INFOX educación sin fronteras

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

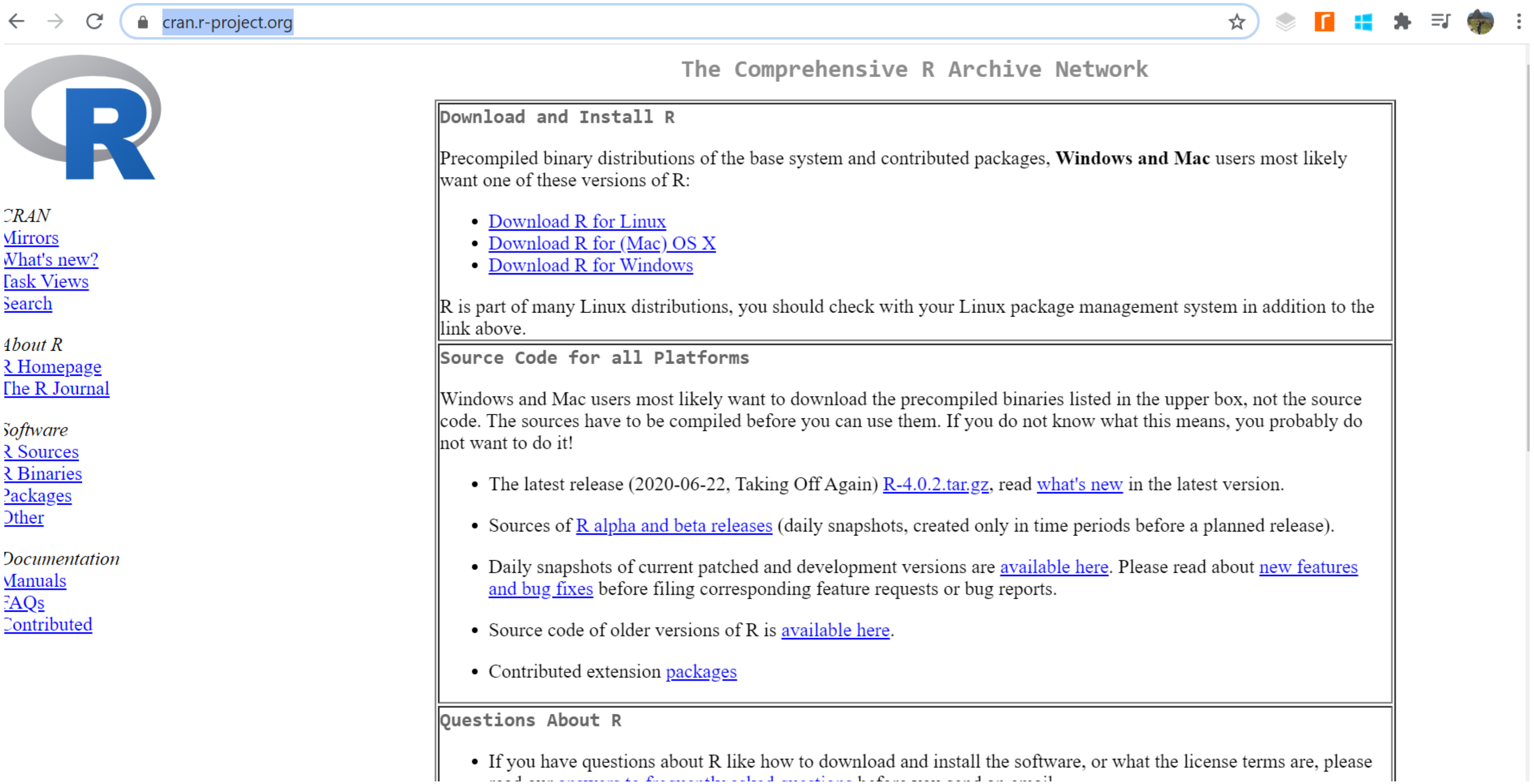
INFOX
educación sin fronteras

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



cran.r-project.org

The Comprehensive R Archive Network

:RAN
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

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

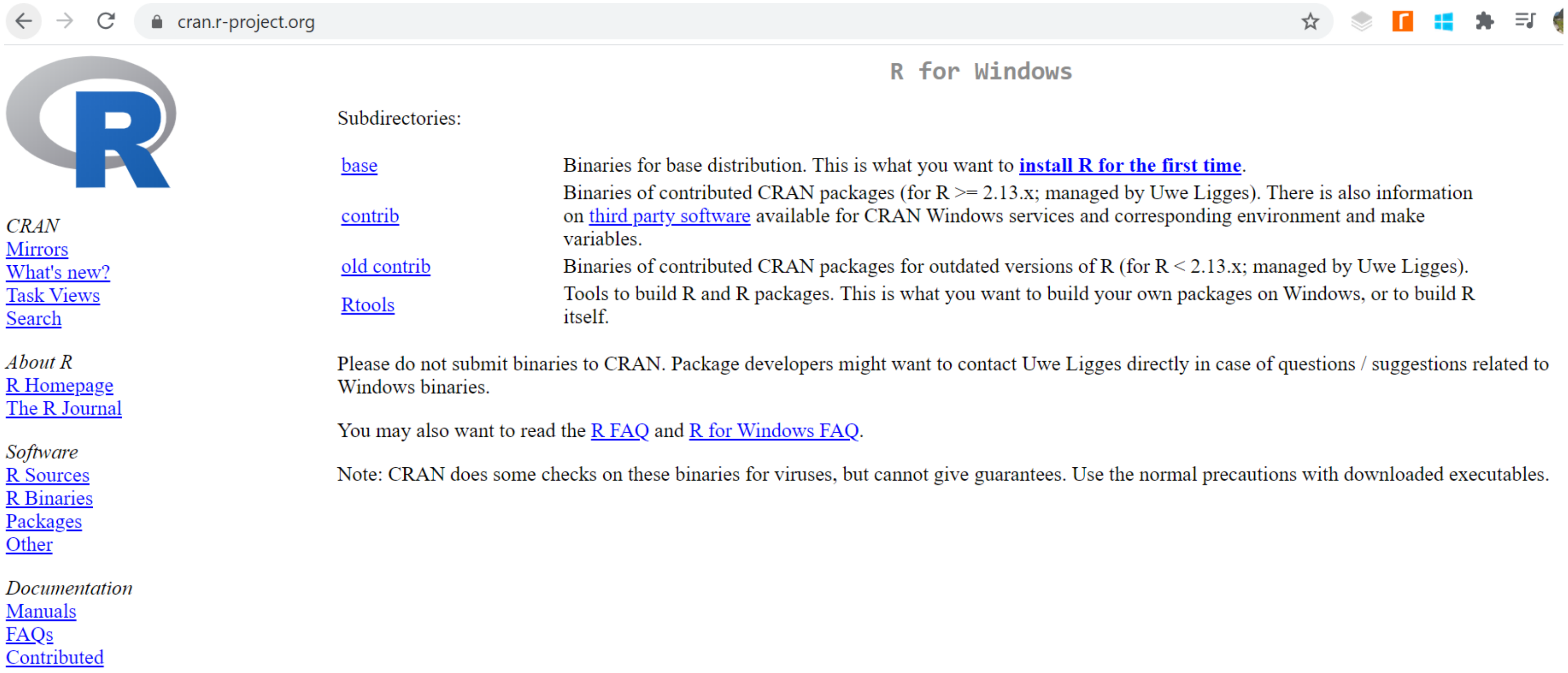
Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- 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

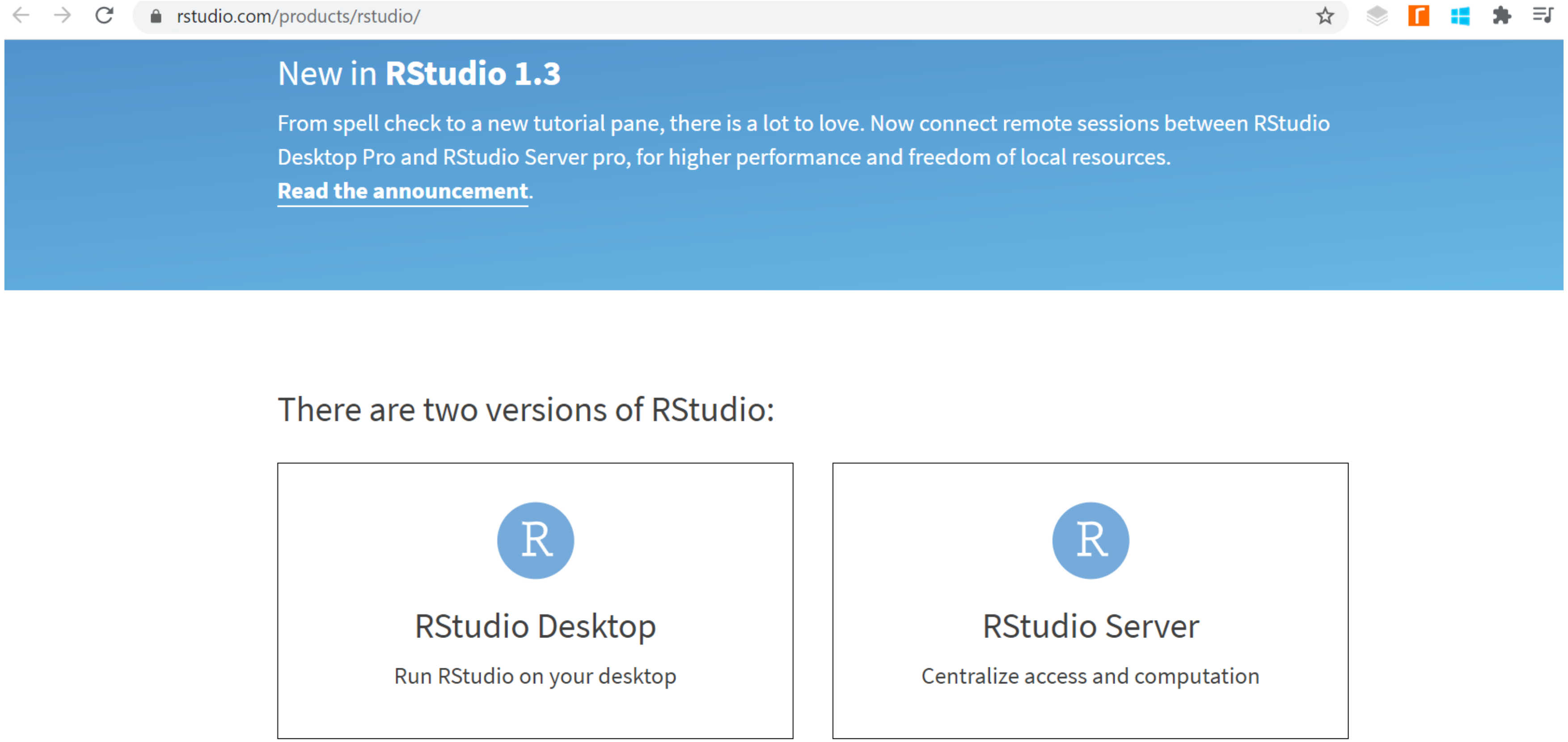
- If you have questions about R like how to download and install the software, or what the license terms are, please

