

BIOS 721 | TOPIC 1: OBJECTS IN R – PART 1

Outline

2

□ Objects in R

▣ Introduce R and R Studio Software

- What are the moving parts?
- How do they work together?
- What do you need to know to get started?

▣ Objects in R

- Building blocks of programming in R

▣ Basic syntax

- How to create objects in R

What is “Programming”?

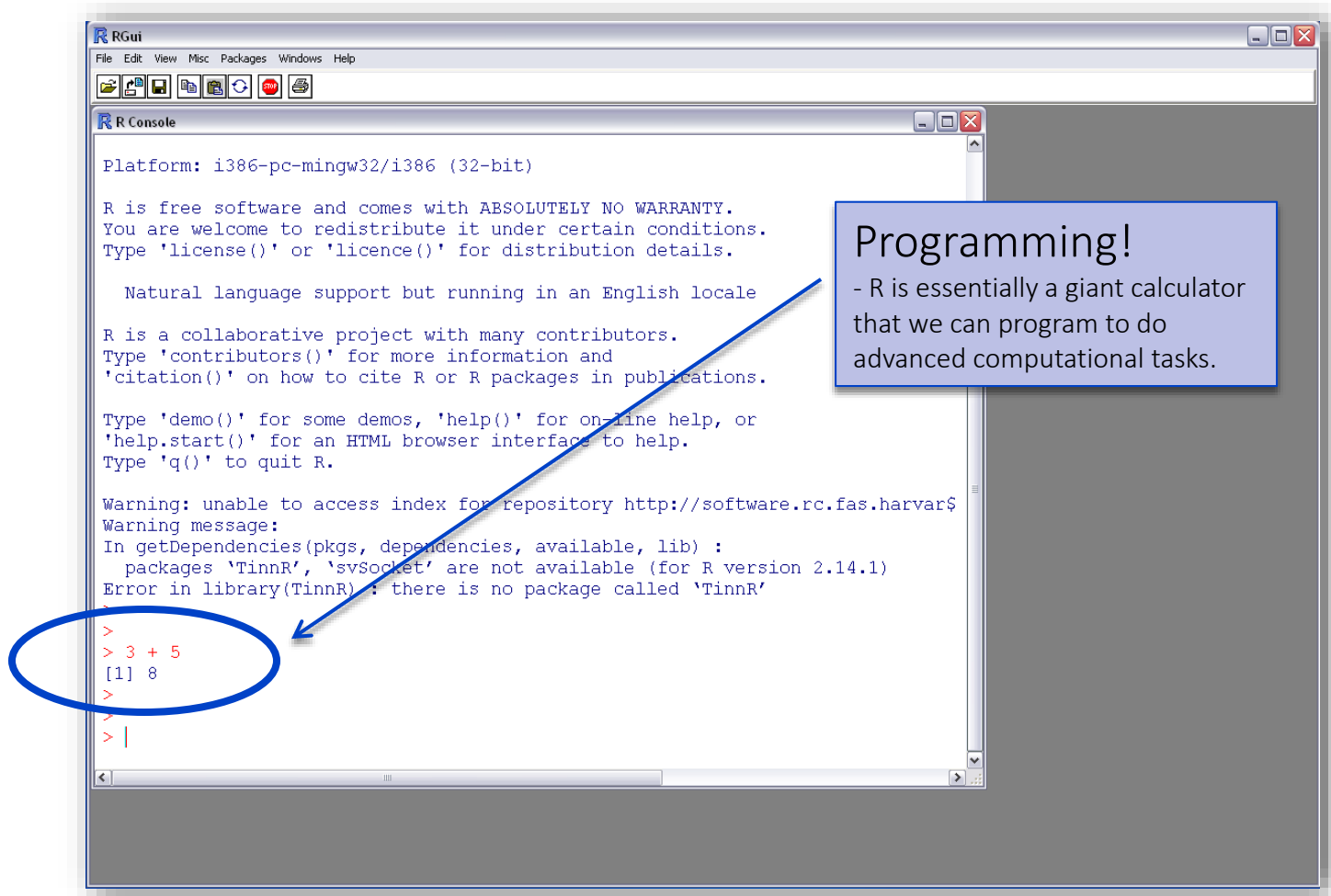
3

- In the simplest form it is ...
 - ▣ Writing a line of code
 - ▣ Submitting it to R
 - ▣ Software processes it
 - ▣ Output is returned

- Example| Submit the code `3+5` to R
 - ▣ `>` is the command prompt
 - ▣ Type `3+5` and hit Enter/Return to submit
 - ▣ R processes the code and returns `[1] 8` as the output/answer

