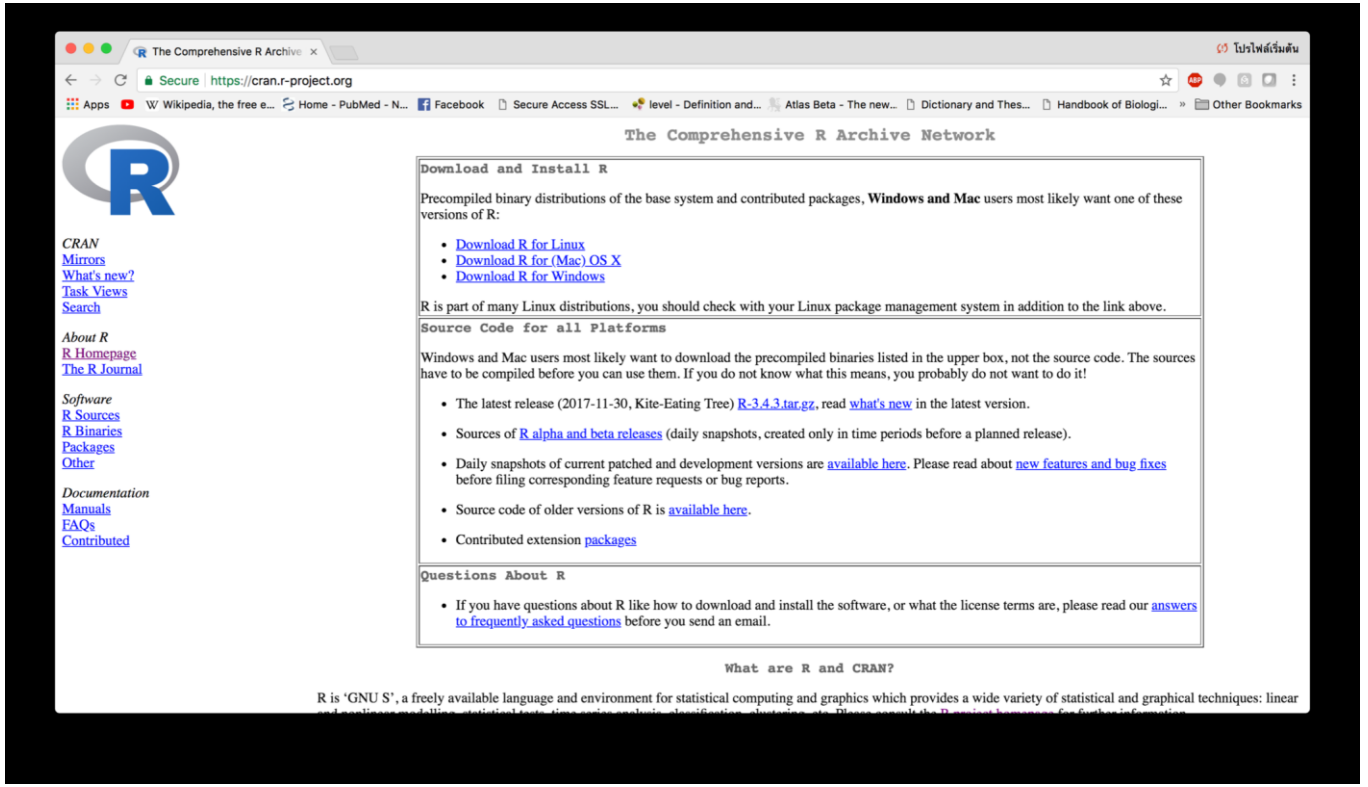
SIRE516

Week1: Introduction to R programming

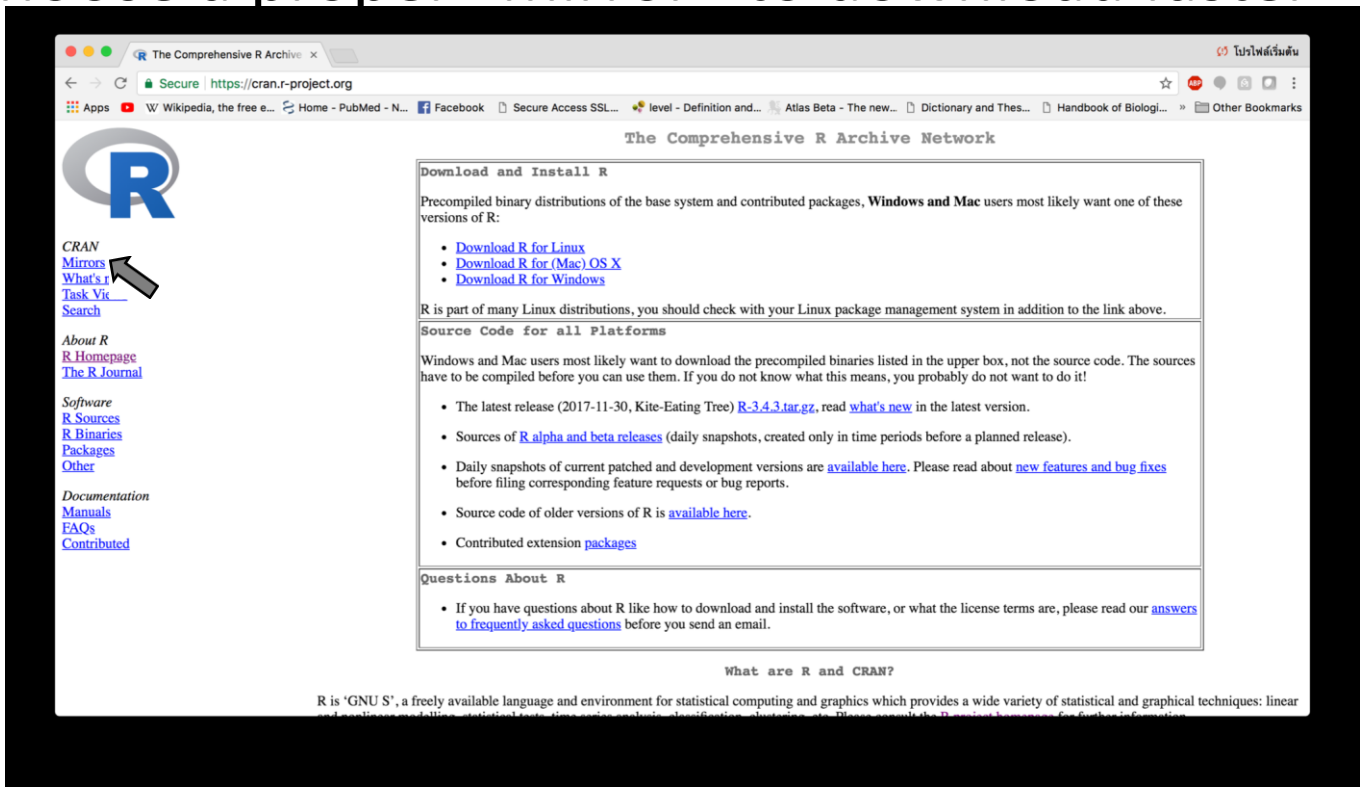
Let's get R

- Download the latest version of R at https://cran.r-project.org/



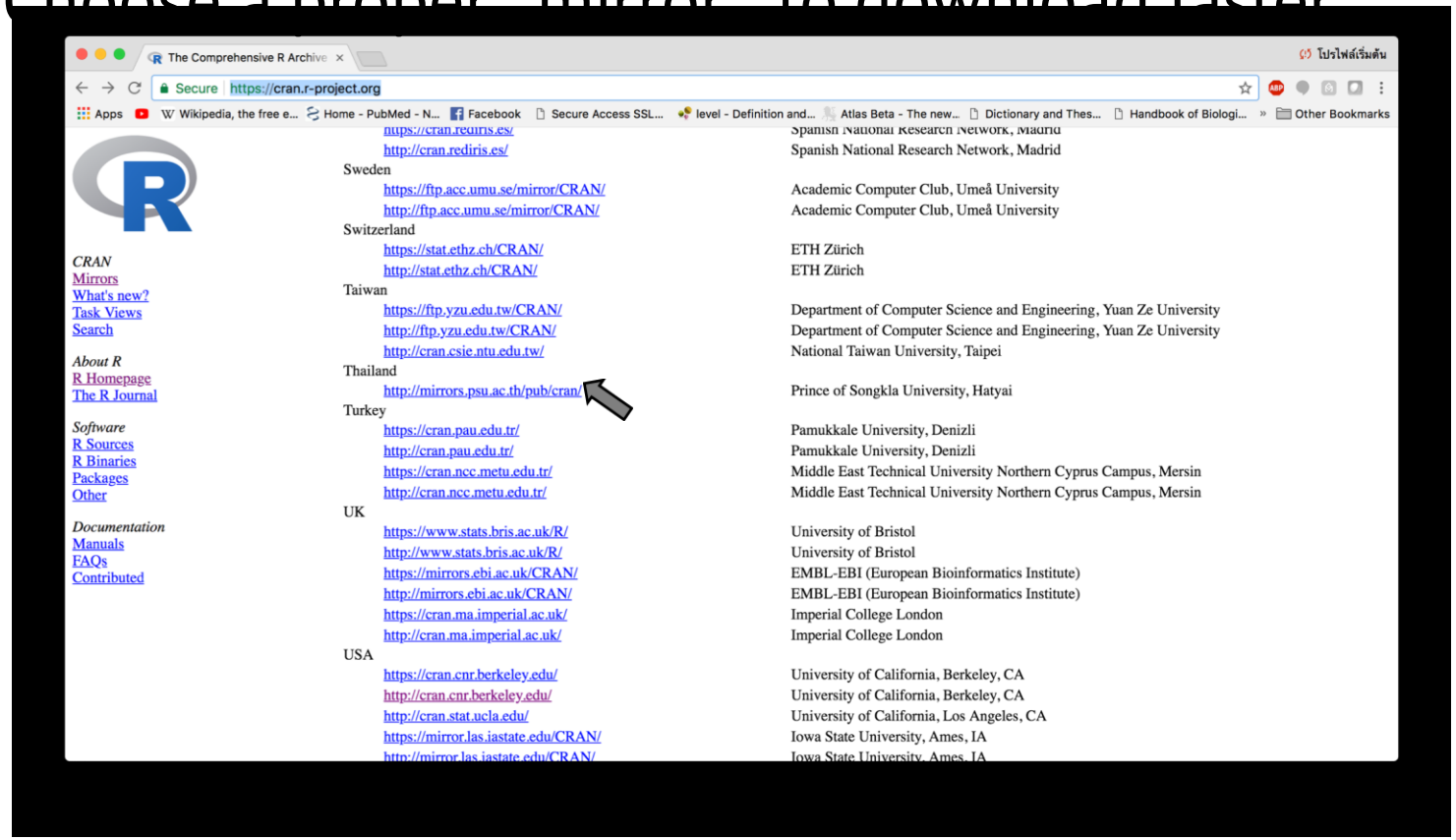
Let’s get R

- Choose a proper ”mirror” to download faster



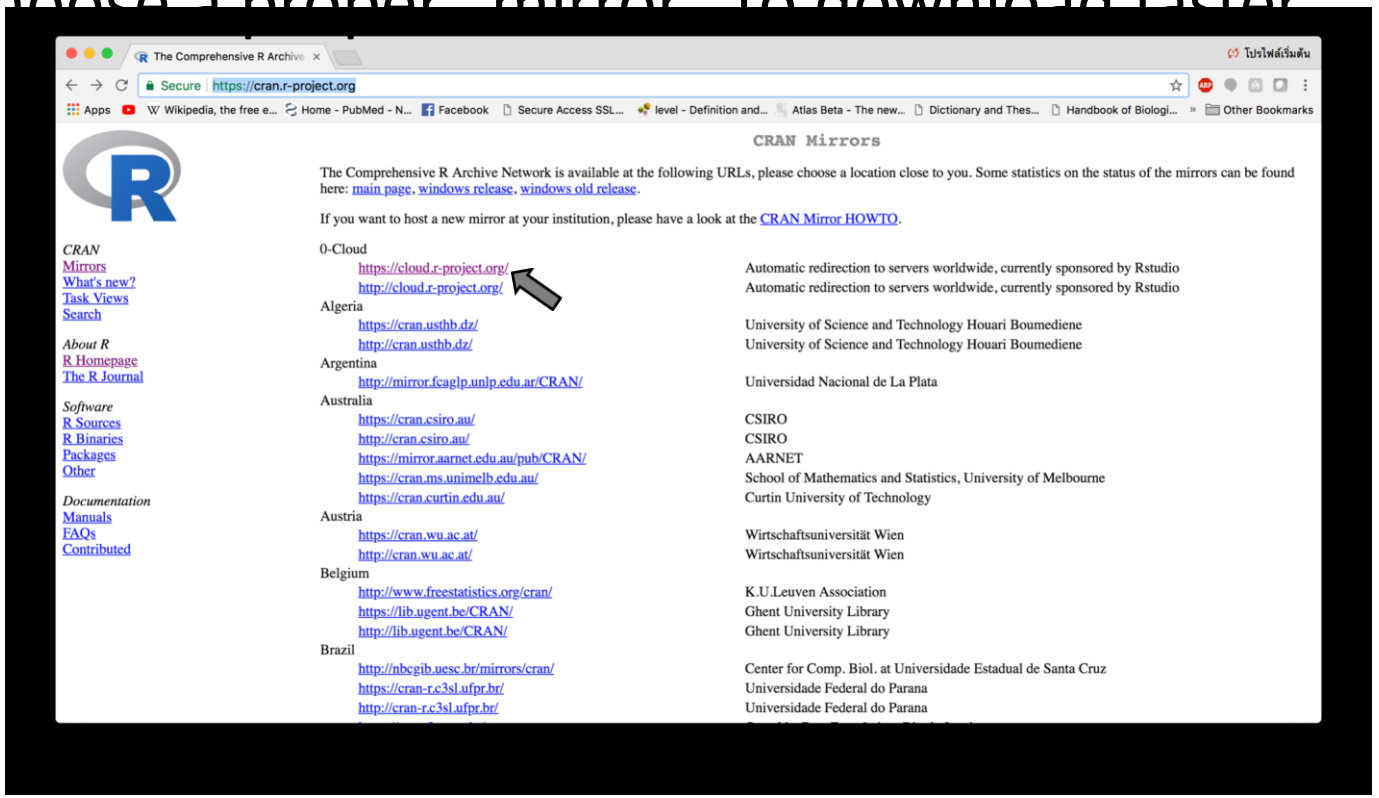
Let’s get R

- Choose a proper ”mirror” to download faster



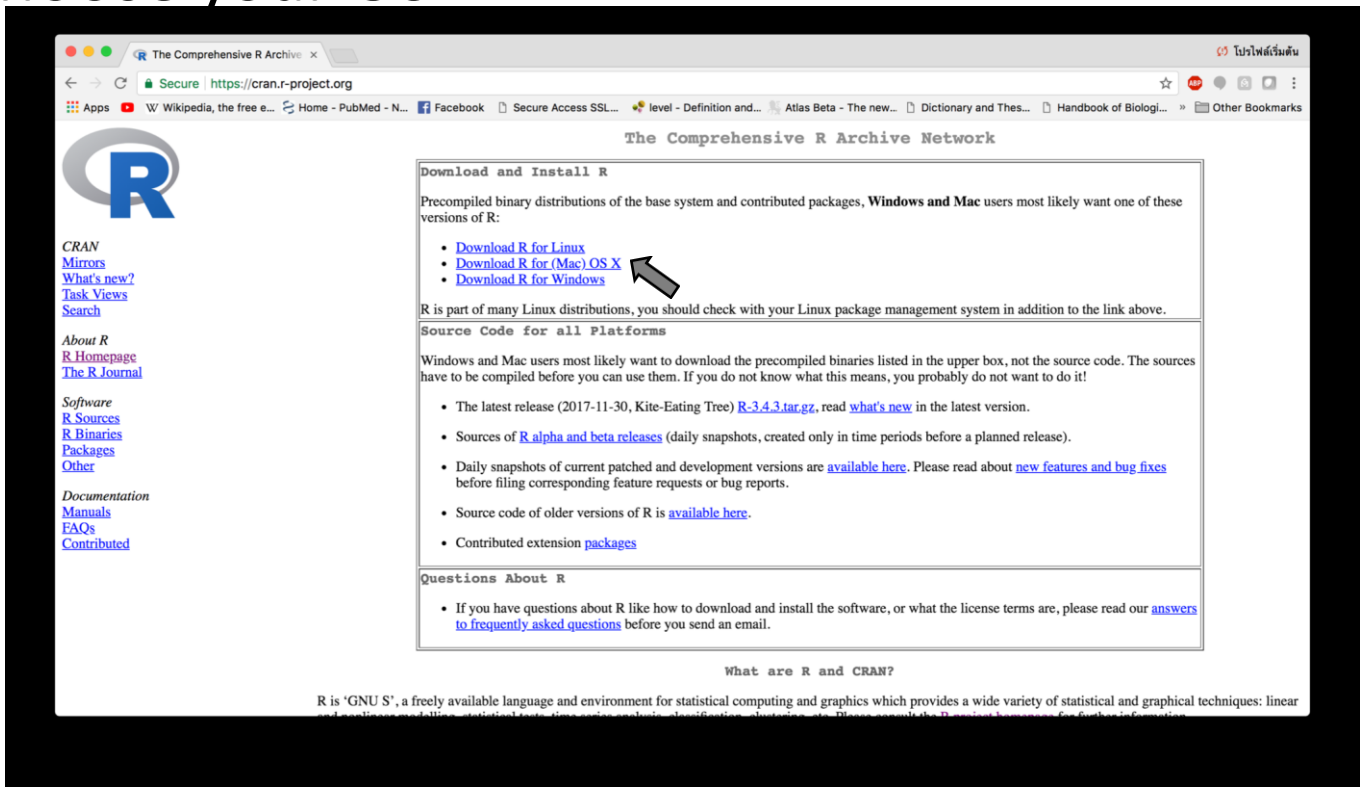
Let’s get R

- Choose a proper ”mirror” to download faster



Let's get R

- Choose your OS



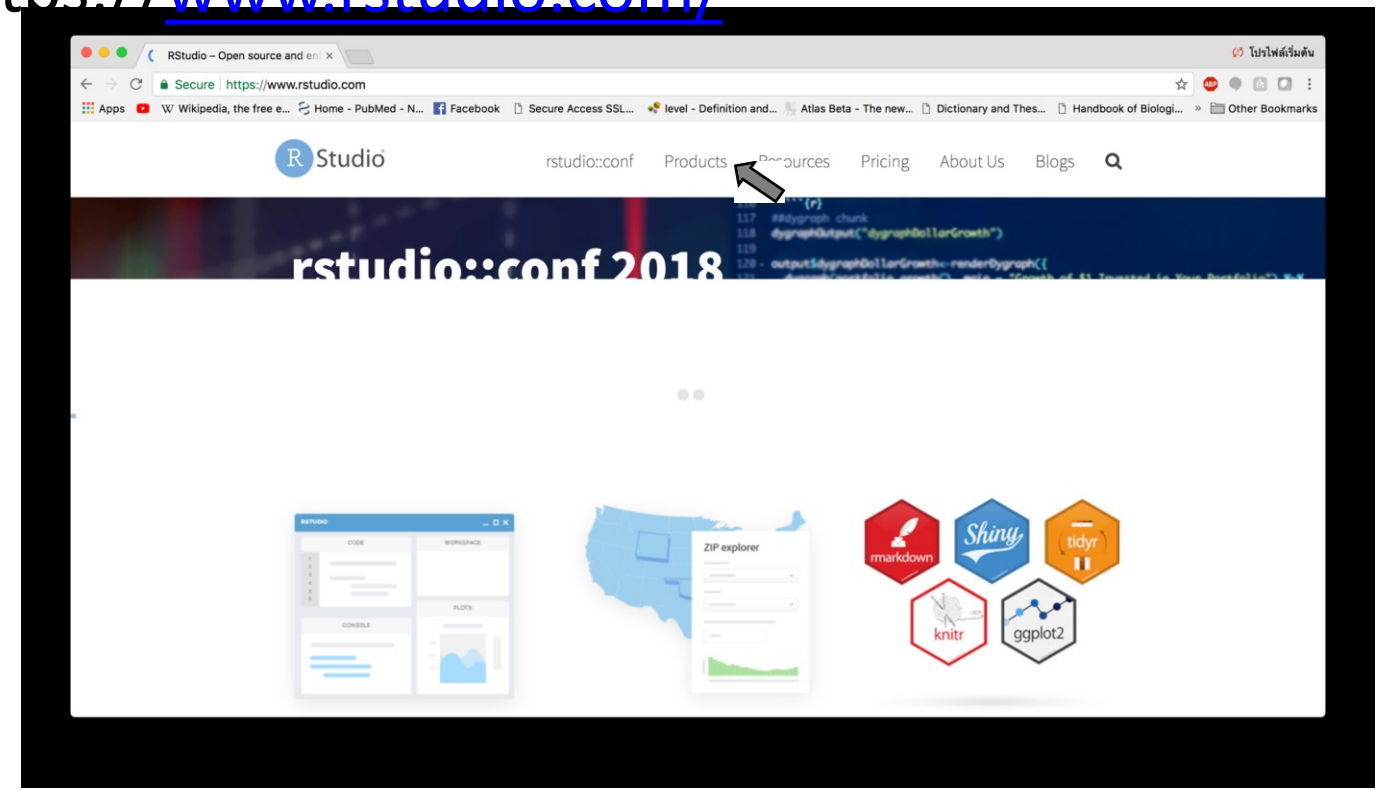
RStudio

- Integrated development environment (IDE) for R
- Organizing your R coding environment
- Facilitate datasets loading
- Facilitate graph and plot resizing and saving

