



Clase 1: Introducción al programa

Mg. Gloria Rivas

Agenda

1. Descargando e instalando el software

2. Qué nos ofrece este software?

3. Lo que debemos saber

4. Manipulación de datos

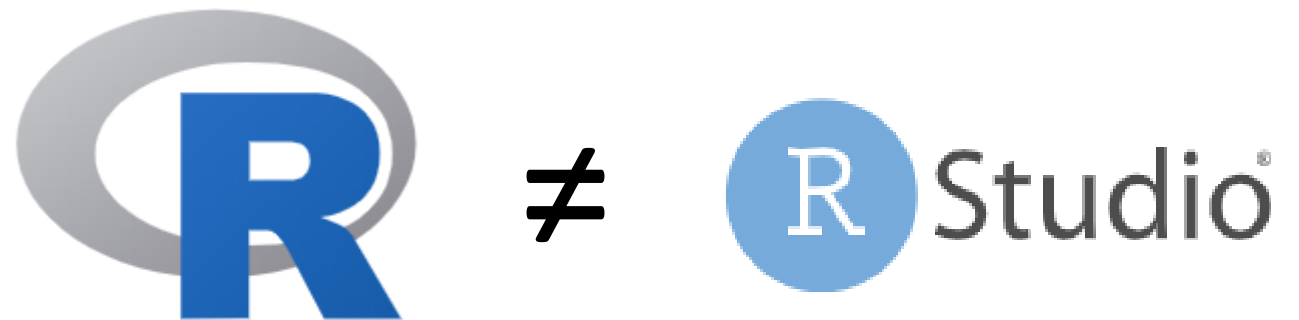
Agenda

1. Descargando e instalando el software

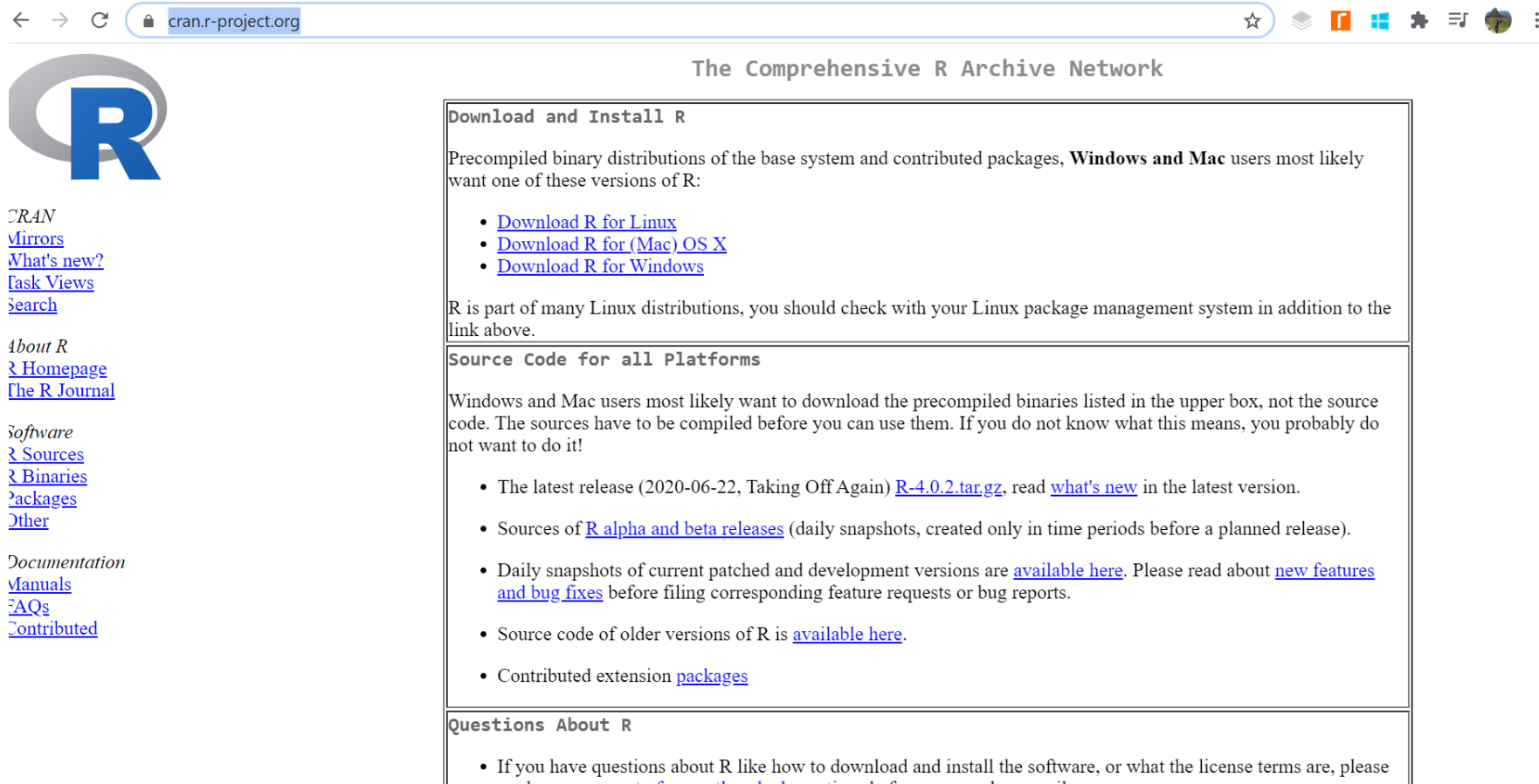
Descargando e instalando el software

Pasos para la descarga:

1. Descarga R
2. Instala R
3. Descarga RStudio
4. Instala RStudio



Descargando e instalando el software



The screenshot shows the CRAN website with the following content:

The Comprehensive R Archive Network

Download and Install R

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

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

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

Source Code for all Platforms

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

- The latest release (2020-06-22, Taking Off Again) [R-4.0.2.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

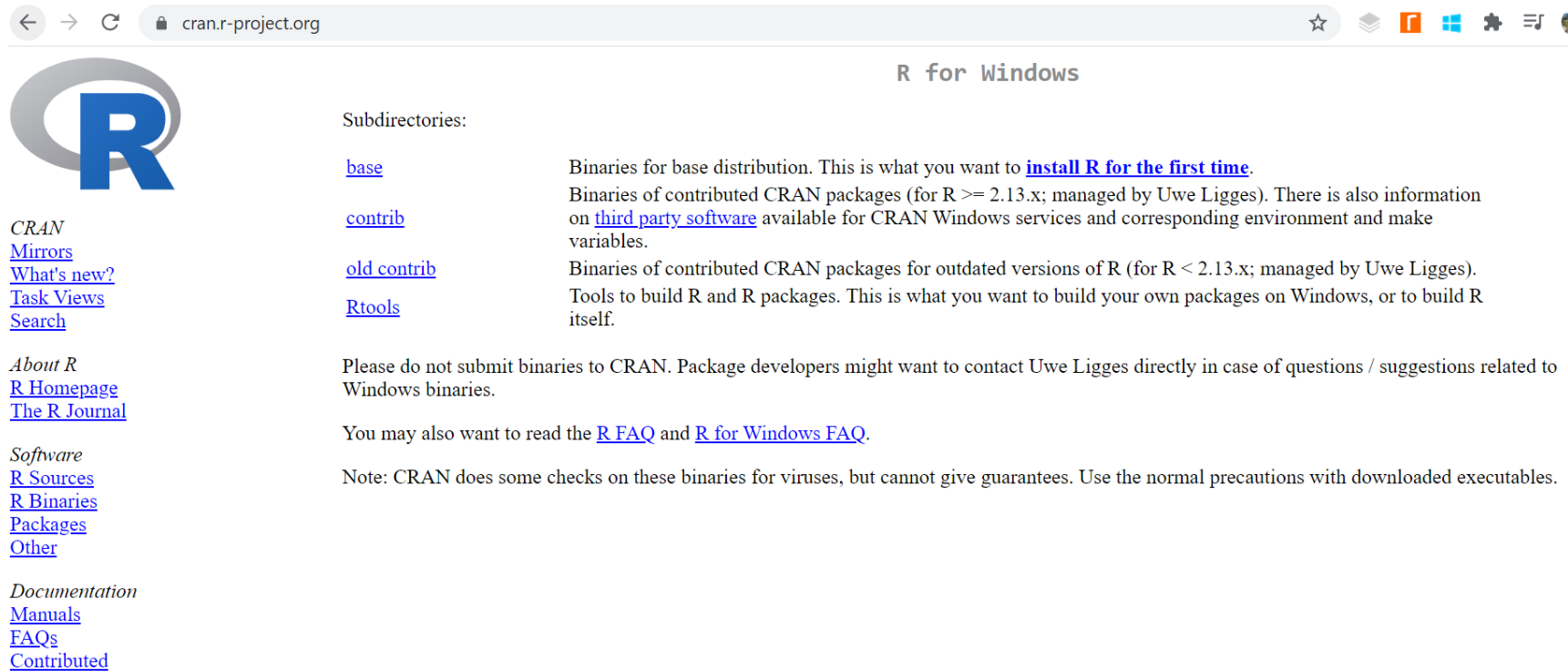
Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our [FAQs](#) before filing corresponding feature requests or bug reports.

Left sidebar links:

- CRAN
- [Mirrors](#)
- [What's new?](#)
- [Task Views](#)
- [Search](#)
- About R
- [R Homepage](#)
- [The R Journal](#)
- Software
- [R Sources](#)
- [R Binaries](#)
- [Packages](#)
- [Other](#)
- Documentation
- [Manuals](#)
- [FAQs](#)
- [Contributed](#)

Descargando e instalando el software



The screenshot shows the CRAN R for Windows website. The browser address bar displays 'cran.r-project.org'. The page features the R logo on the left, a navigation menu with links like 'CRAN', 'Mirrors', 'What's new?', 'Task Views', 'Search', 'About R', 'R Homepage', 'The R Journal', 'Software', 'R Sources', 'R Binaries', 'Packages', 'Other', 'Documentation', 'Manuals', 'FAQs', and 'Contributed'. The main content area is titled 'R for Windows' and lists subdirectories: 'base', 'contrib', 'old contrib', and 'Rtools', each with a brief description. A note at the bottom states: 'Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.'

cran.r-project.org

R for Windows

Subdirectories:

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

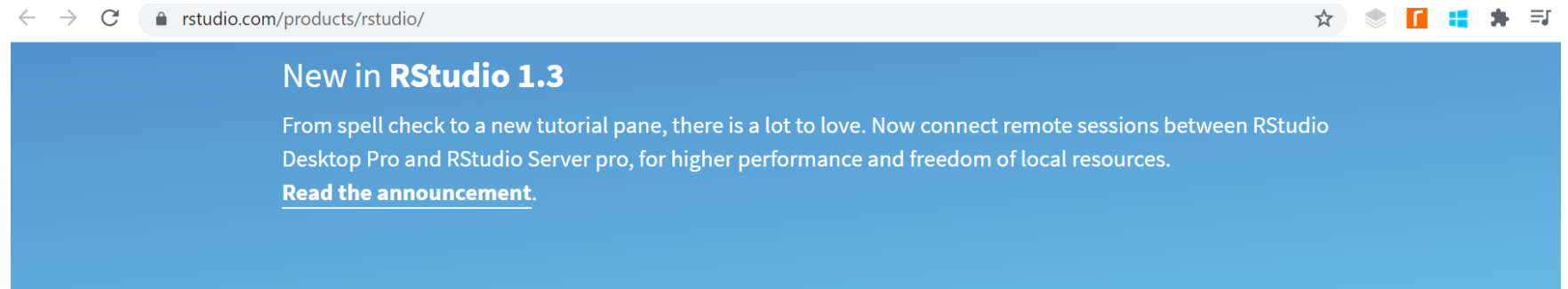
Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the [R FAQ](#) and [R for Windows FAQ](#).

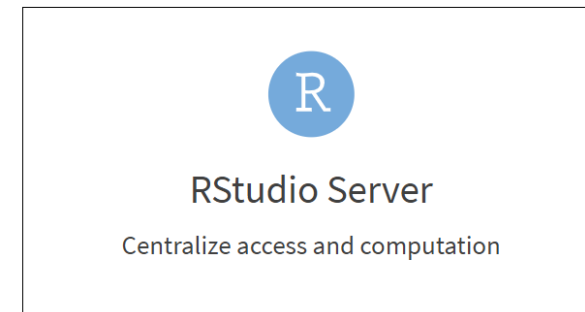
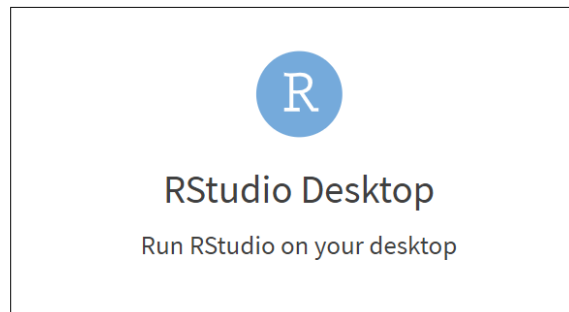
Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

