Introduction to R

PsychoSystems Winter School 2018

Gaby Lunansky (slides adapted from the Great R Magician dr. Sacha Epskamp)

January 29th, 2018

Introduction

- Introductory lecture
- Afterwards we have a practical
- Basical knowledge to follow this Winter School and work with our packages
- Ask questions!

I will also ask you questions during the lecture!

-Start this week awesomely, by showing off how smart you are!

-Get MAD respect

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"
  - Reproducibility!
  - Large active community around R
  - Many contributed packages

Why R?

- Free
- Extremely powerful
- State of the art statistical methods
    - For many analyses, R is simply the *only* choice of software!
- State of the art visualiations

AND: easy to learn!

For a programming language….

Installing R and RStudio

Installing R and RStudio

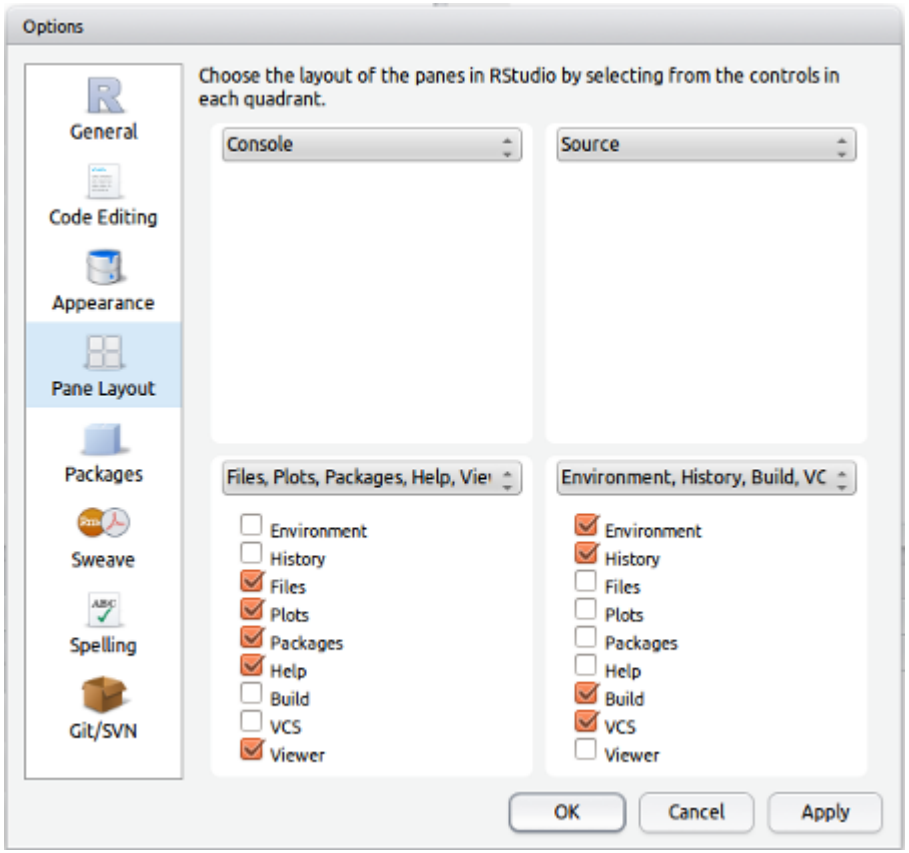
- R
  - This is the base programming language
  - Downloadable from http://cran.r-project.org/
  - It gives a program called R or RGui which we will **never** use
- RStudio
  - New and very good GUI for R
  - Downloadable from http://www.rstudio.com/products/rstudio/download/
- Always use R from within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