Descargando e instalando el software



Descargando e instalando el software

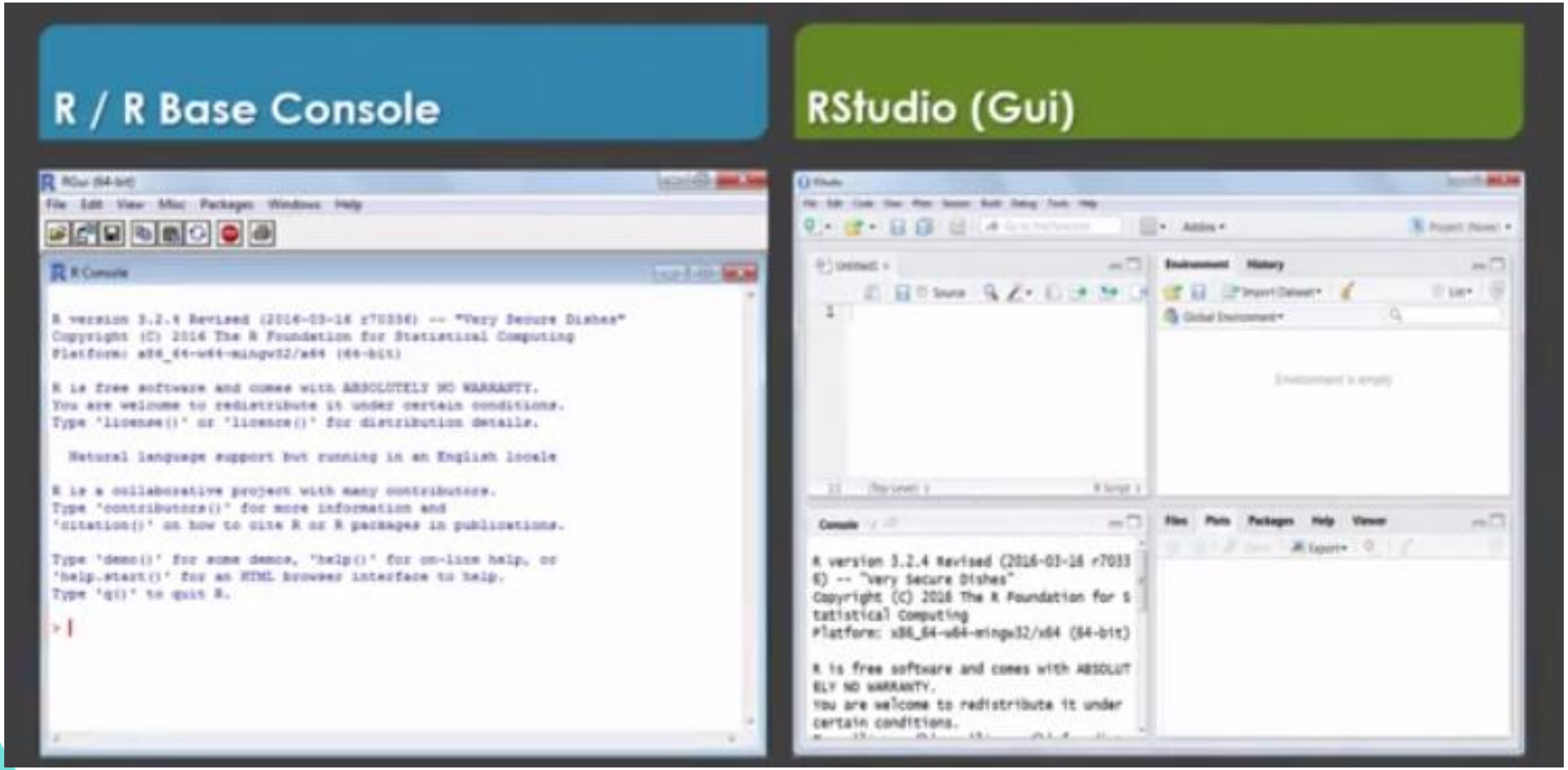
Una vez que tenemos el software R instalado, procedemos a instalar RStudio



INFOX
educación sin fronteras

Descargando e instalando el software

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



Agenda

2. Qué nos ofrece este software?

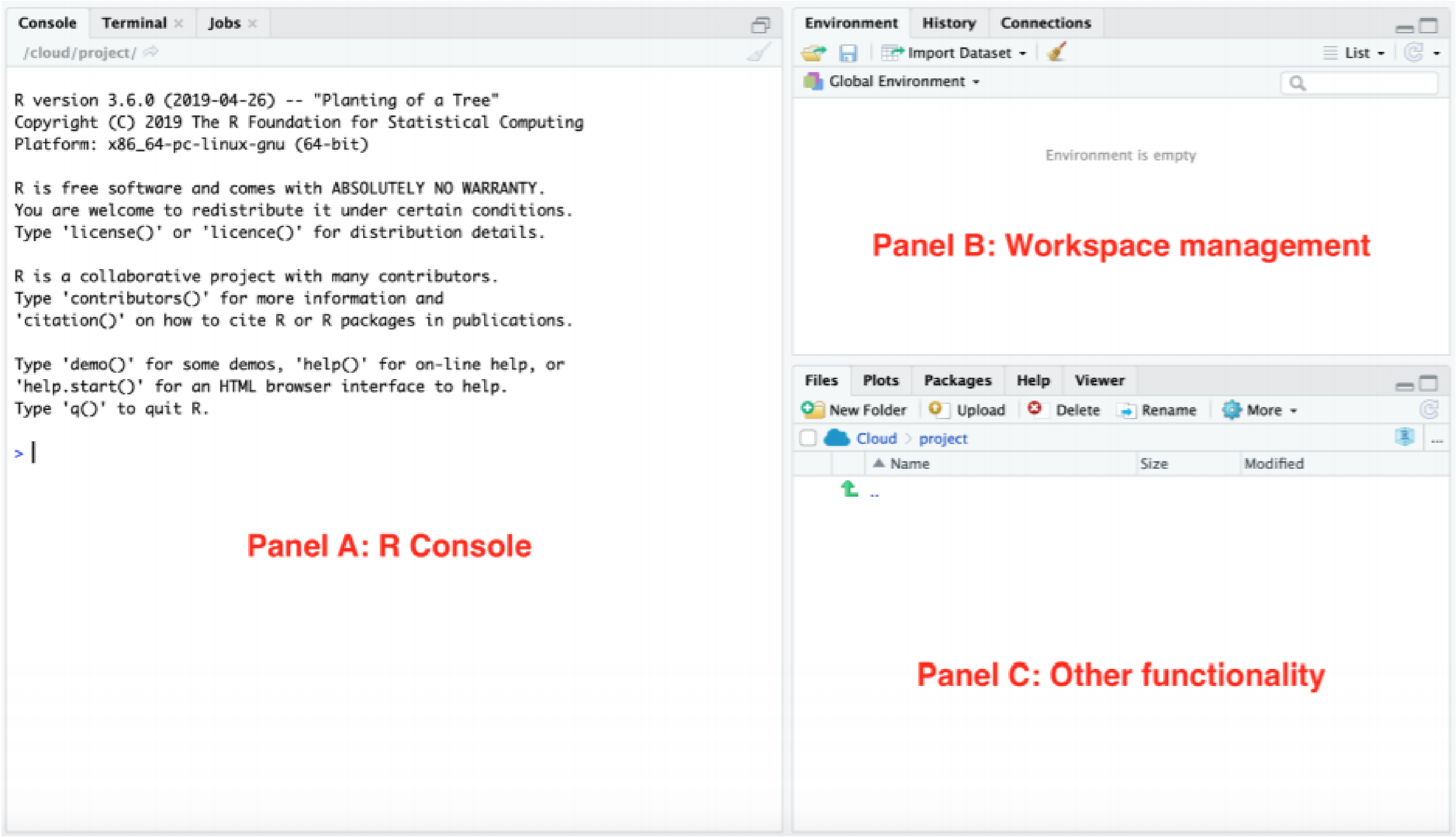
INFOX
educación sin fronteras

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



2.1. Script (.R)

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

Ventajas:

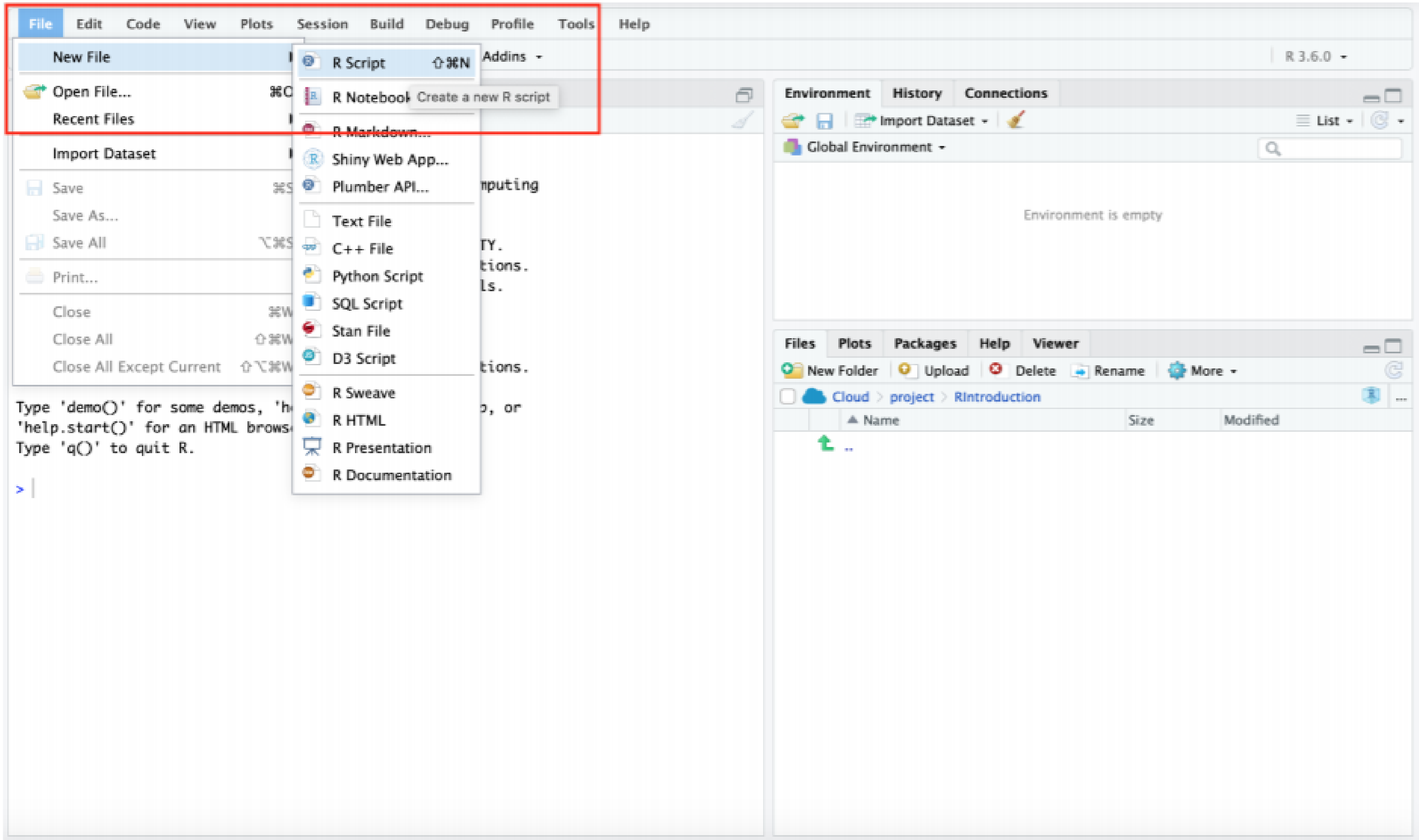
- Documentación de tareas
- Automatización de tareas repetitivas
- Evaluación de cambios

INFOX
educación sin fronteras

2.1. Script (.R)

Script (.R)

Paso 1: Creamos un script file

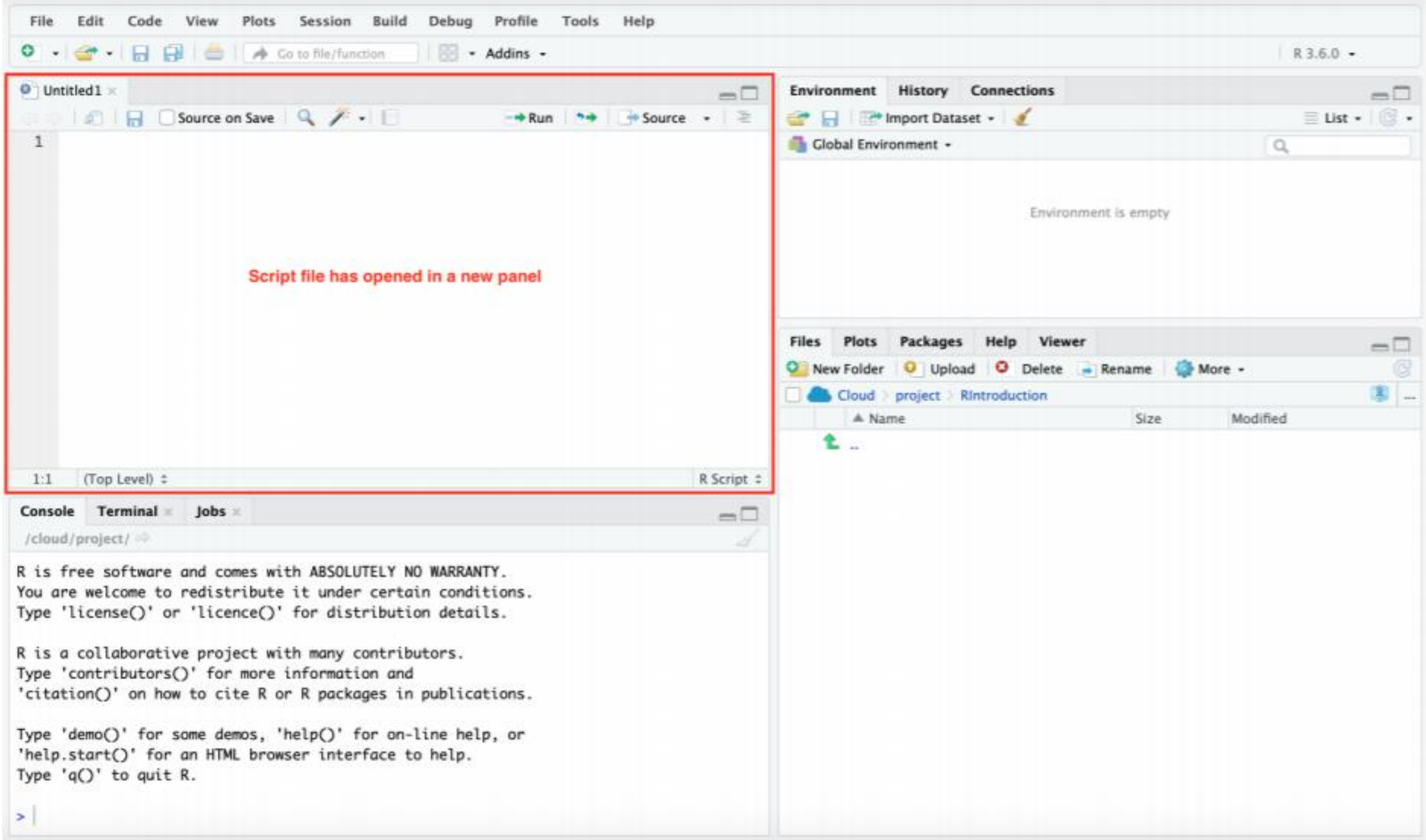


INFOX educación sin fronteras

2.1. Script (.R)

Script (.R)

