in a separate window.
13
14 scatter3d(prestige ~ income + education, id.n=3, data=Duncan)
```

Console:

```
> scatterplot(prestige~income|type, boxplots=FALSE, span=0.75, data=Prestige)
> scatterplotMatrix(~ prestige + income + education, span=0.7, data=Prestige)
```

Workspace:

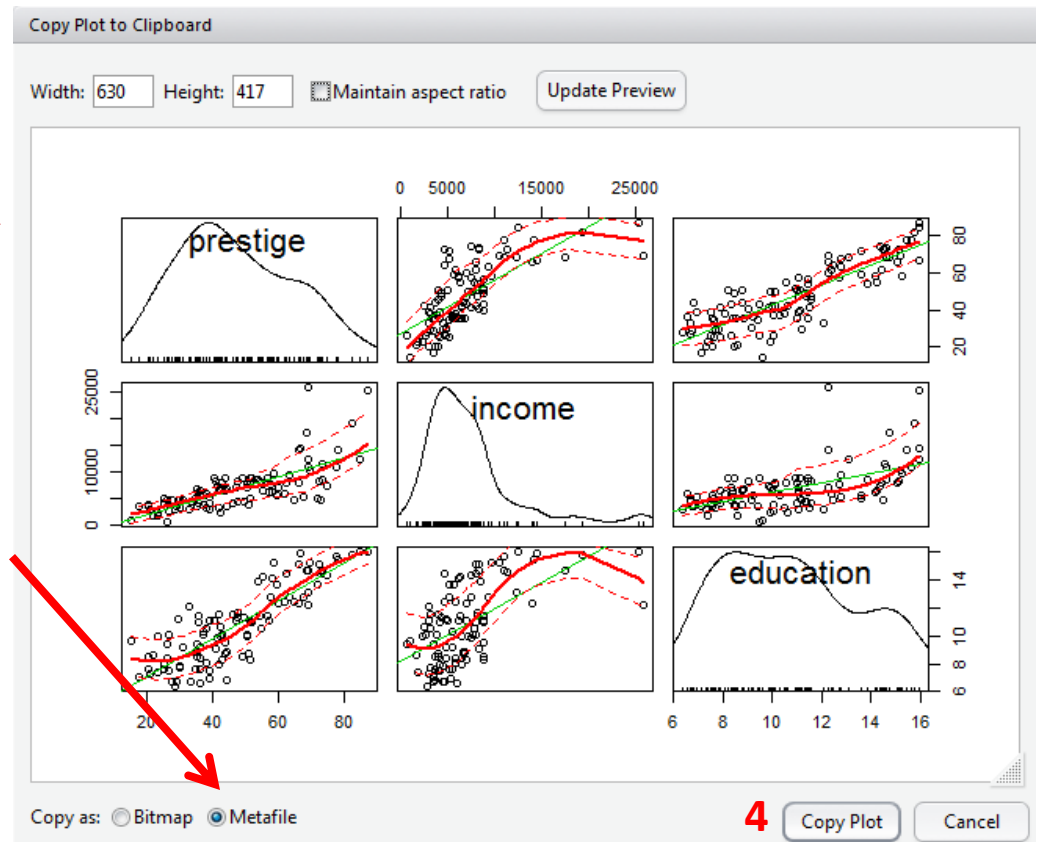
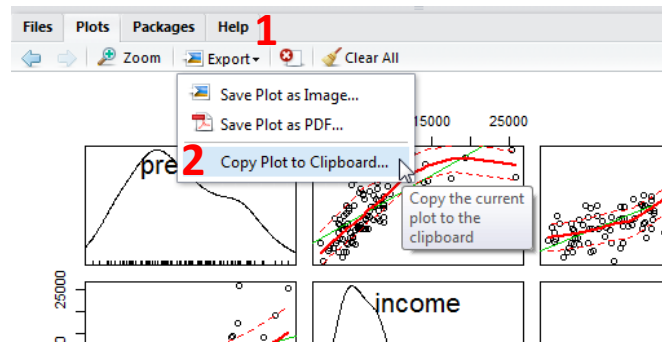
Object	Class
A	4x2 double matrix
B	4x2 double matrix
house.pets	3 obs. of 4 variables
feed	character [3]
pets	character [3]
run	numeric [3]
weight	numeric [3]

Plots tab toolbar: Files, Plots, Packages, Help, Zoom, Export, Clear All

Here there is a second graph (see line 11 above). If you want to see the first one, click on the left-arrow icon.