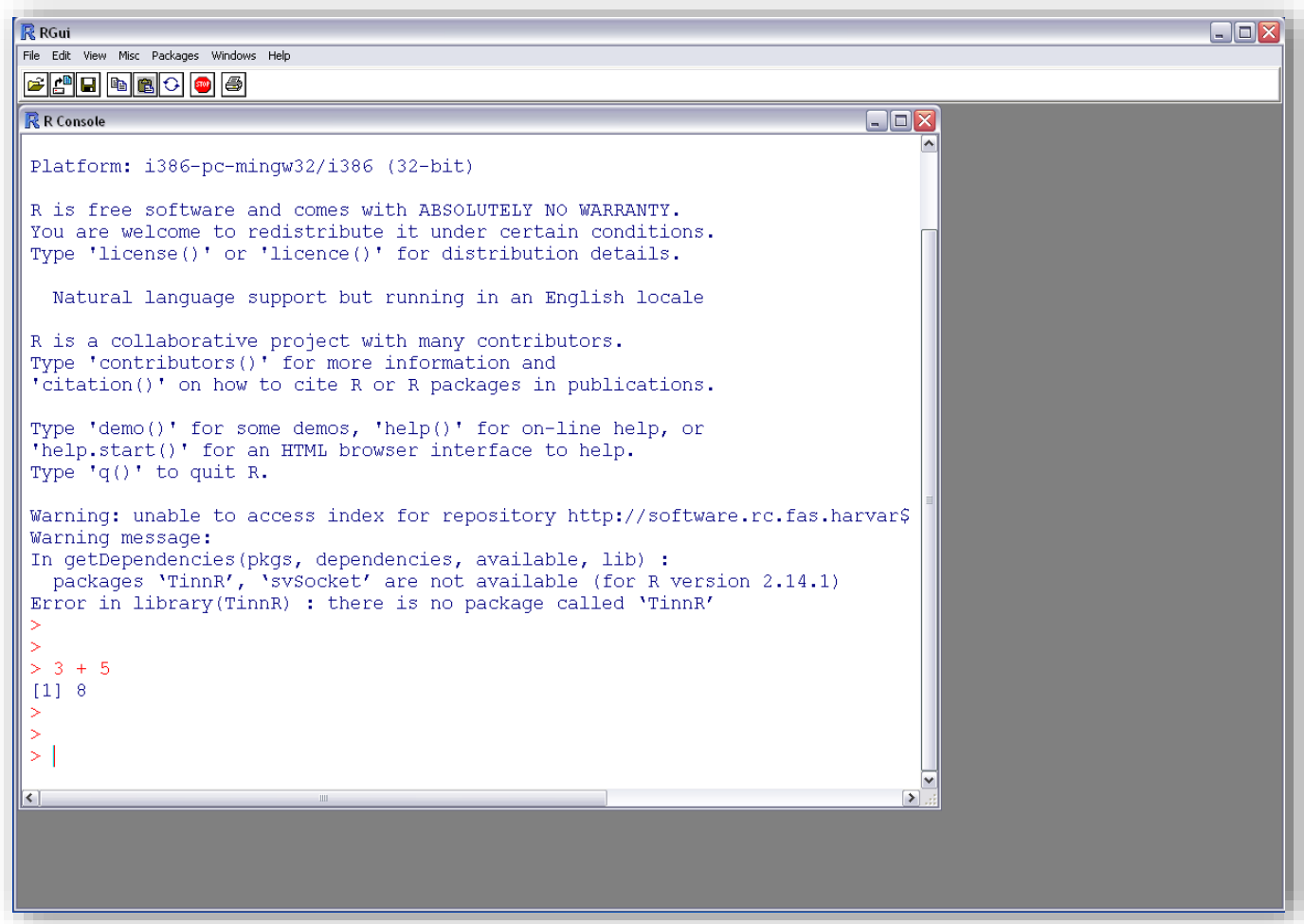
What is “Programming”?

