Introduction to Data Analysis in R-Studio

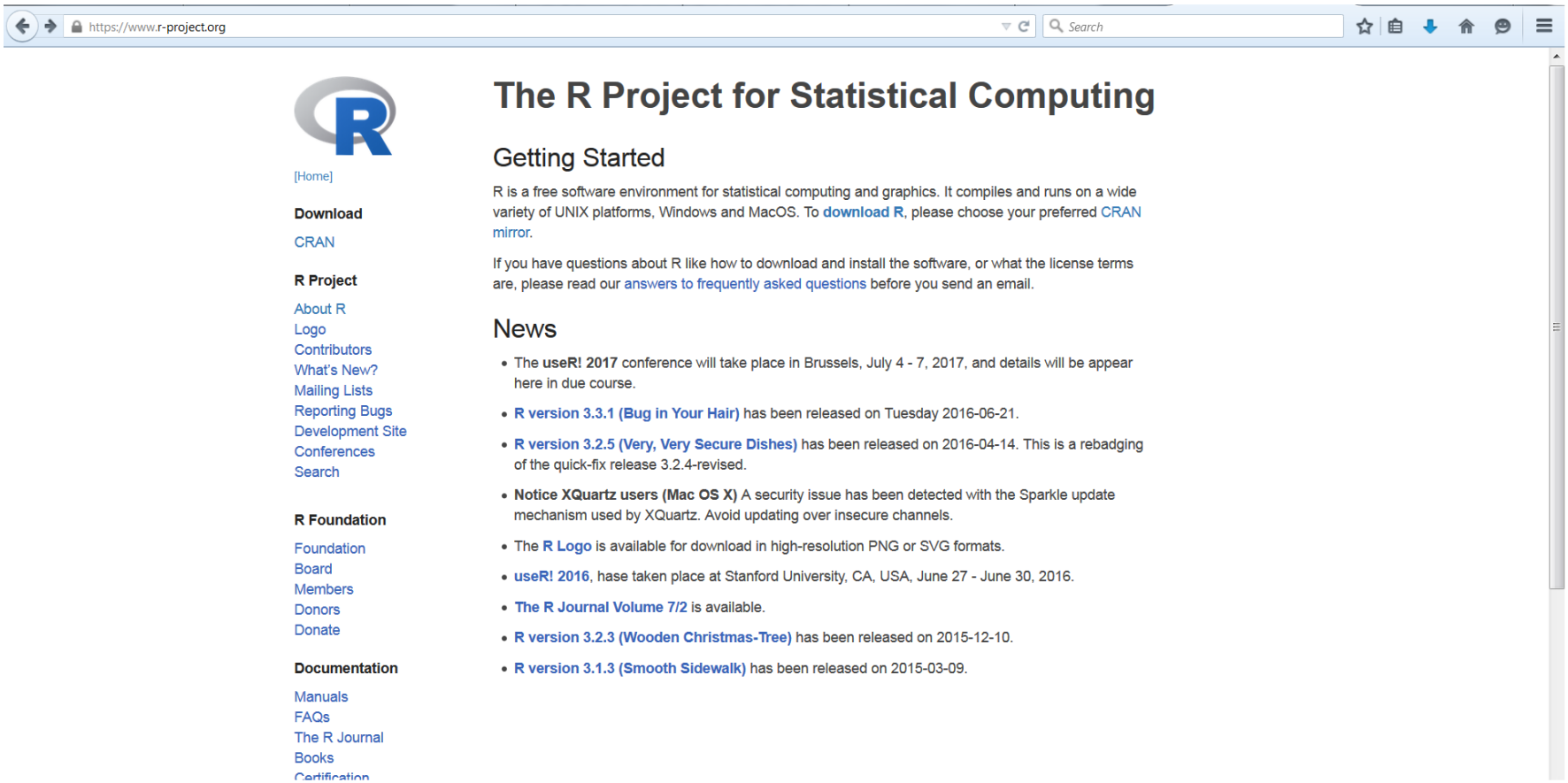
Raim,Ostrouchov,Neerchal

UMBC

About R

- R an open source statistical programming language and environment for statistical computing and graphics
- R supports user defined functions, and is capable of run-time calls to C, C++, Fortran codes
- Capability of R can be extended by packages (1300)
- R needs to be downloaded from `https://cran.r-project.org/` and installed (like any other software)
- Available for Windows, Mac or Linux
- R feels and looks are the same regardless of the underlying operating system (for the most part)
- Developed by Ross Ihaka and Robert Gentlemen, University of Auckland, in 1995.

Installing R

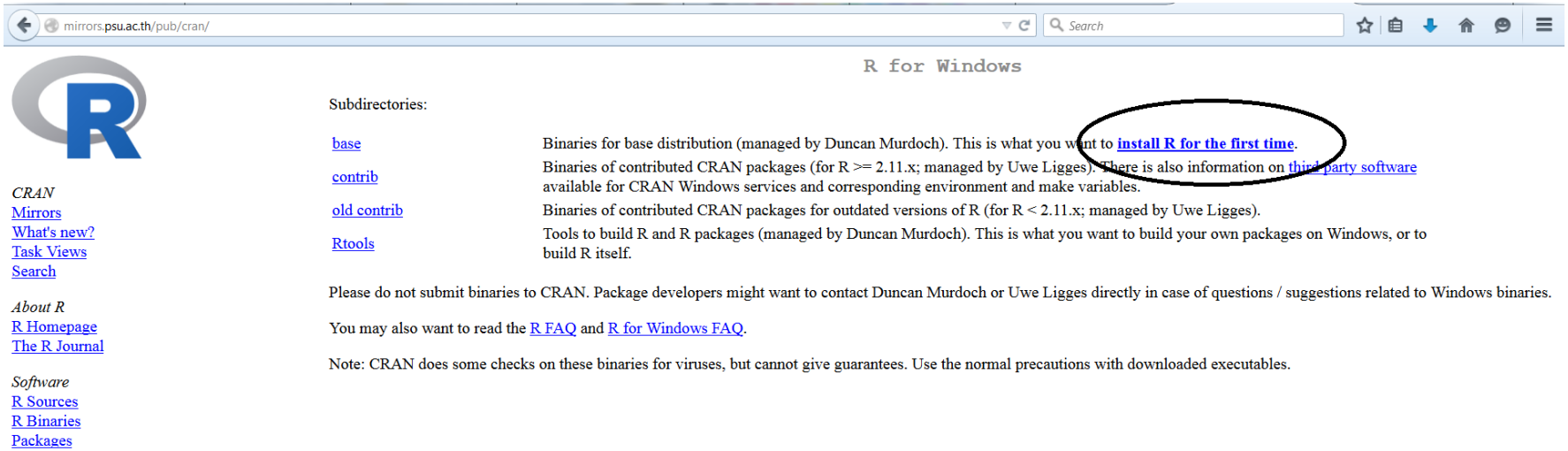


Installing R

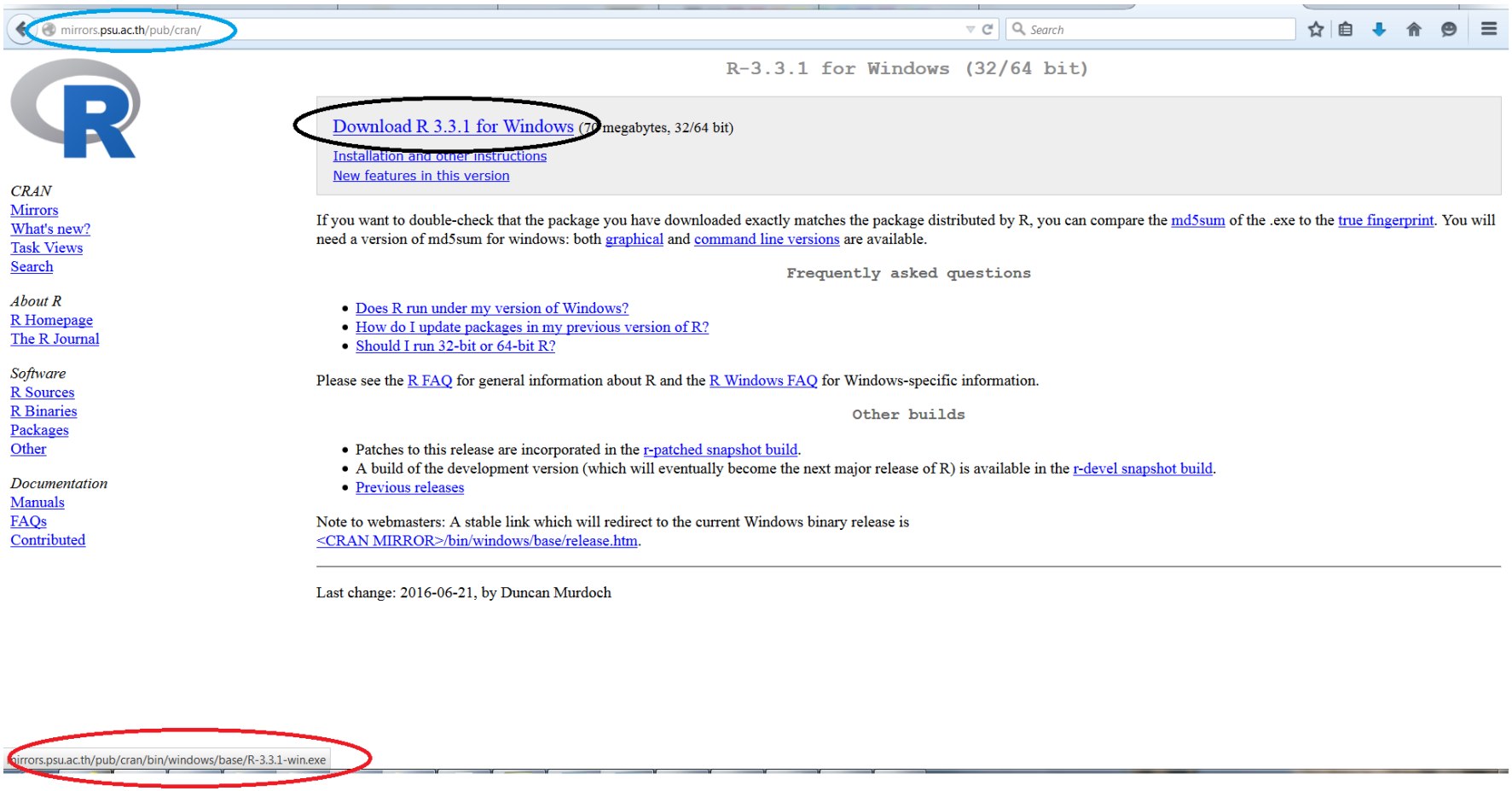
https://stat.ethz.ch/CRAN/	ETH Zürich
http://stat.ethz.ch/CRAN/	ETH Zürich
Taiwan	
http://ftp.yzu.edu.tw/CRAN/	Department of Computer Science and Engineering, Yuan Ze University
http://cran.csie.ntu.edu.tw/	National Taiwan University, Taipei
Thailand	
http://mirrors.psu.ac.th/pub/cran/	Prince of Songkla University, Hatyai
Turkey	
http://cran.pau.edu.tr/	Pamukkale University, Denizli
http://cran.ncc.metu.edu.tr/	Middle East Technical University Northern Cyprus Campus, Mersin
UK	
https://www.stats.bris.ac.uk/R/	University of Bristol
http://www.stats.bris.ac.uk/R/	University of Bristol
https://mirrors.ebi.ac.uk/CRAN/	EMBL-EBI (European Bioinformatics Institute)
http://mirrors.ebi.ac.uk/CRAN/	EMBL-EBI (European Bioinformatics Institute)
http://cran.ma.imperial.ac.uk/	Imperial College London
http://mirror.mdx.ac.uk/R/	Middlesex University London

Installing R

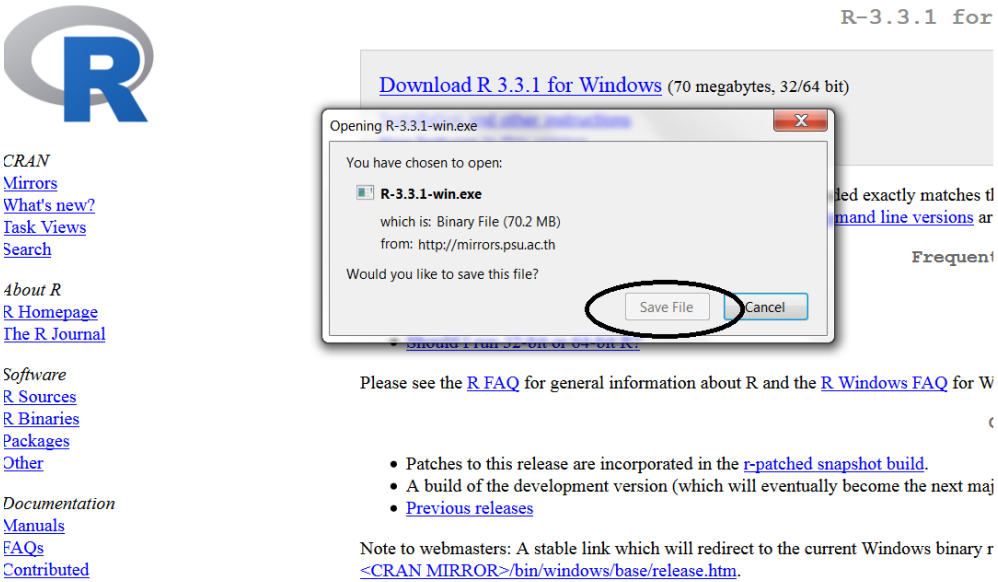
Installing R



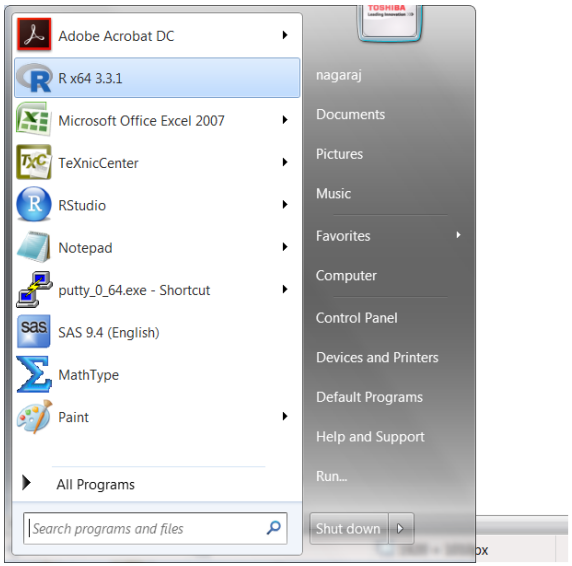
Installing R



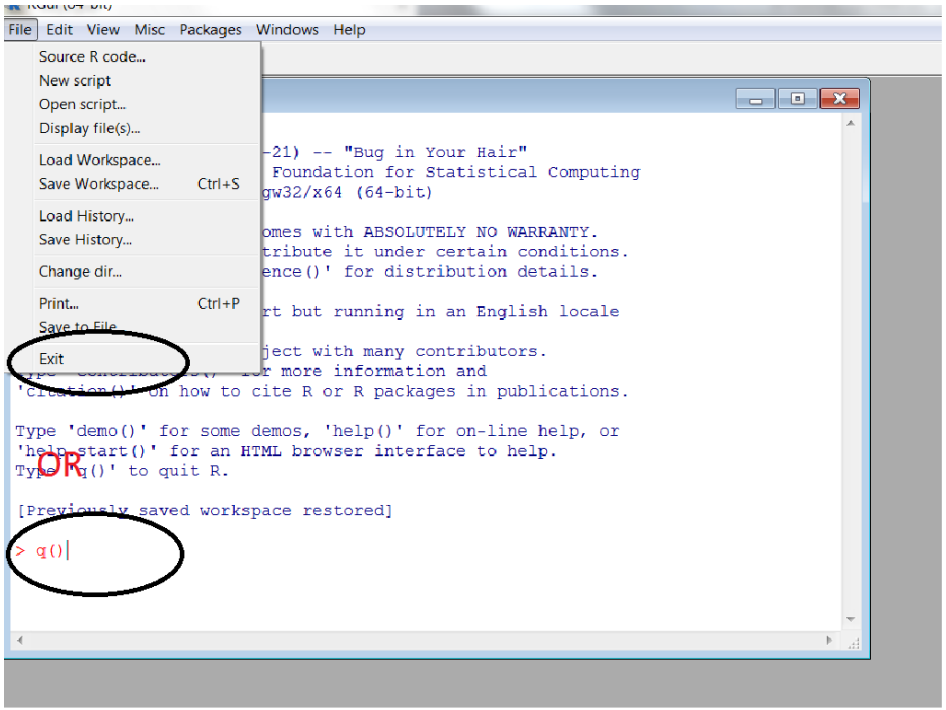
Installing R



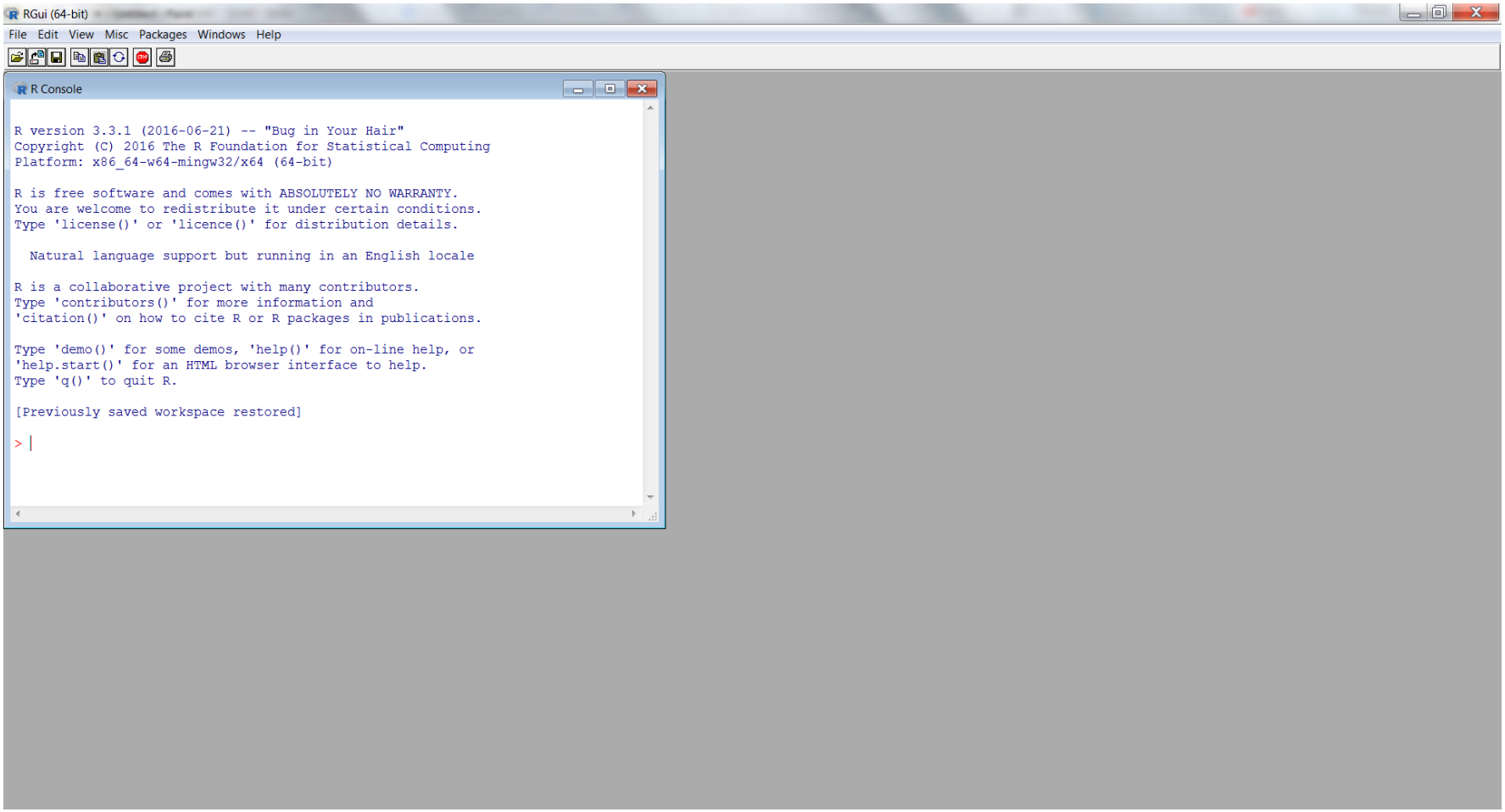
Starting R


Adobe Acrobat DC
R x64 3.3.1
Microsoft Office Excel 2007
TeXnicCenter
RStudio
Notepad
putty_0_64.exe - Shortcut
SAS 9.4 (English)
MathType
Paint
All Programs
Search programs and files
nagaraj
Documents
Pictures
Music
Favorites
Computer
Control Panel
Devices and Printers
Default Programs
Help and Support
Run...
Shut down

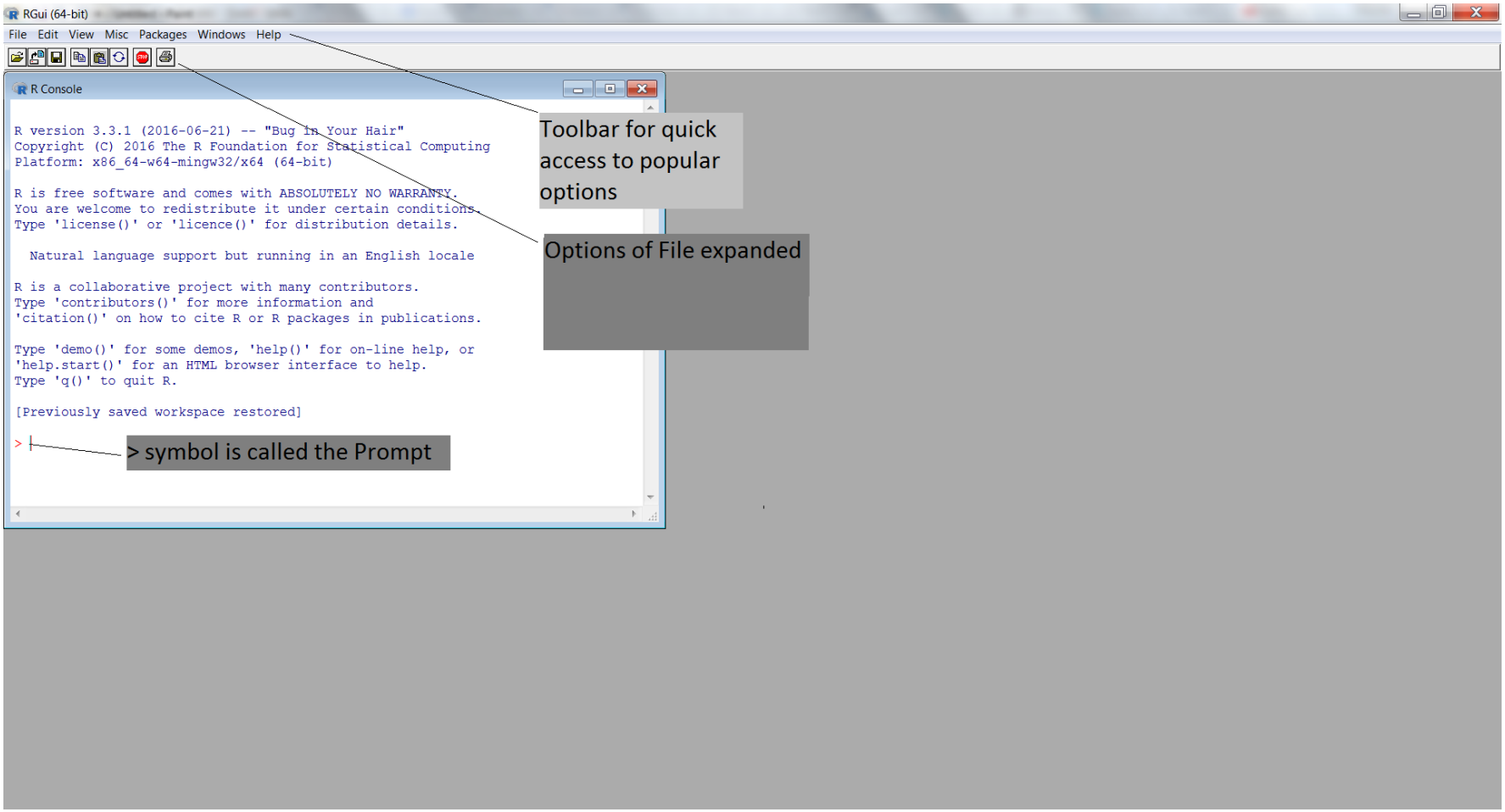
Quitting or Exiting R



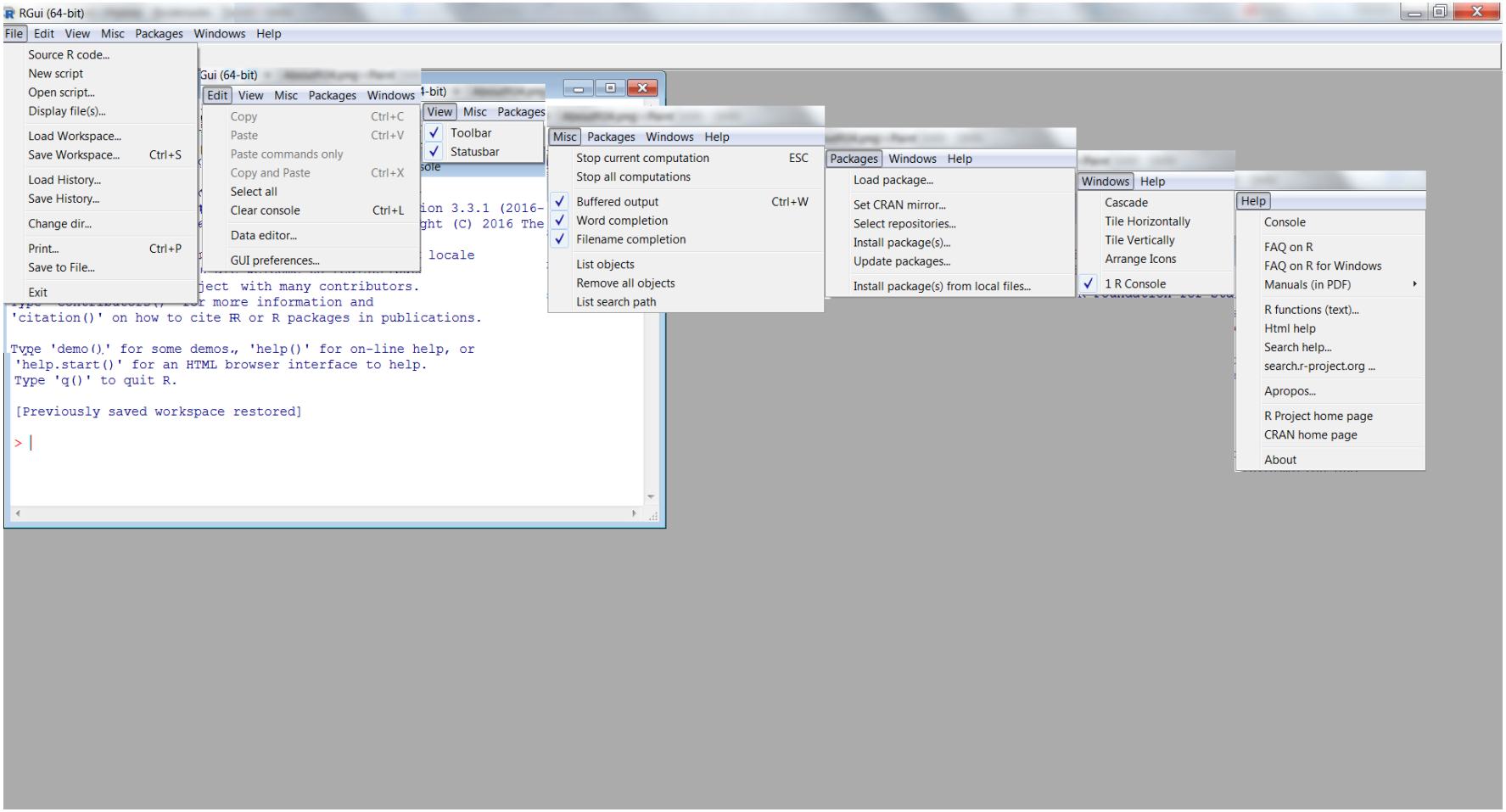
R Interface


RGui (64-bit)
File Edit View Misc Packages Windows Help
R Console
R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (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.
[Previously saved workspace restored]
>

R Interface Components

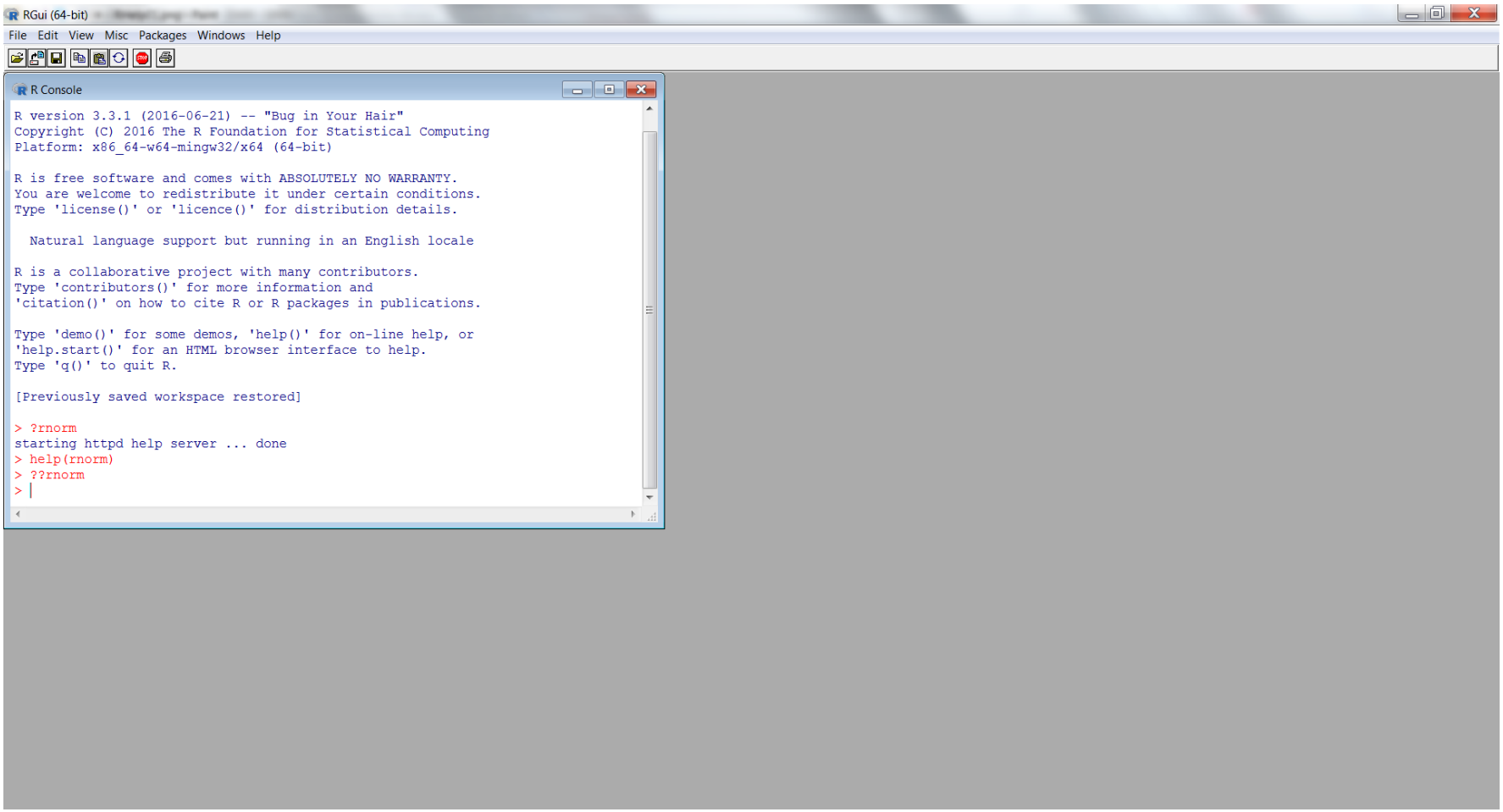


The menu tree of plain vanilla R



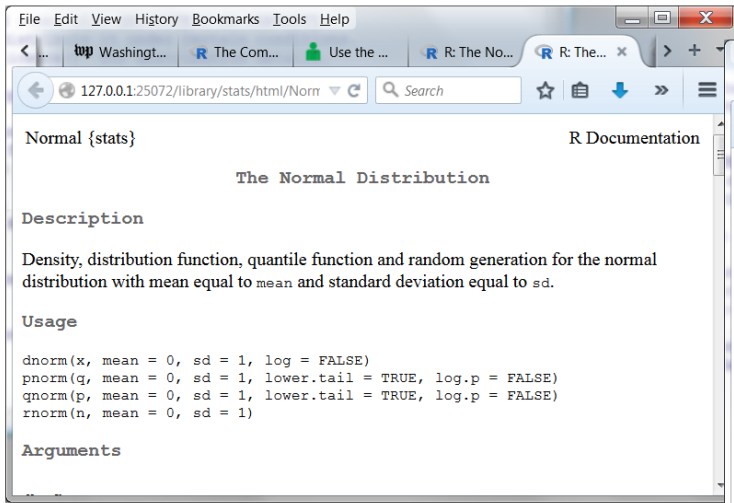
Help in R

- R has an easy-to-use help facility system. For example, if you want to get more information about function "rnorm", you can either type the command:
  $>$ help(rnorm) or $>$?rnorm



Help in R

- >help(rnorm) or >?rnorm or >??rnorm will take us to a help page



Normal {stats} R Documentation

The Normal Distribution

Description

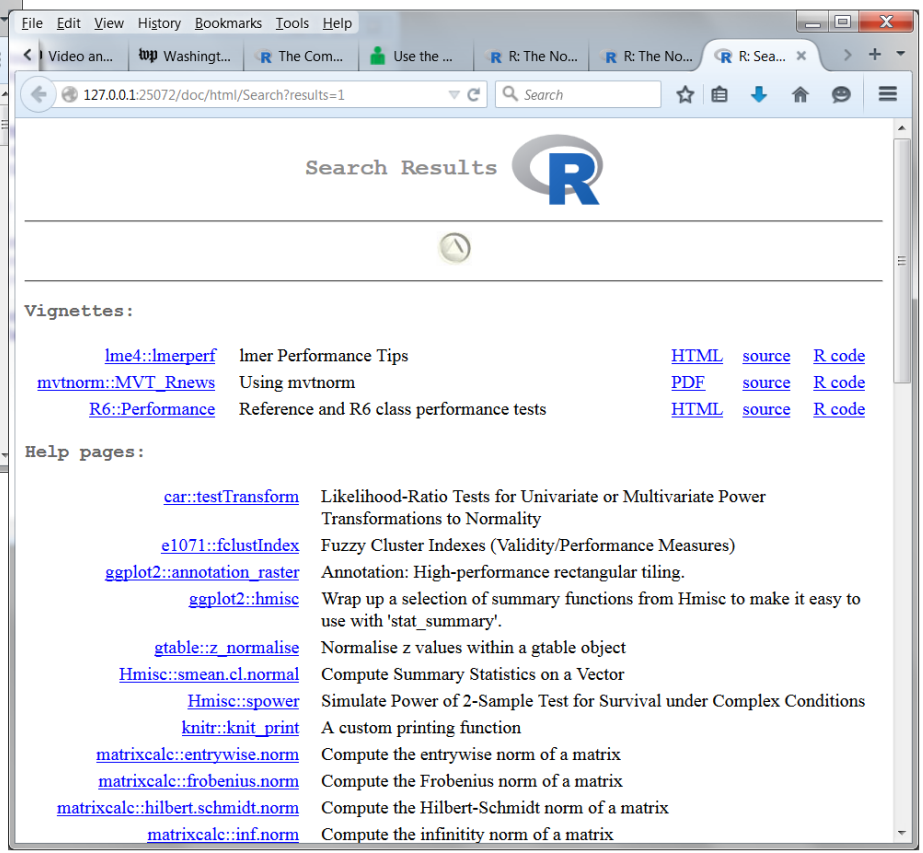
Density, distribution function, quantile function and random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

Usage

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

Arguments

?rnorm or help(rnorm)



Search Results

Vignettes:

lme4::lmerperf	lmer Performance Tips	HTML	source	R code
mvtnorm::MVT_Rnews	Using mvtnorm	PDF	source	R code
R6::Performance	Reference and R6 class performance tests	HTML	source	R code

Help pages:

car::testTransform	Likelihood-Ratio Tests for Univariate or Multivariate Power Transformations to Normality
e1071::fclustIndex	Fuzzy Cluster Indexes (Validity/Performance Measures)
ggplot2::annotation_raster	Annotation: High-performance rectangular tiling.
ggplot2::hmisc	Wrap up a selection of summary functions from Hmisc to make it easy to use with 'stat_summary'.
gtable::z_normalise	Normalise z values within a gtable object
Hmisc::smean.cl.normal	Compute Summary Statistics on a Vector
Hmisc::spower	Simulate Power of 2-Sample Test for Survival under Complex Conditions
knitr::knit_print	A custom printing function
matrixcalc::entrywise.norm	Compute the entrywise norm of a matrix
matrixcalc::frobenius.norm	Compute the Frobenius norm of a matrix
matrixcalc::hilbert.schmidt.norm	Compute the Hilbert-Schmidt norm of a matrix
matrixcalc::inf.norm	Compute the infinitity norm of a matrix