Paso 2: Creamos un script file

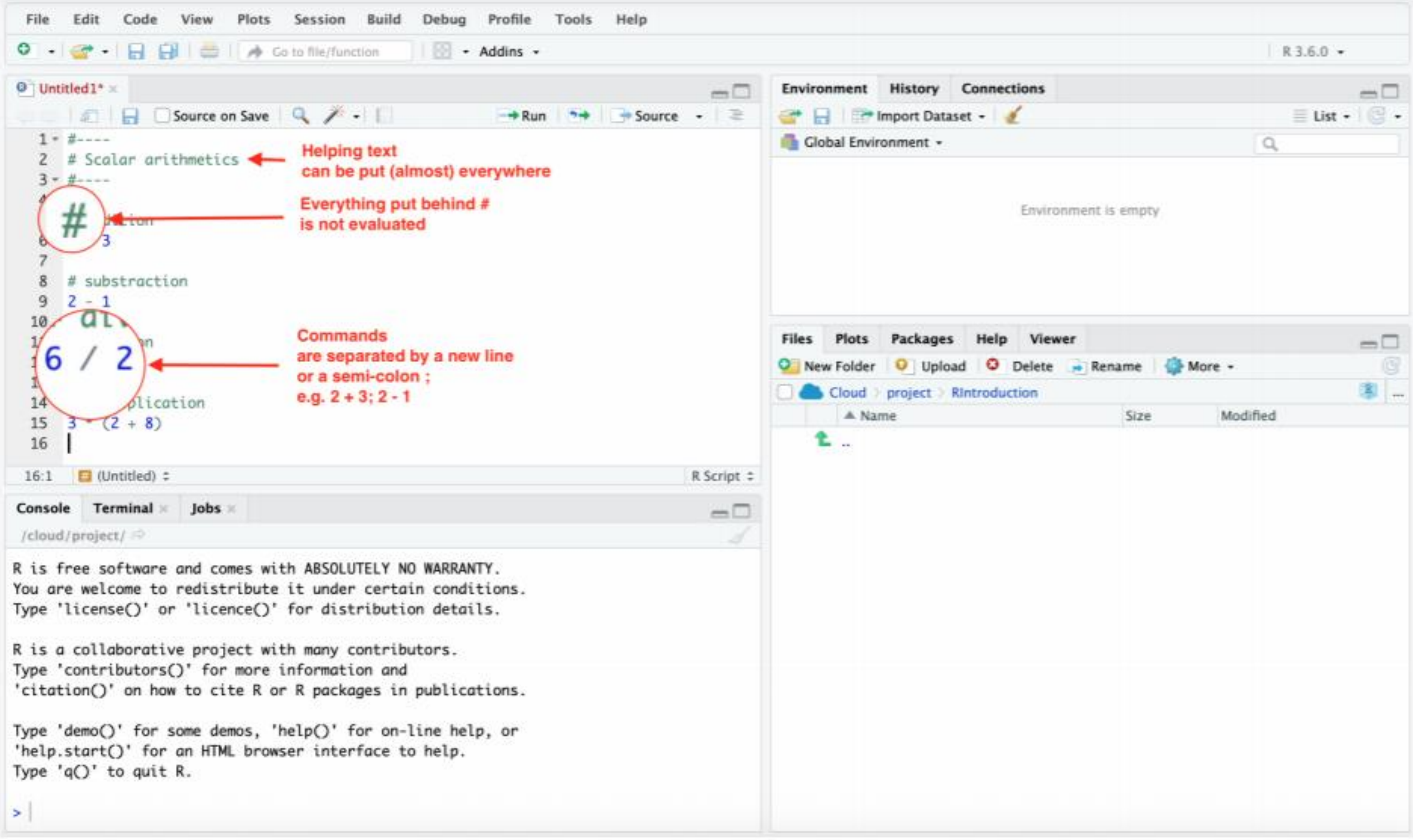


INFOX
educación sin fronteras

2.1. Script (.R)

Script (.R)

Paso 3: Escribimos el script

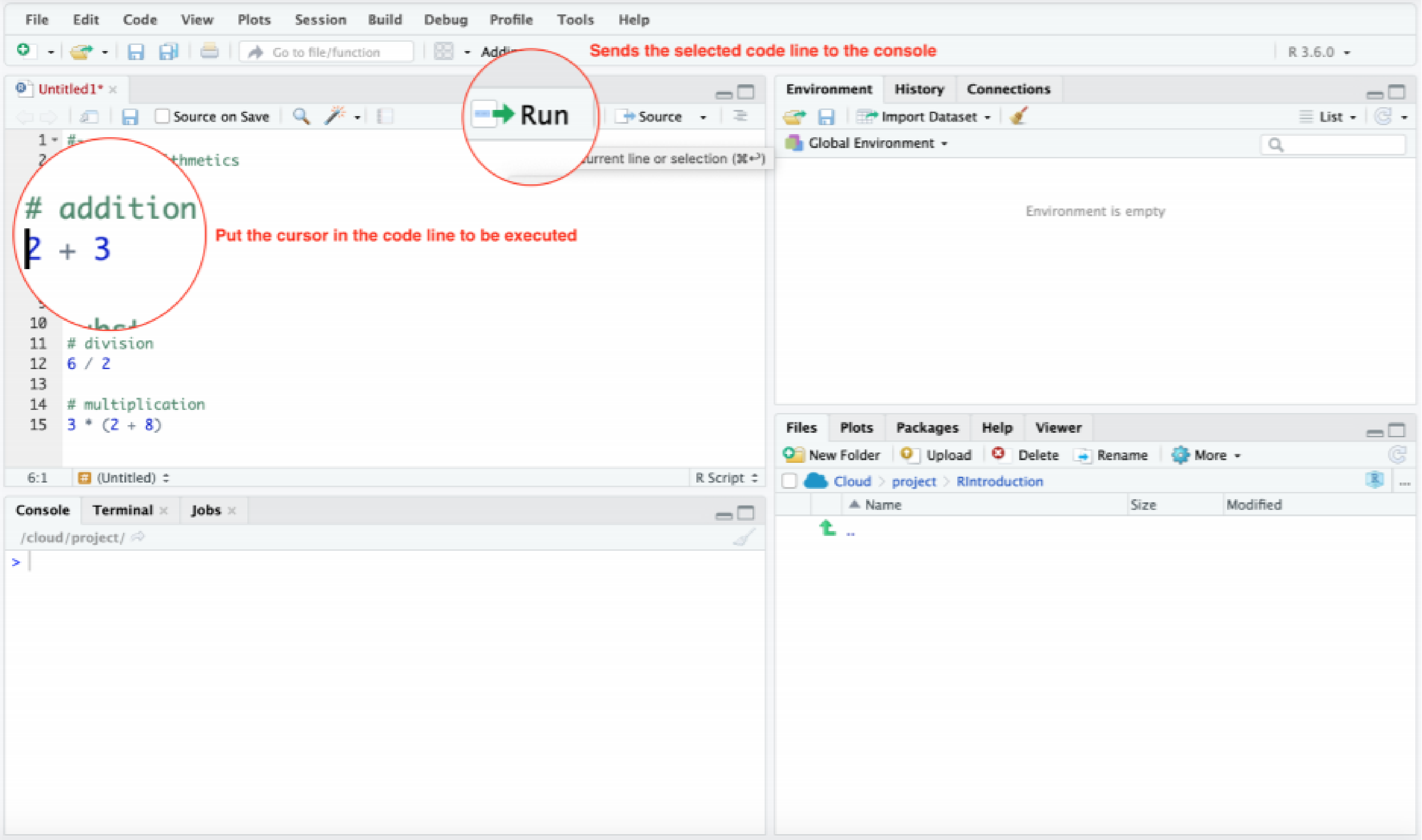


INFOX
educación sin fronteras

2.1. Script (.R)

Script (.R)

Paso 3: Corremos el script

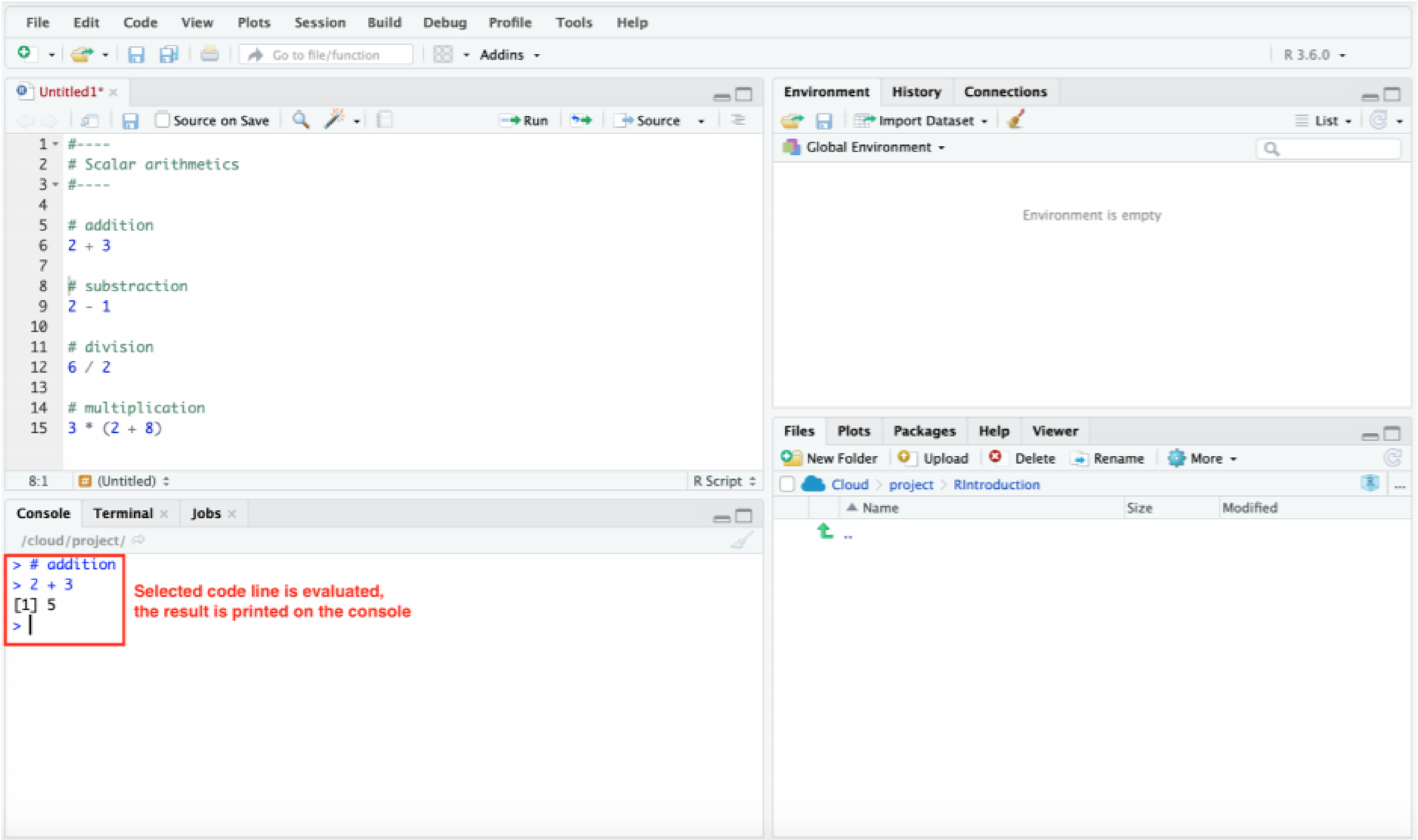


INFOX
educación sin fronteras

2. Qué nos ofrece software?

Script (.R)