*****ALWAYS install R before Rstudio*****

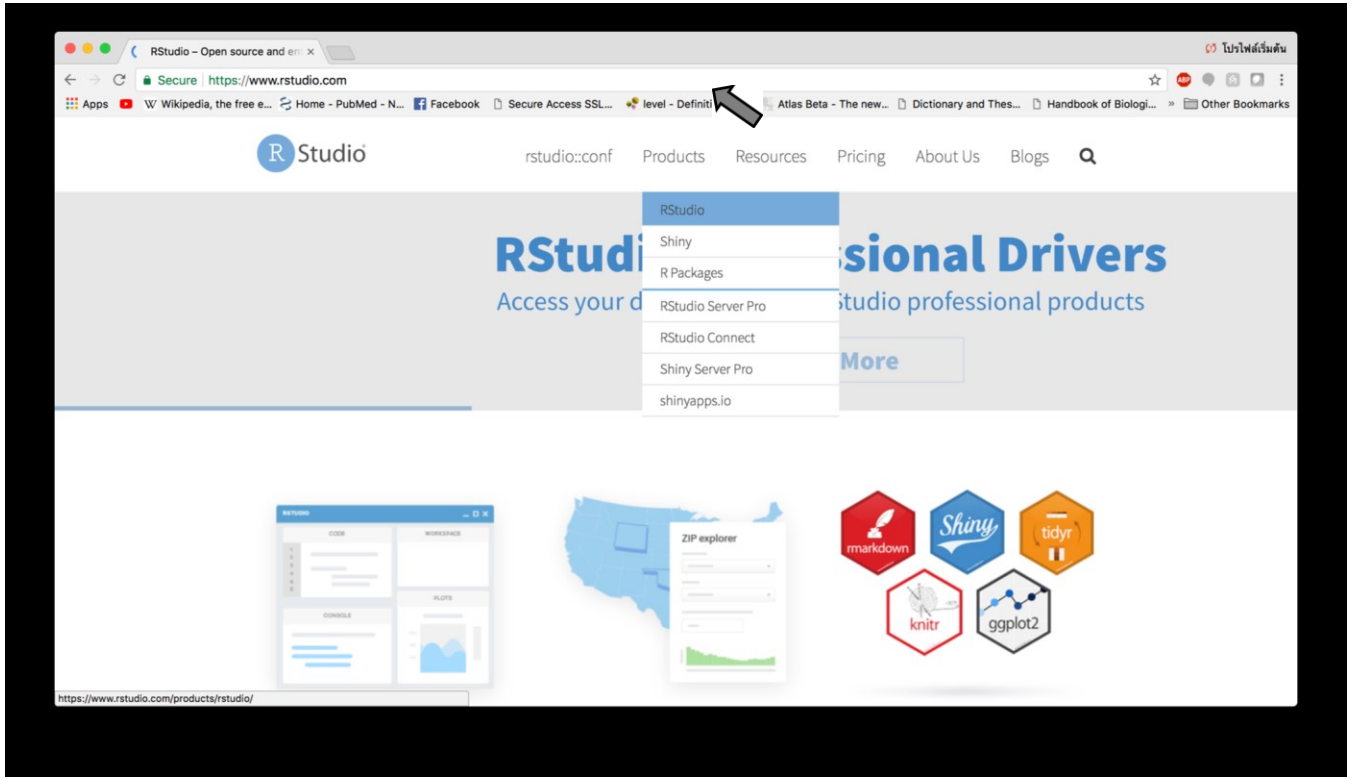
Let’s get RStudio

- Download the latest version of RStudio at https://www.rstudio.com/



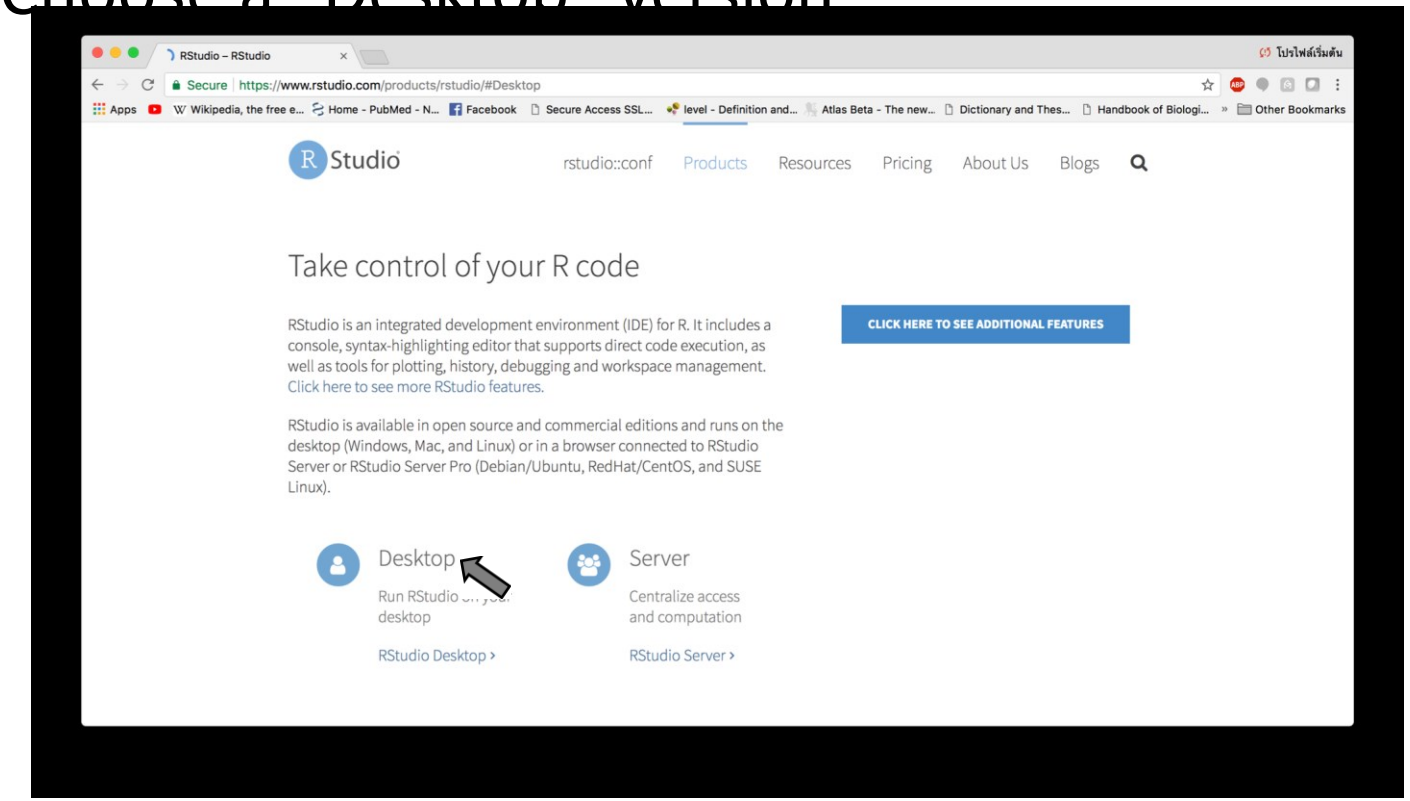
Let’s get RStudio

- Download the latest version of RStudio at https://www.rstudio.com/



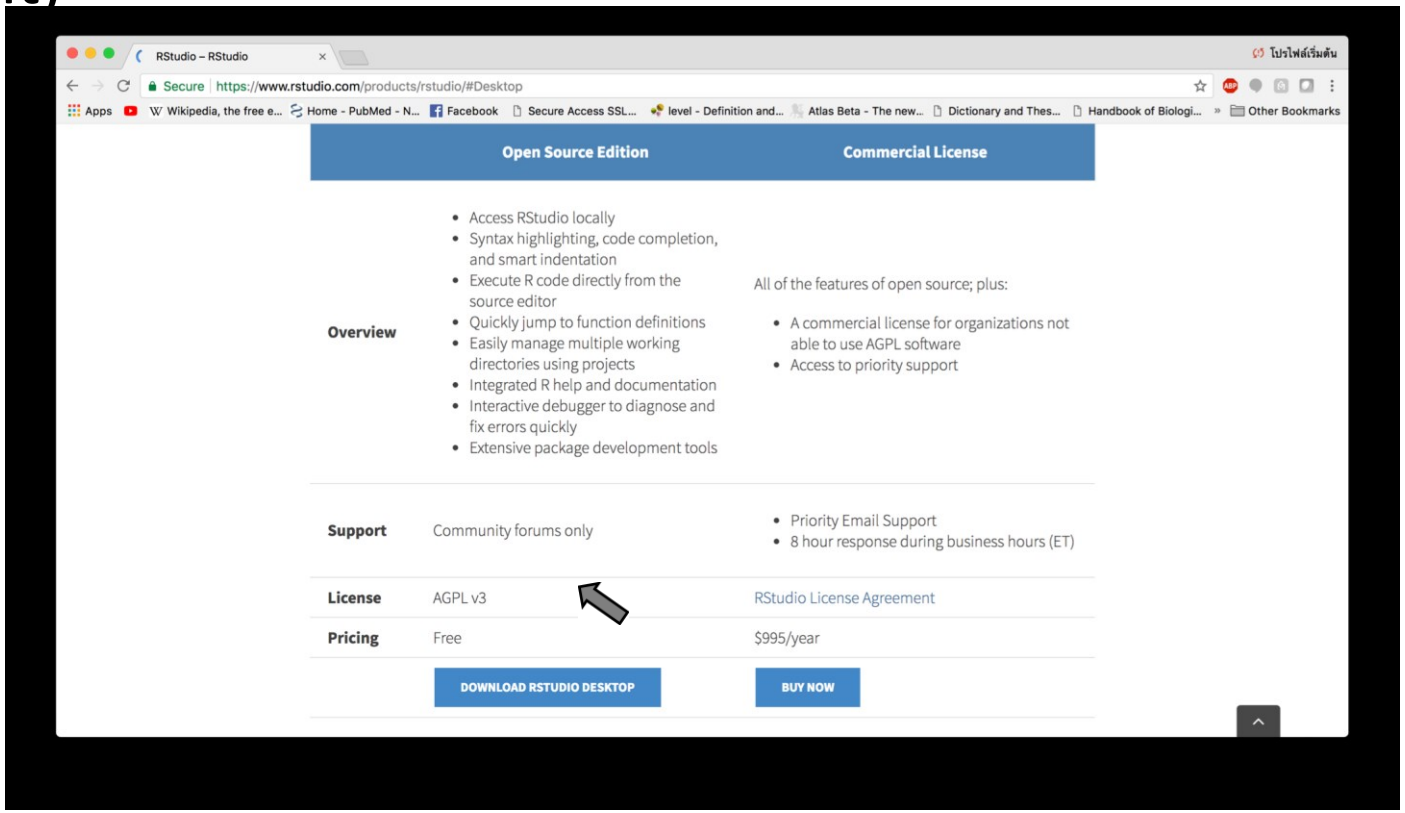
Let’s get RStudio

- Choose a “Desktop” version



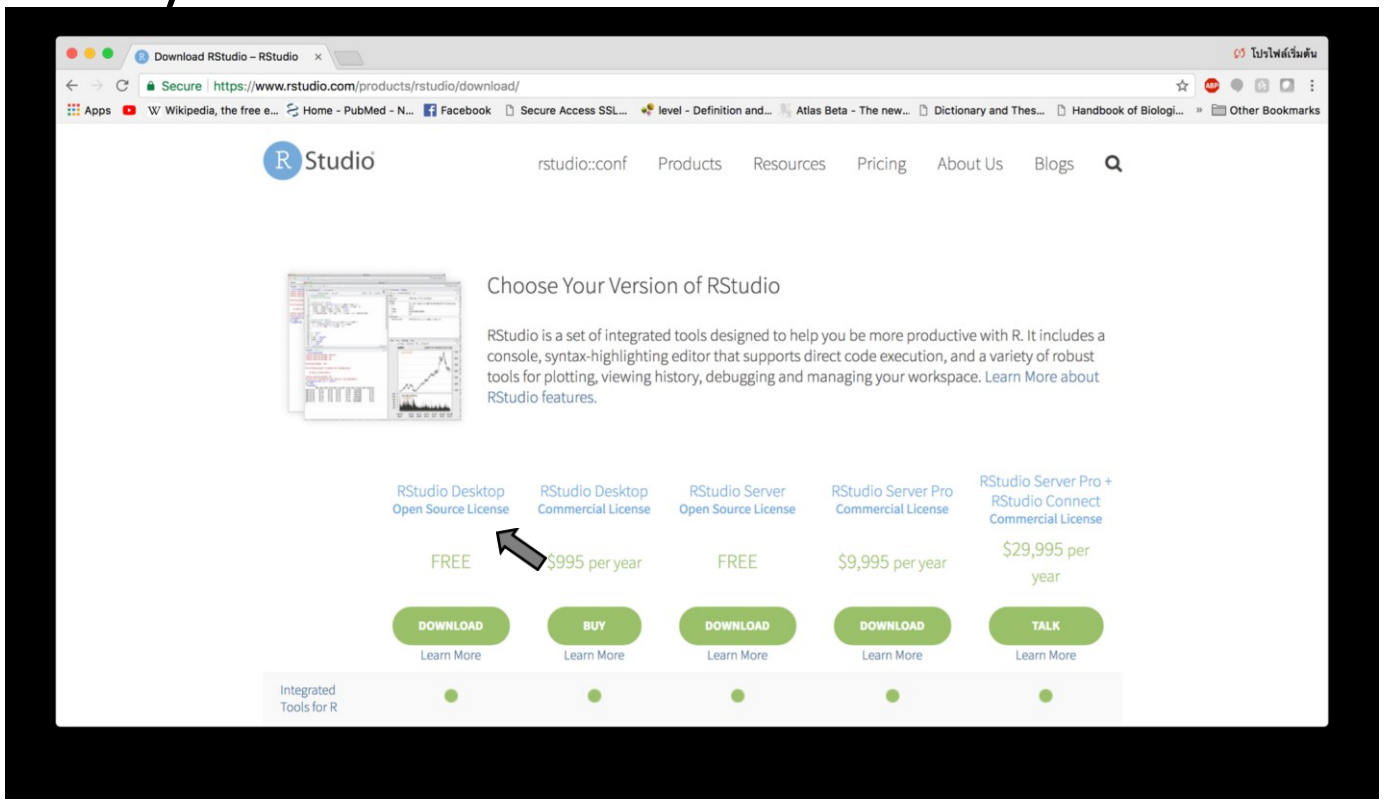
Let’s get RStudio

- Choose a “Open Source” edition (Or buy it if you want)



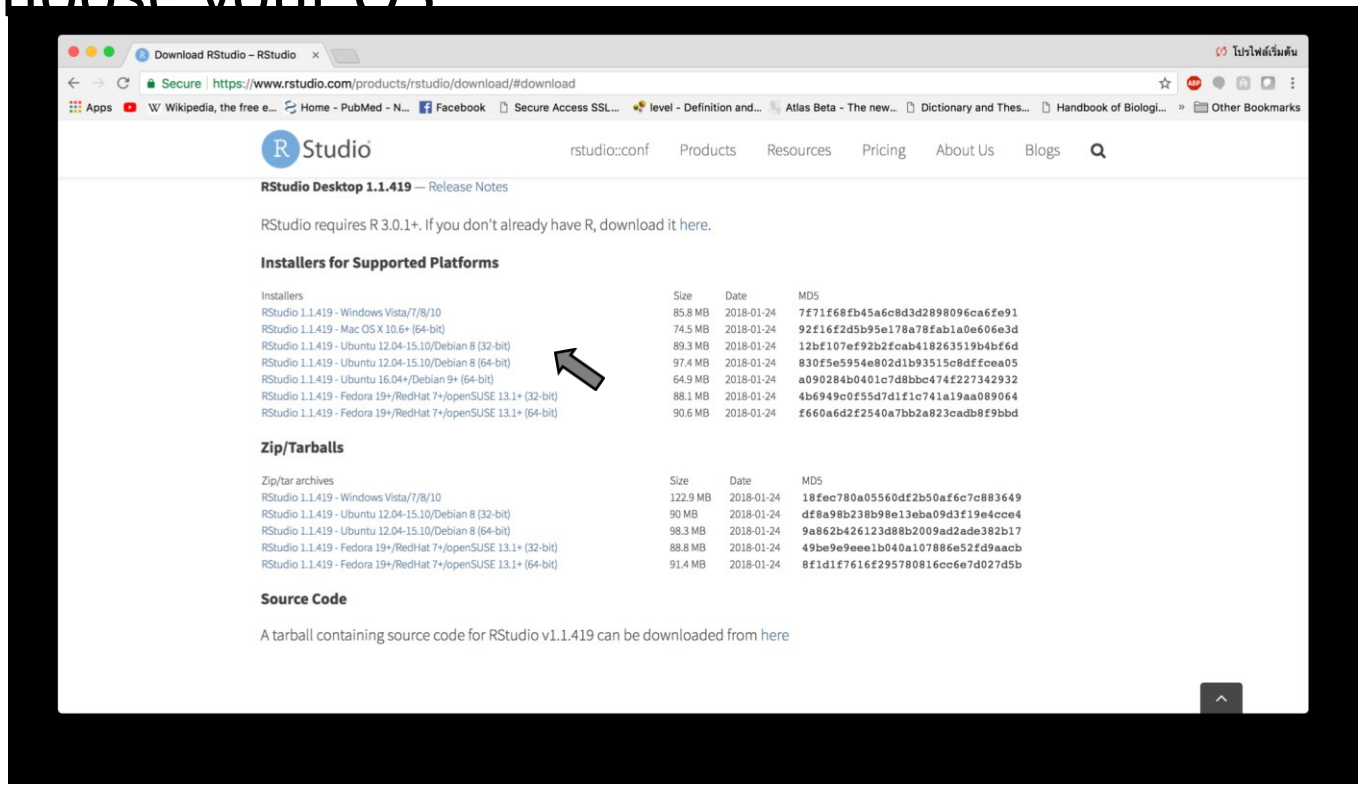
Let’s get RStudio

- Choose a “Open Source” edition (Or buy it if you want)