Plots tab (3) – Graphs export

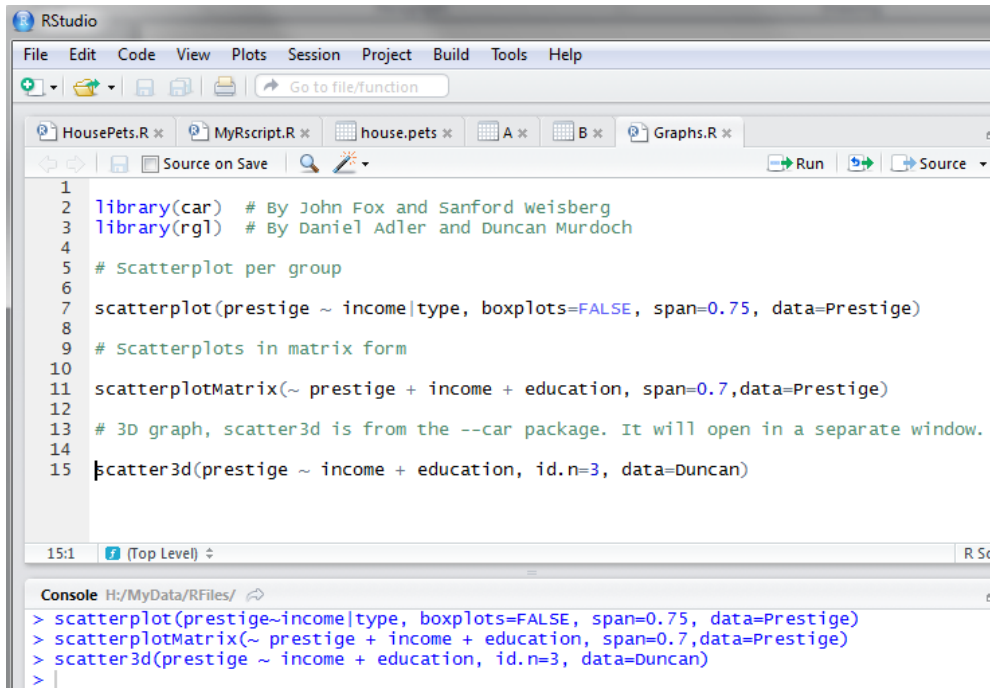
To extract the graph, click on “Export” where you can save the file as an image (PNG, JPG, etc.) or as PDF, these options are useful when you only want to share the graph or use it in a LaTeX document. Probably, the easiest way to export a graph is by copying it to the clipboard and then paste it directly into your Word document.



