Introduction to R

Network Analysis 2017

Sacha Epskamp

02-11-2017

Course information

- All information in syllabus on blackboard!
  - Tomorrow...
- Lectures every Monday and Thursday
  - Thursday will be more practical, but can still be a lecture
- Every week an assignment
  - Deadline 15:00 on Thursday!
  - Conceptual questions and practical questions
- Final Project (more info on Monday)
  - Paper and presentation in groups of 2. Topics may include:
    - Analyze your own dataset
    - Gather and analyze data
    - Review the literature (topics on blackboard)
    - Re-analyze an existing dataset

Individual Assignments

Each week, the assignment will be made available 15:00 on Thursday, and will be due 15:00 the next Thursday. Each assignment will contribute to 10% or 5% (first and last week) of your grade.

- Work on the assignments alone.
- Hand in a PDF file and an .R file.
- Make sure your PDF report is as standalone readable as possible. E.g., if you are asked to report a network, then include the plot in the PDF and not just say "look at .R file".
- Assignments are due before 15:00. If you do not hand in an assignment before 15:00, you will get a 1.
- If you encounter any problems, or have any feedback, please let me know before the deadline, as then I can take it into account or help you.

What is R?

- R is a statistical programming language
  - Statistical analysis
  - Data visualization
  - Data mining
  - General programming
- It is *open-source*
  - Free as in “free beer”" and “free speech”
  - Large active community around R
  - Many contributed packages

What do you need to know about R?

- In this course not much. We intend the methods we will describe to be usable by researchers with little experience in R
  - No programming
  - Cookbook method
- But it is strongly advised that you learn more about R if you want to do more in methodology or psychometrics.
- To install R and RStudio see links section on Blackboard

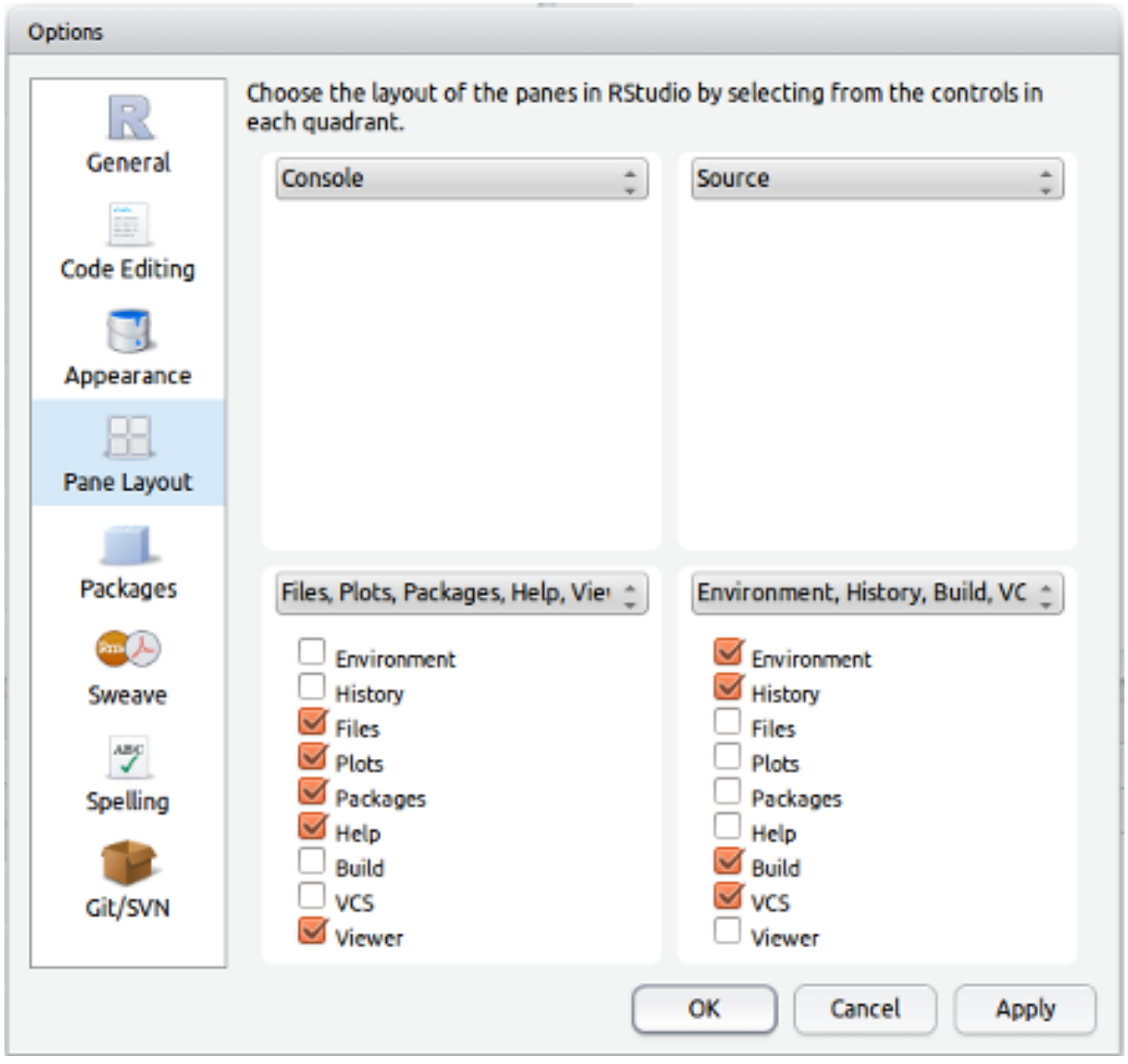
Installing R and RStudio

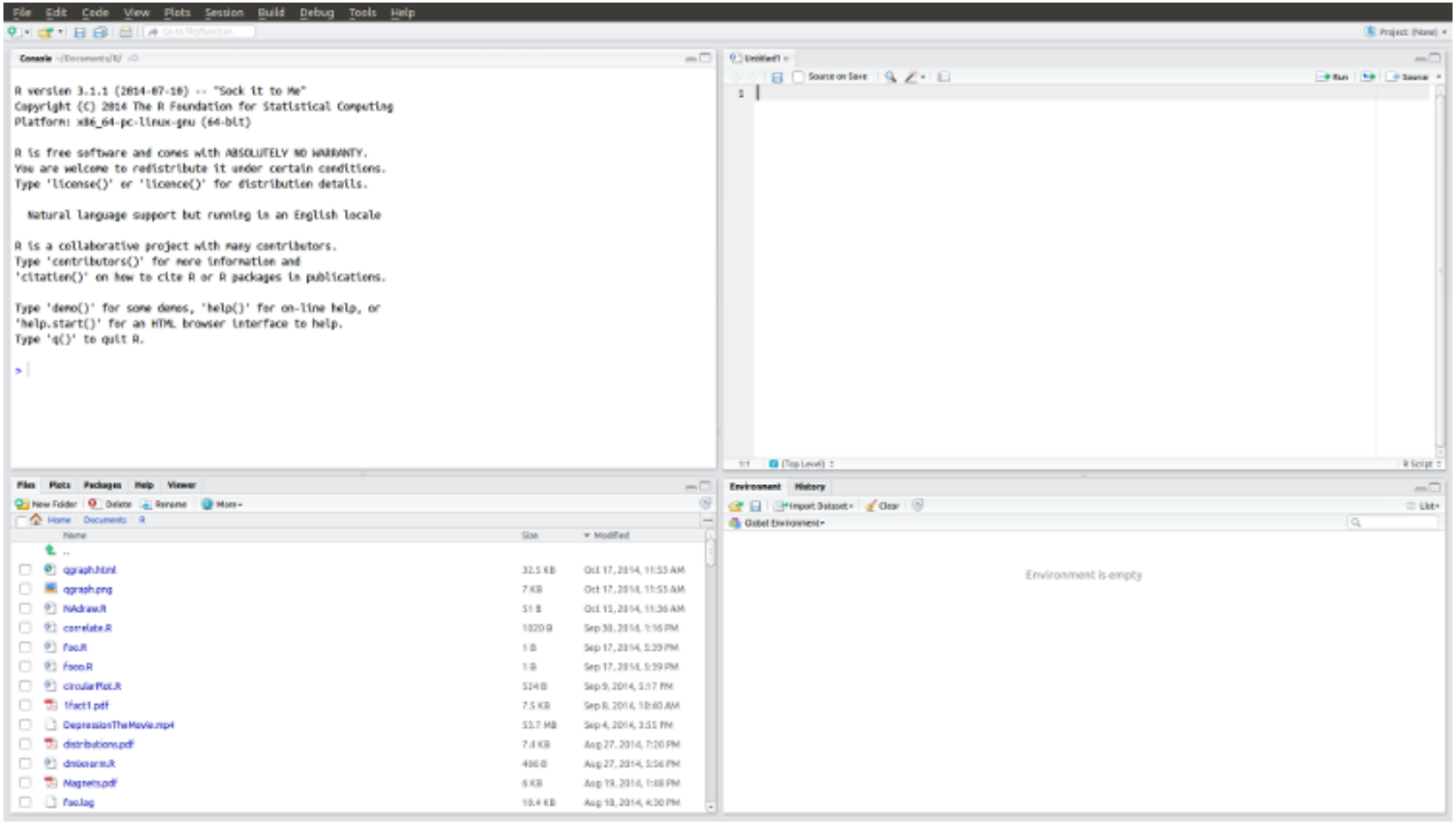
- R
  - This is the base programming language
  - Downloadable from `http://cran.r-project.org/`
  - It gives a program called R or RGui which we will **never** use
- RStudio
  - New and very good GUI for R
  - Downloadable from `http://www.rstudio.com/products/rstudio/download/`
- Always use R from within RStudio!