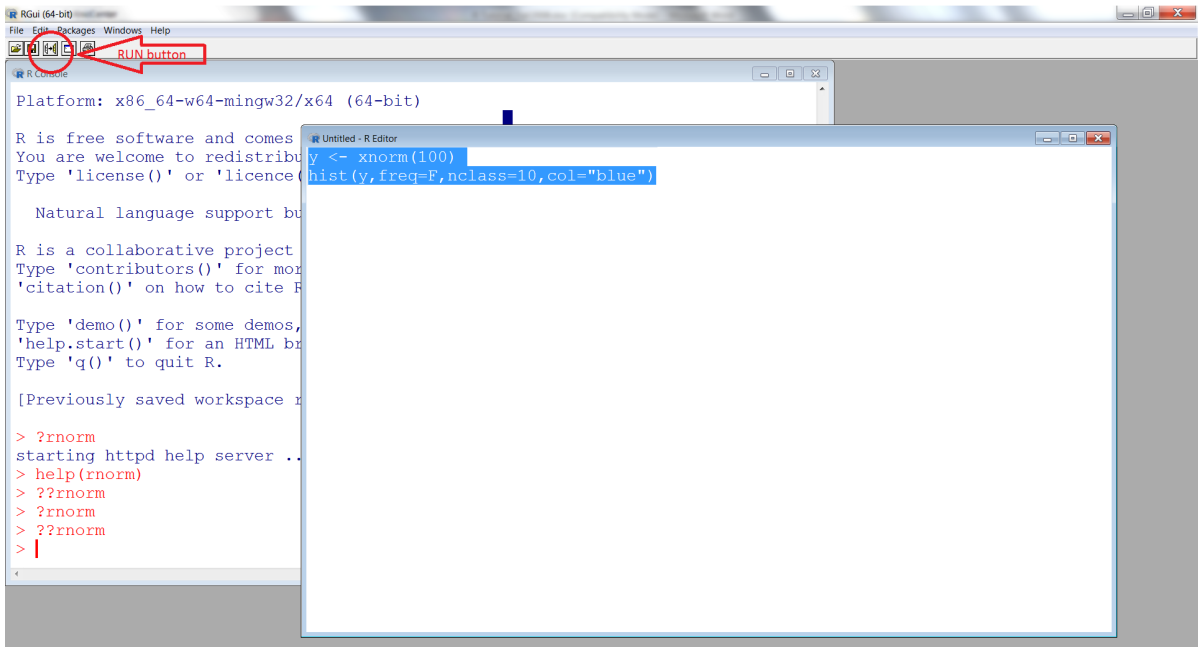
??rnorm

Writing a Program in R

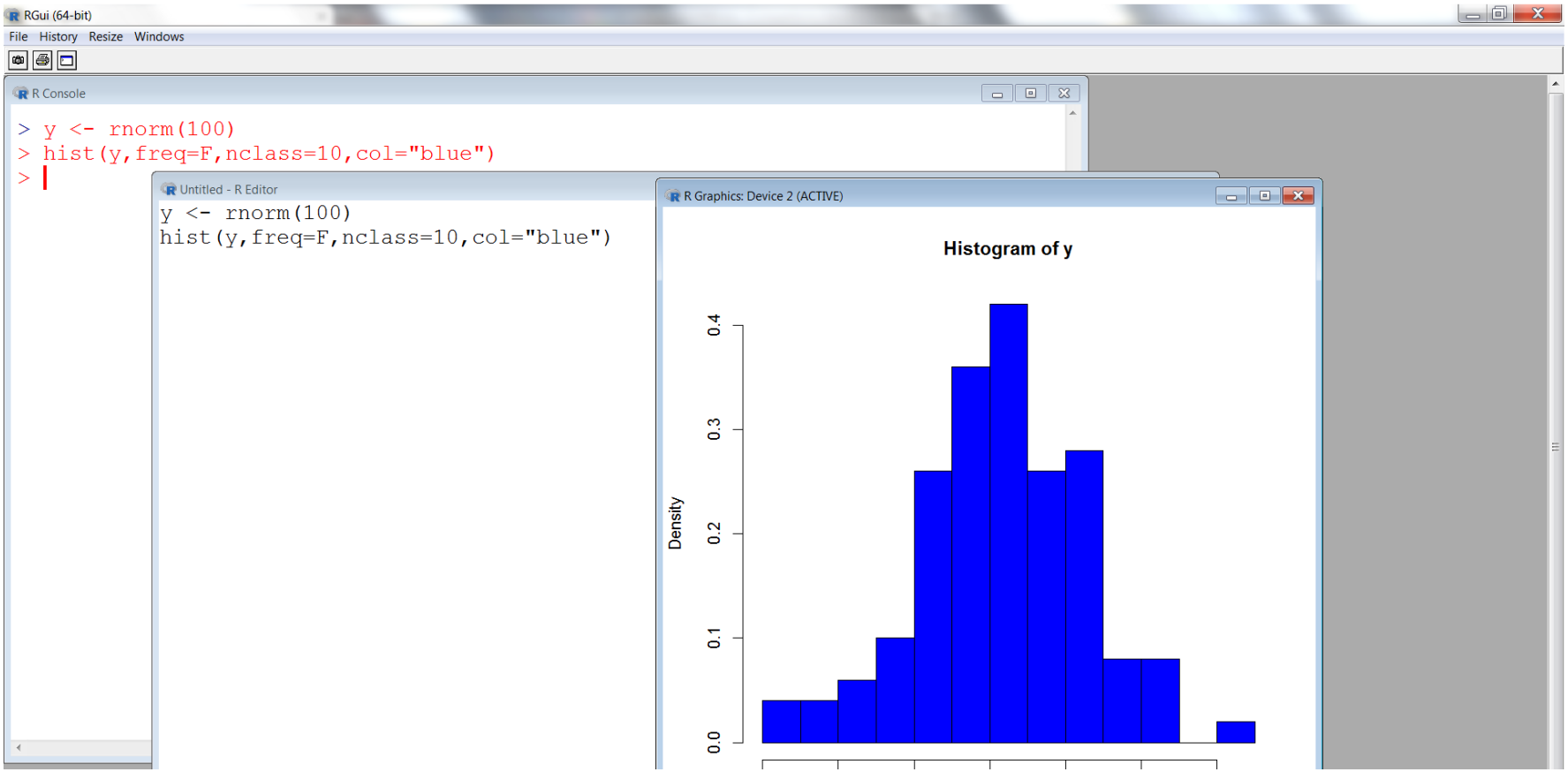
- To create a program, File $\longrightarrow$ New Script; Type the program into the editor

- To execute the program, select all program by CTRL+A, and then press the run button. A histogram will appear in the graphics window

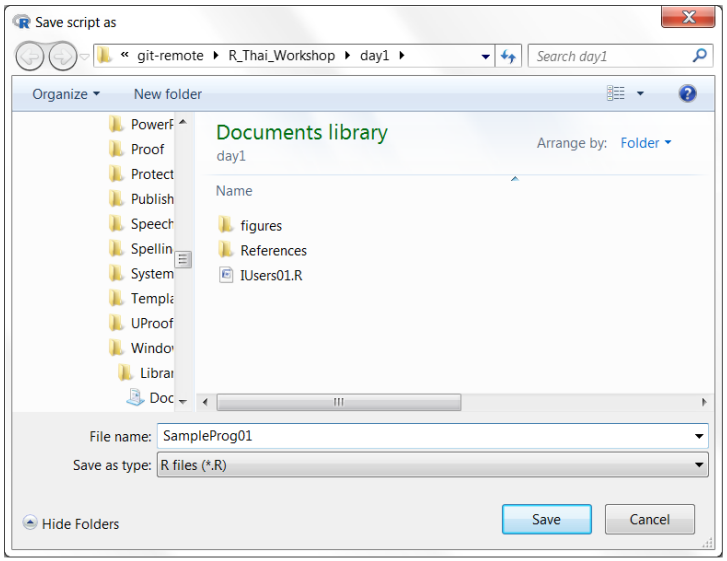
- To save the R script, either click on the Save icon or from selecting the file menu: File $\longrightarrow$ Save as to have the Save as dialogue box. Follow the prompts to save the script file.



Running an R Program



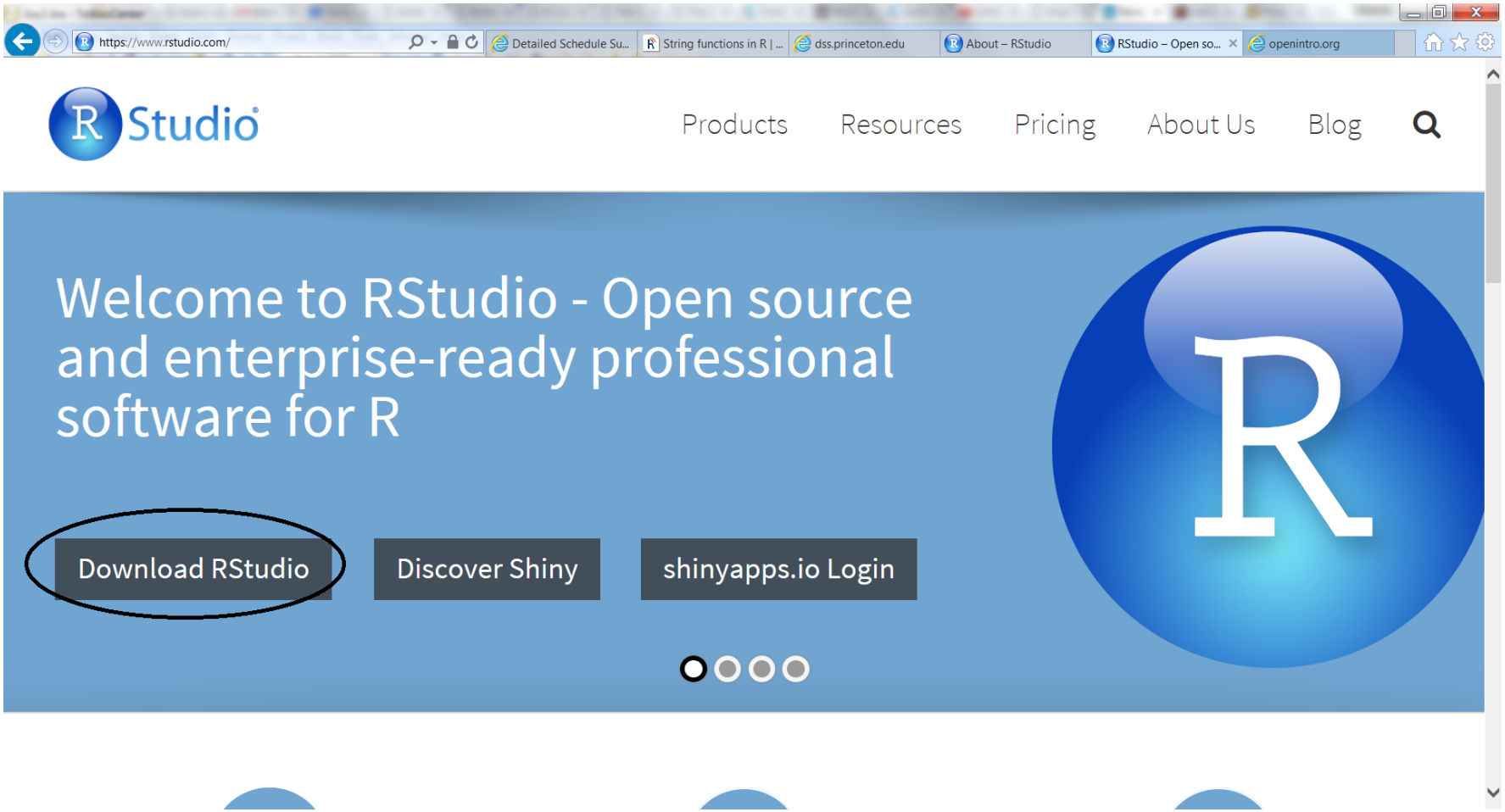
Saving a Program as .R file (Script)



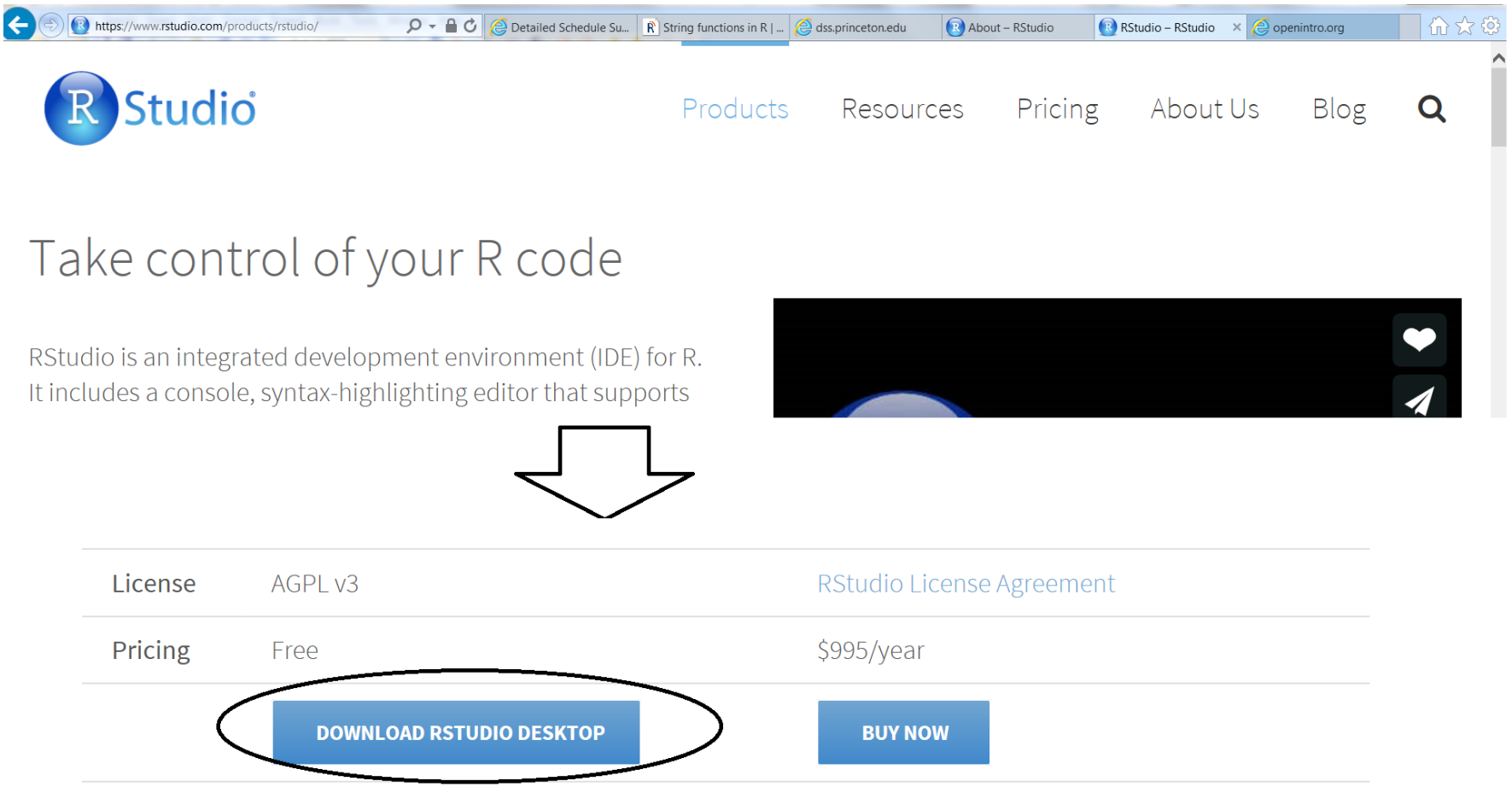
About R Studio

- RStudio is an integrated development environment (IDE) for R.
- RStudio allows the user to run R in a more user-friendly environment.
- Can be downloaded from http://www.rstudio.com/ either open source or commercial editions
- Runs on the desktop (Windows, Mac, and Linux) or in a browser connected to RStudio Server
- RStudio looks and feels the same regardless of the platform
- Created by J.J. Allaire in 2008. Supported by Foundation for Open Access Statistics (FOAS) and R Consortium
- To start either click on the RStudio icon or by type library("Rstudio")