3 Make sure to select 'Metafile'

5 Paste it into your Word document

3D graphs

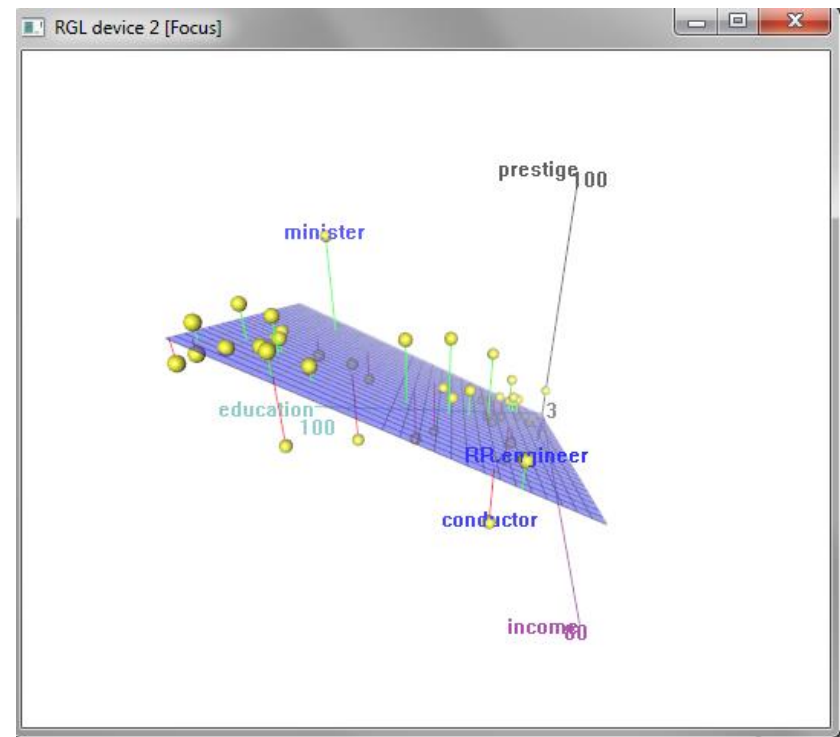


```
1 library(car) # By John Fox and Sanford Weisberg
2 library(rgl) # By Daniel Adler and Duncan Murdoch
3
4 # Scatterplot per group
5
6 scatterplot(prestige ~ income|type, boxplots=FALSE, span=0.75, data=Prestige)
7
8 # Scatterplots in matrix form
9
10 scatterplotMatrix(~ prestige + income + education, span=0.7, data=Prestige)
11
12 # 3D graph, scatter3d is from the --car package. It will open in a separate window.
13
14
15 scatter3d(prestige ~ income + education, id.n=3, data=Duncan)
```

Console H:/MyData/RFiles/

```
> scatterplot(prestige~income|type, boxplots=FALSE, span=0.75, data=Prestige)
> scatterplotMatrix(~ prestige + income + education, span=0.7, data=Prestige)
> scatter3d(prestige ~ income + education, id.n=3, data=Duncan)
```

3D graphs will display on a separate screen (see line 15 above). You won't be able to save it, but after moving it around, once you find the angle you want, you can screenshot it and paste it to you Word document.



Outline for section 2

- 1 Introduction on R
- 2 Working with Vectors and Data Frames
- 3 Plotting
- 4 Simple Statistics Tests
- 5 Linear Regressions
- 6 Output Results

Working with Vectors

click on panel "session => set working directory => choose directory

- Open R Script "working_with_vectors_CanvasWPMay19.R"

Working with Vectors: some useful questions

- > `ls()` - lists all objects created during your session
- > `rm(objectname)` - removes the object
- > If you want to keep a copy of this session you can type
 - > `sink("filename.lis")`
- If you do this nothing you submit at the prompt > will print so set
 - >`sink()`
- which turns sink off
- There are three ways to assign a value to an object
 - `y <- c(1,2,3)` - this is the way it is done in this workshop
 - `z = c(1,2,3)` - this is the easiest way
 - `assign("w",c(1,2,3))` - this is also allowed but inconvenient

Working with Vectors

- First we create the vector x
 - $> x <- c(18,11,19,12,11,16,14,11,19,18,16,19,13,12,20,20,19,13,14,17,17,14,13,15,18,19,13,18,13,18,14,12,19,12,13,19,13,14,16,12,17,11,11,20,15,19,19,12,13,19,13,13,19,15,16,19,19,19,15,17,17,13,18,16,13,18,19,13,18,11,19,17,13,13,15,14,20,11,14,19,11,14,12,13,18,11,18,18,14,11,18,18,14,19,11,18,17,18,19,11)$
 - To create x simply highlight these lines in the working_with..... and
 - Then click the Run arrow in the upper right corner of the file

Working with Vectors

- How many data points in `x`?
 `> length(x)`
- What is the value of the 68th element?
 `> x[68]`
- What are the values of elements 12 to 18?
 `> x[12:18]`
- What are the values of elements 17, 89, 39, 42?
