Data import and Time Series analysis

In this module, you will learn how to import ASCII and Excel files into R and the rudimentary tools of time series analysis.

Lesson Plan

Data import

- **ASCII** files (*.txt*, *.csv*)
- **Excel** files (*.xls*, *.xlsx*)

Exploratory Data Analysis

- **Data manipulation** of R objects: vector, data frame, matrix, ...

- **Sub-set** of datasets
- Use of basic **Rfunctions**

- **Data aggregation** at different time scales

Time Series

- **Construction** time-series objects
- **Plot** of Time Series

- **Decomposition** Time Series

Other Environments for R

The standard R
GUI (Graphical User Interface) editor only implements very rudimentary functionality, that's why there are several active projects of R integrated editors.

A non exhaustive list of these editors is:

Windows

1. **Tinn-R Editor** - GUI for R Language and Environment

2. **R Productivity Environment** (part of REvolution R Enterprise)

3. **RStudio**
4. **Vim**
5. **Emacs** + ESS (Emacs Speaks Statistics)

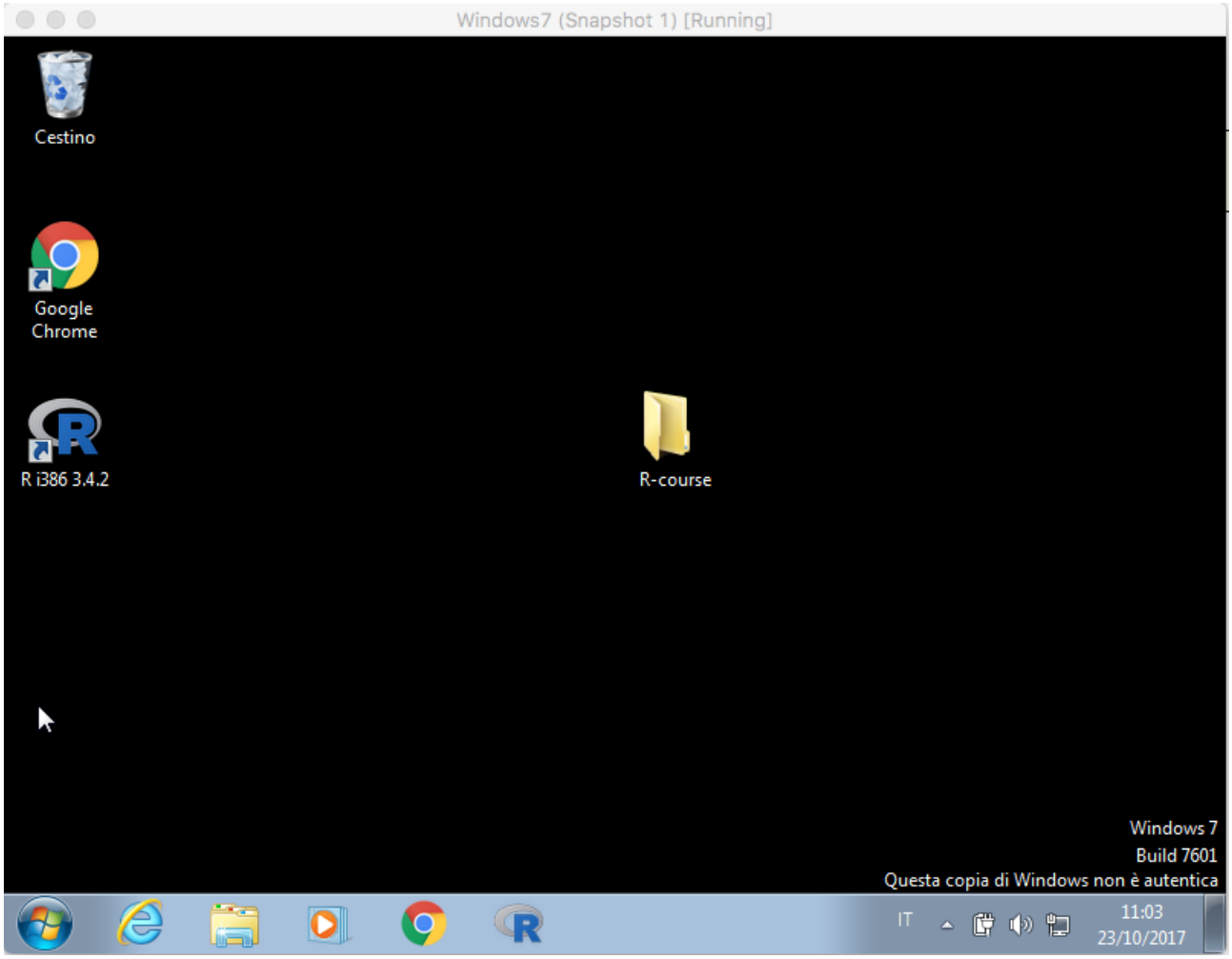
Linux

1. **RStudio**
2. **RGedit**
3. **Rkward**

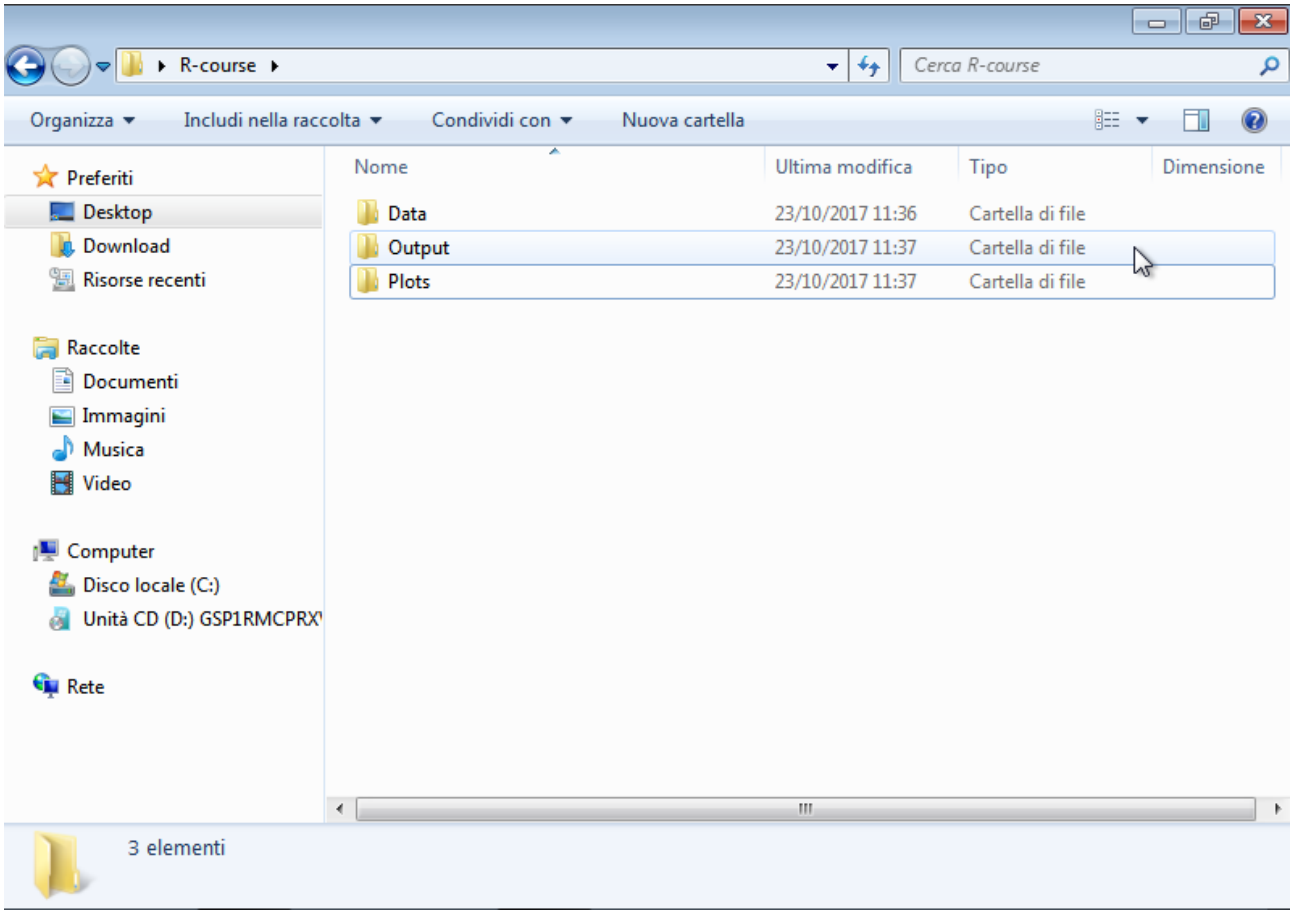
We will use the R editor for these lessons. However, if you look for an integrated environment, I suggest RStudio because the installation process is very easy and it also provides integration with many tools, such as Latex or Svn.

Create a new folder for the R course

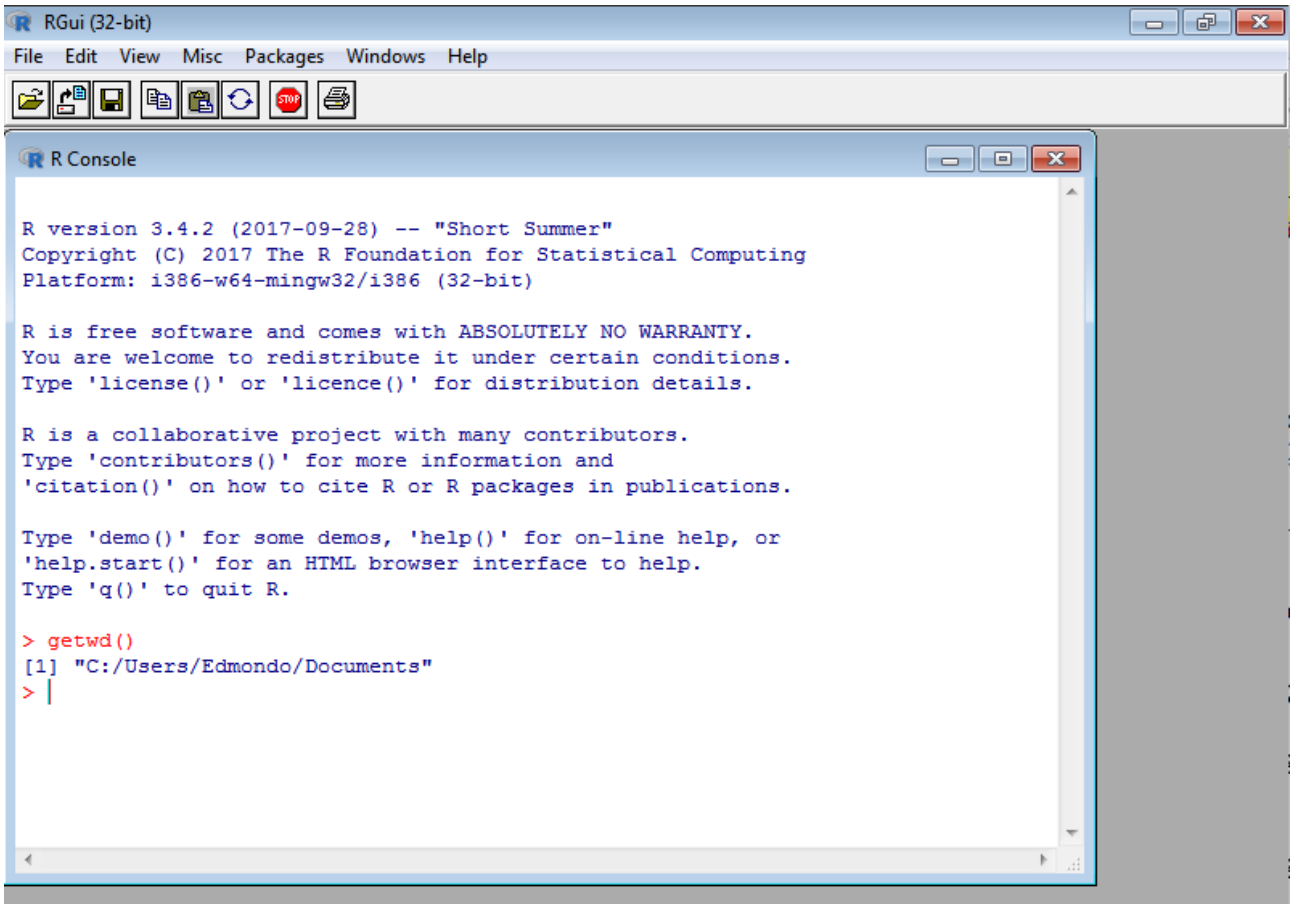
- Firstly, we create a directory R-course where we will save data to be imported and output.