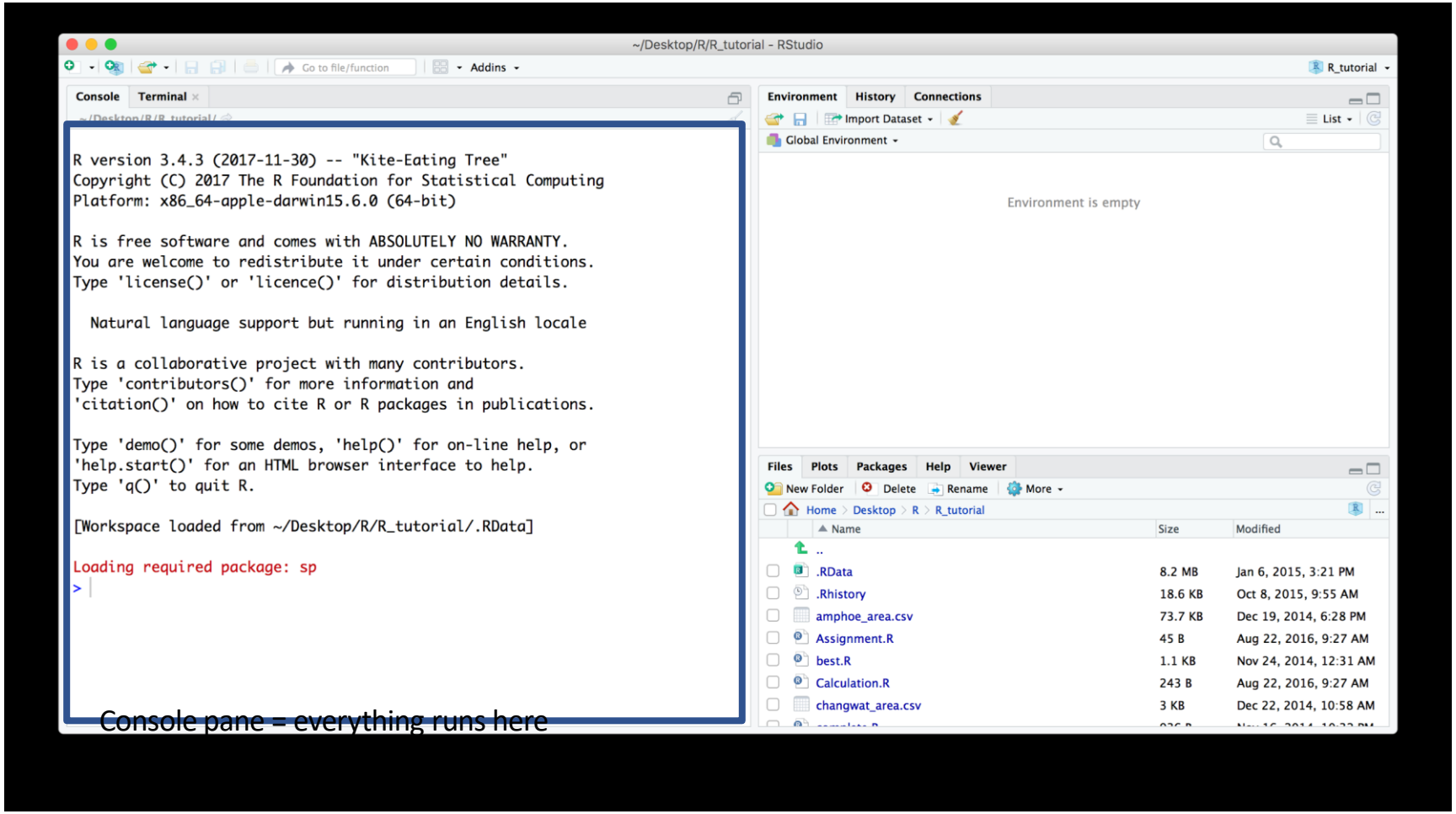


Let’s get RStudio

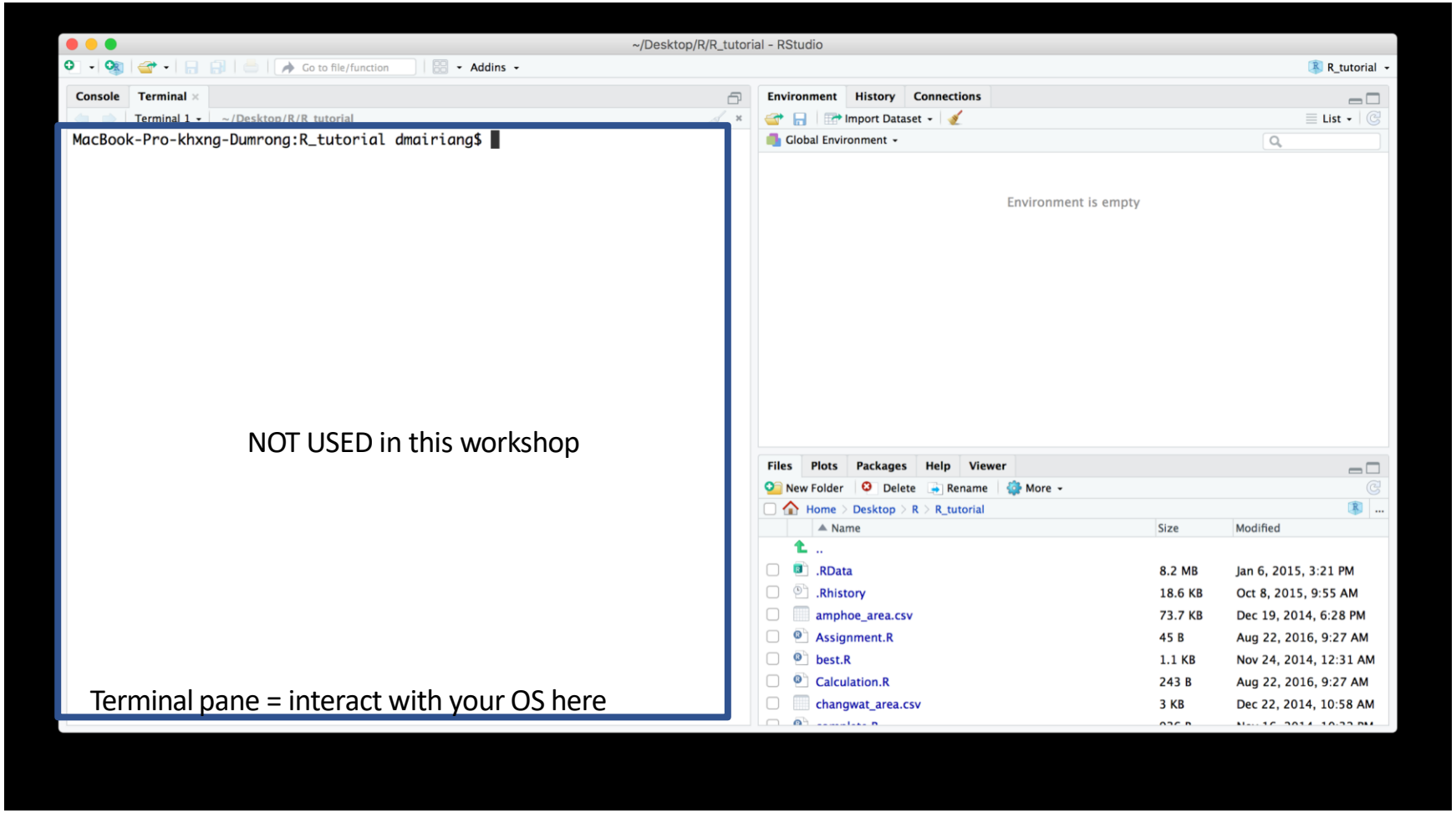
- Choose your OS



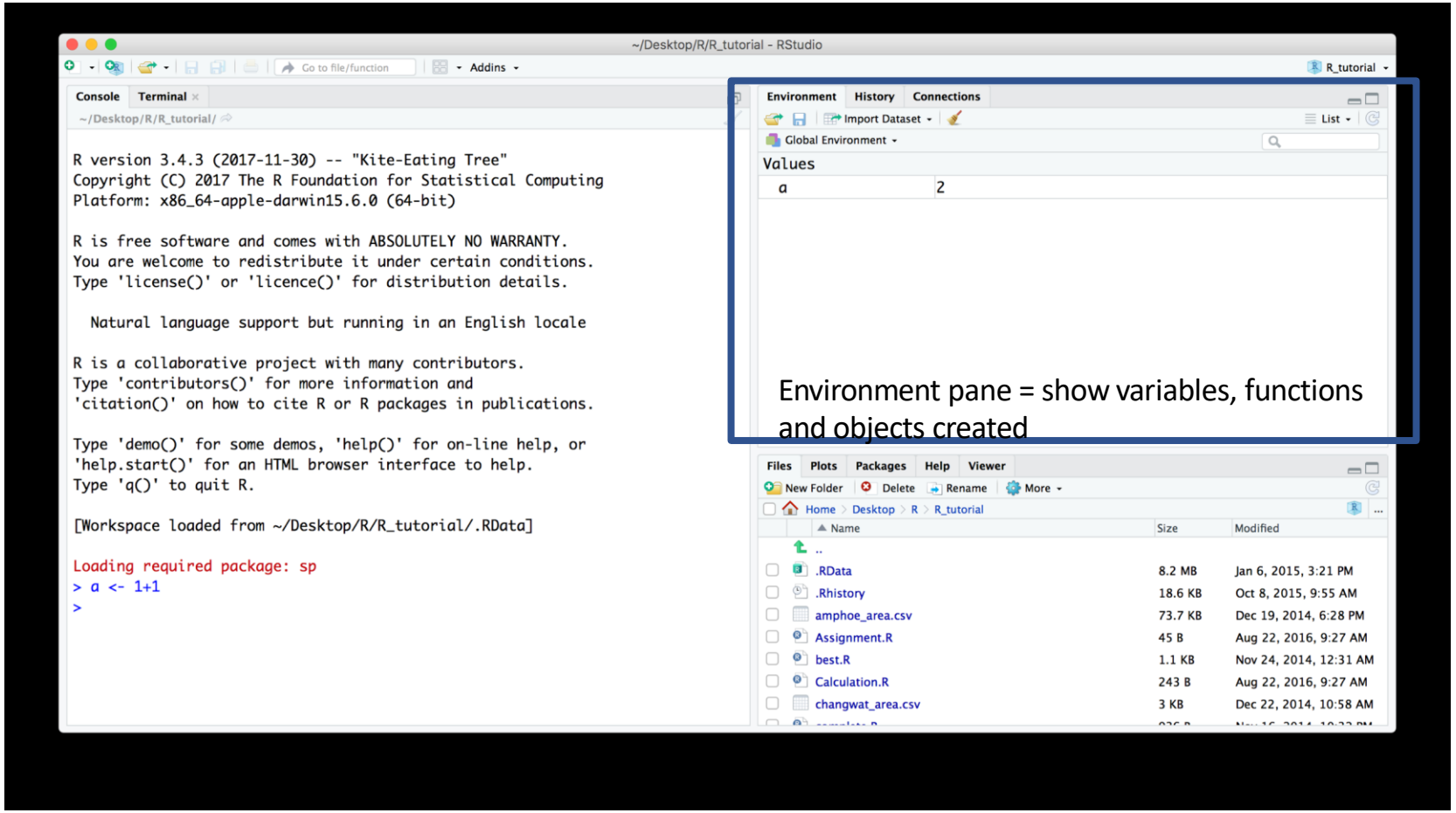
Panes of RStudio



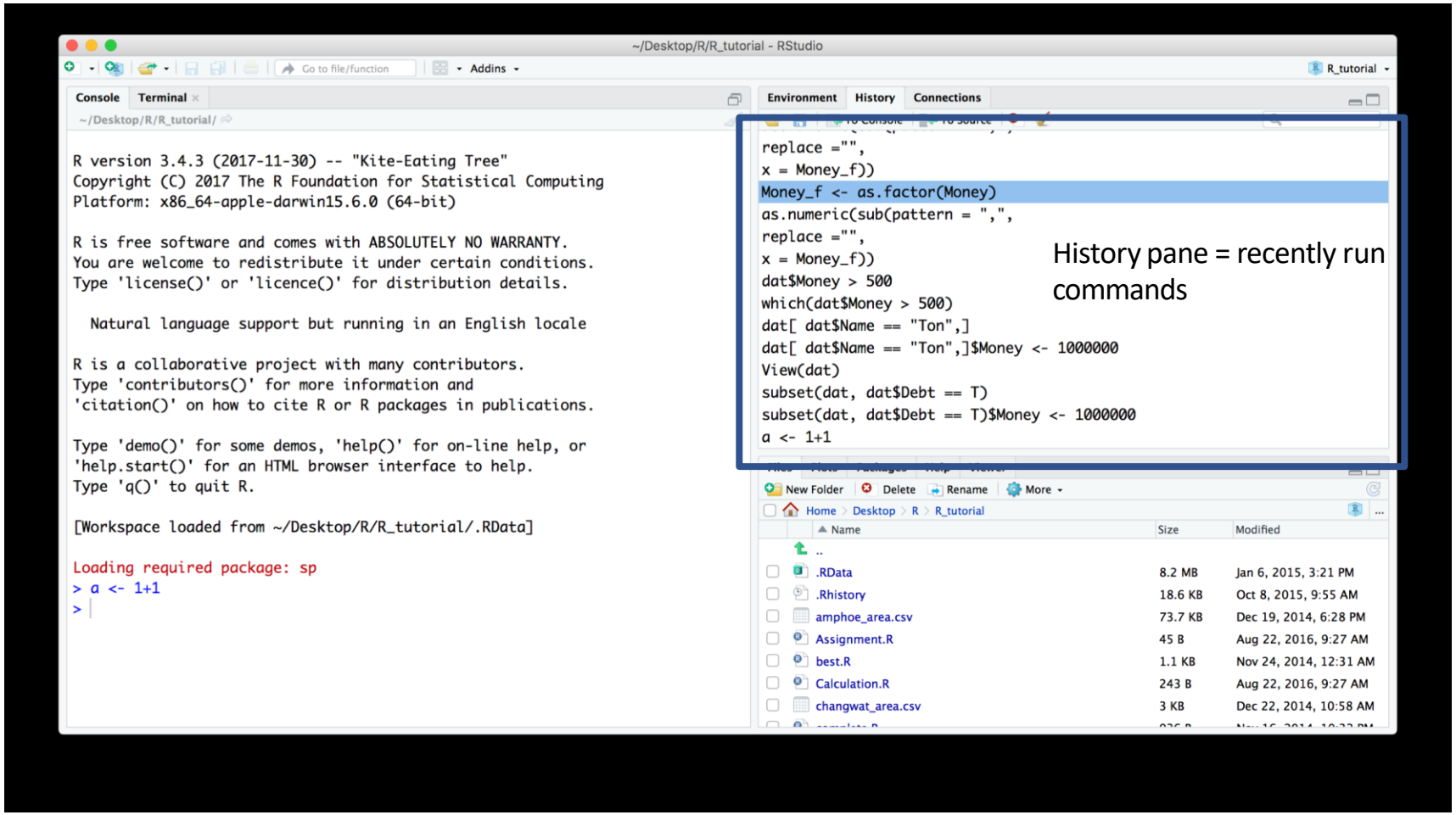
Panes of RStudio



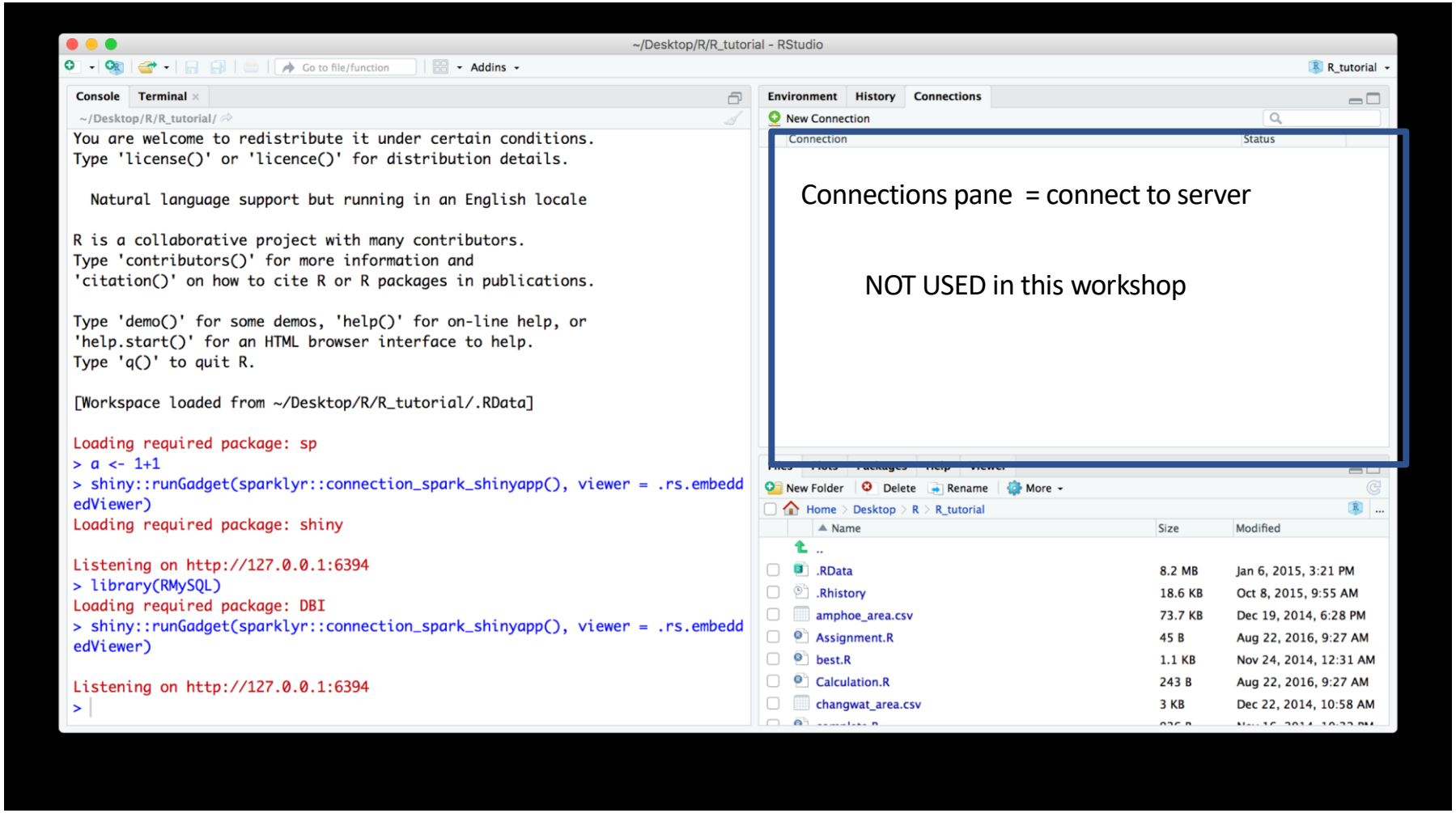
Panes of RStudio



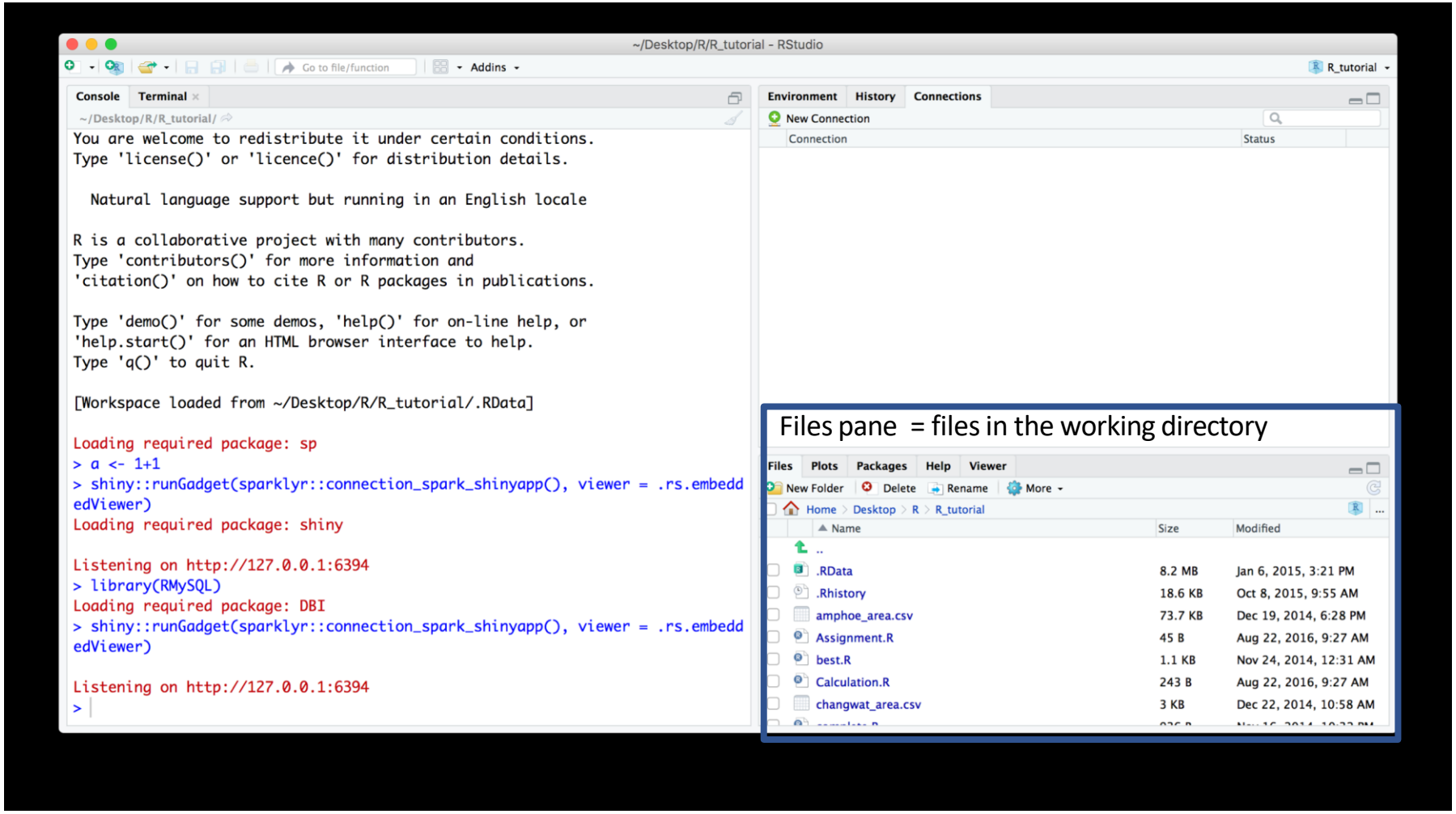
Panes of RStudio



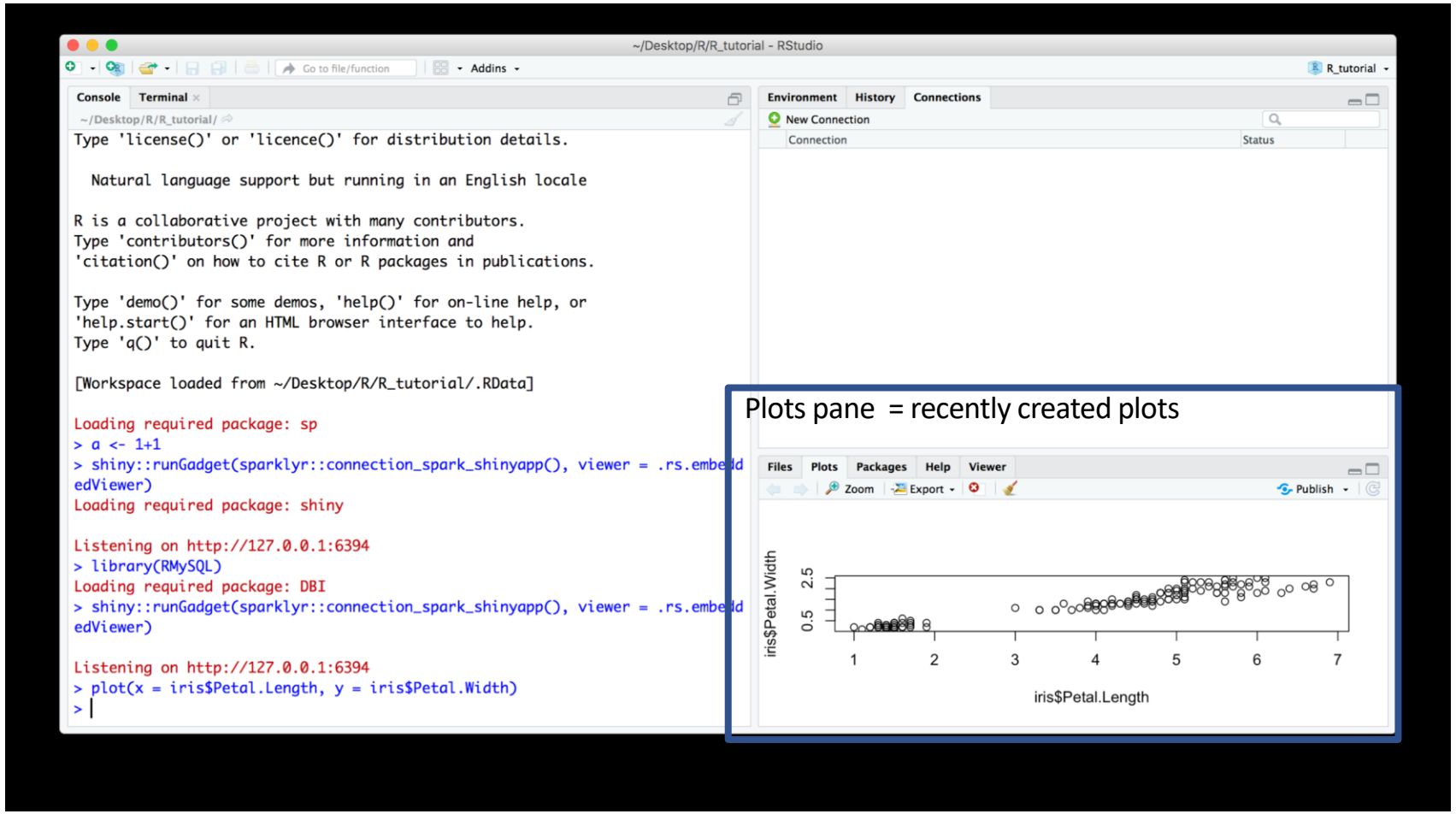
Panes of RStudio



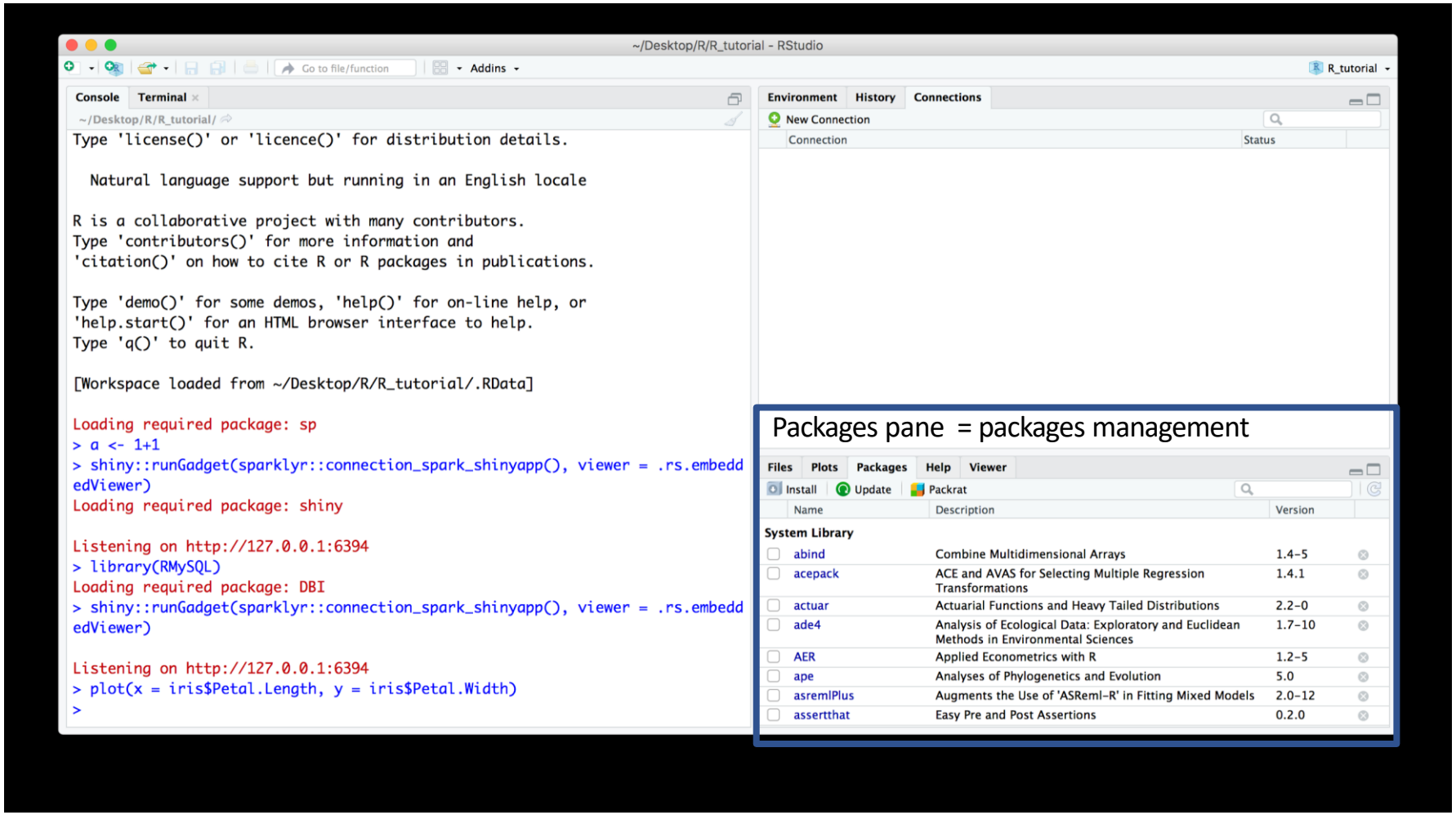
Panes of RStudio