 `> x[c(17, 89, 39, 42)]`
- What is the mean of the data?
 `> mean(x)`
- What is the standard deviation of the data?
 `> sd(x)`

Working with Vectors

- What is the variance, maximum absolute deviation, interquartile range, range, minimum, maximum, median?
 - > var(x); mad(x); IQR(x); range(x); min(x); max(x); median(x)
- What are the 0% 25% 50% 75% 100% percentiles?
 - > quantile(x)
- What the 0.025, 0.05, 0.95, 0.975 percentiles?
 - > quantile(x, probs=c(0.025, 0.05, 0.95, 0.975))
- What is the frequency distribution of the data?
 - > table(x)
- How many values are equal to 11?
 - > sum(x==11)

Working with Vectors

- How many values are equal to or less than 17?
 > `sum(x <= 17)`
- How many values are equal to 17 or equal to 20?
 > `sum(x == 17 | x == 20)`
- Which values are less than 15?
 > `x[x < 15]`

Working with Data Frames

- Open R Script “working_with_dataframes_CanvasWPMay19.R”

The data frame

Data Frames

Description

The function `data.frame()` creates data frames, tightly coupled collections of variables which share many of the properties of matrices and of lists, used as the fundamental data structure by most of R's modeling software.

Usage

```
data.frame(..., row.names = NULL, check.rows = FALSE,
           check.names = TRUE, fix.empty.names = TRUE,
           stringsAsFactors = default.stringsAsFactors())

default.stringsAsFactors() # << this is deprecated !
```

Arguments

<code>...</code>	these arguments are of either the form <code>value</code> or <code>tag = value</code> . Component names are created based on the tag (if present) or the deparsed argument itself.
<code>row.names</code>	<code>NULL</code> or a single integer or character string specifying a column to be used as row names, or a character or integer vector giving the row names for the data frame.
<code>check.rows</code>	if <code>TRUE</code> then the rows are checked for consistency of length and names.
<code>check.names</code>	logical. If <code>TRUE</code> then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names and are not duplicated. If necessary they are adjusted (by make.names) so that they are.
<code>fix.empty.names</code>	logical indicating if arguments which are "unnamed" (in the sense of not being formally called as <code>someName = arg</code>) get an automatically constructed name or rather name <code>""</code> . Needs to be set to <code>FALSE</code> even when <code>check.names</code> is false if <code>""</code> names should be kept.
<code>stringsAsFactors</code>	logical: should character vectors be converted to factors? The 'factory-fresh' default has been <code>TRUE</code> previously but has been changed to <code>FALSE</code> for R 4.0.0. Only as short time workaround, you can revert by setting options(stringsAsFactors = TRUE) which now warns about its deprecation.

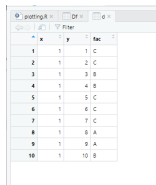
Details

A data frame is a list of variables of the same number of rows with unique row names, given class `"data.frame"`. If no variables are included, the row names determine the number of rows.