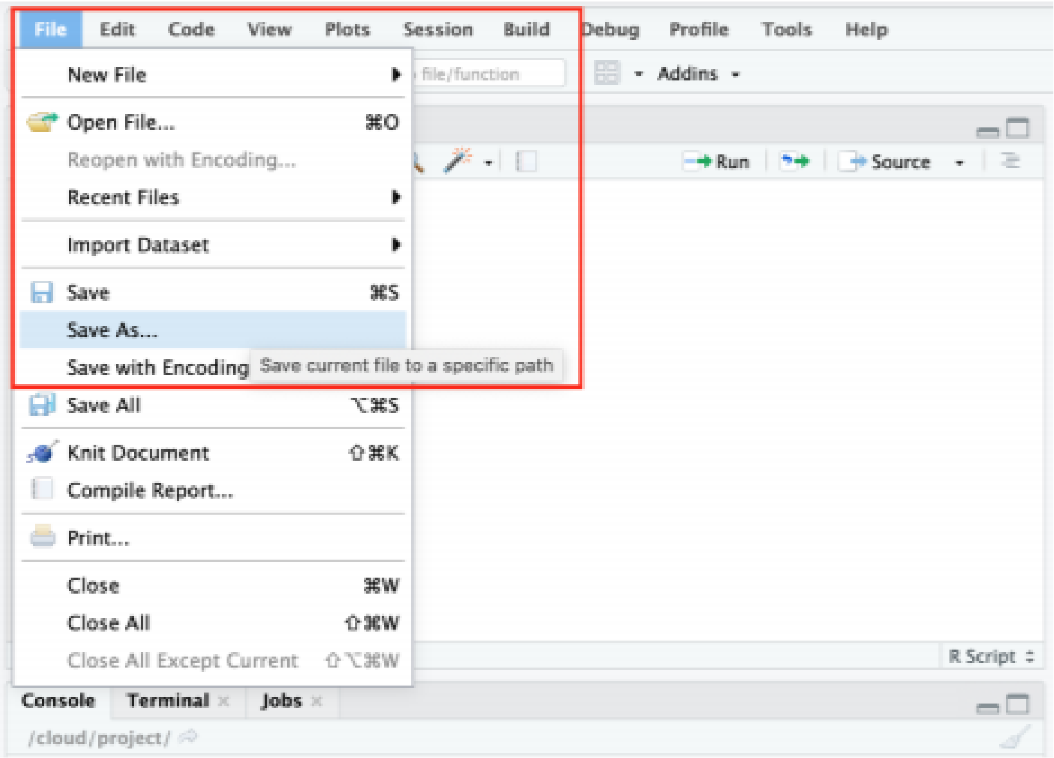
Paso 3: Corremos el script

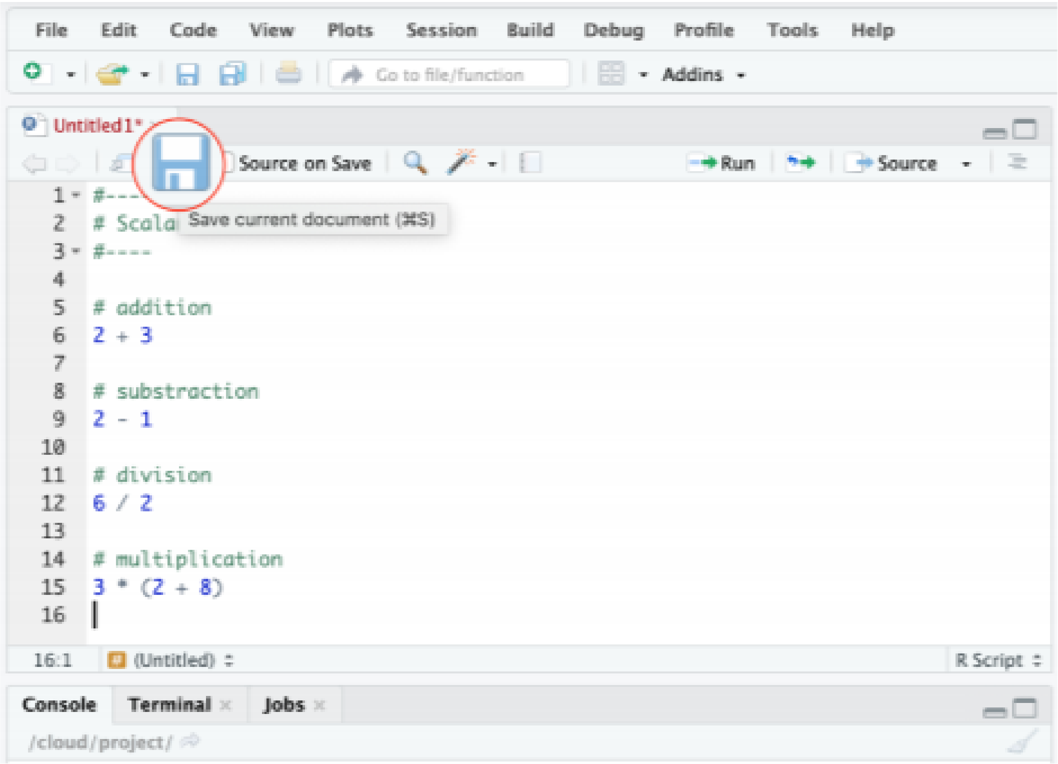


2.1. Script (.R)

Script (.R)

Paso 4: Guardamos el script

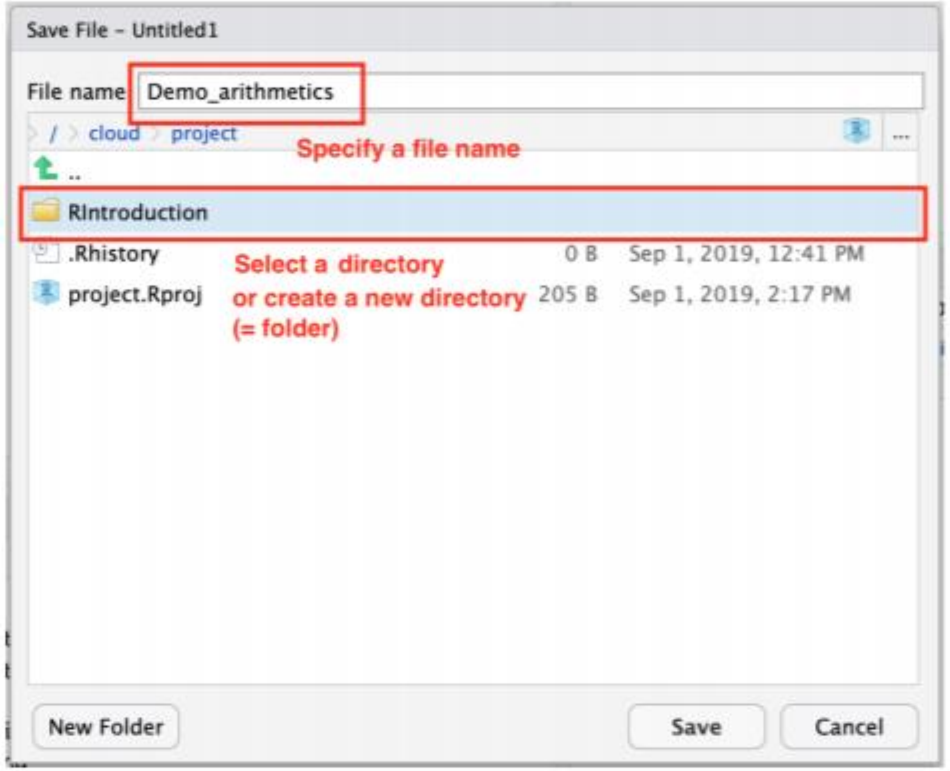




2.1. Script (.R)

Script (.R)

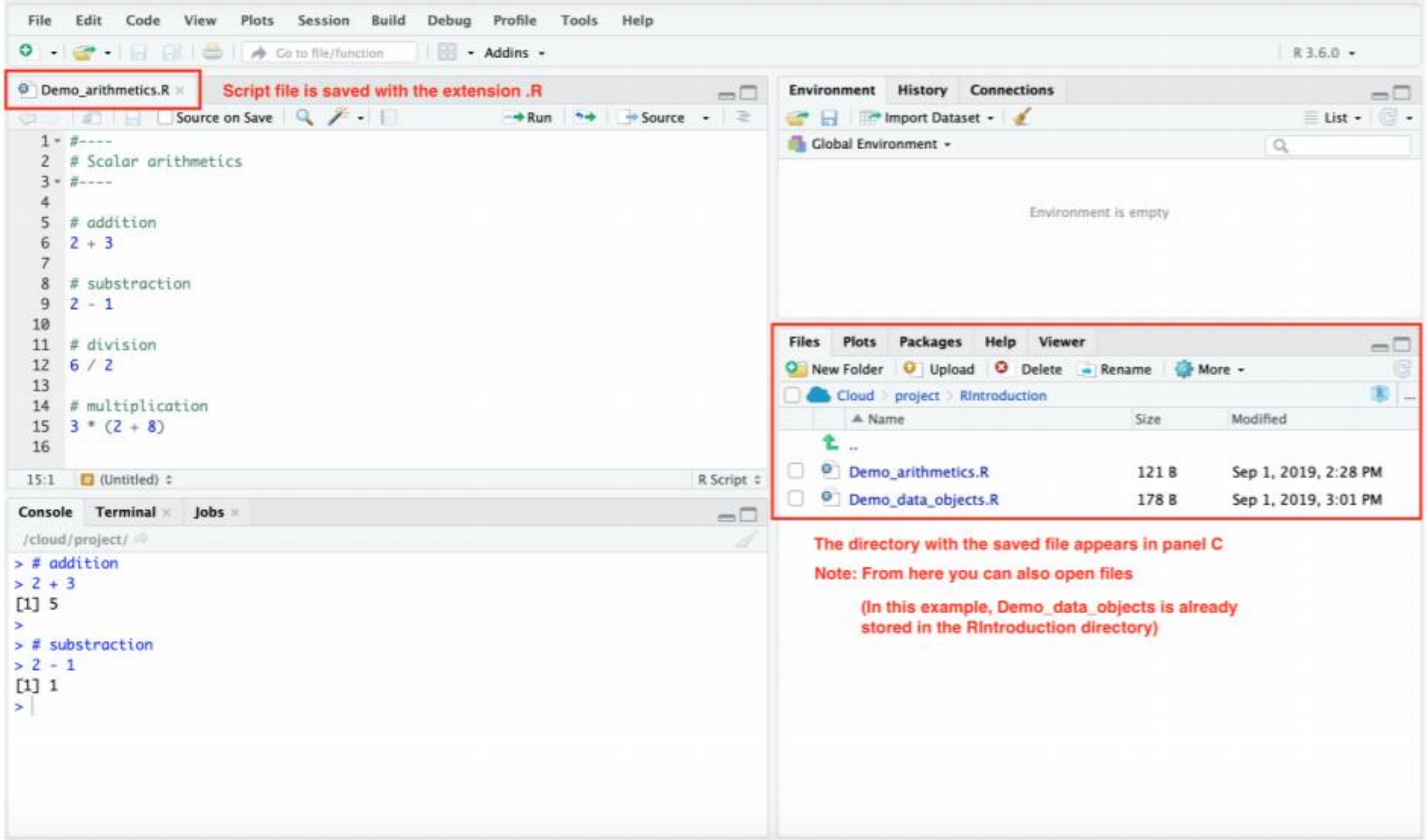
Paso 5: Guardamos el script



2.1. Script (.R)

Script (.R)