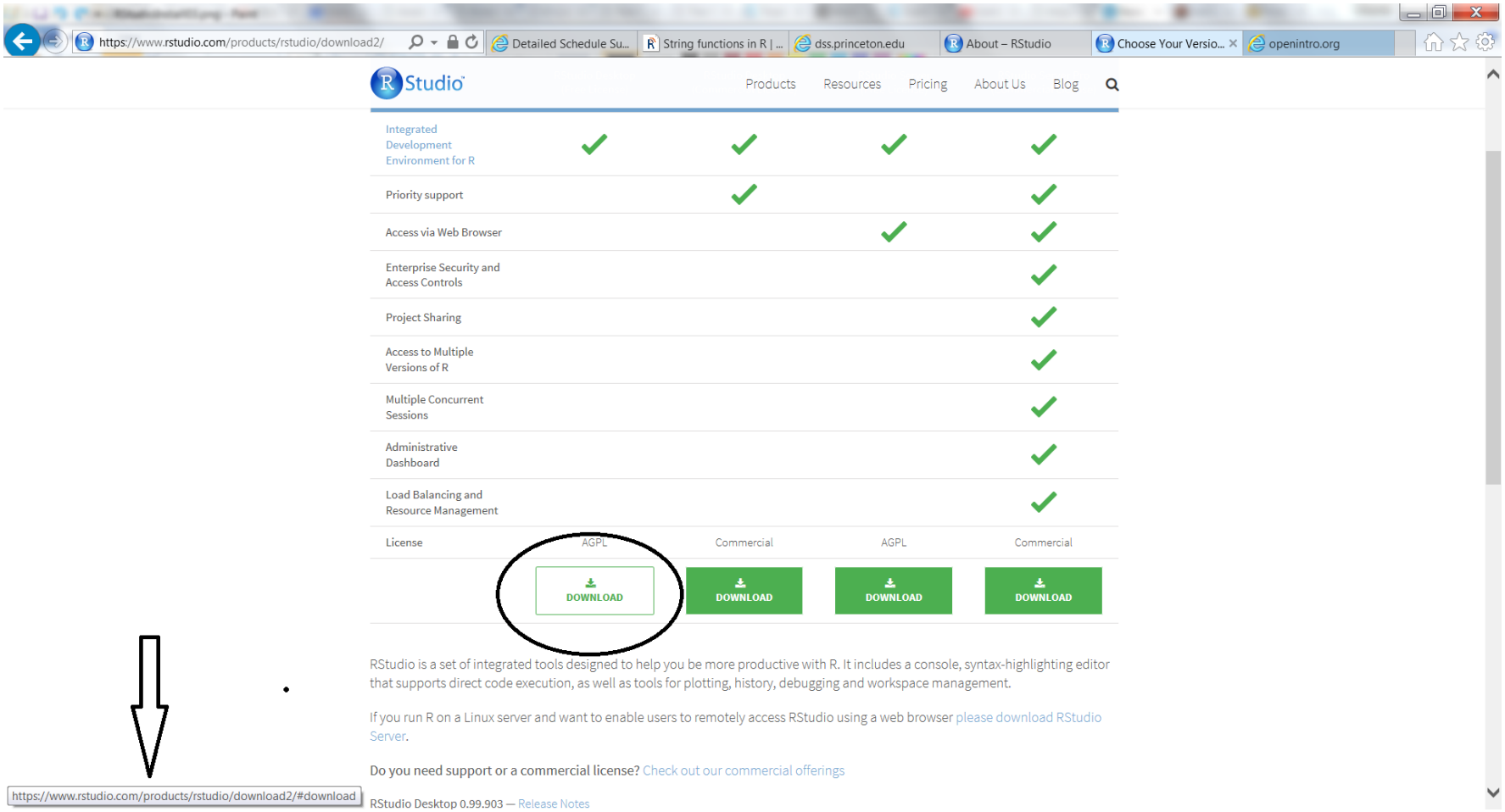
Installing RStudio



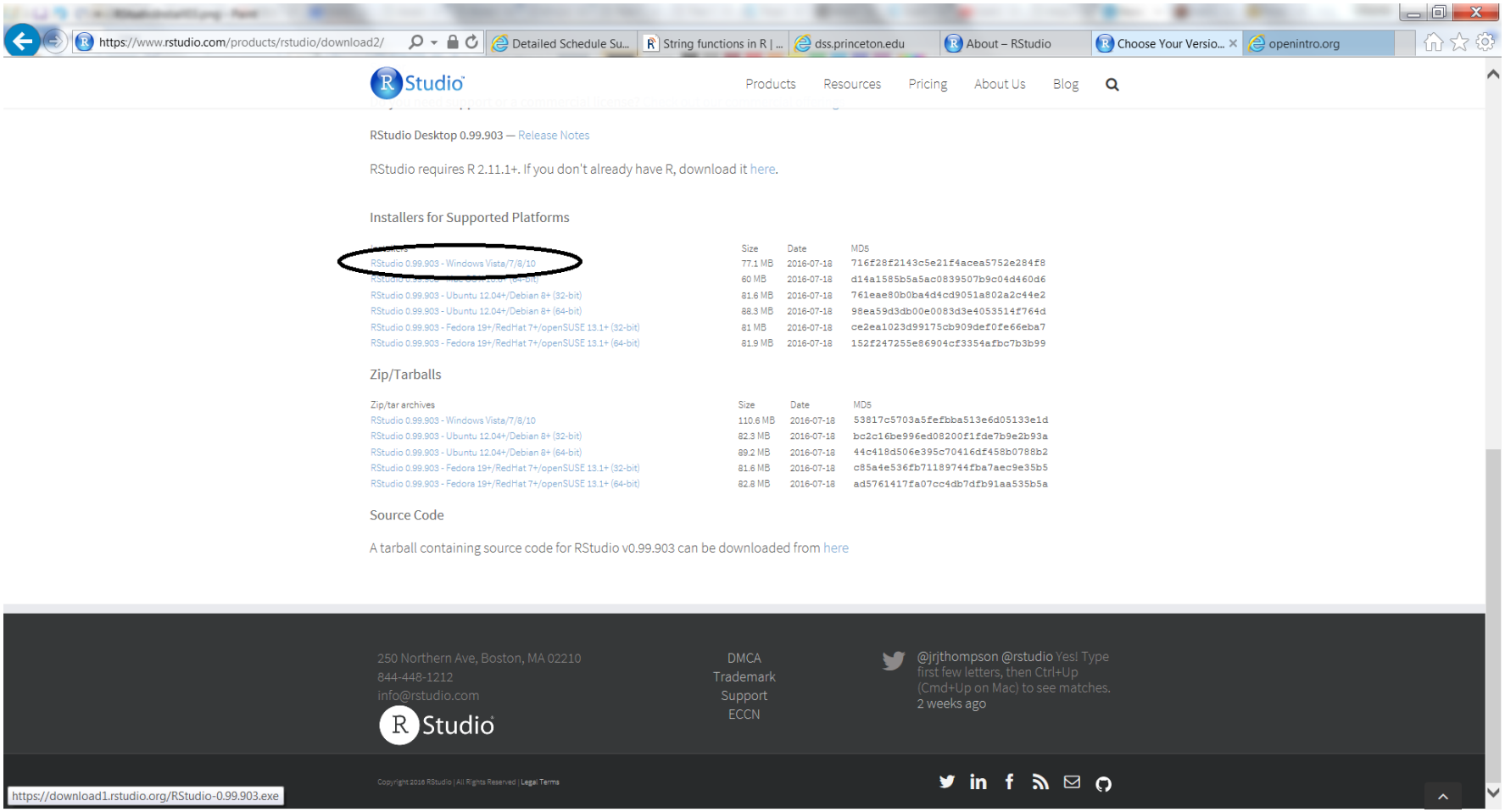
Installing RStudio



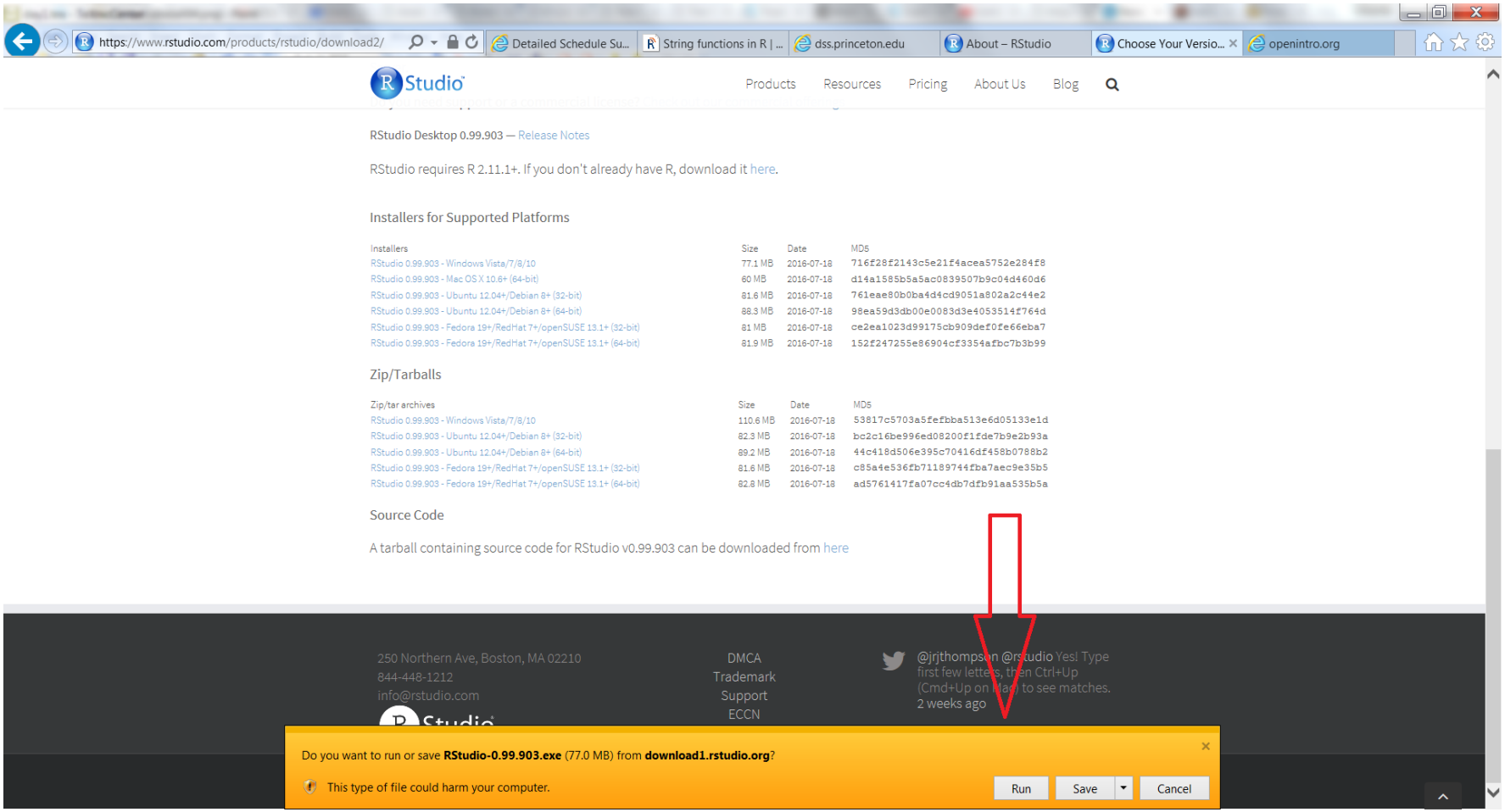
Installing RStudio



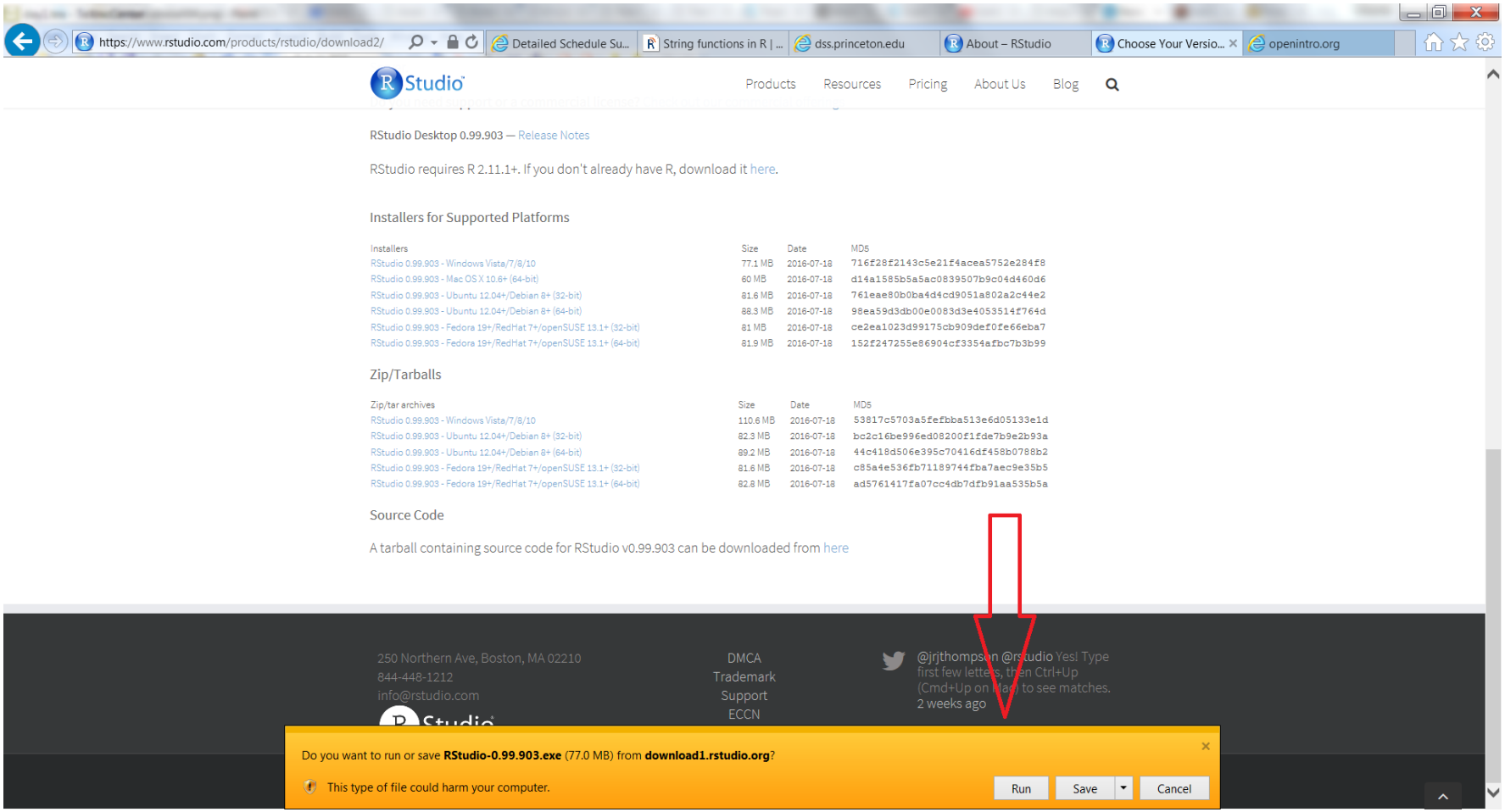
Installing RStudio



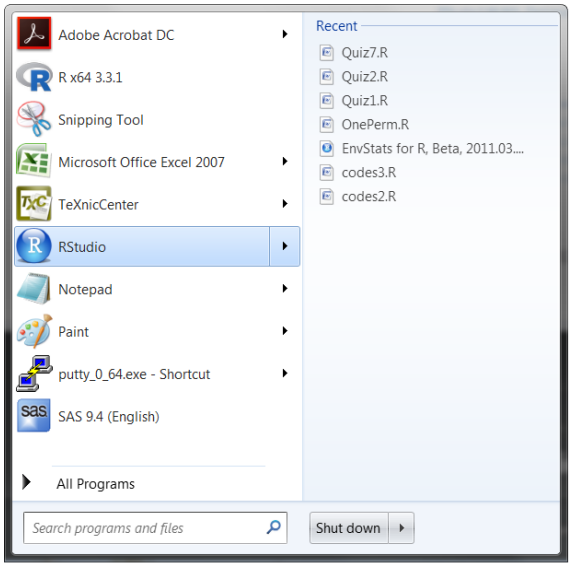
Installing RStudio



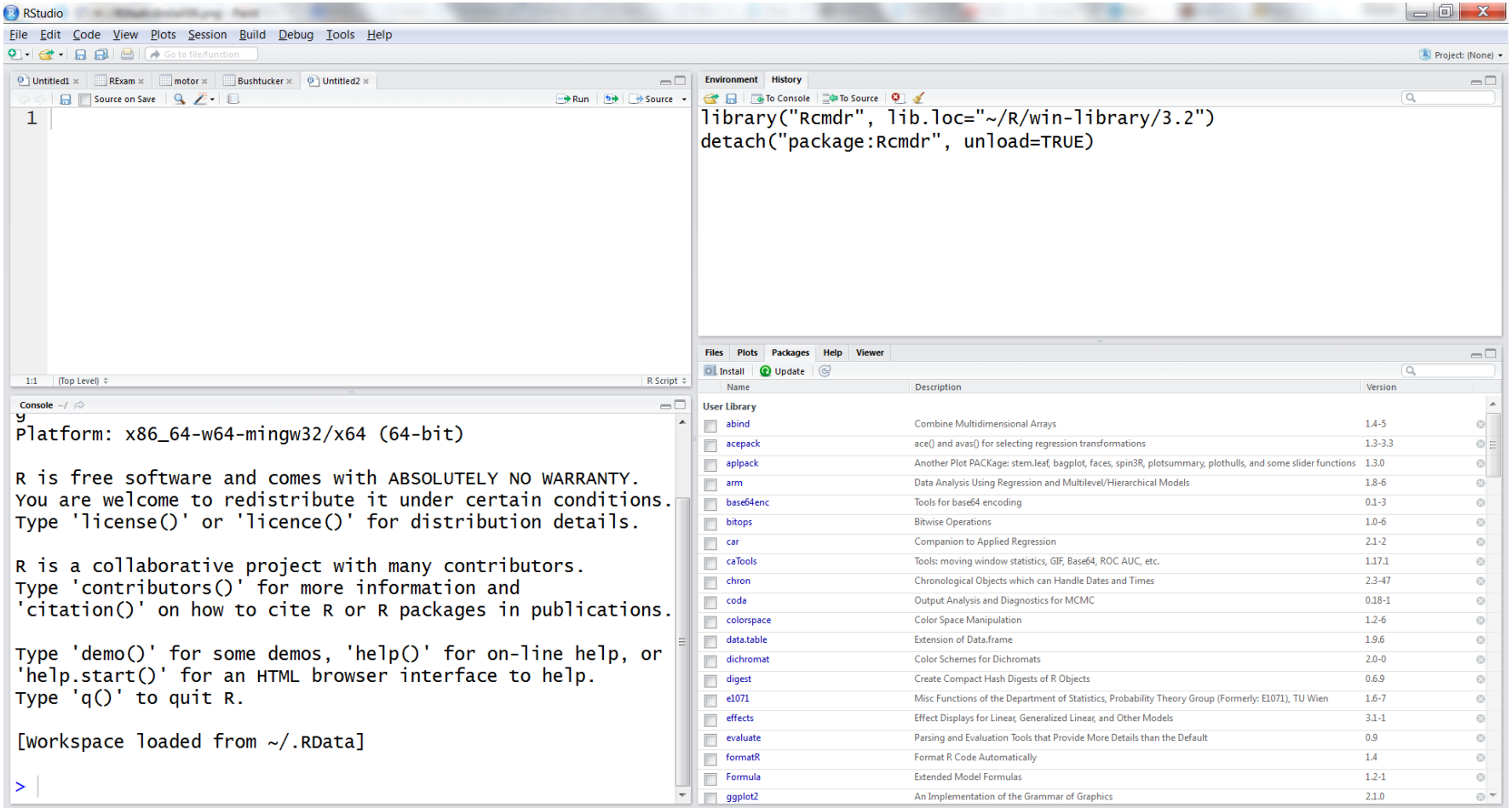
Installing RStudio



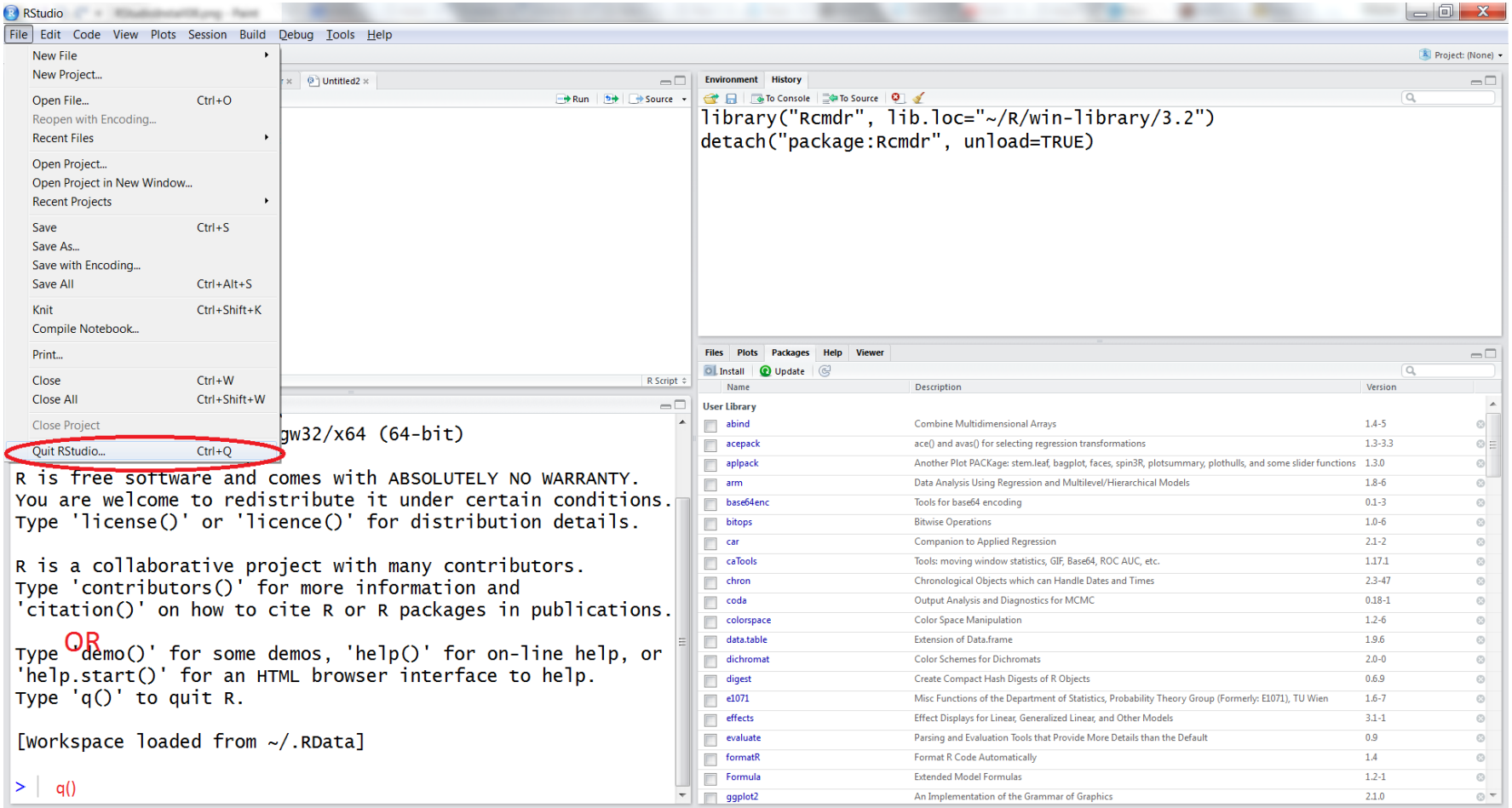
Starting RStudio



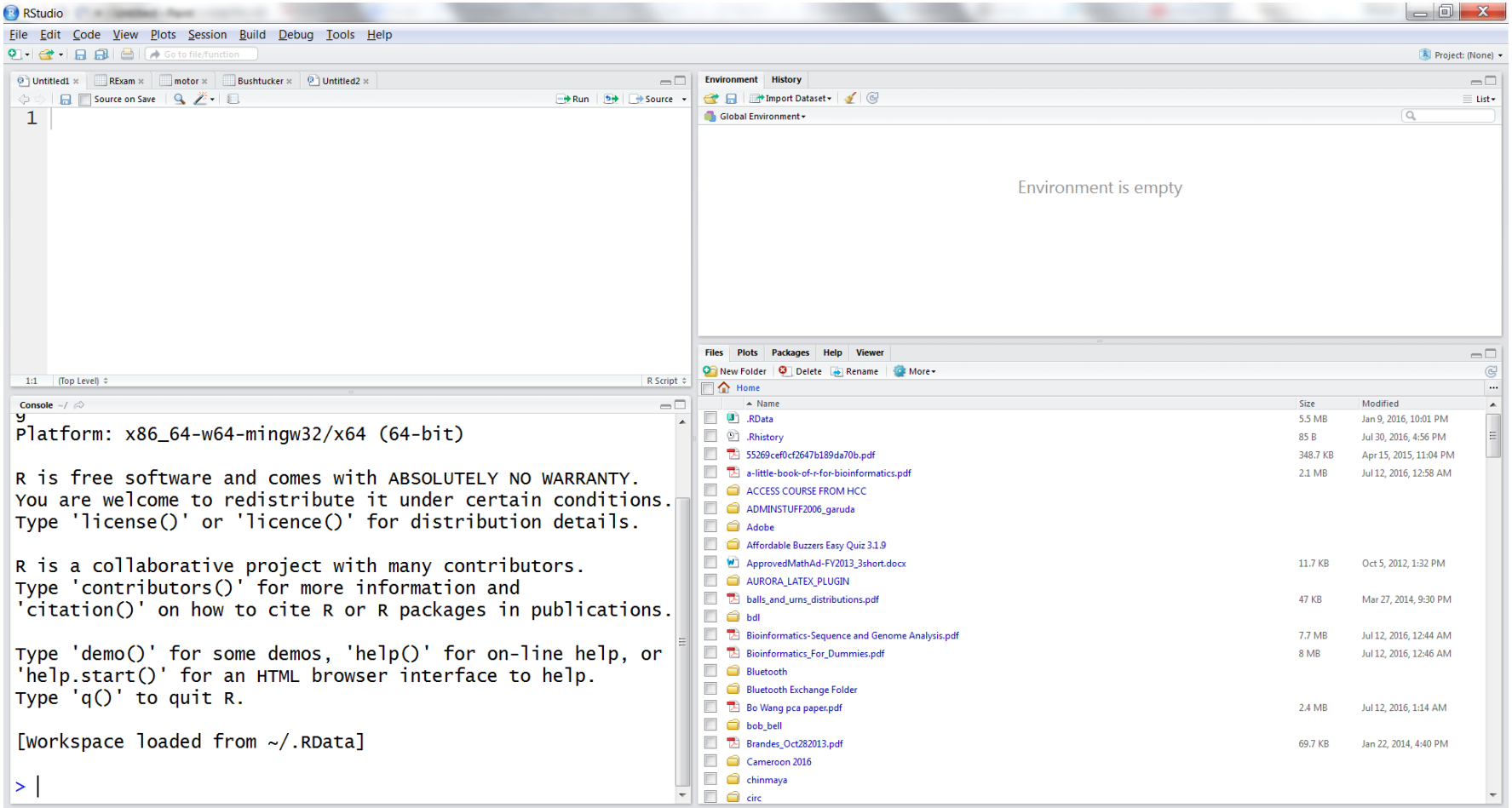
RStudio Dashboard



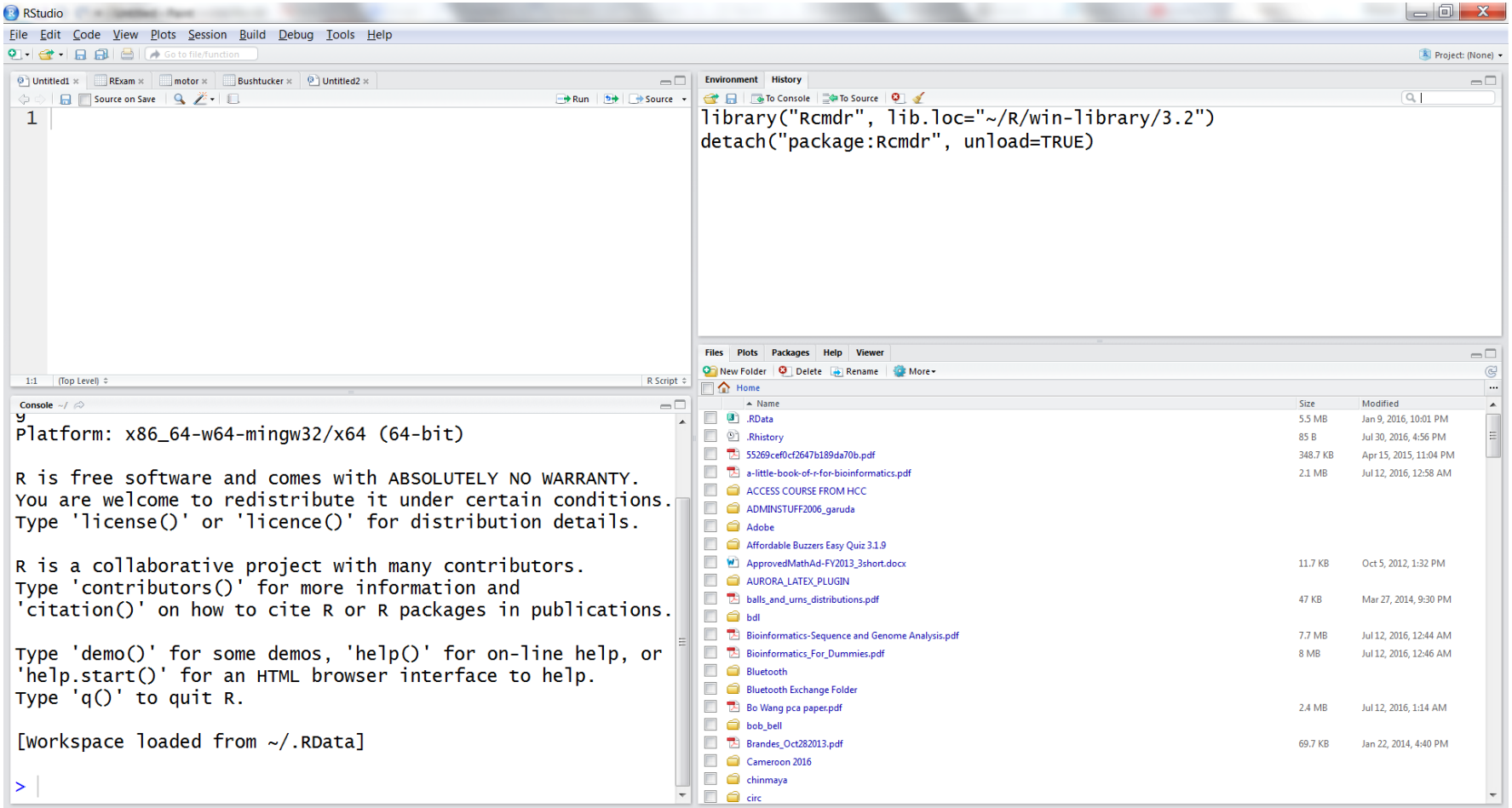
Quitting RStudio



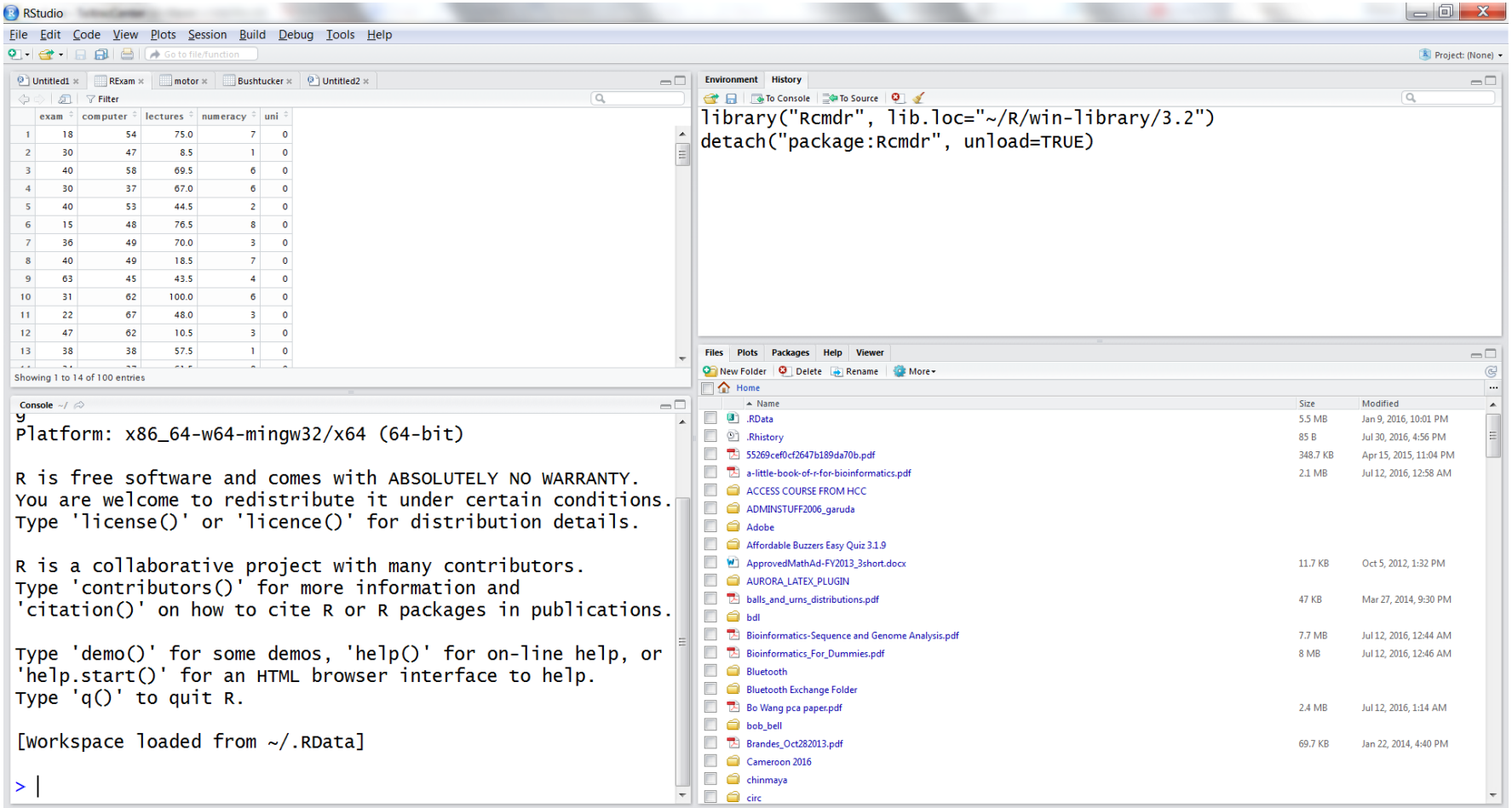
RStudio Dashboard: Windows and Panes



RStudio Dashboard: Windows and Panes



RStudio Dashboard: Windows and Panes



RStudio Dashboard: Windows and Panes

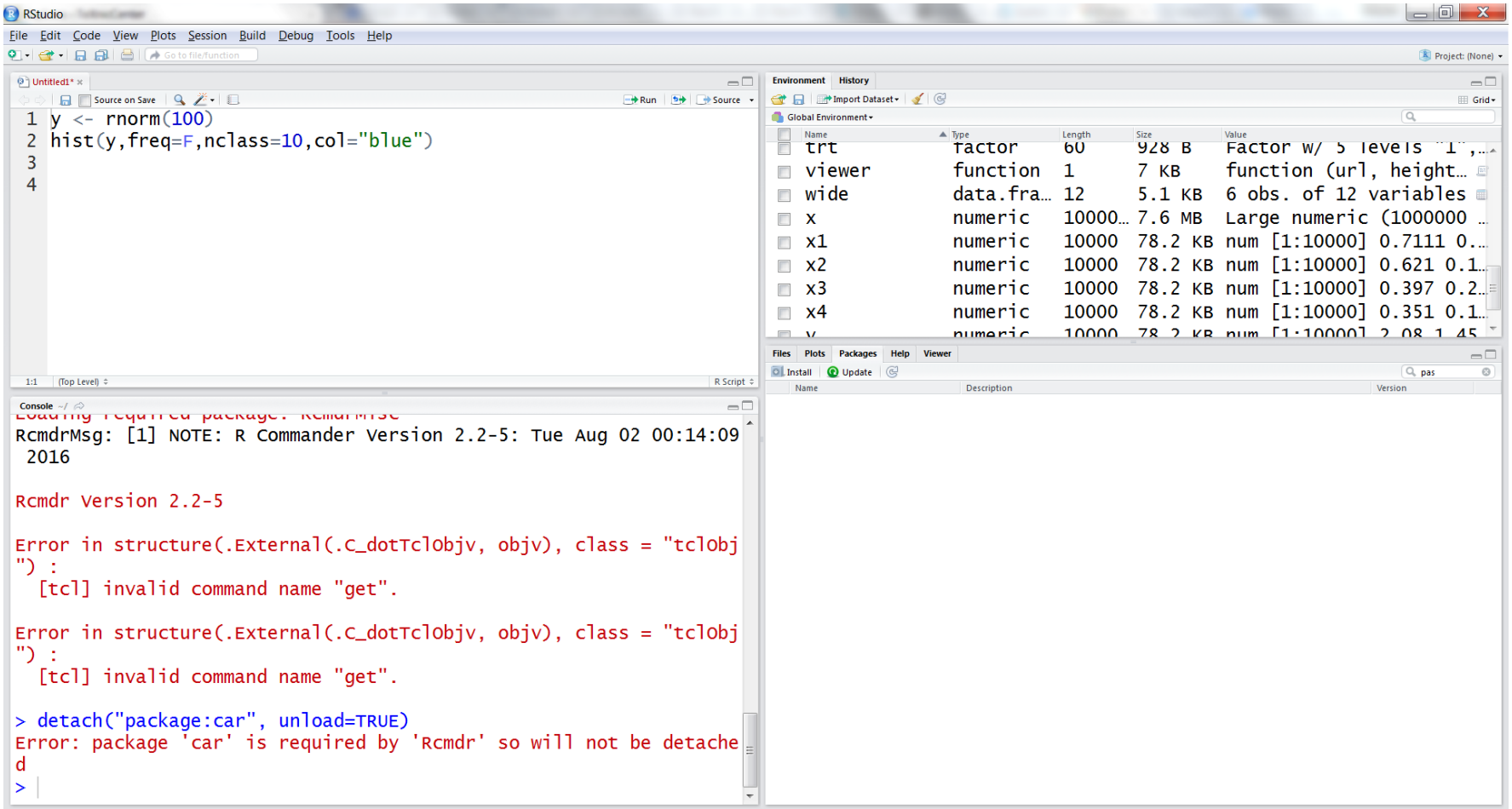
Usually, RStudio Dashboard has four windows: 1) Console. 2) Workspace and history. 3) Files, plots, packages and help. 4)The R script(s) and data view. Each window has one or more panes (or tabs).

- R commands are entered and submitted to run in the **Console**
- **Environment** pane shows all the active objects. The **history** pane shows a list of commands used so far.
- **R Script or Source Editor** is used for creating scripts and save it for later or recalling and running a saved script or also to view R objects
- The **Packages** shows all the packages that are available to be loaded. This pane also provides *search* and *install*.
- **Help** can be used to get help on functions and packages
- **Viewer** pane is used for viewing local web content

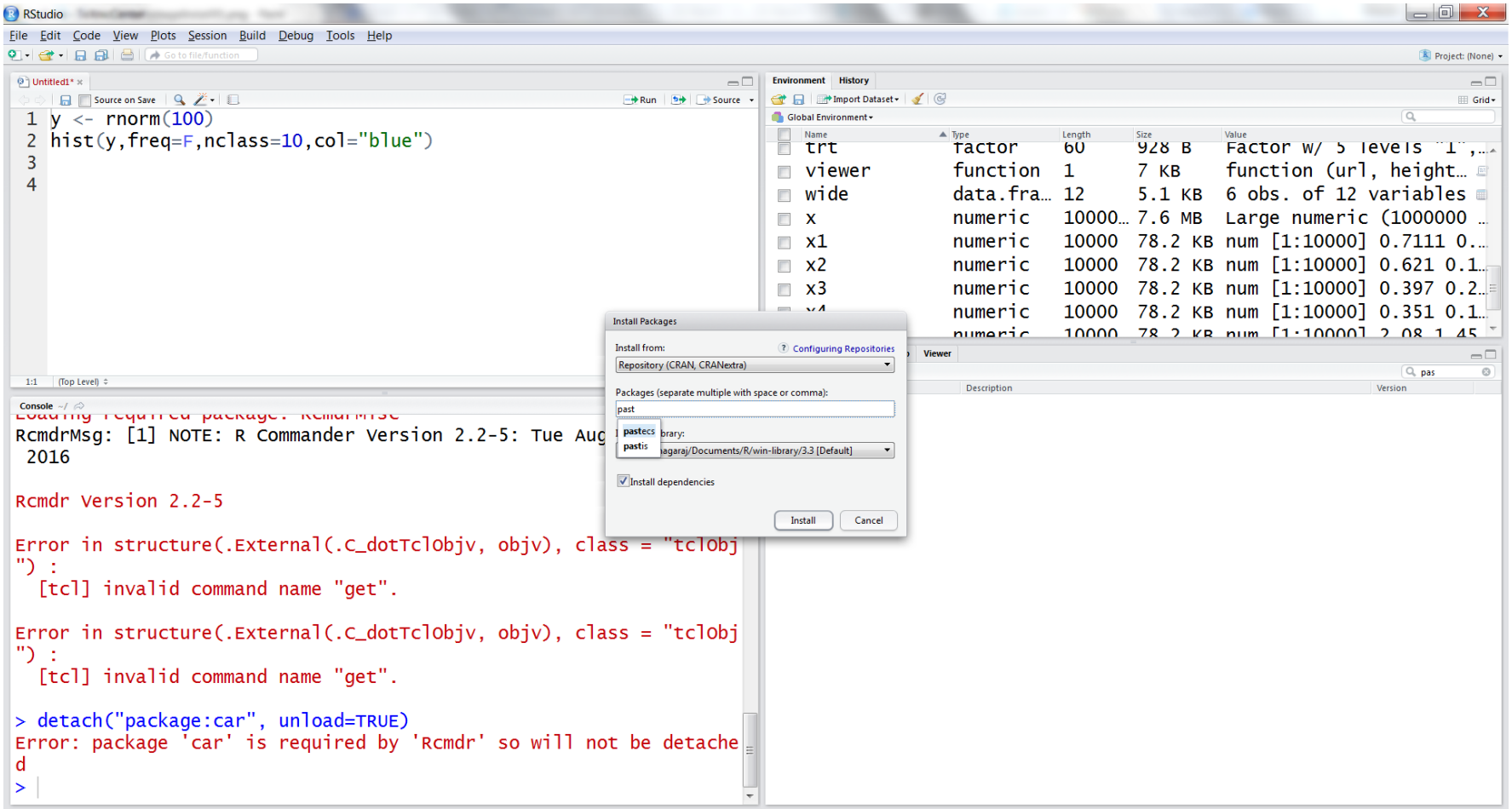
Installing and Loading a package to RStudio

- Activate the pane **packages**
- Enter the first few letters from the name of the package. RStudio will list the package(s) starting with those letters
- Check the package to be loaded
- If the package is not already installed, click on the install button and a dialog box will appear to walk you through the steps to install. Once installed, it will appear in this list to be loaded (or checked off).

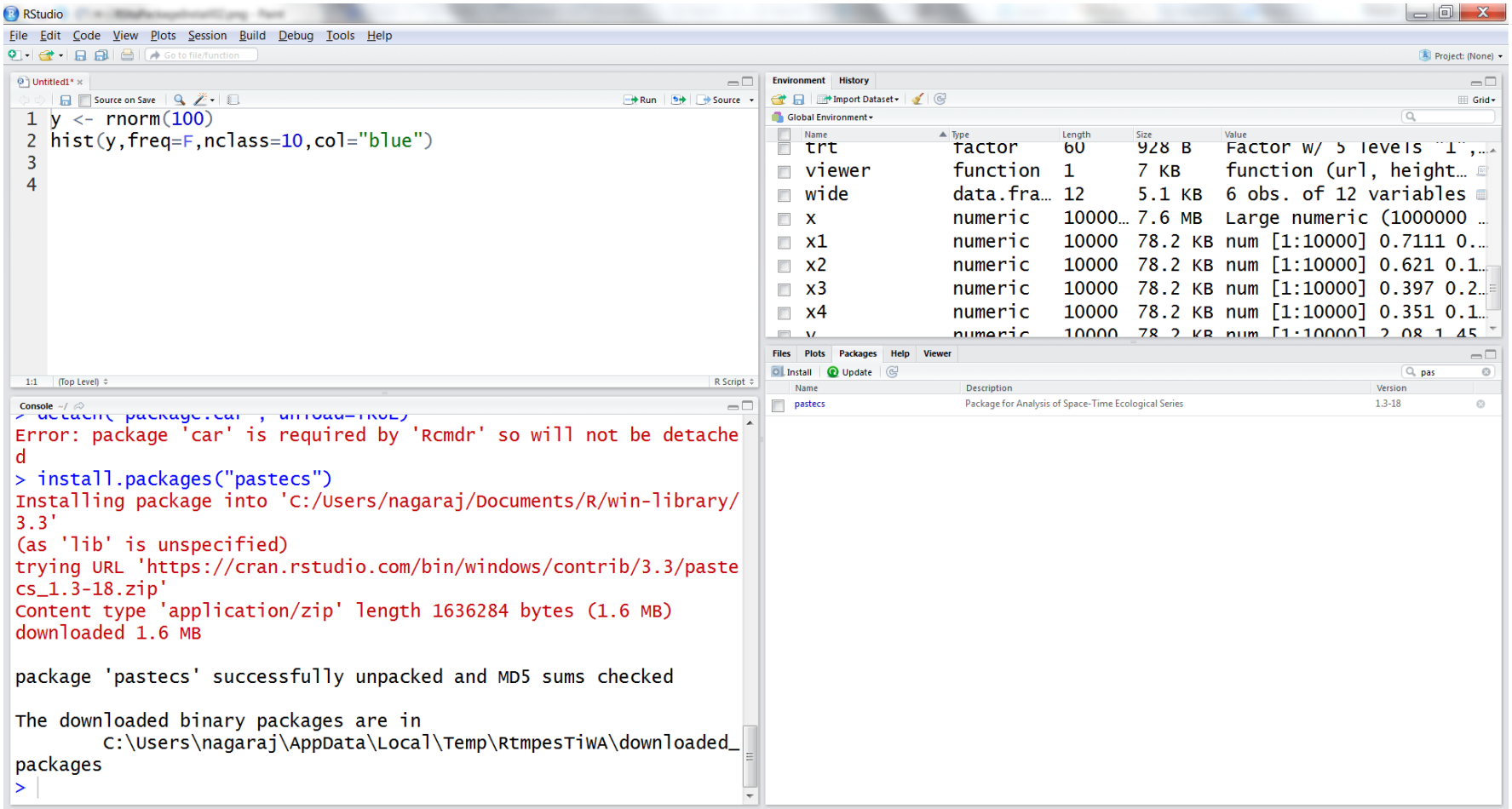
RStudio: Installing a package



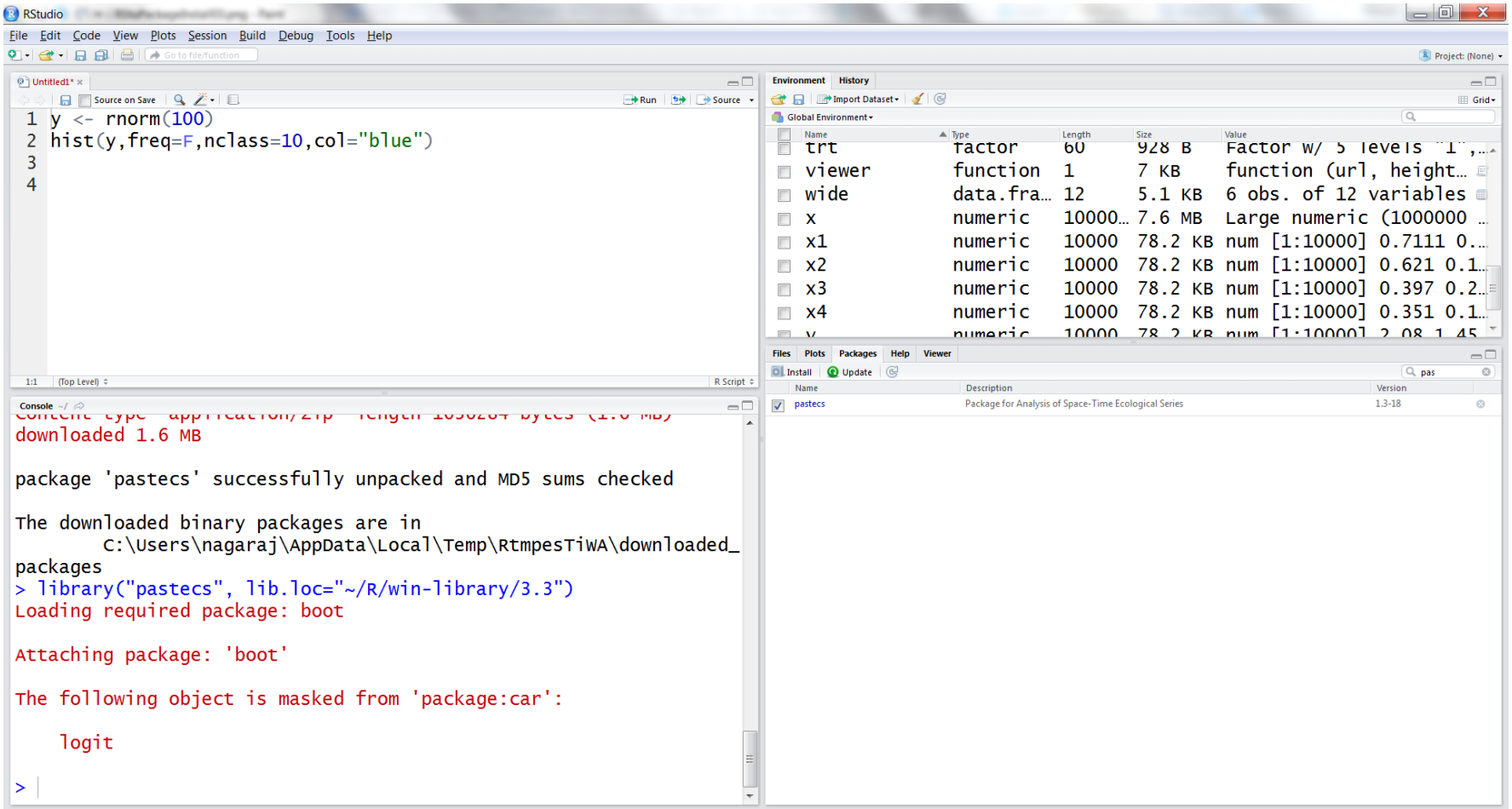
RStudio: Installing a package



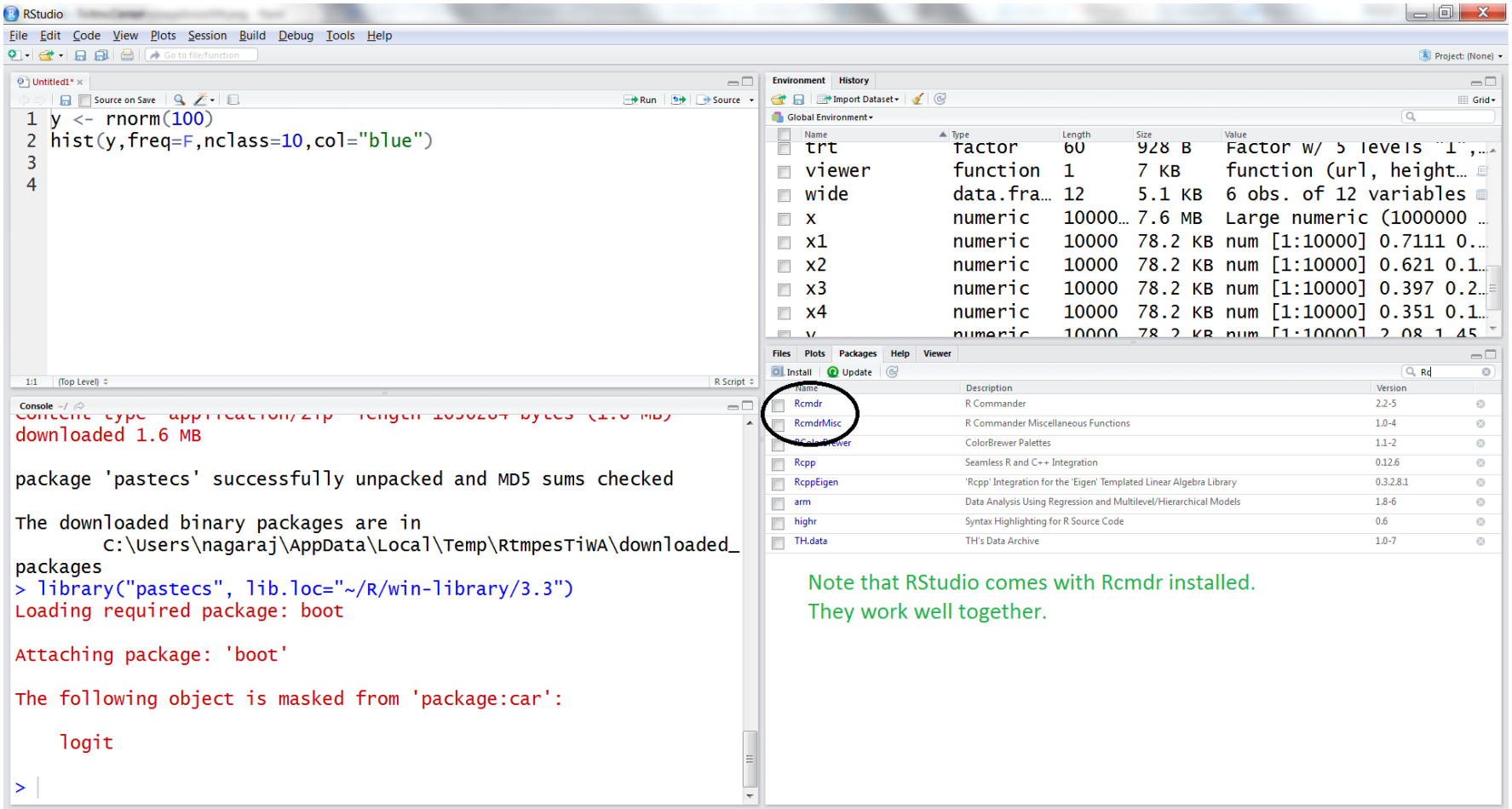
RStudio: Installing a package



RStudio: Installing a package



RStudio: Installing a package



RStudio Console

- Code completion
- Retrieving previous commands
- Console title bar – current working directory
- Using the command history
- Keyboard short cuts:

  Ctrl+1 - Move focus to the Source Editor
  Ctrl+2 - Move focus to the Console
  Ctrl+L - Clear the Console
  Esc - Interrupt R

Working inside RStudio

- To create a new file: File -> New File or Recent Files menu.
- To switch between many open files: View -> Switch to Tab...

  RStudio supports syntax highlighting and other specialized code-editing features for: R scripts, R Markdown documents, HTML files, TeX documents, etc
- Code completion, Find-and-Replace, Comment/Uncomment blocks of code
- Executing the code (Running a program)

  Select the lines and use the Run toolbar button
  To run the entire document use the Source toolbar button.
- Extract a function

  RStudio can automatically convert segments of codes into a function
  Highlight a block of code and use Code -> Extract Function
- Code Folding and Sections

RStudio is an IDE

- RStudio is an Integrated Development Environment (IDE)
- It is not a Graphical User Iterface (GUI) for statistical analysis
- Note that RStudio has a lot of cool tools for managing script and output files, but does not have a menu for computing descriptive statistics
- We will now discuss a couple of packages which convert R into a GUI based statistical analysis system

Useful Tips for Session Management

- Exiting and saving the script:
  - Helps keep track of the session
  - A saved script can be opened in a new session to sart from the last point of save
  - Better option than saving the workspace

- Saving and printing the output
  - Cut (ctrl-c) -and- paste (ctrl-v) output (ongoing) into an open [word/text] report document
  - Best practice is to write notes as the output is moved to the report
  - Graphs may also be similarly pasted or use File $>$ Save as method described earlier

- Commands generated by the R Commander appear in the script window:
  - This window can be edited so that only those commands that worked are saved
  - To send this script you have to highlight the relevant text and press the 'Submit' button.

Basic Programming

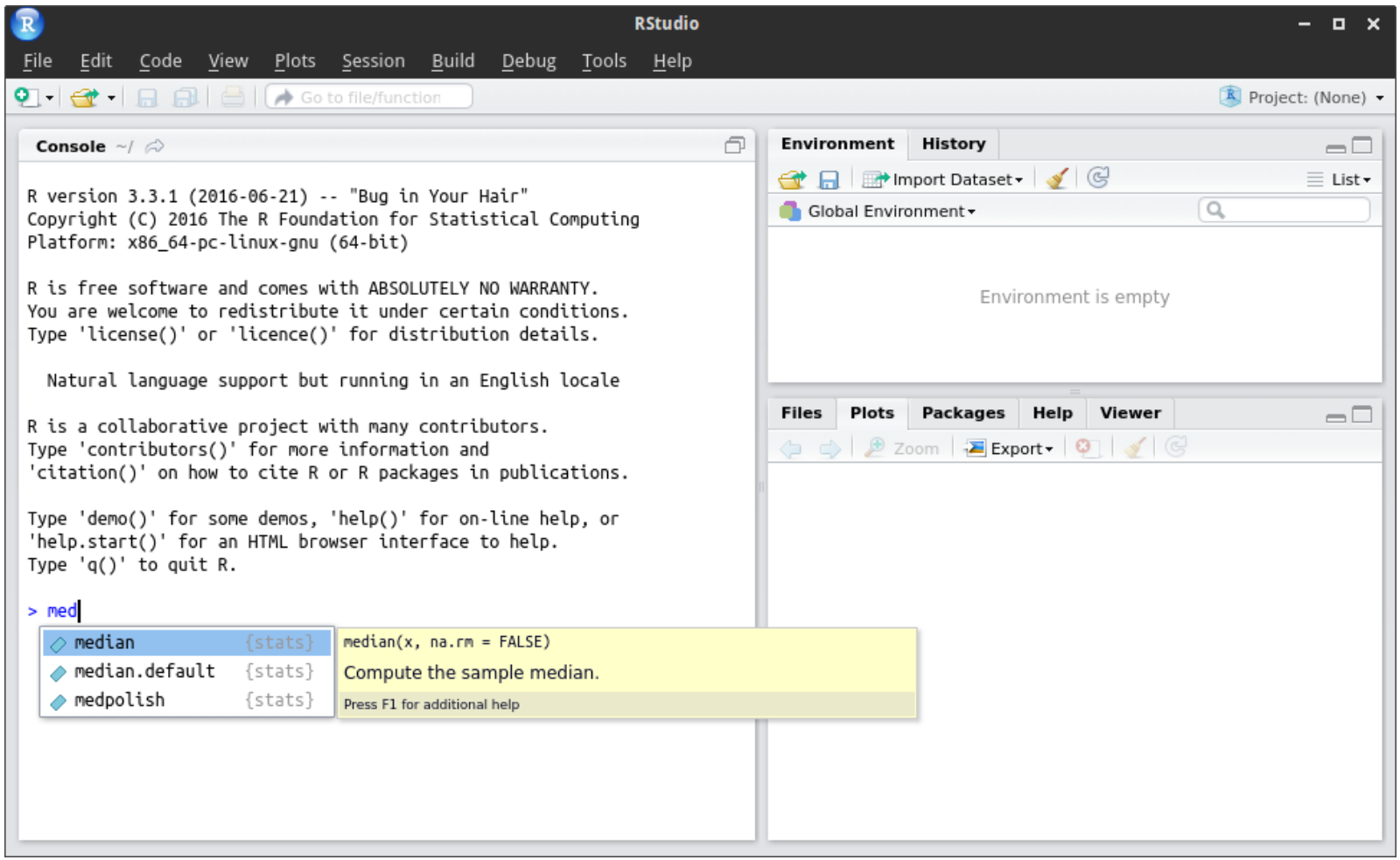
- When launching R we will see a prompt similar to below.

```
R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 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.

> print("hello world")
[1] "hello world"
>
```

- We enter commands after the > character.

Command Completion

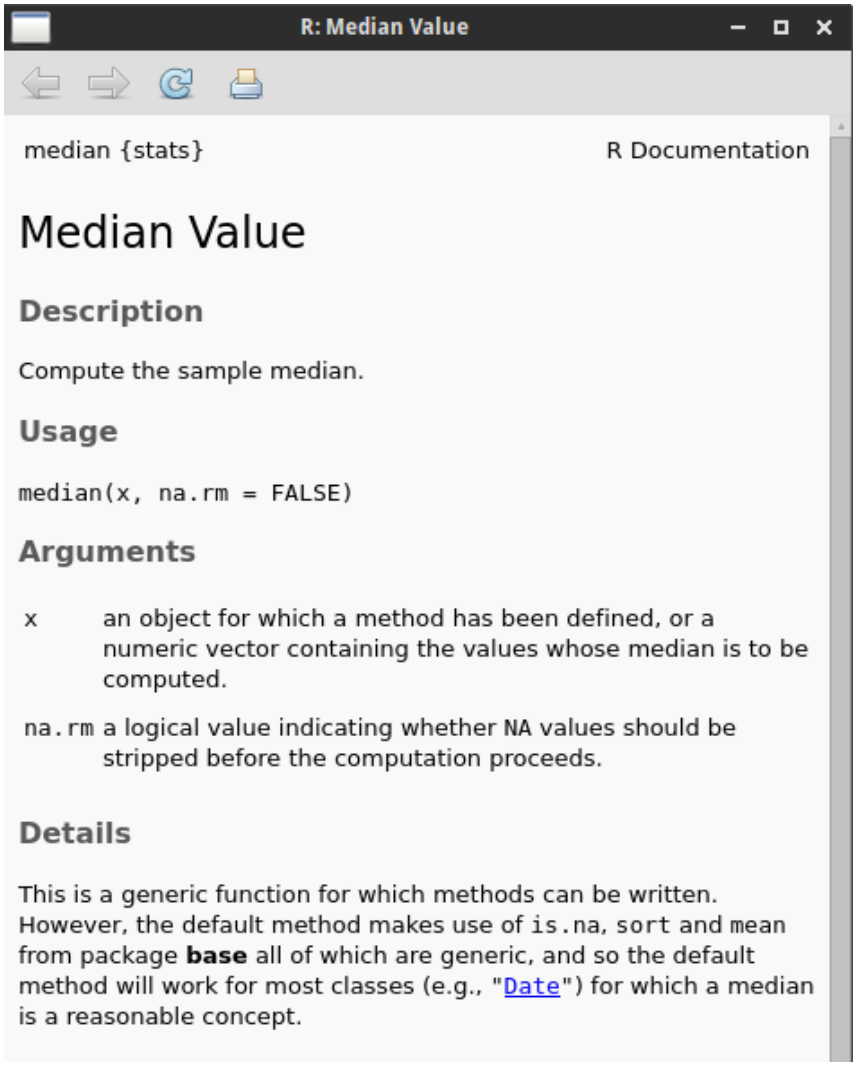
- Use tab or ctrl+space to get inline suggestions while you program.