The column names should be non-empty, and attempts to use empty names will have unsupported results. Duplicate column names are allowed, but you need to use `check.names = FALSE` for `data.frame` to generate such a data frame. However, not all operations on data frames will

An example: Creating a data frame object

- > L3 <- LETTERS[1:3]
- > fac <- sample(L3, 10, replace = TRUE)
 - `sample(x, size, replace=...,prob=...)` takes a random sample from the elements of `x` with replacement if `replace=TRUE`
- > d <- data.frame(x=1,y=1:10,fac=fac)



	x	y	fac
1	1	1	C
2	1	2	C
3	1	3	B
4	1	4	B
5	1	5	C
6	1	6	C
7	1	7	C
8	1	8	A
9	1	9	A
10	1	10	B

- > d[2] - print the 2nd column

Viewing, summaries etc

- Reads in LexicalDecisions.csv file from the Working Directory and creates a dataframe
 - > Df <- read.csv('LexicalDecision.csv', header=T)
- Opens the dataframe Df
 - > View(Df)
- Show first few rows
 - > head(Df)
- Show first 10 rows
 - > head(Df, 10)
- Show first 20 rows
 - > head(Df, 20)
- Provide a summary of variables summary(Df)
 - > summary(Df)

Renaming/Adding new /deleting new variables

- Get the names of the variables
 - `names(Df)`
- Rename name of second column
 - `names(Df)[2] <- 'words'`
 `> View(Df)`
- Add new variable called loglatency that is the log of the latency
 - `Df$loglatency <- log(Df$latency)`
 `> View(Df)`
- Delete the variable that is named loglatency
 - `Df$loglatency <- NULL`
 `> View(Df)`

Discretizing continuous variables

- Create a binary variable that indicates if latency is < 500
 - > `Df$fast.rt <- Df$latency < 500`
- Cut the valence variable into three parts
 - values in $[0, 3)$ are labelled "negative"
 - values in $[3, 6)$ are labelled "neutral"
 - values in $[6, 10)$ are labelled "positive"
- Attach this new variable to the Df frame using name "valence.category"
 - > `Df$valence.category <- cut(Df$valence, breaks = c(0, 3, 6, 10), labels = c('negative', 'neutral', 'positive'))`

Subsets of the Df frame

- Rows 1 to 10, columns "subject" and "frequency"
> Df[1:10, c('subject','frequency')]]
- All rows, and just columns 1 and 3
> Df[, c(1, 3)]
- All rows, and all columns *except* 1 and 3
> Df[, c(-1, -3)]

- All rows of Df frame where the latency variable is greater than 2000
 - > `subset(Df, latency > 2000)`
- All rows of Df frame where the accuracy variable is equal to 1 (i.e. correct)
 - > `subset(Df, accuracy == 1)`
- All rows of Df frame where the accuracy variable is equal to 1 (i.e. correct)
- *AND* latency variable is less than 2000
 - > `subset(Df, accuracy == 1 & latency < 2000)`

Outline for section 3

- 1 Introduction on R
- 2 Working with Vectors and Data Frames
- 3 Plotting
- 4 Simple Statistics Tests
- 5 Linear Regressions
- 6 Output Results

Introduction to Plotting

- Open R Script “plotting_CanvasWPMay19.R”

Plotting: Histogram

Histograms

Description

The generic function `hist` computes a histogram of the given data values. If `plot = TRUE`, the resulting object of [class "histogram"](#) is plotted by [plot.histogram](#), before it is returned.

Usage

```
hist(x, ...)

## Default S3 method:
hist(x, breaks = "Sturges",
     freq = NULL, probability = !freq,
     include.lowest = TRUE, right = TRUE,
     density = NULL, angle = 45, col = "lightgray", border = NULL,
     main = paste("Histogram of" , xname),
     xlim = range(breaks), ylim = NULL,
     xlab = xname, ylab,
     axes = TRUE, plot = TRUE, labels = FALSE,
     nclass = NULL, warn.unused = TRUE, ...)
```

Arguments

x a vector of values for which the histogram is desired.

breaks one of:

- a vector giving the breakpoints between histogram cells,
- a function to compute the vector of breakpoints,
- a single number giving the number of cells for the histogram,
- a character string naming an algorithm to compute the number of cells (see 'Details').
- a function to compute the number of cells.

In the last three cases the number is a suggestion only; as the breakpoints will be set to [pretty](#) values, the number is limited to 128 (with a warning if it was larger). If `breaks` is a function, the `x` vector is supplied to it as the only argument (and the number of breaks is only limited by the amount of available memory).

freq logical; if `TRUE`, the histogram graphic is a representation of frequencies, the

Histogram

- > Df <- read.csv('LexicalDecision.csv', header=T)
- Histogram of log of latency variable
 - > hist(log(Df\$latency))
- Histogram of log of latency variable, 25 bins
 - > hist(log(Df\$latency), 25)
- Histogram of log of latency variable, 25 bin but prettier
 - > hist(log(Df\$latency), xlab='Log of Latency', ylab='Frequency',
xlim=c(5.5, 8), main='My prettier histogram', breaks=25)

Plotting with the `plot` function

Generic X-Y Plotting

Description

Generic function for plotting of R objects.