Descargando e instalando el software

Una vez que tenemos el software R instalado, procedemos a instalar RStudio

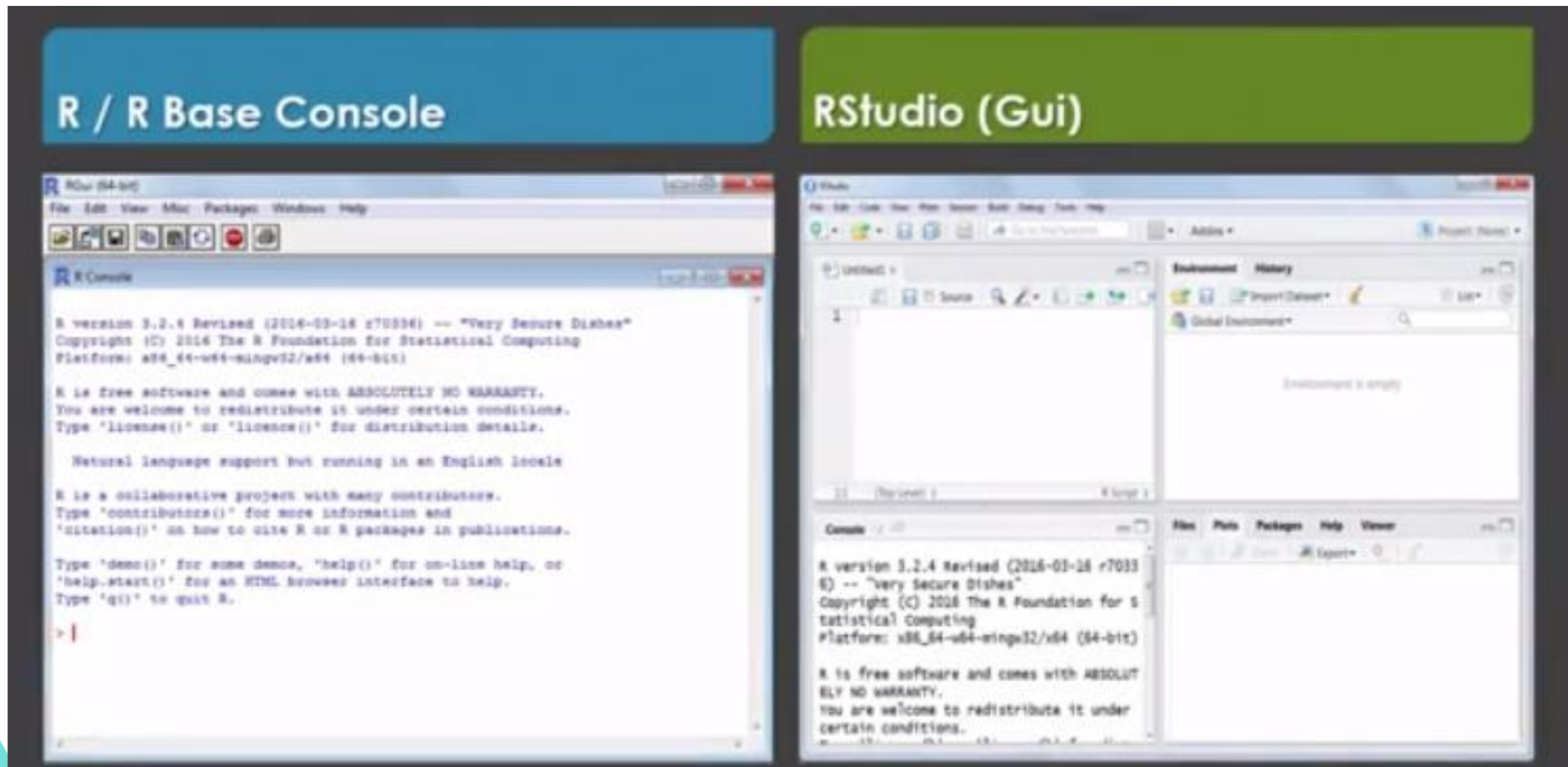


There are two versions of RStudio:



Descargando e instalando el software

Una vez que tenemos el software R instalado, procedemos a instalar RStudio



Agenda

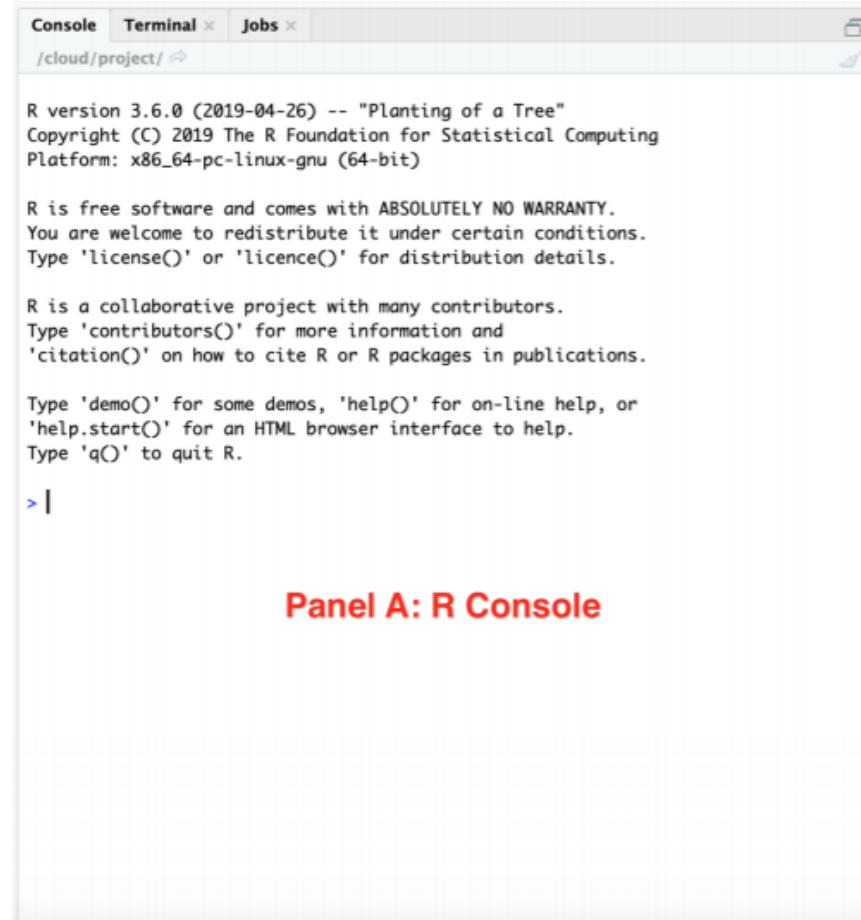
2. Qué nos ofrece este software?

2. Qué nos ofrece software?

RStudio ofrece:

1. Es un fuerte de editor de código que soporta la ejecución de código.
2. Manejo del workspace
3. Debugging, syntax-highlighting, permite completa el código de forma inteligente
4. Fácil comunicación con otros softwares y plataformas

RStudio application



The screenshot shows the RStudio Console panel with the following text:

```
/cloud/project/

R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

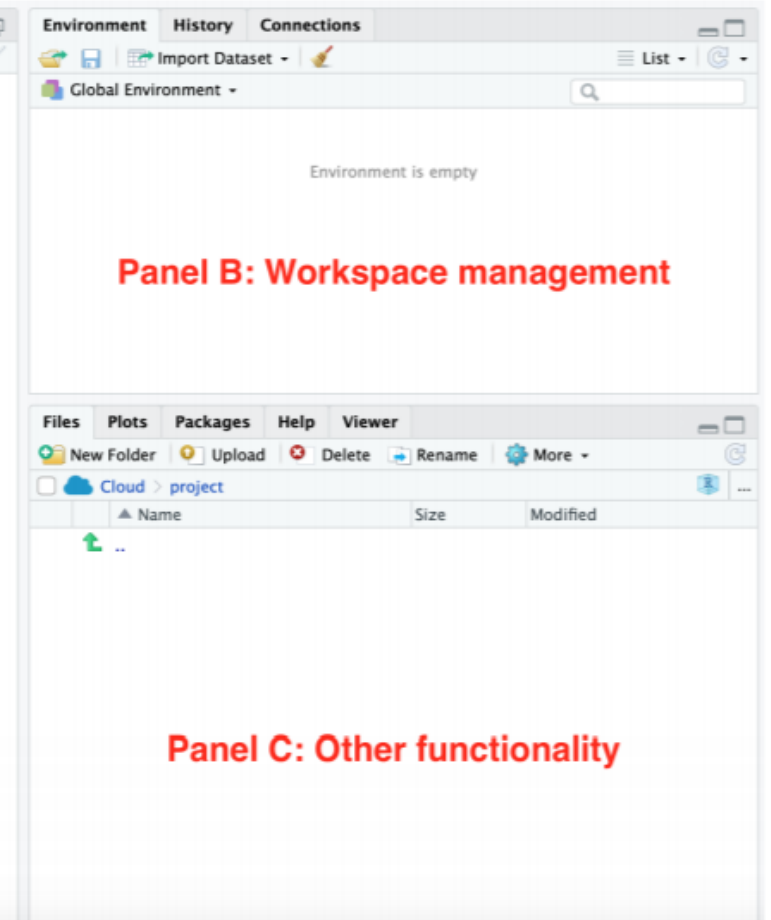
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

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.

> |
```

Panel A: R Console



2.1. Script (.R)

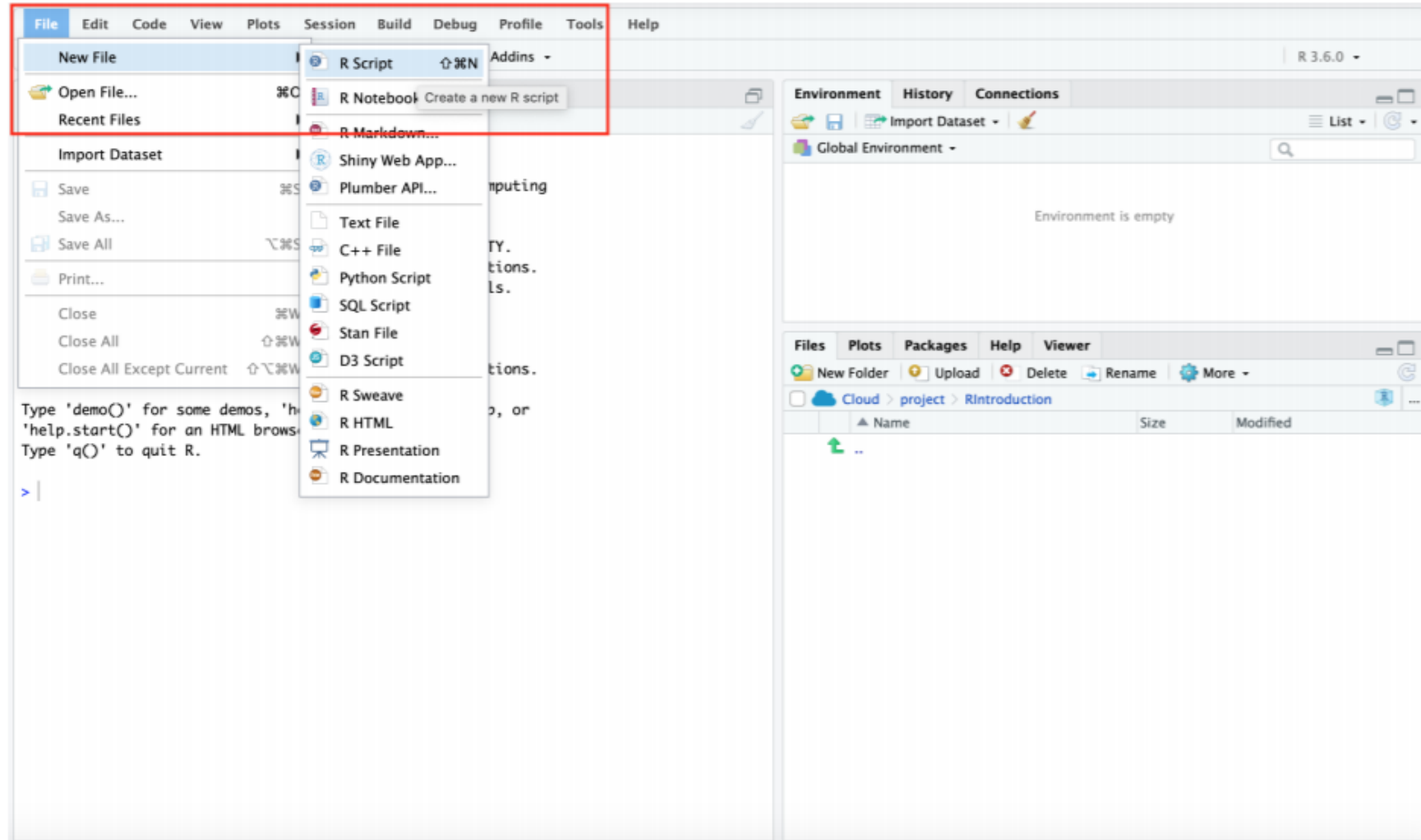
La secuencia de comandos necesaria para análisis es típicamente escrito en textfiles antes de su ejecución.

Ventajas:

- Documentación de tareas
- Automatización de tareas repetitivas
- Evaluación de cambios

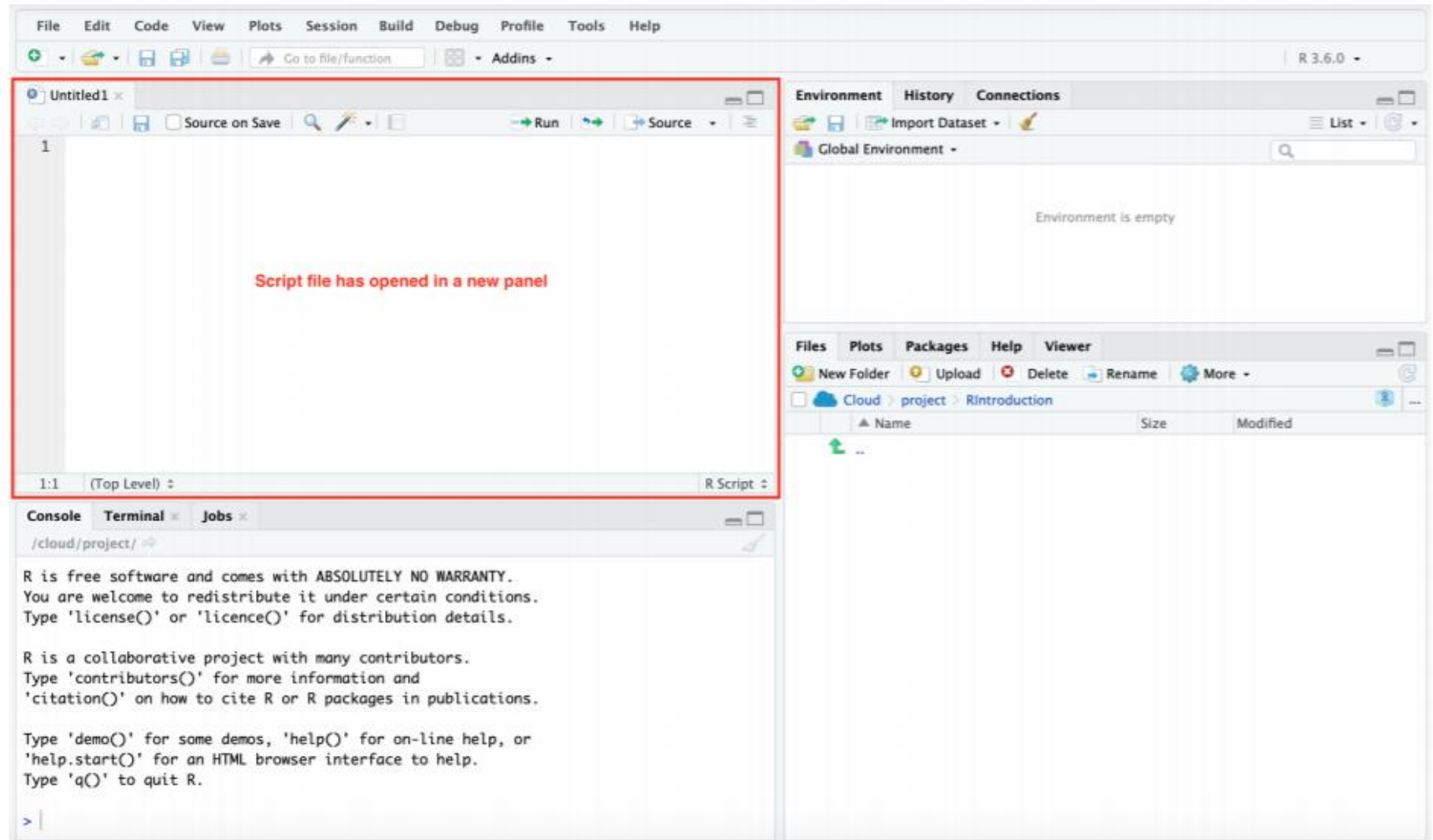
2.1. Script (.R)