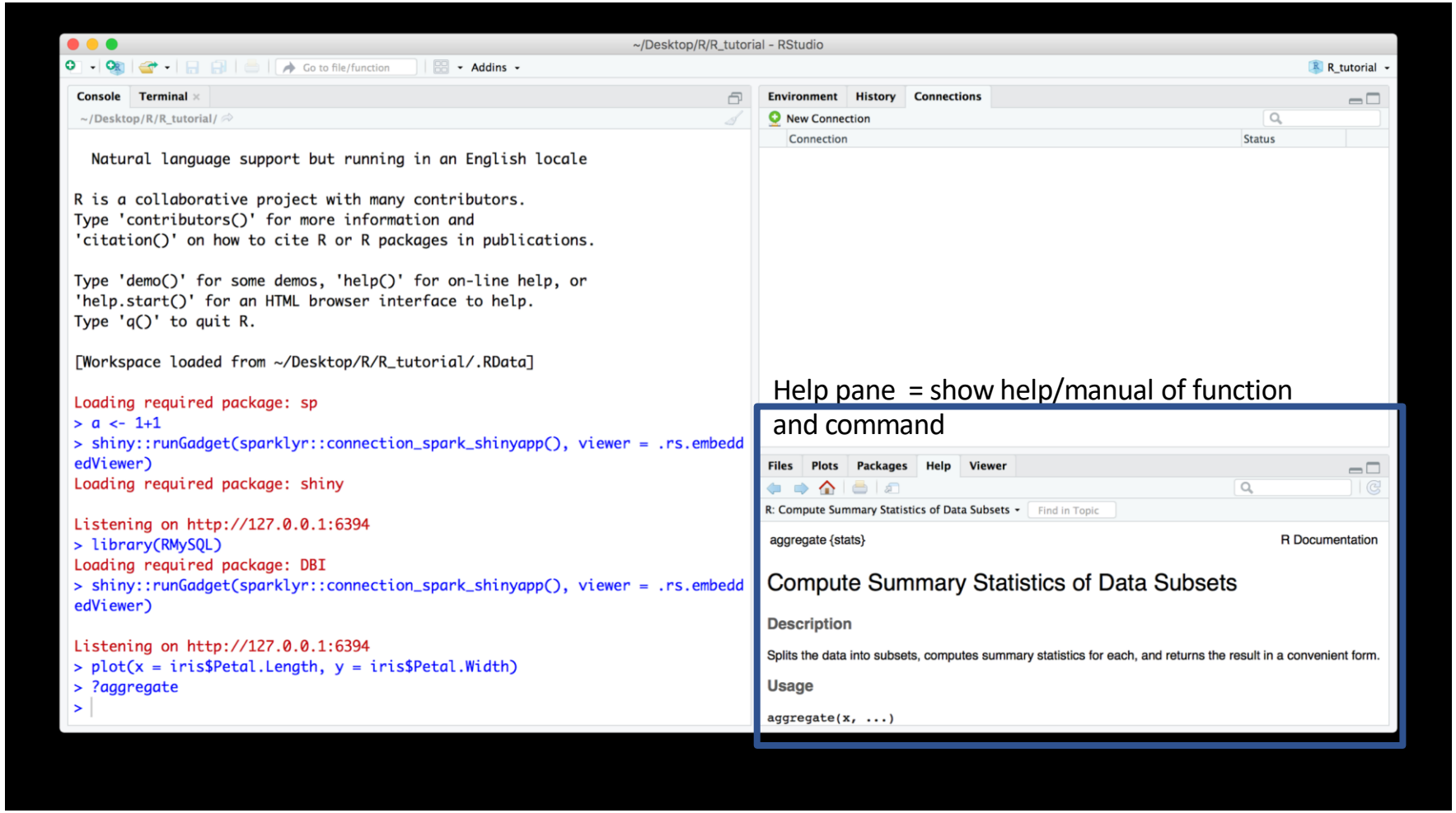
Panes of RStudio



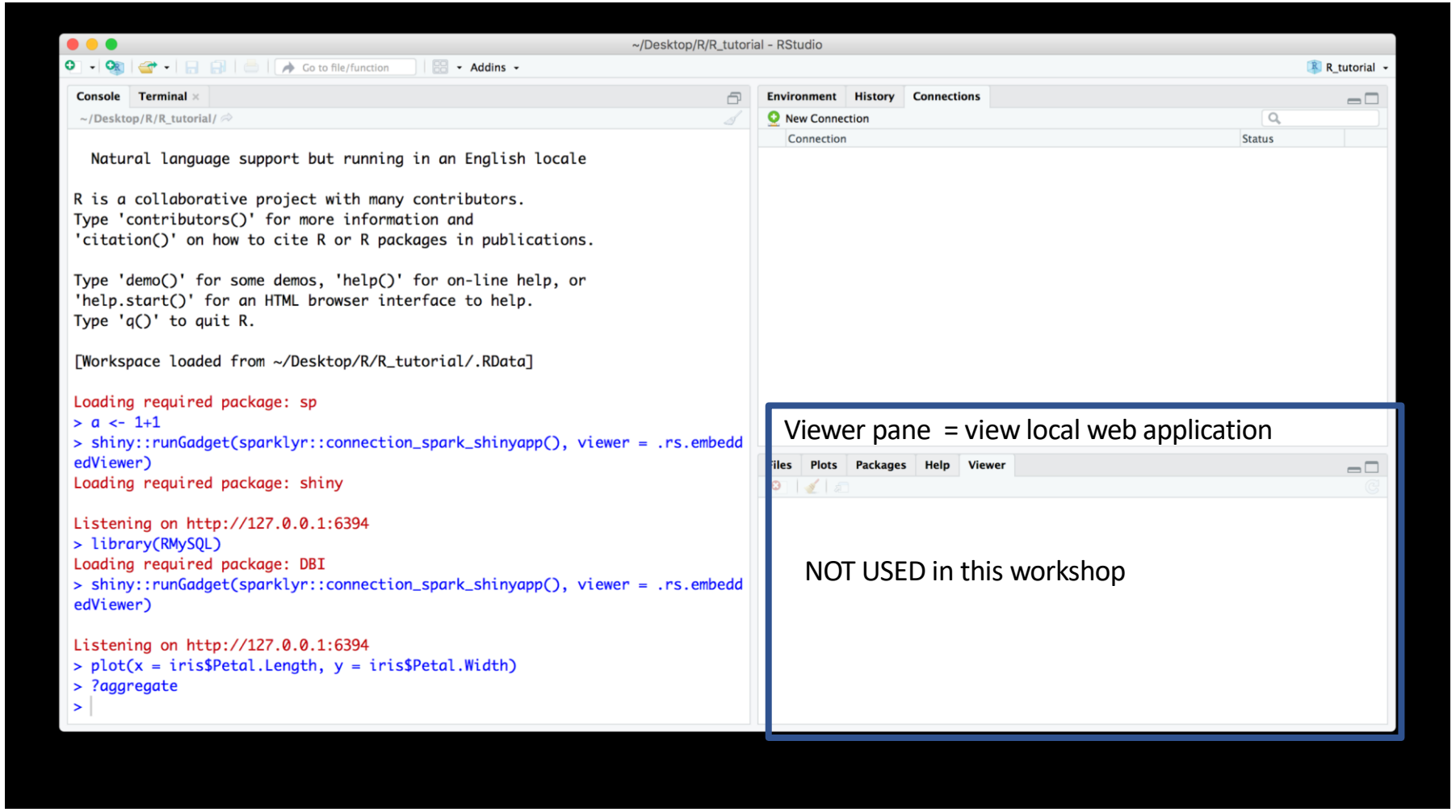
Panes of RStudio



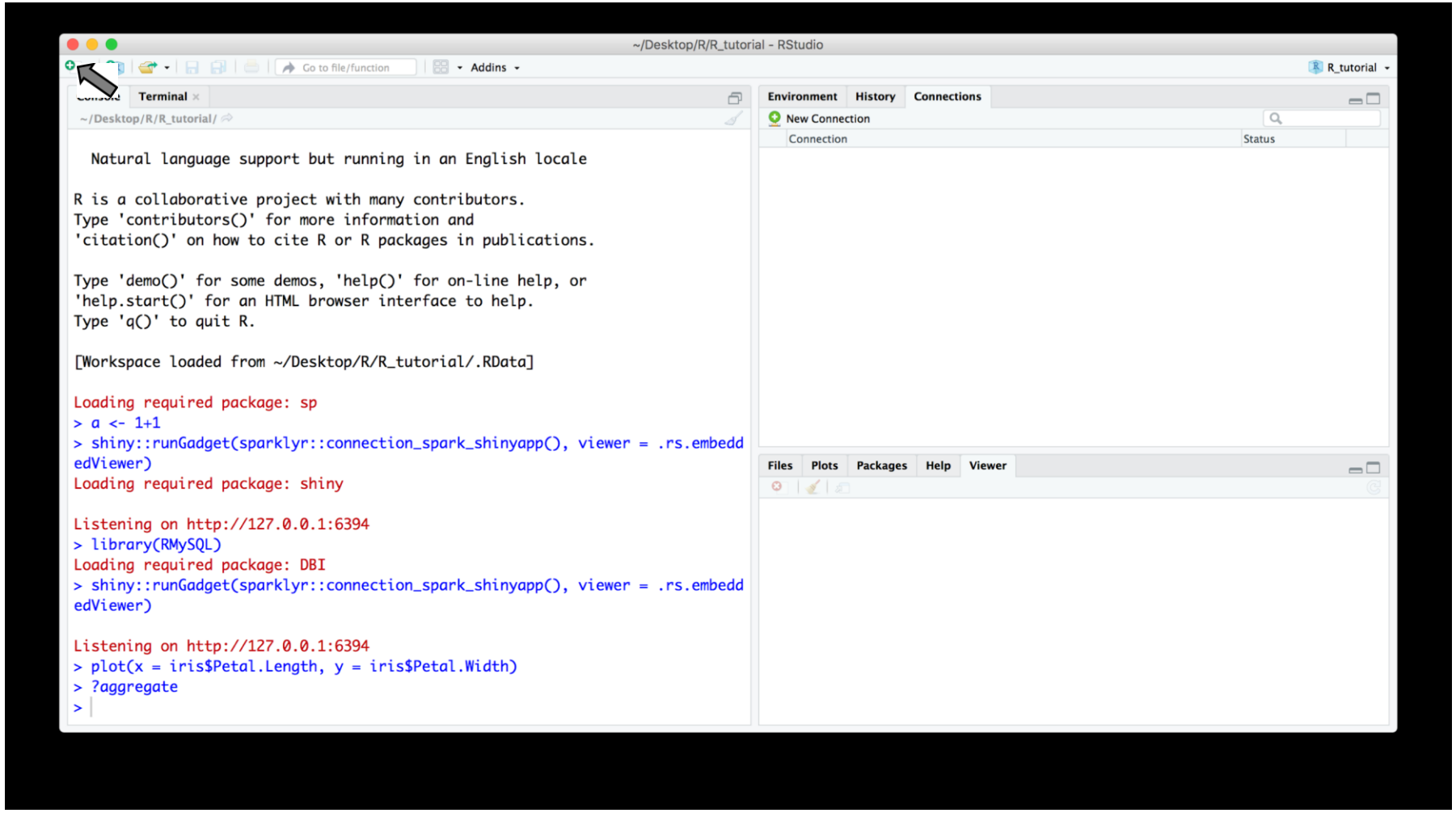
Panes of RStudio



Panes of RStudio



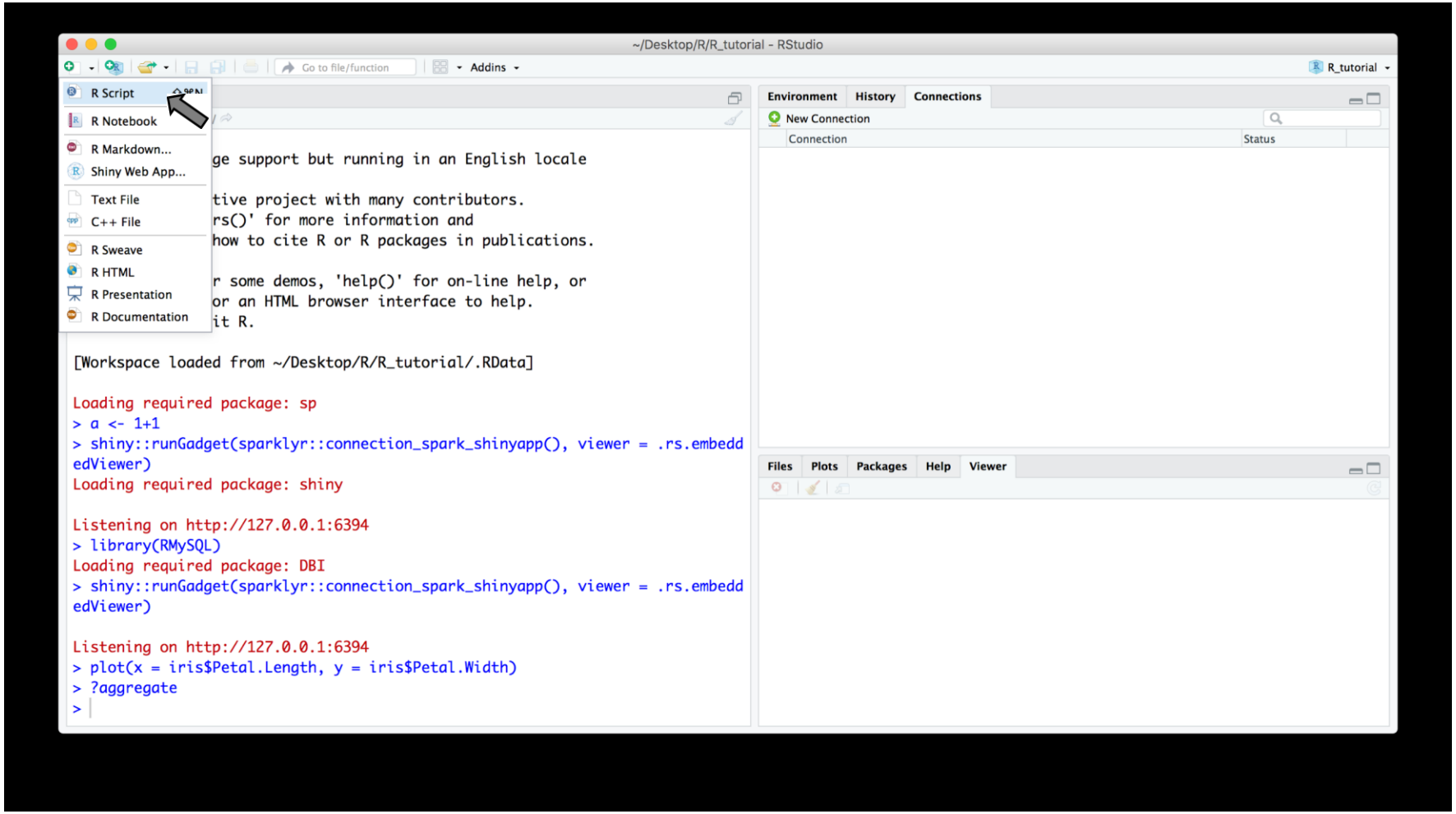
Panes of RStudio



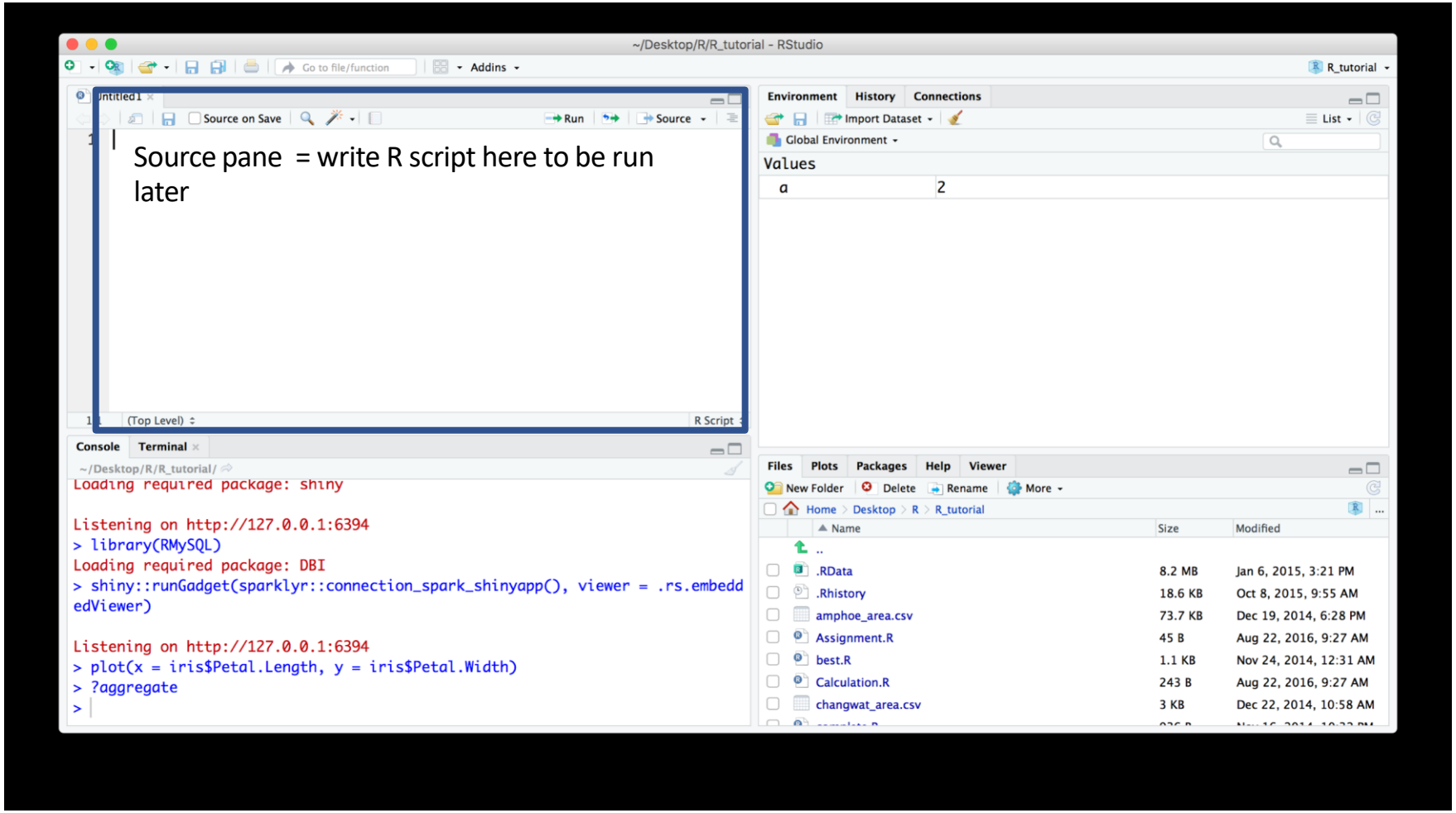
Source Panes

- Commands entered in the Console Pane will be immediately executed.
- To rerun the commands, they must be retyped in the Console Pane or reloaded from the History Pane
- Source pane