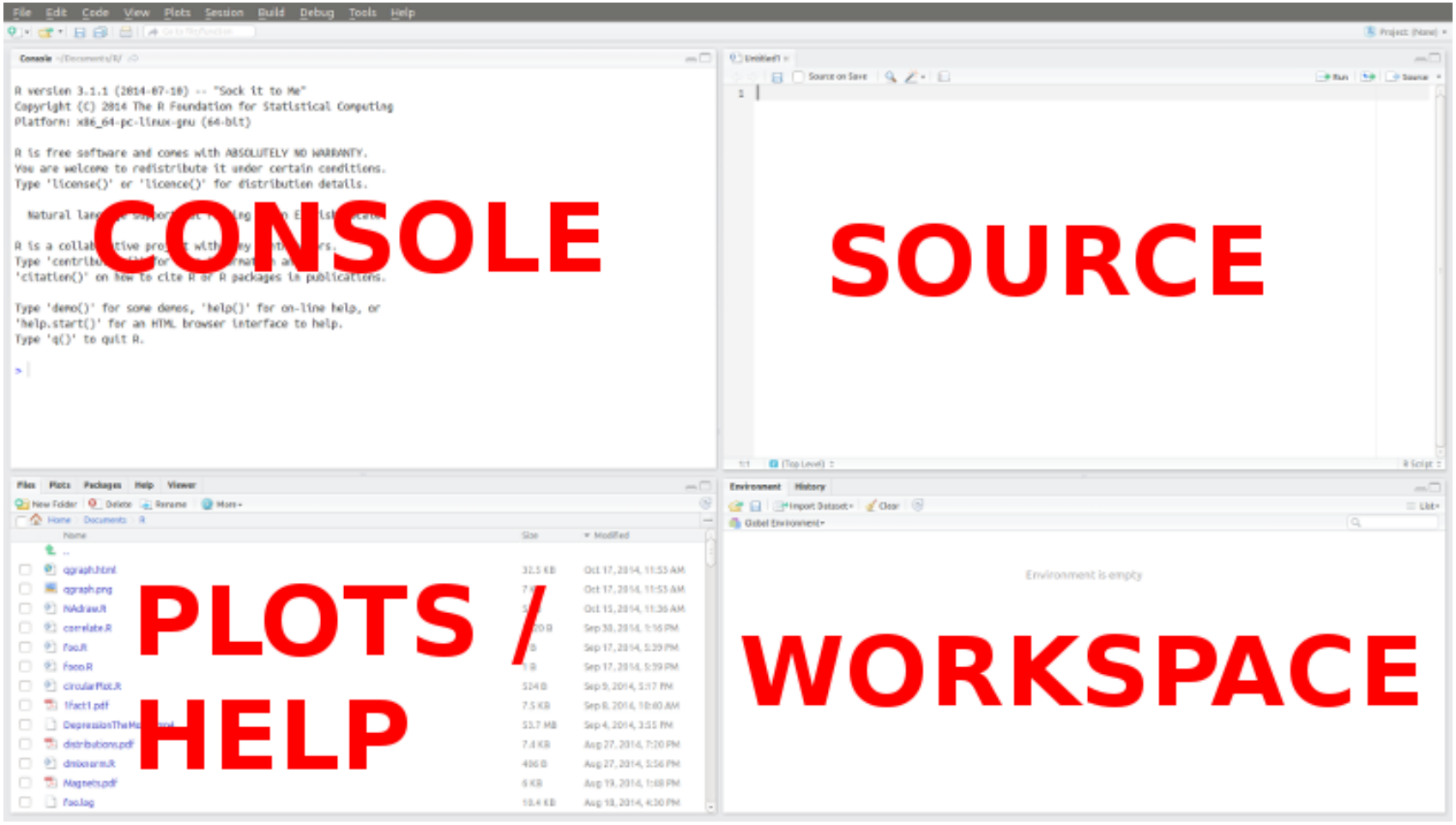
First use of R

- We will use the environment RStudio for our work in R
- RStudio has 4 panes:
  - **Console** This is the actual R window, you can enter commands here and execute them by pressing enter
  - **Source** This is where we can edit *scripts*. It is where you should always be working. Control-enter sends selected codes to the console
  - **Plots/Help** This is where plots and help pages will be shown
  - **Workspace** Shows which objects you currently have
- Anything following a # symbol is treated as a comment!

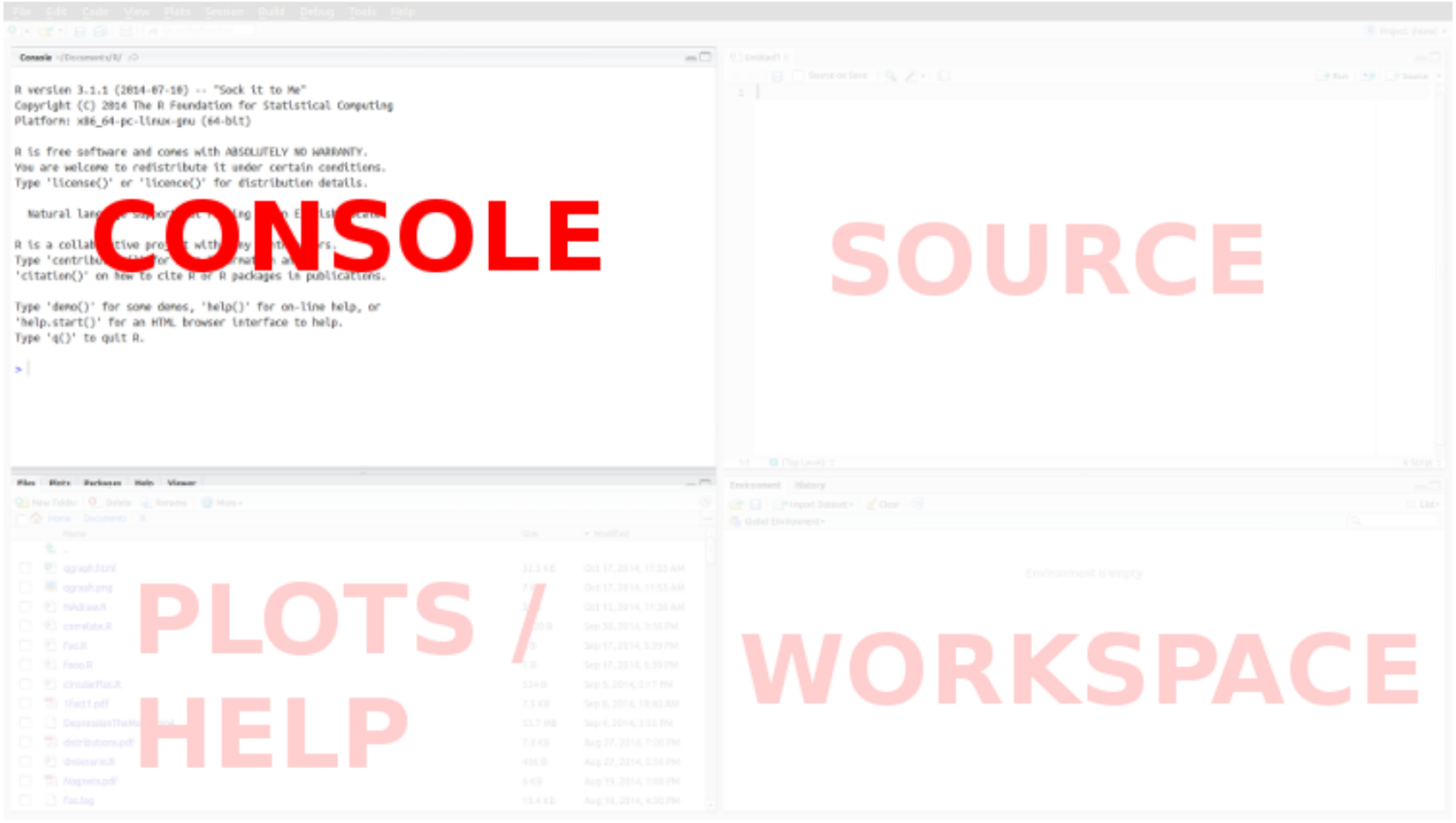
To change the layout of panes in RStudio, fo to Tools -> Global Options -> Pane Layout. Make sure you set it like this:




File Edit Code View Plots Session Build Debug Tools Help
Console
R version 3.1.1 (2014-07-10) -- "Sock It to Me"
Copyright (C) 2014 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.
Natural language support but running in an English locale
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.
Source on Save
Run
Source
Files Plots Packages Help Viewer
New Folder Delete Rename More
Environment History
Import Dataset Clear
Global Environment
Environment is empty


CONSOLE
SOURCE
PLOTS /
HELP
WORKSPACE

The R console



The R console

- The console pane is where R operates
- In the console pane, we can type commands and press enter to have R evaluate them
  - To ask R the answer to $1 + 1$, we can type in `1 + 1` and press enter:

```
> 1 + 1
[1] 2
```

- The arrow-up key lets us go back to a previous command
- This is how R works, we write commands for R to evaluate
  - Commands can be as simple as `1 + 1` or much longer encoding an entire data analysis

In the sheets, we use gray boxes to demonstrate commands we send to R. The values that R returns are shown underneath the gray boxes followed by the ## symbols.