- Secondly, we enter the **R-course** and create 3 further directories: Data, Output, and Plots.

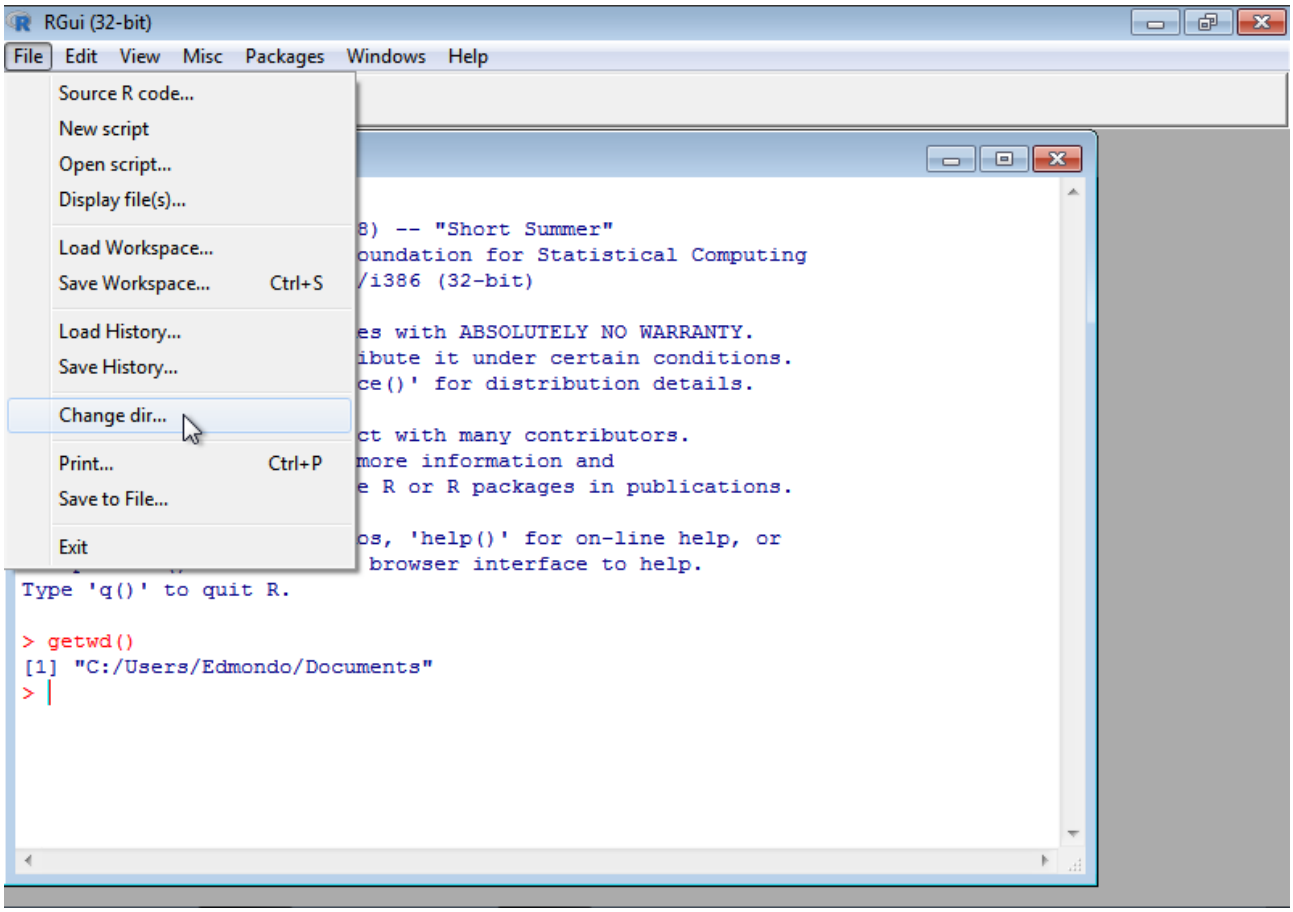


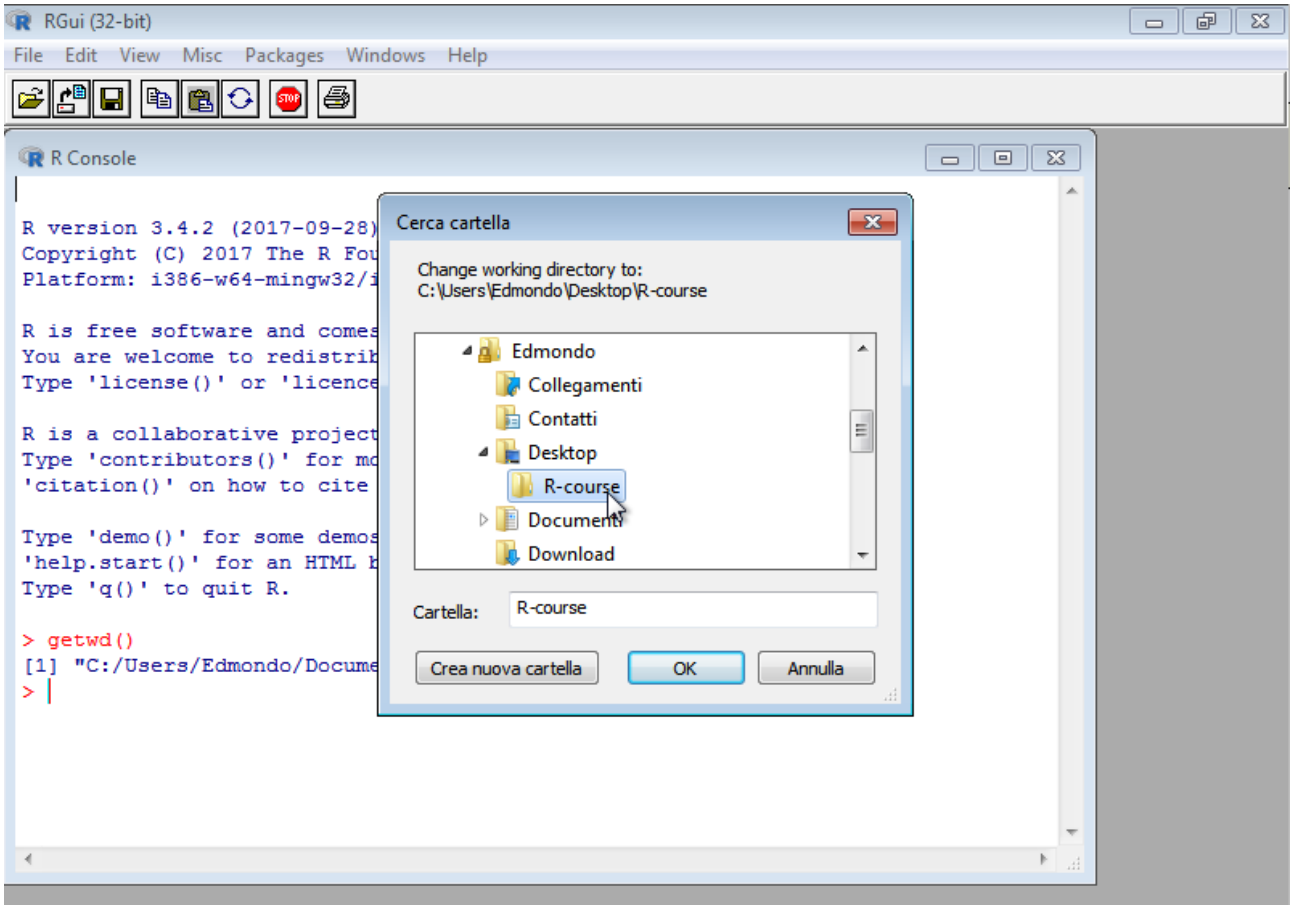
- Thirdly, we launch R Console and find out the default working directory, i.e. where the R engine expects to find files you want to import.



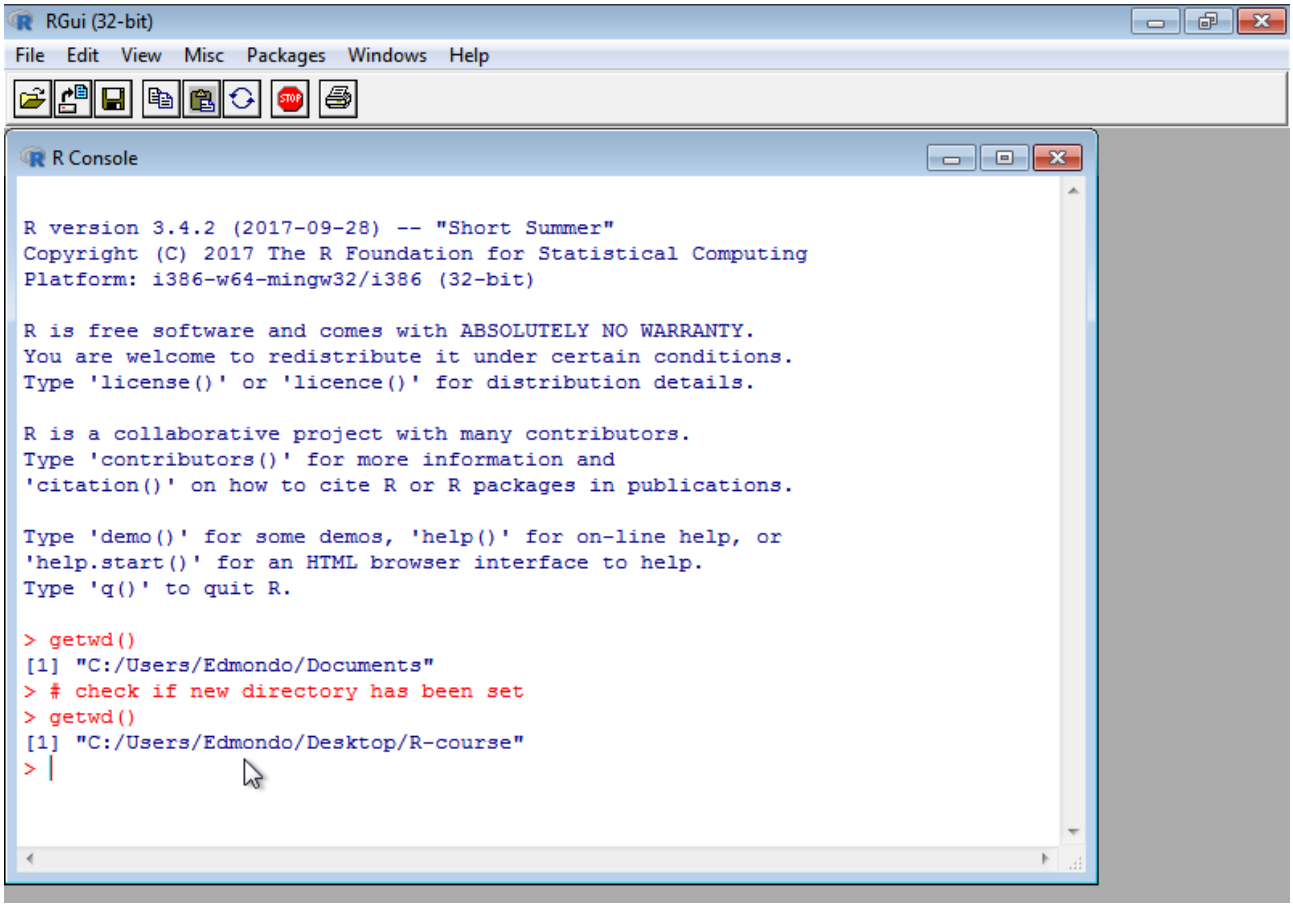
Set the working directory

- In this step, we link the R engine to the **R-course** directory.