4



Moving Parts in R

5



The screenshot shows the RGui application window. The title bar reads 'RGui'. The menu bar includes 'File', 'Edit', 'View', 'Misc', 'Packages', 'Windows', and 'Help'. Below the menu bar is a toolbar with icons for file operations and running code. The main area is the 'R Console', which displays the following text:

```
Platform: i386-pc-mingw32/i386 (32-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.

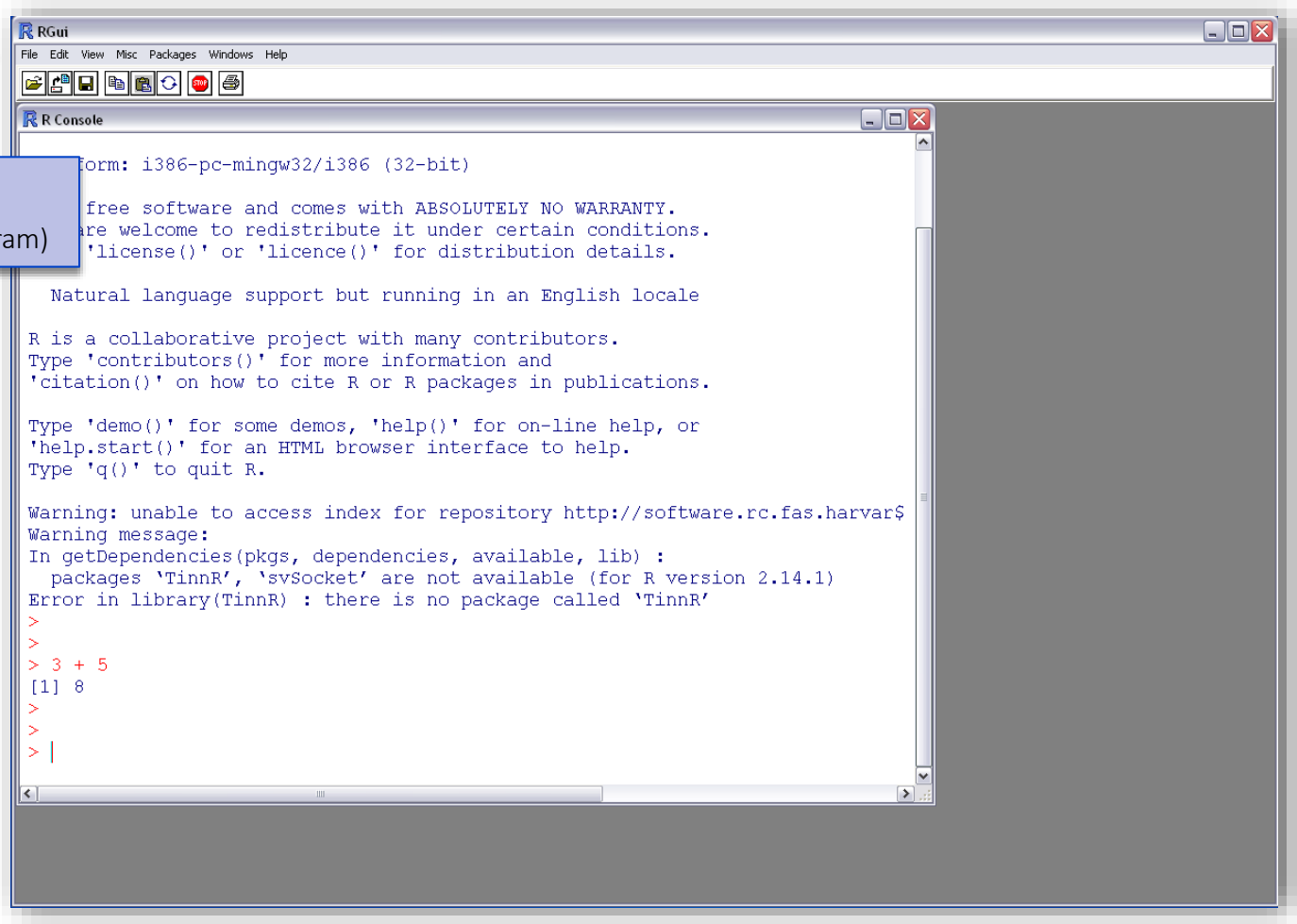
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Warning: unable to access index for repository http://software.rc.fas.harvar$
Warning message:
In getDependencies(pkgs, dependencies, available, lib) :
  packages 'TinnR', 'svSocket' are not available (for R version 2.14.1)
Error in library(TinnR) : there is no package called 'TinnR'
>
>
> 3 + 5
[1] 8
>
>
> |
```