For example, the command:

```
> 1 + 1
[1] 2
```

will be referred to in the sheets as:

```
1 + 1
```

```
## [1] 2
```

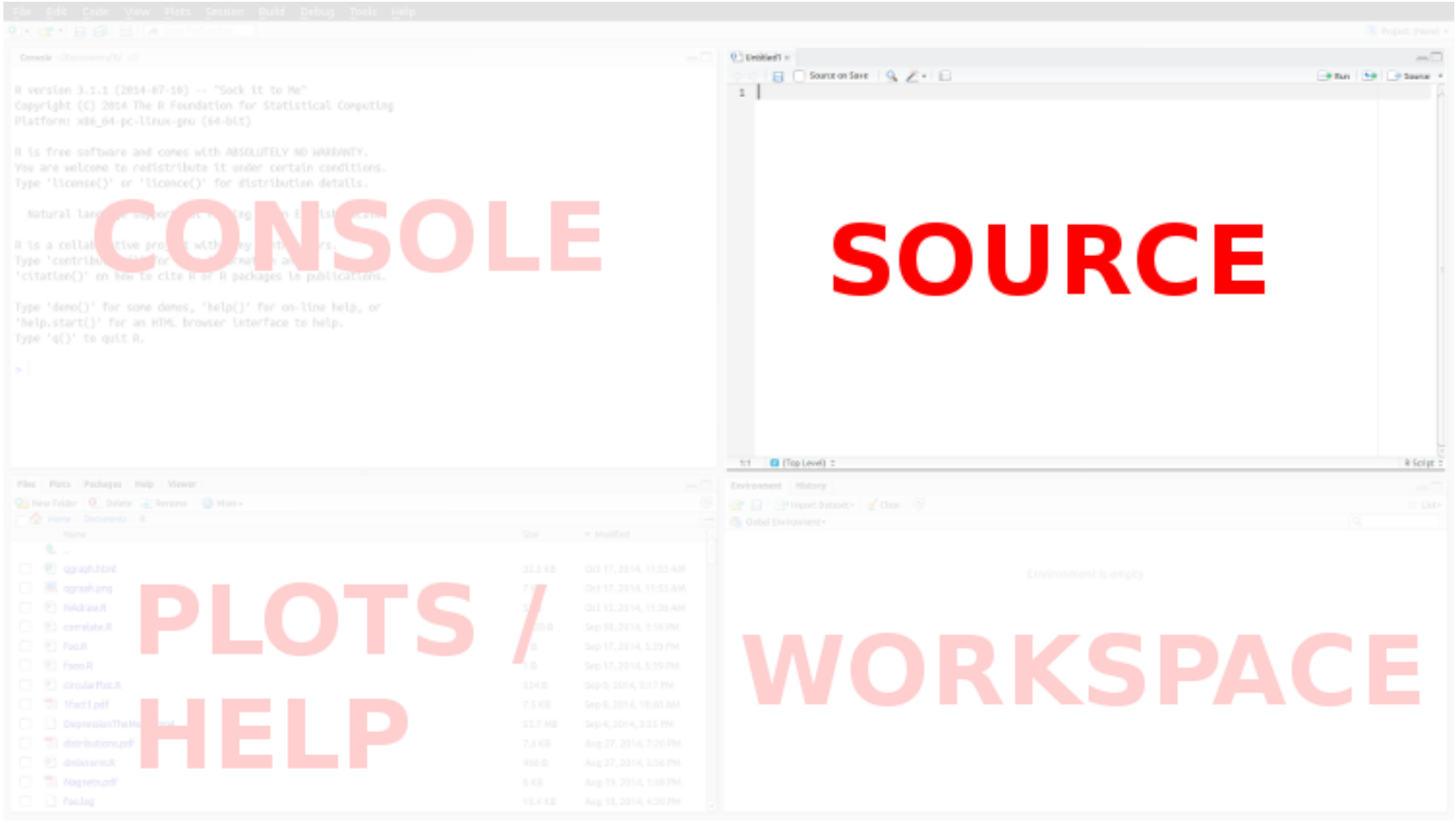
Comments in R

```
1 + 1 # Here I sum 1 and 1!
```

```
## [1] 2
```

- The # sign is used because R stops evaluating any command whenever it sees #
  - Everything after # is **not** an R command.
  - # can be used as comments to clarify your code
    - Do this as often as you can!