Paso 6: Guardamos el script

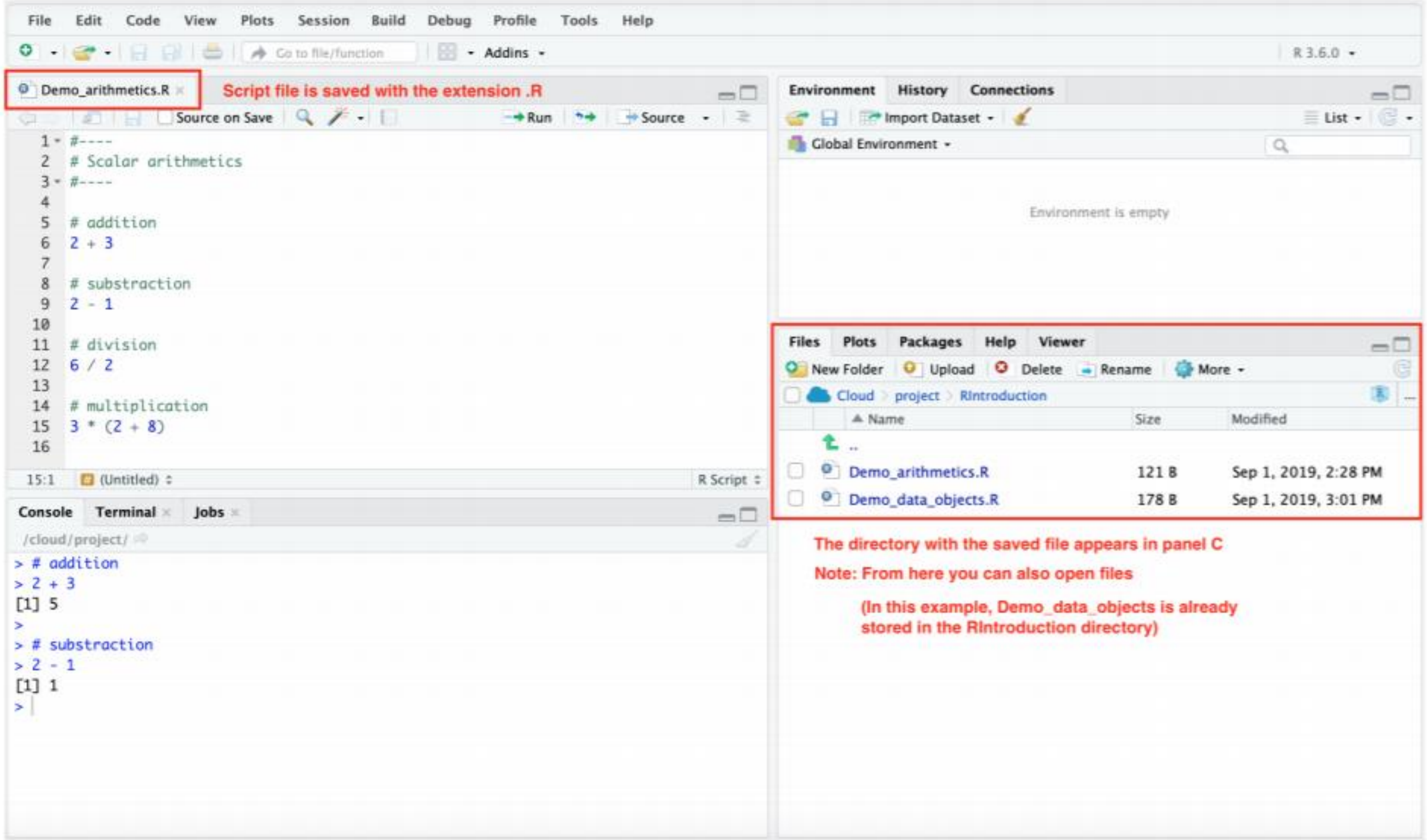


INFOX
educación sin fronteras

2.1. Script (.R)

Script (.R)

Paso 6: Guardamos el script



INFOX educación sin fronteras

2.2. Shortcuts

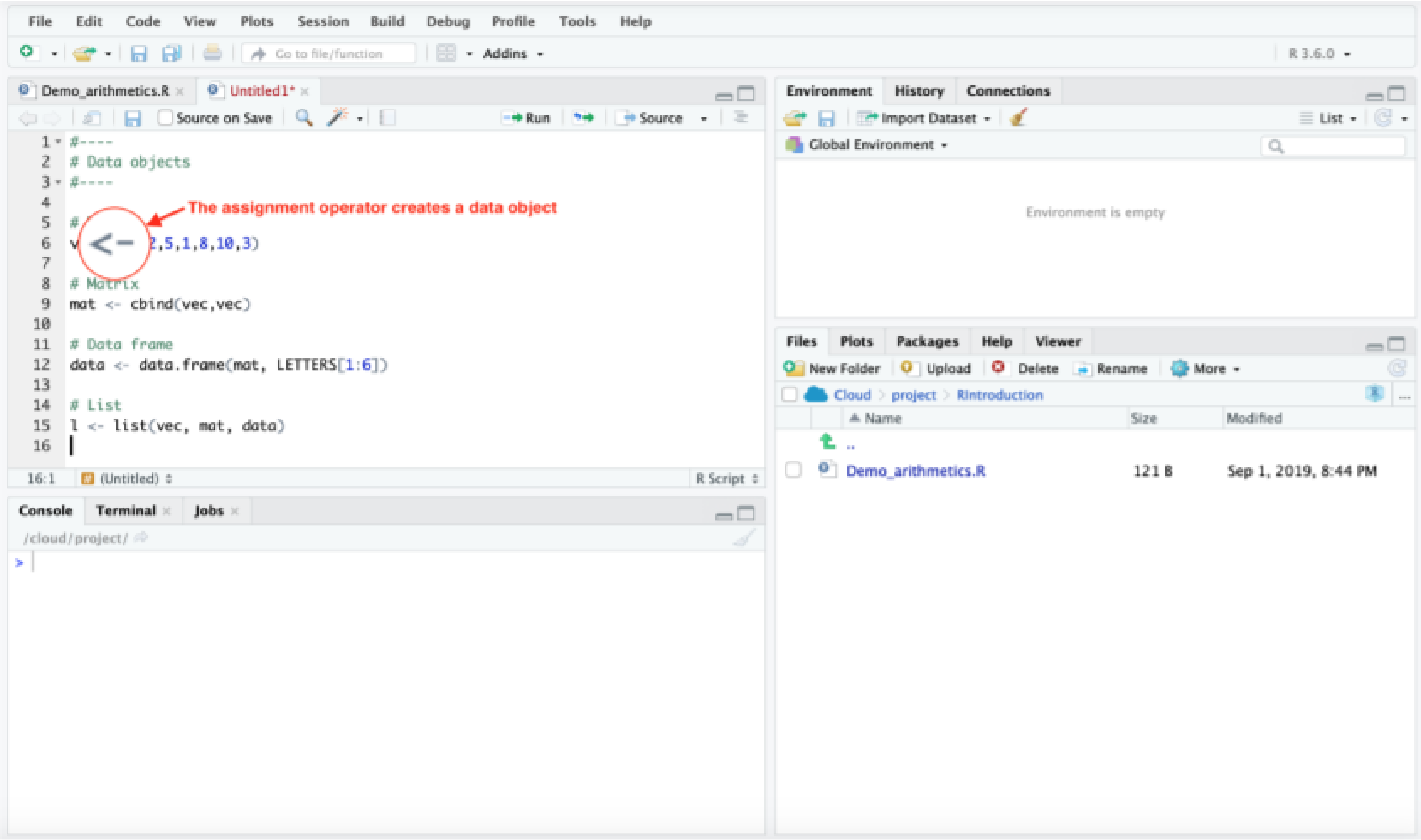
	Windows (Linux)	Mac
New R Script file	Shift + Ctrl +N	Shift + CMD +N
Open file	Ctrl + O	Ctrl + O
Run current line/selection and jump to next line	Ctrl + Enter	CMD + Enter
Run current line/selection DONT jump to next line	Alt + Enter	Alt + Enter
Run whole script	Ctrl + Alt + R	CMD + Alt + R
Run code from 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

INFOX
educación sin fronteras

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.