  - Write commands in this pane to be saved and executed later.
  - Develop R script here

Open Source Pane

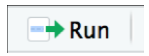


Open Source Pane

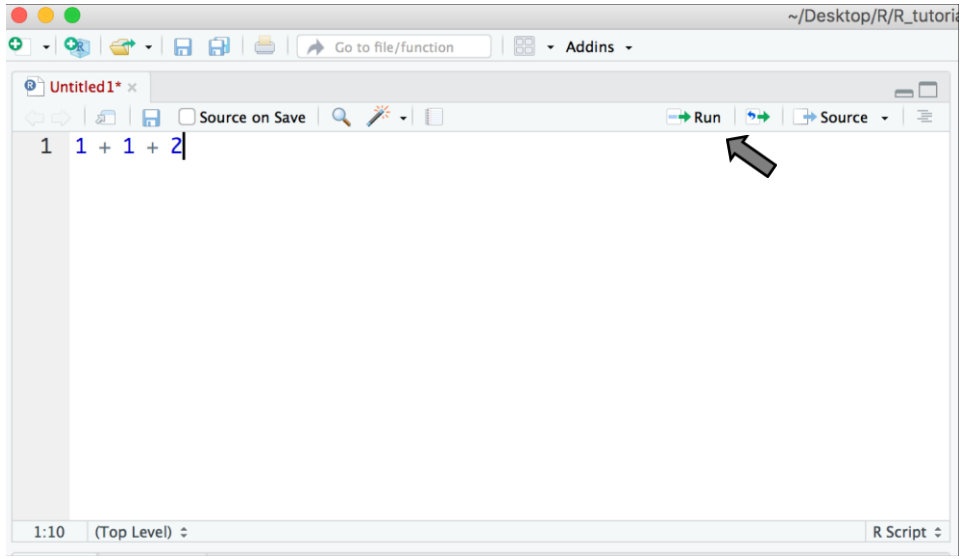


Run scripts in Source Pane

Run a single line of the script

- Leave a cursor at a line
- Click or 

- Ctrl + Enter or
- Mac: Command + Enter

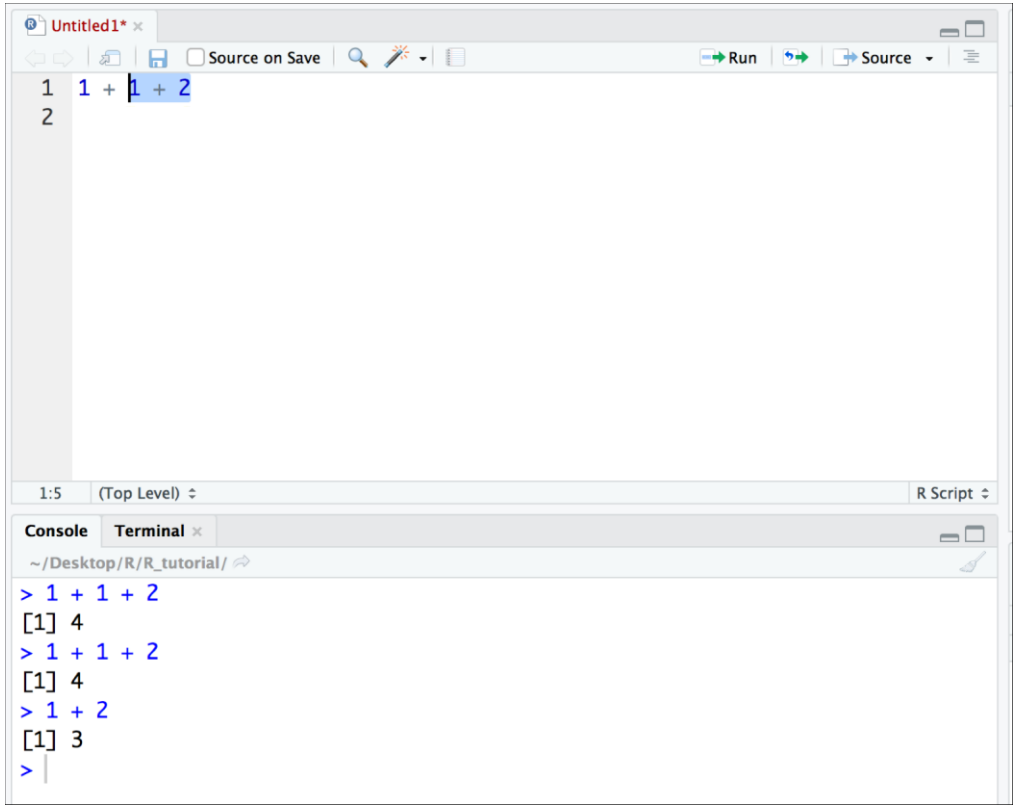


Run scripts in Source Pane

Run a part of the script

- Highlight the part
- Click or 

- Ctrl + Enter or
- Mac: Command + Enter