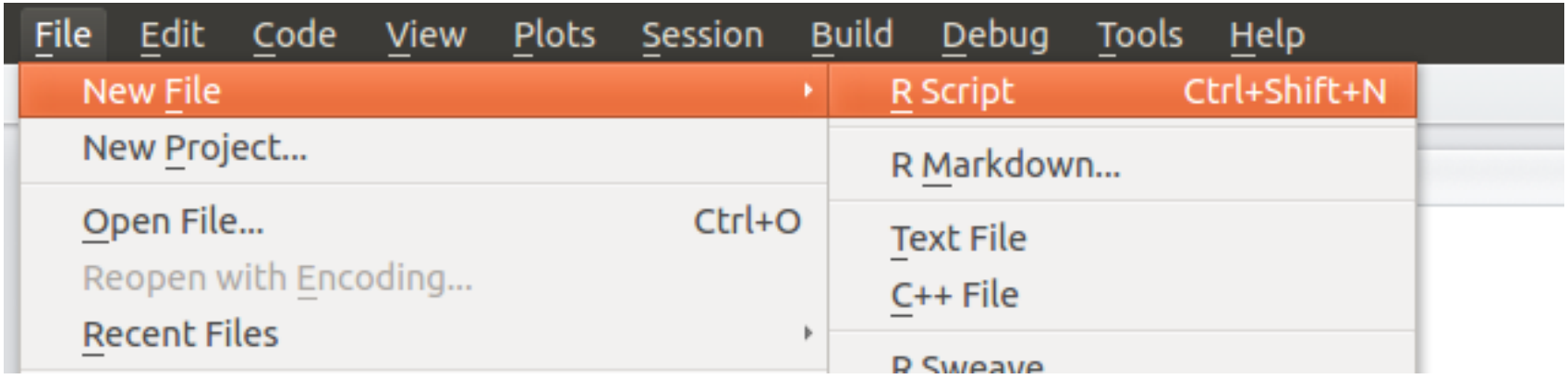
The source pane



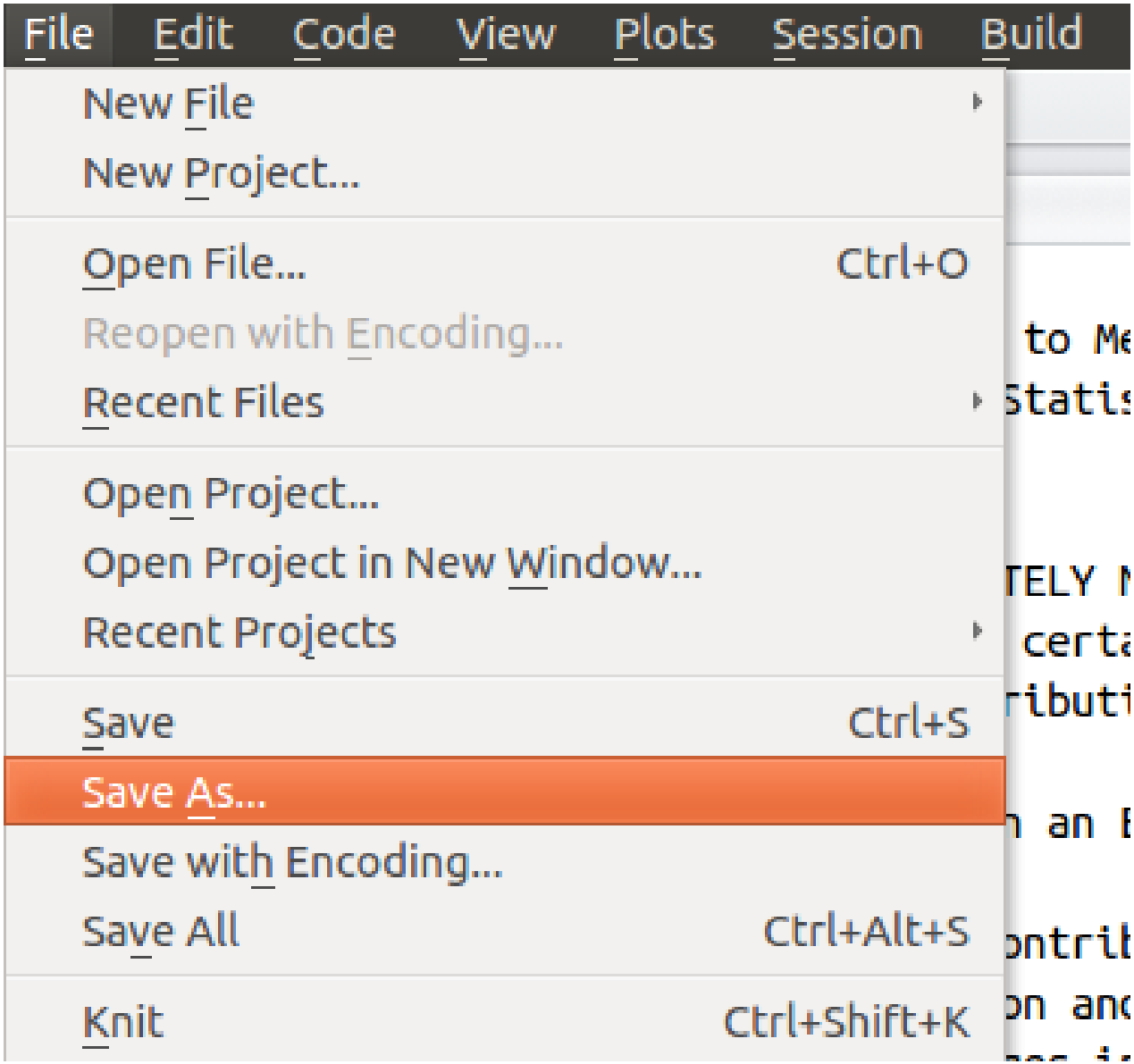
The source pane

- Often, you need to evaluate many commands sequentually
  - A command to read the data
  - Comands to transform the data
  - Commands to plot the data
  - Commands to analyze the data
  - ...
- You want to be able to save these commands so that you can later reuse them
- To do this, you need to write an *R script*
  - Plain-text file with the extension `.R` that contains all your codes
- R scripts are written and edited in the *source pane*

Create a script using file -> New File -> R script:

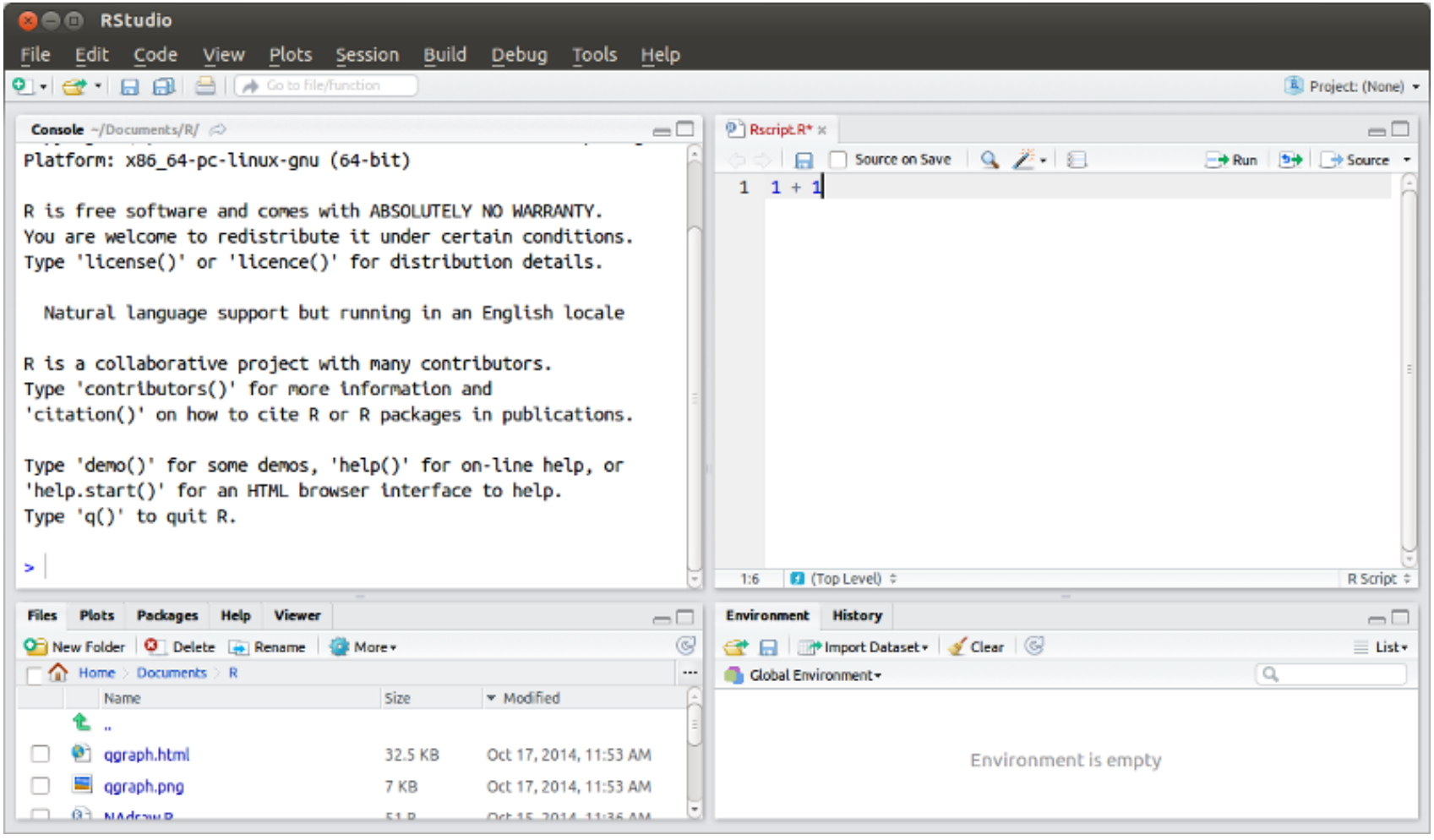


Save the script using file -> Save as:

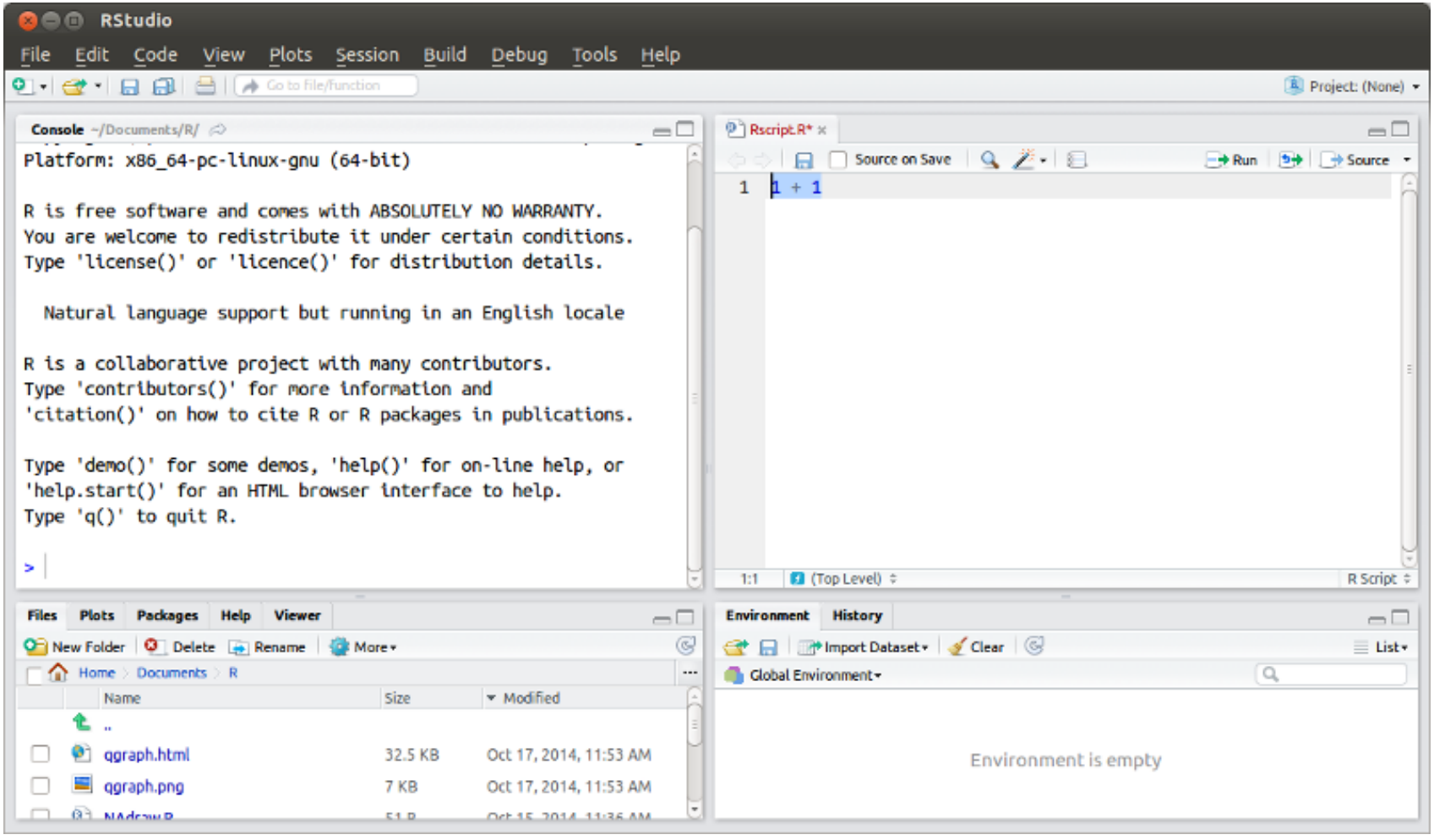


- Save scripts with the extension `.R`.