Moving Parts in R

6

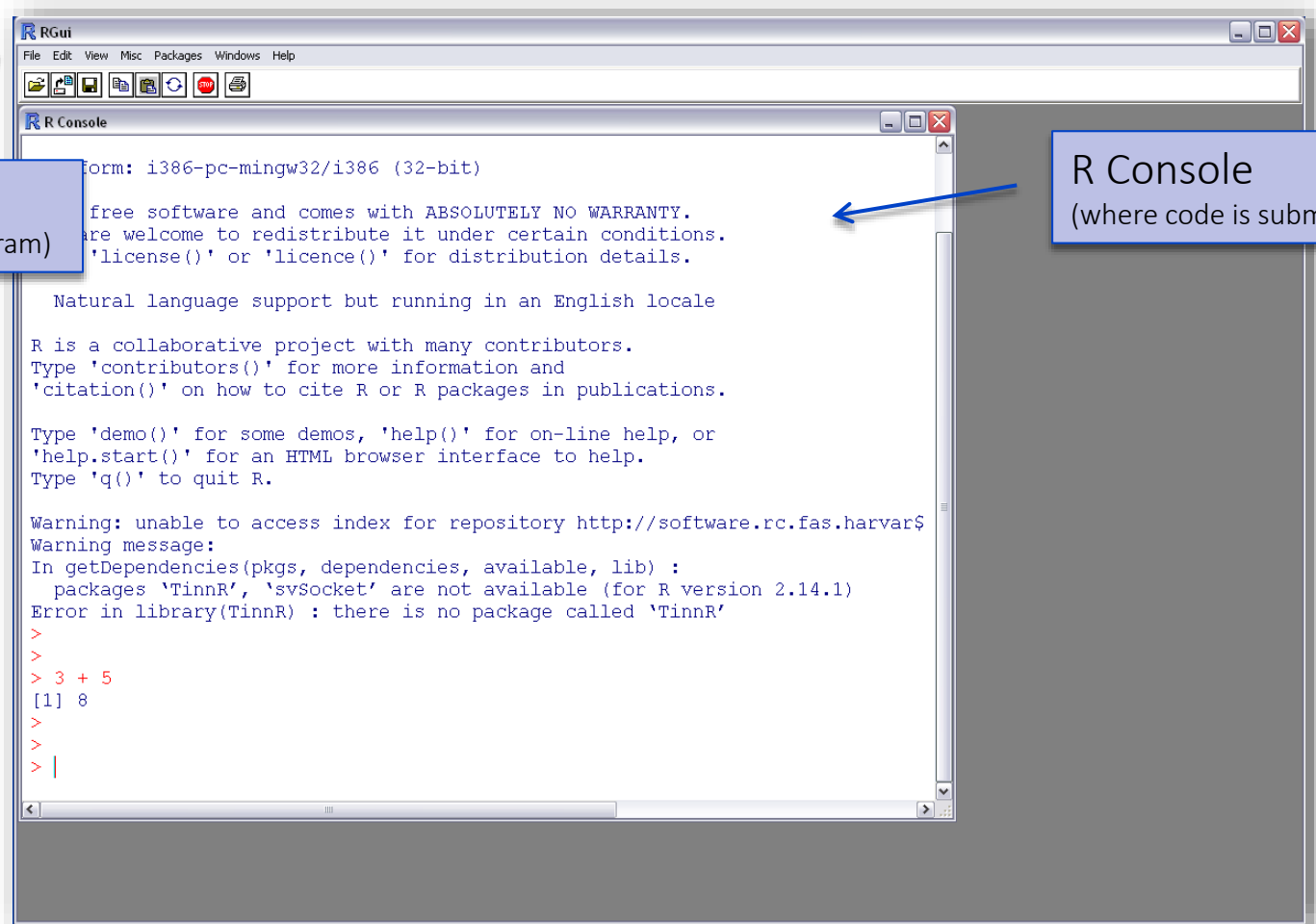
R GUI
(i.e. the software/program)



Moving Parts in R

7

R GUI
(i.e. the software/program)



R Console
(where code is submitted)