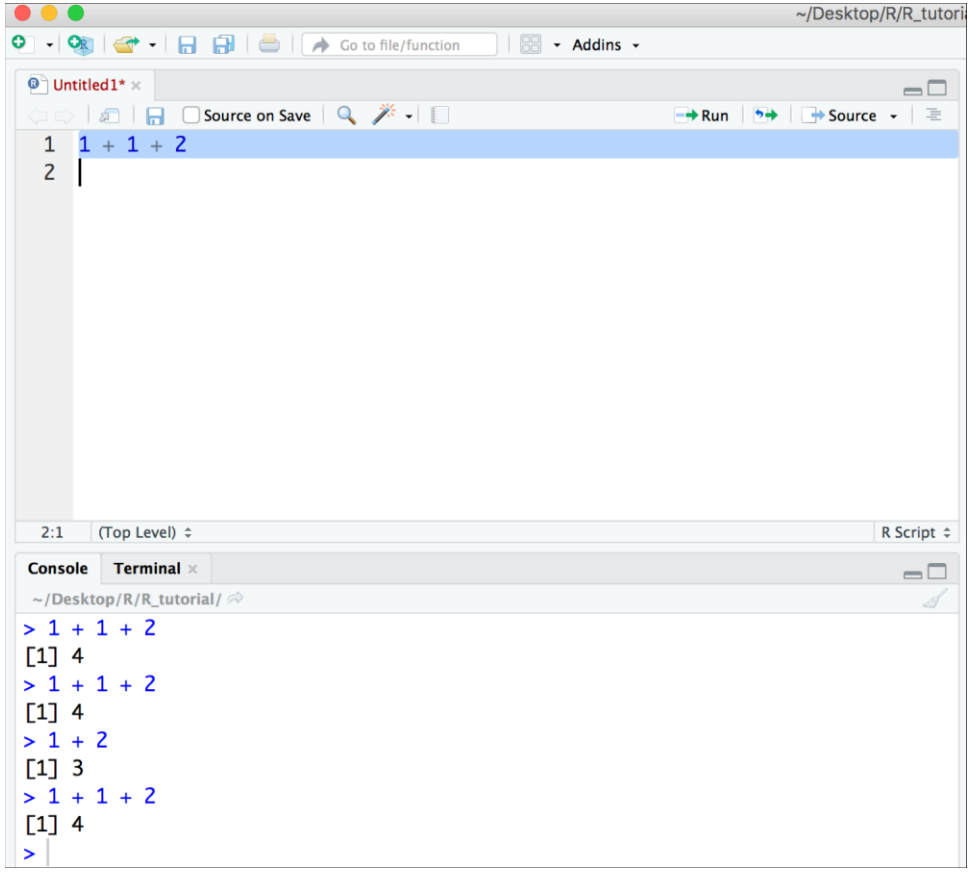


Run scripts in Source Pane

Run all lines of the script

- Click the source pane
- Ctrl/Command + A
- Click or 

- Ctrl/Command + Enter



Useful keyboard shortcuts

- Ctrl + S = Save the script
- Ctrl + L = Clear the console pane
- Ctrl + Z =Undo and Ctrl + Y = Redo
- Ctrl + X =Cut, Ctrl + C = Copy, Ctrl + V =Paste
- **Tab = ***Autofill***
- **↑ = ***Recall previously executed command*****
- Ctrl + Shift + F10 = Restart R session
- Ctrl + Q = Quit R