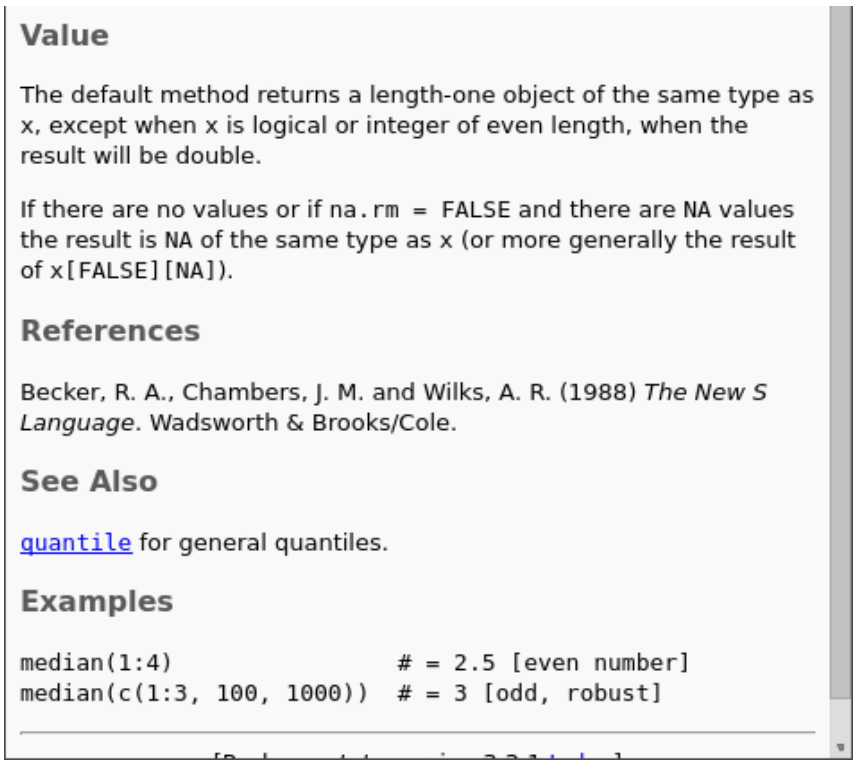


- Sometimes suggestions will be offered as you type normally.

Help System

```
> ?median
```





Arithmetic

- Basic arithmetic

```
> 2 + 3
[1] 5
> 2 - 3
[1] -1
> 2 * 3
[1] 6
> pi / 2
[1] 1.570796
> 2^3
[1] 8
> 21 %% 10
[1] 1
> 2 < 3
[1] TRUE
```

- Built-in functions

```
> exp(1)
[1] 2.718282
> log(10)
[1] 2.302585
> log2(10)
[1] 3.321928
> abs(-1 * 1)
[1] 1
> sqrt(5)
[1] 2.236068
```

Sequences

- Generate a sequence of integers.

```
> 1:9
[1] 1 2 3 4 5 6 7 8 9
> seq(1, 9)
[1] 1 2 3 4 5 6 7 8 9
> seq(from = 1, to = 9)
[1] 1 2 3 4 5 6 7 8 9
> seq(from = 1, to = 9, by = 2)
[1] 1 3 5 7 9
```

- `1:9` is a shorthand for `seq(1,9)`.
- Argument `by` has a default of 1.
- We can specify arguments with or without a label. Arguments without labels can be given in any order.
- Use parentheses to call a function, even if there are no arguments.
- If function name is used without parentheses, `R` will display its definition.

```
> seq
function (...)
UseMethod("seq")
<bytecode: 0x623dc18>
<environment: namespace:base>
```

Variables

- The result of an expression can be assigned it to a named variable with the <- or = operators.

```
> x <- 2
> y <- x + 3
> x
[1] 2
> y
[1] 5
> x <- x + 1
> x
[1] 3
> y
[1] 5
```

- Variable names can be composed from letters, digits, '_', and '.'.
- Some configurations are illegal; e.g. a '.' followed by a digit.
- Some names such as `c`, `q`, `t`, and `median` are already defined. `R` will not prevent them from being reused, but your code might become confusing.

```
> q <- 1
> q
[1] 1
> q()
Save workspace image to ~/.RData? [y/n/c]:
```

Vectors

- A vector is an one-dimensional array of scalars.
- Three basic functions for constructing vectors are
  1. `c(...)` (combine),
  2. `seq(from, to, by)` (sequence),
  3. `rep(x, times)` (repeat).

```
> print(x <- seq(1, 20, by = 2))
 [1]  1  3  5  7  9 11 13 15 17 19

> print(y <- rep(3, 4))
[1] 3 3 3 3

> print(z <- c(y, x))
 [1]  3  3  3  3  1  3  5  7  9 11 13 15 17 19
```

- The : operator takes precedence over operators such as + and −. Use parentheses when in doubt, or to. improve readability

```
> n <- 10
> print(x <- 1:n+1)
 [1]  2  3  4  5  6  7  8  9 10 11
> print(x <- 1:(n+1))
 [1]  1  2  3  4  5  6  7  8  9 10 11
```

Indexing into Vectors

```
> x <- seq(10,20)
> x[9]
[1]  18

> x[12]          # Past the end
[1] NA

> x[2:5]
[1] 11 12 13 14

> ind <- c(1,3,5)
> x[ind]
[1] 10 12 14

> x[-1]
 [1] 11 12 13 14 15 16 17 18 19 20

> x[-c(1,3,5,7)]
[1] 11 13 15 17 18 19 20

> x[x > 11]
[1] 12 13 14 15 16 17 18 19 20

> ind <- which(x > 11)
> ind
[1]  3  4  5  6  7  8  9 10 11
> x[ind]
[1] 12 13 14 15 16 17 18 19 20
```

Arithmetic on Vectors

```
> x <- c(1, 2, 3)
> y <- c(4, 5, 6)

> x + y             # Most operations are elementwise.
[1] 5 7 9

> y - x
[1] 3 3 3

> x * y
[1]  4 10 18

> y / x
[1] 4.0 2.5 2.0

> y^x
[1]   4  25 216

> 1:4 + 1:2         # Unequal length: shorter vector is repeated.
[1] 2 4 4 6

> 1 + 1:10          # Scalar plus vector
 [1]  2  3  4  5  6  7  8  9 10 11

> c(1,2,3) + c(1,2)
[1] 2 4 4
Warning message:
In c(1, 2, 3) + c(1, 2) :
  longer object length is not a multiple of shorter object length
```

Some Useful Functions on Vectors

```
> x <- c(5,4,3,6,7,8,9,10,1,2)
> sort(x)
 [1]  1  2  3  4  5  6  7  8  9 10

> x <- c(1,2,3,4,5,6,7,8,9,10)
> rev(x)
 [1] 10  9  8  7  6  5  4  3  2  1

> x <- seq(1,10)
> mean(x)
[1] 5.5
> sum(x)
[1] 55
> prod(x)
[1] 3628800
> y <- sample(x)
> min(y)
[1] 1

> # Draw a sample (with or without replacement) from x at random
> x <- seq(1,10)
> sample(x)
 [1]  5  7  1  9 10  4  8  6  3  2
> sample(x, size = 5)
[1] 3 1 6 8 9
> sample(x, replace = TRUE)
 [1]  9  3  2  1  3  8 10  7  5  7
> sample(x, replace = TRUE, size = 5)
[1] 10  4  6  5  2
```

Other scalar types

- Complex numbers

```
> a <- complex(5, 3); b <- complex(7, 2)
> a + b
[1] 12+5i
```

- Logicals

```
> x <- c(TRUE, TRUE, FALSE); y <- c(TRUE, FALSE, FALSE)
> x & y
[1]  TRUE FALSE FALSE
> !x
[1] FALSE FALSE  TRUE
> c(sum(x), prod(x))
[1] 2 0
```

- Strings

```
> a <- "my string"
> length(a)
[1] 1
> nchar(a)
[1] 9

> x <- 5
> cat("The answer is", x, ".\n")
The answer is 5 .

> str <- sprintf("The answer is %d.", x)
> print(str)
[1] "The answer is 5."
```

Other scalar types

- Factors represent categorical data. They give descriptive labels, and also emphasize that values are not numbers.

```
> x <- sample(1:3, size = 10, replace = TRUE)
> x
 [1] 3 1 3 1 1 3 3 2 2 2
> y <- factor(x, labels=c("urban", "suburban", "rural"))
> y
 [1] rural    urban    rural    urban    urban    rural    rural
 [8] suburban suburban suburban
Levels: urban suburban rural
> levels(y)
[1] "urban"    "suburban" "rural"
```

- Splitting numerical data into intervals, using the cut function.

```
> x <- rnorm(10)
> x
 [1]  0.3866017 -0.5058934 -2.2927512 -0.7278835  2.6448711
 [6] -0.3614214  1.2178368  0.6046139  0.2895143 -0.9899157
> cut(x, 3)
 [1] (-0.649,1] (-0.649,1] (-2.3,-0.649] (-2.3,-0.649]
 [5] (1,2.65]   (-0.649,1] (1,2.65]      (-0.649,1]
 [9] (-0.649,1] (-2.3,-0.649]
Levels: (-2.3,-0.649] (-0.649,1] (1,2.65]
```

Special scalar types

- Some scalar values with special meanings.
  - `NA` means that no value has been assigned, or the value is missing
  - `NaN` means *not a number*
  - `NULL` represents something which is undefined
  - `Inf` represents infinity
- Functions to check for these states.

```
> y <- log(0)
> y
[1] -Inf
> is.infinite(y)
[1] TRUE
> is.na(y)
[1] FALSE

> x <- log(-1)
> is.na(x)
[1] TRUE

> z <- c(1, 2, 3, NA, 4)
> is.na(z)
[1] FALSE FALSE FALSE  TRUE FALSE
> mean(z)
[1] NA
> mean(z, na.rm = TRUE)
[1] 2.5
```

Rounding error

- To avoid issues due to rounding error, use `all.equal` to test for numerical equality.

```
> 2 * 2 == 4
[1] TRUE

> sqrt(2) * sqrt(2) == 2
[1] FALSE

> all.equal(sqrt(2) * sqrt(2), 2)
[1] TRUE
```

Matrices

```
> matrix(NA, nrow = 3, ncol = 3)
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
[3,]   NA   NA   NA

> matrix(c(1,2,3,4), nrow = 2, ncol = 2)
     [,1] [,2]
[1,]    1    3
[2,]    2    4

> matrix(c(1,2,3,4), nrow = 2, ncol = 2, byrow = TRUE)
     [,1] [,2]
[1,]    1    2
[2,]    3    4

> A <- diag(5, 3)          # A diagonal matrix with custom labels
> rownames(A) <- c("First Row", "Second Row", "Third Row")
> colnames(A) <- c("A", "B", "C")
> print(A)
              A    B    C
First Row     5    0    0
Second Row    0    5    0
Third Row     0    0    5
> nrow(A)
[1] 3
> ncol(A)
[1] 3
```

Indexing Matrices

```
> A <- matrix(seq(1,9), nrow = 3, ncol = 3)
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> A[2,3]        # Access some specific elements
[1] 8
> A[3,2]
[1] 6

> A[1,]         # First row
[1] 1 4 7

> A[,2]         # Second column
[1] 4 5 6

> A[c(1,2),]    # Rows 1 and 2
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8

> A[1,c(2,3)]  # Columns 2 and 3 of row 1
[1] 4 7
```

Matrix Operations

```
> A <- B <- matrix(seq(1,9), nrow = 3, ncol = 3)
> A + B
     [,1] [,2] [,3]
[1,]    2    8   14
[2,]    4   10   16
[3,]    6   12   18

> 3 * A          # Multiply by a scalar
     [,1] [,2] [,3]
[1,]    3   12   21
[2,]    6   15   24
[3,]    9   18   27

> A              # Multiply two matrices
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> B
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    1    1    1    1
[3,]    1    1    1    1
> A %*% B
     [,1] [,2] [,3] [,4]
[1,]   15   15   15   15
[2,]   18   18   18   18
[3,]   21   21   21   21
[4,]   24   24   24   24
```

Matrix Operations

```
> A <- matrix(c(1,2,3), nrow = 3, ncol = 3)
> A
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
[3,]    3    3    3
> t(A)                  # Transpose
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
[3,]    1    2    3

> A <- matrix(c(1,0,0,0,2,0,0,0,3), nrow = 3, ncol = 3)
> A
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    2    0
[3,]    0    0    3
> solve(A)              # Inverse
     [,1] [,2]      [,3]
[1,]    1  0.0 0.0000000
[2,]    0  0.5 0.0000000
[3,]    0  0.0 0.3333333

> x <- matrix(c(1,2,3), 1, 3)
> y <- matrix(c(1,0,0), 1, 3)
> x %x% y               # Kronecker product
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1    0    0    2    0    0    3    0    0
```

Matrix Operations

```
> A <- matrix(c(1,0,0,0,2,0,0,0,3), nrow = 3, ncol = 3)
> eigen(A)       # Eigen value/vector decomposition
$values
[1] 3 2 1

$vectors
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]    0    1    0
[3,]    1    0    0

> svd(A)         # Singular valud decomposition
$d
[1] 3 2 1

$u
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]    0    1    0
[3,]    1    0    0

$v
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]    0    1    0
[3,]    1    0    0
```

These functions return `Lists`, which we will discuss shortly.

Stacking Matrices

```
> A <- matrix(0, nrow = 3, ncol = 3)
> B <- matrix(1, nrow = 3, ncol = 3)

> rbind(A, B)
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
[4,]    1    1    1
[5,]    1    1    1
[6,]    1    1    1

> cbind(A, B)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    0    0    1    1    1
[2,]    0    0    0    1    1    1
[3,]    0    0    0    1    1    1
```

Functions to Operate on matrices

```
> A <- matrix(seq(1,9), nrow = 3, ncol = 3)
> sqrt(A)             # sqrt works elementwise on matrix
         [,1]     [,2]     [,3]
[1,] 1.000000 2.000000 2.645751
[2,] 1.414214 2.236068 2.828427
[3,] 1.732051 2.449490 3.000000

# Note: see the chol function to compute B such that B %*% B = A

> A <- matrix(seq(1,9), nrow = 3, ncol = 3)
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> sum(A)              # sum and max treat the matrix like a vector
[1] 45
> max(A)
[1] 9

> apply(A, 1, max)  # Apply max to each row of A
[1] 7 8 9
> apply(A, 2, max)  # Apply to columns
[1] 3 6 9
```

Arrays

Arrays are similar to matrices, but can have > 2 dimensions.

```
> A <- array(seq(1,27), dim = c
    (3,3,3))
> A
, , 1

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

, , 2

     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18

, , 3

     [,1] [,2] [,3]
[1,]   19   22   25
[2,]   20   23   26
[3,]   21   24   27
```

```
> A[,2,1]  # 2nd col from 1st slice
[1] 4 5 6

> A[,,1]   # 1st slice
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> A[,,1:2] # 1st and 2nd slice
, , 1

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

, , 2

     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18

> apply(A, 3, sum)
[1]  45 126 207
```

Lists

- Vectors, matrices, and arrays are intended for scalars, usually numbers.
- Lists can hold elements of varying types.
- Lists can be used to construct more complicated data structures.

```
> x <- list(a = "string value", b = 5.5, c = TRUE)
> print(x)
$a
[1] "string value"

$b
[1] 5.5

$c
[1] TRUE

> x$a
[1] "string value"
> x[["a"]]
[1] "string value"
> x[[1]]
[1] "string value"
```

- Lists are used to return multiple results from a function (e.g. `eigen`).

Data Frames

- Data frames are convenient for data in table format. Each column represents a variable, and can have its own data type.

- The CO2 dataset comes with R. First, check the help page using ?CO2.

```
CO2                  package:datasets                  R
     Documentation

Carbon Dioxide uptake in grass plants

Description:

     The 'CO2' data frame has 84 rows and 5 columns of data from an
     experiment on the cold tolerance of the grass species
     _Echinochloacrus-galli_.
...
```

- Now let's view the dataset

```
> CO2
   Plant         Type  Treatment conc uptake
1    Qn1       Quebec nonchilled   95   16.0
2    Qn1       Quebec nonchilled  175   30.4
3    Qn1       Quebec nonchilled  250   34.8
...
83   Mc3  Mississippi    chilled  675   18.9
84   Mc3  Mississippi    chilled 1000   19.9
```

Data Frames

- Access each field as a vector using $ operator.

```
> CO2$uptake
 [1] 16.0 30.4 34.8 37.2 35.3 39.2 39.7 13.6 27.3 37.1 41.8
...
[76] 13.7 14.4 10.6 18.0 17.9 17.9 17.9 18.9 19.9
```

- Access rows or columns using matrix indexing.

```
> CO2[3,]
  Plant   Type  Treatment conc uptake
3   Qn1 Quebec nonchilled  250   34.8
> CO2[,4]
 [1]   95  175  250  350  500  675 1000   95  175  250  350
...
[76]  675 1000   95  175  250  350  500  675 1000
> CO2[3,4]
[1] 250
```

- Create a new `data.frame`

```
> data.frame(new.conc = CO2$conc, new.uptake = CO2$uptake)
   new.conc new.uptake
1        95        16.0
2       175        30.4
3       250        34.8
4       350        37.2
5       500        35.3
```

Data Frames

We can sort a data.frame by one or more columns using `order`

```
> idx <- order(CO2$uptake)
> CO2[idx,]
   Plant         Type  Treatment conc uptake
71    Mc2 Mississippi    chilled   95    7.7
29    Qc2      Quebec    chilled   95    9.3
64    Mc1 Mississippi    chilled   95   10.5
43    Mn1 Mississippi nonchilled   95   10.6
...
19    Qn3      Quebec nonchilled  500   42.9
20    Qn3      Quebec nonchilled  675   43.9
14    Qn2      Quebec nonchilled 1000   44.3
21    Qn3      Quebec nonchilled 1000   45.5
```

Data Frames

- We can use `merge` to do an "inner join" of data.frames, like SQL.

```
> data1
  id  color
1  1    red
2  2  green
3  3   blue
4  4 yellow

> data2
  id    shape letter
1  2   square      a
2  3 triangle      b
3  4   circle      c
4  5     star      d

> merge(data1, data2, by="id")
  id  color    shape letter
1  2  green   square      a
2  3   blue triangle      b
3  4 yellow   circle      c
```

- For more examples of manipulating data.frames, see `www.statmethods.net/management`.

Functions

- Functions are objects and can be passed like any other data type in R.

```
> f <- function(x) { x + 1 }
> f(1)
[1] 2
> g <- function(f, x) { f(x)^2 }
> g(f,1)
[1] 4
```

- The argument types and return type are not declared in advance.
- We can also explicitly specify what to return.

```
f <- function(x) {
  if (x %% 2 == 0)
    return("even")
  else
    return("odd")
}
```

Listing 1: Script file: function.R

- Running this script produces the following output.

```
> source("function.R")
> print(f(1))
[1] "odd"
> print(f(2))
[1] "even"
```

Functions

- Can set default values for arguments.

```
> f <- function(x,y,z = 100) { c(x, y, z) }
> f(y = 1, x = 3)
[1] 3 1 100
```

- Argument types can be checked manually.

```
> f <- function(x) { stopifnot(is.numeric(x)); return(x^2) }
> f(1)
[1] 0
> f("a")
Error: is.numeric(x) is not TRUE
```

- Functions can be constructed around variables that are defined outside. This can be a source of confusion.

```
> y <- 5
> f <- function(x) { x + y }
> f(1)
[1] 6
```

Control Flow

- `if` statement

```
u <- runif(1)
if (u < 0.25) {
    y <- 0
} else if (u < 0.5) {
    y <- 1
} else {
    y <- 2
}
```

- `switch` statement

```
x <- "1"
switch(x,
    "1" = "blue",
    "2" = "green",
    "3" = "red",
    "default")
```

- `for` loop

```
for (i in 1:10) { cat("Iteration", i, "\n") }
```

- `while` loop

```
u <- 1
while (u > 0.1) {
    u <- runif(1)
}
```

Control Flow

- `try/catch` statement

```
f <- function(x) { stop("Error!!") }
ret <- tryCatch(f(5),
    error = function(e) {
        return(e)
    }
)
## ret is the message e

g <- function(x) { x }
ret <- tryCatch(g(5),
    error = function(e) {
        return(e)
    }
)
## ret is 5
```

Control Flow

- apply statement

```
> X <- matrix(rnorm(1000*5), 1000, 5)
> apply(X, 2, sd)
 [1] 0.9932699 0.9643422 0.9976412 0.9784432 1.0072679
```

- sapply statement

```
f <- function(x) { if (x %% 2 == 0) { "even" } else { "odd" } }
x <- rpois(8, 5)
sapply(x, f)

> sapply(x, f)
 [1] "odd"  "odd"  "odd"  "even" "odd"  "odd"  "odd"  "even"
```

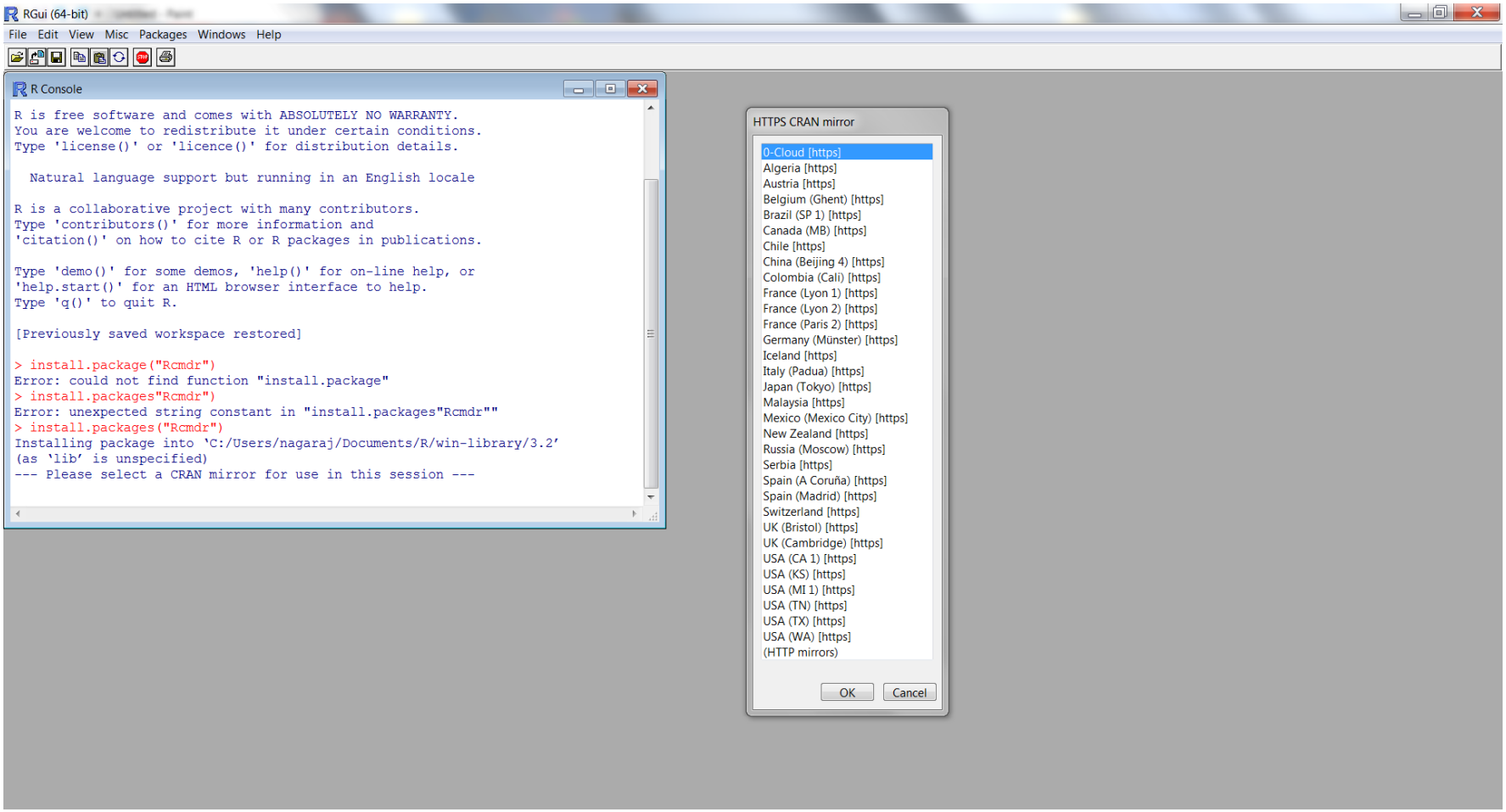
- sapply statement

```
> L <- list()
> L[[1]] <- rnorm(100)
> L[[2]] <- rnorm(100)
> L[[3]] <- rnorm(100)
> lapply(L, function(x) { c(length(x), mean(x), sd(x)) })
[[1]]
[1] 100.00000000   0.07369905   1.03257221

[[2]]
[1] 100.0000000   0.0376098   0.8865248

[[3]]
[1] 100.0000000  -0.1576617   1.0352769
```

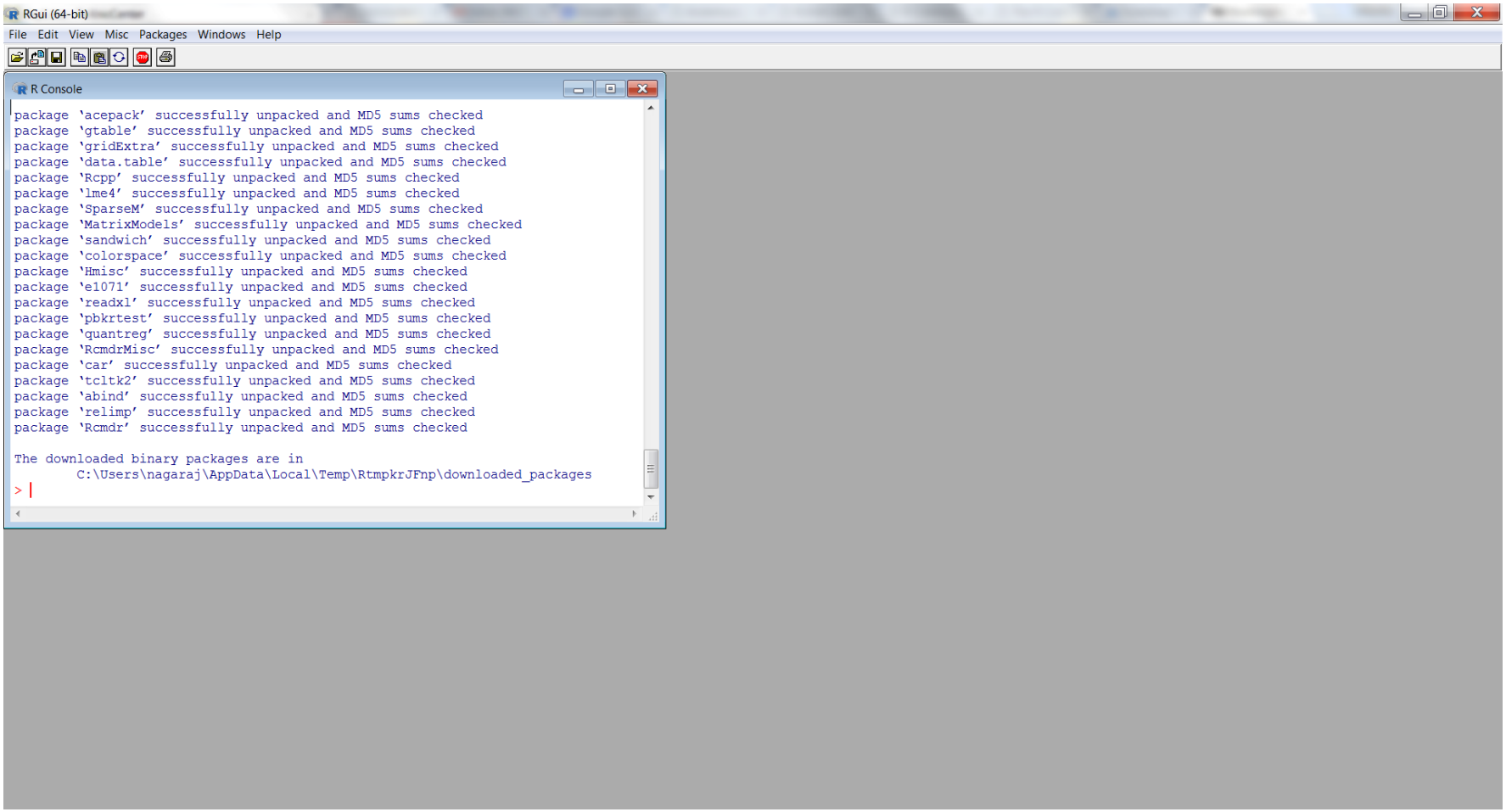
About R Commander

- R commander is a package, to be installed and loaded.
- R commander can be invoked with Rcmdr() or load("Rcmdr")
- Rcmdr opens up a GUI which can be used to
  - read data file types (.txt, .xlsx, etc)
  - use graphical tools such as histogram, boxplot, scatterplot matrix, etc
  - use most commonly used parametric and nonparametric statistical procedures, and implement several linear and generalized linear models
- Many Rcmdr Plugin's are also available: RcmdrPlugin.BCA, RcmdrPlugin.DoE, etc
- Created by John Fox, McMaster University in 2003
- `http://www.rcommander.com/`, `http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/`