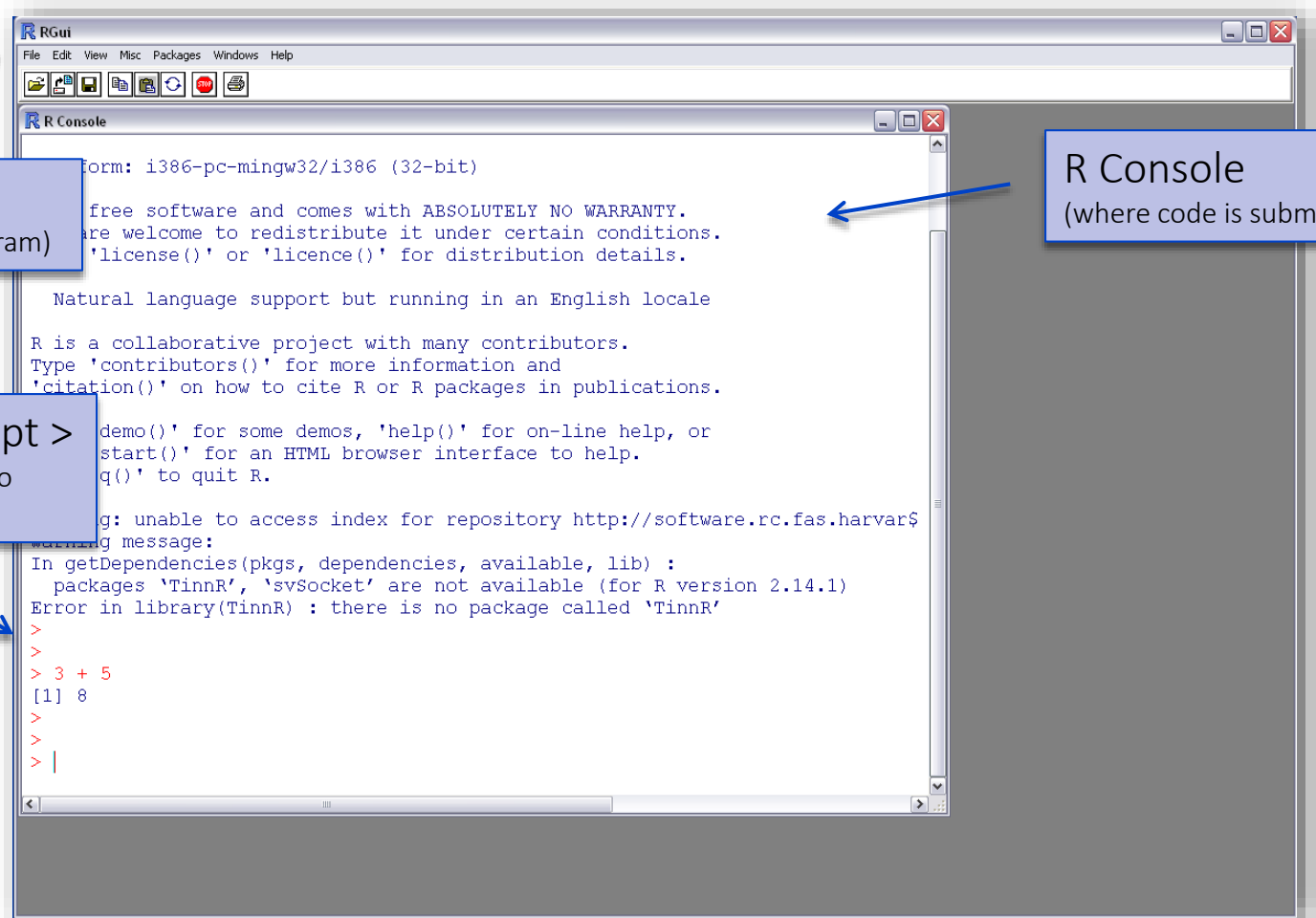
Moving Parts in R

8

R GUI
(i.e. the software/program)

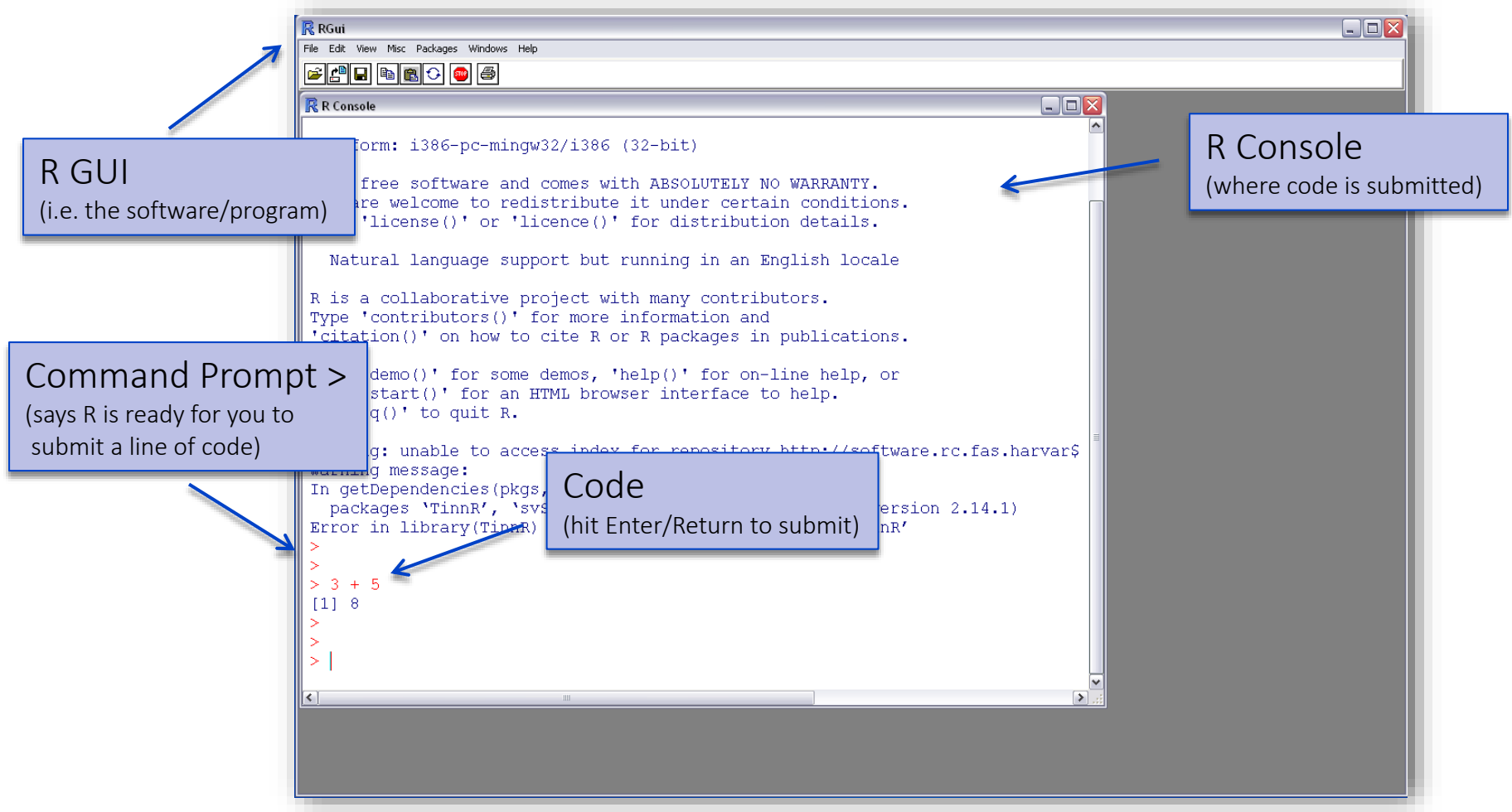
Command Prompt >
(says R is ready for you to
submit a line of code)