For simple scatter plots, [plot.default](#) will be used. However, there are `plot` methods for many R objects, including [functions](#), [data.frames](#), [density](#) objects, etc. Use methods (`plot`) and the documentation for these. Most of these methods are implemented using traditional graphics (the [graphics](#) package), but this is not mandatory.

For more details about graphical parameter arguments used by traditional graphics, see [par](#).

Usage

```
plot(x, y, ...)
```

Arguments

- `x` the coordinates of points in the plot. Alternatively, a single plotting structure, function or *any R object with a `plot` method* can be provided.
- `y` the y coordinates of points in the plot, *optional* if `x` is an appropriate structure.
- `...` Arguments to be passed to methods, such as [graphical parameters](#) (see [par](#)). Many methods will accept the following arguments:

`type`

what type of plot should be drawn. Possible types are

- "p" for points,
- "l" for lines,
- "b" for both,
- "c" for the lines part alone of "b",
- "o" for both 'overplotted',
- "h" for 'histogram' like (or 'high-density') vertical lines,
- "s" for stair steps,
- "S" for other steps, see 'Details' below,
- "n" for no plotting.

All other types give a warning or an error; using, e.g., `type = "punkte"` being

Plotting

- `plot(DataFrame$xaxisvar, DataFrame$yaxisvar,.....)` or
- `plot(xaxisvar ~ yaxisvar,.....,data=Name of Data frame)`
- `abline(a,b,.....)` adds one or more straight lines through
 - current plot with intercept a and slope b
- the `abline(reg.....,)`
 - where `reg` is `lsfit(c(x1,x2,...),depvar)` or `lm(formula)` where `formula` is `depvar ~ xvar1+xvar2....,data=DataFrame` and `xvar1....` refer to variables in `DataFrame`

An example....

- > Plot latency as a function of frequency
 - > `plot(latency ~ frequency, data=Df)`
- > Plot latency as a function of log frequency
 - > `plot(latency ~ log(frequency), data=Df)`
- > Plot latency as a function of valence but with labels etc
 - > `plot(latency ~ log(frequency), xlim=c(-1, 6), ylim=c(0, 1500),
xlab='Log of word frequency', ylab="Reaction time (seconds)",
main='Predicting reaction times', pch=16, cex=0.5, col='blue',
data=Df)`
 - > `abline(lm(latency ~ log(frequency), data=Df), col='red', lwd=3)`
- For more information on how to set/query graphical parameters use the `par()` function
 - type `?par` at the command prompt for information on `pch`, `col`, `lwd`
`cex` (takes numerical value magnifying plotting text and symbols)

ggplot function.....

- Install package
 - `install.packages('ggplot2')`
- Load *ggplot*
 - `library(ggplot2)`

Plotting

Description

`ggplot()` initializes a `ggplot` object. It can be used to declare the input data frame for a graphic and to specify the set of plot aesthetics intended to be common throughout all subsequent layers unless specifically overridden.

Usage

```
ggplot(data = NULL, mapping = aes(), ..., environment = parent.frame())
```

Arguments

<code>data</code>	Default dataset to use for plot. If not already a data.frame, will be converted to one by fortify() . If not specified, must be supplied in each layer added to the plot.
<code>mapping</code>	Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot.
<code>...</code>	Other arguments passed on to methods. Not currently used.
<code>environment</code>	DEPRECATED. Used prior to tidy evaluation.