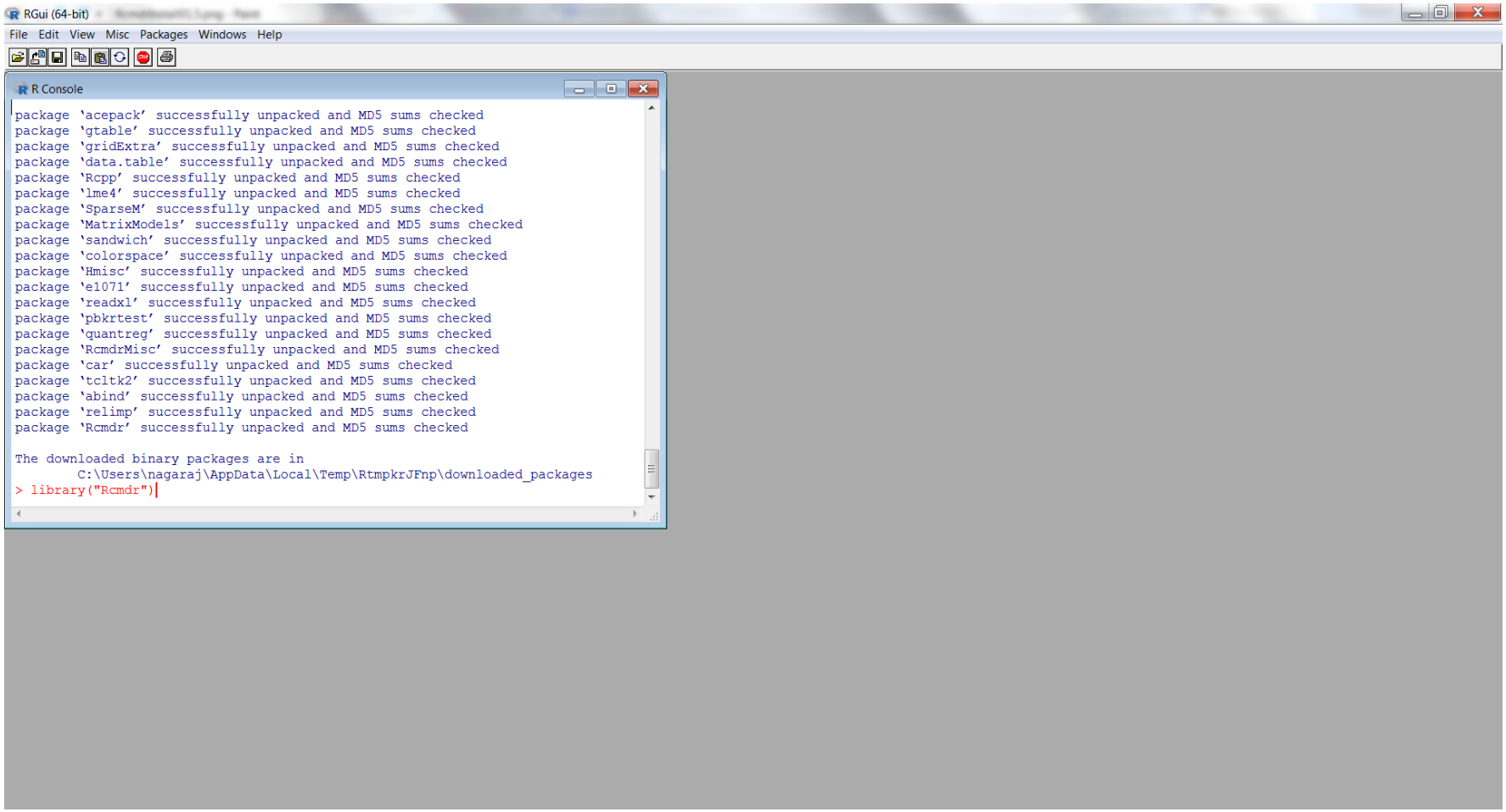
Installing R Commander



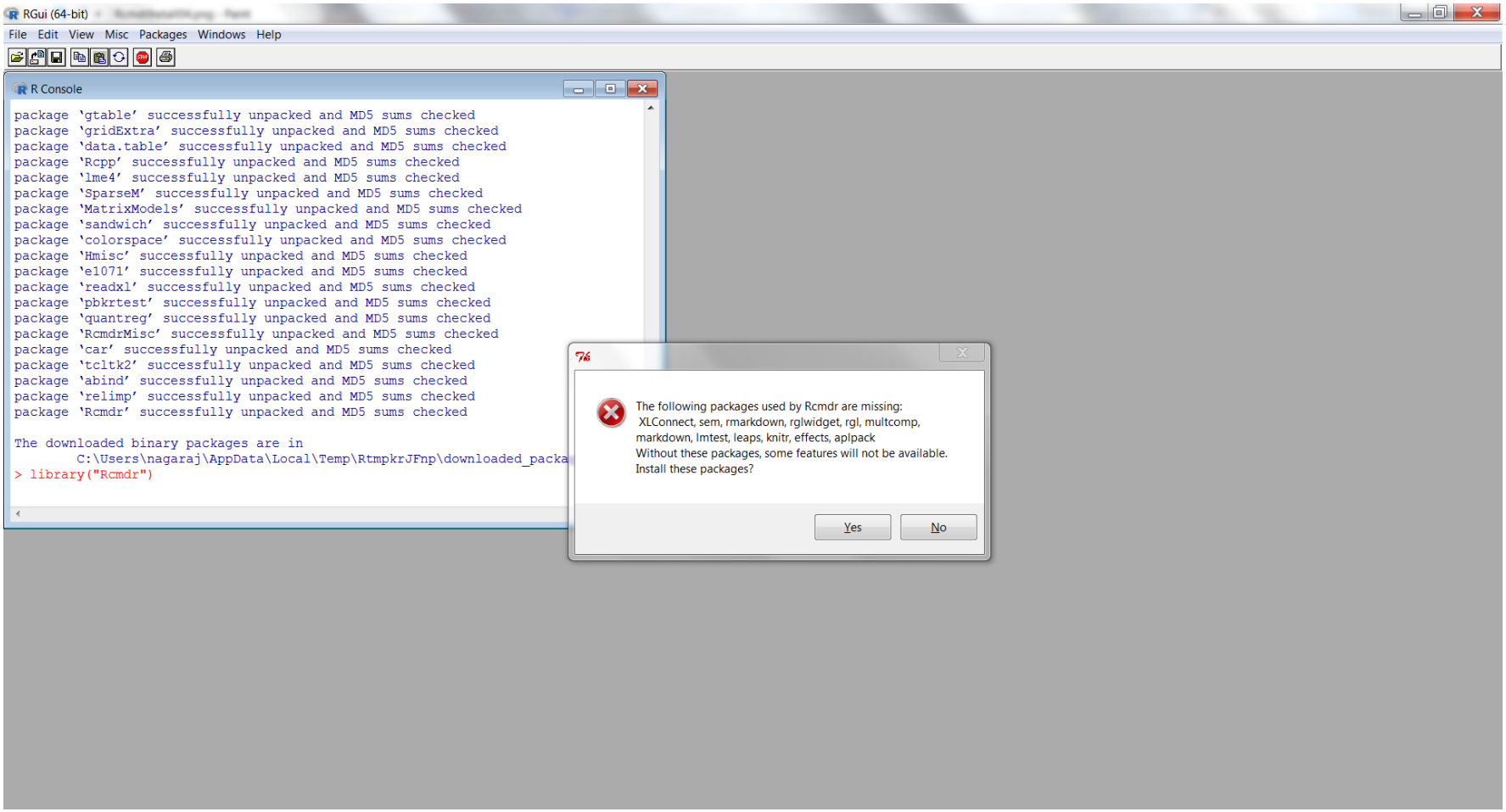
Installing R Commander



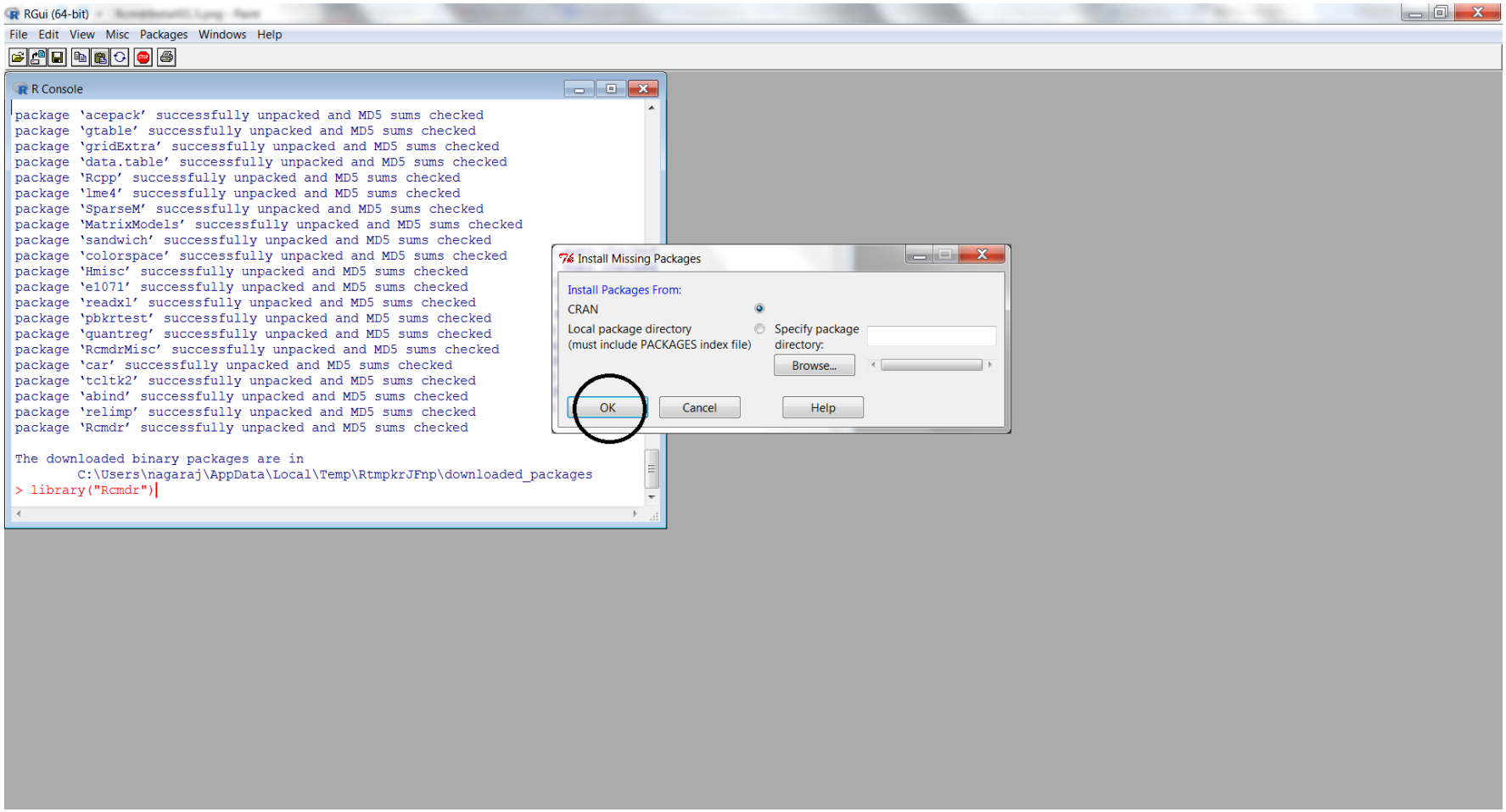
Invoking Rcmdr



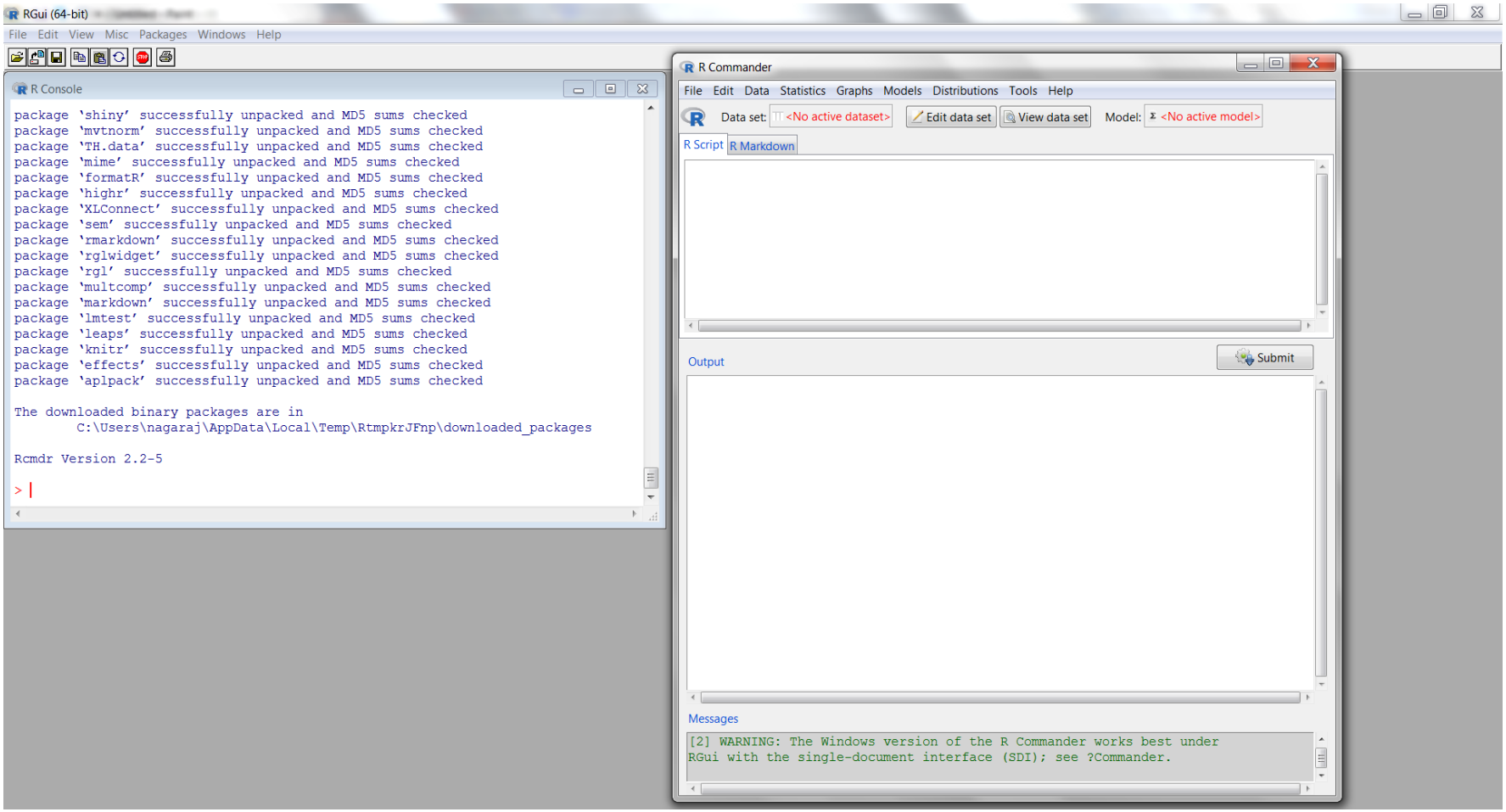
Invoking Rcmdr



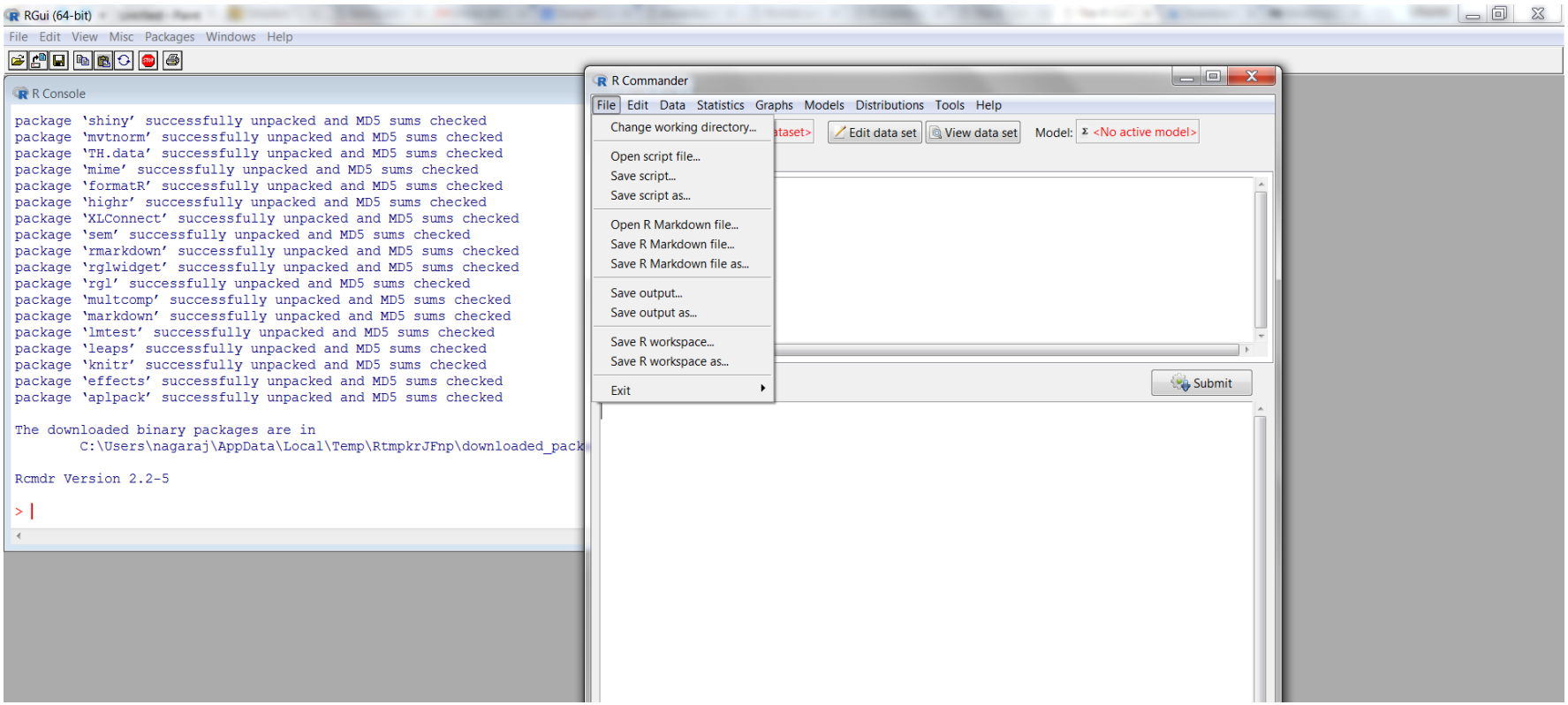
Invoking Rcmdr



Invoking Rcmdr



Menus of Rcmdr



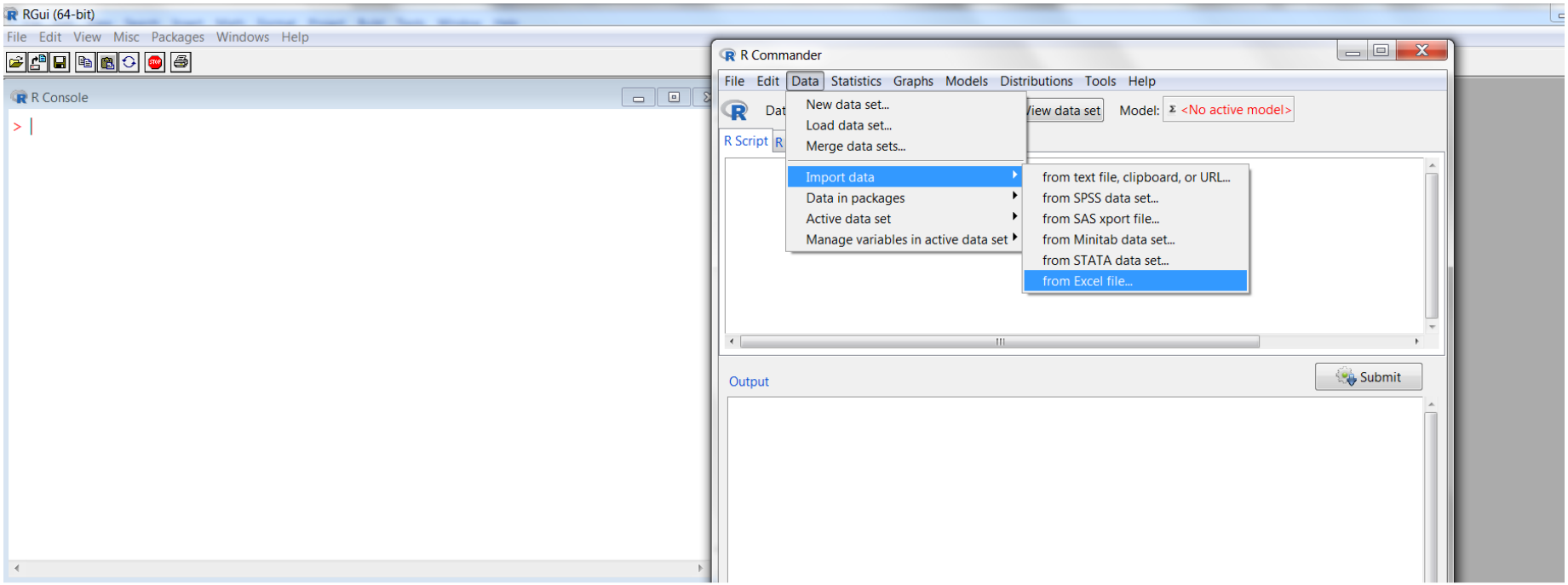
Complete Menu "tree" Rcmdr

- File
  - Open script file
  - Save script
  - Save script as
  - Save output
  - Save output as
  - Save R workspace
  - Save R workspace as
  - Exit
    - from Commander
    - from Commander and R
  - Edit
    - Clear Window
    - Cut
    - Copy
    - Paste
    - Delete
    - Find
    - Select all
- Data
  - New data set
  - Import data
    - from text file
    - from SPSS data set
    - from Minitab data set
    - from STATA data set
  - Data in packages
    - List data sets in packages
    - Read data set from attached package
    - Active data set
      - Select active data set
      - Help on active data set (if available)
      - Variables in active data set
      - Set case names
      - Subset active data set
      - Remove cases with missing data
      - Export active data set
  - Manage variables in active data set
    - Recode variable
    - Compute new variable
    - Standardize variables
    - Convert numeric variable to factor
    - Bin numeric variable
    - Reorder factor levels
    - Define contrasts for a factor
    - Rename variables
    - Delete variables from data set
- Statistics
  - Summaries
    - Active data set
    - Numerical summaries
    - Frequency distribution
    - Table of statistics
    - Correlation matrix
  - Contingency Tables
    - Two-way table
    - Multi-way table
    - Enter and analyze two-way table
  - Means
    - Single sample t-test
    - Independent-samples t-test
    - Paired t-test
    - One-way ANOVA
    - Multi-way ANOVA
  - Proportions
    - Single-sample proportion test
    - Two-sample proportions test
  - Variances
    - Two-variances F-test
    - Bartlett's test
    - Levene's test
  - Nonparametric tests
    - Two-sample Wilcoxon test
    - Paired-samples Wilcoxon test
    - Kruskal-Wallis test
  - Dimensional analysis
    - Scale reliability
    - Principal-components analysis
    - Factor analysis
    - Cluster analysis
      - k-means cluster analysis
      - Hierarchical cluster analysis
      - Summarize hierarchical clustering
      - Add hierarchical clustering to data set
  - Fit models
    - Linear regression
    - Linear model
    - Generalized linear model
    - Multinomial logit model
    - Proportional-odds logit model
- Graphs
  - Index plot
  - Histogram
  - Stem-and-leaf display
  - Boxplot
  - Quantile-comparison plot
  - Scatterplot
  - Scatterplot matrix
  - 3D scatterplot
  - Line graph
  - Plot of means
  - Bar graph
  - Pie chart
  - Save graph to file
    - as bitmap
    - as PDF/Postscript/EPS
    - 3D RGL graph
- Models
  - Select active model
  - Summarize model
  - Add observation statistics to data
  - Confidence intervals
  - Hypothesis tests
    - ANOVA table
    - Compare two models
    - Linear hypothesis
  - Numerical diagnostics
    - Variance-inflation factors
    - Breusch-Pagan test for heteroscedasticity
    - Durbin-Watson test for autocorrelation
    - RESET test for nonlinearity
    - Bonferroni outlier test
  - Graphs
    - Basic diagnostic plots
    - Residual quantile-comparison plot
    - Component+residual plots
    - Added-variable plots
    - Influence plot
    - Effect plots
- Distributions
  - Normal distribution
    - Normal quantiles
    - Normal probabilities
    - Plot normal distribution
  - t distribution
    - t quantiles
    - t probabilities
    - Plot t distribution
  - Chi-squared distribution
    - Chi-squared quantiles
    - Chi-squared probabilities
    - Plot chi-squared distribution
  - F distribution
    - F quantiles
    - F probabilities
    - Plot F distribution
  - Binomial distribution
    - Binomial quantiles
    - Binomial tail probabilities
    - Binomial probabilities
    - Plot binomial distribution
  - Poisson distribution
    - Poisson probabilities
    - Plot Poisson distribution
- Tools
  - Load package(s)
  - Options
- Help
  - Commander help
  - About Rcmdr
  - Introduction to the R Commander
  - Help on active data set (if available)

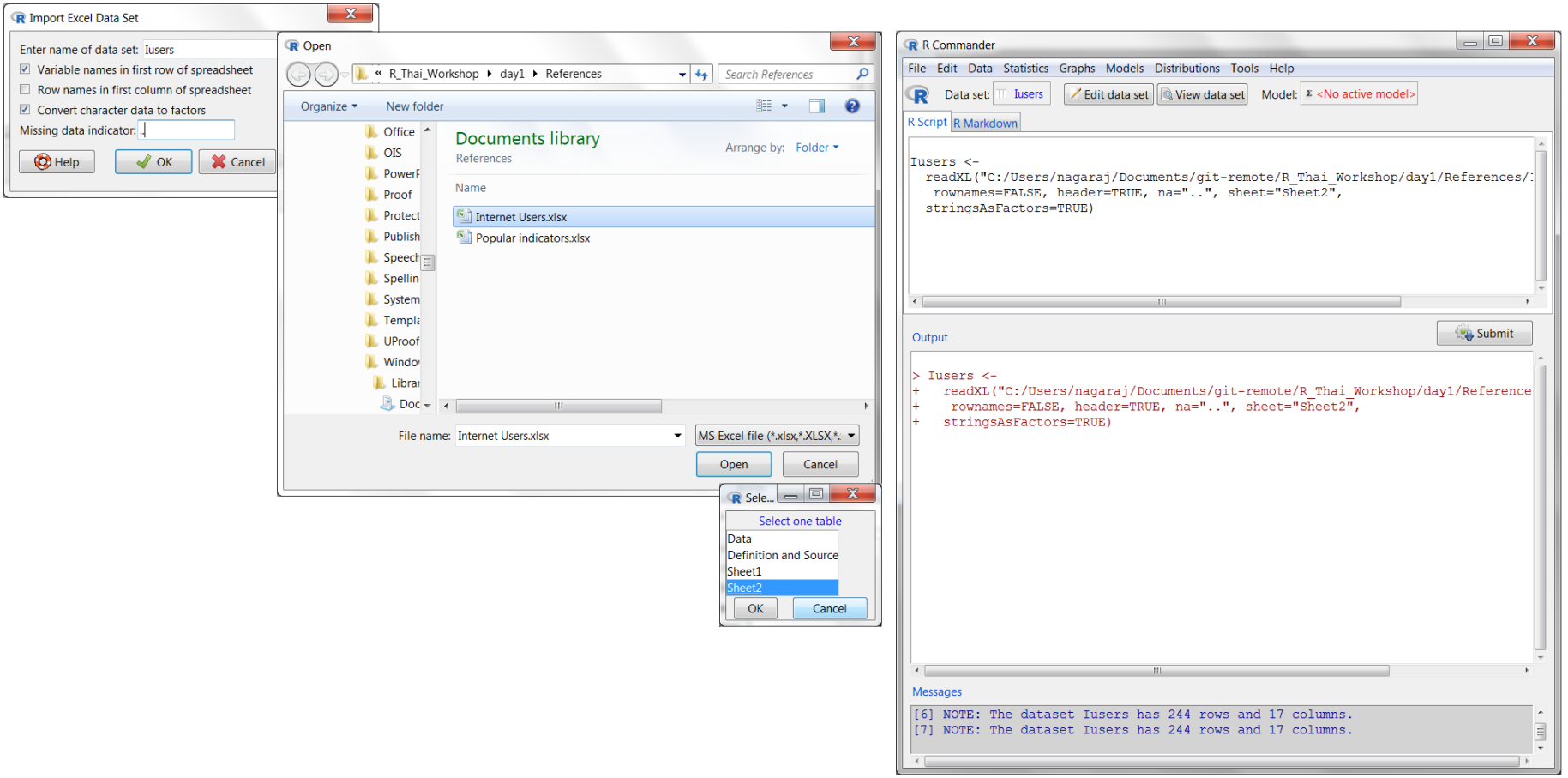
Reading Data in Rcmdr: Example

- Data set consists of annual data (2000-2014) on the 'Number of Internet users per 100' for 244 countries.
- Rcmdr has many options for reading datasets. A typical dataset with both character and numeric variables, either .csv or .xlsx type seem to be quite stable.
- Data can also be read from other R packages.
- InternetUsers.xlsx data set consists of 244 rows and 17 rows. First two rows are 'Country Name' and 'Country Code'. The following 15 columns, IntUsers2000-IntUsers2014, are numeric data.

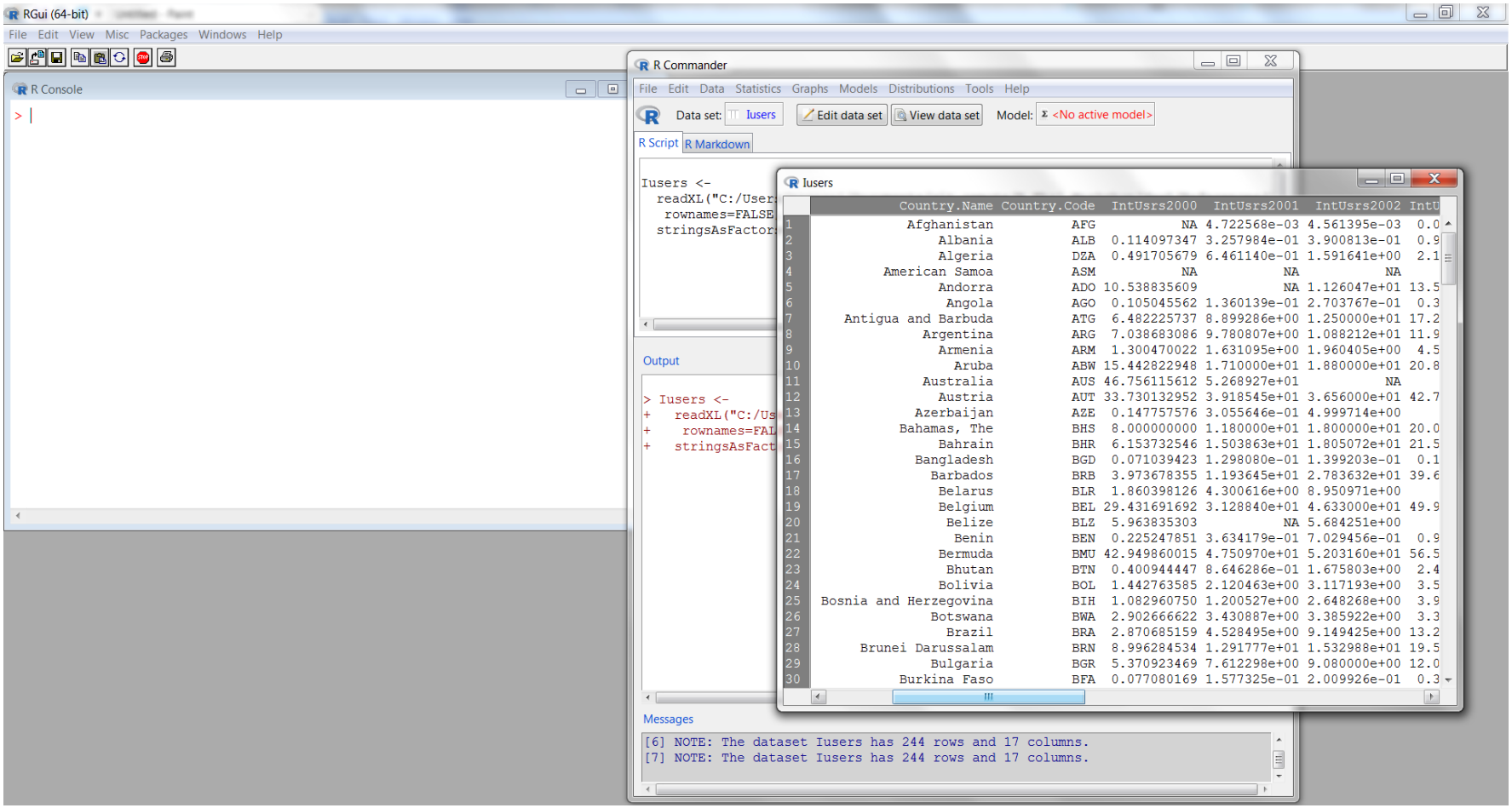
Reading Data in Rcmdr: Example



Reading Data in Rcmdr: Example



Reading Data in Rcmdr: Example

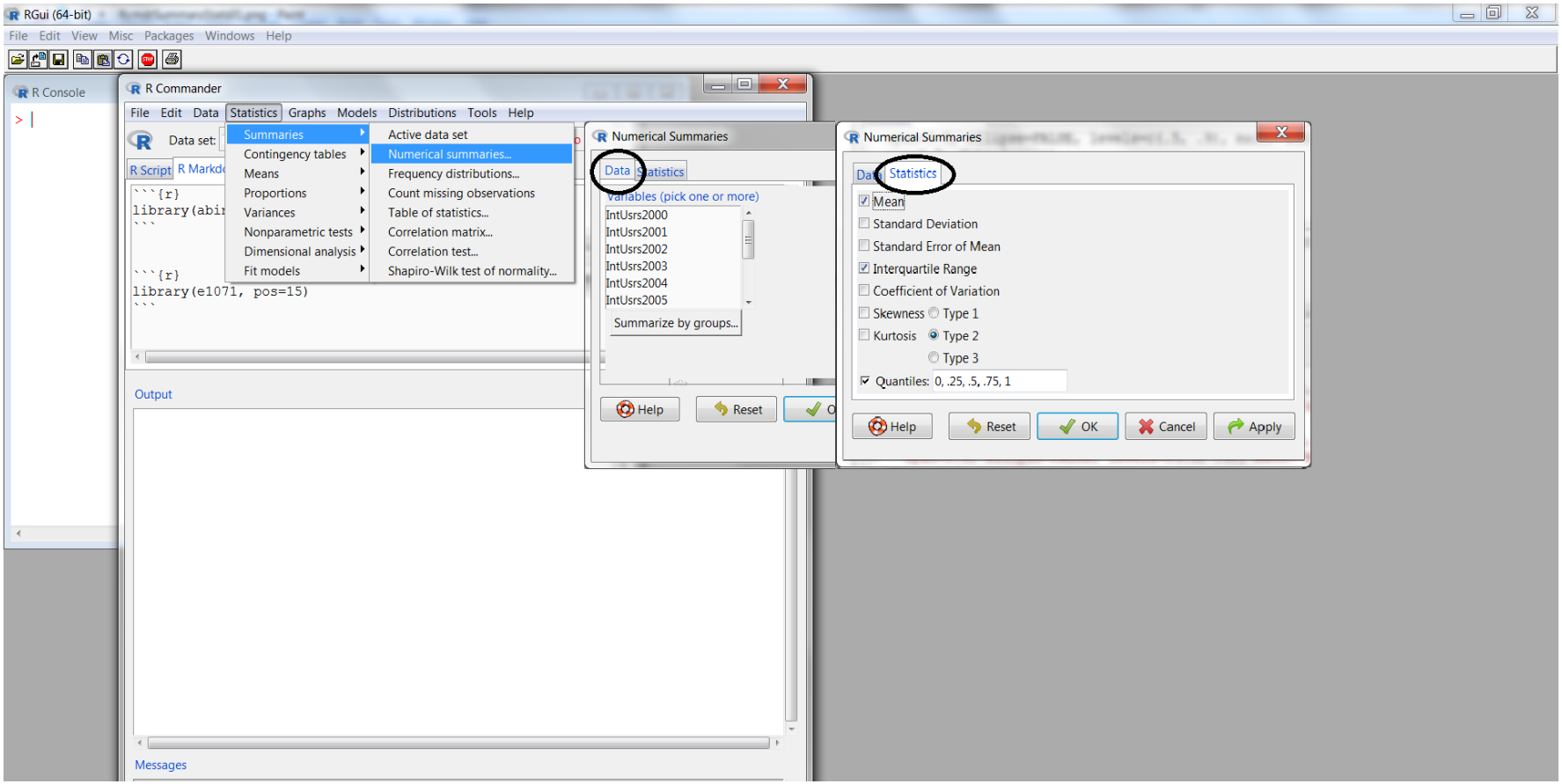


	Country.Name	Country.Code	IntUsrs2000	IntUsrs2001	IntUsrs2002	IntU
1	Afghanistan	AFG	NA	4.722568e-03	4.561395e-03	0.0
2	Albania	ALB	0.114097347	3.257984e-01	3.900813e-01	0.9
3	Algeria	DZA	0.491705679	6.461140e-01	1.591641e+00	2.1
4	American Samoa	ASM	NA	NA	NA	
5	Andorra	ADO	10.538835609	NA	1.126047e+01	13.5
6	Angola	AGO	0.105045562	1.360139e-01	2.703767e-01	0.3
7	Antigua and Barbuda	ATG	6.482225737	8.899286e+00	1.250000e+01	17.2
8	Argentina	ARG	7.038683086	9.780807e+00	1.088212e+01	11.9
9	Armenia	ARM	1.300470022	1.631095e+00	1.960405e+00	4.5
10	Aruba	ABW	15.442822948	1.710000e+01	1.880000e+01	20.8
11	Australia	AUS	46.756115612	5.268927e+01	NA	
12	Austria	AUT	33.730132952	3.918545e+01	3.656000e+01	42.7
13	Azerbaijan	AZE	0.147757576	3.055646e-01	4.999714e+00	
14	Bahamas, The	BHS	8.000000000	1.180000e+01	1.800000e+01	20.0
15	Bahrain	BHR	6.153732546	1.503863e+01	1.805072e+01	21.5
16	Bangladesh	BGD	0.071039423	1.298080e-01	1.399203e-01	0.1
17	Barbados	BRB	3.973678355	1.193645e+01	2.783632e+01	39.6
18	Belarus	BLR	1.860398126	4.300616e+00	8.950971e+00	
19	Belgium	BEL	29.431691692	3.128840e+01	4.633000e+01	49.9
20	Belize	BLZ	5.963835303	NA	5.684251e+00	
21	Benin	BEN	0.225247851	3.634179e-01	7.029456e-01	0.9
22	Bermuda	BMU	42.949860015	4.750970e+01	5.203160e+01	56.5
23	Bhutan	BTN	0.400944447	8.646286e-01	1.675803e+00	2.4
24	Bolivia	BOL	1.442763585	2.120463e+00	3.117193e+00	3.5
25	Bosnia and Herzegovina	BIH	1.082960750	1.200527e+00	2.648268e+00	3.9
26	Botswana	BWA	2.902666622	3.430887e+00	3.385922e+00	3.3
27	Brazil	BRA	2.870685159	4.528495e+00	9.149425e+00	13.2
28	Brunei Darussalam	BRN	8.996284534	1.291777e+01	1.532988e+01	19.5
29	Bulgaria	BGR	5.370923469	7.612298e+00	9.080000e+00	12.0
30	Burkina Faso	BFA	0.077080169	1.577325e-01	2.009926e-01	0.3

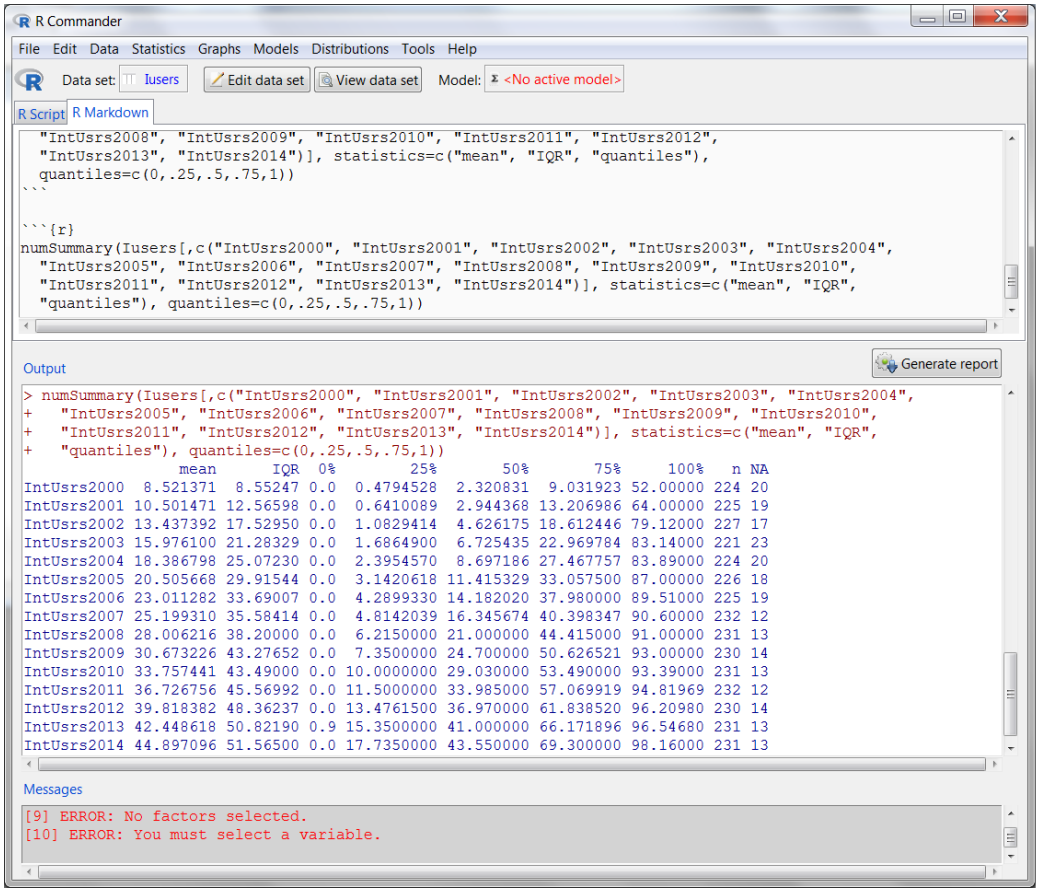
Descriptive Statistics in Rcmdr: Example

- Summary statistics of the uploaded data can be obtained using the menu 'Statistics'
- Statistics $>$ Summaries can access computation of summary statistics for both quantitative as well as qualitative variables
- Summary can be computed for the whole data or by groups (Grouping determined by a 'factor' variable in the data set)
- Results are displayed in the output window of Rcmdr

Descriptive Statistics in Rcmdr: Example



Descriptive Statistics in Rcmdr: Example



Descriptive Statistics in Rcmdr: Example

- Understanding the output:

parameter	What is it?
mean	Measure of central tendency
sd	Standard deviation - a measure of variability in the data
N	Number of readings
NA	Number of missing values
0%	Minimum value
25%	The value below which 25 percent of the observations may be found.
50%	The value below which 50 percent of the observations may be found.
75%	The value below which 75 percent of the observations may be found.
100%	Maximum value

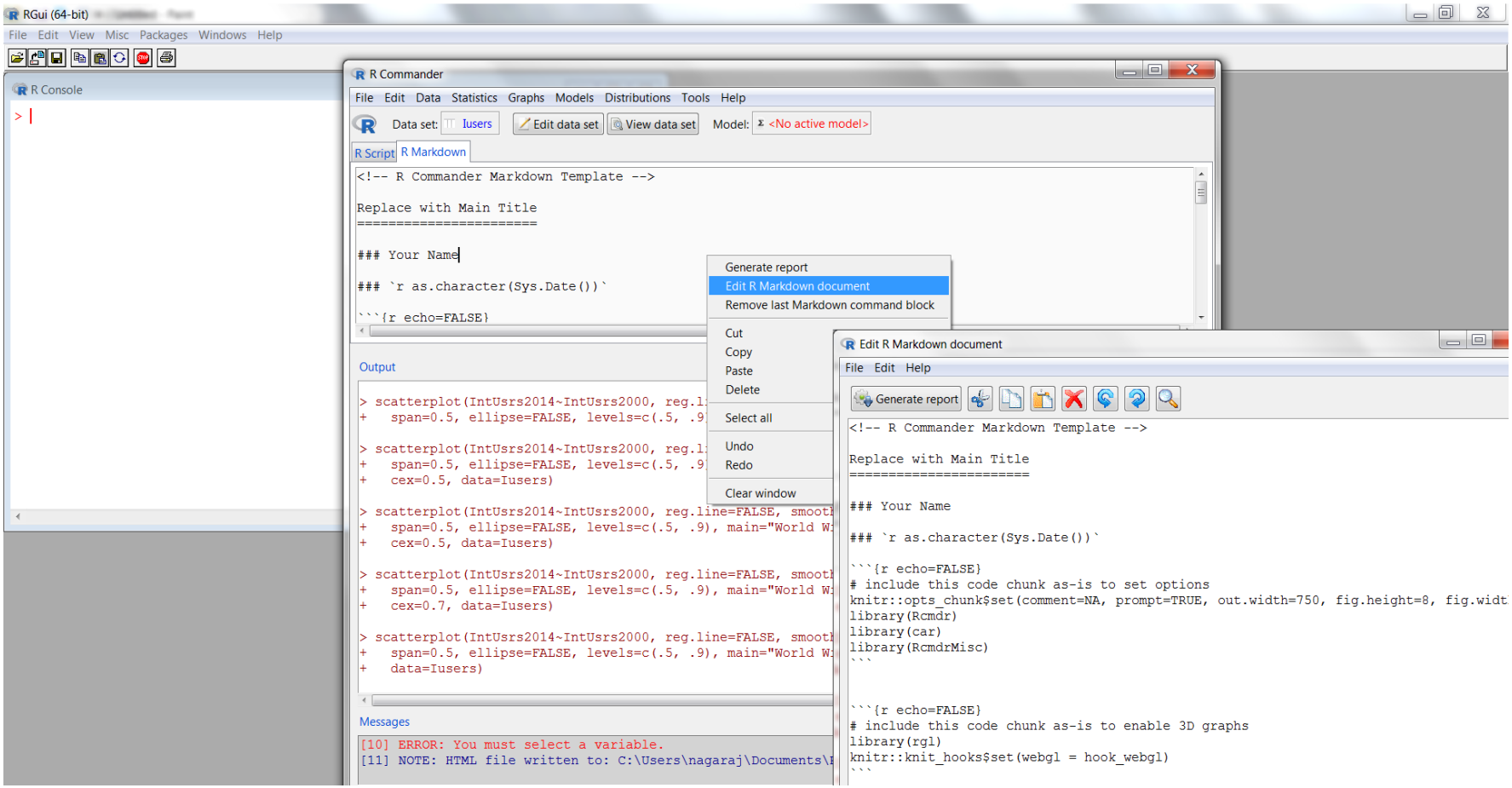
Descriptive Statistics in Rcmdr: Example

- Interpretation of the output:
  - Mean is much higher than the median in the early years due to the highly skewed nature of this data.
  - Over the years, they are becoming closer as the Internet usage is becoming ubiquitous.
  - Even in 2014, 50% of the countries of the world have less than 43 Internet users per 100.
  - Lower percentiles are climbing at a much slower rate than the higher percentiles.