More shortcuts: Tools –> Keyboard Shortcuts Help

Practical: Packages

- R is powerful and versatile because of “Packages”
- “Packages” are codes/scripts created by users and shared to the R community
- 12,102 available packages deposited at CRAN

Practical: Packages

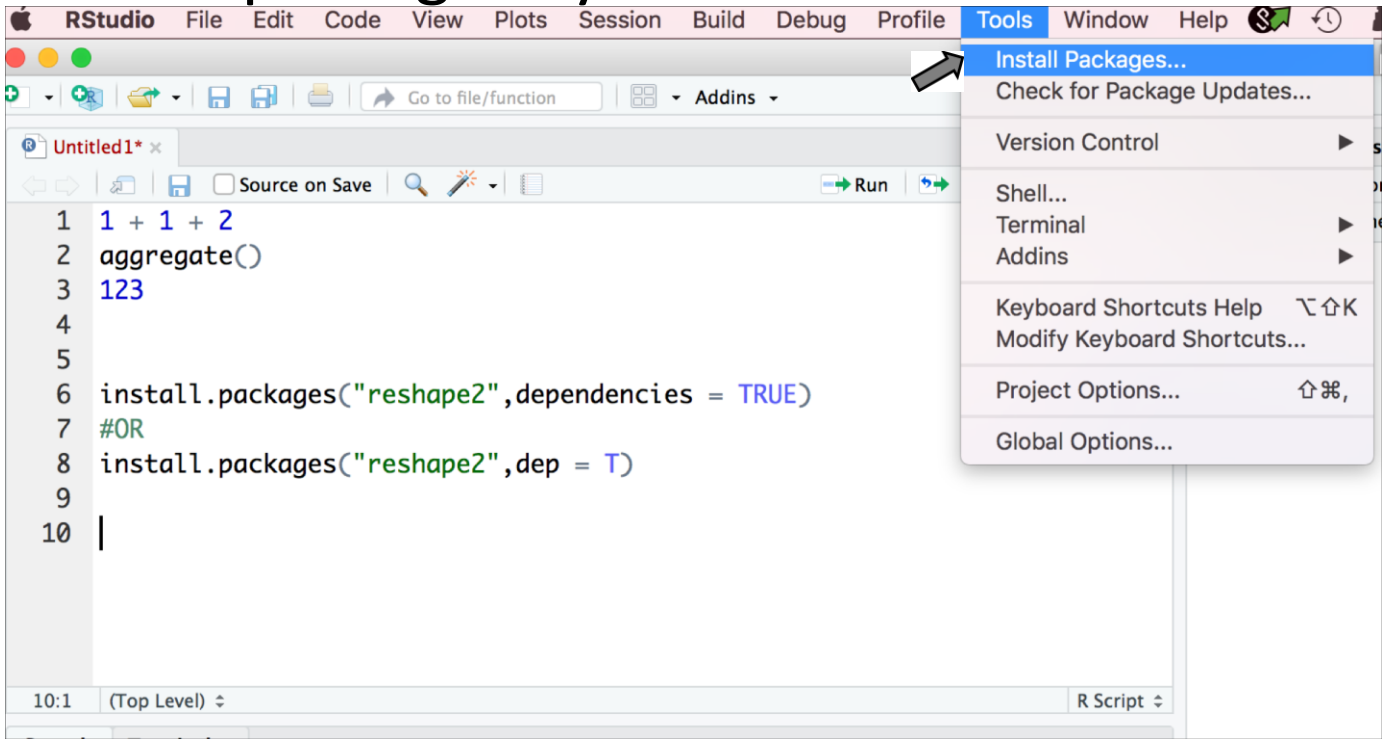
- To install packages by a command line

```
install.packages("reshape2",dependencies = TRUE)
#OR
install.packages("reshape2",dep = T)
```

- install.packages("name of the package in quotation marks", dependencies = TRUE)
- "dependencies = TRUE" means other packages required by this package will also be downloaded

Practical: Packages

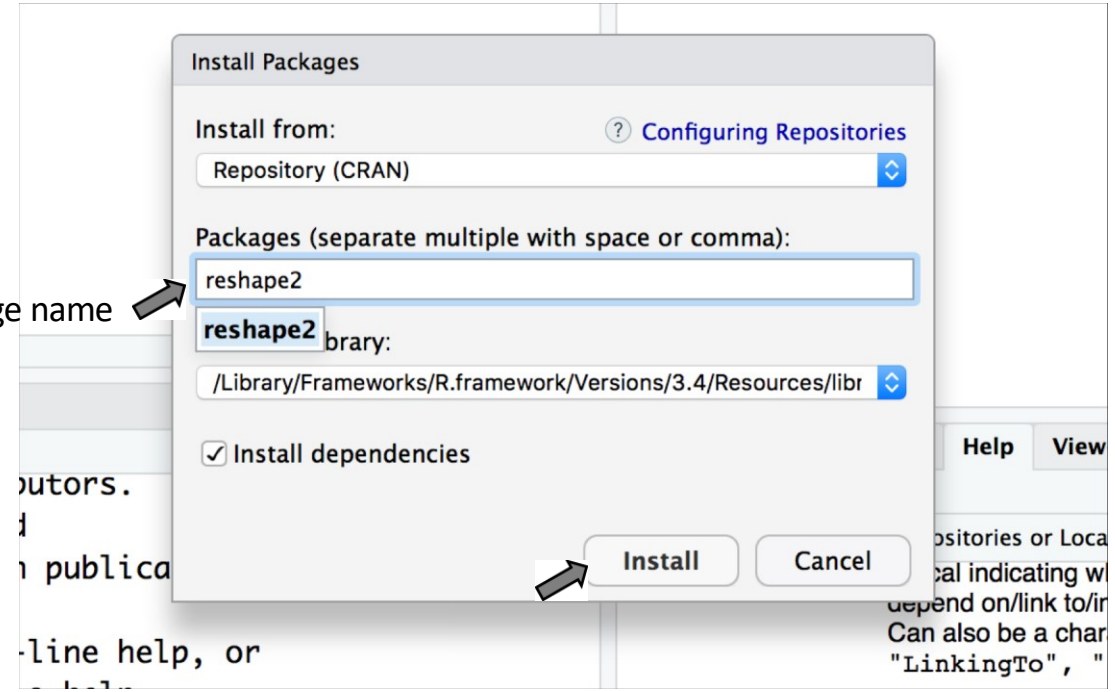
- To install packages by RStudio



Practical: Packages

- To install packages by RStudio

Type the package name

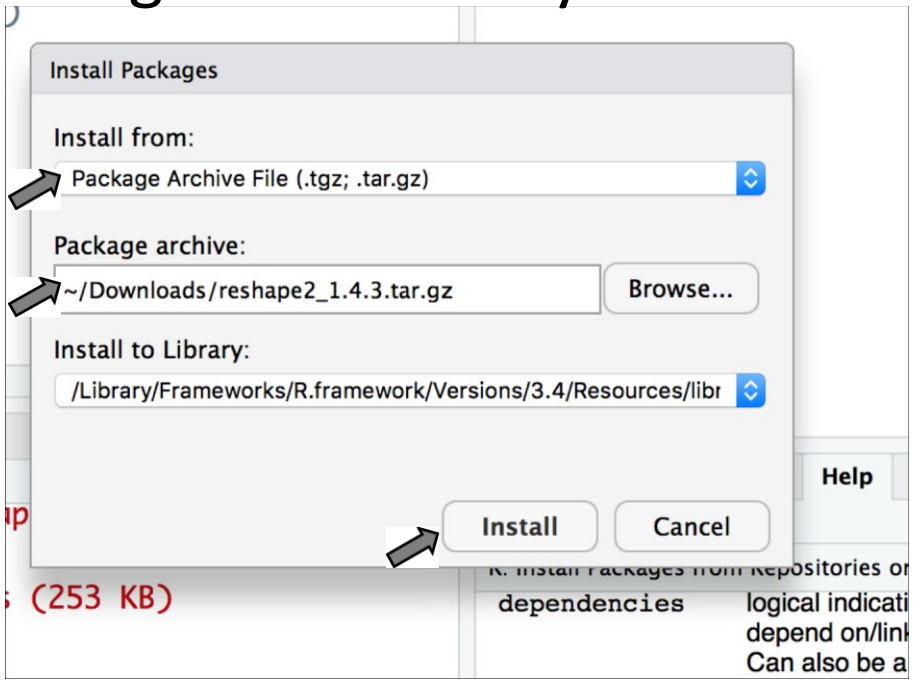


Practical: Packages

- To install packages “offline” by RStudio

Change from CRAN to File

Browse for the previously download package

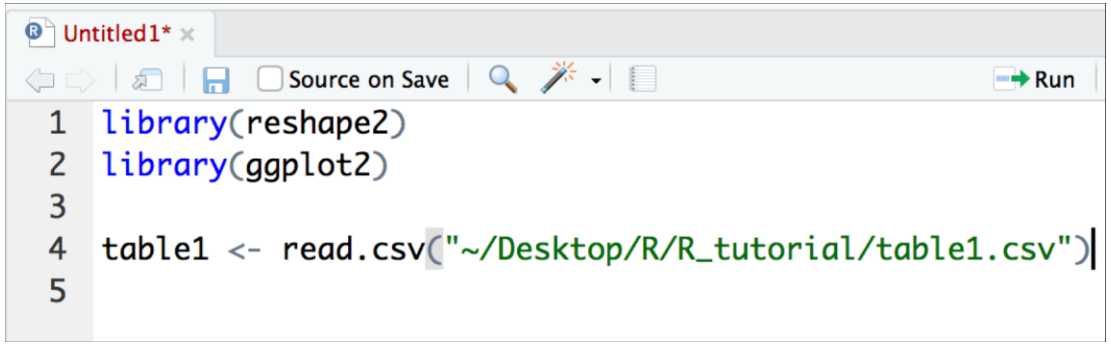


Practical: Packages

- To invoke the package

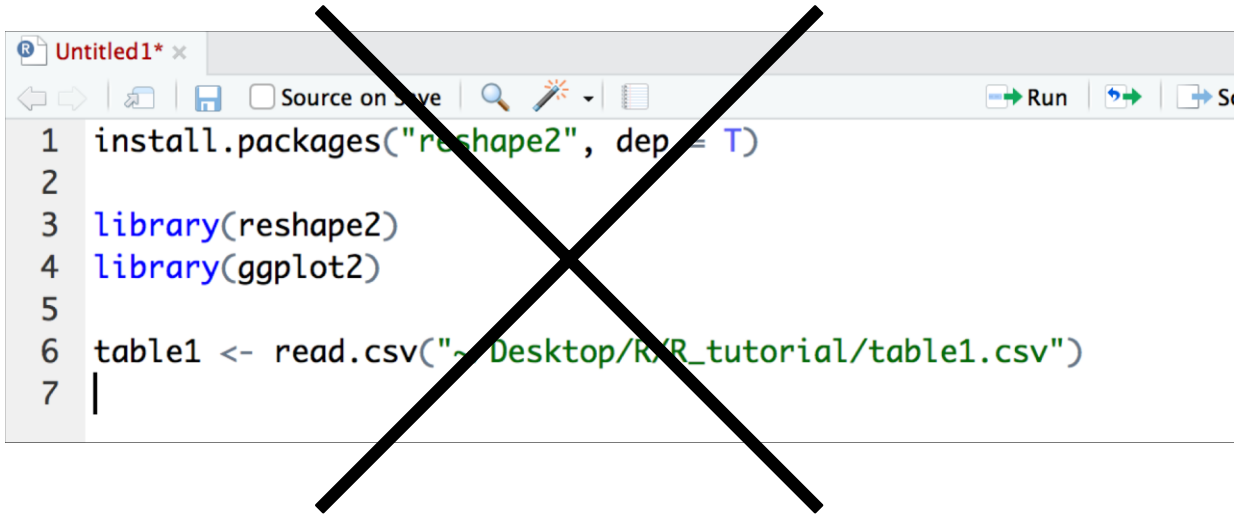
```
library(reshape2) #In your script
#OR
require(rehape2) #In the function/package
```

- Usually at the top of your script/code



Practical: Packages

- DO NOT include “install.packages()” in your script/code
- This will result in reinstalling packages every time the script is run



Practical: Packages

- Your turn:
  - Install and invoke the following packages:
    - reshape2
    - lubridate
    - MASS
    - car
    - Exact
    - hflights
    - readr
    - readxl

Programming Concepts in R

FILE:R2018_Programming_Concepts.R

Before we start…

- R scripts for demonstration are prepared
  - Run scripts and see what happens in your machine
    - Tell us if you see any strange or different results
  - Feel free to copy and modify the scripts for your own work

Calculator

Basic Mathematical Operations

```
1  ####Basic Mathematical Operations pt 1####
2  1 + 2 #Addition
3  1 - 2 #Substraction
4  1 / 2 #Division
5  1 * 2 #Multiplication
6  2 ^ 5 #Power
7  sqrt(4) #Square root
8  5 %/% 2 #Integer division
9  5 %% 2 #Modulo
```

- TIP: '#' = comment sign; Nothing after # to the end of the line will be run!!!