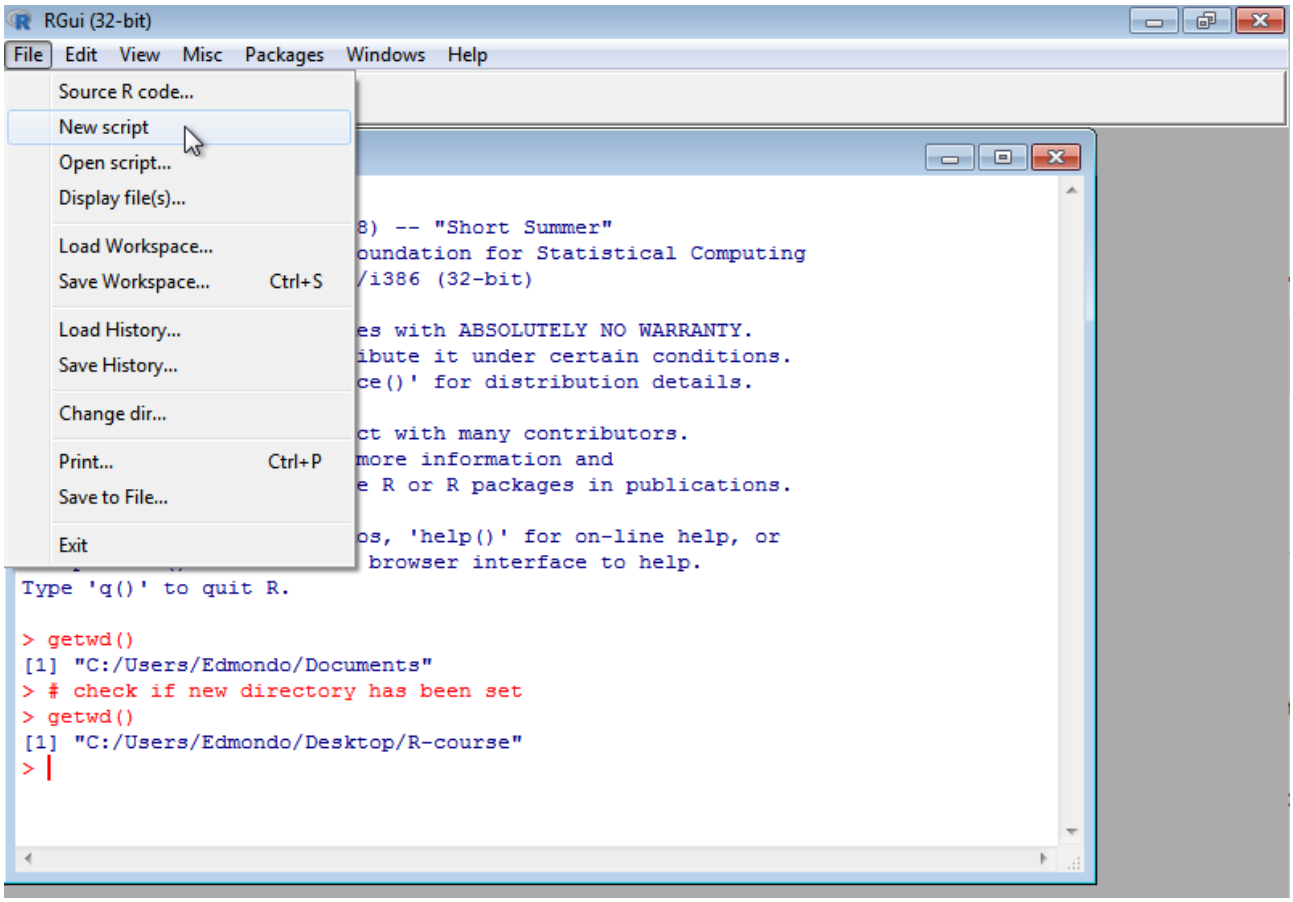
- Check if the working directory is the **R-course** folder.

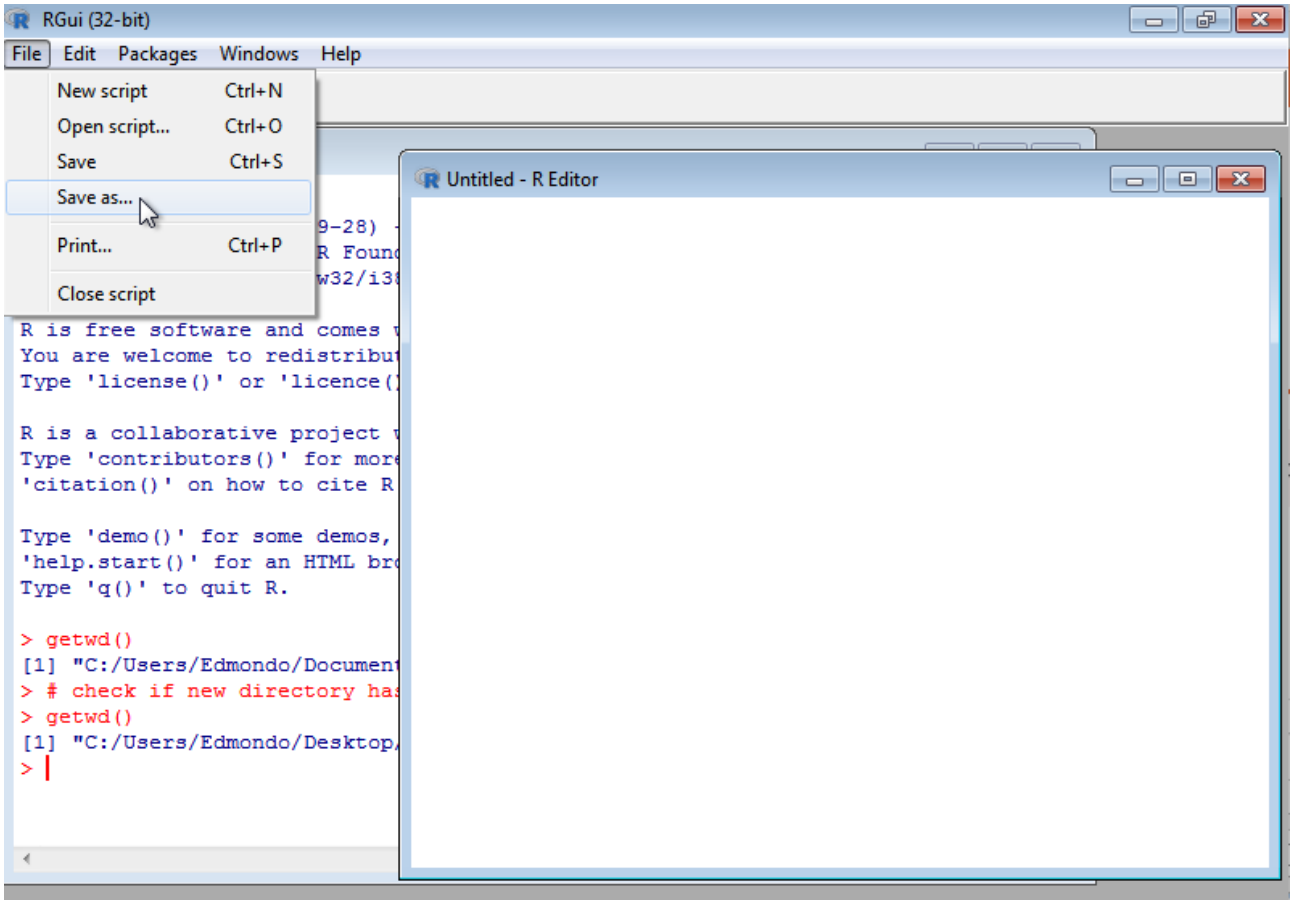


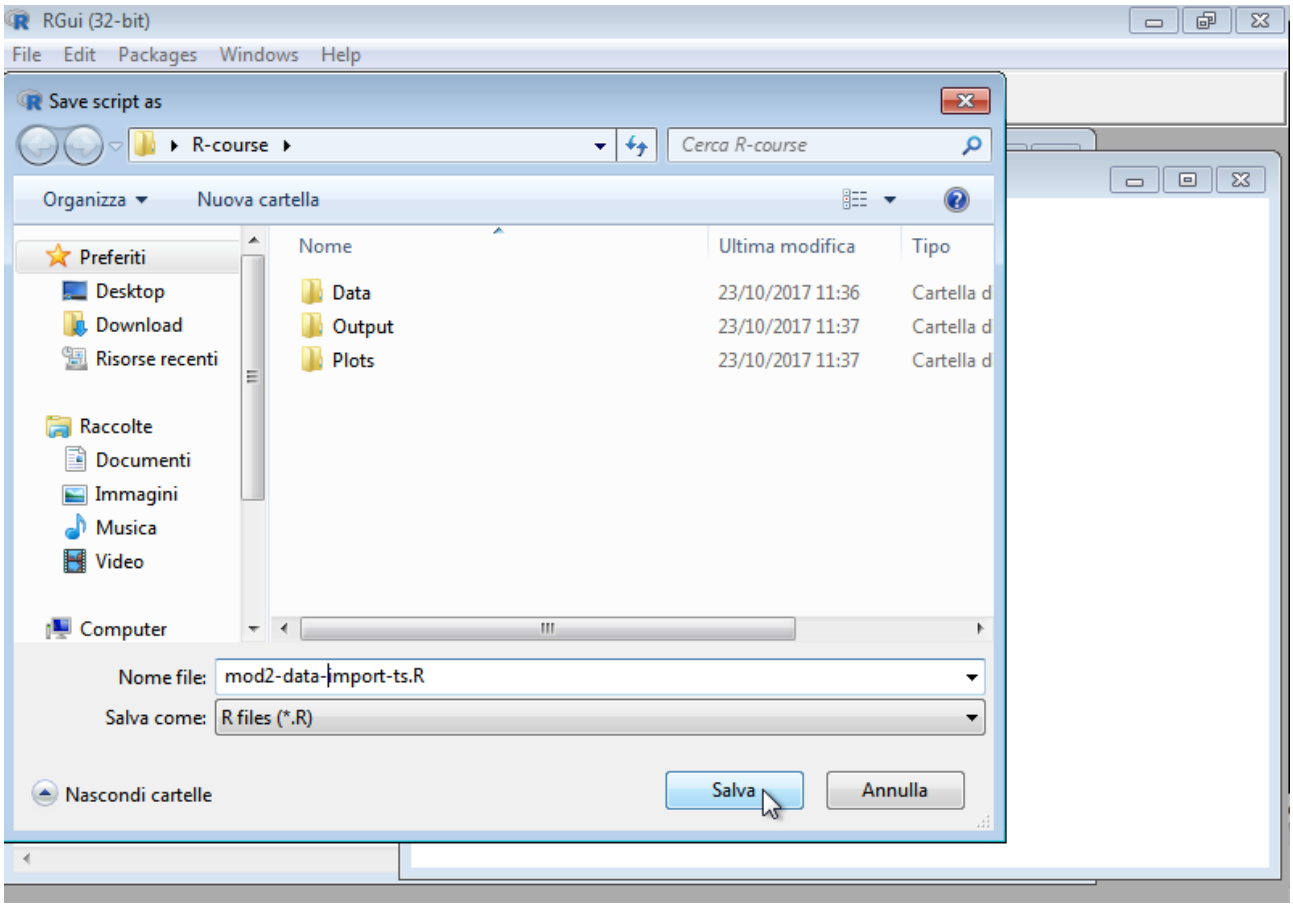
In the next content page, we will save a file into the working directory that contains an empirical time series data.