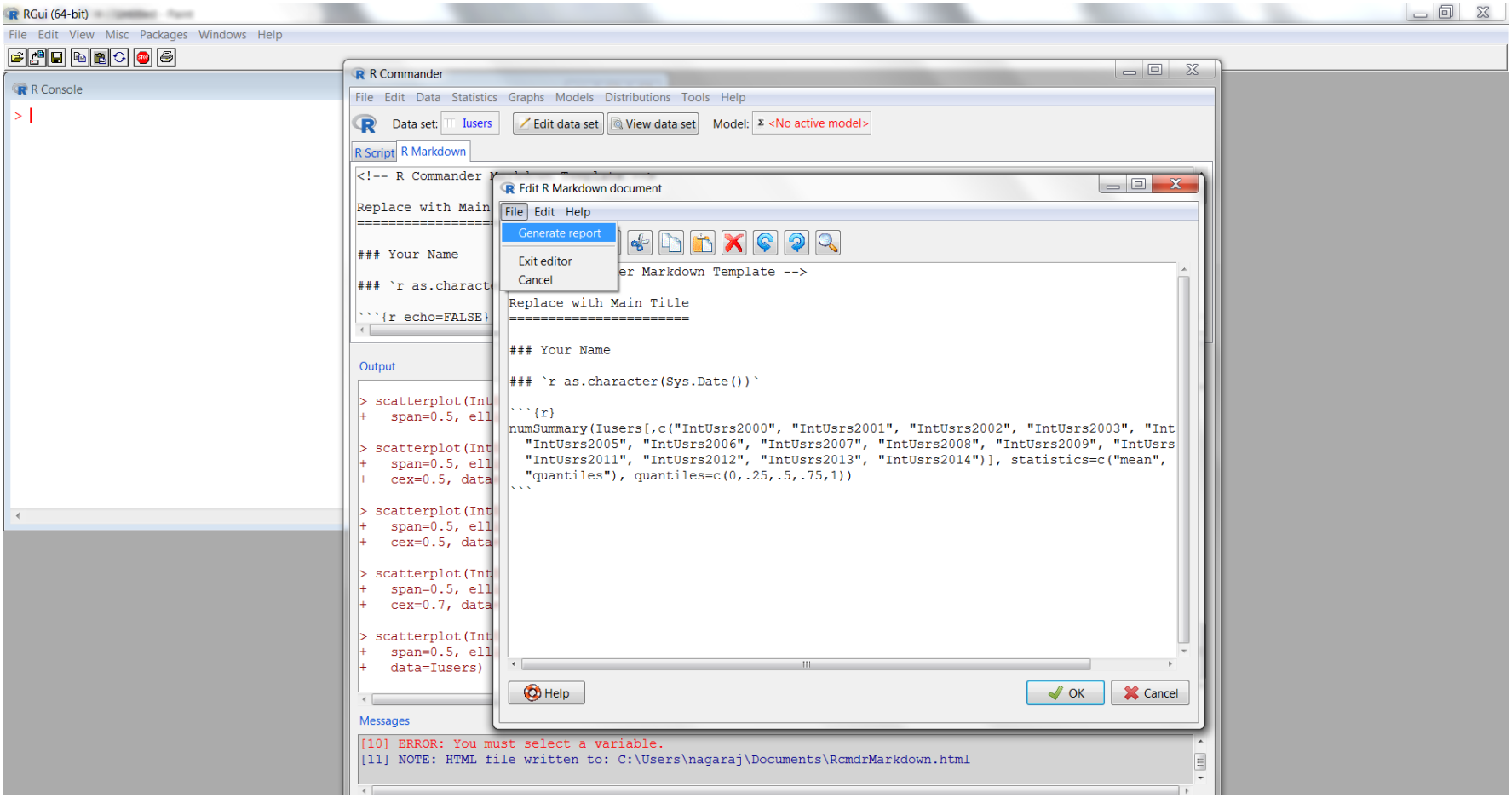
- The results may be cut-and-paste to reports

- R Markdown can also be used for creating reports

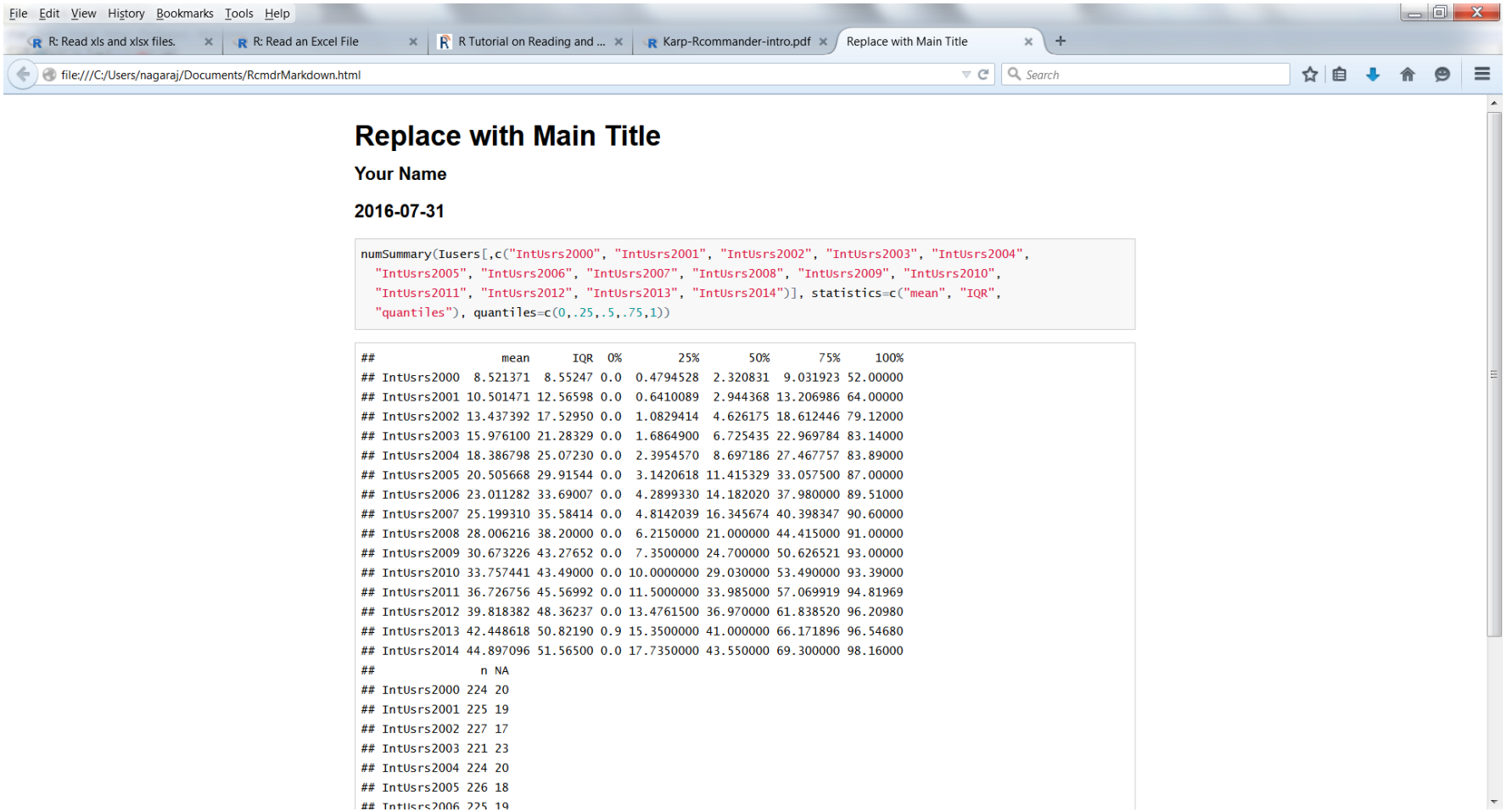
Creating Reports



Creating Reports



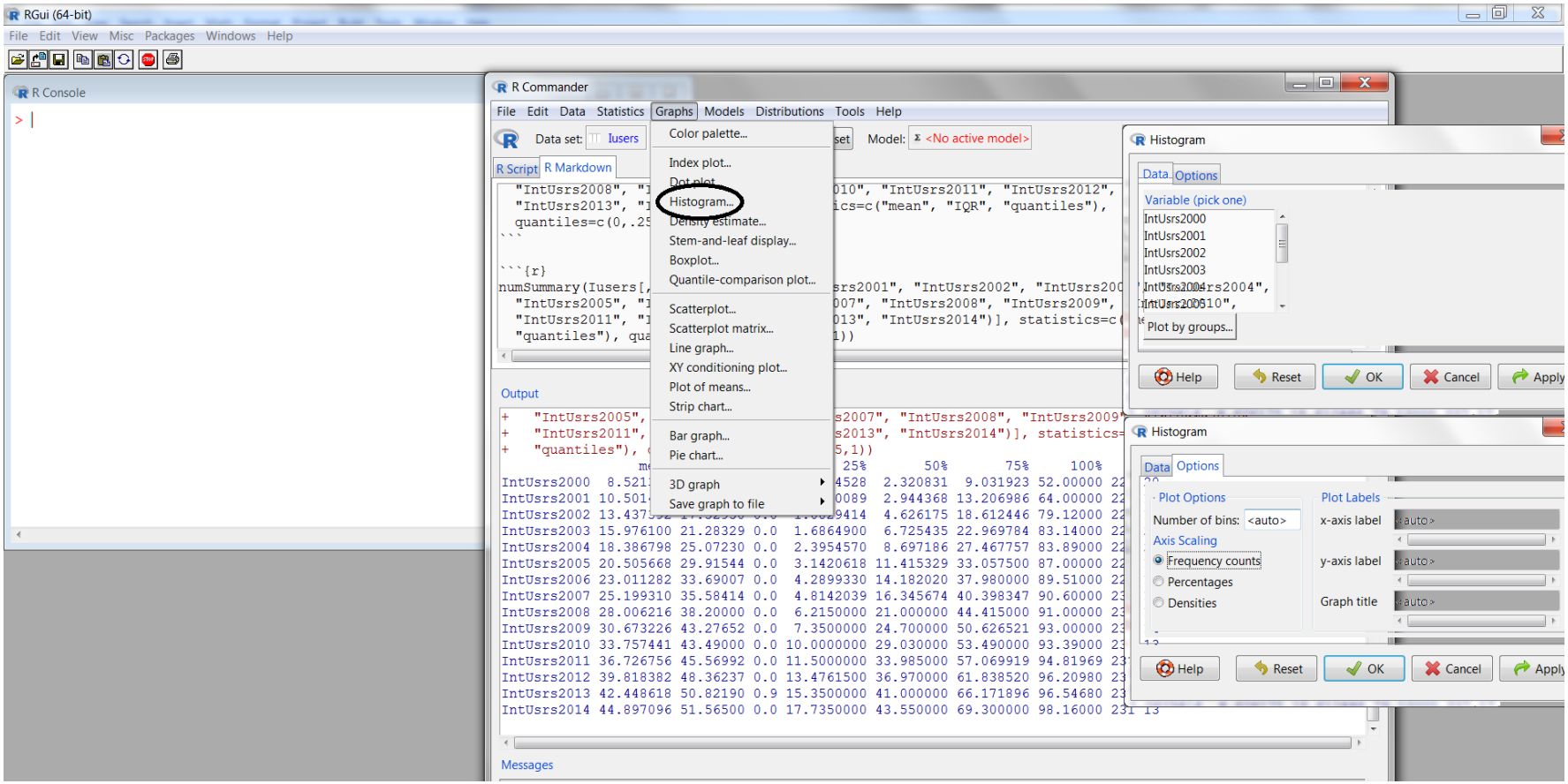
Creating Reports



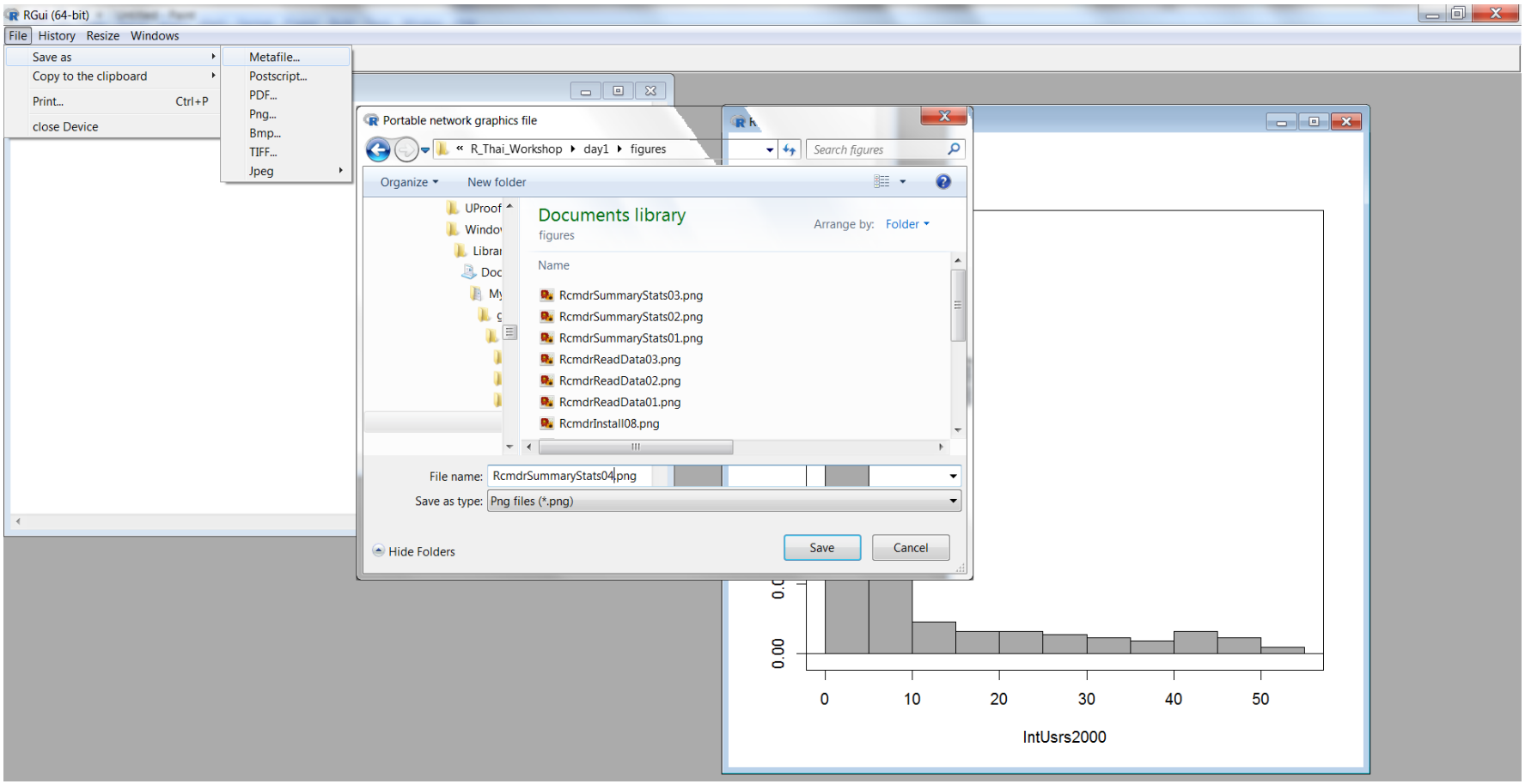
Graphs in Rcmdr

- Rcmdr is also a convenient tool for creating graphical representation of data.
- Next slide shows how to obtain a simple scatterplot.
- There are many choices for [practically] every feature of the plot.
- Try various combinations of choices and determine the best looking graph.

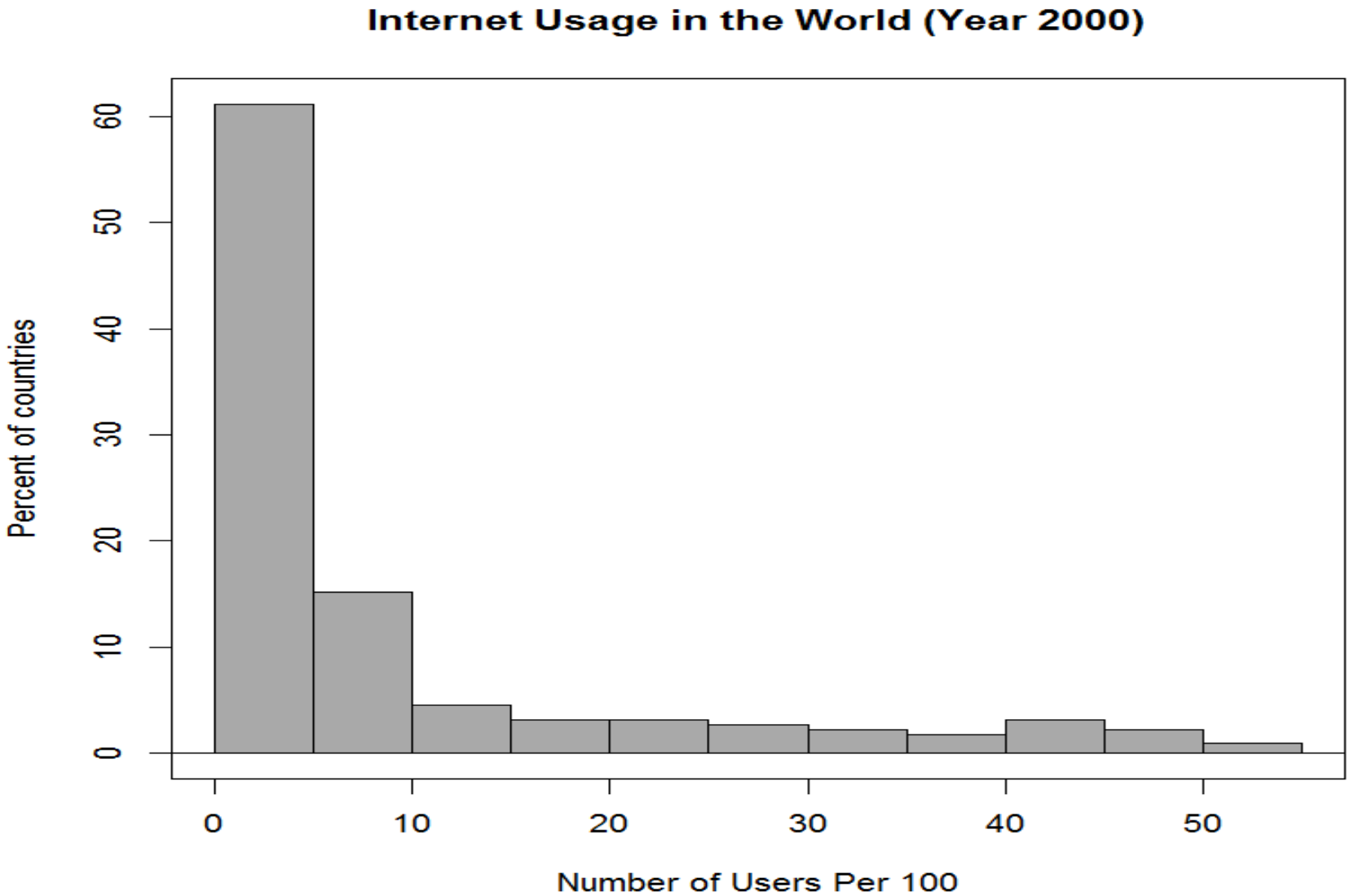
Plotting a Histogram



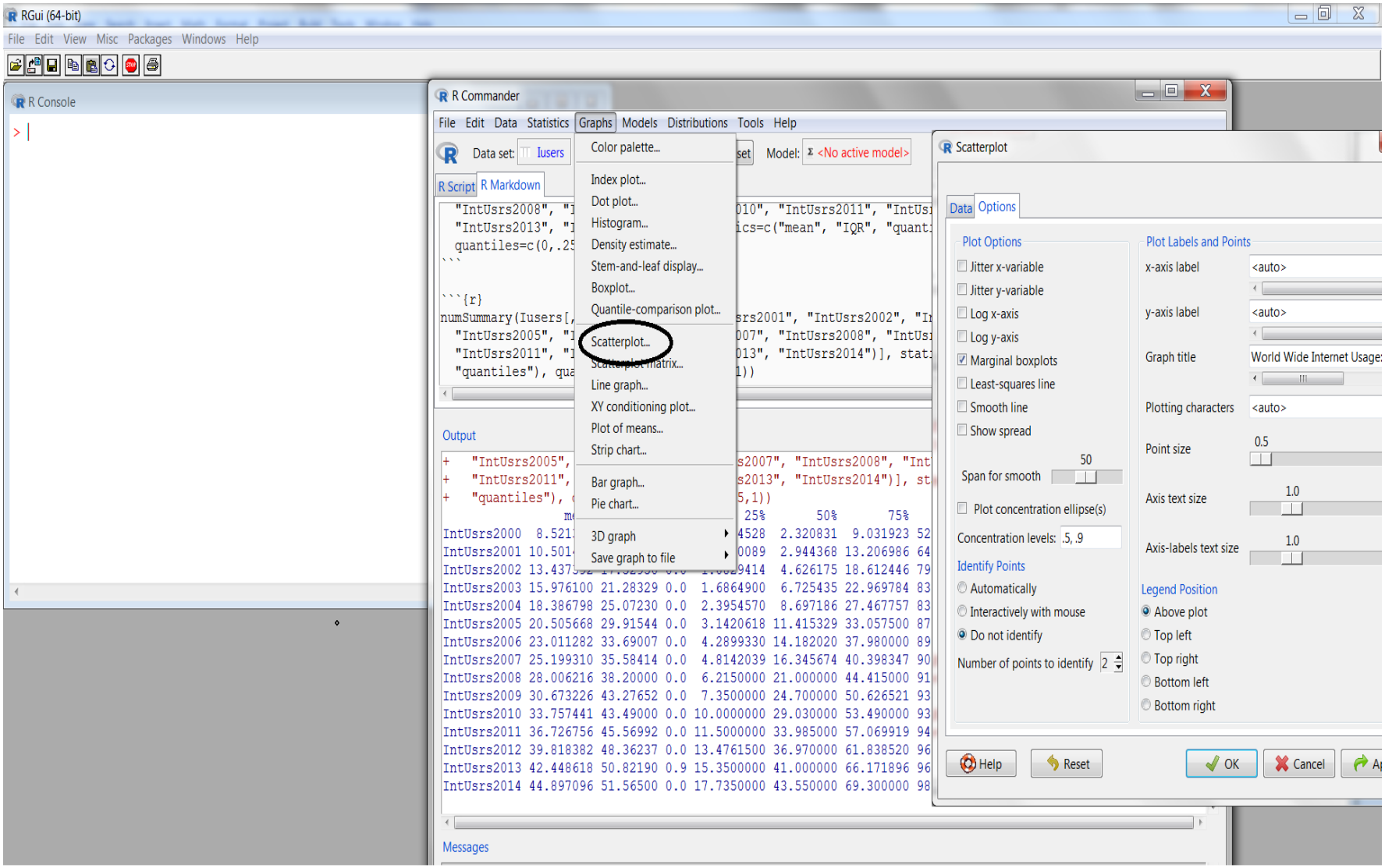
Saving a Plot



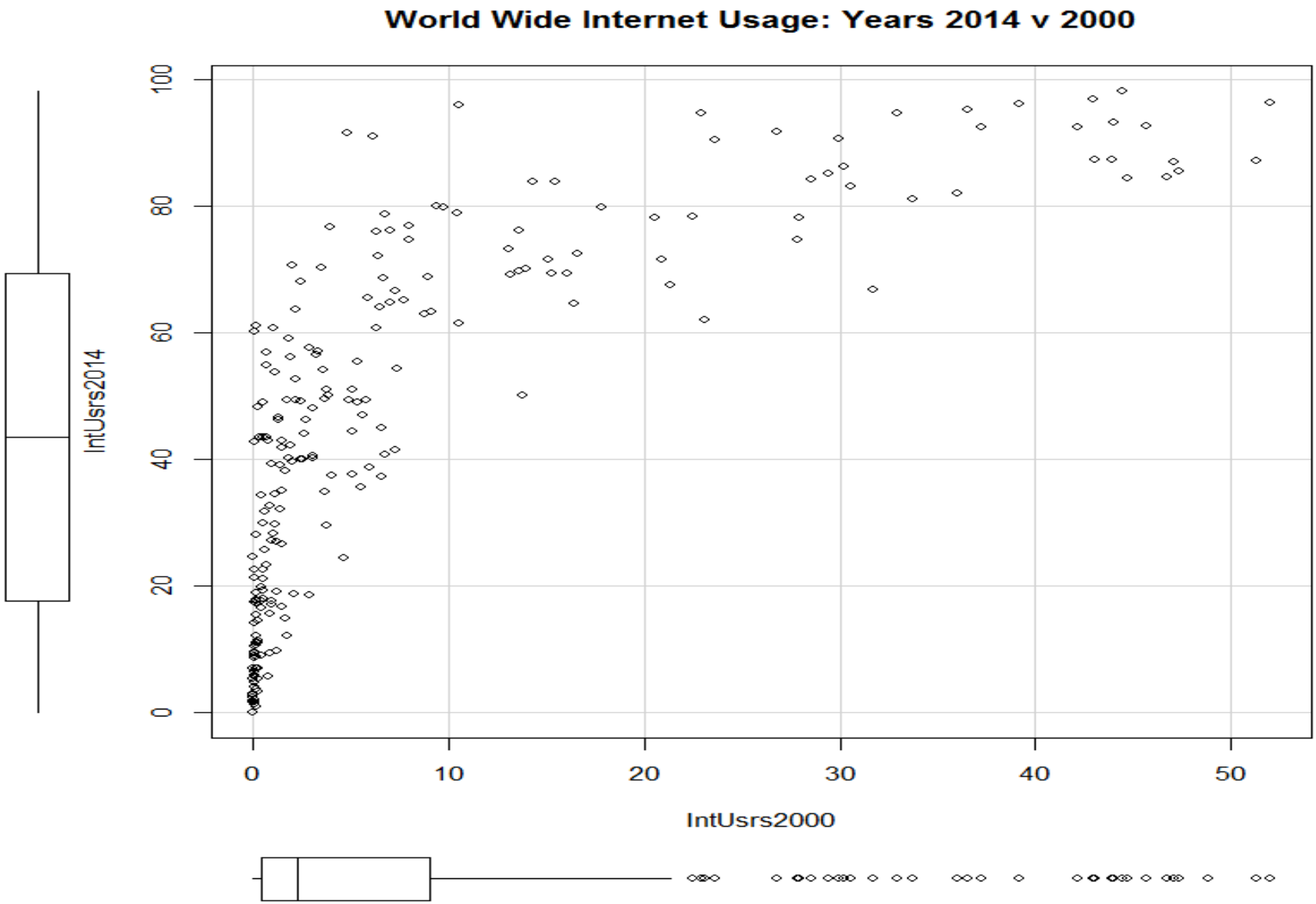
Histogram


Internet Usage in the World (Year 2000)
Percent of countries
0
10
20
30
40
50
60
0
10
20
30
40
50
Number of Users Per 100

Making a Scatterplot



Scatterplot



Modifying a Dataset

- Rcmdr also provides several tools to modify the active data sets. Some examples are:
- Compute a new variable
- Convert a numeric to a factor variable
- Bin a numeric variable
- Sub-dividing the data (either by columns or by rows)

Useful Tips for Session Management

- Exiting and saving the script:
  - Helps keep track of the session
  - A saved script can be opened in a new session to start from the last point of save
  - Better option than saving the workspace

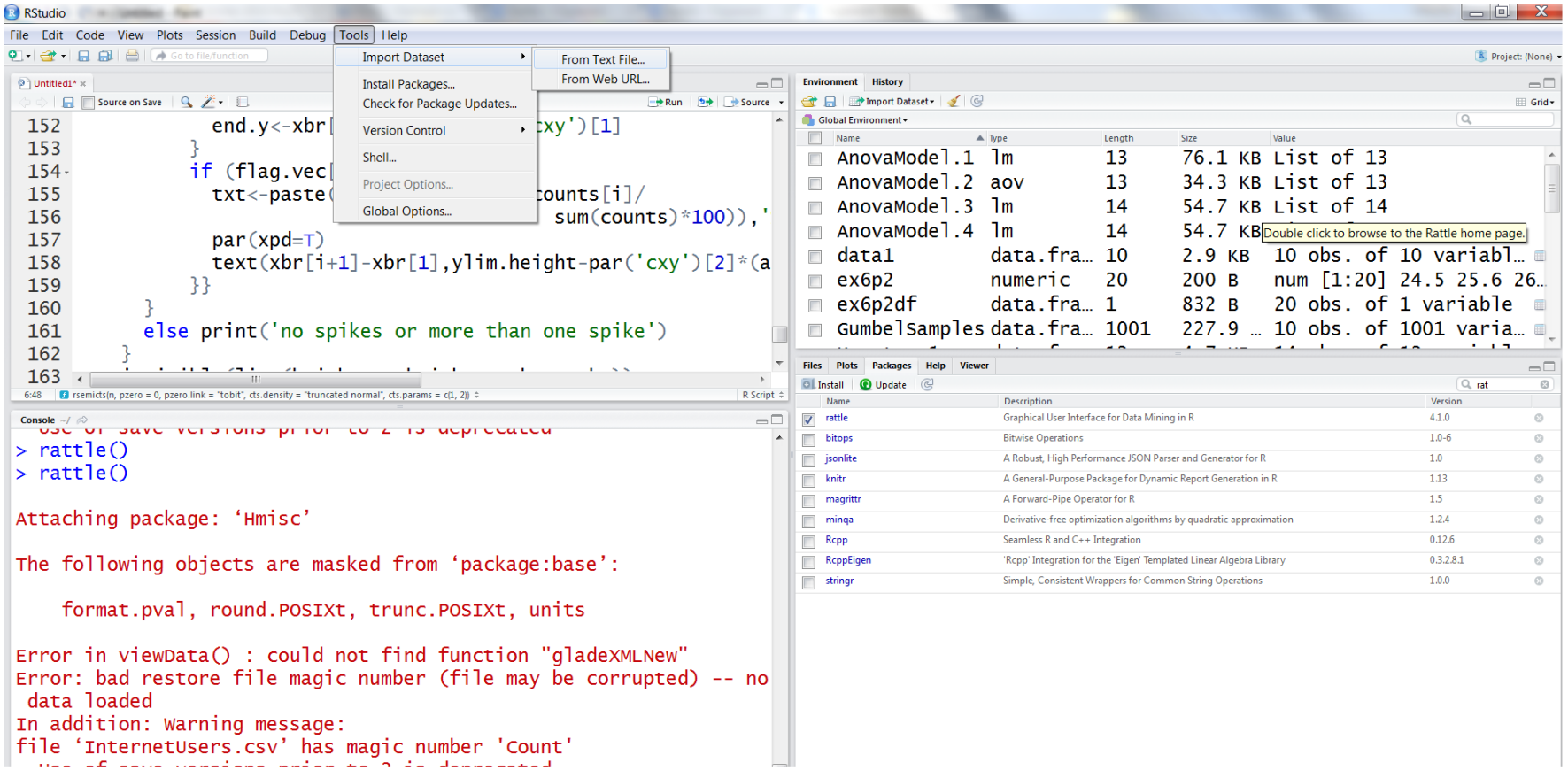
- Saving and printing the output
  - Cut (ctrl-c) -and- paste (ctrl-v) output (ongoing) into an open [word/text] report document
  - Best practice is to write notes as the output is moved to the report
  - Graphs may also be similarly pasted or use File $>$ Save as method described earlier

- Commands generated by the R Commander appear in the script window:
  - This window can be edited so that only those commands that worked are saved
  - To send this script you have to highlight the relevant text and press the 'Submit' button.

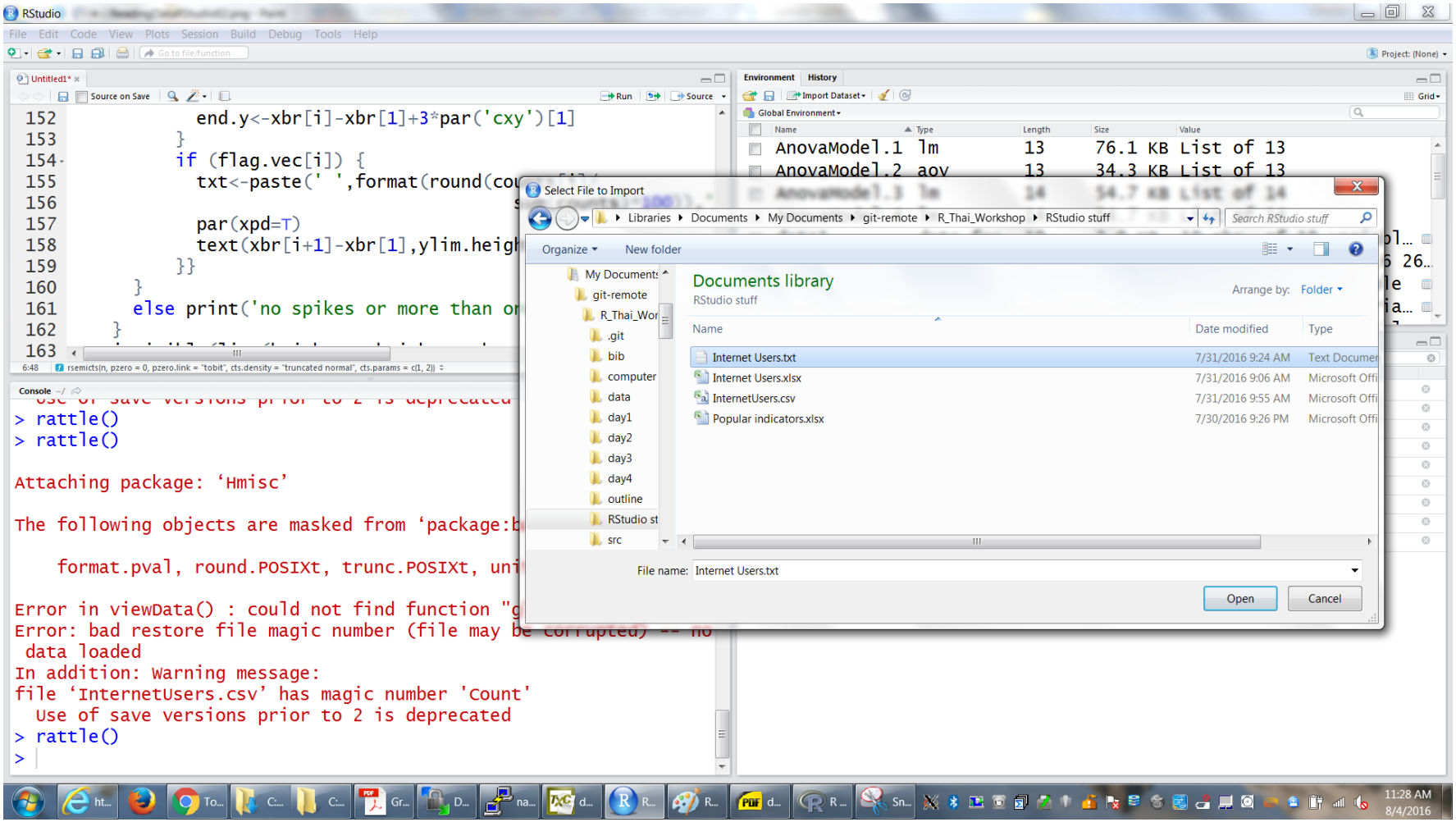
Reading Data in RStudio: Example

- Data set consists of annual data (2000-2014) on the 'Number of Internet users per 100' for 244 countries.
- Rcmdr has many options for reading datasets. A typical dataset with both character and numeric variables, either .csv or .xlsx type seem to be quite stable.
- rattle can also read R data frames, or directly from R scripts etc
- InternetUsers.csv data set consists of 244 rows and 17 rows. First two rows are 'Country Name' and 'Country Code'. The following 15 columns, IntUsers2000-IntUsers2014, are numeric data.
- Tools -> Import Data ->
- Browse down to the desired file