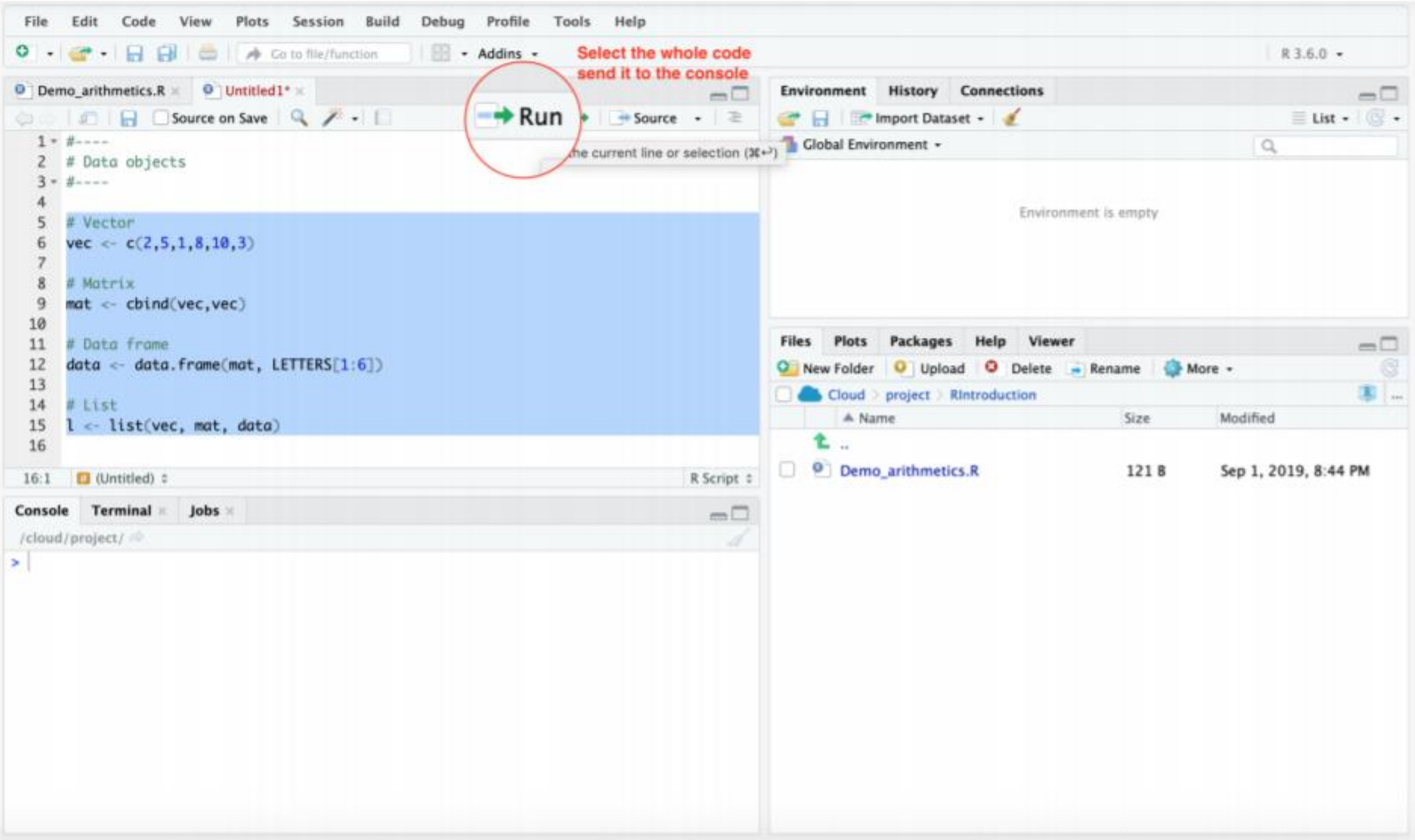
Creating data objects - Step 1



INFOX
educación sin fronteras

2.3. Workspace (.Rdata)

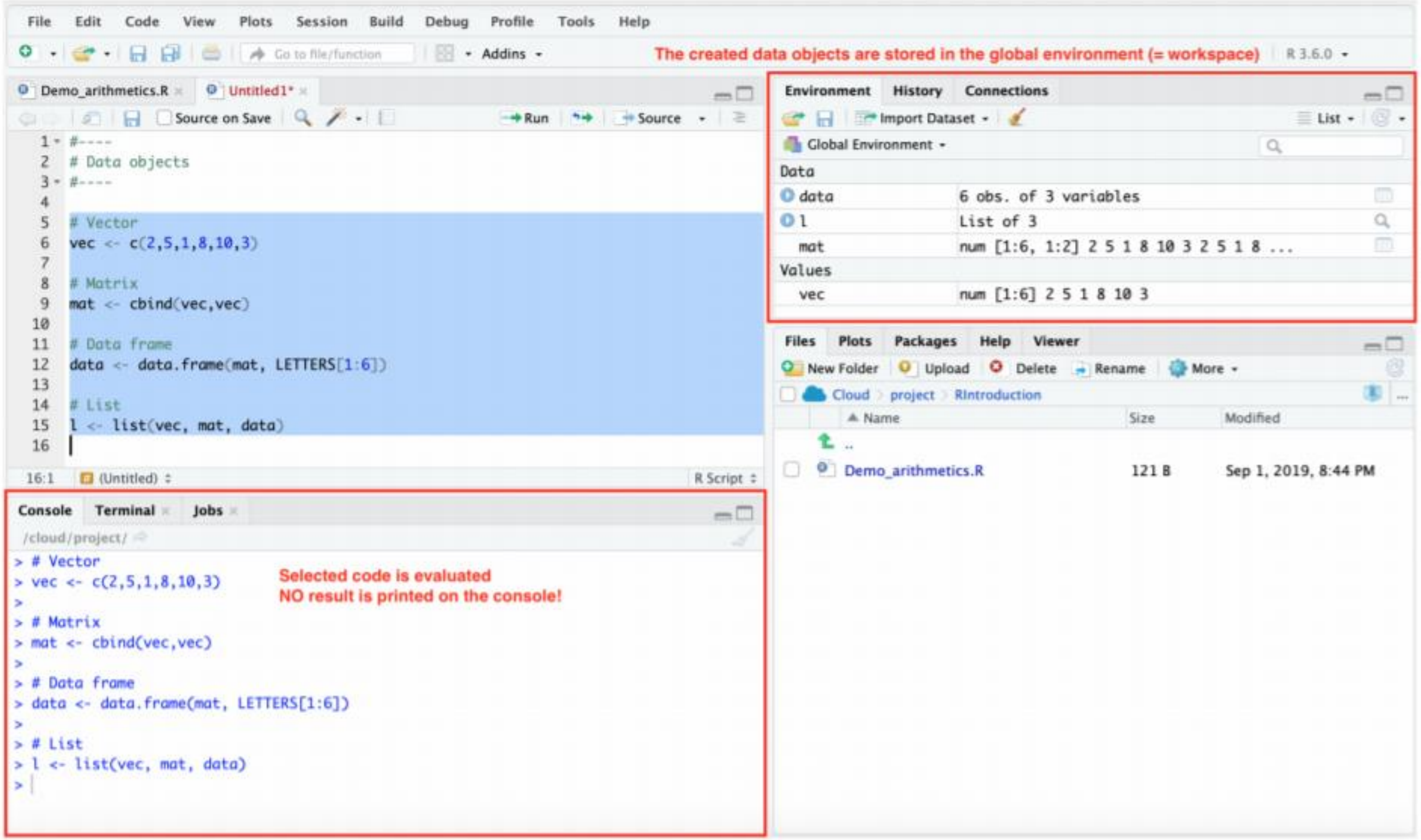
Creating data objects - Step 2



INFOX
educación sin fronte

2.3. Workspace (.Rdata)

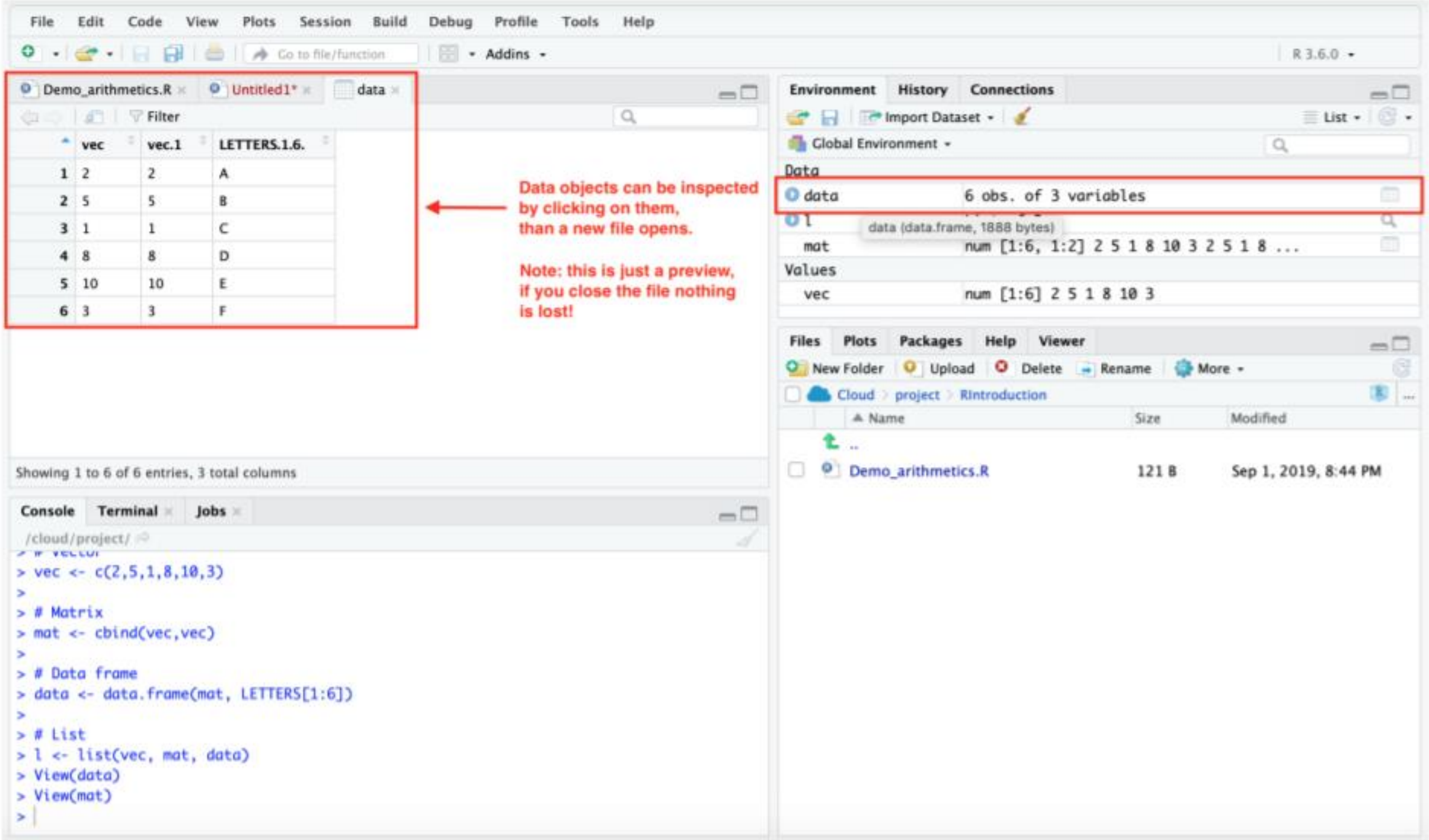
Creating data objects - Step 3



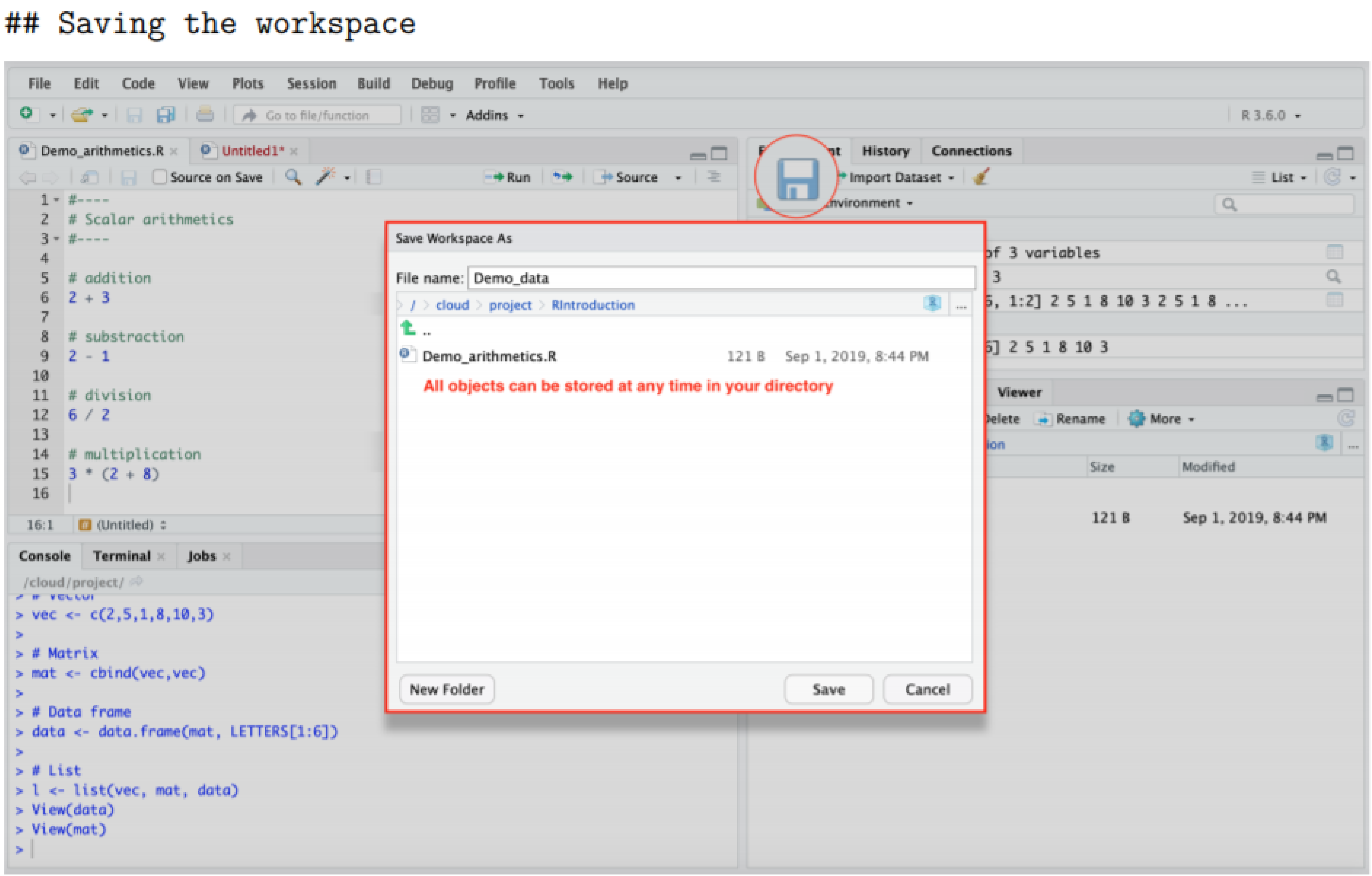
INFOX
educación sin fronteras

2.3. Workspace (.Rdata)