Create a script file

- From R Console, we create a file named **“mod2-data-import-ts.R”** (we are going to fill up this file with a **sequence of R commands**) doing as follows:

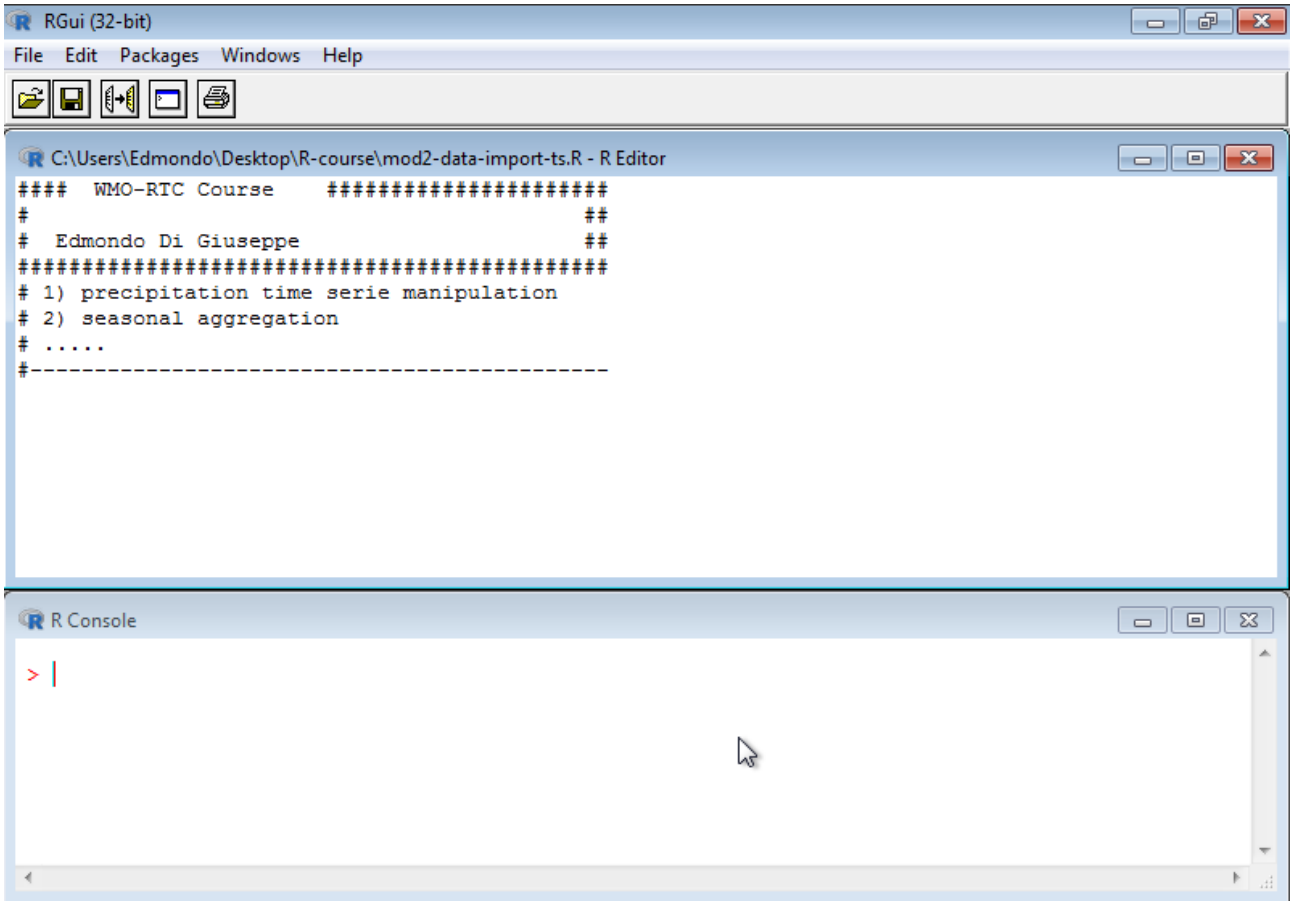



RGui (32-bit)
File Edit Packages Windows Help
New script Ctrl+N
Open script... Ctrl+O
Save Ctrl+S
Save as...
Print... Ctrl+P
Close script
Untitled - R Editor
R is free software and comes
You are welcome to redistribu
Type 'license()' or 'licence(
R is a collaborative project
Type 'contributors()' for mor
'citation()' on how to cite R
Type 'demo()' for some demos,
'help.start()' for an HTML br
Type 'q()' to quit R.
> getwd()
[1] "C:/Users/Edmondo/Document
> # check if new directory ha
> getwd()
[1] "C:/Users/Edmondo/Desktop/
>

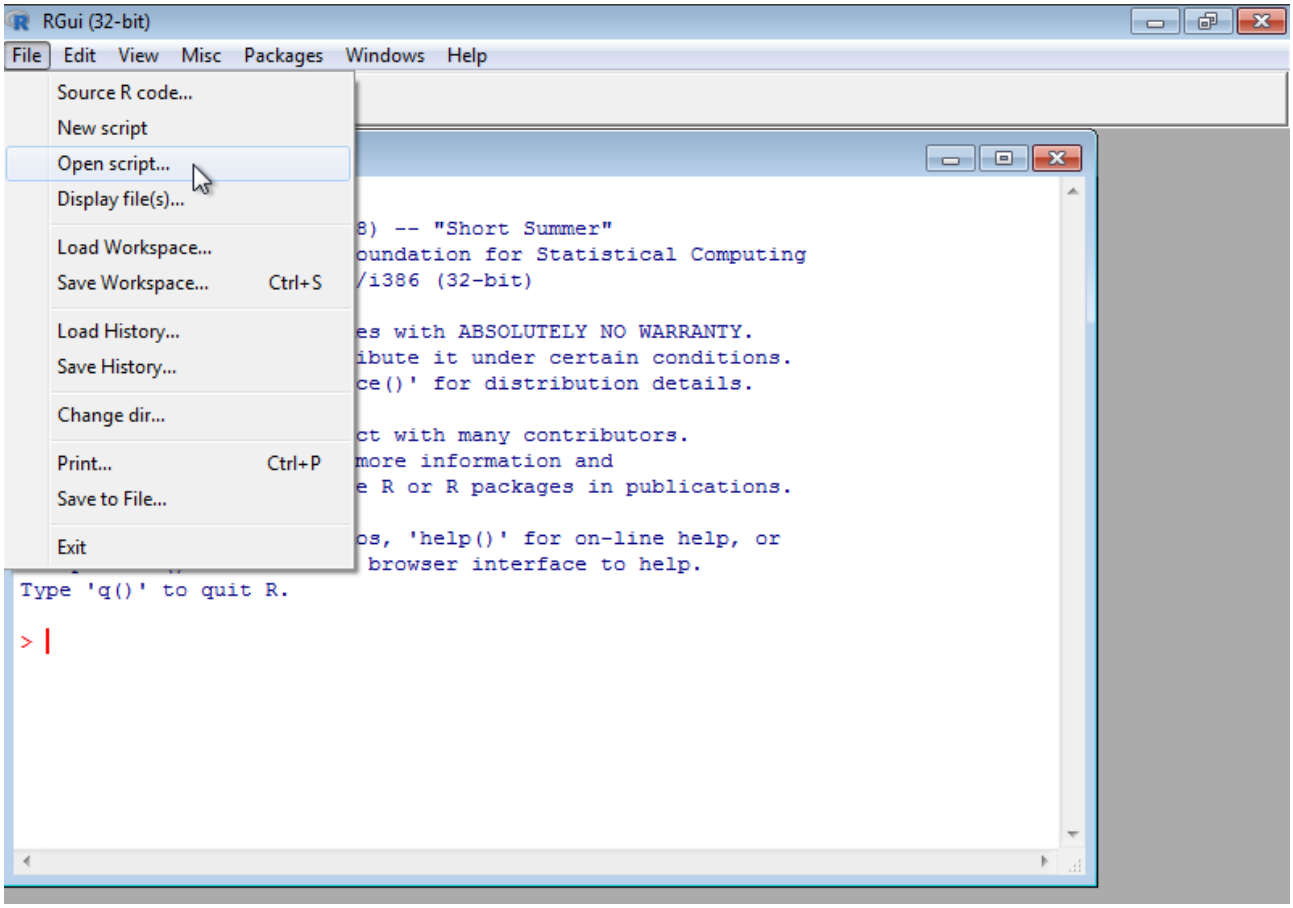

RGui (32-bit)
File Edit Packages Windows Help
Save script as
R-course
Cerca R-course
Organizza
Nuova cartella
Preferiti
Desktop
Download
Risorse recenti
Raccolte
Documenti
Immagini
Musica
Video
Computer
Nome
Ultima modifica
Tipo
Data 23/10/2017 11:36 Cartella d
Output 23/10/2017 11:37 Cartella d
Plots 23/10/2017 11:37 Cartella d
Nome file: mod2-data-import-ts.R
Salva come: R files (*.R)
Nascondi cartelle
Salva
Annulla

Writing code on a script file

- It is strictly recommended to **write down commands on a script file**and run each line or the entire code from it.



- If you close the session, you will be able to open your script file and keep working on it:


RGui (32-bit)
File Edit View Misc Packages Windows Help
Source R code...
New script
Open script...
Display file(s)...
Load Workspace...
Save Workspace... Ctrl+S
Load History...
Save History...
Change dir...
Print... Ctrl+P
Save to File...
Exit
8) -- "Short Summer"
oundation for Statistical Computing
/i386 (32-bit)
es with ABSOLUTELY NO WARRANTY.
ibute it under certain conditions.
ce()' for distribution details.
ct with many contributors.
more information and
e R or R packages in publications.
os, 'help()' for on-line help, or
browser interface to help.
Type 'q()' to quit R.
>

Data Import/1

How to import data from a *.txt* file

Have a look at the data files: they contain metadata and general information about **precipitation time series** of **Roma Ciampino weather station**, extracted from EUROPEAN CLIMATE ASSESSMENT & DATASET (ECA&D)

- RR_STAID000176.txt

- elements.txt
- metadata.txt
- sources.txt
- **stations.txt**

- Download and save the “RR_STAID000176.txt” file into "**R-course/Data**" (the directory already created on your PC).

Data Import/2

Now, we import data from **“RR_STAID000176.txt”** file into R.

We need to read the whole file but the first 19 rows in order to correctly import data.

```
EUROPEAN CLIMATE ASSESSMENT & DATASET (ECA&D), file created on 23-09-2015
THESE DATA CAN BE USED FREELY PROVIDED THAT THE FOLLOWING SOURCE IS ACKNOWLEDGED:

Klein Tank, A.M.G. and Coauthors, 2002. Daily dataset of 20th-century surface
air temperature and precipitation series for the European Climate Assessment.
Int. J. of Climatol., 22, 1441-1453.
Data and metadata available at http://www.ecad.eu

FILE FORMAT (MISSING VALUE CODE IS -9999):

01-06 SOUID: Source identifier
08-15 DATE : Date YYYYMMDD
17-21 RR   : precipitation amount in 0.1 mm
23-27 Q_RR : Quality code for RR (0='valid'; 1='suspect'; 9='missing')

This is the blended series of station ITALY, ROMA CIAMPINO (STAID: 176).
Blended and updated with sources: 100563 118382 916239
See file sources.txt and stations.txt for more info.

 SOUID,    DATE,   RR, Q_RR
100563,19510101,    4,    0
100563,19510102,   84,    0
100563,19510103,   92,    0
100563,19510104,   84,    0
100563,19510105,    0,    0
100563,19510106,    0,    0
100563,19510107,    0,    0
100563,19510108,    1,    0
100563,19510109,    1,    0
100563,19510110,    0,    0
100563,19510111,    0,    0
100563,19510112,  400,    0
100563,19510113,   82,    0
100563,19510114,    0,    0
100563,19510115,    0,    0
100563,19510116,   36,    0
100563,19510117,    0,    0
100563,19510118,    1,    0
100563,19510119,   16,    0
100563,19510120,    0,    0
100563,19510121,    0,    0
100563,19510122,    0,    0
100563,19510123,    1,    0
100563,19510124,    1,    0
100563,19510125,    2,    0
100563,19510126,    3,    0
100563,19510127,    8,    0
100563,19510128,   50,    0
100563,19510129,    4,    0
100563,19510130,    1,    0
100563,19510131,    0,    0
100563,19510201,    1,    0
```

Notice that the **precipitation amount** in the **column RR** is expressed in **0.1 mm**.