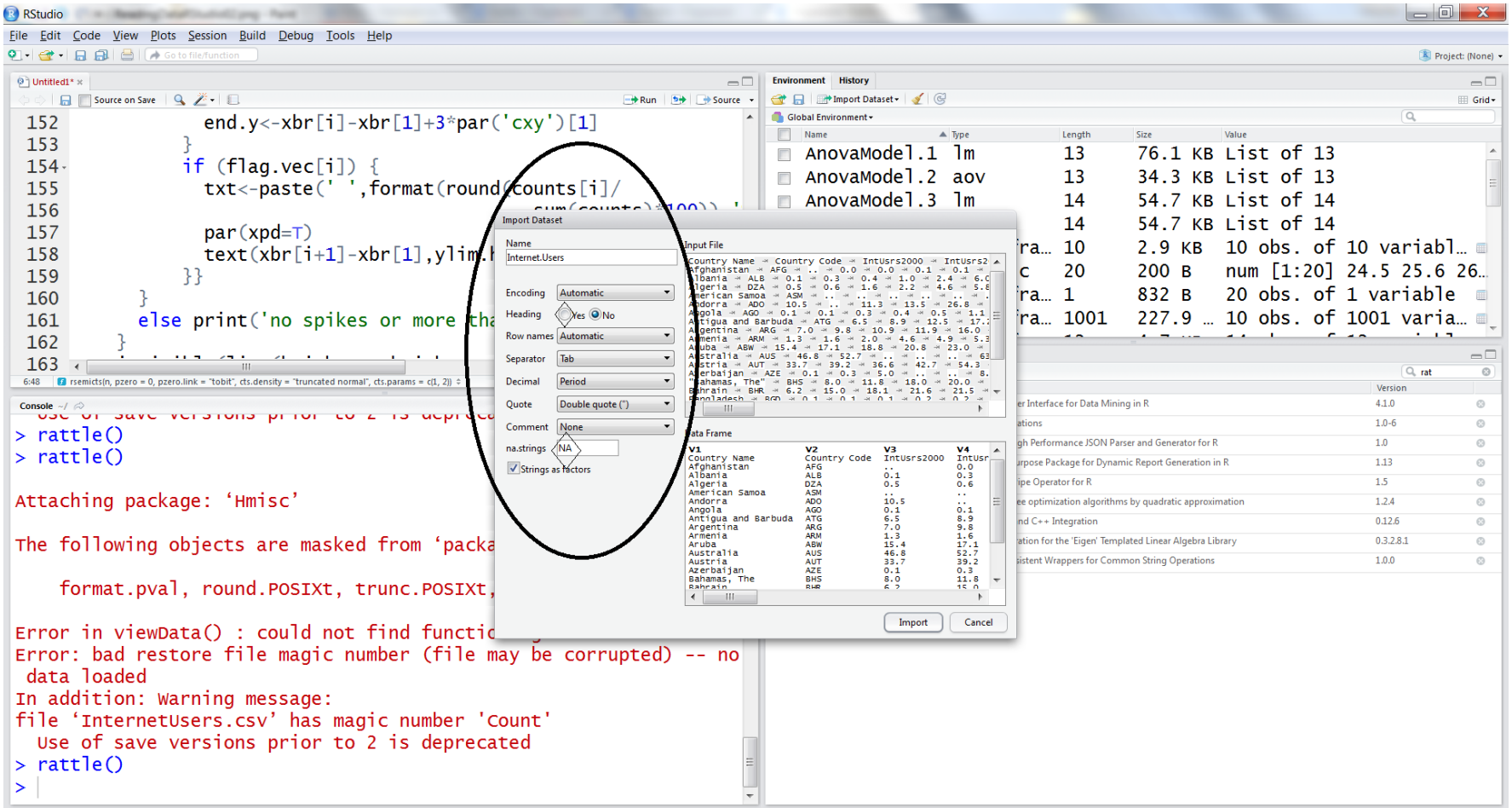
Reading Data in RStudio: Example



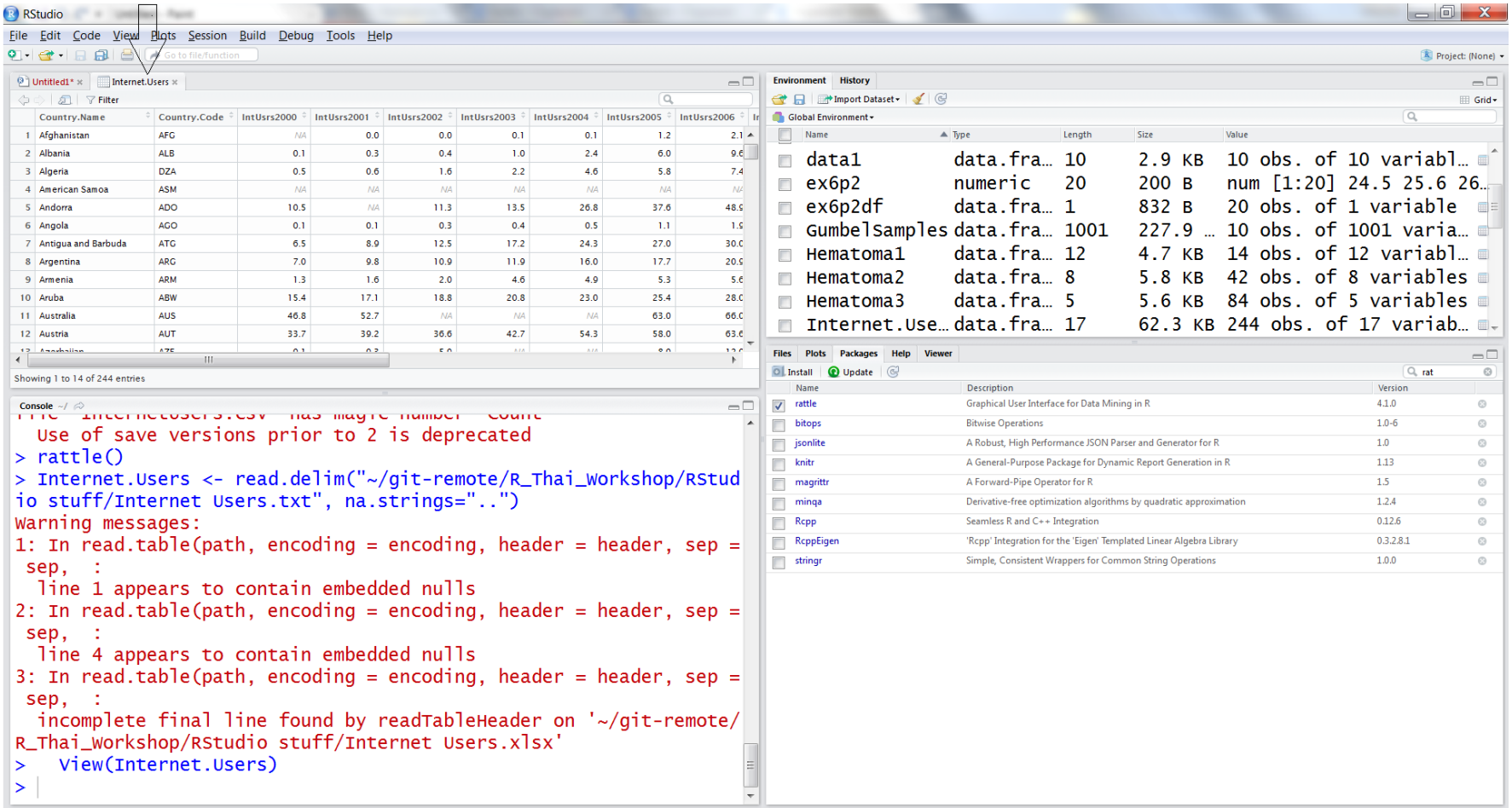
Reading Data in RStudio: Example



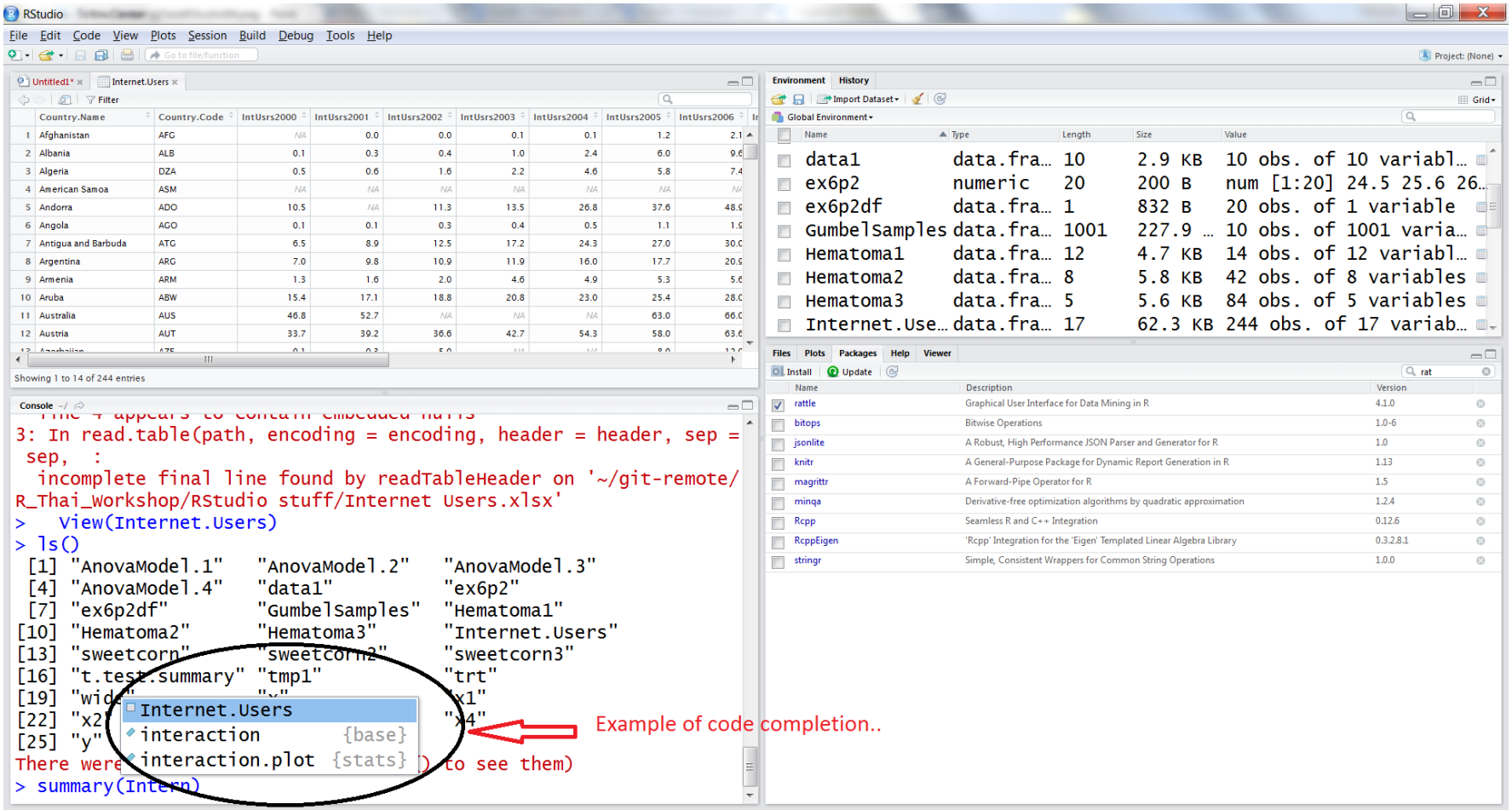
Reading Data in RStudio: Example



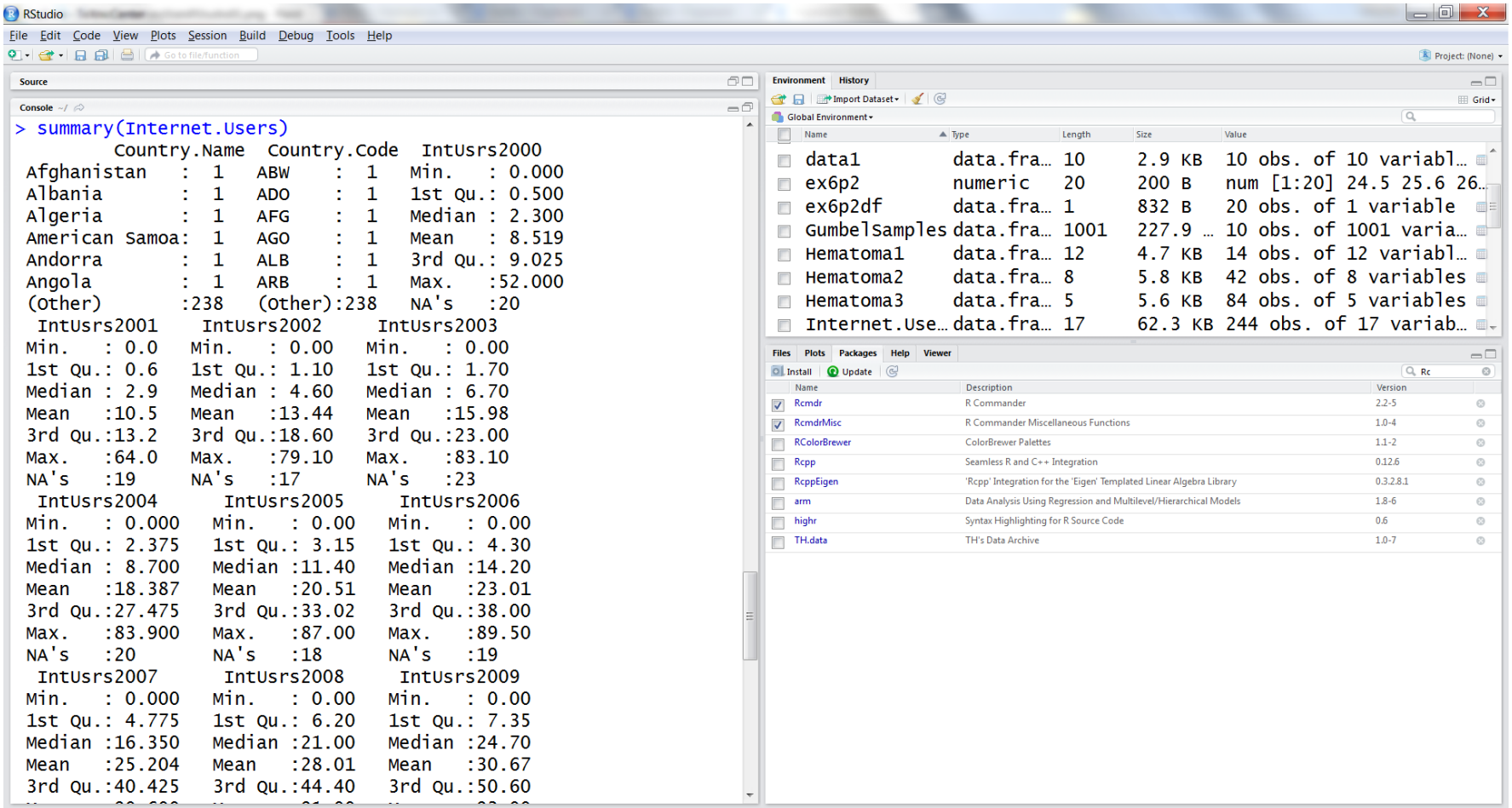
Reading Data in RStudio: Example



Descriptive Statistics in RStudio: Example



Descriptive Statistics in RStudio: Example

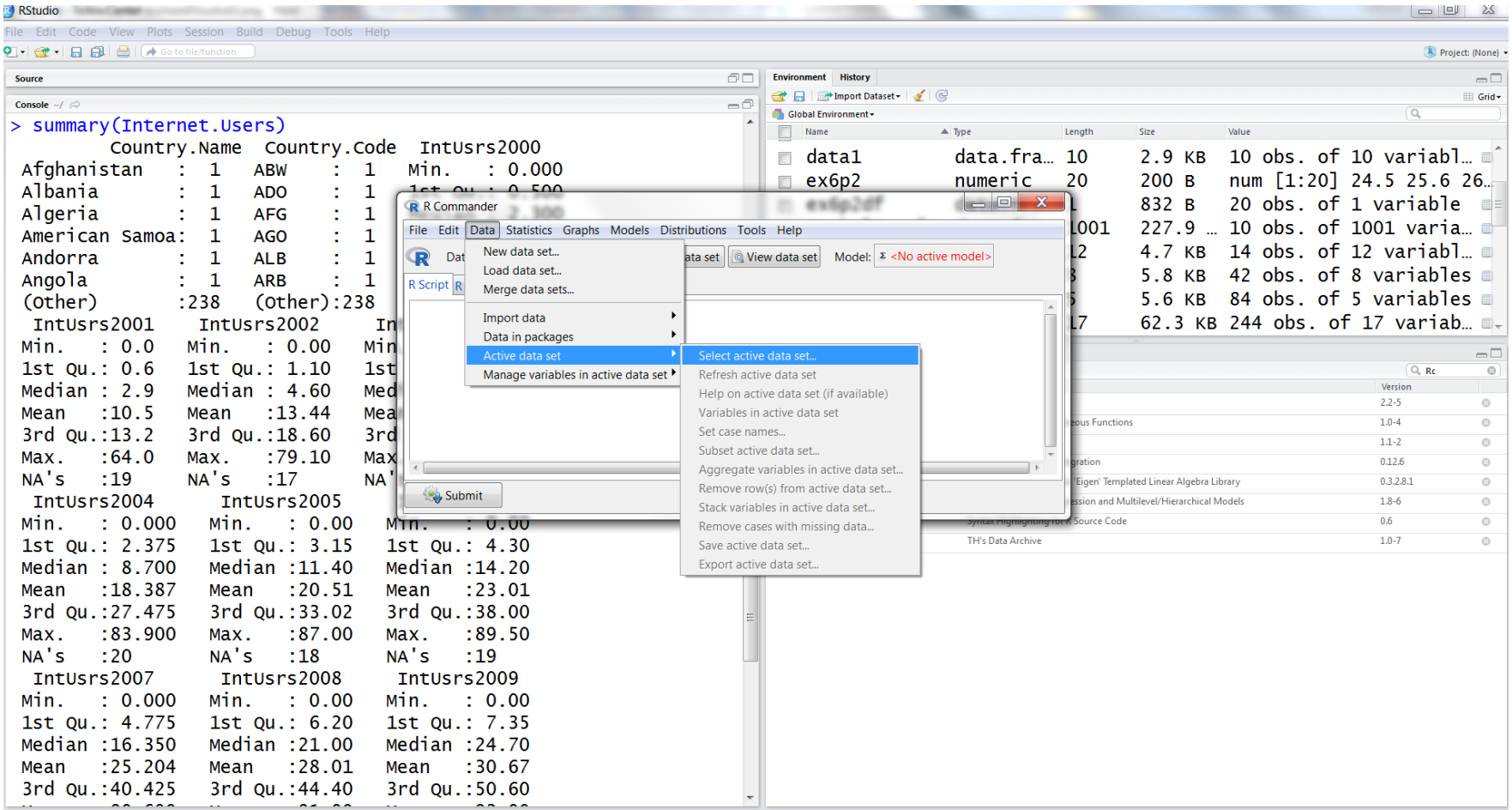


Using Rcmdr from within RStudio

- Once the data is in RStudio as a dataframe, it is available to R or any package there of
- All packages including those with their own GUI can be active simultaneously with RStudio
- Invoke Rcmdr as illustrated before and Rcmdr will open its GUI
- Results of Rcmdr commands are displayed in the RStudio Console
- The Environment pane is also updated accordingly. For example, if a new dataset is read into Rcmdr it will show up in the RStudio Environment pane. (Still need to attach it.)
- The History pane does not seem to be updated.

Descriptive Statistics in RStudio: Example

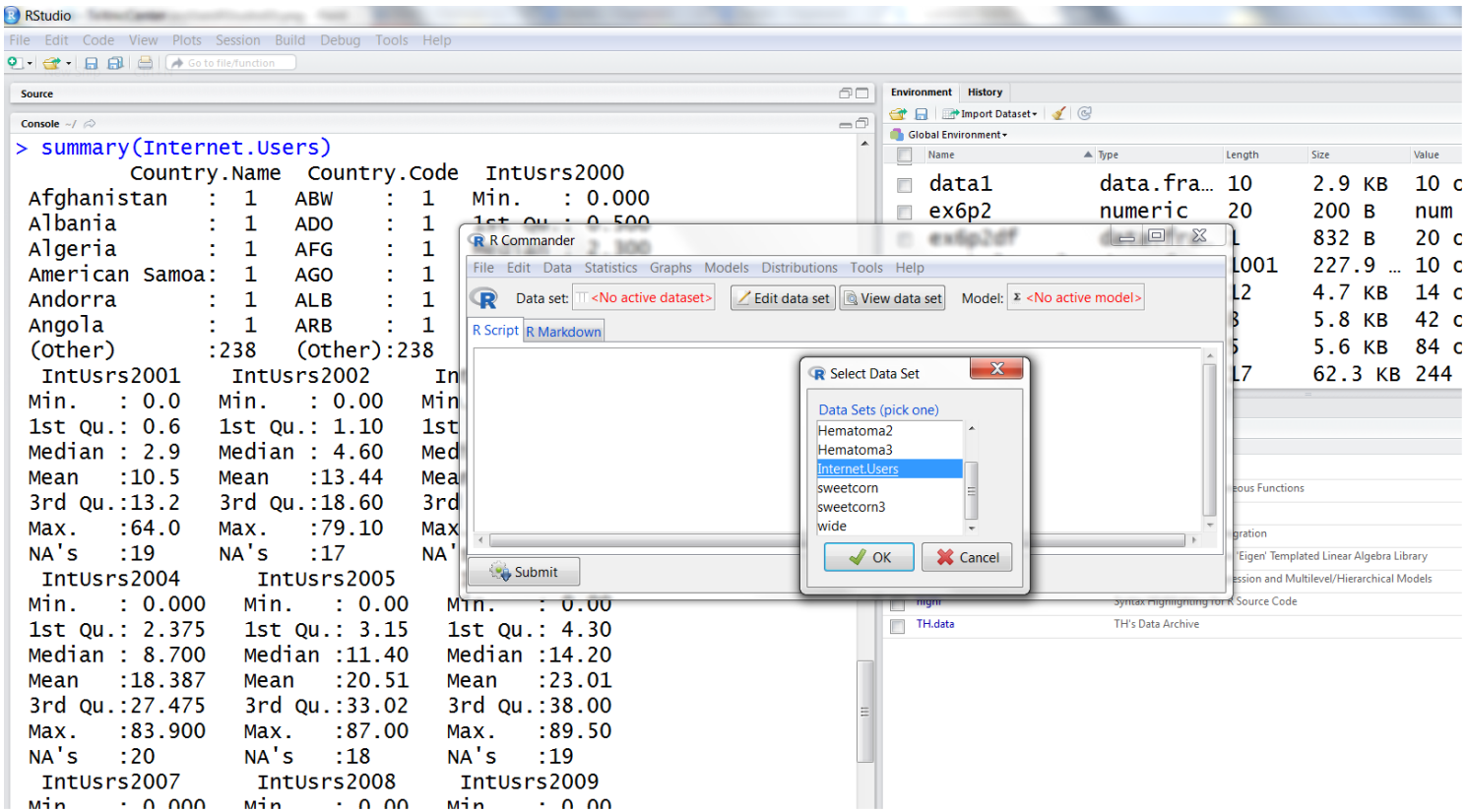
- First, attach the dataframe to Rcmdr as follows



Figure

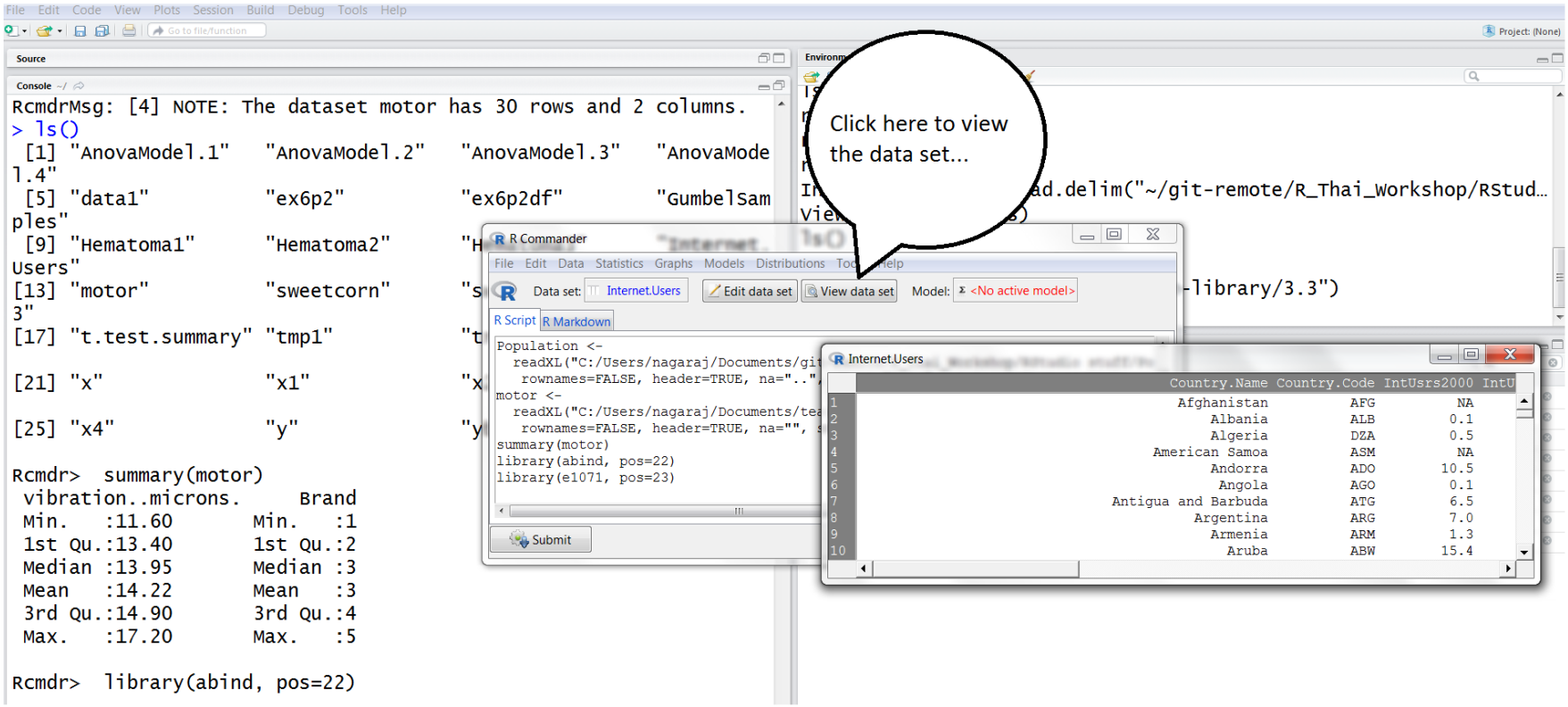
Descriptive Statistics in RStudio: Example

- First, attach the dataframe to Rcmdr



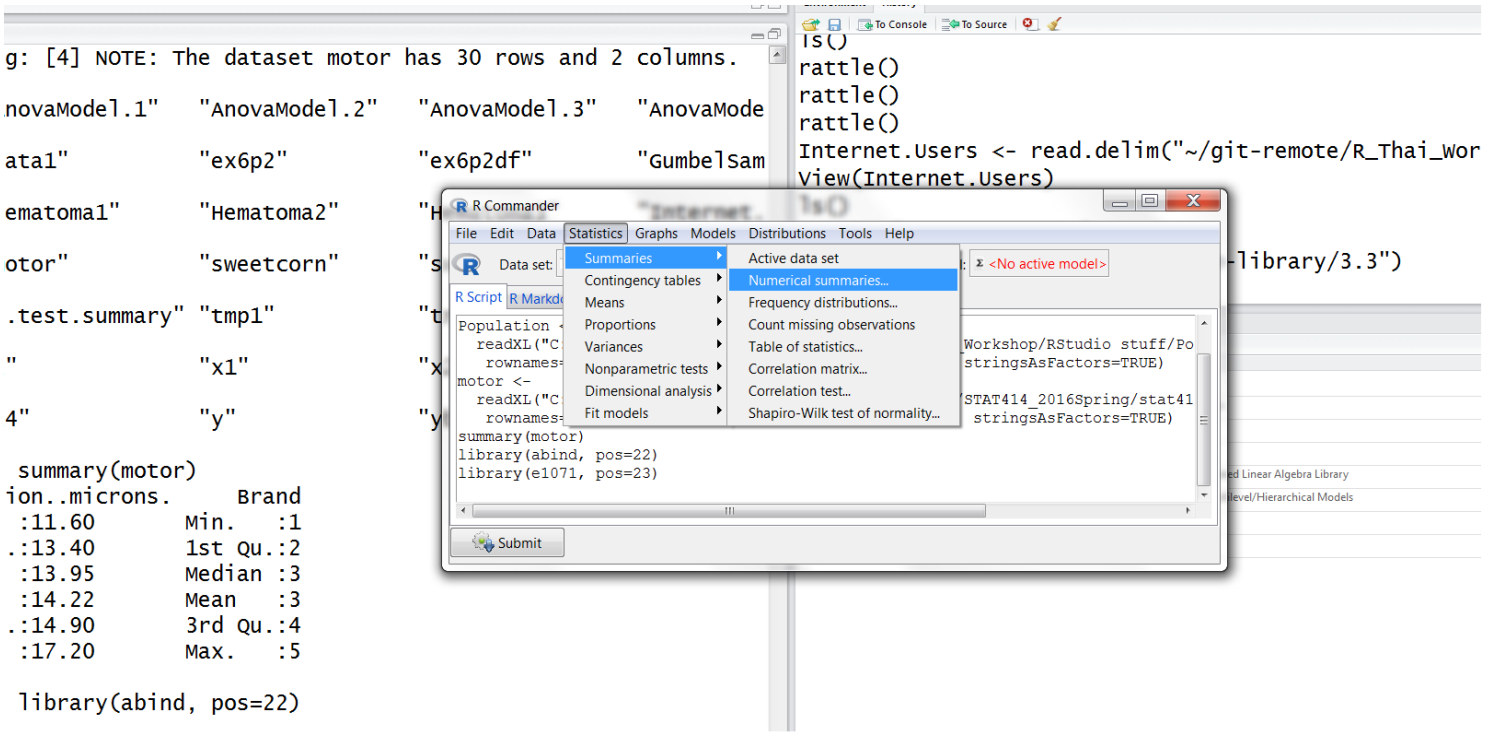
Descriptive Statistics in RStudio: Example

- Second, view the data set



Descriptive Statistics in RStudio: Example

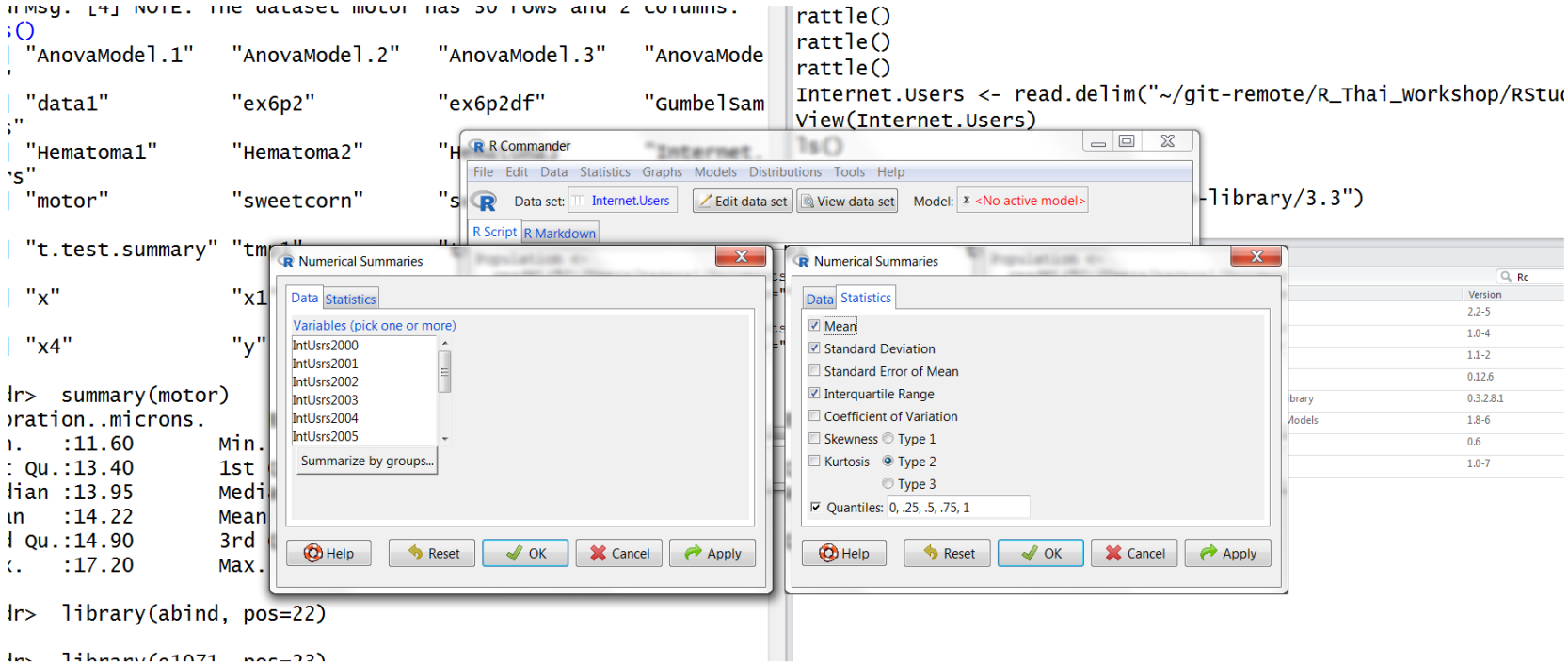
- Third, `Statistics > Summaries > Numerical summaries`



Descriptive Statistics in RStudio: Example

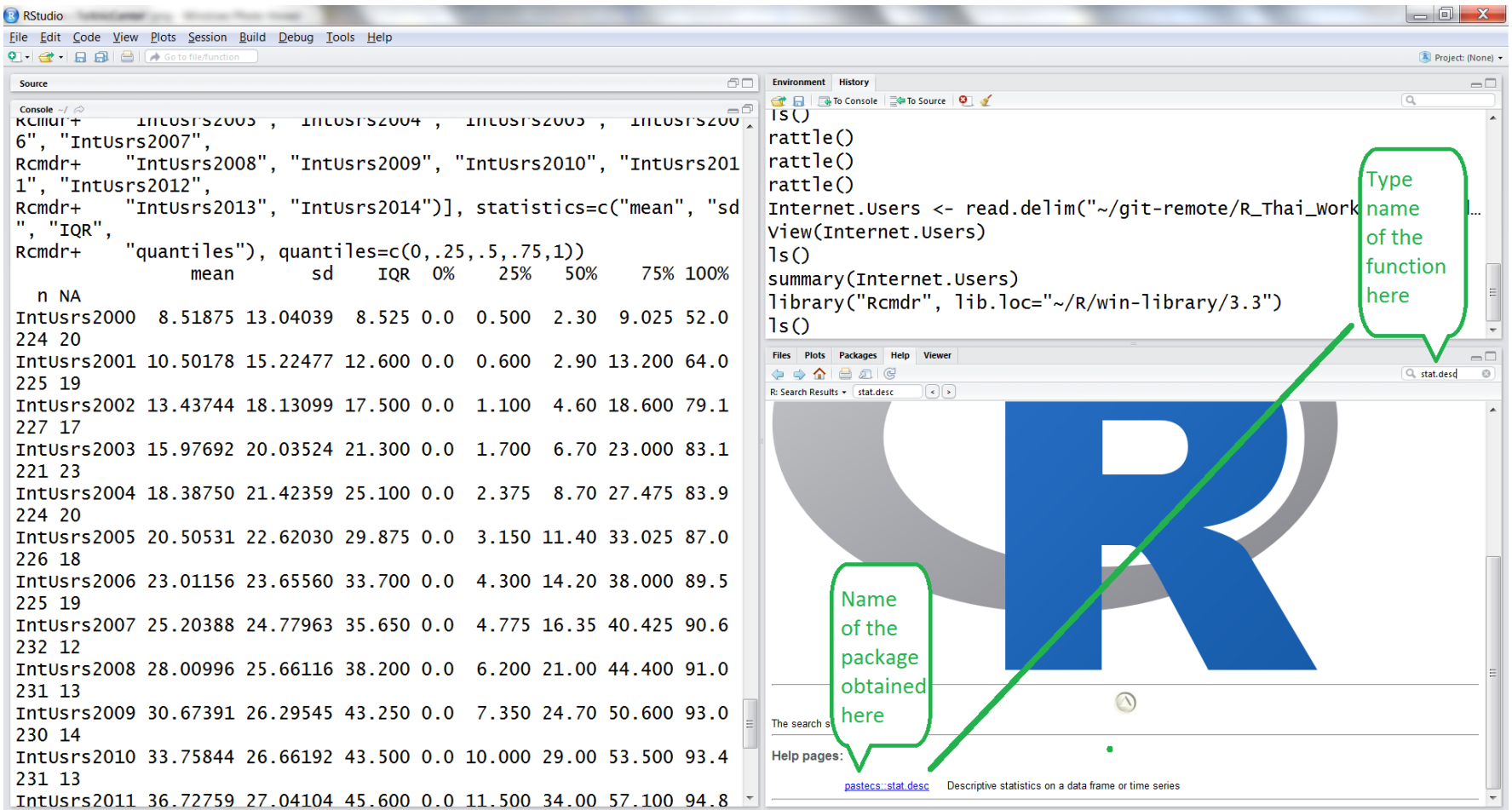
- Fourth, choose variable(s) and select options

- Finally, Click Apply or OK



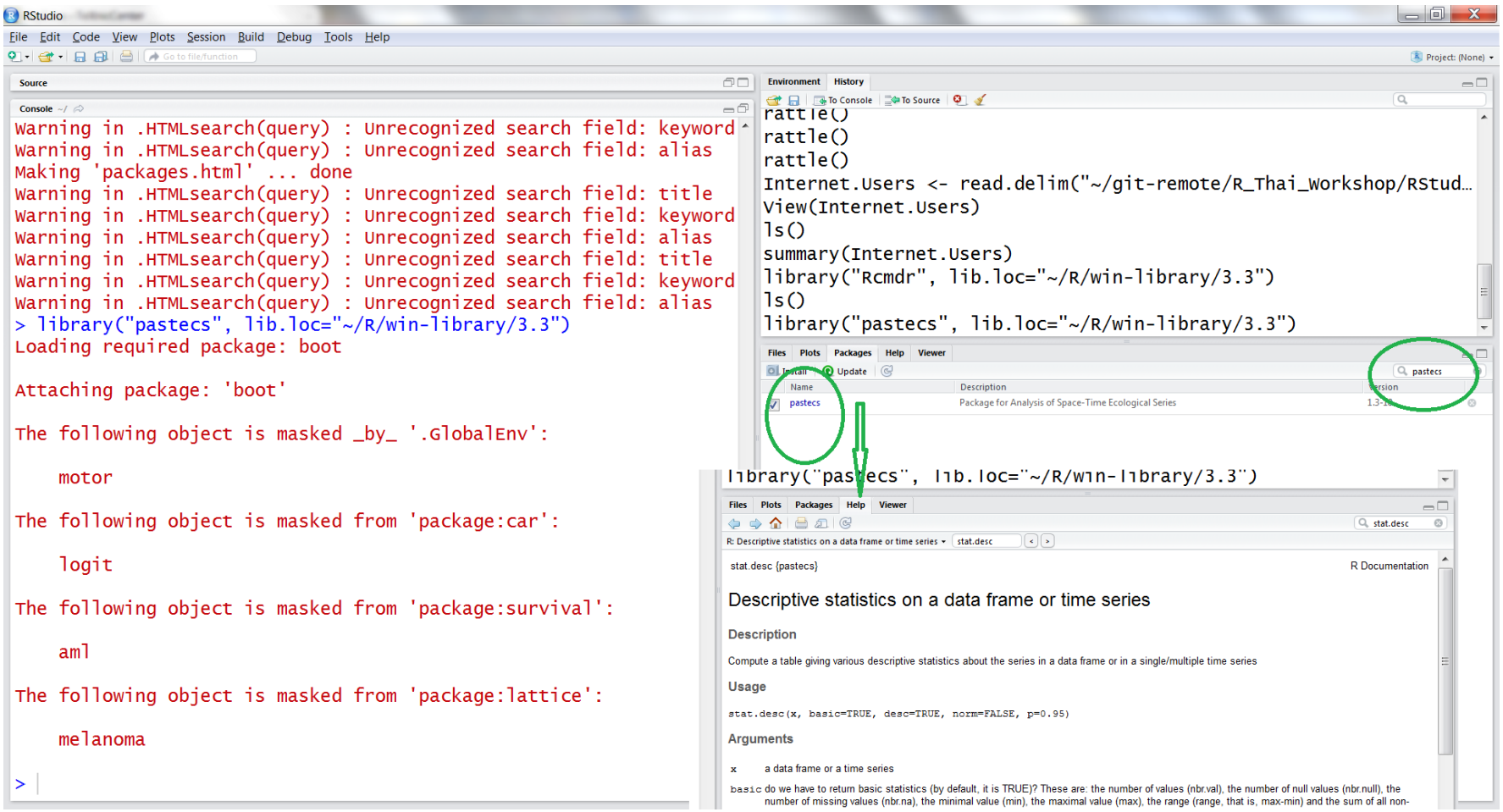
Descriptive Statistics in RStudio: Example

- Formatting of the Rcmdr summaries function is not great. We can look for other options. For example, there is a function called `stat.desc()`. How do we find it?