Script (.R) Paso 1: Creamos un script file



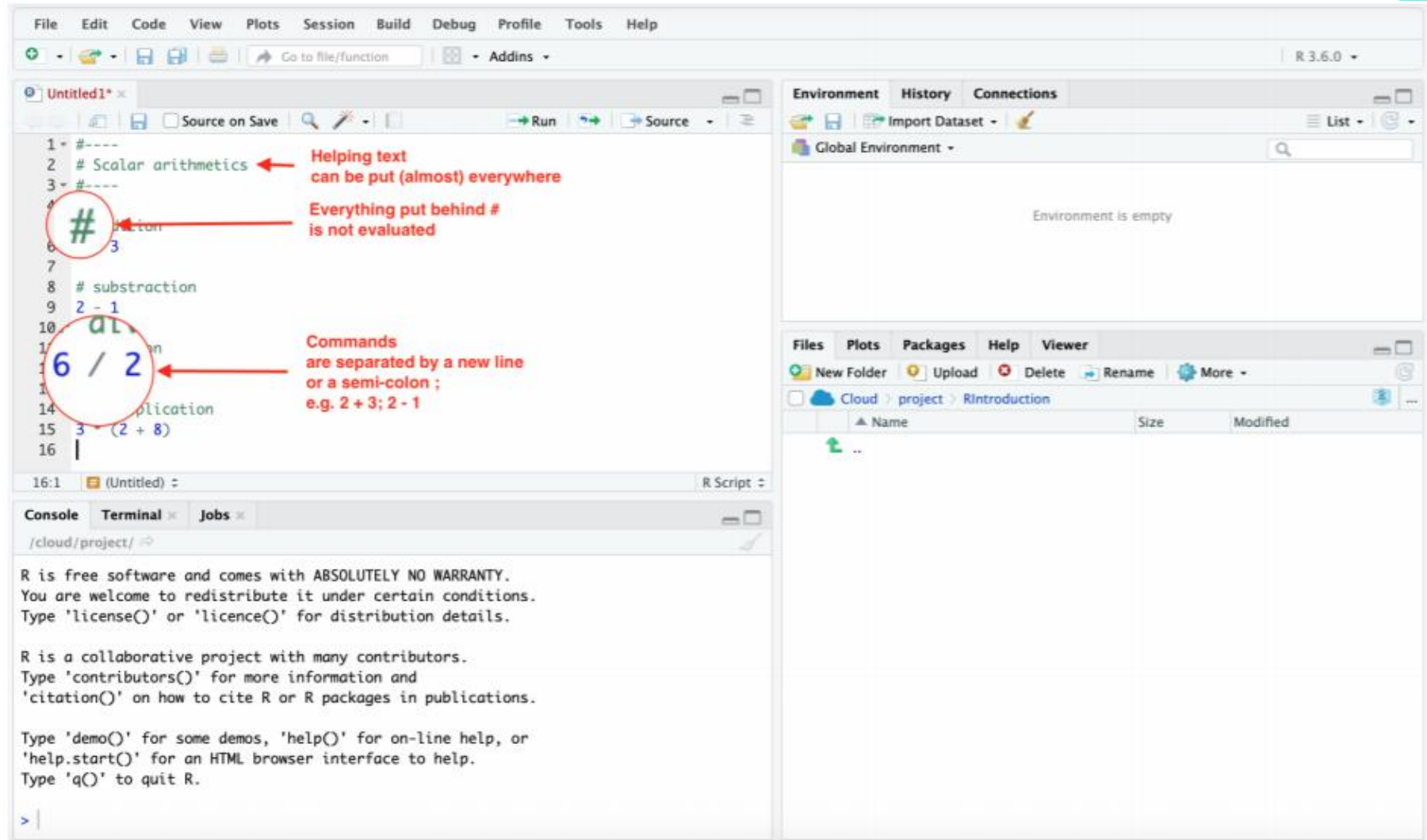
2.1. Script (.R)

Script (.R) Paso 2: Creamos un script file



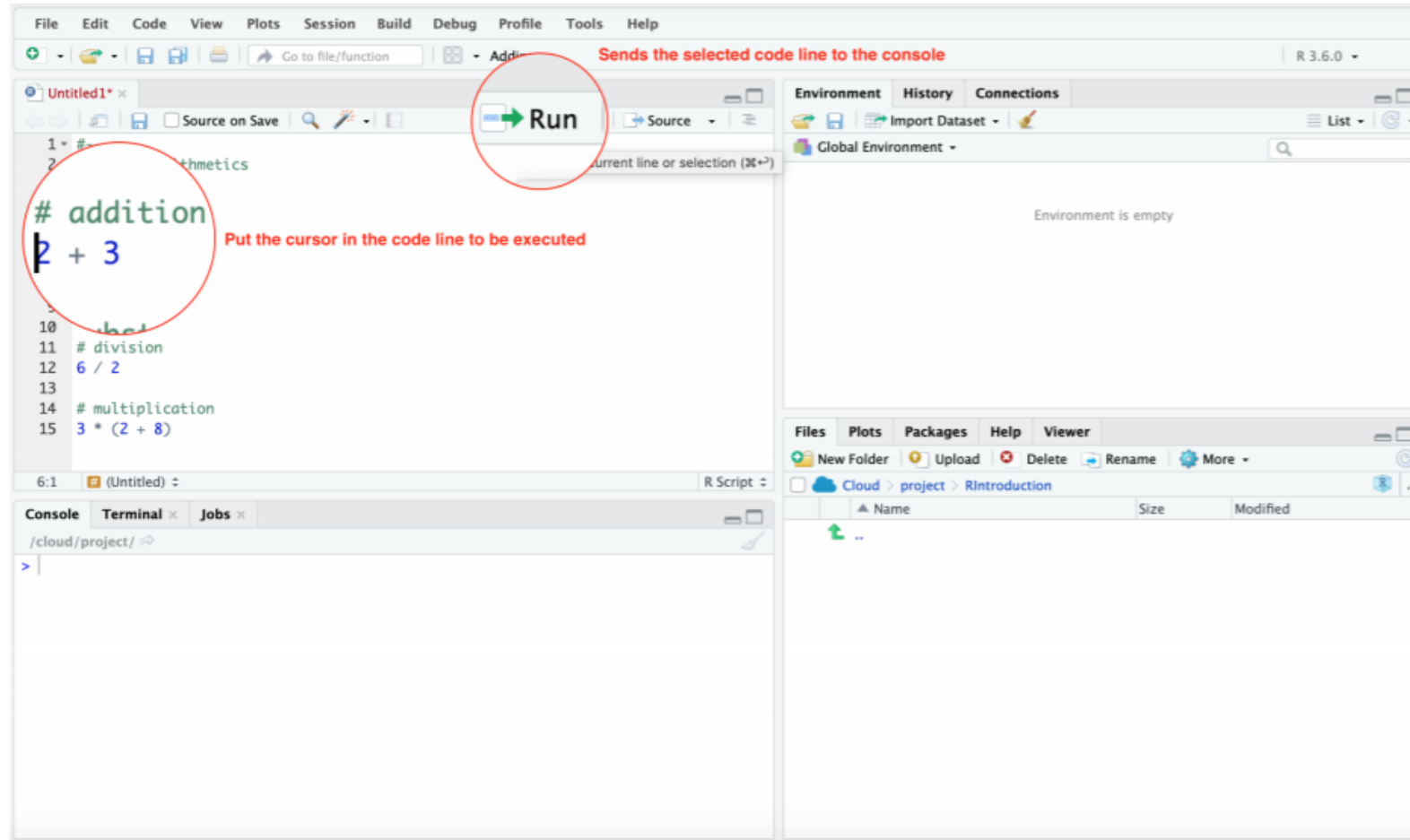
2.1. Script (.R)

Script (.R) Paso 3: Escribimos el script



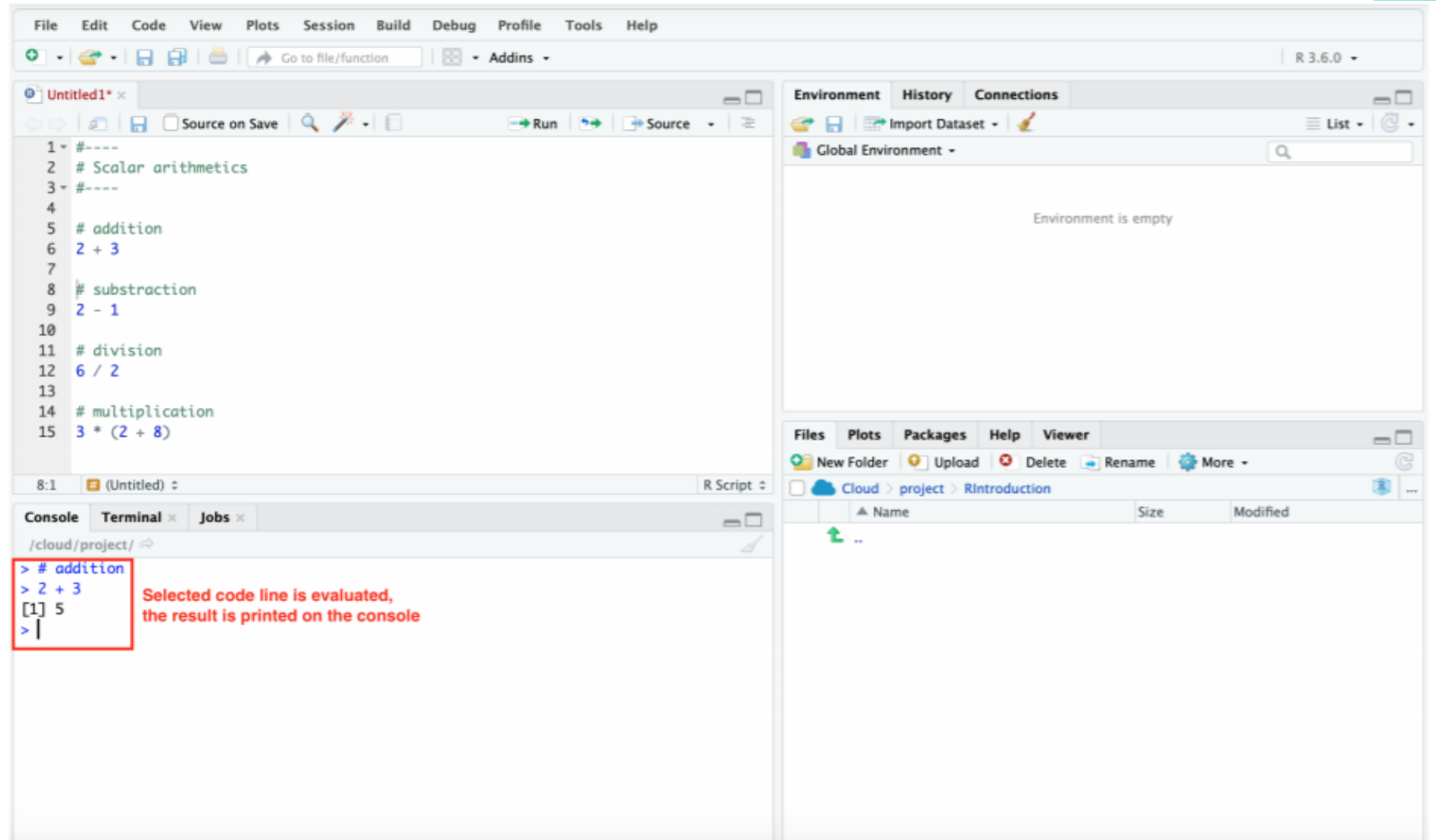
2.1. Script (.R)