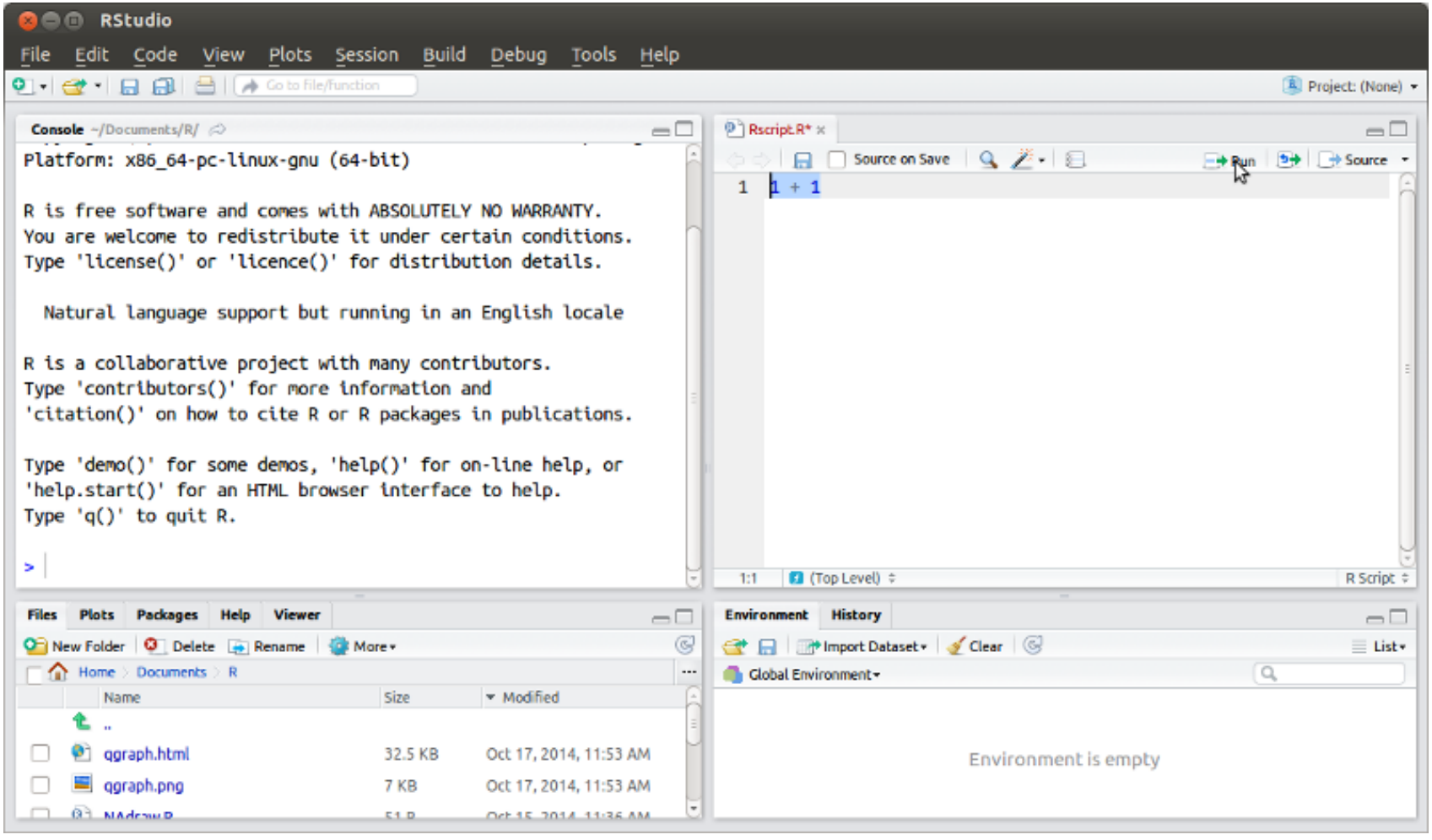
Type **all** your commands into the script:



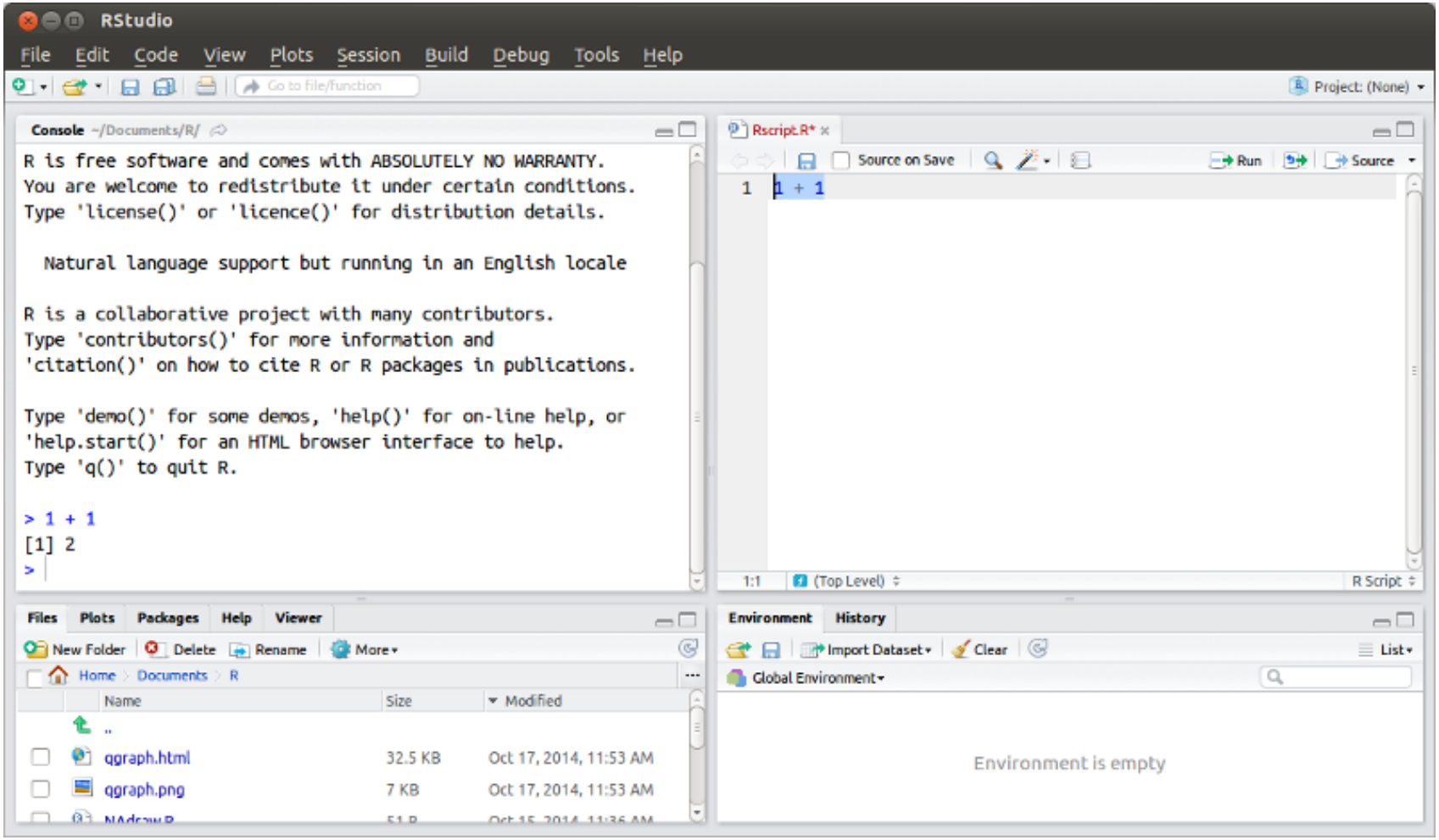
Select the commands you want to evaluate:



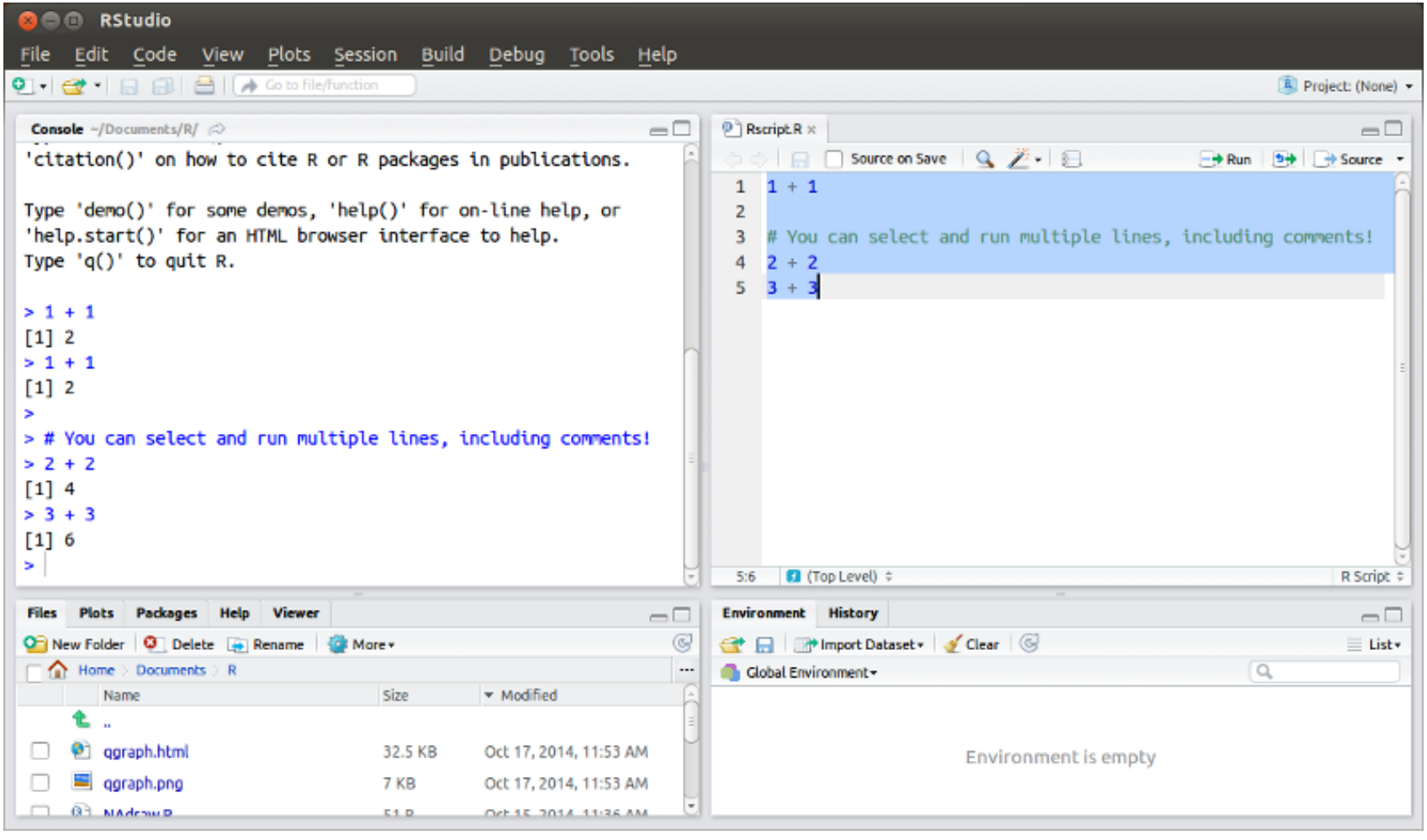
Press the run button or control + enter on your keyboard (command + enter for Mac users)



Now the command is sent to and evaluated in the R console!



- Never directly use the console! Always work like this!


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Project: (None)
Console ~/Documents/R/
'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.
> 1 + 1
[1] 2
> 1 + 1
[1] 2
>
> # You can select and run multiple lines, including comments!
> 2 + 2
[1] 4
> 3 + 3
[1] 6
>
Rscript.R
Source on Save
Run
Source
1 1 + 1
2
3 # You can select and run multiple lines, including comments!
4 2 + 2
5 3 + 3
5:6
(Top Level)
R Script
Files Plots Packages Help Viewer
New Folder
Delete
Rename
More
Home Documents R
Name
Size
Modified
..
qgraph.html
32.5 KB
Oct 17, 2014, 11:53 AM
qgraph.png
7 KB
Oct 17, 2014, 11:53 AM
Environment History
Import Dataset
Clear
List
Global Environment
Environment is empty

Assign and use objects

- The <- operator can be used to store values into *objects*
  - An object can be given an arbitrary name
  - Alternatively = can be used to do the same thing
- an object can contain anything in R
- R expressions that are not stored in an object are *printed*

You can assign the number 1 to an object which I call a:

```
a <- 1
```

You can assign the number 1 to an object which I call a:

```
a <- 1
```

Now whenever I ask R What a is it gives me 1:

```
a
```

```
## [1] 1
```

You can assign the number 1 to an object which I call a:

```
a <- 1
```

Now whenever I ask R What a is it gives me 1:

```
a
```

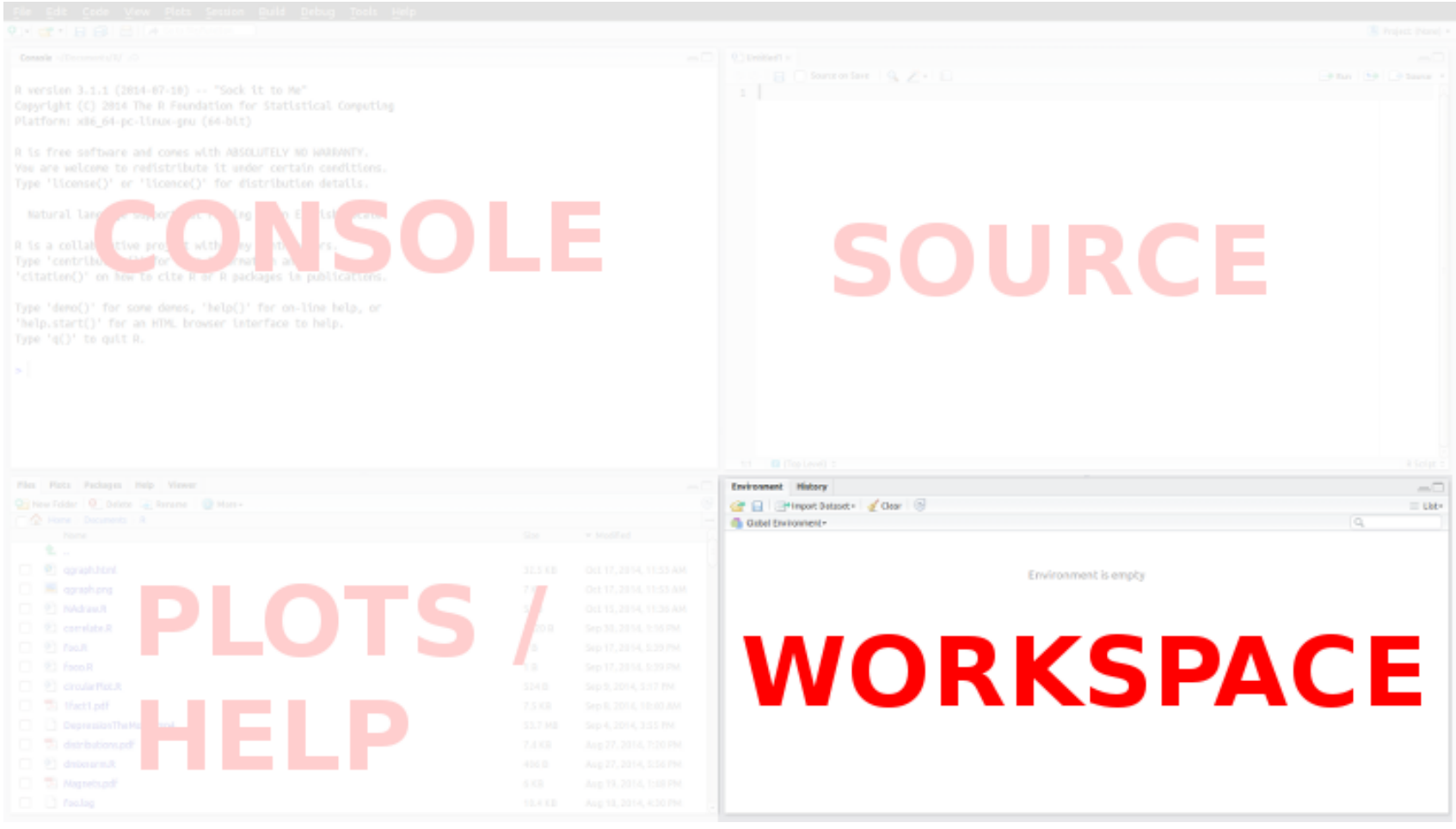
```
## [1] 1
```

You can use this object in calculations:

```
a + 1
```

```
## [1] 2
```

All your objects are listed in the workspace pane:



Objects can have any name you want:

```
anyNameYouWant <- 10
anyNameYouWant
```

```
## [1] 10
```