Inspecting data objects

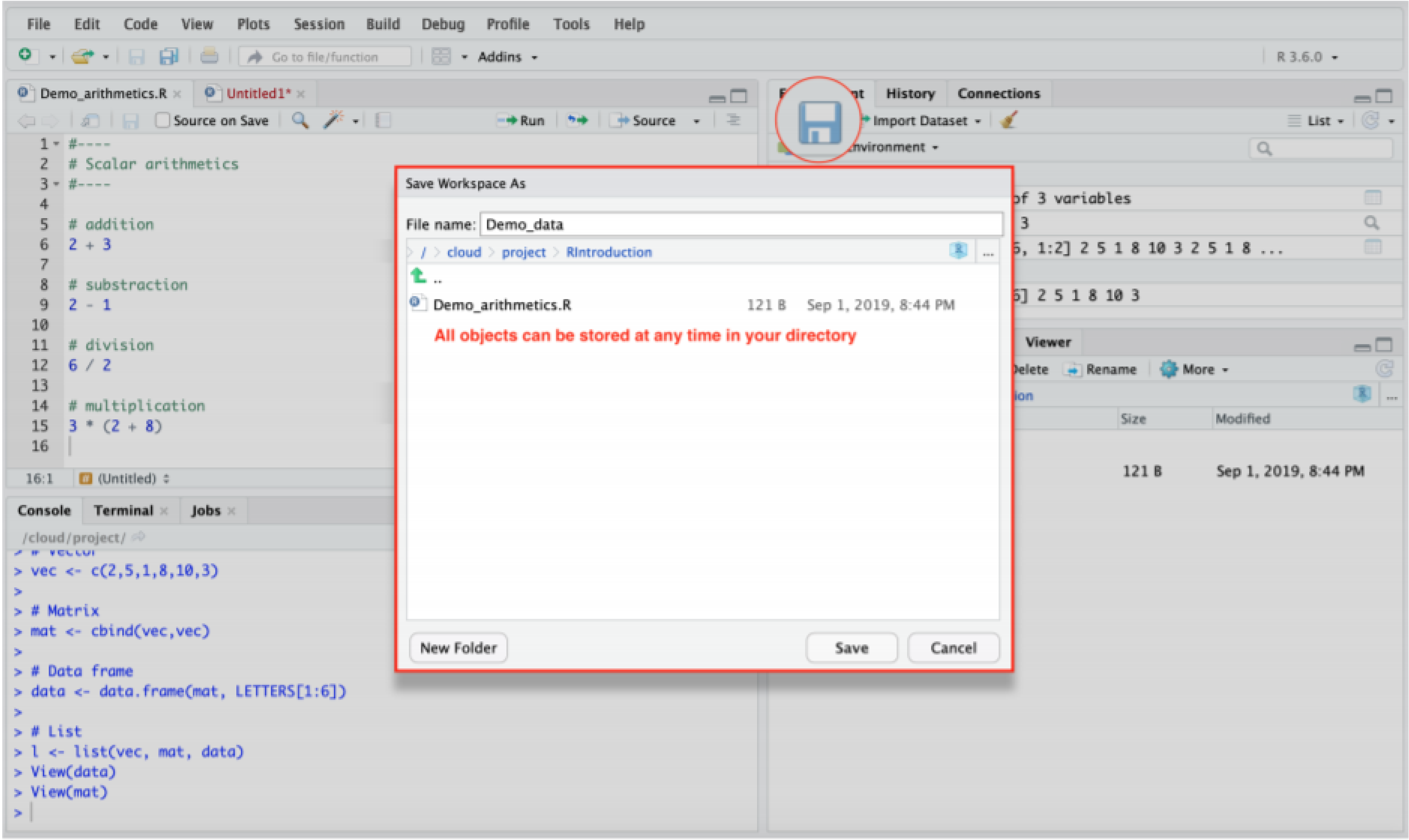


2.3. Workspace (.Rdata)



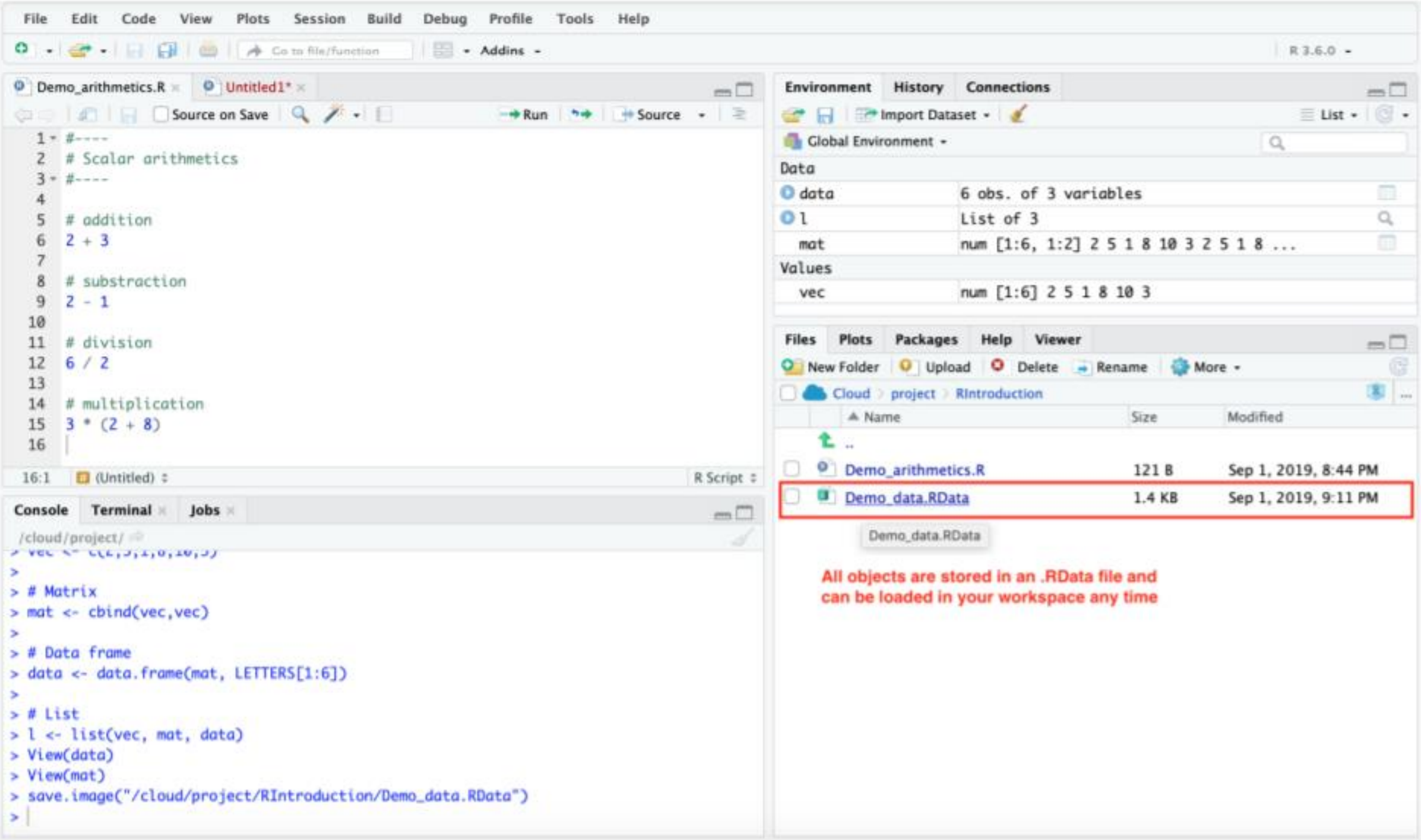
2.3. Workspace (.Rdata)

Saving the workspace



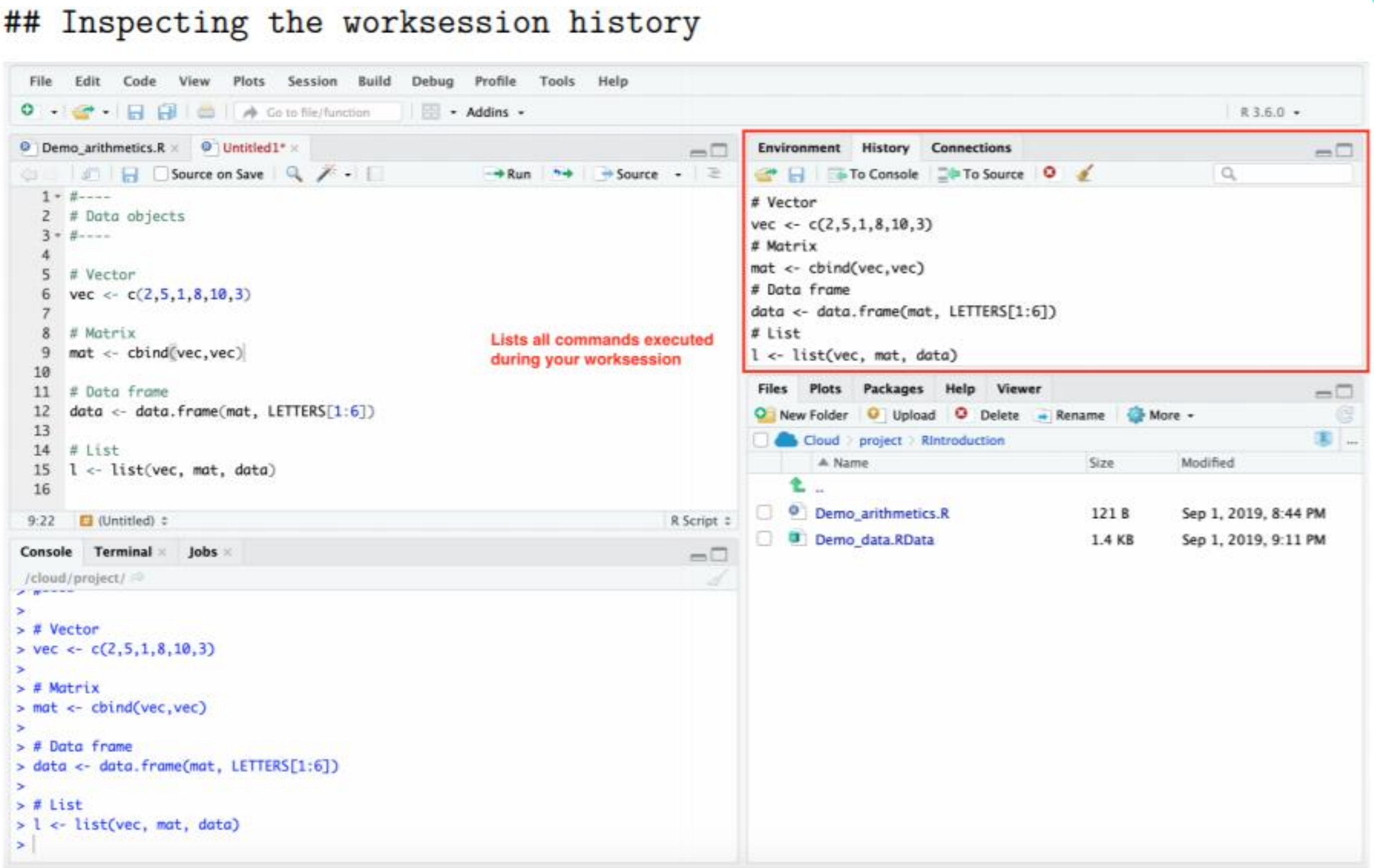
2.3. Workspace (.Rdata)

Loading the workspace



2.4. History (.Rhistory)

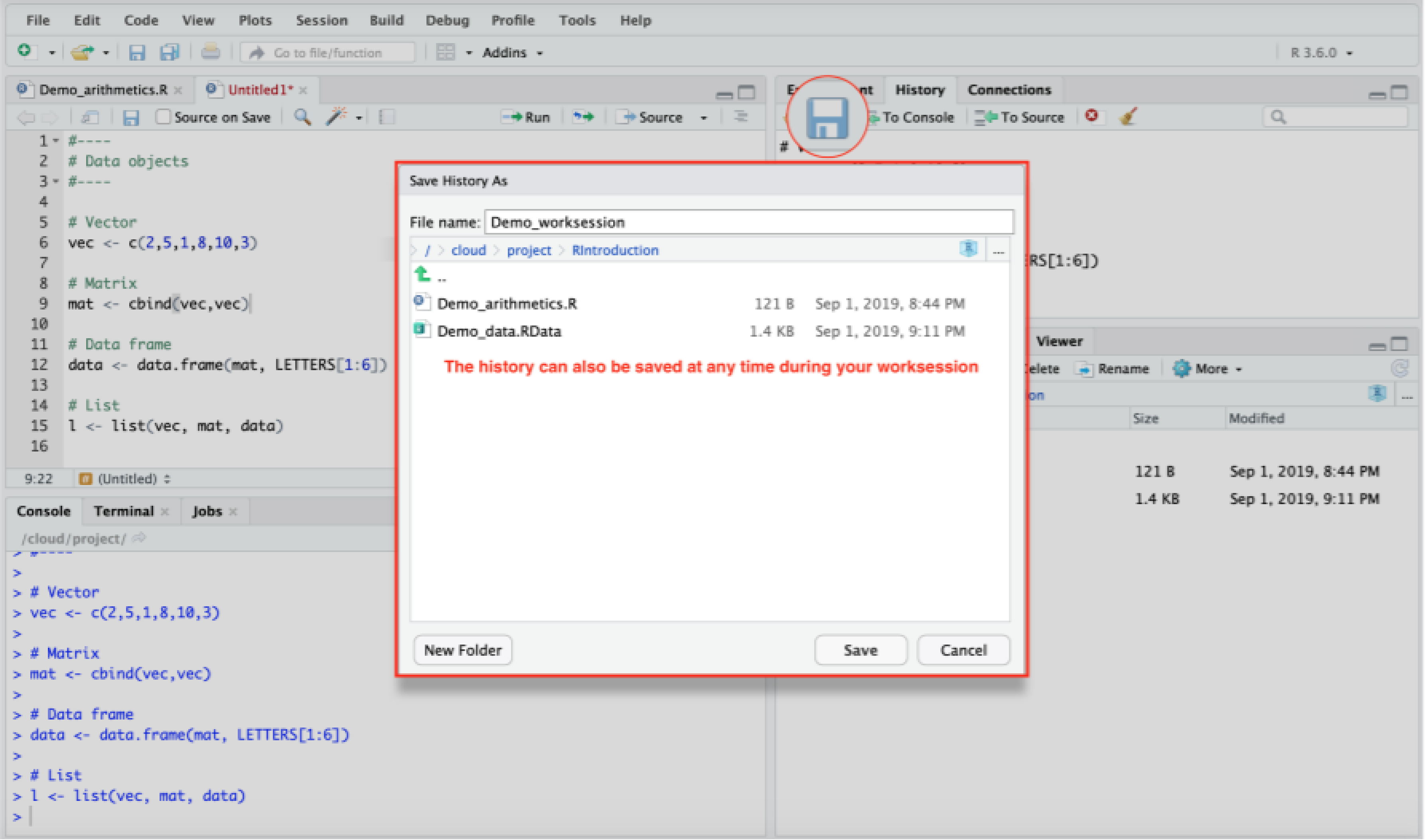
El History file es un text file que lista todos los comandos que se han ido ejecutando durante la sesión.