Data Import/3

Finally, copy and paste the following lines into the **mod2-data-import-ts.R** file:

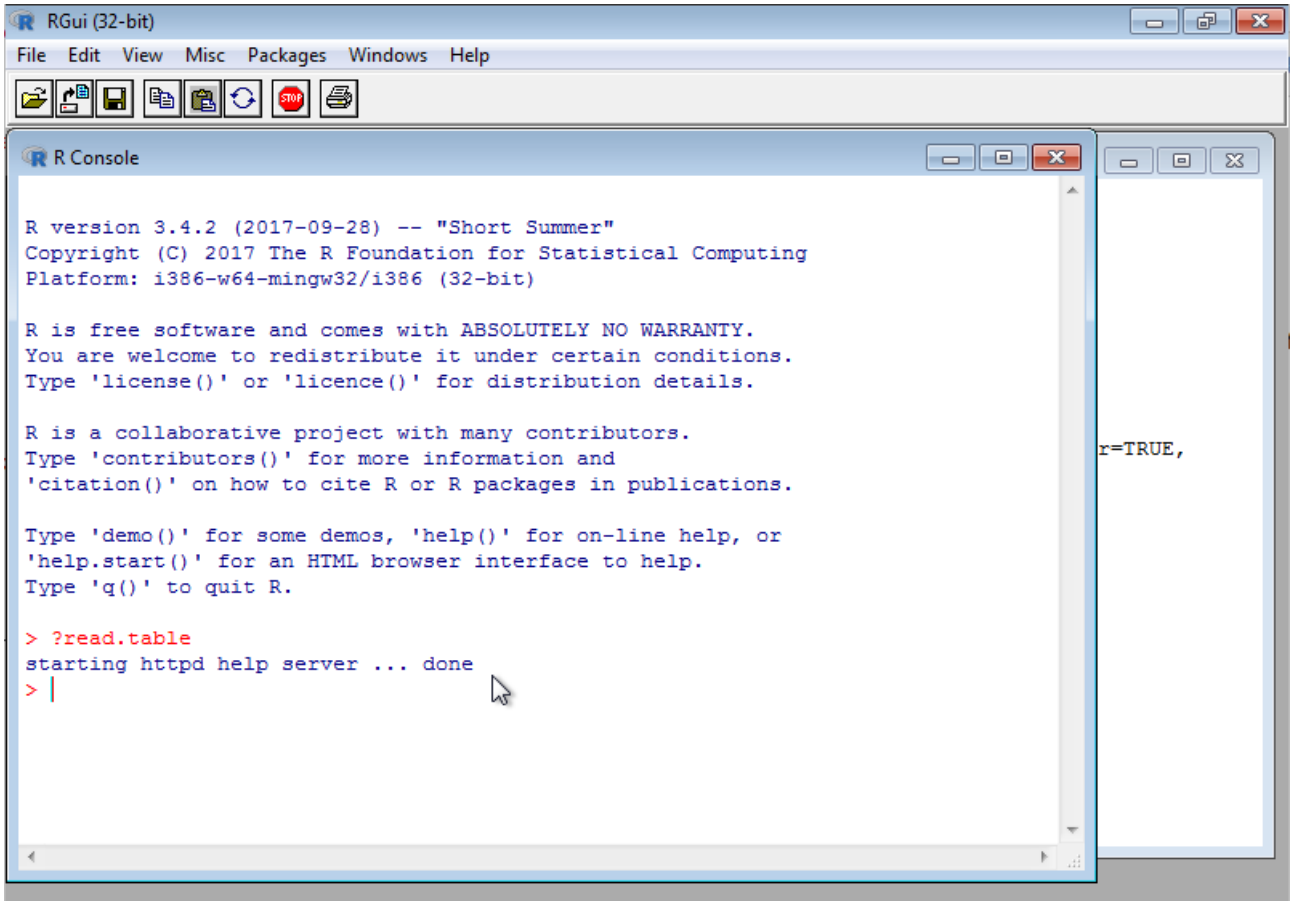
```
dataRR<-read.table(file="Data/RR_STAID000176.txt", skip=19, header=TRUE, sep=",", na.strings=-9999)

dataRR$RR<-dataRR$RR*0.1
```

Function arguments

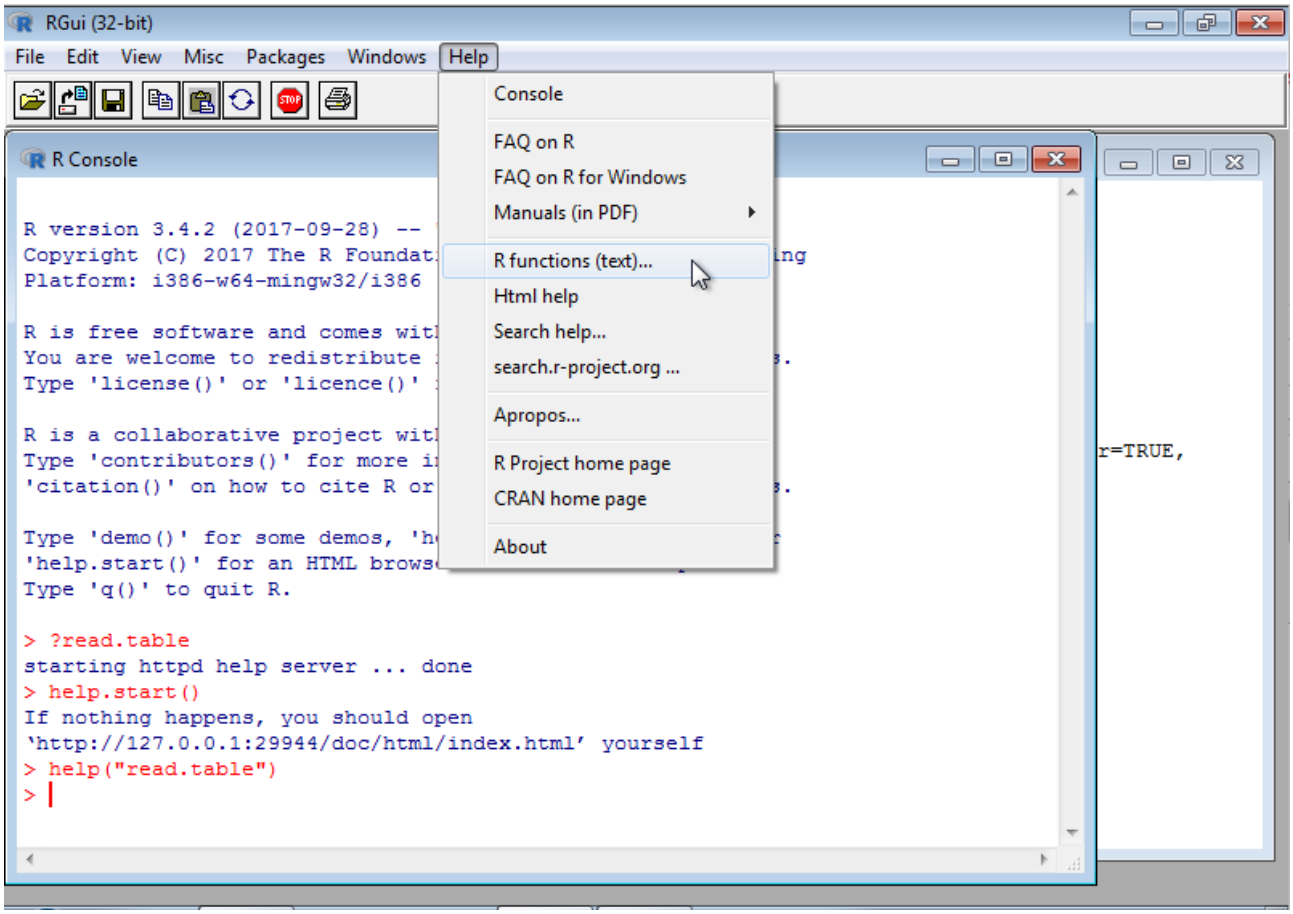
Here, we need to define some arguments in order to correctly read data:

- **file**: the path to the file (TIP: you might avoid specifying the working directory part of the path);
- **skip**: the 19 lines to be skipped from the reading of the file;
- **header**: a logical value indicating whether the file contains the names of the variables as its first line;
- **sep**: the field separator character;
- **na.strings**: a character vector of strings which are to be interpreted as NA values.

You can directly call **help** about arguments definition from the console (press Return to run the following command):

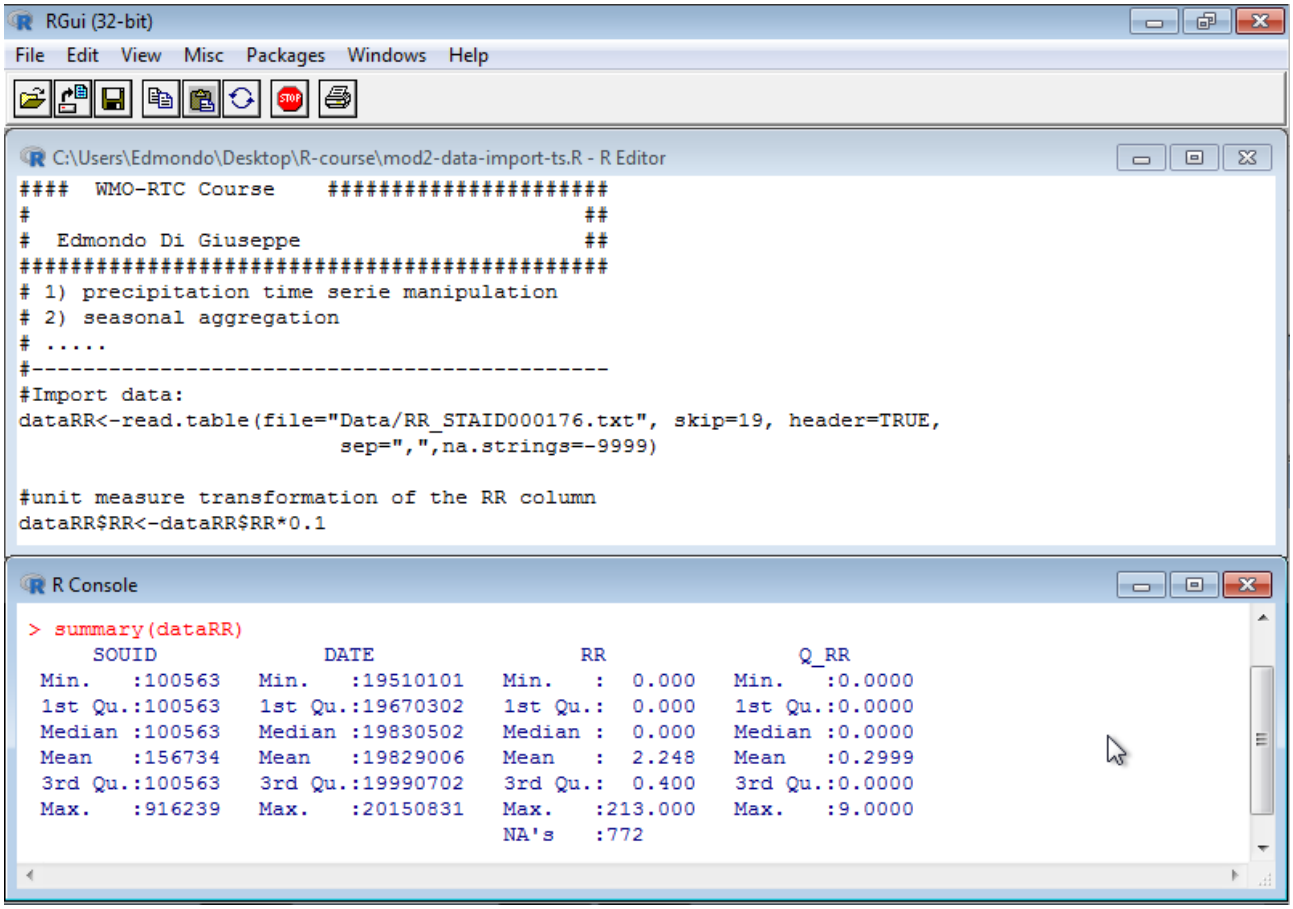


Alternatively, you can type "read.table" into the proper **help box** and press return:

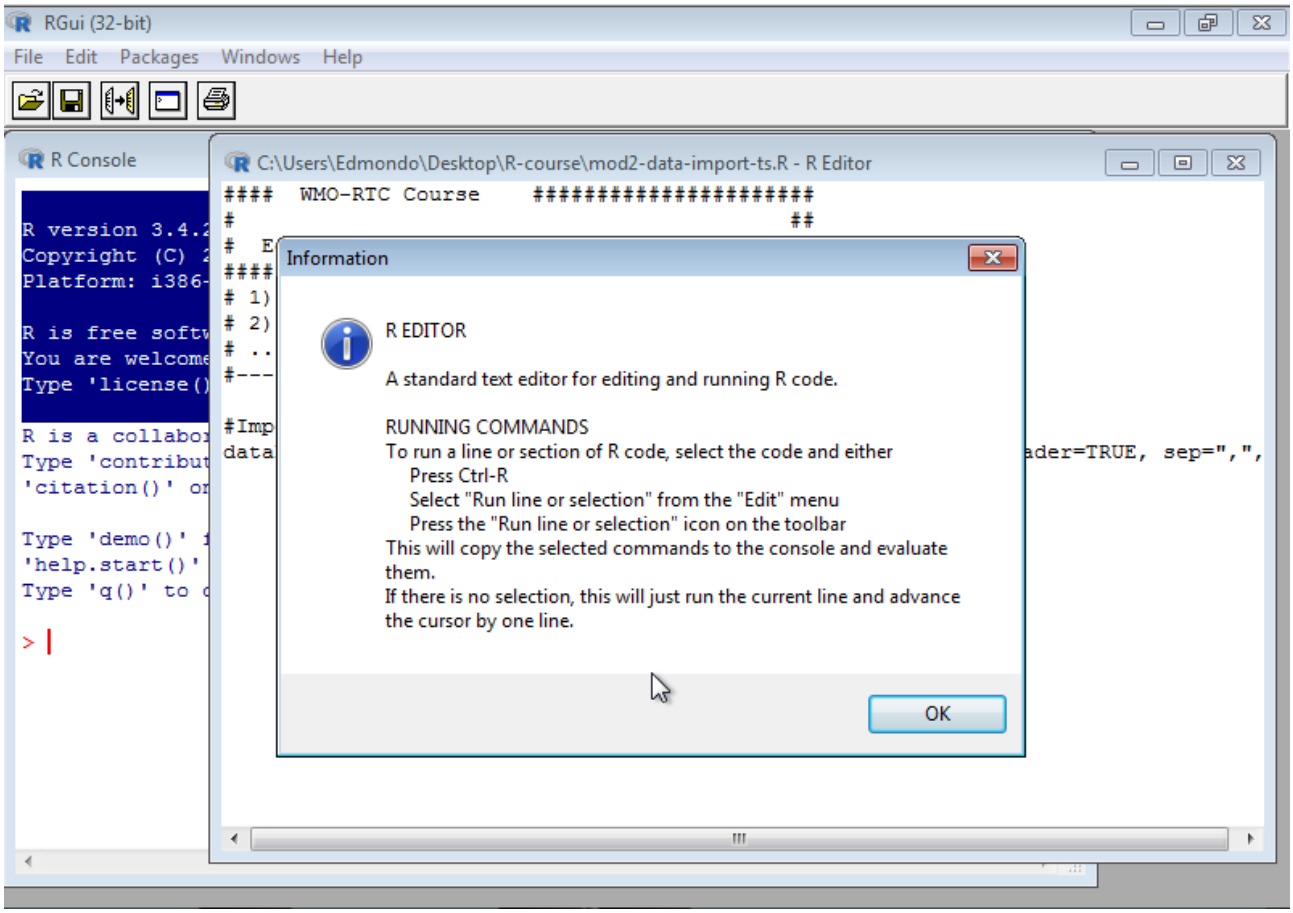


Writing commands on R script file

- you have a **clear list** of what is needed for your analysis;
    - It is also recommended to **describe** every step of your code into the R script file using the **comment character** #



- How to execute commands from the R editor:

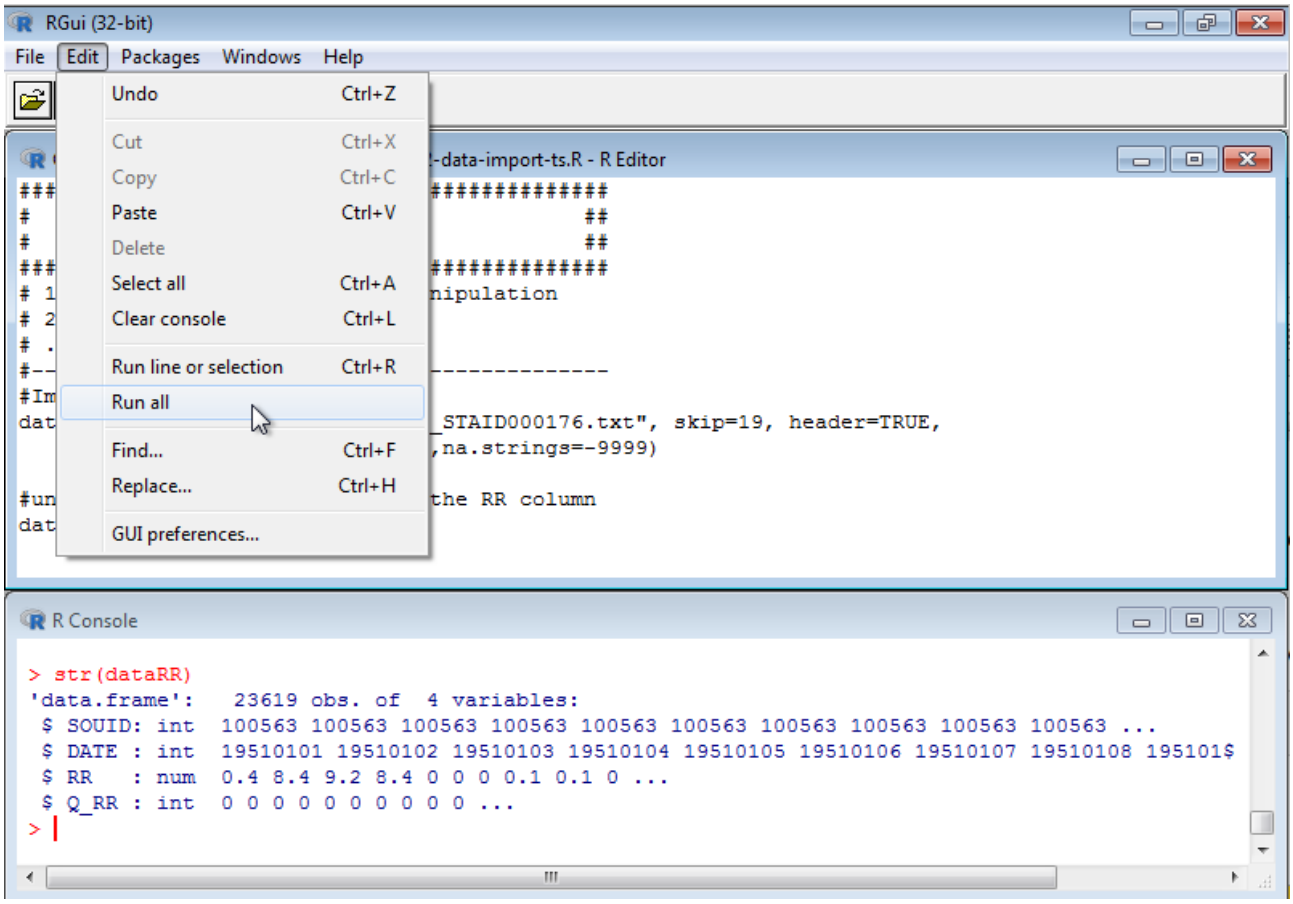


- you can execute the whole commands on the script file in two ways

1. by using the function **source()**:

```
source("mod2-data-import-ts.R")
```

2. by GUI interface:


RGui (32-bit)
File
Edit
Packages
Windows
Help
Undo Ctrl+Z
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete
Select all Ctrl+A
Clear console Ctrl+L
Run line or selection Ctrl+R
Run all
Find... Ctrl+F
Replace... Ctrl+H
GUI preferences...
-data-import-ts.R - R Editor
R Console
> str(dataRR)
'data.frame': 23619 obs. of 4 variables:
$ SOUID: int 100563 100563 100563 100563 100563 100563 100563 100563 100563 100563 ...
$ DATE : int 19510101 19510102 19510103 19510104 19510105 19510106 19510107 19510108 195101$
$ RR : num 0.4 8.4 9.2 8.4 0 0 0 0.1 0.1 0 ...
$ Q_RR : int 0 0 0 0 0 0 0 0 0 0 ...

Data Import - Question 1

Type **str(dataRR)** in the console.

What kind of object **dataRR** is?

vector

matrix

data frame

Data Import/file types

R software can manage to import data from the most used **file types**.

TXT Files

A data table resides in a text file and the cells inside the table are **separated byblank characters**. The function to be called is:

```
read.table()
```

CSV Files - Comma separated values format

Each cell inside such data file is **separated by a special character**, for instance, a comma. The first row of the data file should contain the column names instead of the actual data. The function to be called is:

```
read.csv()
```

Fixed-width files

To read a fixed-width format text file into a data frame, you can use the **read.fwf()** function:

```
datax<-read.fwf(file, widths, header = , sep = , skip = , row.names, col.
names, n = ,buffersize = , ...)
```

Data Import/Excel files

Clipboard

The **easiest way** to import data from Excel is as follows:

1. open Excel sheet, select the entire data table (or part of it) and copy on **clipboard** (CTRL+C);

2. open R console and type

```
datax<-read.table(file= "clipboard", sep = "\t", dec = ",")
```

Notice that you need to specify some arguments of the **read.table()** function accordingly with the characteristics of the file to be imported:

- **sep** is the field separator character;
- **dec** the character used in the file for decimal points;
- **header** a logical value indicating whether the file contains the names of the variables as its first line;
- ....

Read data directly from file

Obviously, the "clipboard" method is inefficient when you need to import multiple files. However, there are several ways to read Excel files:

- **read.xls()** in **{gdata}** package (read/write Excel 97-2004 and 2007+ file format);
- **read.xlsx()** in **{xlsx}** package (read/write Excel 97/2000/XP/2003/2007 file format);
- .....

Connection to file

The **{RODBC}** package provides tools to connect with Microsoft Access database and Excel, as well. Notice that ODBC drivers are required to use this option.

This topic will be part of the face to face course, so I recommend to install the two packages **{gdata}** and **{xlsx}**.

Exploration Data Analysis/1

Copy and paste the following commands into the file "mod2-data-import-ts.R", then execute them from there.

We have already applied the **unit measure transformation** exclusively of the **“RR column”** of data frame:

```
dataRR$RR<-dataRR$RR*0.1
```

Let us **create some new columns** concernings the “date” format:

```
dataRR$YEAR<-as.numeric(substring(dataRR$DATE, 1,4))
```

where we have chosen "YEAR" as a name for this new column and filled it with the first four characters of column "DATE" (notice that we have done a transformation from character to numeric)

```
dataRR$MONTH<-as.numeric(substring(dataRR$DATE, 5,6))
dataRR$DAY<-as.numeric(substring(dataRR$DATE, 7,8))
dataRR$MONTHABB<-month.abb[dataRR$MONTH]
dataRR$YEARMONTH<-paste(dataRR$YEAR, dataRR$MONTHABB, sep="")
dataRR$DATE<-as.Date(as.character(dataRR$DATE), "%Y%m%d")
```

In the previous commands we have used some functions that you haven't encountered yet:

- **as.numeric()**
- **substring()**
- **paste()**
- **as.Date()**

and pre-existing object

month.abb[ ].