R Console
(where code is submitted)



Moving Parts in R

9



Moving Parts in R

10

The image shows a screenshot of the R GUI (RStudio) interface. The main window is titled 'RGui' and contains a menu bar (File, Edit, View, Misc, Packages, Windows, Help) and a toolbar. Below the menu bar is the 'R Console' window, which displays the R startup message and the user's input and output. The console shows the following text:

```
form: i386-pc-mingw32/i386 (32-bit)

free software and comes with ABSOLUTELY NO WARRANTY.
are welcome to redistribute it under certain conditions.
'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.

demo()' for some demos, 'help()' for on-line help, or
start()' for an HTML browser interface to help.
q()' to quit R.

g: unable to access index for repository http://software.rc.fas.harvar$
warning message:
In getDependencies(pkgs,
  packages 'TinnR', 'svs
Error in library(TinnR)
>
>
> 3 + 5
[1] 8
>
> |
```

Annotations with arrows point to the following components:

- R GUI** (i.e. the software/program): Points to the main RGui window.
- Command Prompt >** (says R is ready for you to submit a line of code): Points to the prompt character at the end of the last line of input.
- Code** (hit Enter/Return to submit): Points to the line of code being entered.
- Output []** (R processed the code and return the requested output): Points to the output of the code execution.
- R Console** (where code is submitted): Points to the console window.