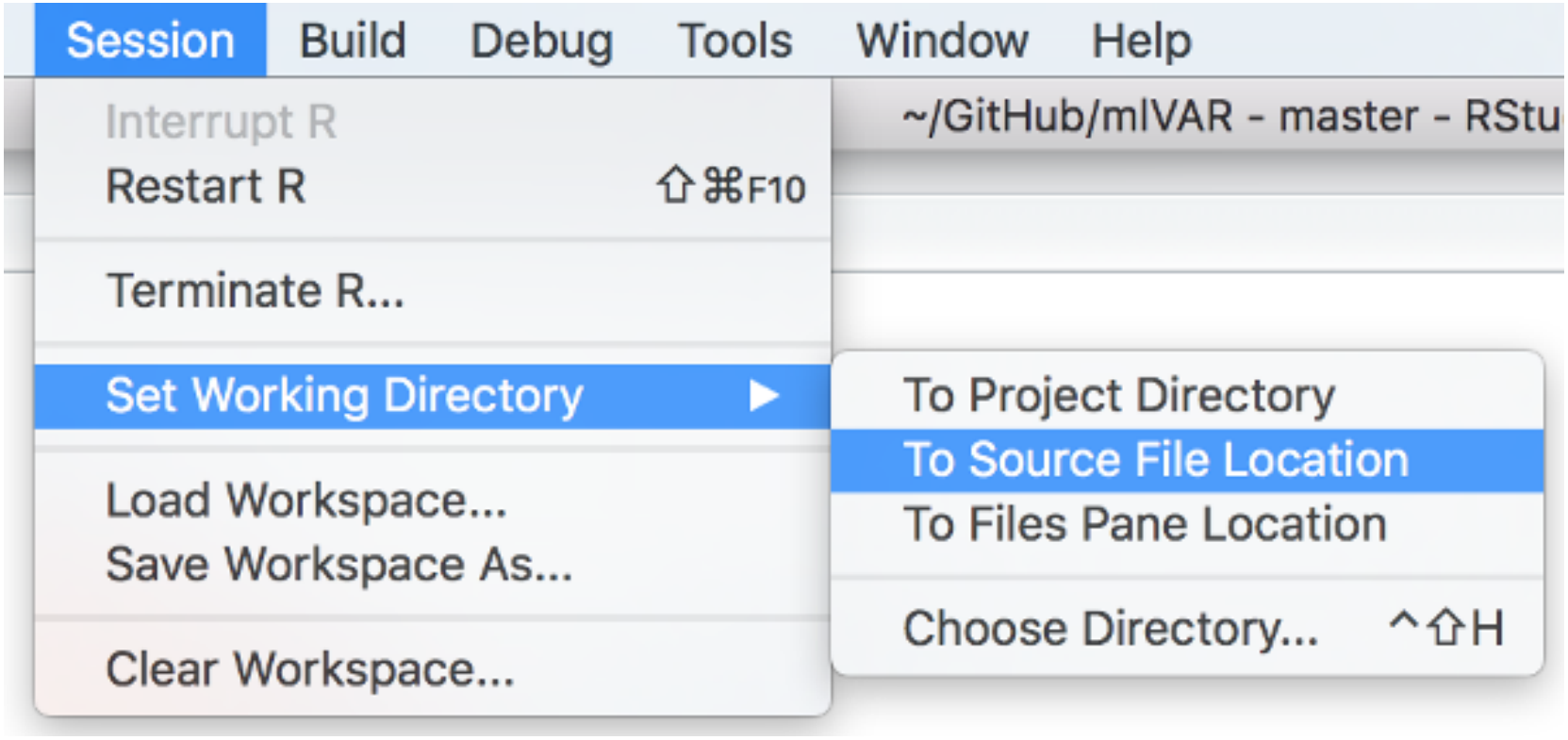
- Case sensitive! `object`, `Object` and `OBJECT` are **different** objects!
- It is recommended to use informative object names
  - `sampleSize`
  - `rawData`
  - `nMales`
- Use *camel case*
  - Start object name with lowercase letters, start new objects with lowercase and capitalize every word

Working Directory

The working directory is the folder on your computer in which R works:

```
getwd()
setwd("~/Documents")
getwd()
```

I recommend to never use `setwd()` or `getwd()` but to use the RStudio menu:



Write R codes assuming this is done (R script in same folder as data and/or output).

Vectors

- A vector is an object that stores multiple values of the same mode
- Use `c(...)` to manually create a vector
  - For example, `c(1,2,3)` creates the vector $\begin{bmatrix}1 & 2 & 3\end{bmatrix}$
- A colon can be used to create an integer sequence

Vectors

Create a vector using `c(...)`:

```
numericVector <- c(1, 5, 10)
numericVector
```

```
## [1]  1  5 10
```

Functions

- A *function* is a small program, it takes input, does something and gives you output
- It is always of the form `name(argument, argument, argument, ...)`
- A documentation can be found for every function using `?name`
  - The documentation tells you what you need to put into a function and what comes out of it
- You need to know the exact name of the function you want to use
  - Reference card
  - Use Google!

Functions

```r
a <- c(5, 2, 10, 1)
# Compute mean of v1:
mean(a)
```

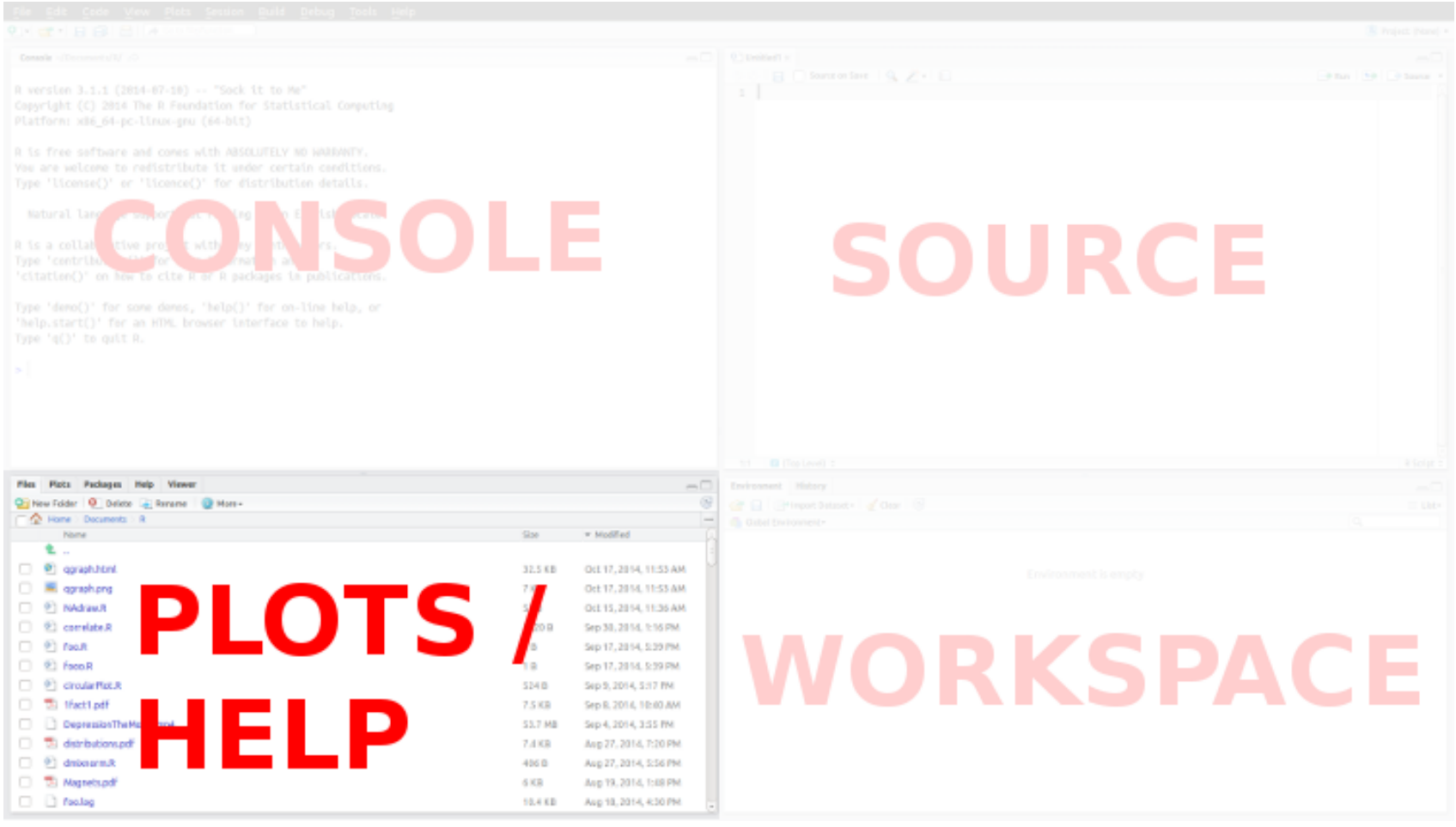
```
## [1] 4.5
```

```r
# Compute sum of v1:
sum(a)
```

```
## [1] 18
```

```r
# Help pages of these functions:
?mean
?sum
```

Help pages appear in the plots/help pane:



Object Modes

- There are different types of objects in R:
  - `numeric`
  - `character`
  - `logical`
- There are called *modes*

The numeric mode stores numbers:

```
numericObject <- 115
numericObject + 1
```

```
## [1] 116
```

Strings

Anything between single or double quotation marks is stored as a `character`, which can be used to encode strings:

```
characterObject <- 'this is a string'
characterObject
```

```
## [1] "this is a string"
```

```
characterObject2 <- '1'
characterObject2  + 1
```

```
## Error in characterObject2 + 1: non-numeric argument to b
```

Logicals

The `logical` mode indicates a boolean object which can only be true (`TRUE`) or false (`FALSE`):

```
logicalObject <- TRUE
logicalObject
```

```
## [1] TRUE
```

- Shortcuts for `TRUE` and `FALSE` are `T` and `F`
  - Avoid using these shortcuts!