INFOX
educación sin fronteras

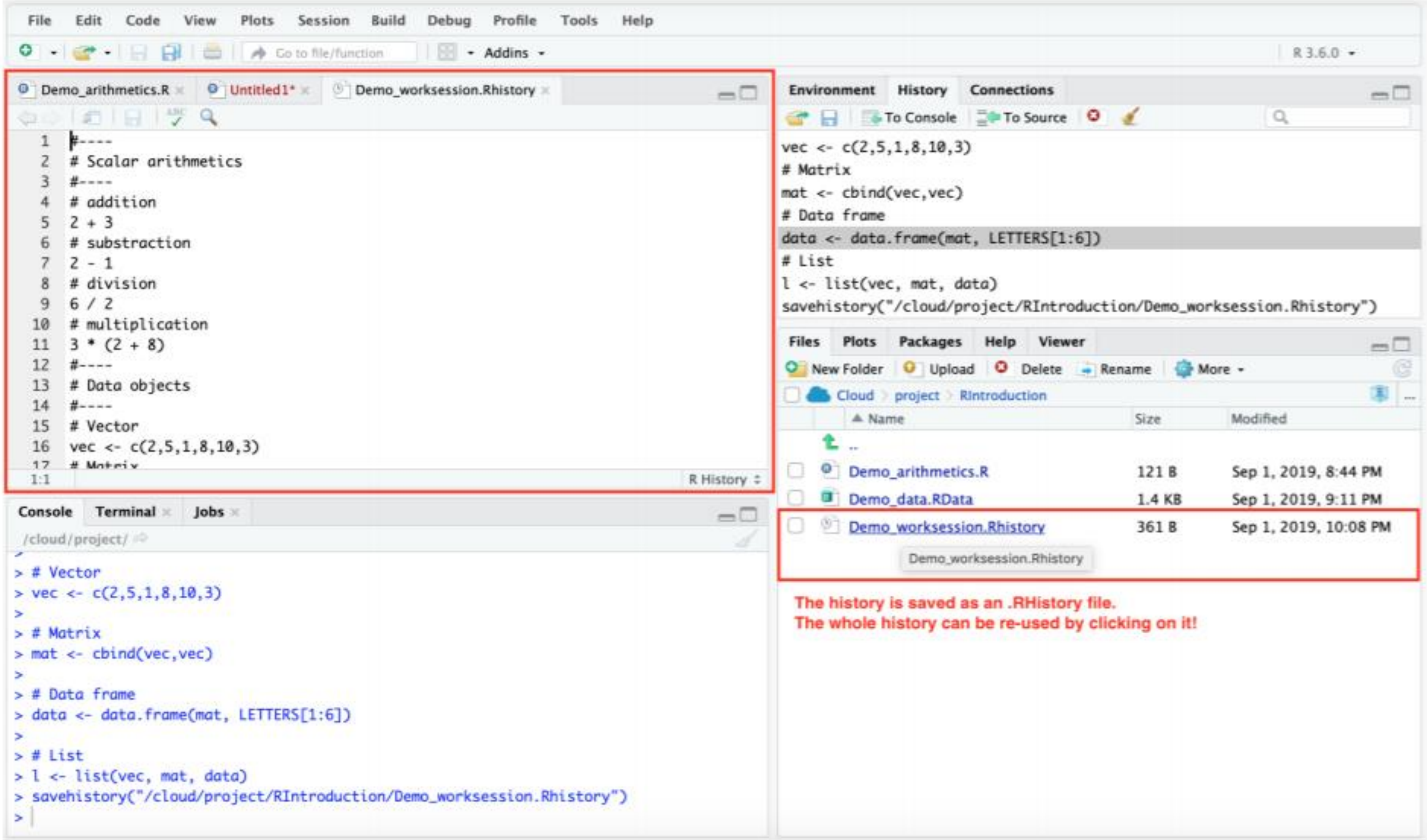
2.4. History (.Rhistory)

Saving the worksession history



2.4. History (.Rhistory)

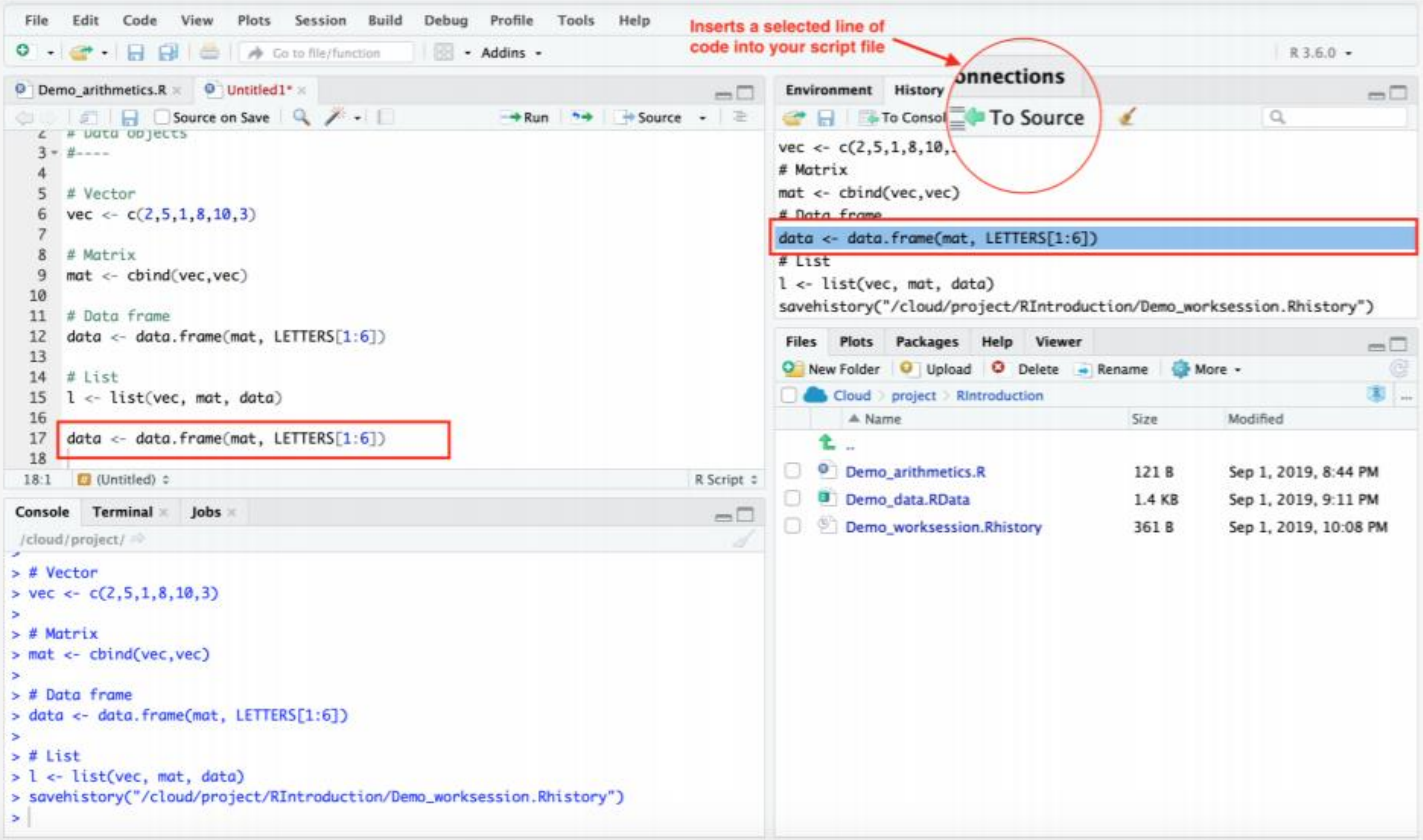
Recycling the worksession history



INFO
educación sin fron

2.4. History (.Rhistory)

Recycling the worksession history



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)

INFOX
educación sin fronteras

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

INFOX
educación sin fronteras

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
```

INFOX
educación sin fronteras

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).

INFOX
educación sin fronteras

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
```

INFOX
educación sin fronteras

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)

INFOX
educación sin fronteras

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
```

INFOX
educación sin fronteras

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"       )           # charcter 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
```

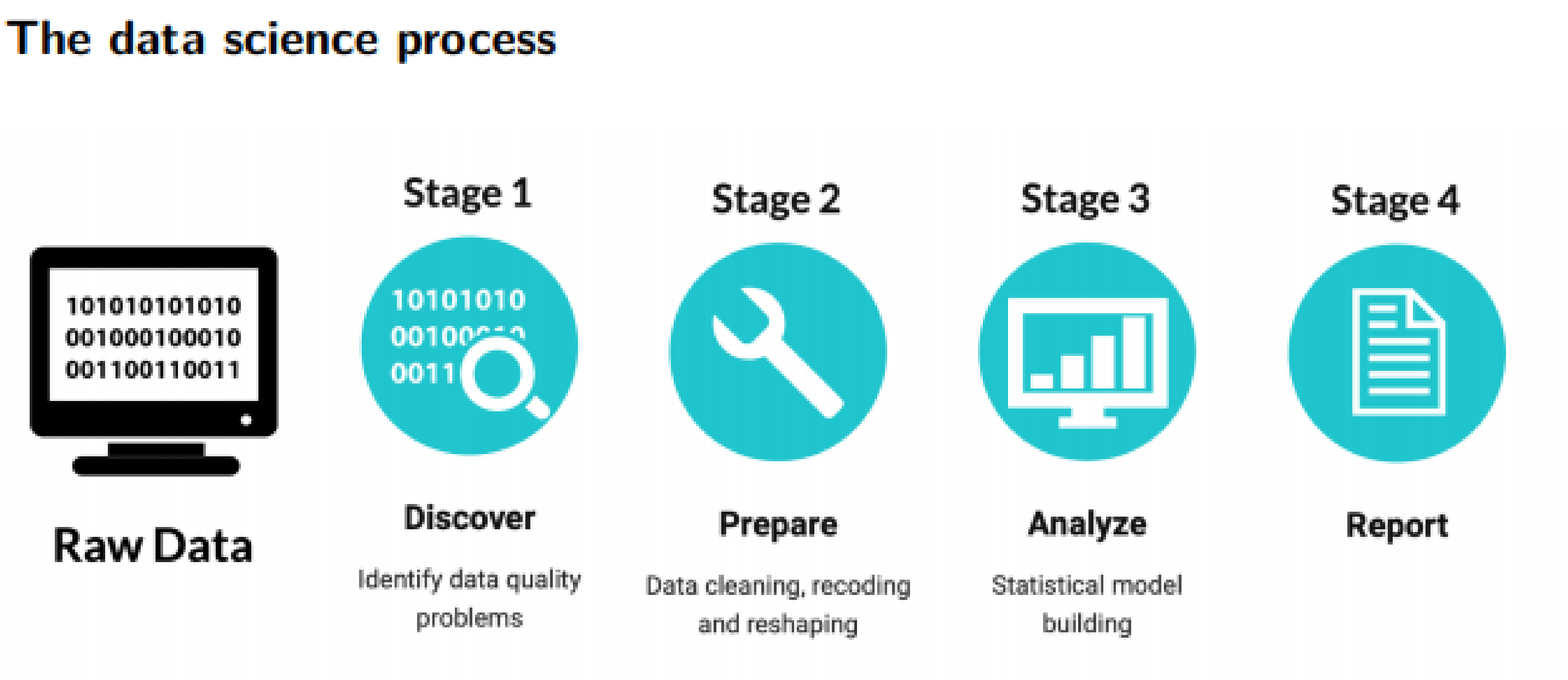
INFOX
educación sin fronteras

Agenda

4. Manipulación de datos

INFOX
educación sin fronteras

3. Manipulación de datos – Parte 1



INFOX
educación sin fronteras

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 prepartion 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?

INFOX
educación sin fronteras

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
```

INFOX
educación sin fronteras

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
```

INFOX
educación sin fronteras

3. Manipulación de datos – Parte 1

Question 3: Are there implausible/ missing values?

Check for observations with 0 or negative pruchases:

```
> 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: !=
- negotiation 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)
```

INFOX
educación sin fronteras

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
```

INFOX
educación sin fronteras

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).

INFOX
educación sin fronteras

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)

INFOX
educación sin fronteras

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).

INFOX
educación sin fronteras

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 <u>and</u> Unit.price.non.deal according package size and fat.

INFOX
educación sin fronteras

¡GRACIAS!

INFOX
educación sin fronteras