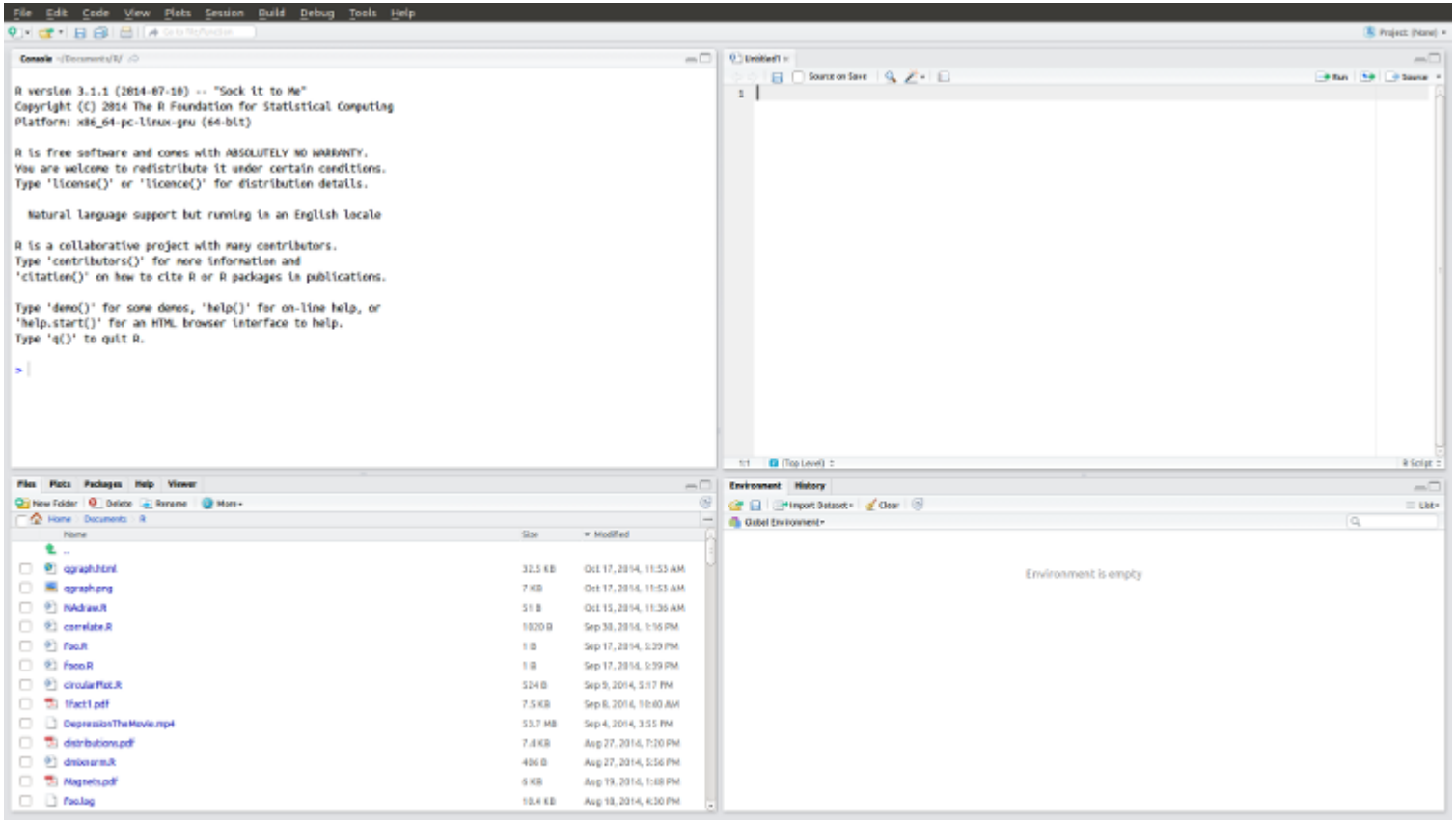
First use of R

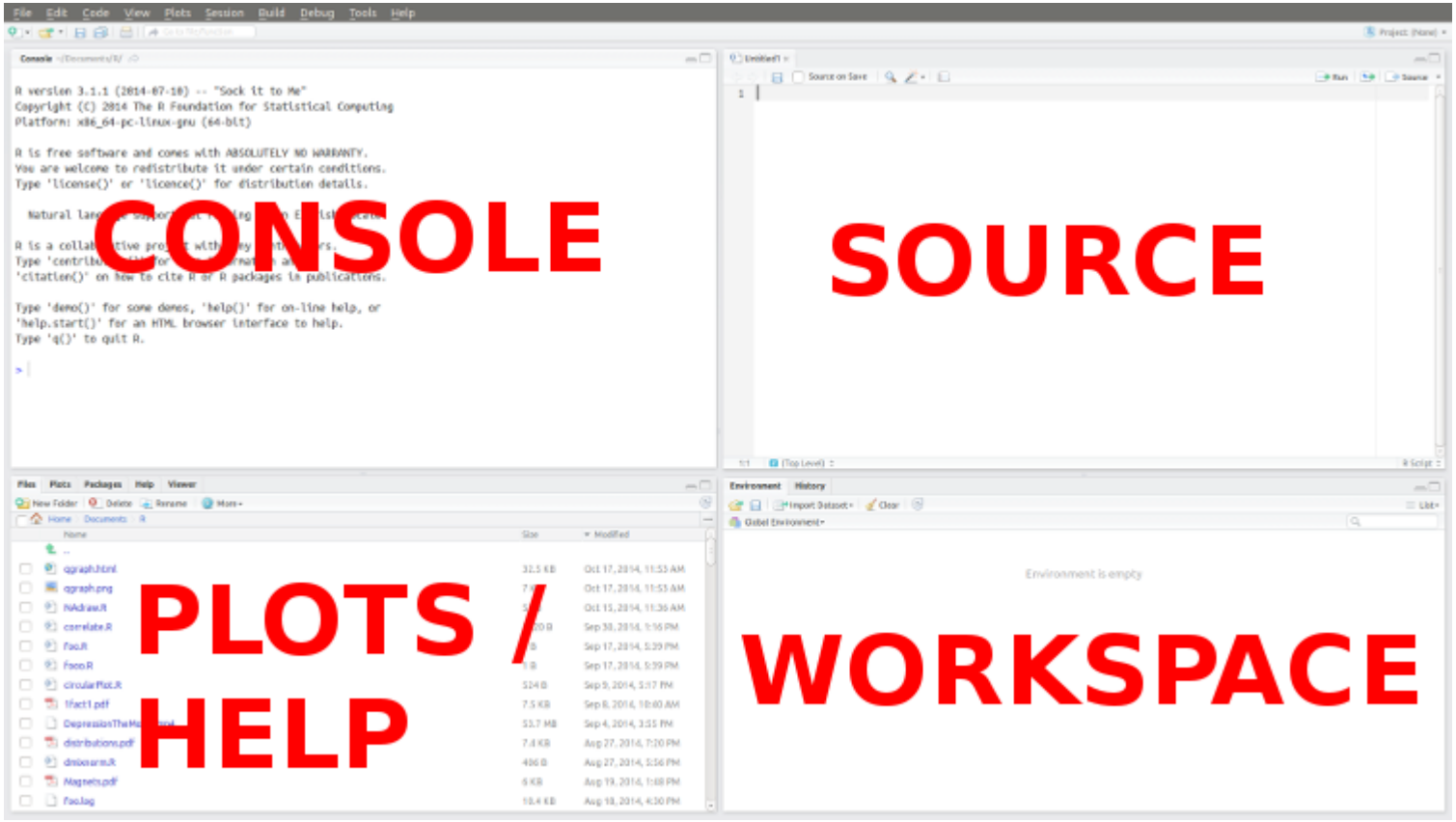
First use of R

- 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!

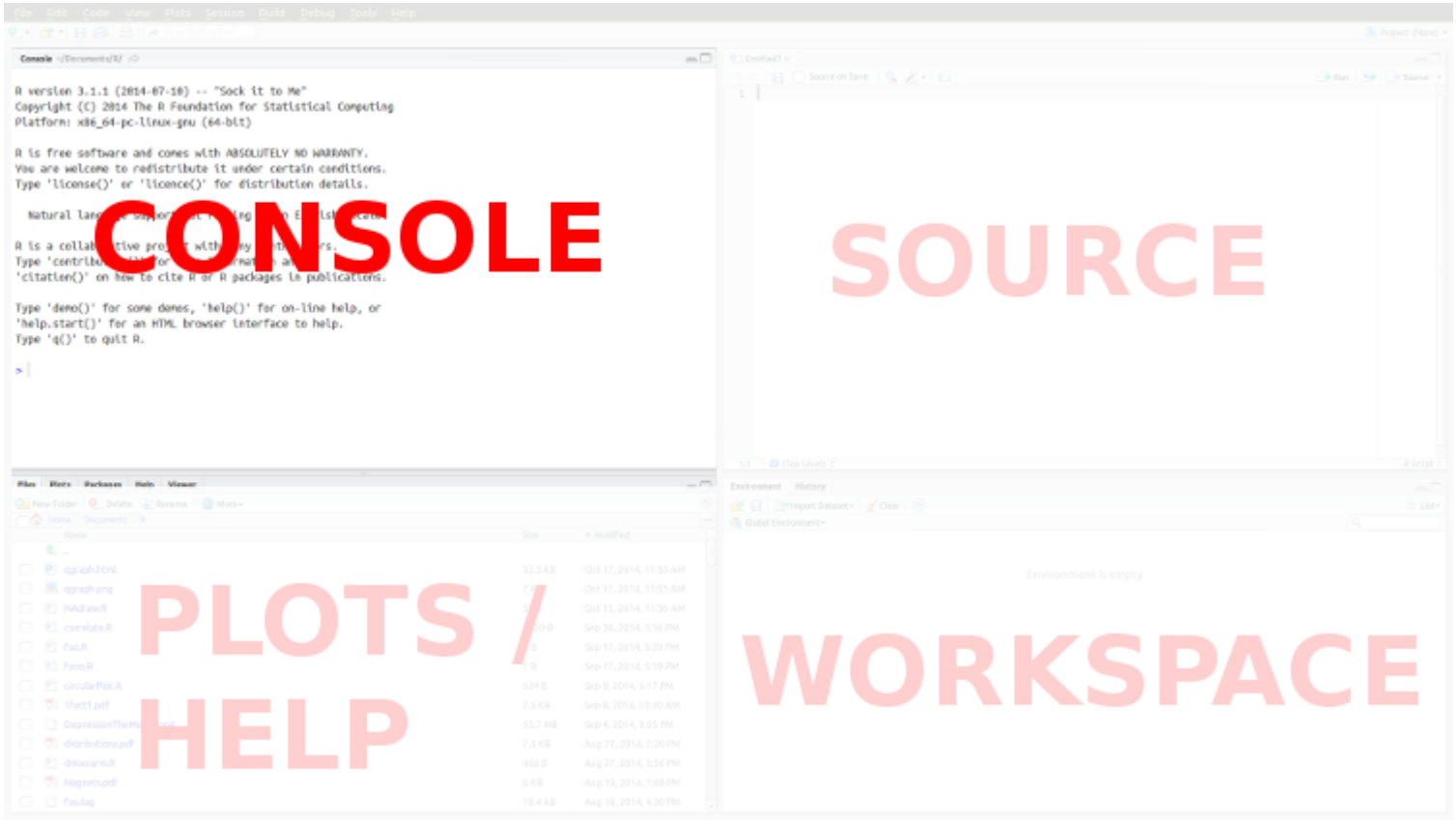
To change the layout of panes in RStudio, fo to Tools -> Global Options -> Pane Layout. Make sure you set it like this:




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
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
NAdraw.R 51 B Oct 15, 2014, 11:36 AM
correlate.R 1020 B Sep 30, 2014, 1:16 PM
Foo.R 1 B Sep 17, 2014, 5:39 PM
Foo0.R 1 B Sep 17, 2014, 5:39 PM
circularPlot.R 524 B Sep 9, 2014, 3:17 PM
1fact1.pdf 7.5 KB Sep 8, 2014, 10:00 AM
DepressionTheMovie.mp4 53.7 MB Sep 4, 2014, 3:55 PM
distributions.pdf 7.4 KB Aug 27, 2014, 7:20 PM
dmixnorm.R 406 B Aug 27, 2014, 5:56 PM
Magnets.pdf 6 KB Aug 19, 2014, 1:08 PM
Foo.log 10.4 KB Aug 18, 2014, 4:30 PM
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
```

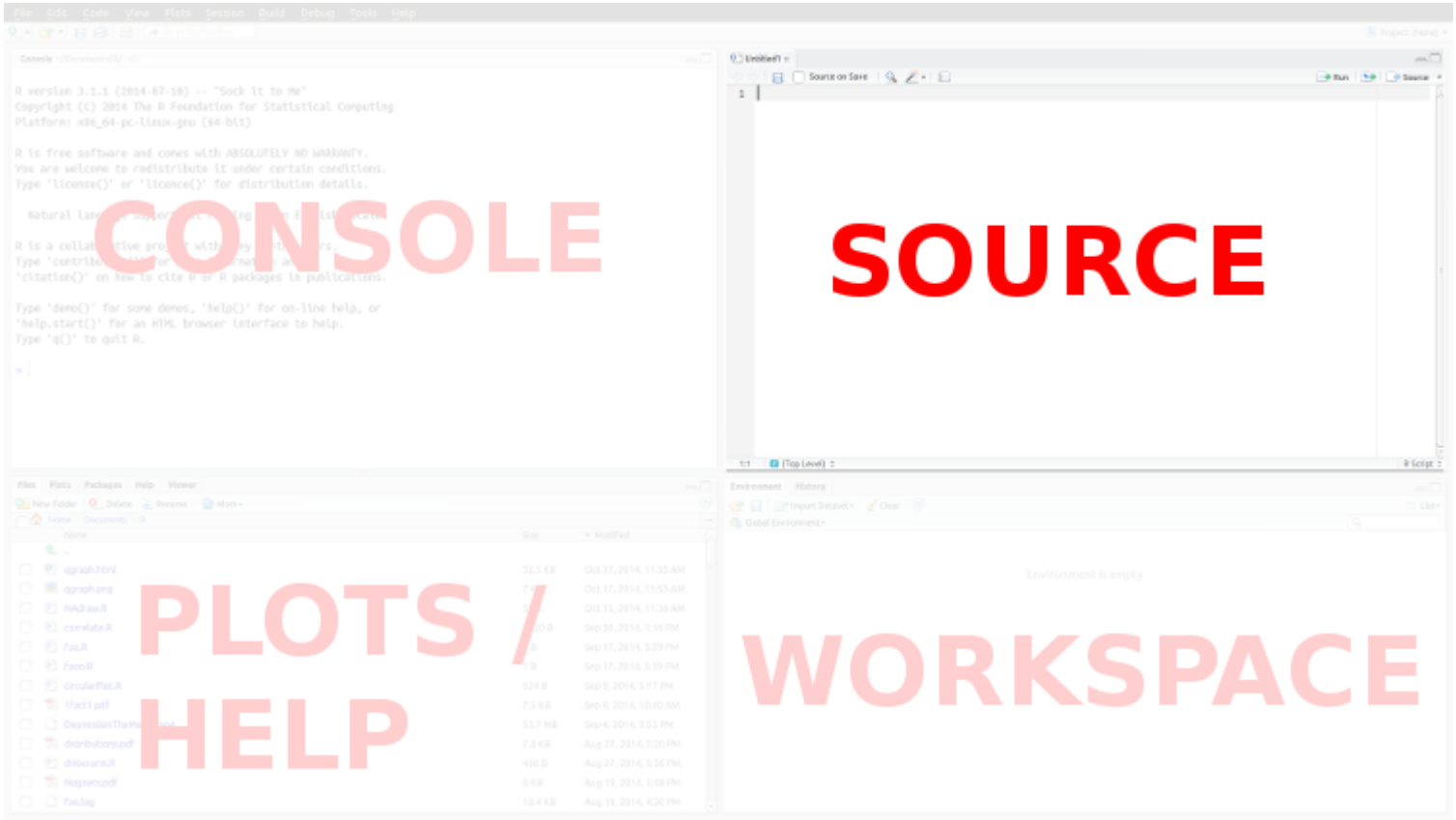
Comments in R

```
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!