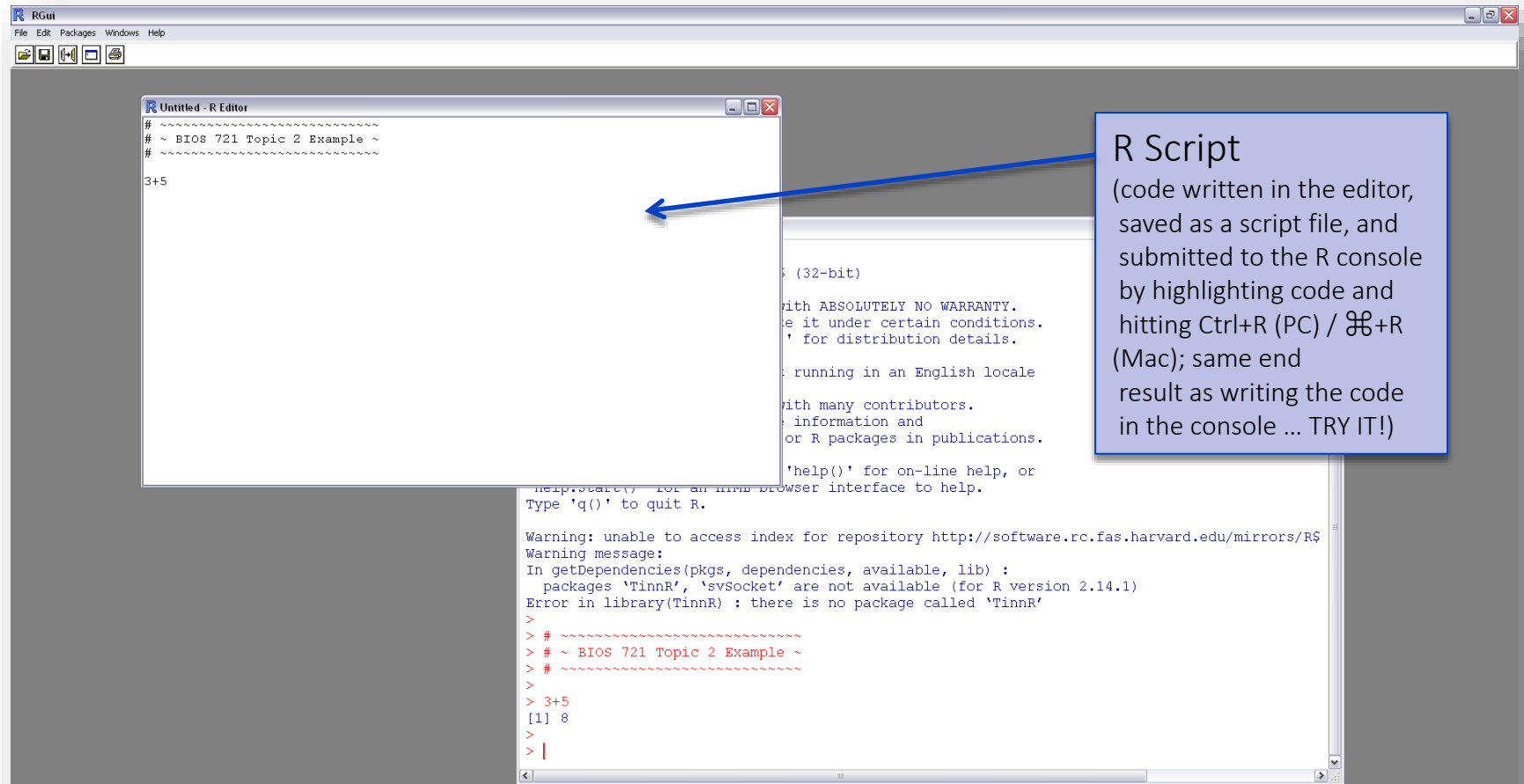
Script Files

11

- In the previous example, the “code” was written and submitted in the same place – the R console.
 - ▣ In R you do have the option to type code directly into console and submit, but this would become cumbersome if you had A LOT of code to submit and it took A LOT of time to develop (which is typically the case).
 - ▣ Instead if code is written and developed using a script file, you can easily save an editable and re-runnable version of the code for future use and development.
- For these reasons, using SCRIPT FILES is highly recommended (and somewhat necessary).
 - ▣ Scripts contain all of the code you have written in a way that you easily re-submit the code to processor as well make changes to the existing code and add new code to the same file.

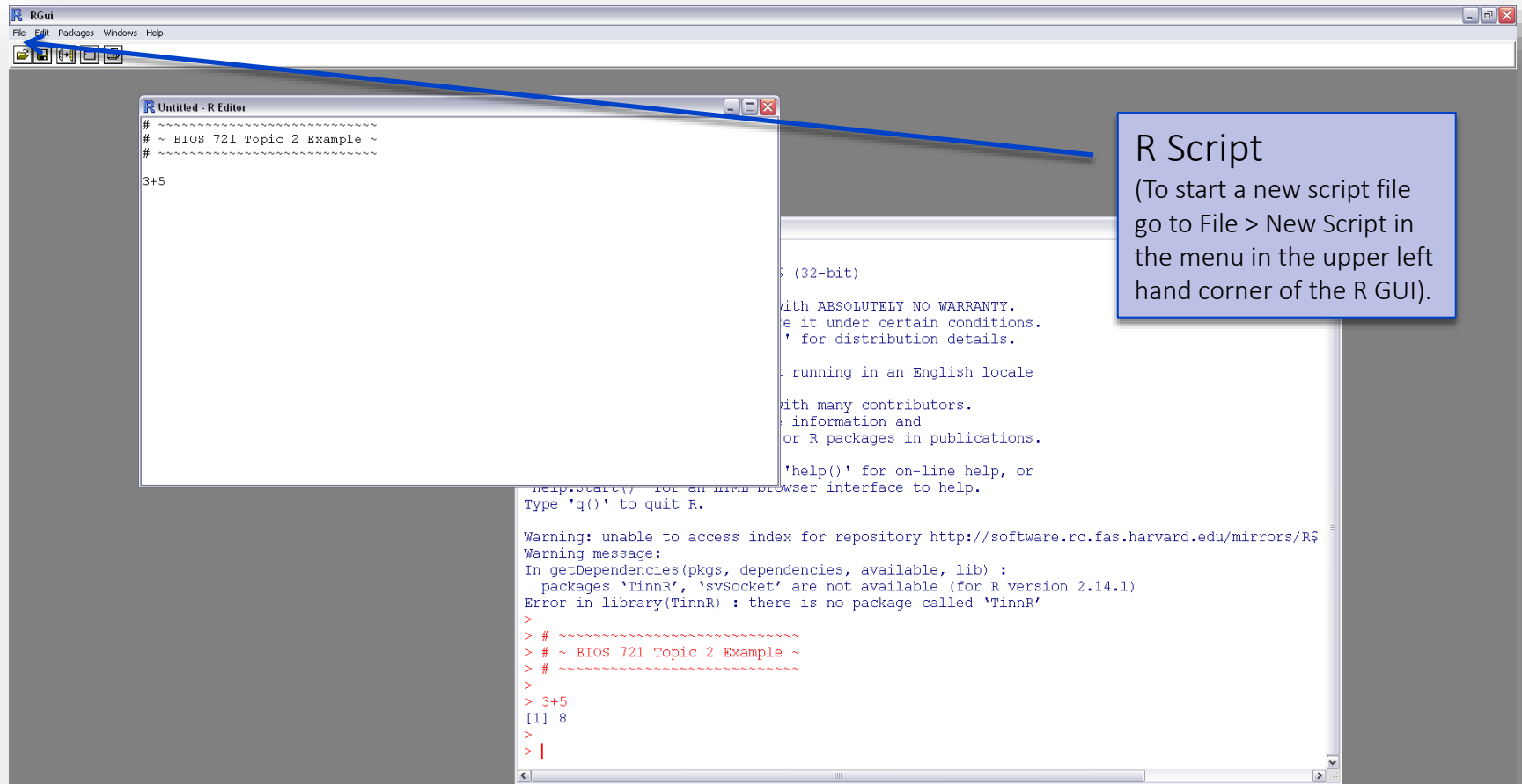
Script Files – Use Them!