Script (.R) Paso 3: Corremos el script



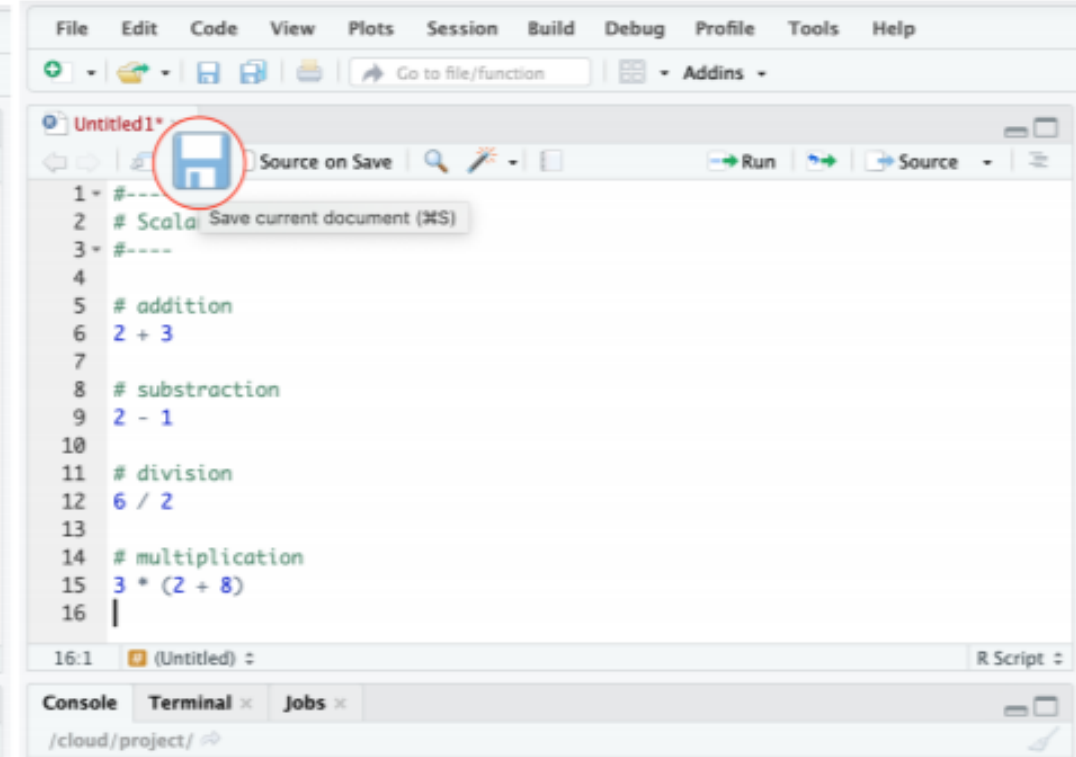
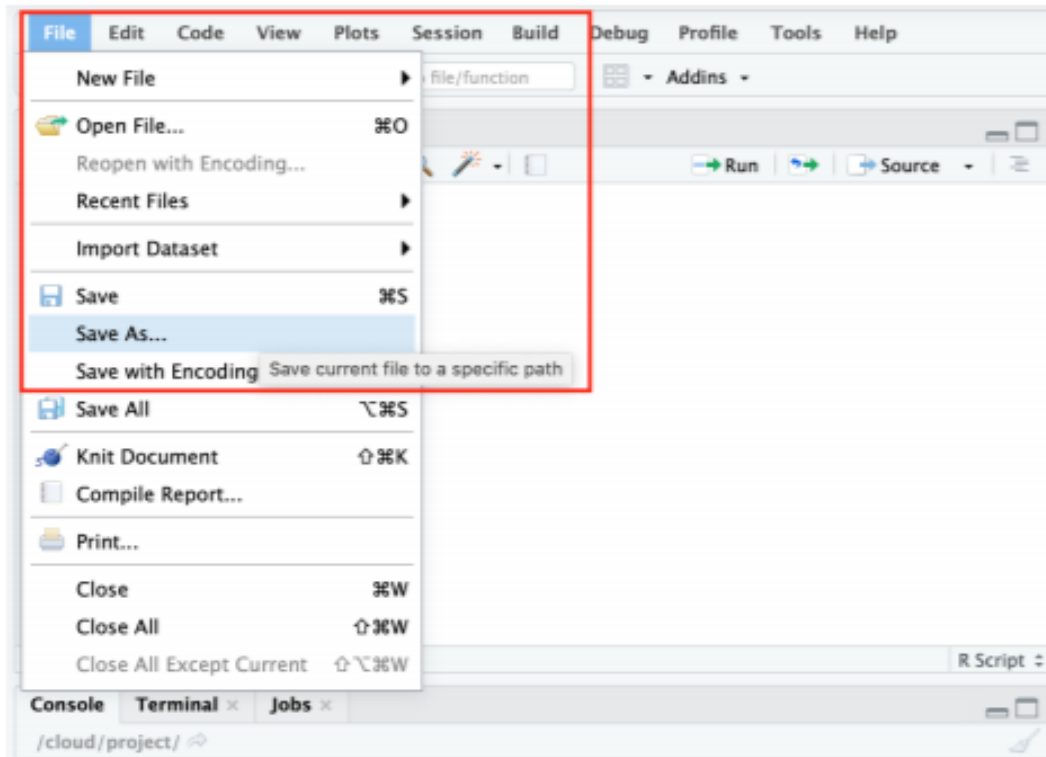
2. Qué nos ofrece software?

Script (.R) Paso 3: Corremos el script



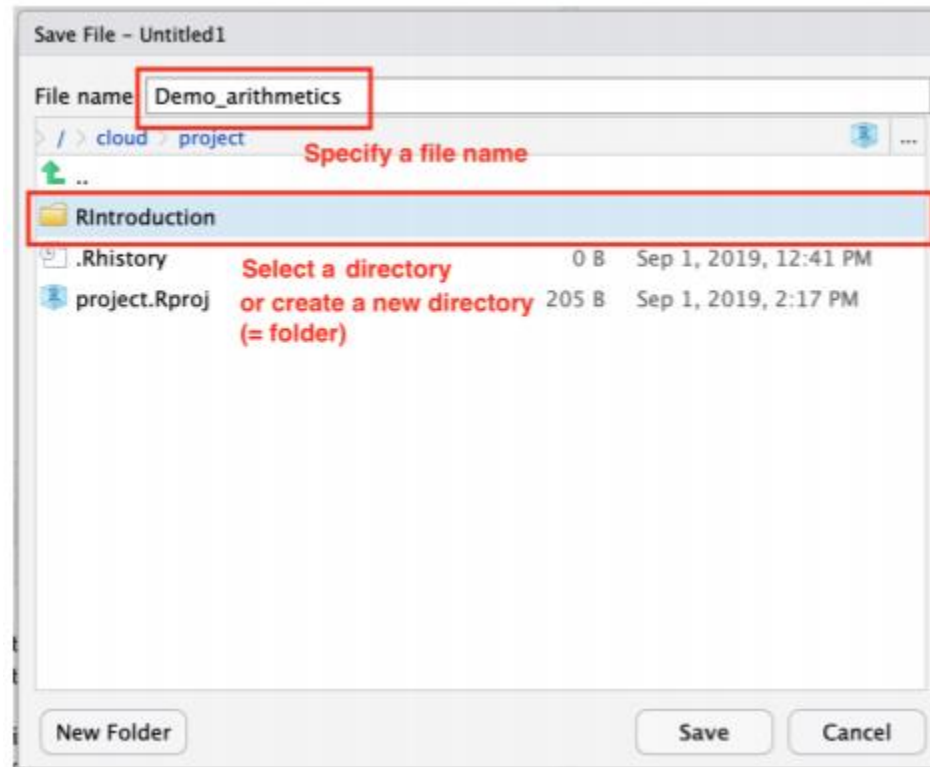
2.1. Script (.R)

Script (.R) Paso 4: Guardamos el script



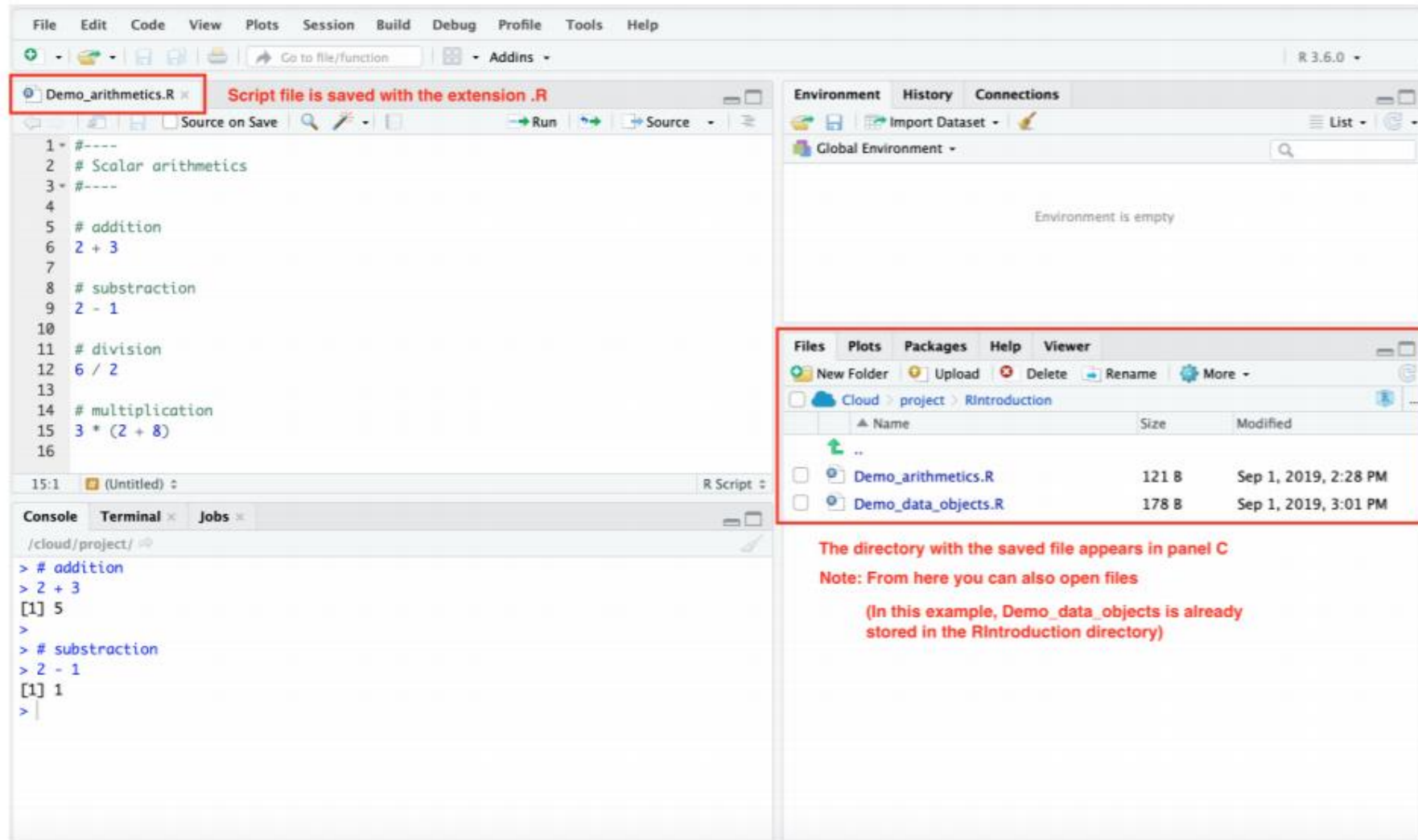
2.1. Script (.R)

Script (.R) Paso 5: Guardamos el script



2.1. Script (.R)

Script (.R) Paso 6: Guardamos el script



The screenshot displays the RStudio interface. The main editor window shows a script named `Demo_arithmetics.R` with the following content:

```
1 #----  
2 # Scalar arithmetics  
3 #----  
4  
5 # addition  
6 2 + 3  
7  
8 # subtraction  
9 2 - 1  
10  
11 # division  
12 6 / 2  
13  
14 # multiplication  
15 3 * (2 + 8)  
16
```

The console window shows the execution of the script:

```
> # addition  
> 2 + 3  
[1] 5  
>  
> # subtraction  
> 2 - 1  
[1] 1  
>
```

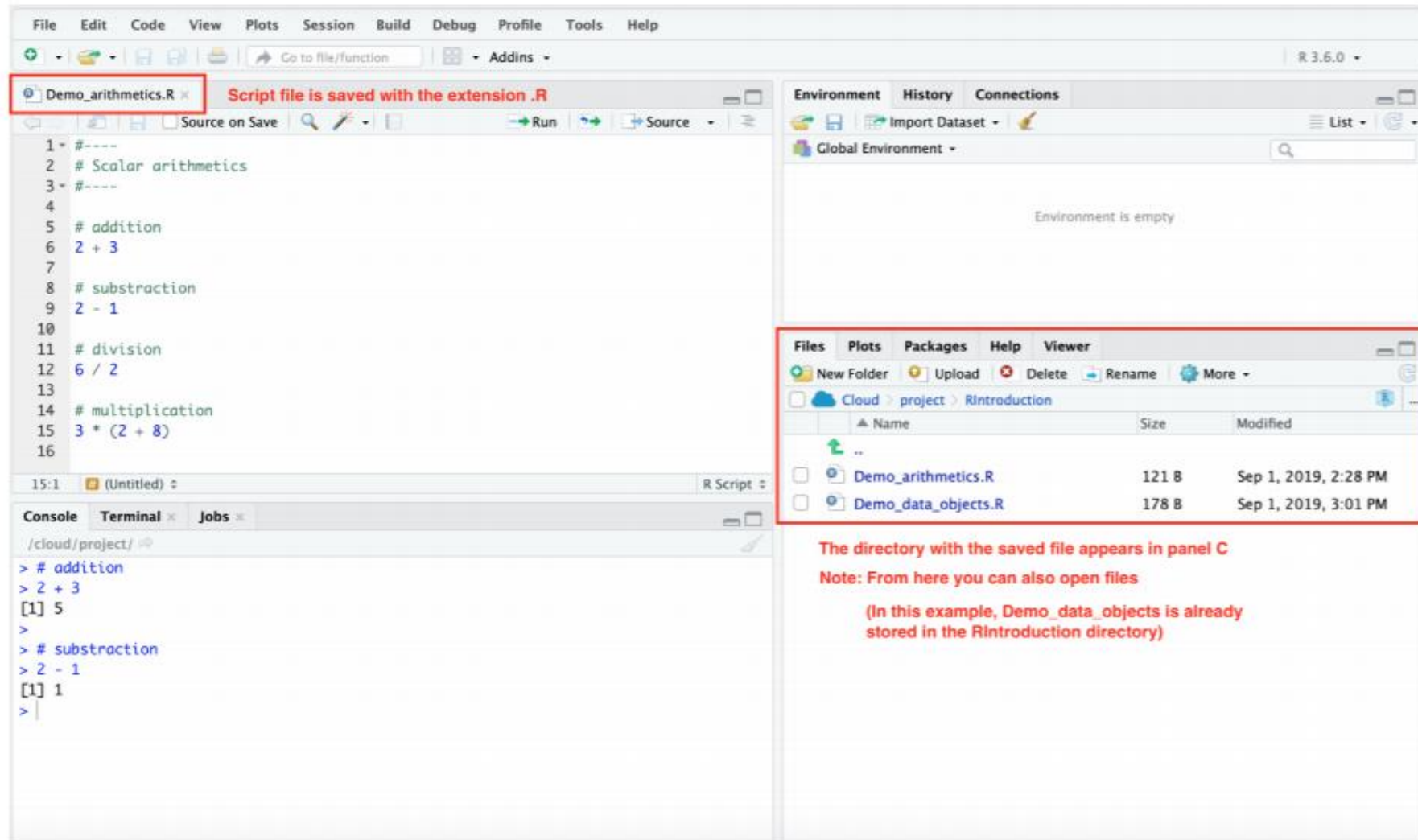
The file explorer panel (bottom right) shows the directory structure:

- Cloud > project > Rintroduction
- Files: `..`, `Demo_arithmetics.R` (121 B, Sep 1, 2019, 2:28 PM), `Demo_data_objects.R` (178 B, Sep 1, 2019, 3:01 PM)