Type these functions in the help box and learn their usage.

Exploration Data Analysis/2

We now introduce a function to calculate the monthly cumulated precipitation:

`tapply()`

`help(tapply)``to learning its use`

and put the output of the function into a new object that we name **mon.cum**:

```
mon.cum<-tapply(X=dataRR$RR, INDEX=list(dataRR$YEAR, dataRR$MONTH),   FUN
=sum, na.rm=TRUE)
```

Notice that we need to specify some arguments of the **tapply()** function:

- **INDEX:** the columns to which group the main variable **dataRR$RR** on;
- **FUN:** the function to be applied to each group.

Exploratory Data Analysis - Exercise 1

You have just calculated monthly cumulated precipitation using the line:

```
mon.cum<-tapply(X=dataRR$RR, INDEX=list(dataRR$YEAR, dataRR$MONTH), FUN=sum, na.rm=TRUE)
```

Now, change the **na.rm** argument to **"FALSE"** and count the **NA** (missing data) in the dataset. When you set the **na.rm=FALSE** in the configuration of **tapply()** given above, i.e. a monthly aggregation of daily values, the output will result to NA if **one or more daily values are missing**.

The counting of monthly aggregated missing data is:

10

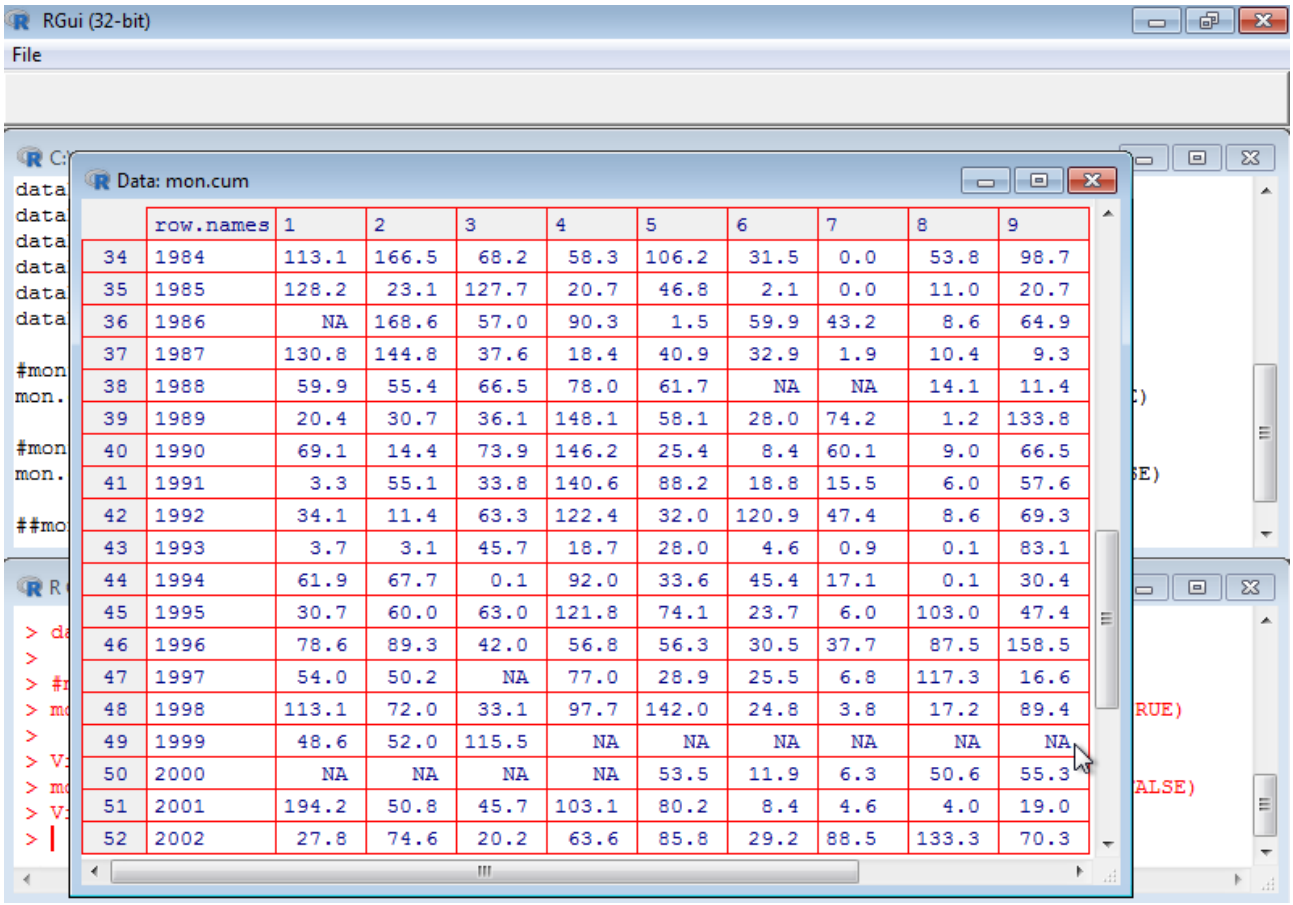
none

63

Exploration Data Analysis/3

Visualize the **mon.cum** object using the function:

```
View(mon.cum)
```



	row.names	1	2	3	4	5	6	7	8	9
34	1984	113.1	166.5	68.2	58.3	106.2	31.5	0.0	53.8	98.7
35	1985	128.2	23.1	127.7	20.7	46.8	2.1	0.0	11.0	20.7
36	1986	NA	168.6	57.0	90.3	1.5	59.9	43.2	8.6	64.9
37	1987	130.8	144.8	37.6	18.4	40.9	32.9	1.9	10.4	9.3
38	1988	59.9	55.4	66.5	78.0	61.7	NA	NA	14.1	11.4
39	1989	20.4	30.7	36.1	148.1	58.1	28.0	74.2	1.2	133.8
40	1990	69.1	14.4	73.9	146.2	25.4	8.4	60.1	9.0	66.5
41	1991	3.3	55.1	33.8	140.6	88.2	18.8	15.5	6.0	57.6
42	1992	34.1	11.4	63.3	122.4	32.0	120.9	47.4	8.6	69.3
43	1993	3.7	3.1	45.7	18.7	28.0	4.6	0.9	0.1	83.1
44	1994	61.9	67.7	0.1	92.0	33.6	45.4	17.1	0.1	30.4
45	1995	30.7	60.0	63.0	121.8	74.1	23.7	6.0	103.0	47.4
46	1996	78.6	89.3	42.0	56.8	56.3	30.5	37.7	87.5	158.5
47	1997	54.0	50.2	NA	77.0	28.9	25.5	6.8	117.3	16.6
48	1998	113.1	72.0	33.1	97.7	142.0	24.8	3.8	17.2	89.4
49	1999	48.6	52.0	115.5	NA	NA	NA	NA	NA	NA
50	2000	NA	NA	NA	NA	53.5	11.9	6.3	50.6	55.3
51	2001	194.2	50.8	45.7	103.1	80.2	8.4	4.6	4.0	19.0
52	2002	27.8	74.6	20.2	63.6	85.8	29.2	88.5	133.3	70.3

The **mon.cum** object has missing data since we set **na.rm=FALSE** to solve the exercise. Then, re-execute the previous command setting **na.rm=TRUE** before proceeding.

Exporation Data Analysis/4

How to compute **CLIMATOLOGY of monthly cumulated precipitation** in a **specified period**?

For instance, we can average over the entire period using the **apply()** or the **colMeans()** function (notice that **mon.cum** object is actually a matrix, **str(mon.cum)** to prove it). We can propose the following two options:

```
1)
mon.cum.ave
1<-apply(X=mon.cum, MARGIN=2, FUN=mean, na.rm=TRUE)    # first option

2)
mon.cum.ave2<-colMeans(x=mo
n.cum, na.rm=TRUE)                          # second option
```

It can also be useful to round decimal digits:

```
mon.cum.ave<-round(mon.cum.ave1, digits=1)
```

Exploratory Data Analysis - Exercise 2

Exploratory Data Analysis - Exercise 2

More interestingly, we can average over the period 1971-2000 sub-setting the largest matrix of monthly data.

Firstly, notice that **mon.cum** object has **2 dimnames** attributes and the first one is for row.

```
str(mon.cum)
 num [1:65, 1:12] 87.1 56.4 58.1 60.1 26.7 37.9 87.7 87.5 42.2 77.4 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:65] "1951" "1952" "1953" "1954" ...
  ..$ : chr [1:12] "1" "2" "3" "4" ...
```

Then, we can build an index based on years to select the period 1971-2000 from**mon.cum:**

```
Years.Index<-as.character(c(1971:2000))
Years.Index
 [1] "1971" "1972" "1973" "1974" "1975" "1976" "1977" "1978" "1979" "1980
" "1981" "1982" "1983"
[14] "1984" "1985" "1986" "1987" "1988" "1989" "1990" "1991" "1992" "1993
" "1994" "1995" "1996"
[27] "1997" "1998" "1999" "2000"
```

and use it to sub-set **mon.cum** as follows:

```
mon.cum.7100<-mon.cum[Years.Index,]
```

Your turn now: compute 1971-2000 climatology and put the result into a new object named, say **mon.cum.ave.7100**.

Which is the value of October's cumulated precipitation in mm?

123.9

125.0

165.6

Exploration Data Analysis/5

Seasonal Aggregation

Now, we'll se how to **aggregate** data according to **DJF, MAM, JJA and SON seasons**.

As usual in programming, there are several ways to do the same thing: you'll find your way, till then follow this one:

```
library(seas)     # load the seas library
> Error in library(seas): there is no package called 'seas'
```

SUGGESTION: always read error messages carefully, you'll find out how to get the solution…….

```
install.packages("seas")
```

Since the function **mkseas()** of **{seas}** package **requires a specific name** for the column that contains the date, we need to rename the column **dataRR$DATE** as follows:

```
names(dataRR)[2]<-"date"
```

Then, add a new column named, say "SEAS" to **dataRR** data frame:

```
dataRR$SEAS<-mkseas(x=dataRR, width="DJF")
```

and another named "SEASYEAR"

```
dataRR$SEASYEAR<-paste(dataRR$YEAR, dataRR$SEAS, sep="")
str(dataRR)
```

Exploration Data Analysis - Exercise 3

Exploration Data Analysis- Exercise 3

Compute 1971-2000 seasonal climatology.

Which is the value of Spring's cumulated precipitation (MAM)?

193.0

327.4

65.0

Time Series/1

Now we show you how to use the R software to carry out some simple analyses that are common in analyzing time series data.

Building a time series

The ts() function will convert a numeric vector into a R time series object.

The format is:

ts(*vector*, start=, end=, frequency=)

where **start**and**end** are the times of the first and last observation and **frequency** is the number of observations per unit time

(1=annual, 4=quartly, 12=monthly, 365.25=daily).