Details

`ggplot()` is used to construct the initial plot object, and is almost always followed by `+` to add component to the plot. There are three common ways to invoke `ggplot`:

An example using ggplot

- `ggplot()` creates the outline of the plot used by the following R functions:
 - `geom_point()` - add points
 - `stat_smooth()` - lines
 - `coord_cartesian()` - coordinates
- Scatter plot
 - > `ggplot(Df, aes(x=log(frequency), y=latency)) + geom_point()`
- Scatter plot with line of best fit plus error with tighter axes
 - > `ggplot(Df, aes(x=log(frequency), y=latency)) + geom_point(size=2, shape=5, colour='blue') + stat_smooth(method='lm', formula=y~x) + coord_cartesian(xlim = c(0, 5), ylim=c(250, 1000))`
- Create a density for latency
 - > `ggplot(Df,aes(latency)) + stat_density(kernel = 'gaussian')`
- For more information on `ggplot` type
 - > `vignette("ggplot2-specs")` at the command prompt

Making scatter plot

- Load a new data set `load('beautyeval.Rda')`
 - > `load('beautyeval.Rda')`
- Make scatterplot point colour by male/female
 - > `ggplot(beautyeval, aes(x=beauty, y=eval, colour=sex)) +
geom_point(size=3) + scale_color_manual(values=c('blue', 'red')) +
scale_y_continuous(name='Teaching evaluation score') +
scale_x_continuous(name='Attractiveness rating')`
- Scatterplot with lines of best fit and errors
 - > `ggplot(beautydata, aes(x=beauty, y=eval, colour=sex)) +
stat_smooth(method='lm', formula=y~x) + geom_point(size=3) +
scale_color_manual(values=c('blue', 'red')) +
scale_y_continuous(name='Teaching evaluation score') +
scale_x_continuous(name='Attractiveness rating') +
theme(axis.text=element_text(size=14),
axis.title=element_text(size=16, face="bold"))`

Some comments on the scale...and theme()

- `scale_x_continuous()` - are the default scales for continuous x
 - same applies to y
- `theme()` - customize the non-data components of plots
 - can change
 - `axis.title`,
 - `axis.title.x`
 - `axis.text`
 - `axis.text.y`
 - these are set equal to `element_text()` where the size of the title or text are changed using by setting `size=#`

Outline for section 4

- 1 Introduction on R
- 2 Working with Vectors and Data Frames
- 3 Plotting
- 4 Simple Statistics Tests**
- 5 Linear Regressions
- 6 Output Results

Simple Statistics Tests (e.g. t-tests, Chi-sq test, etc)

- Open R Script “simple_stats_CanvasWPMay19.R”

Statistics

- Generate Data from Standard Normal Distribution
 - > set.seed(101)
 - > x <- rnorm(10)
 - > y <- rnorm(12)
 - **rnorm**(number of points, mean = #, sd = #)
- Independent samples t-test
 - > t.test(x, y)
- or, if we assume variance equal,
 - > t.test(x, y, var.equal=T)
- Save the output
 - > M <- t.test(x, y, var.equal=T)
 - object M is a dataframe
- This will give us access to the t-statistics
 - > M\$statistic
- To see the elements of M then
 - > names(M)

- The degrees of freedom
 - > M\$parameter
- The p-value
 - > M\$p.value
- A one sample t-test is t.test(x)
 - > t.test(x)

Binomial test

- > set.seed(103)
- > N <- 100
- Randomly samples from Heads and Tails N Times
 - > coin.flips <- sample(c('Heads', 'Tails'), size=N, replace=TRUE)
- Number of heads
 - > n.heads <- sum(coin.flips=='Heads')
- Binomial test that the probability of success $p = 0.5$
 - > binom.test(n.heads,N,p=0.5)
- Binomial test that the probability of success $p = 0.25$
 - > binom.test(n.heads,N,p=0.25)

Outline for section 5

- 1 Introduction on R
- 2 Working with Vectors and Data Frames
- 3 Plotting
- 4 Simple Statistics Tests
- 5 Linear Regressions
- 6 Output Results

Linear Regressions

- Open R Script "linear_models_CanvasWPMay19.R"

R packages: MASS (functions and datasets to Venables & Ripley); car (datasets); psych (experimental psychology functions)

- Install the packages
 - > `install.packages('MASS')`
 - > `install.packages('car')`
 - > `install.packages('psych')`
 - > `install.packages('carData')`
- Load them
 - > `library(MASS)`
 - > `library(car)`
 - > `library(psych)`
 - > `library(carData)`

more linear regression

- > read.csv('sat.csv',header=T)
- > Df<sat
- > Convert gender to a factor
 - > Df\$gender=factor(Df\$gender)
- Simple linear regression
 - > M <- lm(ACT ~ education, data=Df)
 - > summary(M)
- Simple plot and add the estimated regression line to the plot
 - > plot(ACT ~ education, data=Df)
 - > abline(M, lwd=3, col='red')

- Diagnostics: plots such as qq-plot, leverage etc...