The directory with the saved file appears in panel C
Note: From here you can also open files
(In this example, Demo_data_objects is already stored in the Rintroduction directory)

2.1. Script (.R)

Script (.R) Paso 6: Guardamos el script



The screenshot displays the RStudio interface. The main editor window shows a script named `Demo_arithmetics.R` with the following content:

```
1 #----  
2 # Scalar arithmetics  
3 #----  
4  
5 # addition  
6 2 + 3  
7  
8 # subtraction  
9 2 - 1  
10  
11 # division  
12 6 / 2  
13  
14 # multiplication  
15 3 * (2 + 8)  
16
```

The console window shows the execution of the script:

```
> # addition  
> 2 + 3  
[1] 5  
>  
> # subtraction  
> 2 - 1  
[1] 1  
>
```

The file explorer panel (bottom right) shows the directory structure:

- Cloud > project > Rintroduction
- Files: `..`, `Demo_arithmetics.R` (121 B, Sep 1, 2019, 2:28 PM), `Demo_data_objects.R` (178 B, Sep 1, 2019, 3:01 PM)

The directory with the saved file appears in panel C

Note: From here you can also open files

(In this example, `Demo_data_objects` is already stored in the `Rintroduction` directory)

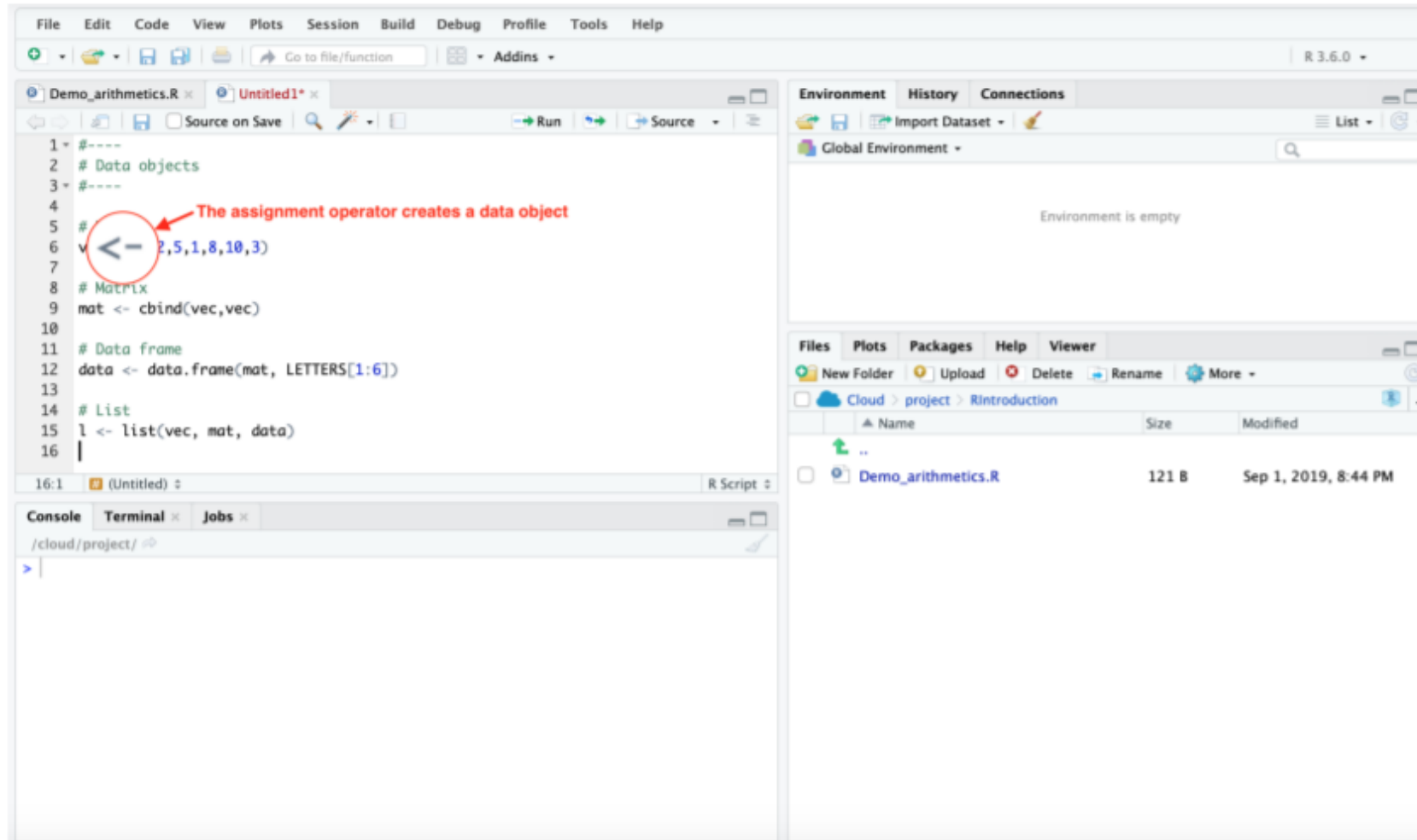
2.2. Shortcuts

	Windows (Linux)	Mac
New R Script file	Shift + Ctrl +N	Shift + CMD +N
Open file	Ctrl + O	Ctrl + O
Run current line/selection and jump to next line	Ctrl + Enter	CMD + Enter
Run current line/selection DONT jump to next line	Alt + Enter	Alt + Enter
Run whole script	Ctrl + Alt + R	CMD + Alt + R
Run code from beginning to the current line	Ctrl + Alt + B	CMD + Alt + B
Run code from current line to end	Ctrl + Alt + E	CMD + Alt + E
Save current file	Ctrl + S	CMD + S
Close file	Ctrl + W	Ctrl + W
Move cursor to source editor	Ctrl + 1	Ctrl +1
Move cursor to console	Ctrl + 2	Ctrl + 2
Delete current selection	Ctrl + D	CMD + D
Clear console	Ctrl + L	Ctrl + L
Navigate console history	Up/Down	Up/Down
Move code line up and down (avoids copy-paste work)	Alt + Up/Down	Alt + Up/Down
Interrupt currently executing command	Esc	Esc

2.3. Workspace (.Rdata)

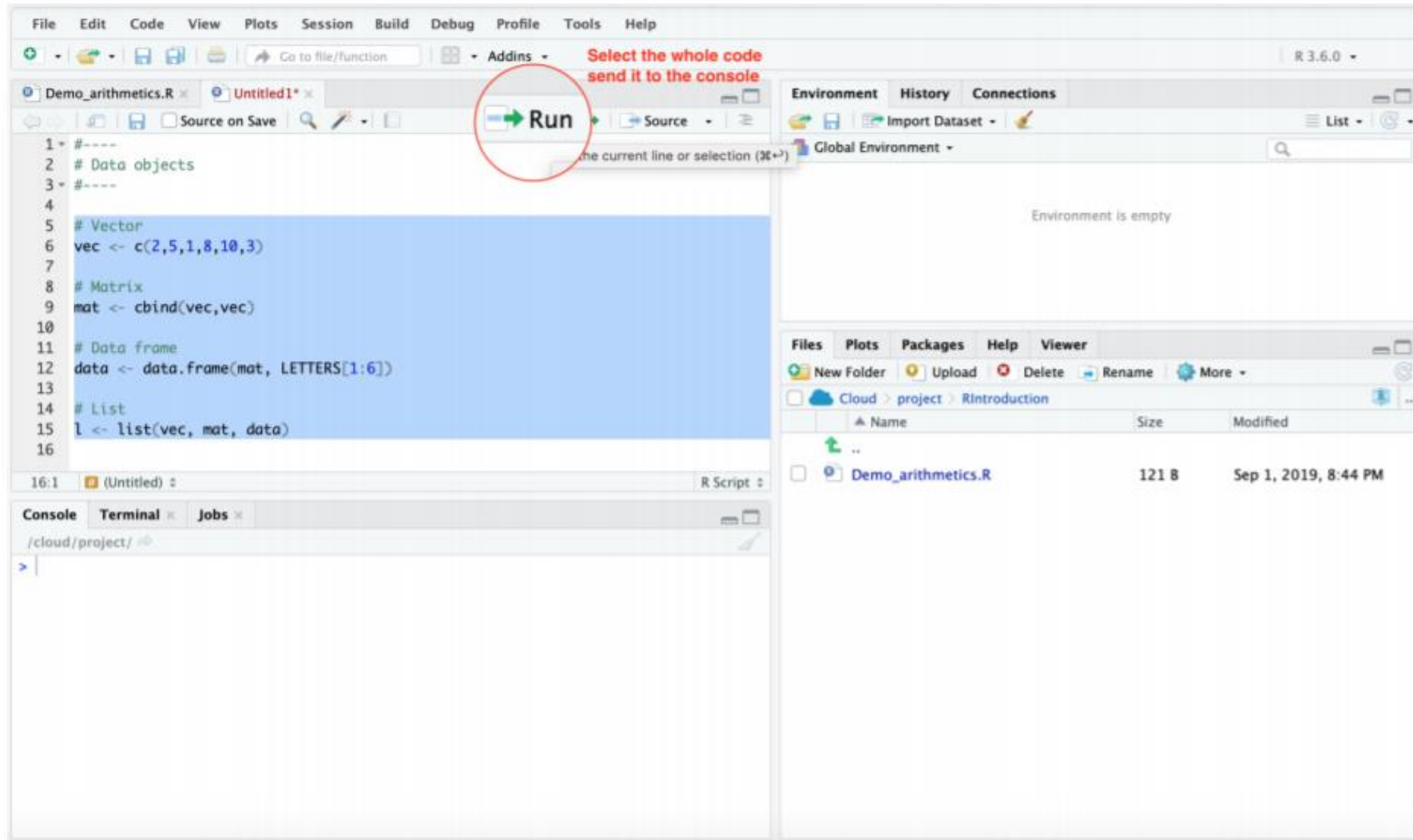
El workspace de R consiste en todos los objetos creados o cargados durante la sesión de R. Para crear objetos vamos a pegar y copiar el contenido de Demo_data_objects.txt en un nuevo script.

Creating data objects - Step 1



2.3. Workspace (.Rdata)

Creating data objects - Step 2



The screenshot displays the RStudio interface with the following components:

- Source Editor:** Contains R code for creating data objects. Lines 5 through 15 are highlighted in blue. A red circle highlights the 'Run' button (a green arrow) in the toolbar, with a red text annotation: "Select the whole code send it to the console".
- Code:**

```
1 #----  
2 # Data objects  
3 #----  
4  
5 # Vector  
6 vec <- c(2,5,1,8,10,3)  
7  
8 # Matrix  
9 mat <- cbind(vec,vec)  
10  
11 # Data frame  
12 data <- data.frame(mat, LETTERS[1:6])  
13  
14 # List  
15 l <- list(vec, mat, data)  
16
```
- Environment:** Shows the 'Global Environment' with the message 'Environment is empty'.
- Files:** A file explorer showing the project structure. It includes a file named 'Demo_arithmetics.R' with a size of 121 B, last modified on Sep 1, 2019, at 8:44 PM.
- Console:** Shows the prompt '>' at the bottom, ready for input.

2.3. Workspace (.Rdata)

Creating data objects - Step 3

The screenshot displays the RStudio interface with the following components:

- Source Editor:** Contains R code for creating data objects. Lines 5-15 are highlighted in blue.
- Environment Pane:** Shows the global environment with the following objects:

Object	Details
data	6 obs. of 3 variables
l	List of 3
mat	num [1:6, 1:2] 2 5 1 8 10 3 2 5 1 8 ...
vec	num [1:6] 2 5 1 8 10 3
- Console:** Shows the execution of the code. A red note states: "Selected code is evaluated NO result is printed on the console!".
- Files Pane:** Shows the project structure with the file "Demo_arithmetics.R" (121 B, Sep 1, 2019, 8:44 PM).

The code in the source editor is as follows:

```
1 #----
2 # Data objects
3 #----
4
5 # Vector
6 vec <- c(2,5,1,8,10,3)
7
8 # Matrix
9 mat <- cbind(vec,vec)
10
11 # Data frame
12 data <- data.frame(mat, LETTERS[1:6])
13
14 # List
15 l <- list(vec, mat, data)
16
```


2.3. Workspace (.Rdata)

Inspecting data objects

The screenshot displays the RStudio environment with several key components:

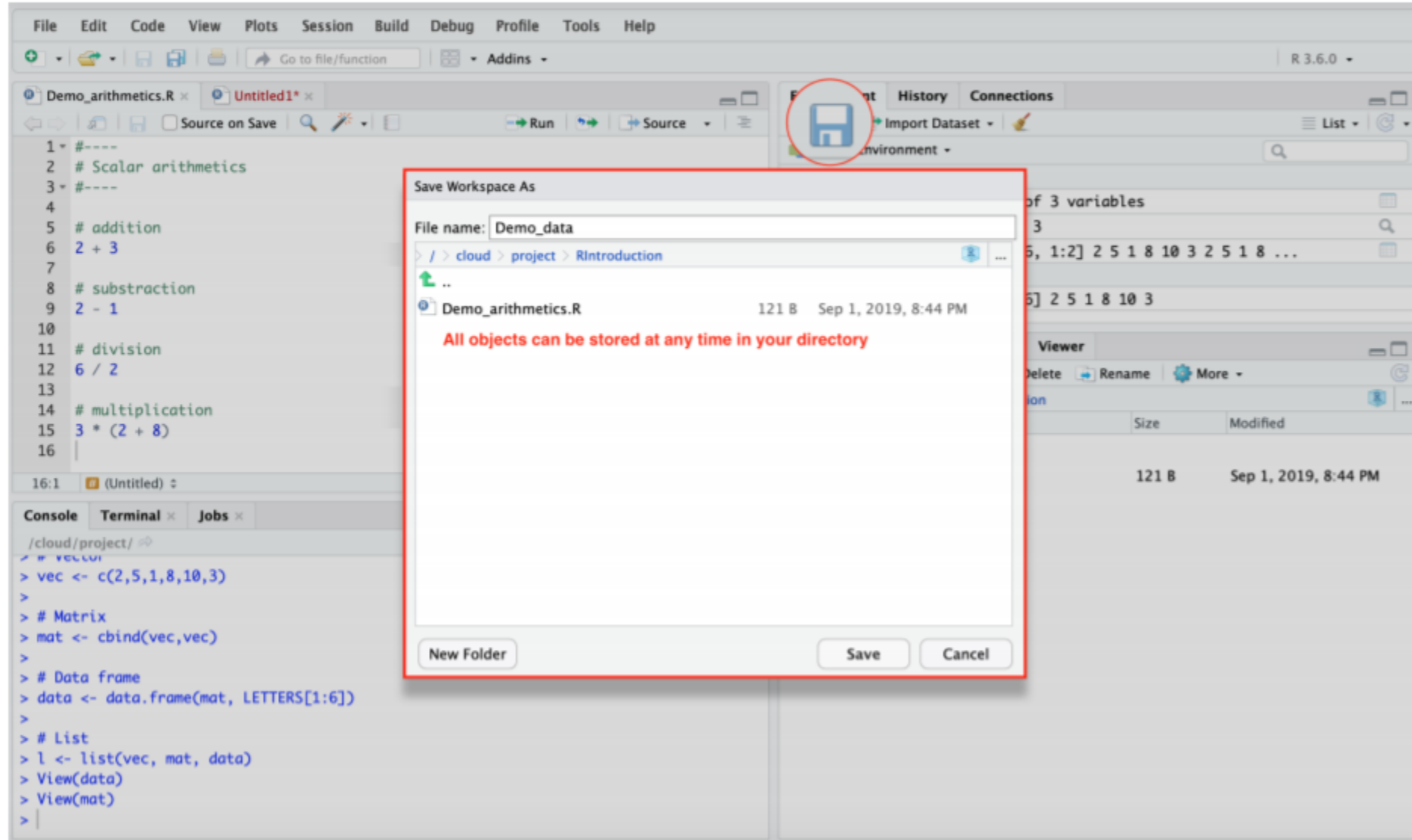
- Editor:** Shows a script file named `Demo_arithmetics.R` with the following R code:

```
> # Vector
> vec <- c(2,5,1,8,10,3)
>
> # Matrix
> mat <- cbind(vec,vec)
>
> # Data frame
> data <- data.frame(mat, LETTERS[1:6])
>
> # List
> l <- list(vec, mat, data)
> View(data)
> View(mat)
>
```
- Environment:** The `Environment pane on the right shows the Global Environment with a Data section. The data object is highlighted, showing it has 6 observations and 3 variables. Below it, the Values section shows the structure of the vec and mat objects.`
- Files:** The `Files` pane at the bottom right shows the project structure, including a file named `Demo_arithmetics.R` with a size of 121 B, last modified on Sep 1, 2019, at 8:44 PM.
- Console:** The `Console` pane at the bottom left shows the execution of the R code, with the current directory set to `/cloud/project/`.

A red box highlights the `data` object in the Environment pane, and a red arrow points to it from the text: "Data objects can be inspected by clicking on them, than a new file opens. Note: this is just a preview, if you close the file nothing is lost!"

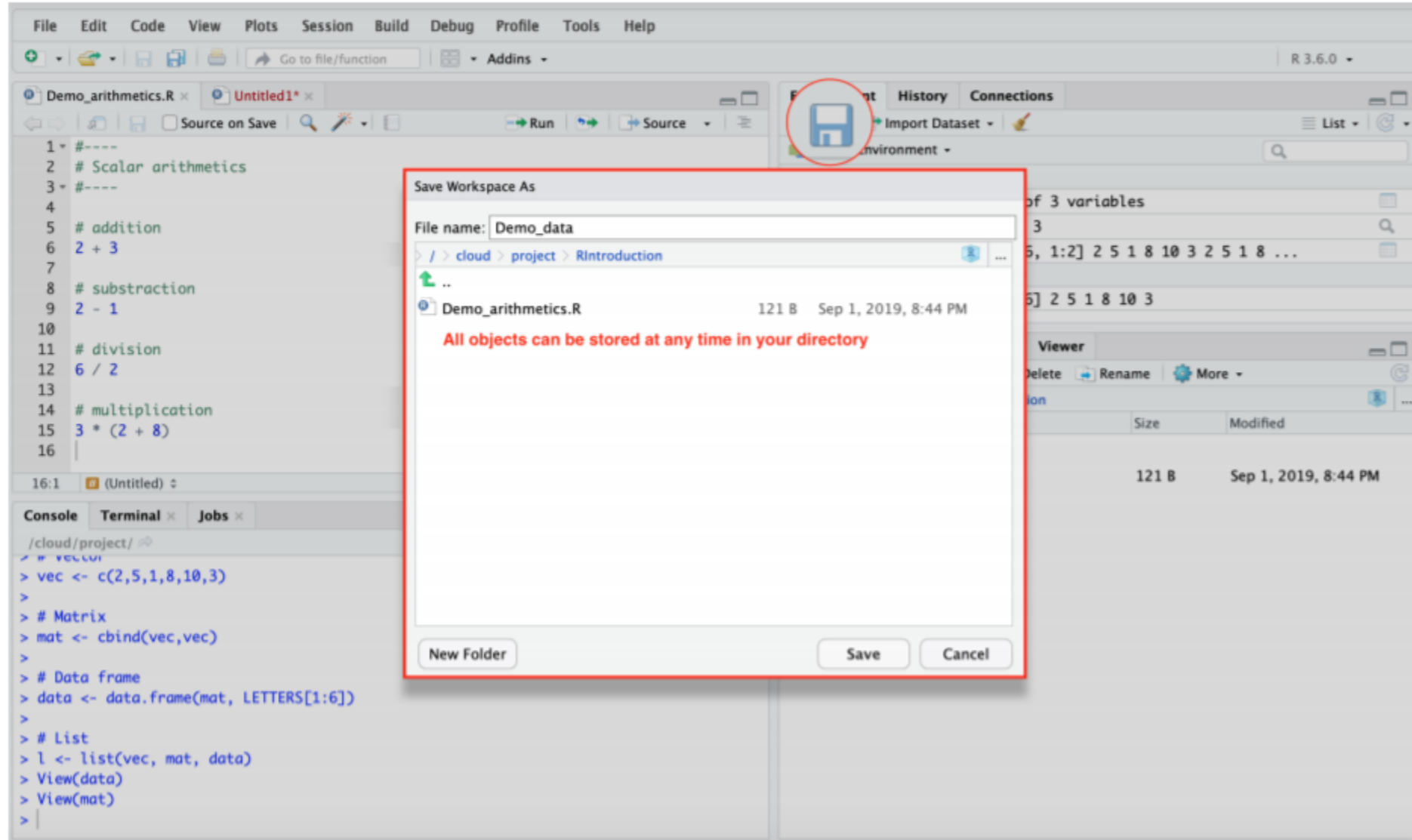
2.3. Workspace (.Rdata)

Saving the workspace



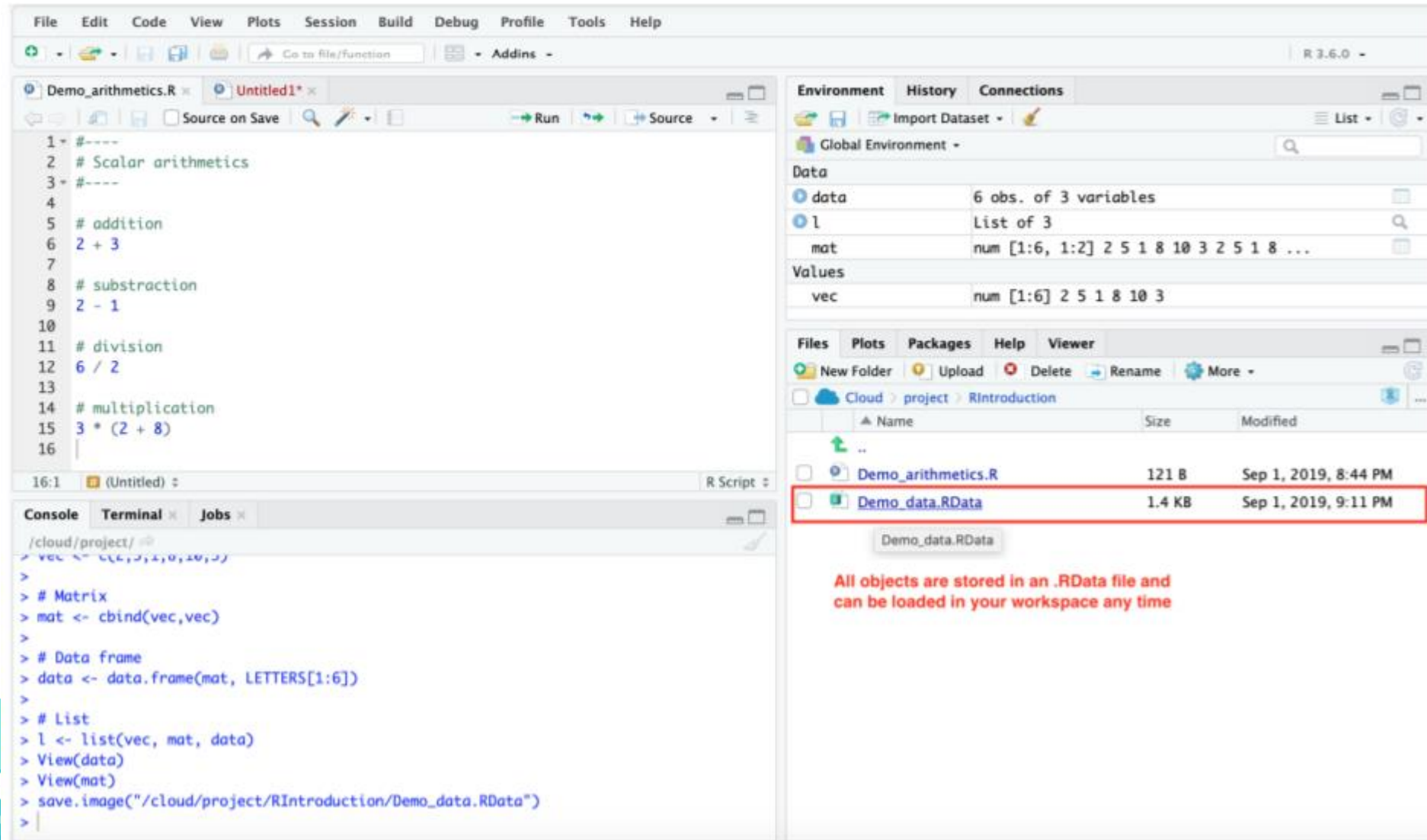
2.3. Workspace (.Rdata)

Saving the workspace



2.3. Workspace (.Rdata)

Loading the workspace



The screenshot displays the RStudio interface with three main panes:

- Source Pane:** Contains an R script file named `Demo_arithmetics.R`. The script includes comments for scalar arithmetic, addition, subtraction, division, and multiplication, followed by corresponding R code lines.
- Console Pane:** Shows the execution of R commands. The first command is `vec <- c(2, 5, 1, 8, 10, 3)`. Subsequent commands create a matrix `mat` from `vec`, a data frame `data` from `mat` and `LETTERS[1:6]`, a list `l` containing `vec`, `mat`, and `data`, and finally save the workspace to `Demo_data.RData` using `save.image("/cloud/project/RIntroduction/Demo_data.RData")`.
- Environment Pane:** Displays the current workspace. It shows the `Global Environment` with a search bar. Below, it lists objects: `data` (6 obs. of 3 variables), `l` (List of 3), `mat` (num [1:6, 1:2] 2 5 1 8 10 3 2 5 1 8 ...), and `vec` (num [1:6] 2 5 1 8 10 3). The `Files tab shows a file explorer view of the project > RIntroduction directory, listing Demo_arithmetics.R (121 B) and Demo_data.RData (1.4 KB, modified Sep 1, 2019, 9:11 PM), which is highlighted with a red box.`

All objects are stored in an .RData file and can be loaded in your workspace any time

2.4. History (.Rhistory)

El History file es un text file que lista todos los comandos que se han ido ejecutando durante la sesión.

Inspecting the worksession history

The screenshot displays the RStudio IDE interface. The main editor window on the left shows a script file named 'Demo_arithmetics.R' with the following R code:

```
1 #----  
2 # Data objects  
3 #----  
4  
5 # Vector  
6 vec <- c(2,5,1,8,10,3)  
7  
8 # Matrix  
9 mat <- cbind(vec,vec)  
10  
11 # Data frame  
12 data <- data.frame(mat, LETTERS[1:6])  
13  
14 # List  
15 l <- list(vec, mat, data)  
16
```

A red text annotation points to the History pane: "Lists all commands executed during your worksession".

The History pane on the right, titled 'History', shows a list of commands executed during the session:

```
# Vector  
vec <- c(2,5,1,8,10,3)  
# Matrix  
mat <- cbind(vec,vec)  
# Data frame  
data <- data.frame(mat, LETTERS[1:6])  
# List  
l <- list(vec, mat, data)
```

The bottom pane shows the Console with the same commands being executed:

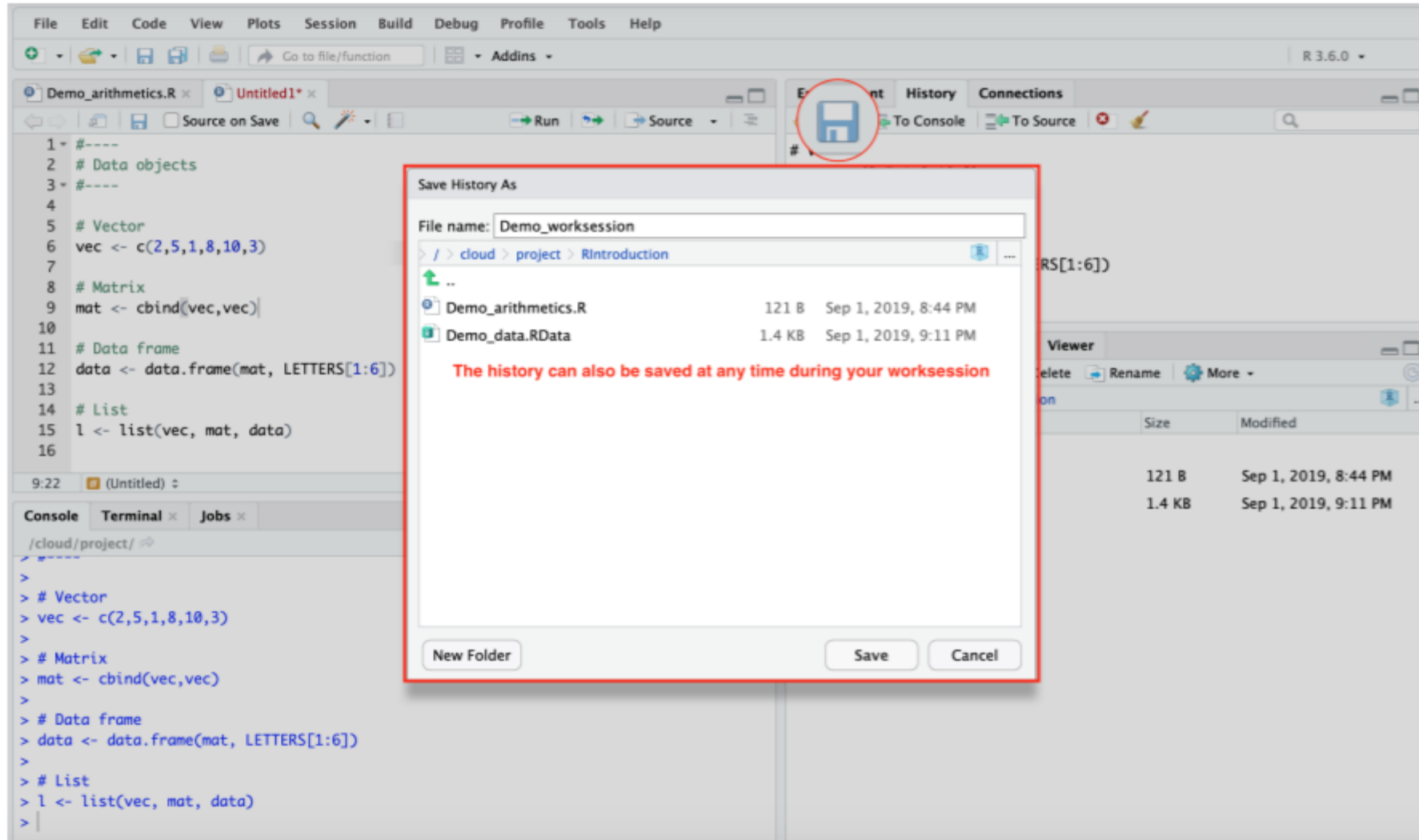
```
> # Vector  
> vec <- c(2,5,1,8,10,3)  
>  
> # Matrix  
> mat <- cbind(vec,vec)  
>  
> # Data frame  
> data <- data.frame(mat, LETTERS[1:6])  
>  
> # List  
> l <- list(vec, mat, data)  
>
```

The bottom right pane shows the Files view with a table of files:

Name	Size	Modified
..		
Demo_arithmetics.R	121 B	Sep 1, 2019, 8:44 PM
Demo_data.RData	1.4 KB	Sep 1, 2019, 9:11 PM

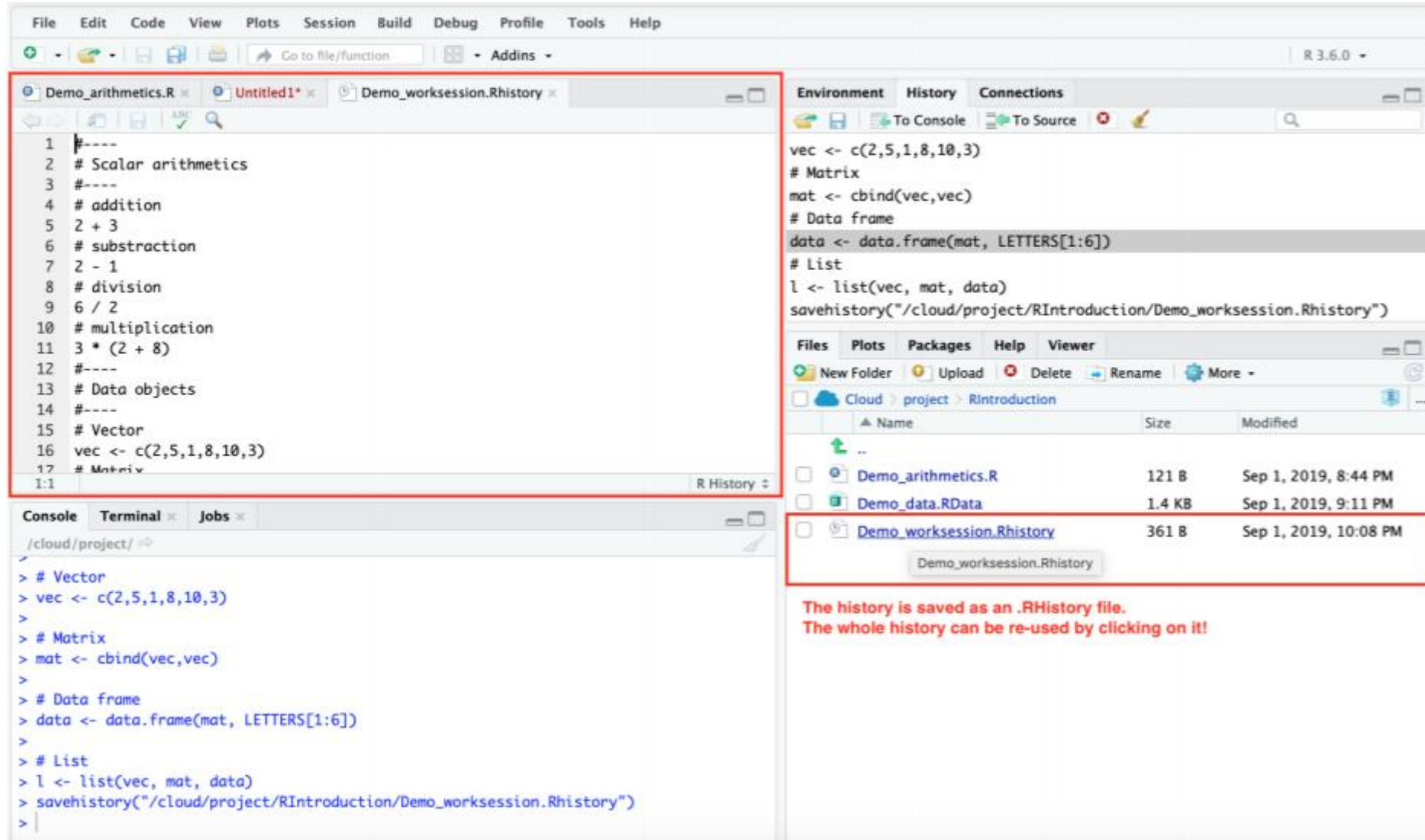
2.4. History (.Rhistory)

Saving the worksession history



2.4. History (.Rhistory)

Recycling the worksession history



The screenshot displays the RStudio interface with three main panels. The top-left panel shows a script file named 'Demo_worksession.Rhistory' with R code comments and commands. The bottom-left panel shows the console output of these commands. The right panel shows the 'History' pane, which lists the executed commands and their corresponding objects. The 'Demo_worksession.Rhistory' file is highlighted in the History pane.

Script File: Demo_worksession.Rhistory

```
1 #----  
2 # Scalar arithmetics  
3 #----  
4 # addition  
5 2 + 3  
6 # subtraction  
7 2 - 1  
8 # division  
9 6 / 2  
10 # multiplication  
11 3 * (2 + 8)  
12 #----  
13 # Data objects  
14 #----  
15 # Vector  
16 vec <- c(2,5,1,8,10,3)  
17 # Matrix  
1:1
```

Console Output:

```
> # Vector  
> vec <- c(2,5,1,8,10,3)  
>  
> # Matrix  
> mat <- cbind(vec,vec)  
>  
> # Data frame  
> data <- data.frame(mat, LETTERS[1:6])  
>  
> # List  
> l <- list(vec, mat, data)  
> savehistory("/cloud/project/RIntroduction/Demo_worksession.Rhistory")  
>
```

History Pane:

Name	Size	Modified
..		
Demo_arithmetics.R	121 B	Sep 1, 2019, 8:44 PM
Demo_data.RData	1.4 KB	Sep 1, 2019, 9:11 PM
Demo_worksession.Rhistory	361 B	Sep 1, 2019, 10:08 PM

The history is saved as an .Rhistory file.
The whole history can be re-used by clicking on it!

2.4. History (.Rhistory)

Recycling the worksession history

The screenshot displays the RStudio environment with several key components:

- Source Editor:** Contains R code for creating data objects. Line 17, `data <- data.frame(mat, LETTERS[1:6])`, is highlighted with a red box.
- Environment/History Panel:** Located on the right, it shows the execution history. The same line of code from the source editor is highlighted in blue. A red circle around the 'To Source' button, with an arrow pointing to it and the text 'Inserts a selected line of code into your script file', indicates how to reuse the code.
- Files Panel:** Shows a file explorer view with a table of files in the project directory.
- Console:** Displays the output of the R commands executed in the source editor.

Name	Size	Modified
..		
Demo_arithmetics.R	121 B	Sep 1, 2019, 8:44 PM
Demo_data.RData	1.4 KB	Sep 1, 2019, 9:11 PM
Demo_worksession.Rhistory	361 B	Sep 1, 2019, 10:08 PM

```
> # Vector
> vec <- c(2,5,1,8,10,3)
>
> # Matrix
> mat <- cbind(vec,vec)
>
> # Data frame
> data <- data.frame(mat, LETTERS[1:6])
>
> # List
> l <- list(vec, mat, data)
> savehistory("/cloud/project/RIntroduction/Demo_worksession.Rhistory")
>
```


Agenda

3. Lo que debemos saber

3. Lo que debemos saber

Tipos de datos

En R tenemos 3 básicos tipos de datos:

1. a. Numérico (números reales)

Los dos tipos de data más comunes son 'double' e 'integer'

- b. Complejo (números imaginarios y reales)

2. Lógico (boolean)

Lógicas como TRUE y FALSE

3. Carácter

Es un tipo de data para almacenar letras y símbolos (strings, text)

3. Lo que debemos saber

Estructura de datos

1. Scalar
2. Vector: Collection of elements of a single (“atomic”) data type
3. Matrix: Collection of elements arranged in a two-dimensional rectangular layout (a twodimensional generalization of a vector). Same as vector, all elements must be of a single data type.
4. Data frame: More general matrix like structure (“data matrix”). Different columns can have different data types.
5. List: Generic vector containing other objects. No restriction on data types or length of the single components

3. Lo que debemos saber

Estructura de datos

Vectors

Concatinating elements together with `c()`.

```
> c(0.5, 0.6, 0.25)           # double
> c(9L, 10L, 11L, 12L, 13L)   # integer
> c(9:13)                      # integer sequence
> c(TRUE, FALSE, FALSE)       # logical
> c(1+0i, 2+4i)                # complex
> c("a", "b", "c")            # character
```

3. Lo que debemos saber

Assign the vectors to names:

```
> dbl <- c(0.5, 0.6, 0.25)
> chr <- c("a", "b", "c")
```

Print out the dbl and chr vectors on the console:

```
> dbl
```

```
[1] 0.50 0.60 0.25
```

```
> chr
```

```
[1] "a" "b" "c"
```

Check the number of elements in dbl and chr:

```
> length(dbl)
```

```
[1] 3
```

```
> length(chr)
```

```
[1] 3
```

Check the data type dbl and chr:

```
> typeof(dbl)
```

```
[1] "double"
```

```
> typeof(chr)
```

```
[1] "character"
```

Combine two vectors:

```
> c(dbl,dbl)
```

```
[1] 0.50 0.60 0.25 0.50 0.60 0.25
```

```
> c(dbl, chr)
```

```
[1] "0.5" "0.6" "0.25" "a"  "b"  "c"
```

! The automatic change of the data type of the resulting vector is called **coercion**. Coercion ensures the same data type for each element in the vector is maintained.

3. Lo que debemos saber

Vector arithmetic

Define two new numeric vectors a and b each having 4 elements:

```
> a <- c(1, 2, 3, 4)
> b <- c(10, 20, 30, 40)
```

Multiply each element in a by 5 (scalar multiplication):

```
> a * 5
```

```
[1] 5 10 15 20
```

Multiply the elements in a by the elements in b (vector multiplication):

```
> a * b
```

```
[1] 10 40 90 160
```

Multiply the elements in a by the elements of some numeric vector v of length 5:

```
> v <- c(1.1, 1.2, 1.3, 1.4, 1.5)
> a * v
```

```
Warning in a * v: Länge des längeren Objektes
ist kein Vielfaches der Länge des kürzeren Objektes
```

```
[1] 1.1 2.4 3.9 5.6 1.5
```

! Arithmetic operations of vectors are performed **elementwise**. If two vectors are of unequal length, the shorter vector will be **recycled** in order to match the longer one (here, the first element in a is used again).

3. Lo que debemos saber

Matrices

Option (1): Combining two vectors columnwise with `cbind()`:

```
> A <- cbind(a, b)  # two columns  
> A
```

```
      a  b  
[1,] 1 10  
[2,] 2 20  
[3,] 3 30  
[4,] 4 40
```

Option (2): Combining two vectors rowwise with `rbind()`:

```
> B <- rbind(a, b)  # two rows  
> B
```

```
      [,1] [,2] [,3] [,4]  
a         1     2     3     4  
b        10    20    30    40
```

Option (3): Creating a matrix from elements of a vector with `matrix()`:

```
> A <- matrix(a, ncol=2, nrow=2)  # matrix with 2 columns and 2 rows  
> A
```

```
      [,1] [,2]  
[1,]     1     3  
[2,]     2     4
```

3. Lo que debemos saber

The arguments `nrow` and `ncol` indicate the number of rows and number of columns the resulting matrix consists of.