Firstly, we need to transform the **mon.cum** object from **matrix** to **vector** form, where t() is the transposing function:

```
mon.cum.V<-as.vector(t(mon.cum))
```

then to build a time series object

```
mon.cum.ts<-ts(mon.cum.V, start=c(1951,1), frequency=12)
```

Time Series - Question 2

What kind of objects **mon.cum.V**and **mon.cum.ts** are?

vector and ts object

vector and matrix

vector and data frame

Time Series/2

Plotting and sub-setting a time series

Plotting time series object:

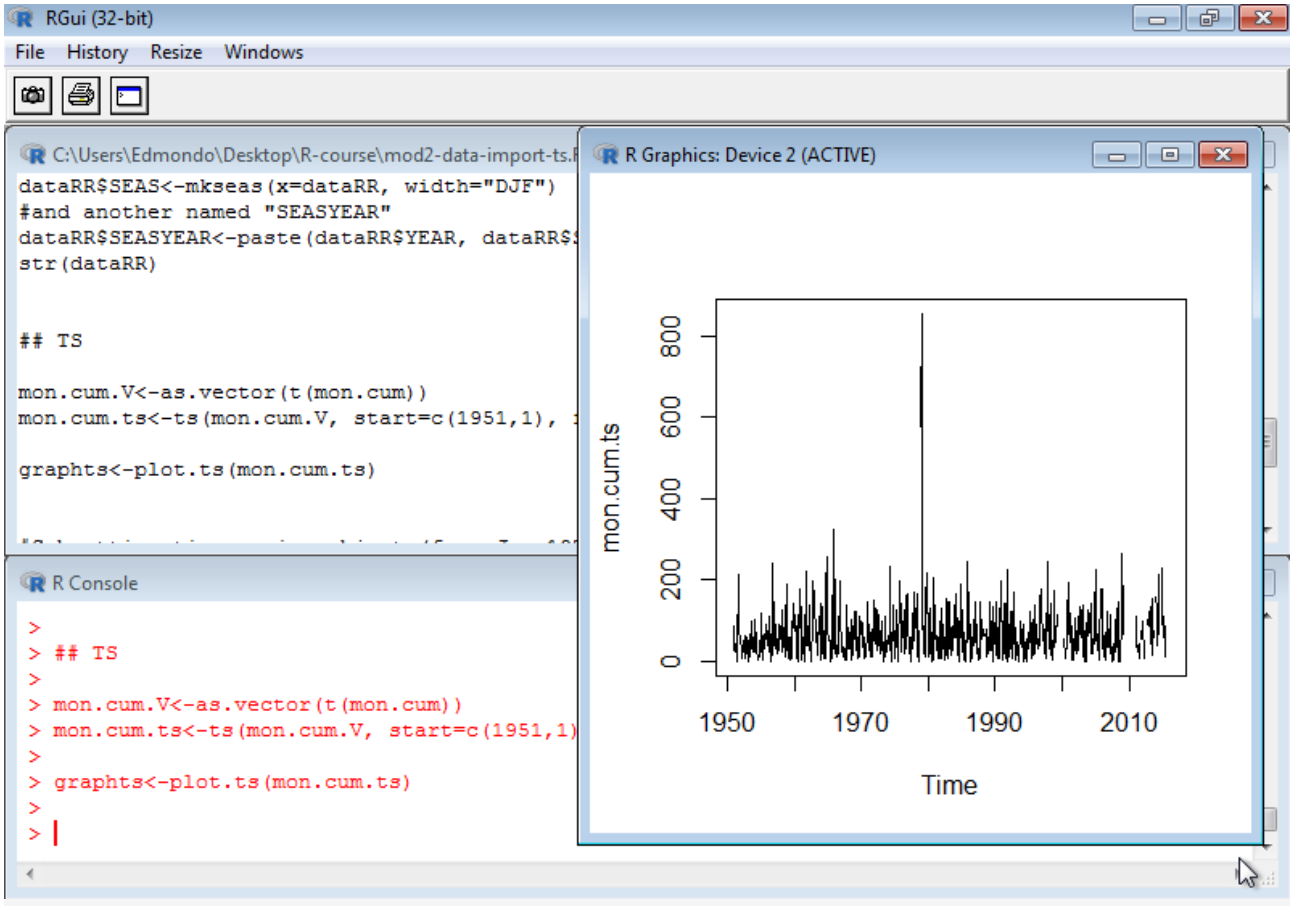
```
graphts<-plot.ts(mon.cum.ts)
```

Subsetting time series object (from Jan 1971 to Dec 2000):

```
mon.cum.ts7100<-window(mon.cum.ts, start=c(1971,1), end=c(2000,12))
```

Plotting the time series subset:

```
graphts7100<-plot.ts(mon.cum.ts7100, ylab="mm", main="Monthly Precipitation 1971-2000")
```

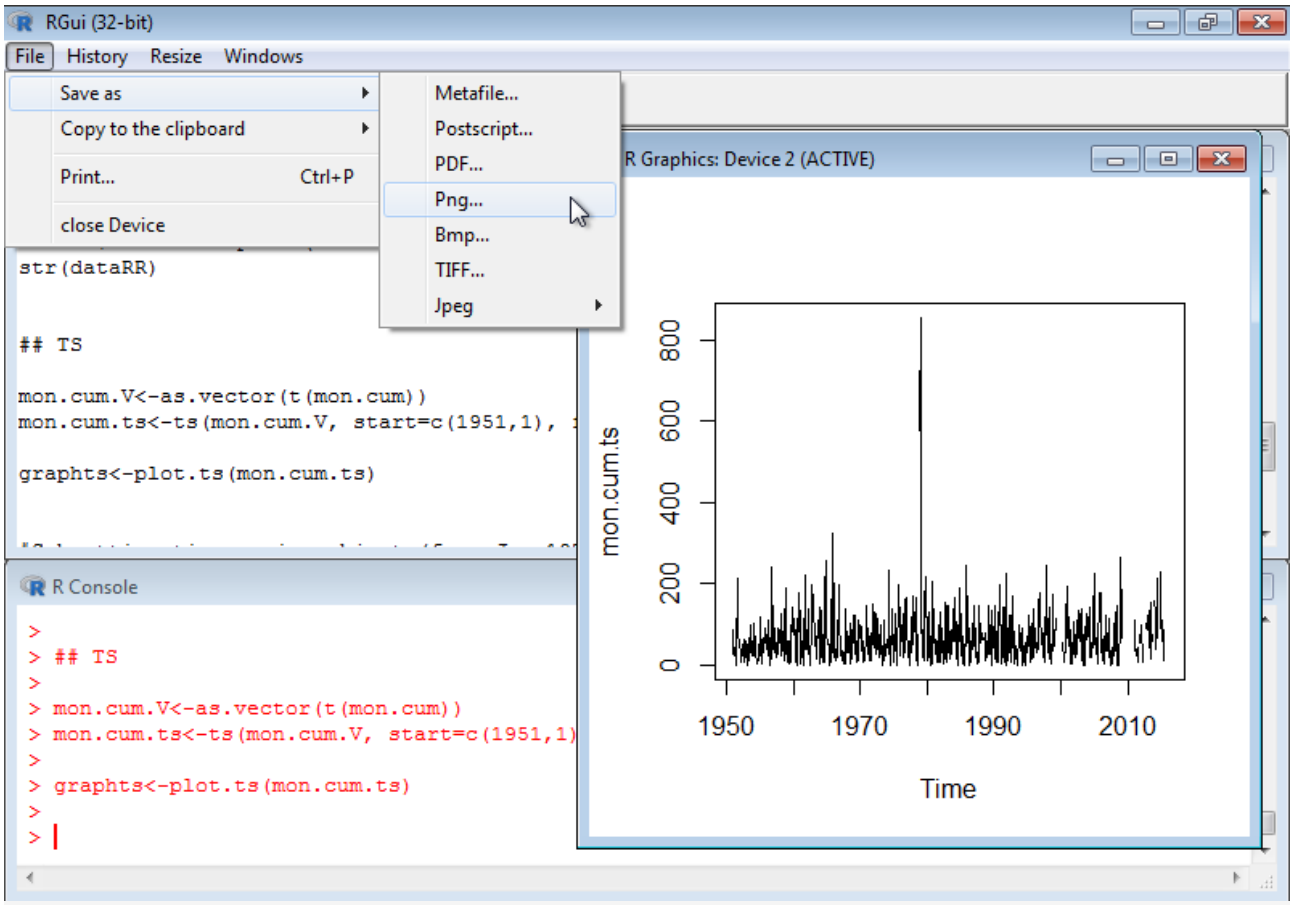


Time Series/3

Saving a Plot

(2 options)

1. from the RGUI **"File-Save as"** panel



2. from **Console** by typing (useful if you need to plot multiple graphs):

```
png(file="Plots/time-series-RR.png")
graph<-plot.ts(mon.cum.ts7100)
dev.off()
```

Time Series - Exercise 4

Time Series - Exercise 4

Finding Outliers

Find the dates of the 5 outliers (say RR>500 mm) by visualizing the time series graph.

Oct 1978, Nov 1978, Dec 1978, Jan 1979, Feb 1979

Dec 1978, Jan 1979, Feb 1979, Mar 1979, Apr 1979

Dec 1979, Jan 1980, Feb 1980, Mar 1980, Apr 1980

Time Series/4

Now, we'll replace the **5 outliers** with the correspondent **monthly value of 1971-2000 climatology.**

Firstly, we need to create an alias of **mon.cum.ts7100**:

```
mon.cum.ts7100.imp <- mon.cum.ts7100
```

Outlier dates are **from Oct 1978 to Feb 1979**, we **select** and **substitute** values:

```
window(mon
.cum.ts7100.imp, sta
rt=c(1978,10), end=c(1979,2)) <-
c(mon.cum.ave.7100[10:12], mon.cum.ave.7100[1:2])
```

mon.cum.ts7100.imp is now a new time series object **without outliers**!

SUGGESTION: you have already built the **mon.cum.ave.7100** object in a previous exercise.

Time Series/5

Seasonal Decomposition: an example of analysis done working on **ts objects.**

A time series with additive trend, seasonal, and irregular components can be decomposed. Three way to do that, among others:

1.by loess smoothing

```
decomposition<-stl(mon.cum.ts7100.imp, s.window="period")
decomposition
plot(decomposition)
```

you may want to control the window for **trend** extraction, for instance you impose a **3-years filter (36 months)**:

```
decomposition<-stl(mon.cum.ts7100.imp, s.window="period", t.window=36)
plot(decomposition)
```

2. by Structural time series models are (linear Gaussian) state-space models for (univariate) time series based on a decomposition of the series into a number of components:

```
decomposition2<-StructTS(mon.cum.ts7100.imp,type="BSM")
str(decomposition2)
plot(decomposition2$fitted)

#check the function output for fitted values
```

3. by Arima models (three integer components (p, d, q) order, the degree of differencing, and the MA order):

```
decomposition3<-arima(mon.cum.ts7100.imp,order=c(2,0,3))

predict(decomposition3,n.ahead=12) #predicting the 12 months ahead
```

Summary

-

At this point you should be able to:

- manipulate all types of R objects
- apply generic R functions
- aggregate data at different time scales
- perform basic analysis of climate time series

In the next module, you will learn how to import **NetCDF files**, which is **one of the most common format for climate data**.

You will also be introduced to the basic operation of **raster package** such as **cutting**, **summarizing** and **interpolating gridded datasets**.

You may want to download the R script containing all the commands used in this module:

R script: mod2-data-import-ts.R

FINAL EXERCISE

FINAL EXERCISE

At the end of this module, you are asked to do an exercise (close this lesson and see the correspondent assignment).

“Plot the time series of yearly precipitation anomaly with respect to 1981-2010 climatology”

Solution Exercise 1

Solution:

```
mon.cumF<-tapply(X=dataRR$RR, INDEX=list(dataRR$YEAR, dataRR$MONTH), FUN=
sum, na.rm=FALSE)

length(mon.cumF[is.na(mon.cumF)==TRUE])
```

The dataset contains 63 NA

Solution Exercise 2

Solution:

```
mon.cum.ave.7100<-round(colMeans(x=mon.cum.7100, na.rm=T), digits=1)
mon.cum.ave.7100[10]
```

October cumulated precipitation amounts to **123.9**mm

Solution Exercise 3

Solution:

Seasonal cumulated precipitation:

```
seas.cum<-tapply(X=dataRR$RR, INDEX=list(dataRR$YEAR, dataRR$SEAS), FUN=sum, na.rm=TRUE)
```

Sub setting the matrix **seas.cum**:

```
seas.cum.7100<-seas.cum[as.character(c(1971:2000)),]
```

Seasonal cumulated precipitation averaged over 1971-2000:

```
seas.cum.ave.7100<-round(colMeans(x=seas.cum.7100, na.rm=T), digits=1)
```

Spring (MAM) cumulated precipitation amounts to 193.0 mm