Basic Mathematical Operations

```
####Basic Mathematical Operations pt 2####
log10(100) #Logarithm base 10
log(100, base = 10) #Logarithm base 10
log(100) #Natural logarithm
exp(1) #Exponential i.e. e^1
round(2.111, digits = 1) #Rounding
floor(2.111) #Round down to integer
ceiling(2.111) #Round up to integer
abs(-123) #Absolute value
```

Basic Logical Operations

```
####Basic Logical Operations pt 1####
20 == 18 #Comparing whether 'left' equals to 'right.'

20 = 18 #This is wrong. Try and see what happens.

20 != 18 #Comparing whether 'left' does not equal to 'right.'
20 < 18 #Comparing whether 'left' is less than 'right.'
20 > 18 #Comparing whether 'left' is greater than 'right.'
20 <= 18 #Comparing whether 'left' is less than or equal to 'right.'
20 >= 18 #Comparing whether 'left' is greater than or equal to 'right.'

20 =< 18 #This is wrong. Try and see what happens.
20 => 18 #This is wrong. Try and see what happens.
```

Basic Logical Operations

```
35 ####Basic Logical Operations pt 2####
36 (20 > 18) & (20 < 18) #'AND' operation
37 (20 > 18) | (20 < 18) #'OR' operation
38 !(20 < 18) #'NOT' operation
```

Variable Assignment

Variable assignment

```
40 ####Variable assignment pt 1####
41 (20 > 18) & (20 < 18)
42 #Versus
43 a <- (20 > 18)
44 b <- (20 < 18)
45 c <- a & b
46 print(c) #'print()' is to show the values of the variable
47
48 d <- (20 == 18)
49 e <- c | d #Versus ((20 > 18) & (20 < 18)) | (20 == 18)
50 print(e)
```

- Critical concept in programming
- Save results/outputs for later
- Make your code more legible
- More legible = Easier for debugging

Variable assignment

- How to assign values to variables

```
52 ####Variable assignment pt 2####
53 a = 1 #Generic: Assign the 'right' value to the 'left' variable
54
55 a <- 1 #R: Assign the 'right' value to the 'left' variable
56 1 -> a #R: Assign the 'left' value to the 'right' variable
57 a <- 1 -> b #R: Multiple assignments
58 a <- b <- 1 #R: Multiple assignments
```

- ‘ = ‘ and ‘<– ‘ are mostly interchangeable
- Personally, I prefer ‘<-’ to avoid the confusion between ‘=‘ and ‘==‘

Variable assignment

- Naming your variables
  - Start with alphabet or ‘ . ’
  - No space or special character in the name except ‘ . ’ and ‘_’
  - Be careful! R is case-sensitive

```
####Variable assignment pt 3####
a <- 123 #Correct
a1 <- 123 #Correct
1a <- 123 #Incorrect
a 1 <- 123 #Incorrect
a_1 <- 123 #Correct
a.1 <- 123 #Correct #Different meaning in Java or Python
a! <- 123 #Incorrect
a? <- 123 #Incorrect
.a <- 123 #Correct
print(a1) #R is case-sensitive
print(A1) #R is case-sensitive
```

Types of Data

Basic types of data

```r
####Types of data pt 1####
a <- 2 #numeric
b <- TRUE #logical
c <- "Hello, World!" #character #Must be in " " or ' '
c <- Hello, World! #Try and see
d <- NA #missing (logical)
e <- NaN #'Not a Number' = ill-defined e.g. 0/0 (logical)

typeof(a) #Check the 'R internal' type of data by 'typeof()'
class(a) #Check the 'customized' type of data by 'class()'
#class() is used more often.

is.logical(b) #'is.logical()' is for checking if the data is 'logical'
is.logical(d)
is.na(b) #'is.na()' is an essential function for managing missing data.
is.na(d)
#is.nan(); is.numeric(); is.character() and others
```

Complex types of data

- More complex types of data or ‘objects’
- Containing multiple elements of basic data types (i.e. They are still based on basic types of data).
- Some types were developed for certain applications (e.g. spatial objects, survival objects)
- They might be converted to basic types of data.

Factor

- Recoding ‘categorical’ or ‘ordinal’ values to calculable values
  - Character + Numeric
- Example: Categorical variables in regression analysis

```
90  ####Types of data pt 2####
91  a <- "good"
92  class(a)
93  typeof(a)
94  af <- factor(a, levels = c("terrible","bad","neutral","good","excellent"))
95  print(af)
96  class(af)
97  typeof(af)
98  as.numeric(af) #Convert to numeric, i.e. extracting numerical elements
99  as.character(af) #Convert to character, i.e. extracting character elements
```

Date’ and ‘Time’

- Recoding texts of ‘date’, ‘time’ or ‘datetime’ data to calculable values
- Example: Used in time series analyses, survival analyses

```
####Types of data pt 3####
library(lubridate)

a <- "27-3-2018"
class(a)
typeof(a)
ad <- dmy(a) #function in lubridate package converting 'character' to 'date'
print(ad)
class(ad)
typeof(ad)
as.numeric(ad) #Convert to numeric (i.e. Days since 1-1-1970)
as.character(ad) #Convert to character, i.e. extracting character elements
```

Data type conversion

- Frequently used commands for data type conversion

```
####Conversion####
a <- 1
print(a)
as.character(a) #Convert to character
as.factor(a) #Convert to factor; cannot change levels or labels
factor(a, levels = c(3,2,1)) #Specify levels
factor(a, levels = c(1,2,3), labels = c("one","two","three")) #Specify labels

b <- "111"
as.numeric(b)
c <- "1,111"
as.numeric(c) #CAUTION: Data conversion may introduce missing data
d <- "1111"
as.numeric(d)
```

Data Structure
(Data container)

Vector

- One-dimensional container
- Every member in the vector must be the same type
- c() to construct a vector

```
114 ####Vector pt 1####
115 A <- c(1,2,3,4)
116 print(A)
117 class(A)
118
119 B <- c("a","b","c","d")
120 print(B)
121 class(B)
122
123 C <- c(TRUE, FALSE, T, F)
124 print(C)
125 class(C)
```

```
127 ####Vector pt 2####
128 D <- c(1,NA,3,0/0)
129 print(D)
130 class(D)
131
132 E <- c(1,TRUE,0/0,"a")
133 print(E)
134 class(E)
```

Matrix

- Two-dimensional container
- Every member in the matrix must be the same type
- matrix() to construct a matrix

```
####Matrix pt 1####
a <- 1:8 #a <- c(1,2,3,4,5,6,7,8) or a <- seq(from=1,to=8,by=1)
print(a) #vector of data
A <- matrix(data = a,nrow = 4)
#load 'a' vector into a matrix
#nrow' indicates number of rows
print(A)

B <- matrix(data = a,ncol = 4) #'ncol' indicates number of columns
print(B)

C <- matrix(data = a,ncol = 4, byrow = T)
#'byrow' fill values by row
# Default is by column (i.e. byrow = F)
print(C)
```

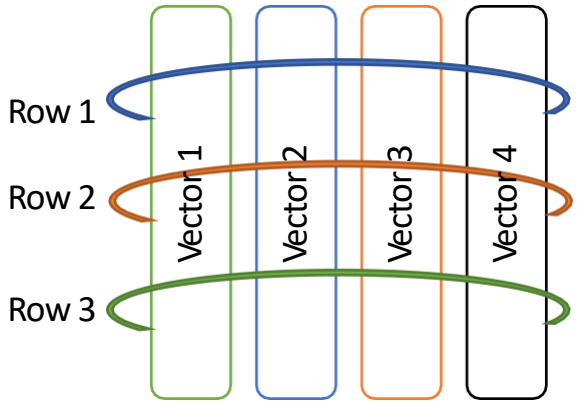
Matrix

- Two-dimensional container
- Every member in the matrix must be the same type
- matrix() to construct a matrix

```
152 ####Matrix pt 2####
153 D <- matrix(data = c(1:6,NA,T),ncol = 4, byrow = T)
154 print(D) #Look at the last member of the matrix
155
156 E <- matrix(data = c(1:6,T,"a"),ncol = 4, byrow = T)
157 print(E) #Look at the type of data in the matrix
```

Data frame

- Two-dimensional container
- In the “data handling” section of this workshop, we will mostly use data frames
- Every member in each “column” of the data frame must be the same type
- Each row can contain different types of data
- Imagine multiple vectors with identical lengths with each member at the same position bound together as a ‘row.'



Data frame

- data.frame() to construct a data frame

```
159 ####Data frame pt 1####
160 A <- data.frame( name = c("A","B","C"), #Vector 1 assigned as 'name' column
161                 weight = c(50,60,55), #Vector 2 assigned as 'weight' column
162                 height = c(150,166,162)) #Vector 3 assigned as 'height' column
163 print(A)
164
165 A2 <- data.frame(c("A","B","C"), #No column names assignment
166                 c(50,60,55),
167                 c(150,166,162))
168 print(A2)
169
170 B <- data.frame( name = c("B","C"),
171                 weight = c(50,60,55),
172                 height = c(150,166,162)) #Try and see what happens
```

Data frame

- data.frame() to construct a data frame

```
175 ####Data frame pt 2####
176 C <- data.frame( name = c("A","B","C"),
177                  weight = c(50,60,"Not measured"),
178                  height = c(150,166,162))
179 print(C) #Try and see
180 A$weight #'A$weight' is for calling the 'weight' column of 'A' data frame
181 #We will get into how to access data in the data frame in the later session
182 class(A$weight)
183
184 C$weight
185 class(C$weight)
```

List

- Multi-dimensional container
- Each member in the list can be any type of data, or even data container (i.e. a data frame in a list > two dimensions)
- No restriction of length for each member
- Used as a container of results of analysis (e.g. regression results)

List

- list() to construct a list

```
188 ####List####
189 a <- c(T,F)
190 b <- 1:6
191 c <- c("A","E","I","O","U")
192 
193 A <- list(a,b,c) #Combine vectors as a list
194 print(A)
195 
196 A2 <- list(col1 = a, col2 = b, col3 = c) #'name' can be assigned to each member
197 print(A2)
198 
199 d <- data.frame( name = c("A","B","C"),
200                                  weight = c(50,60,55),
201                                  height = c(150,166,162))
202 B <- list(A,d) #list and data frame in the list
203 print(B)
```

Practical 1

Practical 1: Data type conversion

- Numerical data are sometimes created or imported to R as ‘factor’ data.
- Incorrectly using factor data can be problematic

```
C <- data.frame( name = c("A","B","C"),
                 weight = c(50,60,"Not measured"),
                 height = c(150,166,162))
weight <- C$weight
#Convert weight from kilograms to pounds
#1 Kilo = 2.2 pounds
weight * 2.2 #Try and see
```

Practical 1: Data type conversion

1. Convert 'weight' to numerical data
2. Convert 'weight' to pound (1 kilo ~ 2.2 pound)

- HINT1: Correct answer = c(110, 132, NA)
- HINT2: as.numeric(); as.character() for data type conversion

```
C <- data.frame( name = c("A","B","C"),
                 weight = c(50,60,"Not measured"),
                 height = c(150,166,162))
weight <- C$weight
#Convert weight from kilograms to pounds
#1 Kilo = 2.2 pounds
weight * 2.2 #Try and see
```

Flow Control

Curly brackets { }

- ‘if-else’, ‘for’, ’while’ and ‘function’

```
command(conditions or values){
        do something
        do something
        do something
}
```

If and Else

- Modify the behavior of the code based on the input
- Evaluate the input -> Select what to do -> Generate the output
- Note: If-Else must address all possible conditions

```
192 ####If and else####
193 A <- 1 #Assign the input here
194 
195 if(!is.numeric(A)){
196   print("A is not a number")
197 }else if(A %% 2 == 0){
198   print("A is even")
199 }else if(A %% 2 == 1){
200   print("A is odd")
201 }else{
202   print("A is not an integer")
203 }
204 
205 #Try changing A to 2, 1.1, NA and 0/0
```

For loop

- Repeating an operations with changing inputs
- NOTE: ‘For loop’ in R is slow (try row/column operations and apply())

```
####For loop####
A <- c(1,2,3,4,5,6,7)
for(i in A){
  #'i' is a temporary variable with its value = each member of A
  #Reassignment of 'i' occured at every new round
  print(i)
}

B <- letters #letters and LETTERS are built-in vectors containing alphabets
for(j in B){
  #'j' is a temporary variable with its value = each member of A
  #Reassignment of 'j' occured at every new round
  print(j)
}
```

While loop

- Repeating an operations until the ‘breaking condition’ is triggered
- CAUTION: ‘Infinite loop’ = the ‘breaking condition’ is never triggered
- NOTE: ‘While loop’ in R is slow

```
222 ####While loop####
223 i <- 1
224 
225 while(i < 100){ #Breaking/Exiting condition
226   print(i) #Do something here
227   i <- i + 1 #DO NOT forget to the change breaking condition!!!
228 }
```

Practical 2

Practical 2: For loop

- ‘paste()’ is a function that combines two or more text data into one
- Try paste("I", "love", "you.")
- ‘paste0()’ is a function that combines two or more text data into one without adding a space between texts
- Try paste0("I", "love", "you.")

Practical 2: For loop

1. Our input is 2005:2018 (i.e. 2005, 2006,…,2017, 2018)
2. We will use the for loop and the ‘paste’ function to generate the following sentences:
   - This year is 2005.
   - This year is 2006.
   - This year is 2007.
   - …
   - …
   - This year is 2018.

```
1 for(k in 2005:2018){
2   #Do something with k
3   #print(something)
4 }
```

Practical 2: For loop

CHALLENGE

1. Leap years are divisible by 4
2. For leap years, add ‘This is a leap year.’ at the end of the sentence
3. Generate the following sentences:
   - This year is 2005.
   - …
   - This year is 2008. This is a leap year.
   - …
   - This year is 2018.

- HINT 1: If-else
- HINT 2: Modulo

Function

Function

- You can customize your own function
- Function calls multiple lines of code in a simple command
- NOTE: “Package” is a bundle of functions, manual/documentations and sample datasets

```
####If and else####
A <- 1 #Assign the input here

if(!is.numeric(A)){
  print("A is not a number")
}else if(A %% 2 == 0){
  print("A is even")
}else if(A %% 2 == 1){
  print("A is odd")
}else{
  print("A is not an integer")
}
```

Function

- Function should be defined at the beginning of the code after loading packages

```
####Function pt 1####
library(reshape2) #Nothing to do with the code
#Just demo how the code should look like
odd_even <- function(A){ #Define the 'odd_even' function
  if(!is.numeric(A) | is.nan(A)){#Debugged for NaN
    result <-  "Your input is not a number"
  }else if(A %% 2 == 0){
    result <-  "Your input is even"
  }else if(A %% 2 == 1){
    result <-  "Your input is odd"
  }else{
    result <-  "Your input is not an integer"
  }
  return(result)
}

odd_even(A = 12) #Call the function by name
odd_even(0/0)
```

Function

- Function can take multiple arguments
- Default value may be defined

```
####Function pt 2####
testFn <- function(A,B,C = 2){#C is 2 by default
  result <- (A+B)*C
  return(result)
}

testFn(A = 1, B = 2) #C is 2 by default
testFn(A = 2, B = 3, C = 4)
testFn(0,4,1) #A = 0, B = 4, C = 1 -> Assign by the order of arguments
testFn(C = 0,B = 4, A = 1) #Specifically assign value to A,B,C
```

Help!!!

Help!!!

- General programming concepts in R are similar to other languages
- Tough parts are ‘function’
- Find the right function and learn to use it can be difficult

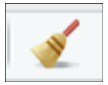
```
260 ####Help####
261 help("shapiro.test")
262 ?shapiro.test
263 ??shapiro.test
```

Prepare your workspace

Working directory

- getwd() = the current working directory
  - Anything saved without a specified path will be here
- setwd() = change to the new working directory
  - Organizing the location in which files will be read and written

Before running your codes

- Restart “R session”: Session -> Restart R
- Clear the environment 
- Check your working directory

Practical 3

Practical 3: Customize your function

- Quadratic equation
  - $ax^2 + bx + c = 0$
- Quadratic formula is the solution of the quadratic equation
  - $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Practical 3: Customize your function

1. Create a function ‘quardForm()’ that takes input a, b, c from a quadratic equation to solve for x
2. Test your function with the following equations:
   - $x^2 + 4x - 21 = 0$
   - $10x^2 + 13x - 3 = 0$
   - $x^2 - 25 = 0$
   - $x^2 + 4 = 0$

Practical 3: Customize your function

- HINT: quadratic formula needs to return two values

```
1 quadForm <- function(A,B,C){
2   #answer1 <- ???
3   #answer2 <- ???
4   return(c(answer1, answer2))
5 }
```