! For 4 elements and `ncol = 2` the matrix can only have 2 rows. Thus, there is no need to specify both arguments.

```
> A <- matrix(a, ncol=2) # matrix with 2 columns and 2 rows  
> A
```

	[,1]	[,2]
[1,]	1	3
[2,]	2	4

! By default the matrix is filled up column after column (R treats a matrix object internally as a column vector). If the matrix should be filled up row after row the argument `byrow = TRUE` is required.

```
> B <- matrix(a, ncol=2, byrow=TRUE) # matrix filled-up rowwise  
> B
```

	[,1]	[,2]
[1,]	1	2
[2,]	3	4

3. Lo que debemos saber

Matrix actions

Checking the number of rows:

```
> nrow(B)
```

```
[1] 2
```

Checking the number of columns:

```
> ncol(B)
```

```
[1] 2
```

Checking the dimension [nrow, ncol]:

```
> dim(B)
```

```
[1] 2 2
```

Combine two matrices:

```
> D.wide <- cbind(A,A)
```

```
> D.wide
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	3	1	3
[2,]	2	4	2	4

```
> D.long <- rbind(A,A)
```

```
> D.long
```

	[,1]	[,2]
[1,]	1	3
[2,]	2	4
[3,]	1	3
[4,]	2	4

3. Lo que debemos saber

Matrix addition:

```
> B + B
```

```
      [,1] [,2]  
[1,]    2    4  
[2,]    6    8
```

Scalar multiplication:

```
> B * 2
```

```
      [,1] [,2]  
[1,]    2    4  
[2,]    6    8
```

Elementwise multiplication:

```
> B * B
```

```
      [,1] [,2]  
[1,]     1    4  
[2,]     9   16
```

Matrix multiplication:

```
> B %*% B
```

```
      [,1] [,2]  
[1,]     7   10  
[2,]    15   22
```

More matrix arithmetic

- Transpose `t()`

```
> D.wide
```

```
      [,1] [,2] [,3] [,4]  
[1,]     1    3    1    3  
[2,]     2    4    2    4
```

```
> t(D.wide)
```

```
      [,1] [,2]  
[1,]     1    2  
[2,]     3    4  
[3,]     1    2  
[4,]     3    4
```

- Determinant `det()`

```
> det(B)
```

```
[1] -2
```

- Inverse `solve()` (only if `det()` $\neq 0$)

```
> solve(B)
```

```
      [,1] [,2]  
[1,] -2.0  1.0  
[2,]  1.5 -0.5
```

- Eigenvalues `eigen()` (only for square and symmetric matrices)

3. Lo que debemos saber Dataframes

```
> dbl <- c(0.5, 0.6, 0.25, 1.2, 0.333)    # double
> int <- c(9L, 10L, 11L, 12L, 13L)        # integer
> lgl <- c(TRUE, FALSE, FALSE, TRUE, TRUE) # logical
> chr <- c("a", "b", "c", "d", "e")       # character
> df <- data.frame(dbl,int,lgl,chr)
> df
```

	dbl	int	lgl	chr
1	0.500	9	TRUE	a
2	0.600	10	FALSE	b
3	0.250	11	FALSE	c
4	1.200	12	TRUE	d
5	0.333	13	TRUE	e

Data frame actions

Checking the number of rows:

```
> nrow(df)
```

```
[1] 5
```

Checking the number of columns:

```
> ncol(df)
```

```
[1] 4
```

Checking the dimension [nrow, ncol]:

```
> dim(df)
```

```
[1] 5 4
```

3. Lo que debemos saber

Lists

```
> a <- 1L                                # scalar
> dbl <- c(0.5, 0.6, 0.25, 1.2, 0.333)   # numeric vector of length 5
> chr <- c("a", "b", "c"                ) # character vector of length 3
> v <- c(1.1, 1.2, 1.3, 1.4)
> mat <- matrix(v, ncol=2)               # 2 x 2 matrix
> l <- list(a, dbl, chr, mat)
> l

[[1]]
[1] 1

[[2]]
[1] 0.500 0.600 0.250 1.200 0.333

[[3]]
[1] "a" "b" "c"

[[4]]
  [,1] [,2]
[1,] 1.1 1.3
[2,] 1.2 1.4
```

Agenda

4. Manipulación de datos

3. Manipulación de datos – Parte 1

The data science process



3. Manipulación de datos – Parte 1

Discover

- Real-world data is generally noisy, incomplete and inconsistent. The initial exploration of the raw data set helps you to spot such data quality problems.
- Scanning your data also helps you to discover first insights into it and provides guidance on applying the right kind of further statistical treatment to it.

Prepare and Analyse

- Analytical models fed with poor quality data can lead to misleading predictions. The data preparation stage resolves data issues and ensures the dataset used in the modeling stage is acceptable and of improved quality.
- Data preparation tasks are likely to be performed multiple times during interactive data analysis and model building stages (also called **data wrangling** or **data munging**). That's why data preparation typically consumes around 80% of overall time of an analytics project.

3. Manipulación de datos – Parte 1

Stage 1: Discover

Data inspection constitutes a set of simple tools to answer questions like:

Question 1: What is the size the data set?

Question 2: What variables are included?

Question 3: Are there implausible/ missing values?

Question 4: How are values distributed over variables?

3. Manipulación de datos – Parte 1

Ejemplo: Dataset BenAndJerry ice cream- Subsample of the Nielson homescan data, a consumer panel consisting of 70, 000 households and all their purchases.

Question 1: What is the size of the data set?

Check the number of columns and rows (or alternatively the dimension) of the data set:

```
> nrow(BenAndJerry)
```

```
[1] 21974
```

```
> ncol(BenAndJerry)
```

```
[1] 12
```

```
> dim(BenAndJerry)
```

```
[1] 21974    12
```

3. Manipulación de datos – Parte 1

Question 2: What variables are included?

Check the column names:

```
> names(BenAndJerry)
```

[1]	"quantity"	"price_paid_deal"	"price_paid_non_deal"
[4]	"coupon_value"	"promotion_type"	"total_spent"
[7]	"size1_descr"	"flavor_descr"	"formula_descr"
[10]	"household_id"	"female_head_birth"	"male_head_birth"

Display the first observations:

```
> head(BenAndJerry)
```

Display the last observations:

```
> tail(BenAndJerry)
```

3. Manipulación de datos – Parte 1

Display the structure of the data set:

```
> str(BenAndJerry)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':  21974 obs. of  12 variables:
 $ quantity      : int  2 1 1 1 1 2 2 1 1 1 ...
 $ price_paid_deal : num  6.82 3.5 3.5 0 0 0 0 2.14 3.5 3 ...
 $ price_paid_non_deal: num  0 0 0 3 3.99 7.78 7.78 0 0 0 ...
 $ coupon_value   : num  1 0 0 0 0 0 0 0 0 1.25 ...
 $ promotion_type : int  2 1 1 NA NA NA NA 1 1 2 ...
 $ total_spent    : num  37.2 31 15.5 113 67.3 ...
 $ size1_descr    : chr  "16.0 MLOZ" "16.0 MLOZ" "16.0 MLOZ" "16.0 M"..
 $ flavor_descr   : chr  "CAKE BATTER" "VAN CARAMEL FUDGE" "VAN CARA"..
 $ formula_descr  : chr  "REGULAR" "REGULAR" "REGULAR" "REGULAR" ...
 $ household_id   : int  2001456 2001456 2001456 2001637 2002791 2002..
 $ female_head_birth : chr  NA NA NA "10/1/36" ...
 $ male_head_birth : chr  "10/1/61" "10/1/61" "10/1/61" NA ...
```

Here: one line for each column in the data set (including its name, data type and the first few observations) is displayed.

! The `str()`-function gives a reasonable output for any R object by compactly displaying its content. It is particularly well-suited for **list** objects. Recap: a list is a generic vector containing other objects.

3. Manipulación de datos – Parte 1

! Technically, R considers a **data frame** internally as a list object. Thus, a data frame is a **list of equal-length vectors**. Therefore, the `length()` of a data frame is the length of the underlying list and gives the same result as `ncol()`; whereas `nrow()` gives the number of rows.

```
> length(BenAndJerry)
```

```
[1] 12
```

```
> ncol(BenAndJerry)
```

```
[1] 12
```

```
> nrow(BenAndJerry)
```

```
[1] 21974
```

! Use `$` to extract columns by their name!

```
> head(BenAndJerry$price_paid_deal)
```

```
[1] 6.82 3.50 3.50 0.00 0.00 0.00
```

3. Manipulación de datos – Parte 1



Question 3: Are there implausible/ missing values?

Check for observations with 0 or negative purchases:

```
> BenAndJerry$total_spent <= 0
```

```
[1] FALSE FALSE FALSE FALSE FALSE FALSE
```

Here: the result is of type **logical**.

Internally, R treats TRUE and FALSE as 0 and 1 values. Thus, we can sum over all TRUE and FALSE values and do not need to check each single value:

```
> sum(BenAndJerry$total_spent <= 0)
```

```
[1] 0
```

⇒ There are no observations with 0 or negative purchases.

More logical operators

- larger than: >; less than: <
- larger than or equal to: >=; less than or equal to: <=
- exactly equal to: ==; not equal to: !=
- negation of x: !x
- all x values larger than 0 AND smaller than 1: (x > 0) & (x < 1)
- any x value larger than 0 OR smaller than 1: (x > 0) | (x < 1)

! Check for observations with missing purchases (missings are coded with NA):

```
> is.na(BenAndJerry$total_spent)
```

```
[1] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
> sum(is.na(BenAndJerry$total_spent))
```

```
[1] 0
```

⇒ There are no missing values for purchases.

3. Manipulación de datos – Parte 1

Question 4: How are values distributed over variables?

Calculate the average price paid for deal and non deal:

```
> mean(BenAndJerry$price_paid_deal)
```

```
[1] 1.74206
```

```
> mean(BenAndJerry$price_paid_non_deal)
```

```
[1] 2.452375
```

Calculate the spread of the values:

```
> var(BenAndJerry$price_paid_deal)
```

```
[1] 6.519148
```

```
> sqrt(var(BenAndJerry$price_paid_deal))
```

```
[1] 2.553262
```

```
> sd(BenAndJerry$price_paid_deal)
```

```
[1] 2.553262
```

More statistic functions

- `min()` / `max()` (minimum, maximum value)
- `range()` (`max()` - `min()`)
- `median()`
- `mode()`

```
> summary(BenAndJerry)
```


3. Manipulación de datos – Parte II

Stage 2: Prepare

Data preparation covers all activities used to construct the final dataset from the initial raw data

- Creating new data objects (transform or recode values)
- Excluding implausible values (subset)
- Constructing new data sets (aggregate over observations, reshape the data set)

Example: Ben&Jerry ice-cream (continued).

Calculate the `price_paid_deal` and `price_paid_non_deal` per unit. Create two new data objects `Unit.price.deal` and `Unit.price.non.deal`:

```
> Unit.price.deal <- BenAndJerry$price_paid_deal/BenAndJerry$quantity  
> Unit.price.non.deal <- BenAndJerry$price_paid_non_deal/BenAndJerry$quantity
```

Calculate the average unit price paid for deal and non deal:

```
> mean(Unit.price.deal)
```

```
[1] 1.328127
```

```
> mean(Unit.price.non.deal)
```

```
[1] 1.986501
```

3. Manipulación de datos – Parte II

The newly created unit price data vectors can also be appended to the data matrix by using \$:

```
> BenAndJerry$Unit.price.deal <- Unit.price.deal  
> BenAndJerry$Unit.price.non.deal <- Unit.price.non.deal  
> names(BenAndJerry)
```

```
[1] "quantity"           "price_paid_deal"     "price_paid_non_deal"  
[4] "coupon_value"       "promotion_type"      "total_spent"  
[7] "size1_descr"        "flavor_descr"        "formula_descr"  
[10] "household_id"       "female_head_birth"   "male_head_birth"  
[13] "Unit.price.deal"    "Unit.price.non.deal"
```

⇒ The unit price data vectors are added on the last position (the last two columns).

3. Manipulación de datos – Parte II

It might be useful to also examine `Unit.price.deal` and `Unit.price.non.deal` in combination with other attributes like `size1_descr` (package size), `formula_descr` (fat) and `flavor_descr` (flavor). This can be done by using the `aggregate()`-function:

```
aggregate(formula, data, FUN, subset, na.action = na.omit, ...)
```

- formula: A formula argument where the response variables are numeric data, to be split into groups according to the values in the predictor variables.
- data: data frame or list.
- FUN: function to compute the summary statistics, for example, `mean()` or `var()`.
- subset: specify a subset of the observations to be analysed (see also `subset()`).
- na.action: what should happen with observations containing NA (missing) values? (ignored by default)

3. Manipulación de datos – Parte II

! In R, formulas are used to express a relationship between variables. Most commonly, the relationship between one **response** and one (or several) **predictor** variable(s) is described.

- The formula is characterized by the tilde ~ symbol. The **response** variable stands on the **left** hand side and the **predictor** variable on the **right** hand side of the tilde:

```
response ~ predictor
```

- Predictor variables can be added with the plus + symbol:

```
response ~ predictor1 + predictor2
```

More formulas

- leave out predictor2:

```
response ~ predictor1 - predictor2
```

- tensor product (interactions) of predictor1 and predictor2:

```
response ~ predictor1 : predictor2
```

- crossing:

```
response ~ predictor1 * predictor2
```

same as

```
response ~ predictor1 + predictor2 + predictor1 : predictor2
```

In the following, we consider `Unit.price.deal` as the response variable and `size1_descr` as the predictor variable for specifying the formula argument in the `aggregate()`-function:

3. Manipulación de datos – Parte II

```
> aggregate(Unit.price.deal ~ size1_descr,  
+           FUN = mean, data = BenAndJerry)
```

	size1_descr	Unit.price.deal
1	16.0 MLOZ	1.3355396
2	32.0 MLOZ	0.8940921

⇒ Groups the average Unit.price.deal according to the different package sizes.

```
> aggregate(Unit.price.deal ~ size1_descr + formula_descr,  
+           FUN = mean, data = BenAndJerry)
```

```
> aggregate(Unit.price.deal ~ size1_descr * formula_descr,  
+           FUN = mean, data = BenAndJerry)
```

	size1_descr	formula_descr	Unit.price.deal
1	16.0 MLOZ	LIGHT HALF THE FAT	1.7091622
2	16.0 MLOZ	REGULAR	1.3290296
3	32.0 MLOZ	REGULAR	0.8940921

⇒ Groups the mean of Unit.price.deal according to package size crossed with amount of fat (factorial design).

3. Manipulación de datos – Parte II

```
> aggregate(Unit.price.deal ~ size1_descr + formula_descr,  
+           FUN = mean, subset = total_spent > 0 , data=BenAndJerry)
```

	size1_descr	formula_descr	Unit.price.deal
1	16.0 MLOZ LIGHT HALF THE FAT		1.7091622
2	16.0 MLOZ	REGULAR	1.3290296
3	32.0 MLOZ	REGULAR	0.8940921

⇒ Only observations with purchases larger than 0 are considered (i.e. purchases smaller than or equal to 0 are excluded).

```
> aggregate(cbind(Unit.price.deal,Unit.price.non.deal) ~ size1_descr + formula_descr,  
+           FUN = mean, subset = total_spent > 0 , data = BenAndJerry)
```

	size1_descr	formula_descr	Unit.price.deal	Unit.price.non.deal
1	16.0 MLOZ LIGHT HALF THE FAT		1.7091622	1.701405
2	16.0 MLOZ	REGULAR	1.3290296	1.956975
3	32.0 MLOZ	REGULAR	0.8940921	3.971491

⇒ Groups both the average Unit.price.deal and Unit.price.non.deal according package size and fat.



¡GRACIAS!