Packages

- Packages are extensions contributed to R containing extra functions
  - Stored on the Comprehensive R Archive Network (CRAN)
  - Over 8000 contributed packages! All containing functions
- Packages can be installed using `install.packages()`
  - Needs to be done only once
- Afterwards they can be loaded using `library()`
- More information on packages on the CRAN site for that package:
  - cran.r-project.org/package=PACKAGENAME
  - Or search for CRAN PACKAGENAME

Packages

```
# Install package "qgraph"
install.packages("qgraph")

# Load package "qgraph":
library("qgraph")
```

Matrices

- A very important function for this course is `matrix()`. It creates a matrix, which is basically a two dimensional table.
- Technically, a matrix is a vector with two dimension attributes
- Rows indicate horizontal lines of cells
- Columns indicate vertical lines of cells
- The first argument of `matrix` is a vector to fill the matrix with, the second argument the number of rows and the third argument the number of columns (see also `?matrix`)
- Again they can be indexed with square brackets, but now need both row and column information, separated by a comma

Matrices

```
# A matrix:
m <- matrix(1:9, nrow=3, ncol=3)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Indexing matrices

```
# Index first row second column:
m[1,2]
```

```
## [1] 4
```

```
# Overwrite an element:
m[3,3] <- 0
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    0
```

Indexing matrices

```
# First row and all columns:
m[1,]
```

```
## [1] 1 4 7
```

```
# First 2x2 block:
m[1:2,1:2]
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
```

```r
# Number of rows:
nrow(m)
```

```
## [1] 3
```

```r
# Number of columns:
ncol(m)
```

```
## [1] 3
```

Lists

```r
# A vector:
v1 <- c(5,10,1,3)
v1
```

```
## [1]  5 10  1  3
```

```r
# A matrix:
m1 <- matrix(c(5, 2, 10, 1),2,2)
m1
```

```
##      [,1] [,2]
## [1,]    5   10
## [2,]    2    1
```

```r
# Put them in a list:
l1 <- list(v1 = v1, m1 = m1)
```

Lists

```
str(l1)
```

```
## List of 2
##  $ v1: num [1:4] 5 10 1 3
##  $ m1: num [1:2, 1:2] 5 2 10 1
```

Indexing lists

```
# Index the vector v1:
l1$v1
```

```
## [1]  5 10  1  3
```

```
l1[['v1']]
```

```
## [1]  5 10  1  3
```

```
# Change an element in the matrix m1:
l1$m1[2,2] <- 0
l1$m1
```

```
##      [,1] [,2]
## [1,]    5   10
## [2,]    2    0
```

Data frames

- The `data.frame()` function can be used to create a *data frames*
- A data frame is a combination of a matrix and a list
  - Looks like a matrix and can be indexed as one
  - Contains variables as each row, which can be indexed as in lists
- This is the structure you will use when working with data
- Very similar to how data is stored in SPSS!

Data frames

```r
# A character vector:
sex <- c("male","female","male","female")

# A logical vector:
exp <- c(TRUE,TRUE,FALSE,FALSE)

# 2 numeric vectors:
A <- c(5,10,1,3)
B <- 1:4

# Put them in a data frame:
df1 <- data.frame(sex=sex,exp=exp,A=A,B=B)
```

Data frames

```
df1
```

```
##      sex   exp  A B
## 1   male  TRUE  5 1
## 2 female  TRUE 10 2
## 3   male FALSE  1 3
## 4 female FALSE  3 4
```

Indexing data frames

```
# Index the vector sex:
df1$sex
```

```
## [1] male   female male   female
## Levels: female male
```

```
df1[['sex']]
```

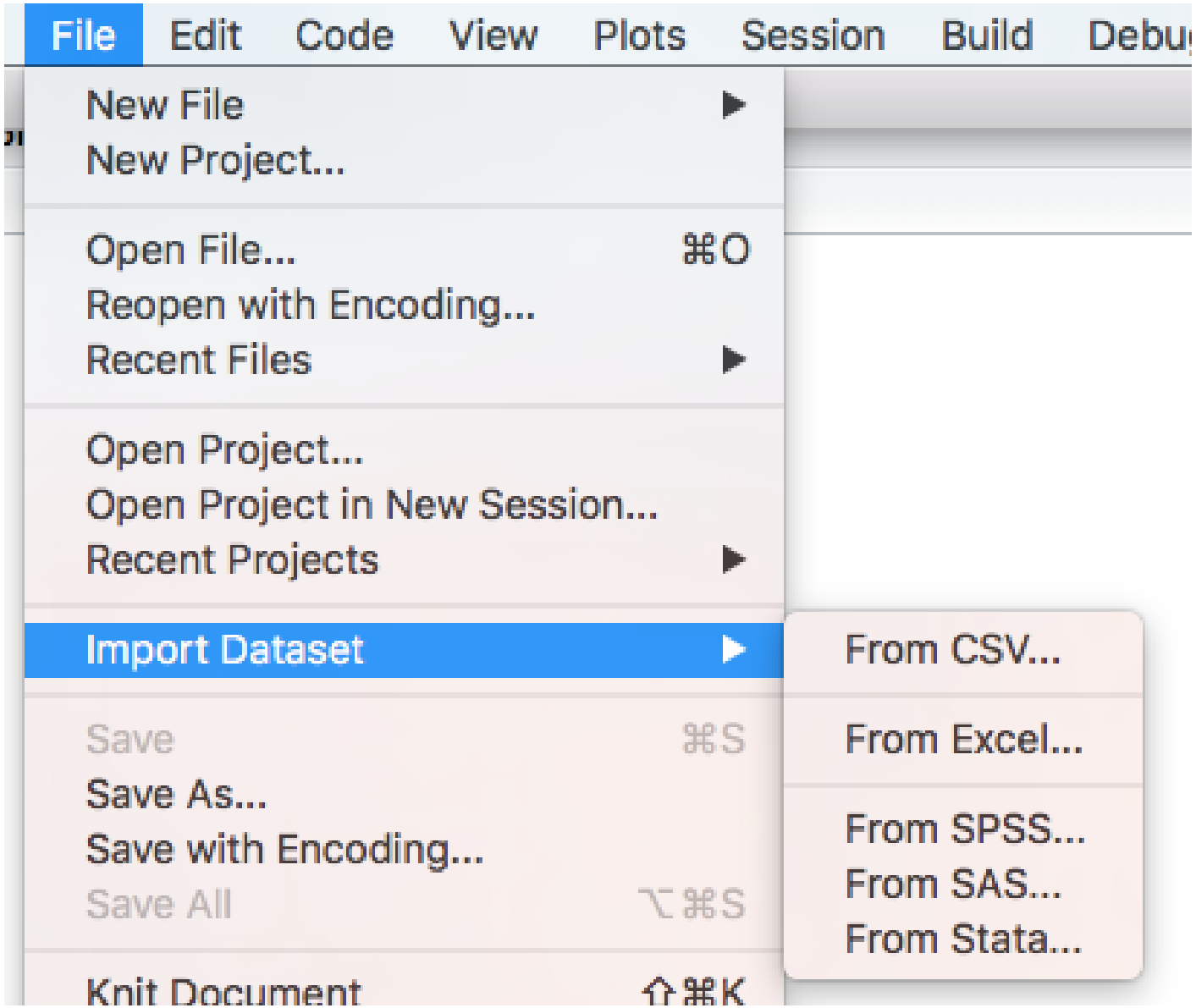
```
## [1] male   female male   female
## Levels: female male
```

```
# Subset of the data containing only A and B:
df1[,c('A','B')]
```

```
##    A B
## 1  5 1
## 2 10 2
## 3  1 3
```

Importing data

Is now very easy!



Correlation matrix

Some functions take a data frame as input. Very important in this course is the function `cor`, which takes a data frame as input and computes the correlation matrix. The function `cov` computes the covariance (unstandardized correlations).

```
# Covariance matrix:
cov(df1[,c('A','B')])
```

```
##           A         B
## A 14.91667 -2.500000
## B -2.50000  1.666667
```

```
# Correlation matrix:
cor(df1[,c('A','B')])
```

```
##              A          B
## A  1.0000000 -0.5013947
## B -0.5013947  1.0000000
```

Missing data

Missing data is encoded in R with `NA`

```
a <- NA
NA
```

```
## [1] NA
```

Missing data needs to be handled often using arguments in functions

Missing data

Check the help files! `?cor`

```
a <- c(1,2,NA,4)
b <- c(1,3,2,8)
cor(a,b)
```

```
## [1] NA
```

```
cor(a,b,use = "pairwise.complete.obs")
```

```
## [1] 0.9986254
```

Missing data

Or use `na.omit`

```
Data <- cbind(a,b)
cor(na.omit(Data),use = "pairwise.complete.obs")
```

```
##           a         b
## a 1.0000000 0.9986254
## b 0.9986254 1.0000000
```

Please work through the (very) short introduction to R (on blackboard and available at `https://cran.r-project.org/doc/contrib/Torfs+Brauer-Short-R-Intro.pdf`) *before* starting this assignment. You can skip section 11 on programming.