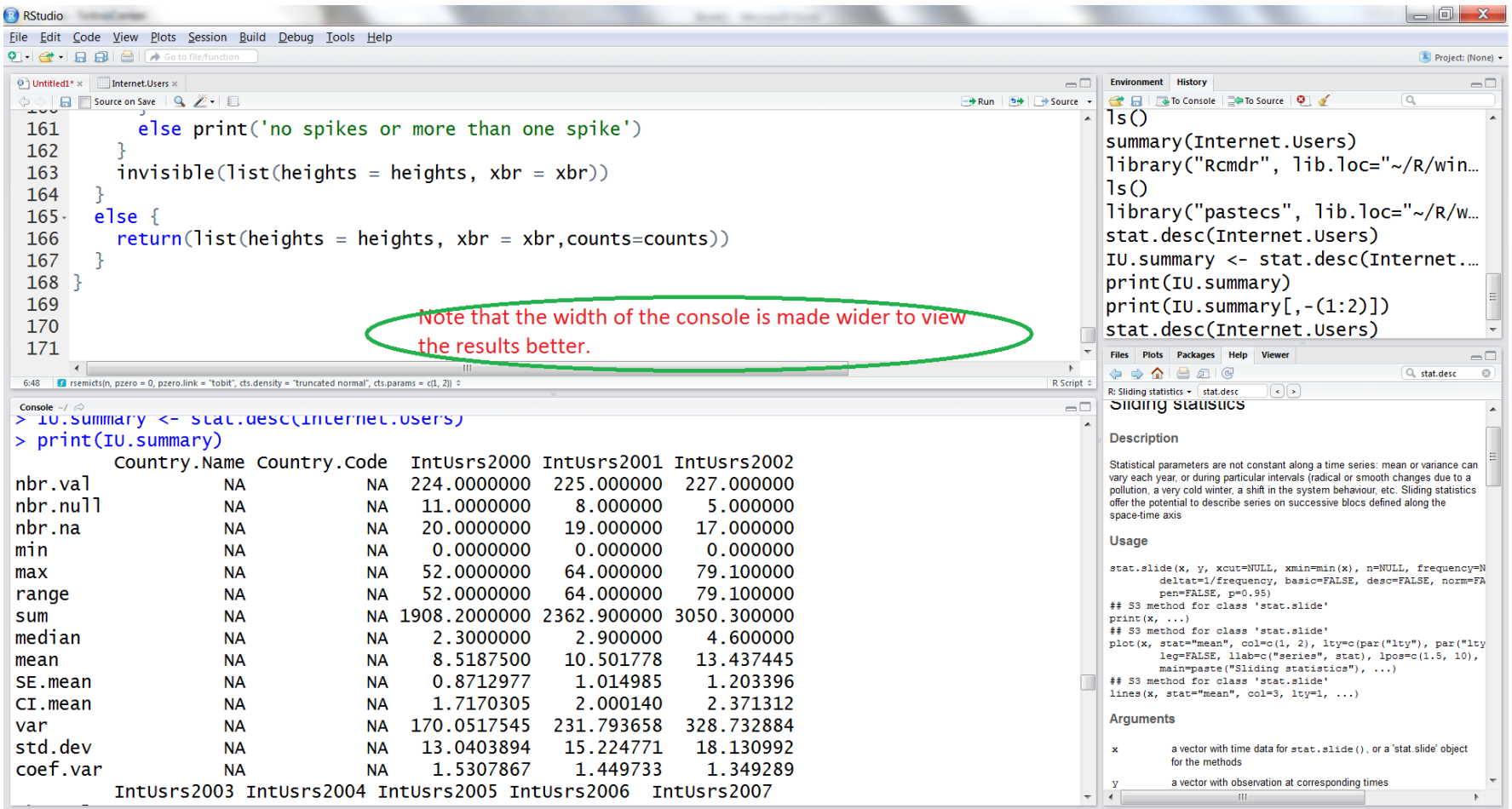
Descriptive Statistics in RStudio: Example

- Now we can install the package pastecs, and then get help for `stat.desc()`:



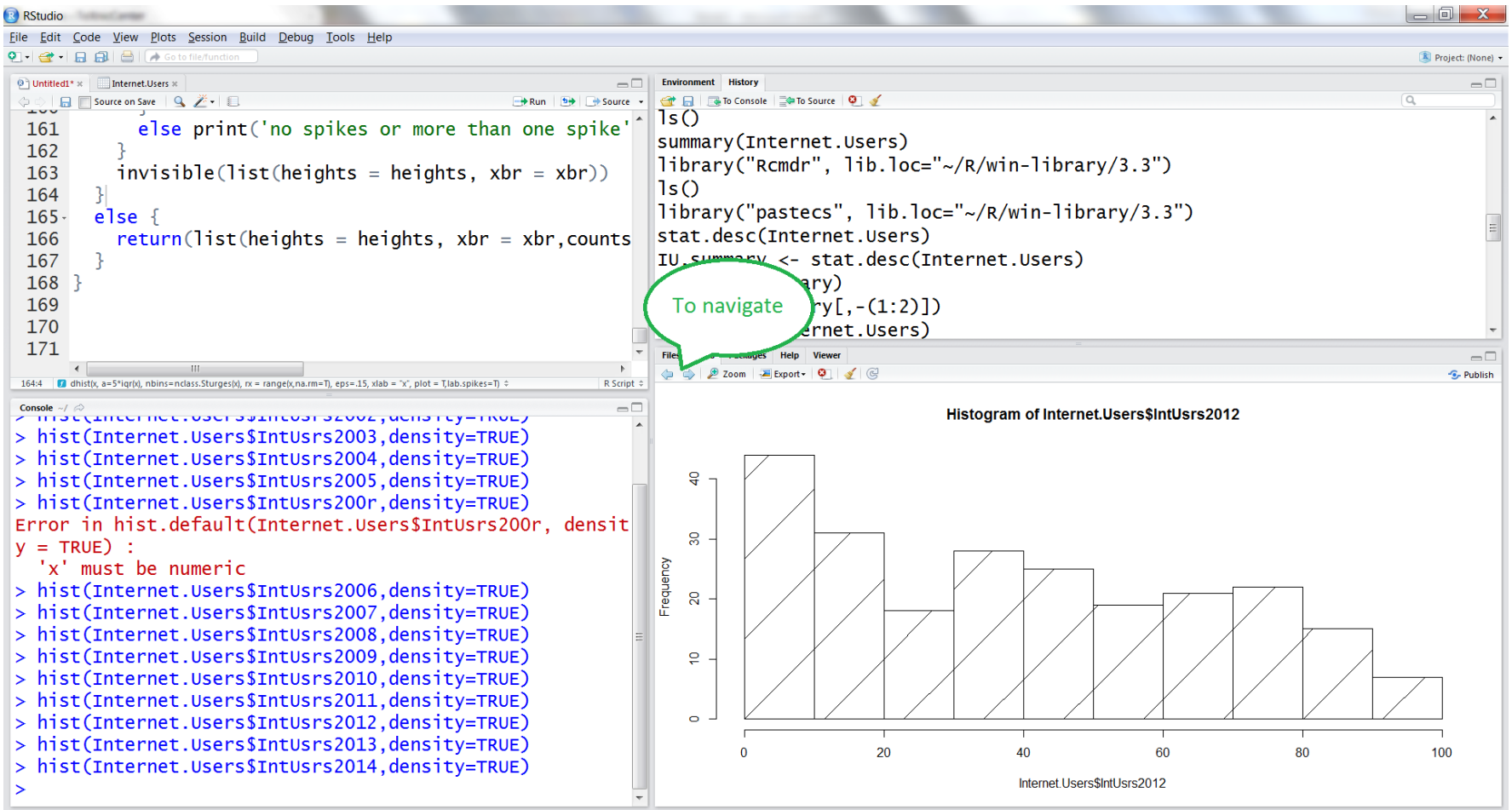
Descriptive Statistics in RStudio: Example

- These results may be `cut-and-pasted` to a report in Word or similar word processing tool



Graphs in RStudio

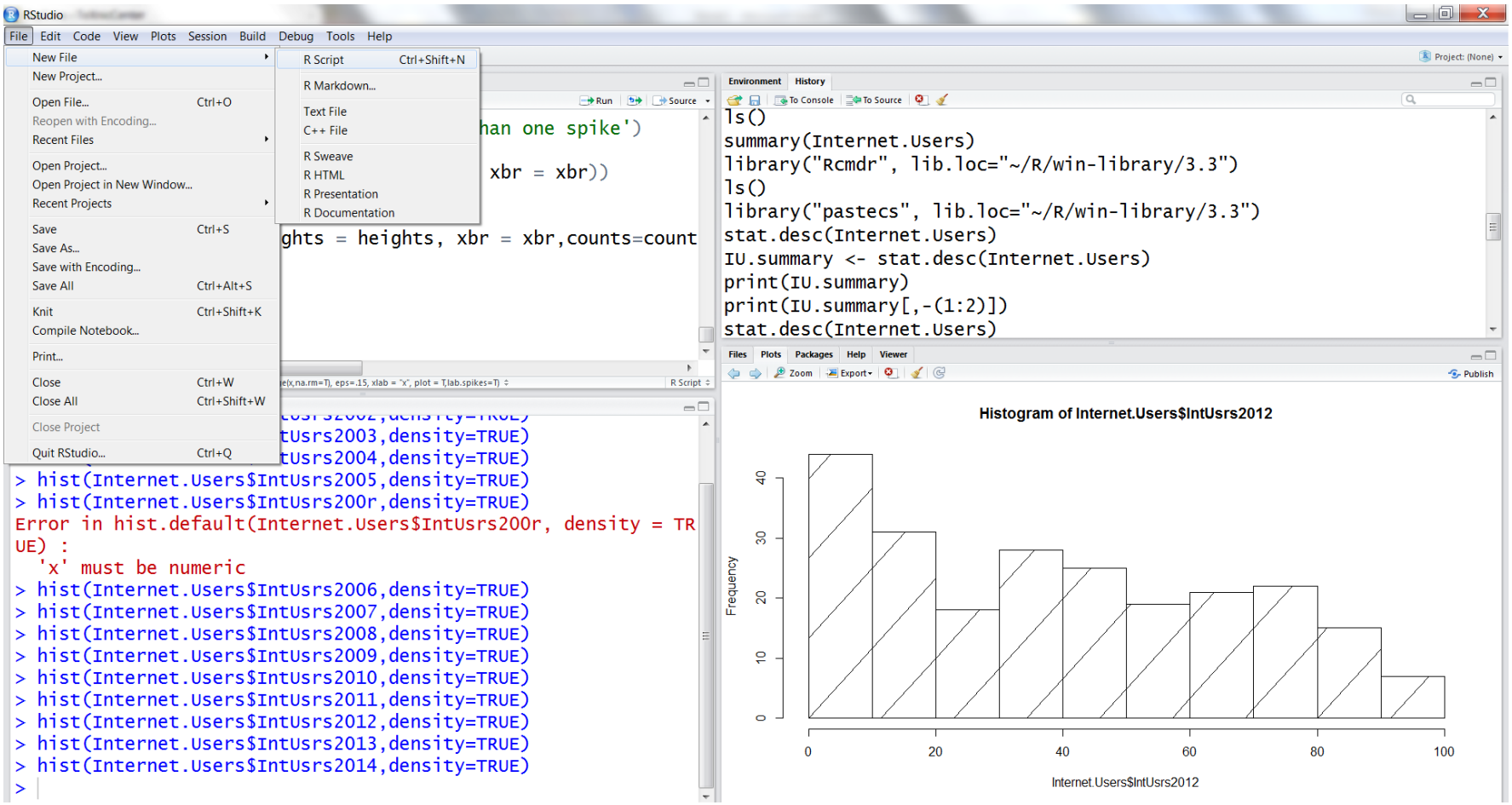
- Can use any R function that plots, and the plot will appear in the Plot pane. Successive plots will be saved, and can be navigated using the arrow buttons



Graphs in RStudio

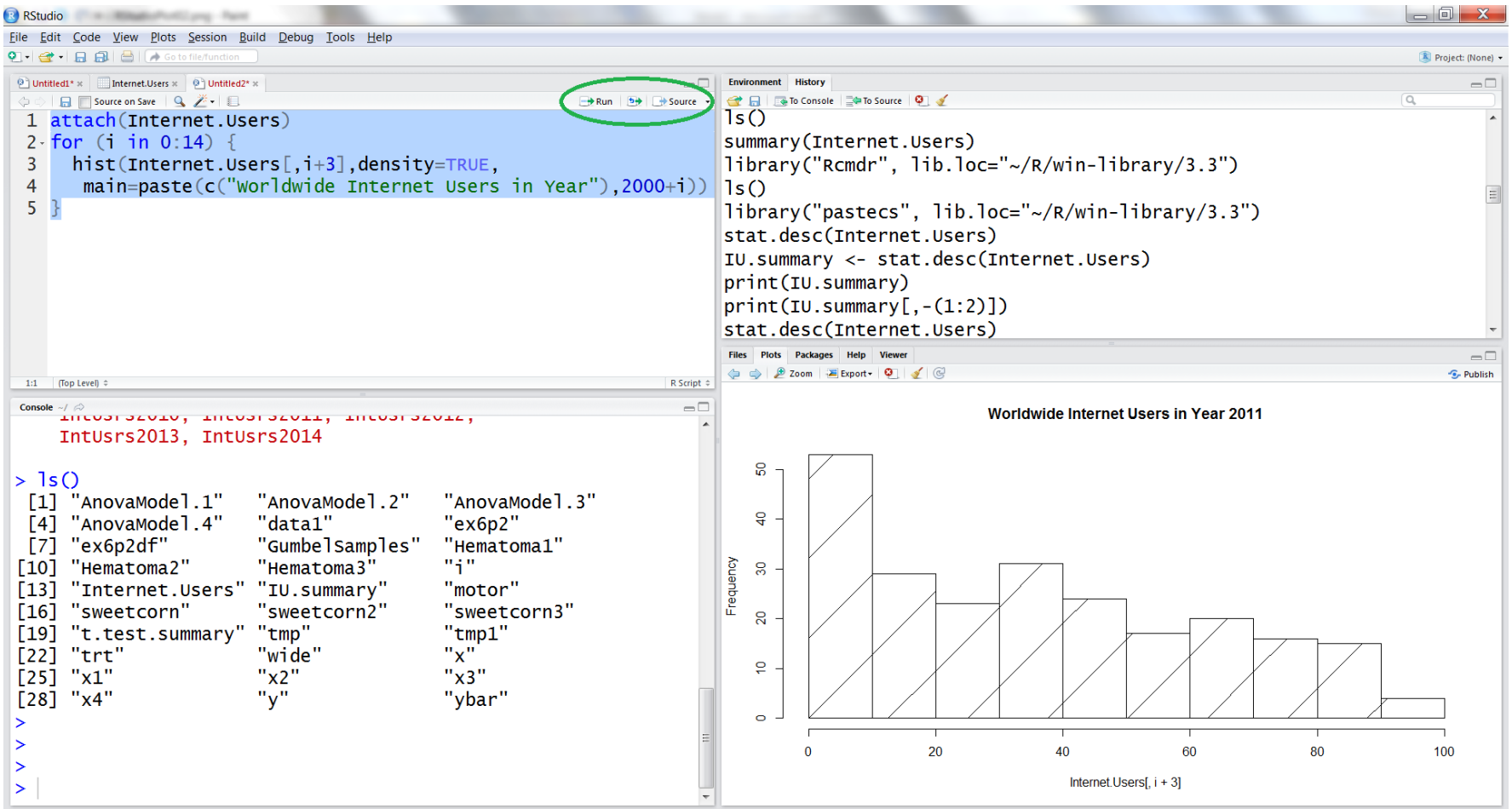
- A better program would be to write a loop.

- Start editing a R Script file as shown below.



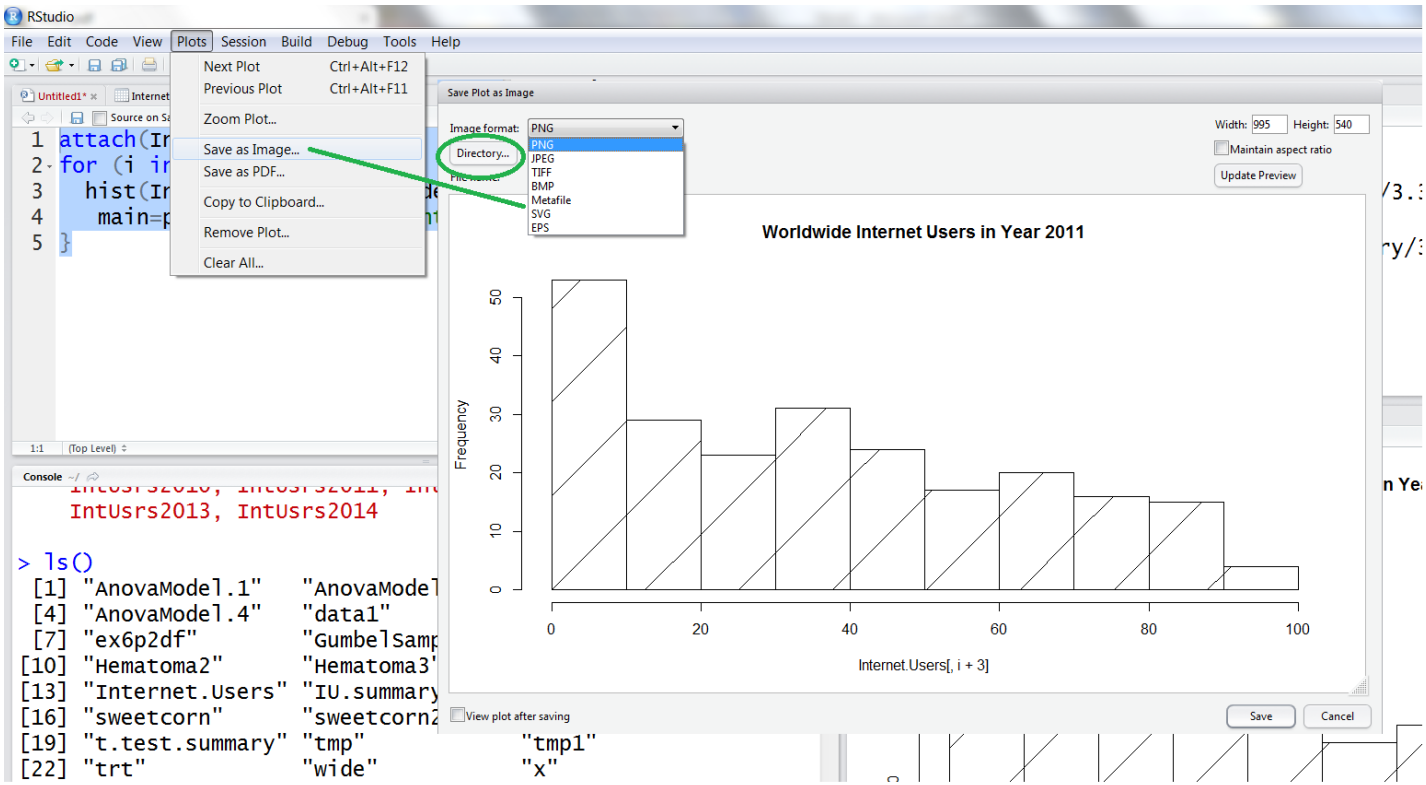
Graphs in RStudio

- Here's a simple loop to plot all histograms, with a trick to ensure the correct title for each histogram



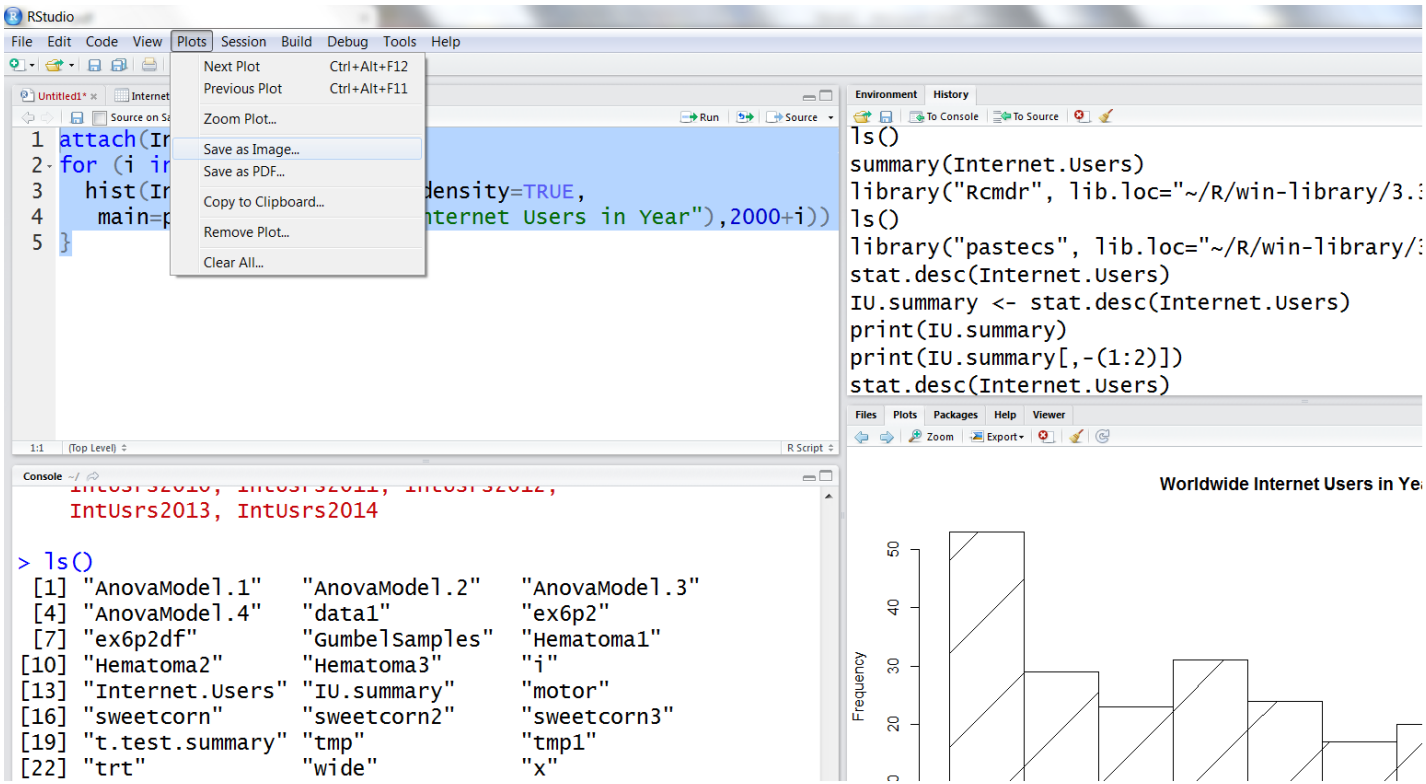
Graphs in RStudio

- Plots can be saved to the local directory: `Plots -> Save as Image...`



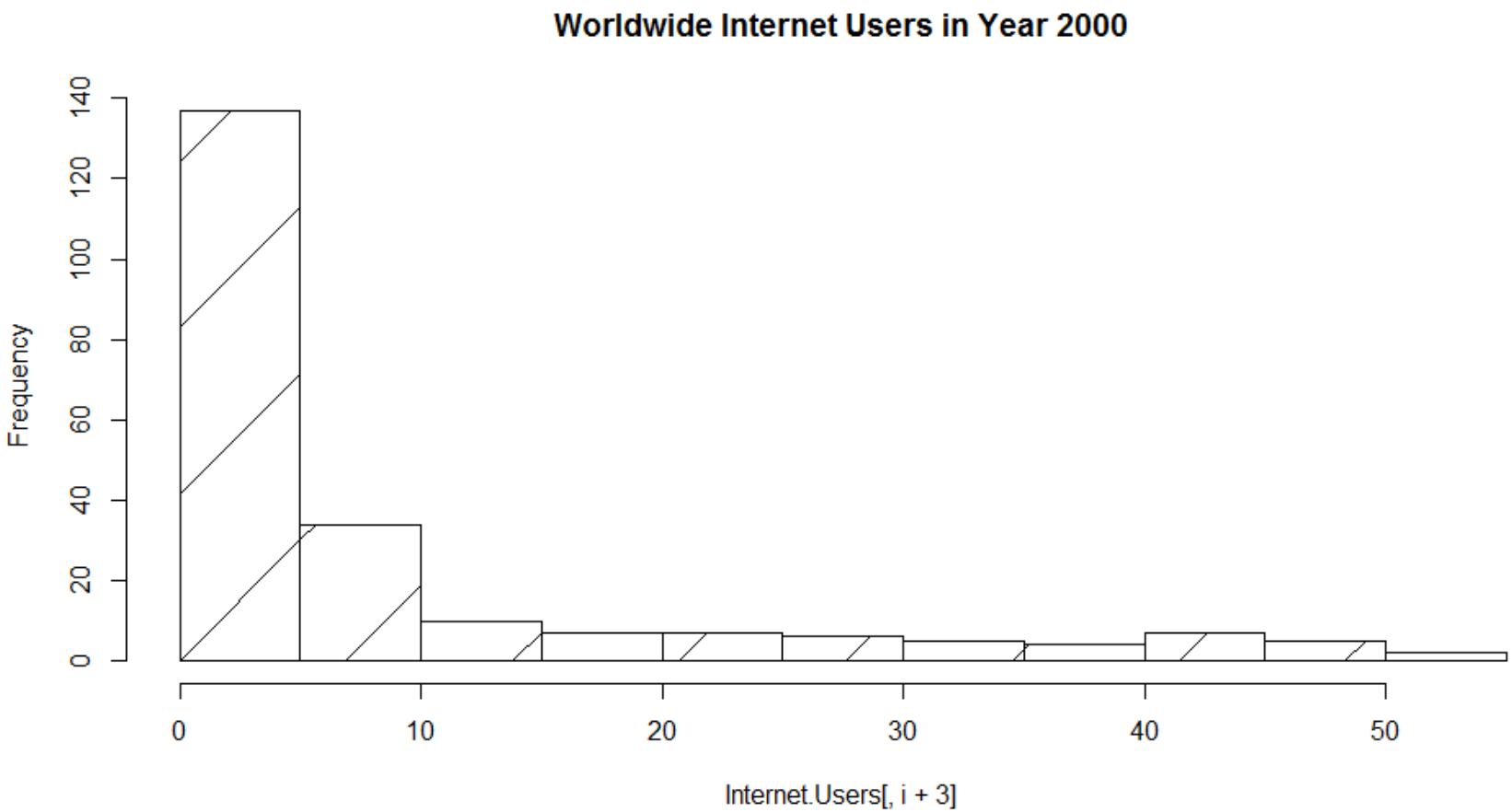
Graphs in RStudio

- Also note other options in the `Plots` menu:



Graphs in RStudio

- Plot is saved as a `filename.png` file and inserted into this tex document:



Scatterplot

RStudio

File Edit Code View Plots Session Build Debug Tools Help

Untitled1* | Internet.Users | Untitled2*

Source on Save | Run | Source

```
1 attach(Internet.Users)
2 for (i in 0:14) {
3   hist(Internet.Users[,i+3],density=TRUE,
4    main=paste(c("Worldwide Internet Users in Year"),2000+i))
5 }
6
7 plot(Internet.Users[,3],Internet.Users[,17],xlab="IntUsers200
8 ylab="IntUsers2014")
9 title(main="World Wide Intenet Usage 2014 vs 2000")
```

6:1 (Top Level) R Script

Environment | History

To Console | To Source

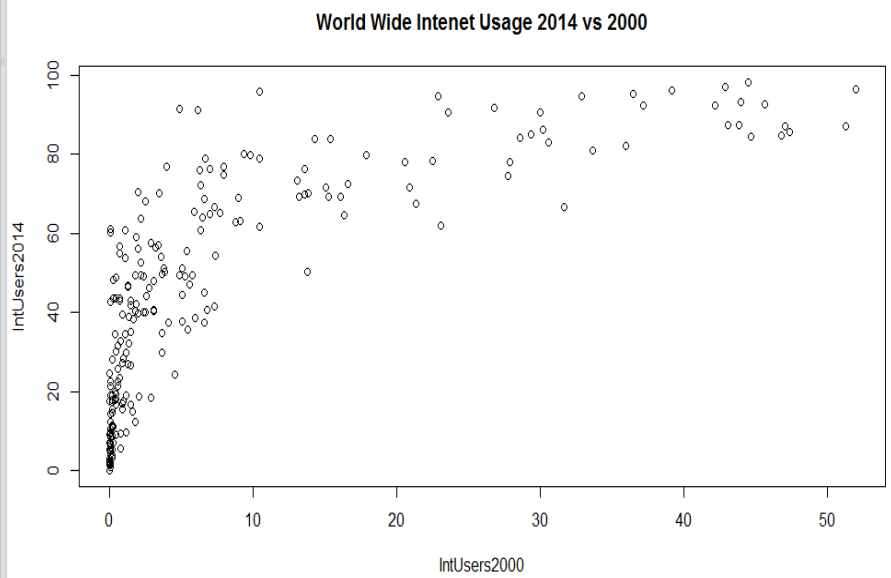
```
ls()
summary(Internet.Users)
library("Rcmdr", lib.loc="~/R/win-library/3.3")
ls()
library("pastecs", lib.loc="~/R/win-library/3.3")
stat.desc(Internet.Users)
IU.summary <- stat.desc(Internet.Users)
print(IU.summary)
print(IU.summary[,-(1:2)])
stat.desc(Internet.Users)
```

Console ~/

```
[19] "t.test.summary" "tmp"           "tmp1"
[22] "trt"            "wide"          "x"
[25] "x1"             "x2"            "x3"
[28] "x4"             "y"             "ybar"
>
>
>
>
>
> plot(Internet.Users[,3],Internet.Users[,17])
> plot(Internet.Users[,3],Internet.Users[,17],xlab="IntUsers20
00",
+ ylab="IntUsers2014")
> plot(Internet.Users[,3],Internet.Users[,17],xlab="IntUsers20
00",
+ ylab="IntUsers2014")
> title(main="World Wide Intenet Usage 2014 vs 2000")
>
```

Files | Plots | Packages | Help | Viewer

Zoom | Export | Publish



Side by Side Boxplots

- Suppose we want to compare the shapes of the distributions of Internet users over time.
- We need to reorganize the data before we can do this
- We can use the stack() function for this purpose

```
#   Reshaping the Internet Users Data
#   for plotting side-by-side boxplots
Internet.Users.Long <-
   stack(Internet.Users,
      measured=paste0(c("IntUsrs"),2000:2014))
```

- Other functions for such reorganization of data: melt(), cast(), data.table()
- Change the setting from Grid to List in the Environment pane. Click on the Internet.Users.Long data.frame to bring it to the Script window Or use `view()`

Side by Side Boxplots

RStudio

File Edit Code View Plots Session Build Debug Tools Help

Untitled1* | Internet.Users | Untitled2* | Internet.Users.Long

Filter

	values	ind
1	NA	IntUsrs2000
2	0.1	IntUsrs2000
3	0.5	IntUsrs2000
4	NA	IntUsrs2000
5	10.5	IntUsrs2000
6	0.1	IntUsrs2000
7	6.5	IntUsrs2000
8	7.0	IntUsrs2000
9	1.3	IntUsrs2000
10	15.4	IntUsrs2000
11	46.8	IntUsrs2000

Showing 1 to 11 of 3,660 entries

Environment | History

Import Dataset | List

Global Environment

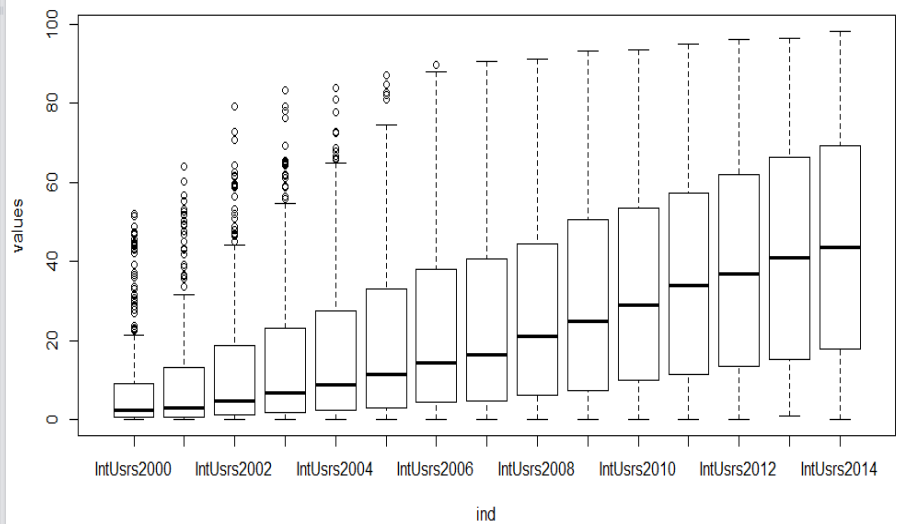
Hematoma1	14 obs. of 12 variables
Hematoma2	42 obs. of 8 variables
Hematoma3	84 obs. of 5 variables
Internet.Users	244 obs. of 17 variables
Internet.Users.L...	3660 obs. of 2 variables
IU.summary	14 obs. of 17 variables
motor	30 obs. of 2 variables
sweetcorn	12 obs. of 5 variables
sweetcorn3	60 obs. of 2 variables

Console ~/

```
od= y )
 [1] "144"  "34"   "161"  "139"  "186"  "11"   "185"  "102"
"86"   "137"
[11] "388"  "278"  "346"  "430"  "383"  "255"  "429"  "405"
"478"  "330"
[21] "574"  "632"  "673"  "541"  "554"  "522"  "674"  "625"
"600"  "590"
[31] "818"  "917"  "876"  "785"  "798"  "834"  "918"  "935"
"869"  "766"
[41] "1062" "1161" "1029" "1120" "1078" "1042" "1113" "1162"
"1040" "1010"
[51] "1273" "1306" "1357" "1364" "1405" "1550"

Rcmdr>  Boxplot(values~ind, data=Internet.Users.Long, id.meth
od="none")
> Boxplot(values~ind, data=Internet.Users.Long, id.method="no
ne")
> attach(Internet.Users.Long)
> View(Internet.Users.Long)
>
```

Files | Plots | Packages | Help | Viewer

Zoom | Export | Publish



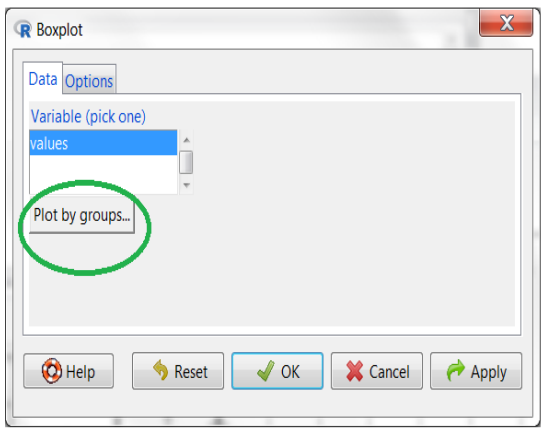
Side by Side Boxplots in Rcmdr

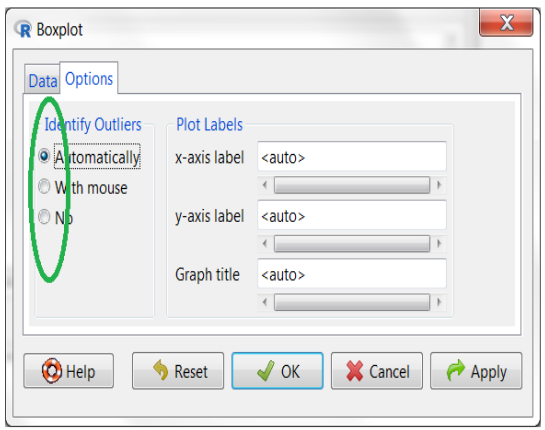
- Choose `Internet.Usrs.Long` to active data set in Rcmdr:

  `Data -> Active Data set -> Select active data set`

- Use `Graphs -> Boxplot....` to open the dialog box for boxplot

- Click on `Plot by groups` and select the variable Ind as the grouping variable.





So far....

- Introduced R, RStudio, Rcmdr
- Took a brief look at their capabilities
- Compared and contrasts their capabilities
- Saw examples of how Rcmdr and RStudio interact
- Saw examples of using Rcmdr from RStudio
- Saw examples of combining them to accomplish a specific task
- We will now discuss a very comprehensive package called `rattle`

Practice Excercise

- Descriptive Statistics
- Distributions
- Univariate statistical methods

Toxics Release Inventory Data

TRI: Industries meeting certain reporting requirements self-report annual amount of designated chemicals. Main variables in the data set are as follows.

- TRI Facility Identification (TRI_FACILITY_ID), Industry sector it belongs to, Parent company
- The name of the chemical or chemical category, carcinogenicity status
- Latitude, Longitude
- Total release to all media (Unit of reporting is *pounds per year*)

Toxics Release Inventory Data

This data can be downloaded from the United States Enviornmental Protection Agency website by following the steps below.

1. Navigate to TRI website: https://www.epa.gov/toxics-release-inventory-tri-program.

2. Click "Get TRI Data" in the middle of the page.

3. Near the bottom of the page under "More Information to Help You Use TRI Data", click on "Basic Data Files".

4. Now you should be at TRI Basic Data Files: Calendar Years 1987 - 2016 (https://www.epa.gov/toxics-release-inventory-tri-program/tri-basic-data-files-calendar-years-1987-2016).

Toxics Release Inventory Data

1. Perform all data summary and analysis steps shown in the previous slides for the TRI data set

2. Perform the following steps for the total release column
   - Compute the five number summary
   - Compute the percentages of data points in the intervals mean +-SD, mean +-2SD. What are these percentages expected to be and how close are they.
   - Repeat part b for a variety of transformations of the salary data. What transformation of the data provides a good fit of the data to a Gaussian assumption on the data?