 - > plot(M)
- Confidence intervals
 - > confint(M)
- Combine coefficients table with confidence intervals
 - > obj<-cbind(summary(M)\$coefficients, confint(M))
- Make predictions using estimated regression
 - > hypothetical.data <- data.frame(education = c(1, 2, 5, 10, 15))
 - > predict(M, newdata=hypothetical.data)
 - > obj
- Multiple linear regression
 - > M <- lm(ACT ~ education + age + gender, data=Df)
 - > summary(M)

- Load data
 - > `load('beautyeval.Rda')`
- Model 1: evaluation score as a function of beauty
 - > `M.1 <- lm(eval ~ beauty, data=beautydata)`
- Model 2: evaluation score as a function of beauty and gender
 - > `M.2 <- lm(eval ~ beauty + sex, data=beautydata)`
- Model 2: evaluation score as a function of beauty, gender and their interaction
 - > `M.3 <- lm(eval ~ beauty+sex+beauty * sex, data=beautydata)`
- Is M.2 better than M.1
 - > `anova(M.1, M.2)`
- Is there an interaction?
 - > `anova(M.2, M.3)`
- *anova(object,.....)* - compute analysis of variance tables for one or more fitted models
 - object contains results returned from a model fitting function ie *lm* or *glm*

Outline for section 6

- 1 Introduction on R
- 2 Working with Vectors and Data Frames
- 3 Plotting
- 4 Simple Statistics Tests
- 5 Linear Regressions
- 6 Output Results

Output Results

- Output to clipboard
- Output to CSV
- Output to Excel
- Output to \LaTeX

Output Results

- Open R Script “output_results_CanvasWPMay19.R”

Output results

- Output to clipboard
 - > `install.packages("clipr")`
 - > `library(clipr)`
- Generate artificial results and put them in a matrix call Res
 - > `Res <- matrix(1:9, nrow = 3, ncol=3)`
- Output to Clipboard so it can be copied to an open document using your rodent
 - > `write_clip(Res, sep="\t")`
 - Now open a MS Word document and use your rodent to paste
- Give Names to Rows and Columes
 - > `rownames(Res) <- c('Row1', 'Row2', 'Row3')`
 - > `colnames(Res) <- c('Col1', 'Col2', 'Col3')`
 - > `Res`
- Output to CSV
 - > `write.csv(Res, file = 'Res.csv')`

Output....

- Output to Excel

- > `install.packages('xlsx')`
- > `library(xlsx)`
- > `write.xlsx(Res,file = 'Res.xlsx',caption = "")`
 - `write.xlsx()` won't work if you use latest version of RStudio, works in previous release

- Output to LaTeX

- > `install.packages('xtable')`
- > `library("xtable")`
- > `outtable <- xtable(Res,caption='This is a Table',label='tableref',align=c("r|","c|","c" "l|"))`
- > `print(outtable,file='outtable.tex',append=F,table.placement='!ht',caption.placement='bottom', hline.after=seq(from=-1,to=nrow(outtable),by=1))`

xtable package

- `xtable`(R object, caption = 'Something', label = 'something', align=*character vector indicating alignment of information in columns ie c("r|","c|","c","l")*)
 - For types of R objects type `methods(xtable)` at the command prompt
- `print`(object created by `xtable`, outfile.tex, append =F, placement='!ht', caption.placement = 'top', hline.after=seq(from=-1,to=row(outtable)))