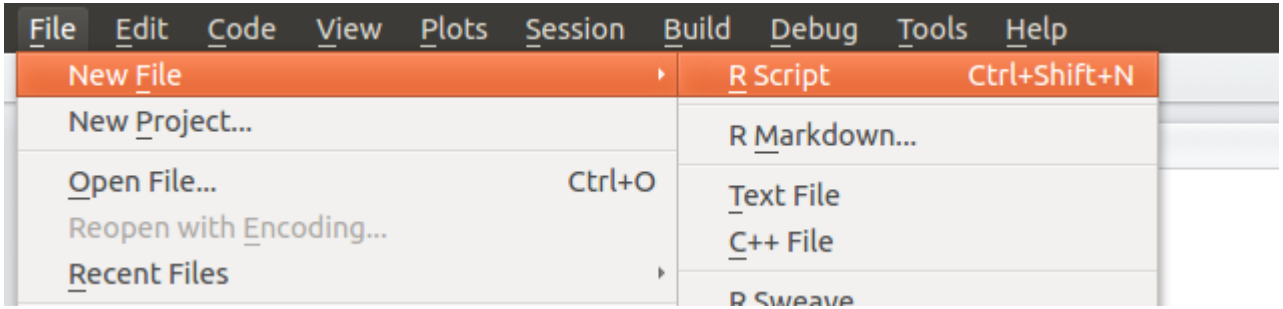
The source pane



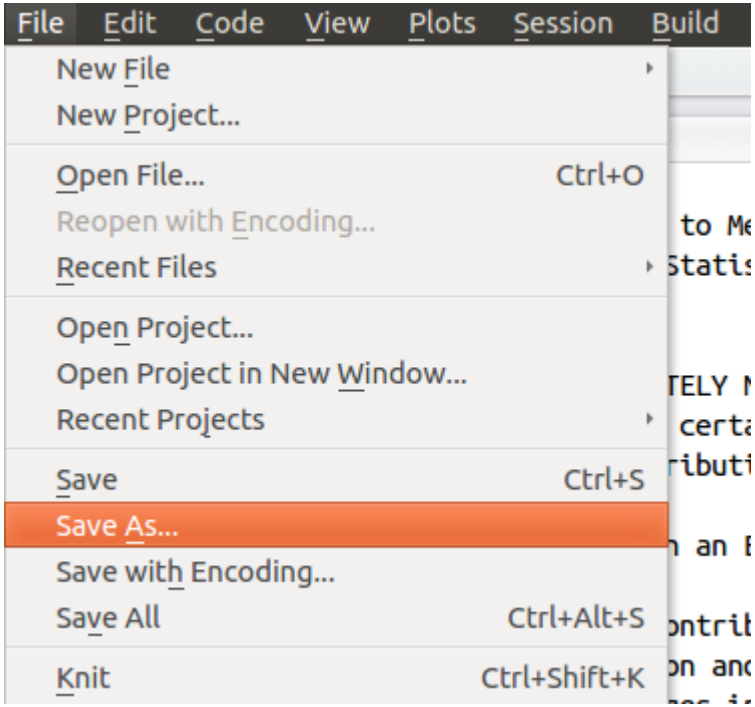
The source pane

- 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:

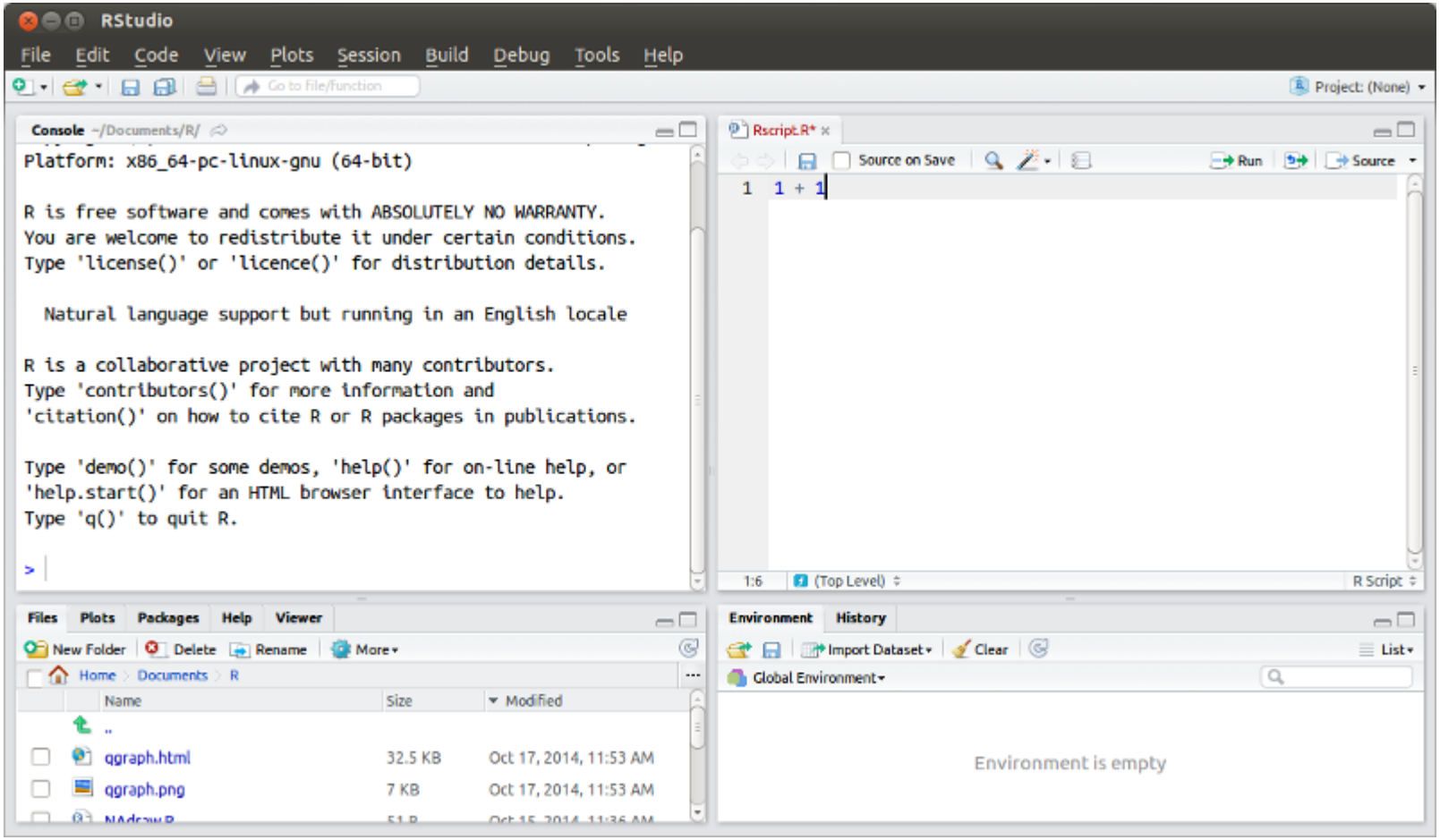


Save the script using file -> Save as:

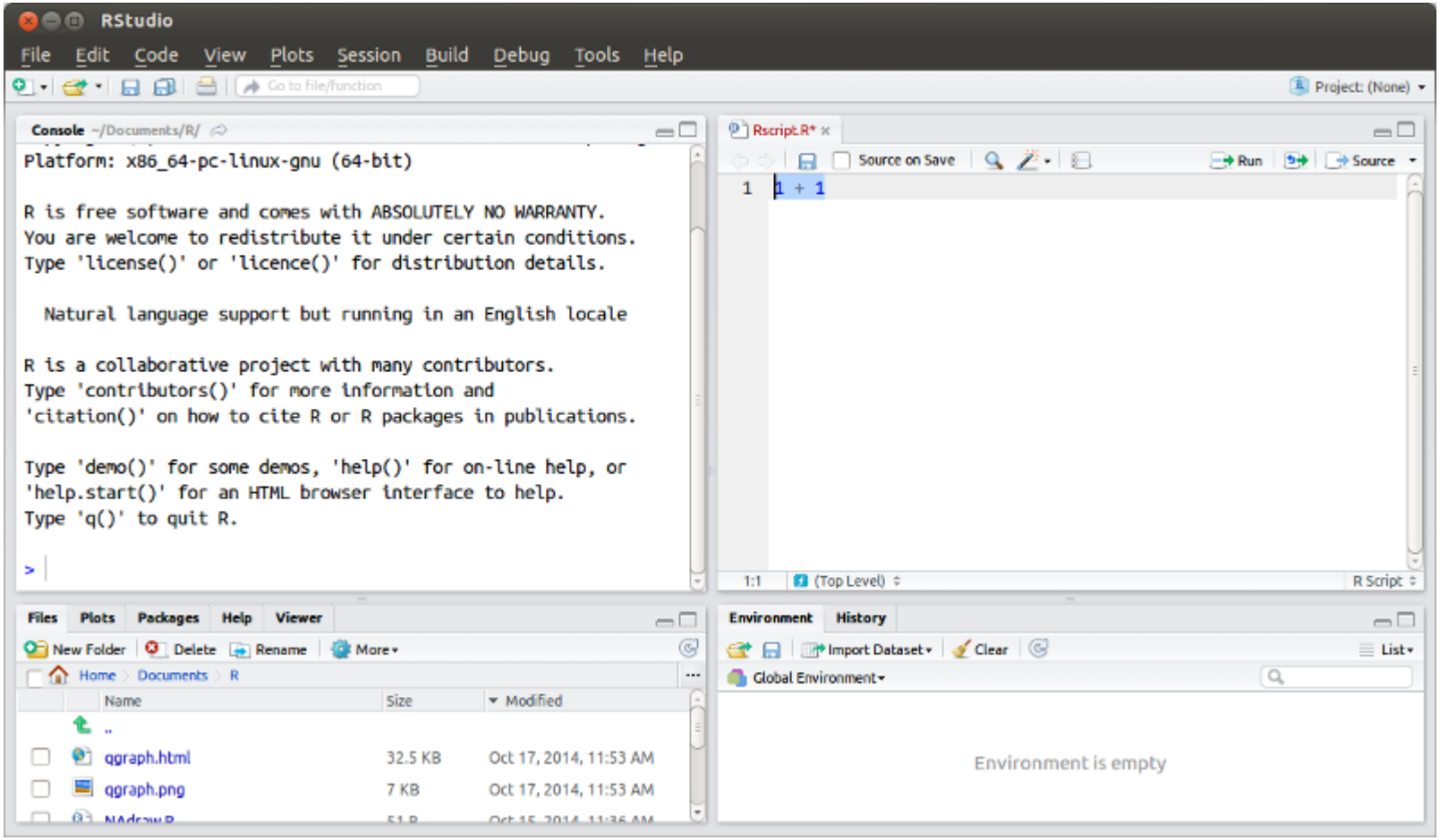


- Save scripts with the extension `.R`.
- Save your scripts often!
    - control+S

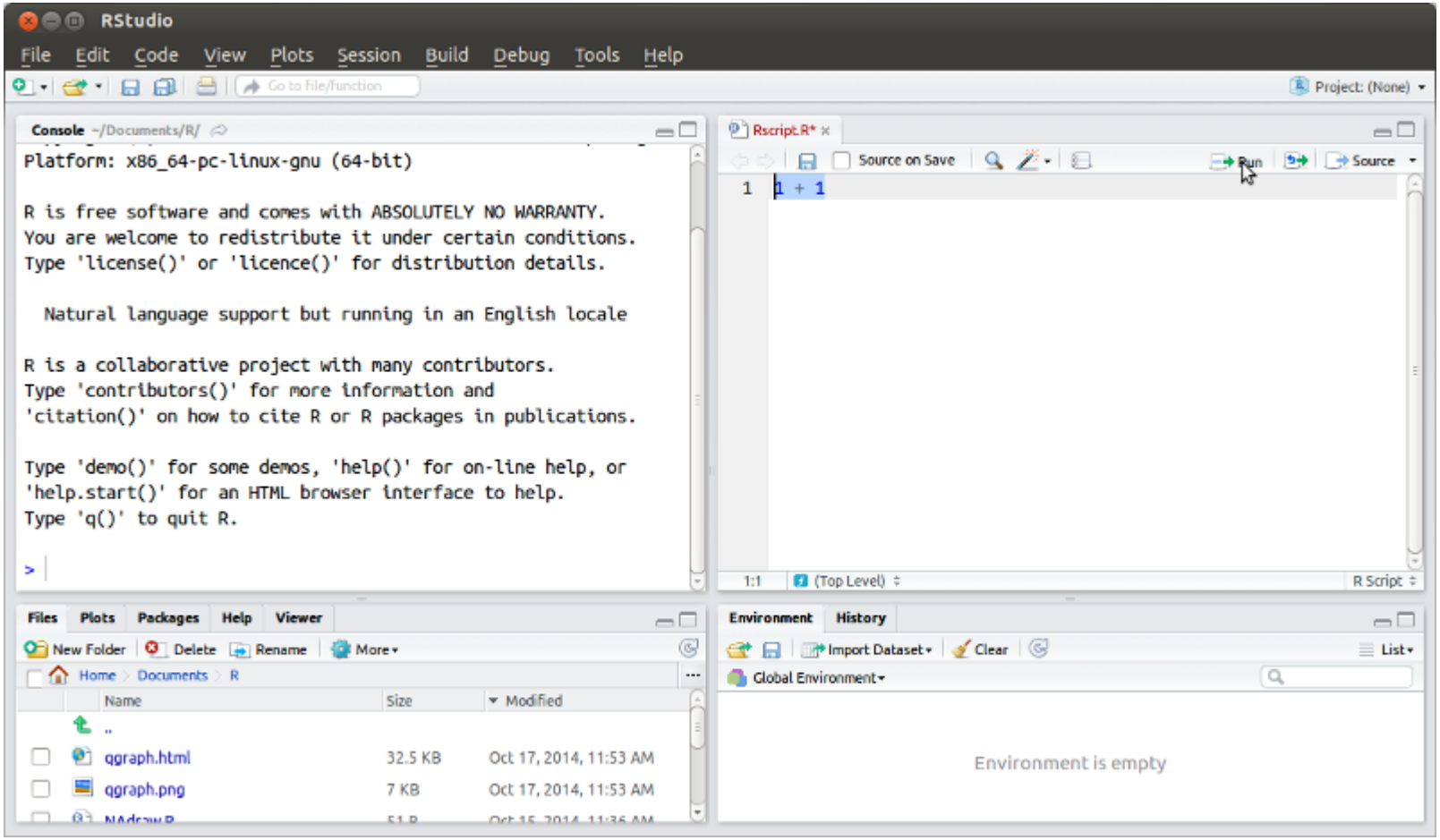
Type **all** your commands into the script:



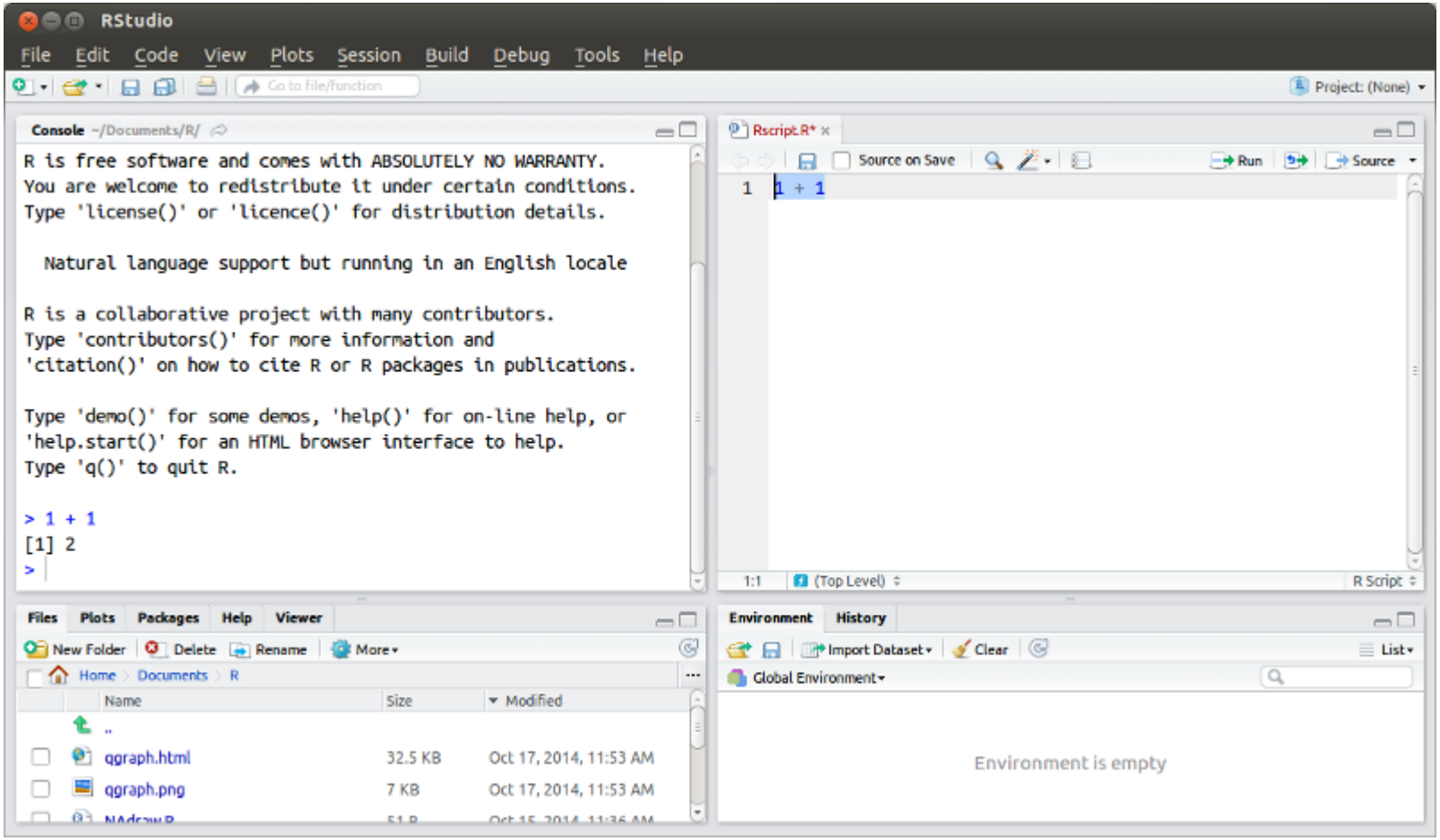
Select the commands you want to evaluate:

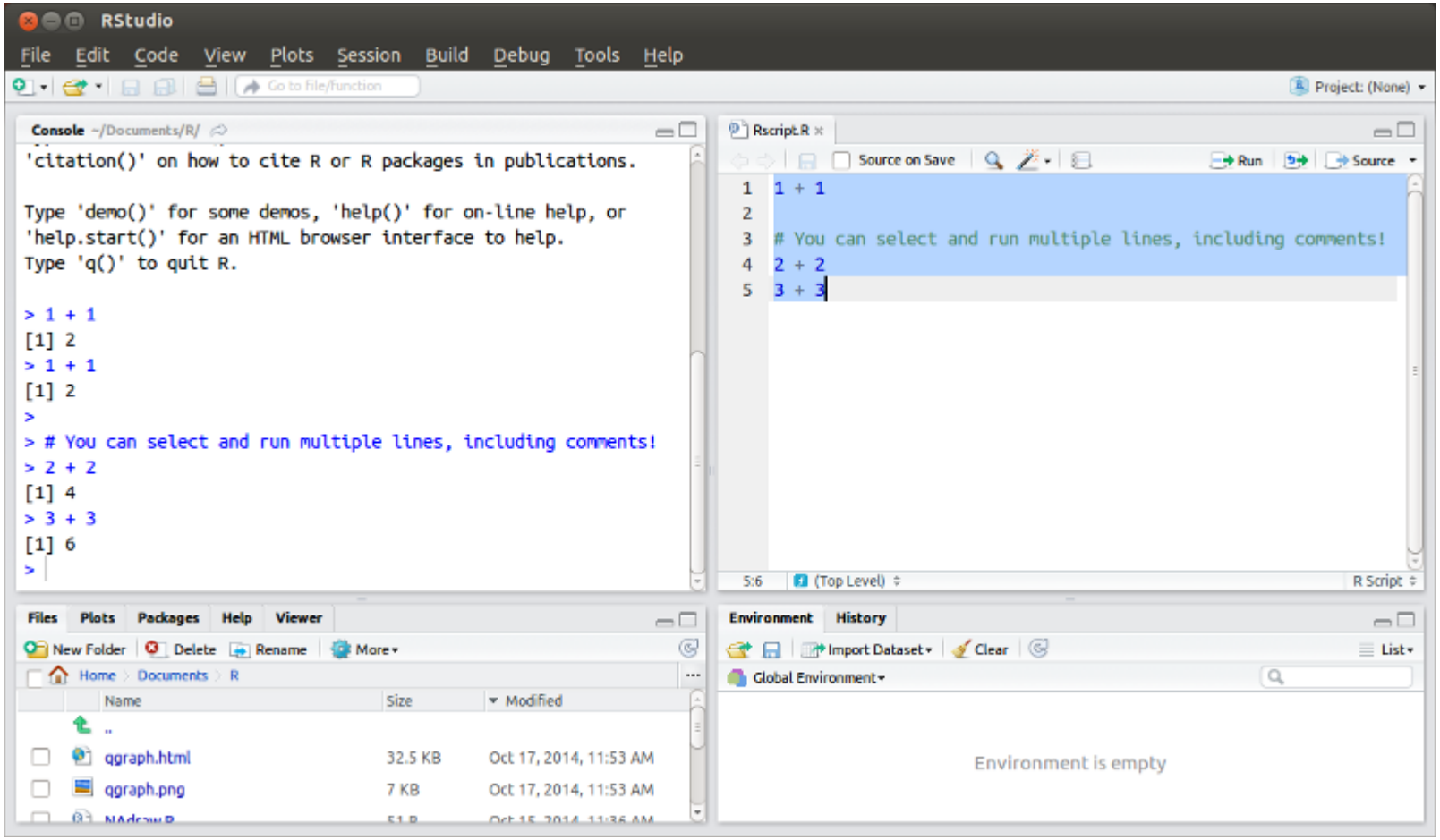


Press the run button or control + enter on your keyboard (command + enter for Mac users)



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




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

Hermione Question

What happens if I only change stuff in the Source panel?

Use R as calculator

```
1 + 1
```

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

```
(10 * 20) / 100
```

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

```
exp(1.5 * (2.1 - 1.8)) / (1 + exp(1.5 * (2.1 - 1.8)))
```

```
## [1] 0.6106392
```

Objects

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
```

You can overwrite the object `a` with a new value:

```
a <- 10
a
```

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

You can overwrite the object `a` with a new value:

```
a <- 10
a
```

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

You can even overwrite `a` using the old value of `a`:

```
a <- a + 1
a
```

```
## [1] 11
```

You can overwrite the object `a` with a new value:

```
a <- 10
a
```

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

You can even overwrite `a` using the old value of `a`:

```
a <- a + 1
a
```

```
## [1] 11
```

Hermione question

What happens if I now type `a` again?

Or assign other objects:

```
b <- a + 10
a + b
```

```
## [1] 32
```

Objects can have any name you want:

```
anyNameYouWant <- 10
anyNameYouWant
```

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

- 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

Object Modes

Object Modes

- There are different types of objects in R:
  - `numeric`
  - `character`
  - `logical`
- There are called *modes*
- To request the mode of object `x` use `mode(x)`

The numeric mode stores numbers:

```
numericObject <- 115
mode(numericObject)
```

```
## [1] "numeric"
```

Numbers

The `numeric` mode stores numbers:

```
numericObject <- 115
mode(numericObject)
```

```
## [1] "numeric"
```

You can use numbers in mathematical expressions:

```
numericObject + 10
```

```
## [1] 125
```

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'
mode(characterObject)
```

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

```
characterObject2 <- "this is also a string"
mode(characterObject2)
```

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

Strings

You can **not** do mathematical expressions with character strings, even when the string looks like a number!

```
a <- '1' # Make a a character
a # a looks like a number
```

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

```
mode(a) # But is not a number!
```

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

Hermione Question

What happens now if I do `a + 1`?

Logicals

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

```
logicalObject <- TRUE
mode(logicalObject)
```

```
## [1] "logical"
```

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

Logicals

You obtain a `logical` result if you ask R a comparison test:

```
1 == 1 # Is 1 equal to 1?
```

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

```
1 < 2 # Is 1 smaller than 2?
```

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

```
1 >= 2 # Is 1 greater or equal to 2?
```

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

Transforming modes

I can use functions named `as.` followed by the name of the mode to transform objects into a different mode:

```
"1" + 1
```

```
## Error in "1" + 1: non-numeric argument to binary operator
```

```
as.numeric("1") + 1
```

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

Vectors

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
```

Vectors

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

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

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

To create a sequence use a colon:

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Indexing

- *indexing* means you want a subset, or a single element, of a vector
- You can index a vector in R by using square brackets `[` and `]`
    - Follow the name of a vector with square brackets
    - Put in the square brackets:
        - A vector containing numbers of the elements you wish to keep

```r
a <- c(2,4,6,8,10)
a
```

```
## [1]  2  4  6  8 10
```

```r
a[5] # Get the fifth element
```

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

Hermione Question

What do you think happens if I do `a[c(1,5)]` ?

All elements of `a` that are lower than 5:

```
a[a < 5]
```

```
## [1] 2 4
```

```r
age <- c(22,20,28,25,32,21,25)
gender <- c('male','male','female','female','male','female', 'male')

# Age of males:
age[gender == "male"]
```

```
## [1] 22 20 32 25
```

```r
# Age of females:
age[gender == "female"]
```

```
## [1] 28 25 21
```

Function

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

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

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

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

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

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

Packages

Packages

- Packages are extensions contributed to R containing extra functions
  - Stored on the Comprehensive R Archive Network (CRAN)
  - Over 5000 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()`

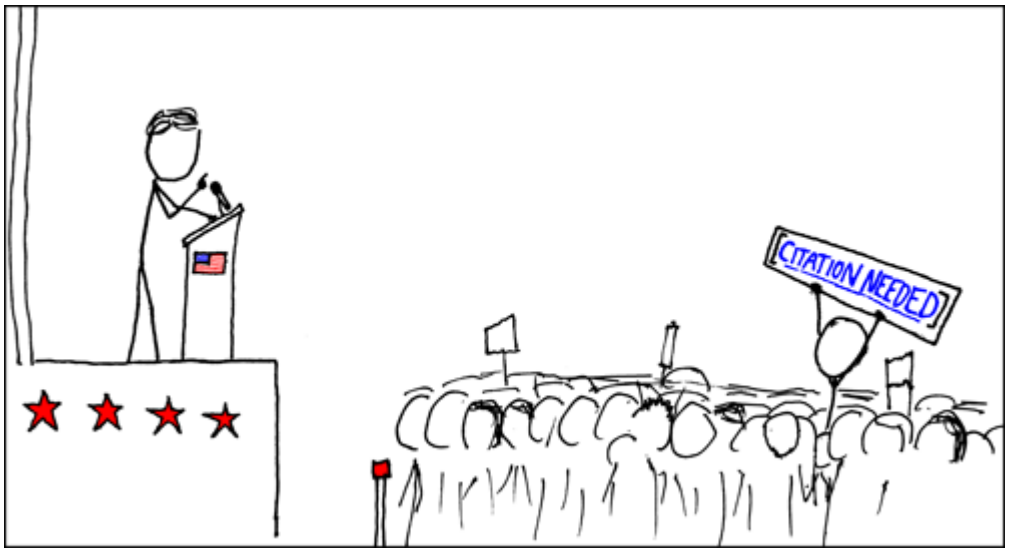
Packages

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

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

Packages give you new functions to do *anything*

```
library("RXKCD")
getXKCD(285)
```



Many state of the art analyses have been implemented as R packages:

Package	Description
`lavaan`	Structural Equation Modeling
`lme4`	Multi-level generalized linear models
`glmnet`	Regularized generalized linear models
`depmixS4`	Latent (hidden) Markov models
`huge`	Graphical model selection
`BayesFactor`	Bayesian analyses
`ltm`	Item-response theory

Data Structures

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
```

Order of indexing matrices

RC -> Roman Catholic

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
```

Lists

- The `list()` function can be used to create a *list*
- This is an object that can contain other objects
- To index a list use double square brackets, or a dollar sign (see example on next slides).

Lists

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

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

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

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

```
# 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 column, 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

```
# 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
```

Correlation matrix

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

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

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

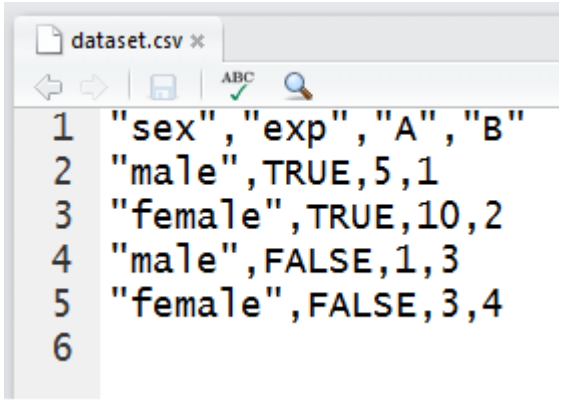
```
# Correlation matrix
cor(df1[,c('A','B')])
```

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

Reading and writing datasets in R

Reading and writing datasets in R

- The best way to store data in R is to use plain text files
  - Often with the extension `.txt` or `.csv`
  - Can be opened in RStudio!
  - Can be exported from Excel!
- These can be read with:
  - `read.table`, `read.csv` and some others
  - These are the same function with different defaults!
- Look at what the data looks like!
- Other file types can be read. Check out `read.spss` in the `foreign` package!



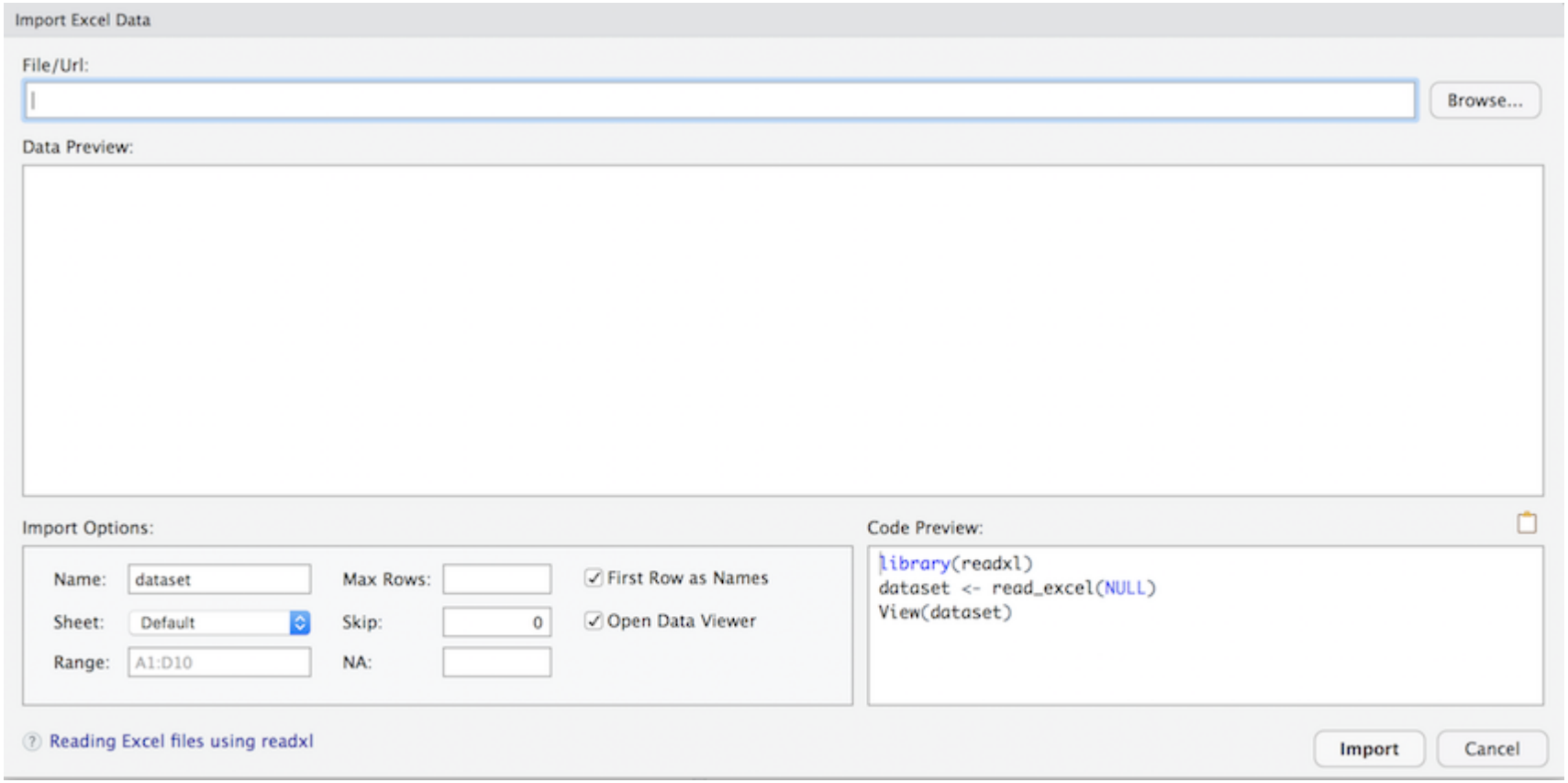

```
file <- file.choose() # Choose the right file
read.table(file, header=TRUE, sep = ",")
```

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

```
read.csv(file)
```

Newest feature!

Import dataset by simply clicking


Import Excel Data
File/Url:
Browse...
Data Preview:
Import Options:
Name:
dataset
Max Rows:
First Row as Names
Sheet:
Default
Skip:
0
Open Data Viewer
Range:
A1:D10
NA:
Code Preview:
library(readxl)
dataset <- read_excel(NULL)
View(dataset)
Reading Excel files using readxl
Import
Cancel

Working Directory

- Reading and writing files in R is done from the **working directory**
- Can be set with `setwd()` or requested with `getwd()`
- Or from the menu: session -> set working directory

Writing files from R

To write dataframes you can read in Excel or other programs, use `write.table`, `write.csv`, etcetera

```
write.table(df1, file = "sometable.txt")
read.table('sometable.txt')
```

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

- Files are stored in your working directory!

Thank you for your attention!

Now we can get to work!