12



Script Files – Use Them!

13



R Script

(To start a new script file go to File > New Script in the menu in the upper left hand corner of the R GUI).

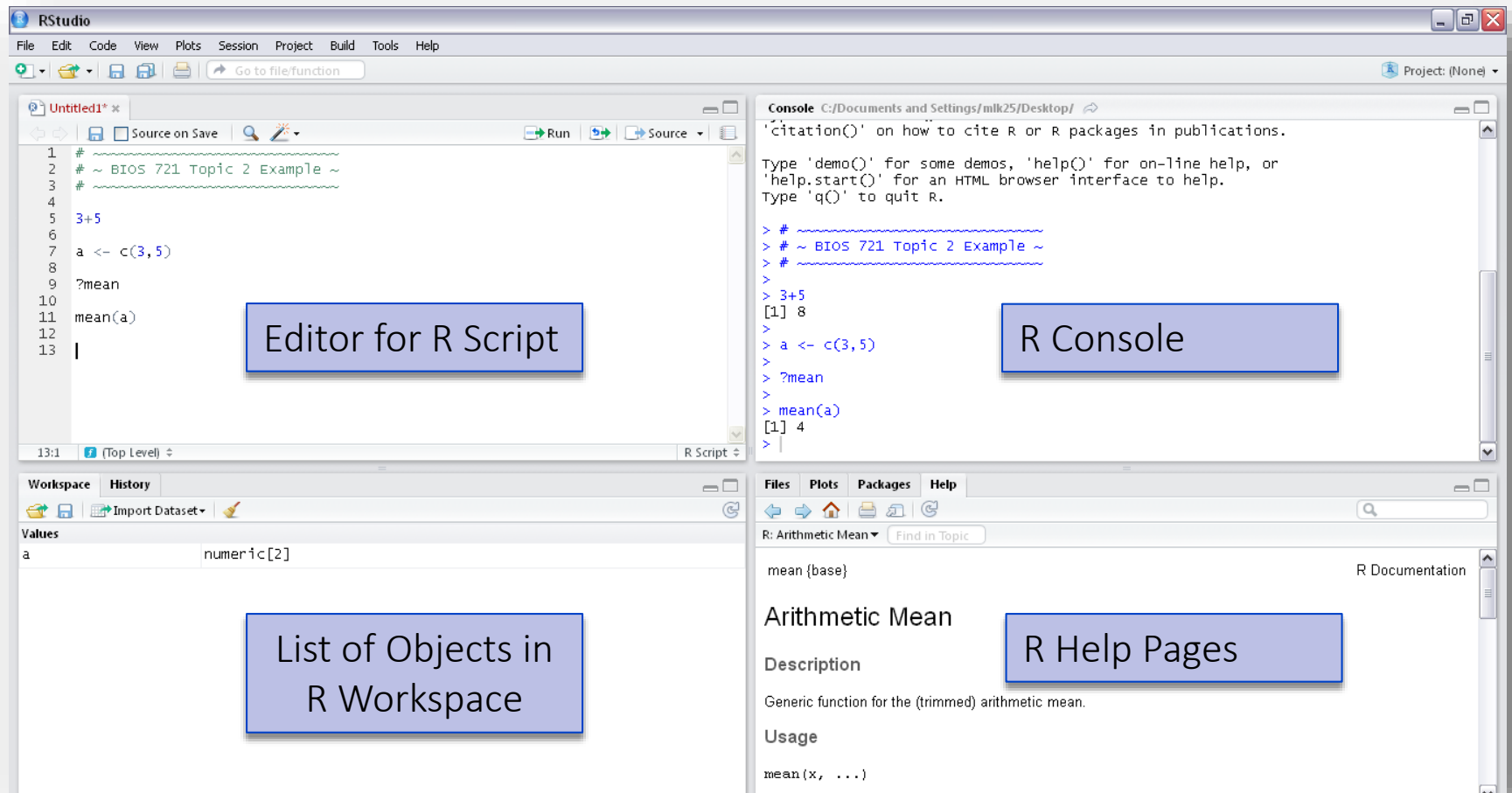
R and R Studio

14

- Where to code in R and how to submit it?
 - ▣ Can do it all within the RGui
 - File > New Script for an editor
 - Highlight code and hit Ctrl+R/⌘+R to submit code to R console
 - ▣ But I prefer an IDE called R Studio
 - It puts all of the moving parts into a single window (yeah!)
 - Its editor is color coded and has informative highlighting
 - What I will use in this course
 - But you may use whatever you prefer!
 - ▣ In all editors, scripts saved as .R files!

Moving Parts in R Studio

15



Setting Up R Studio

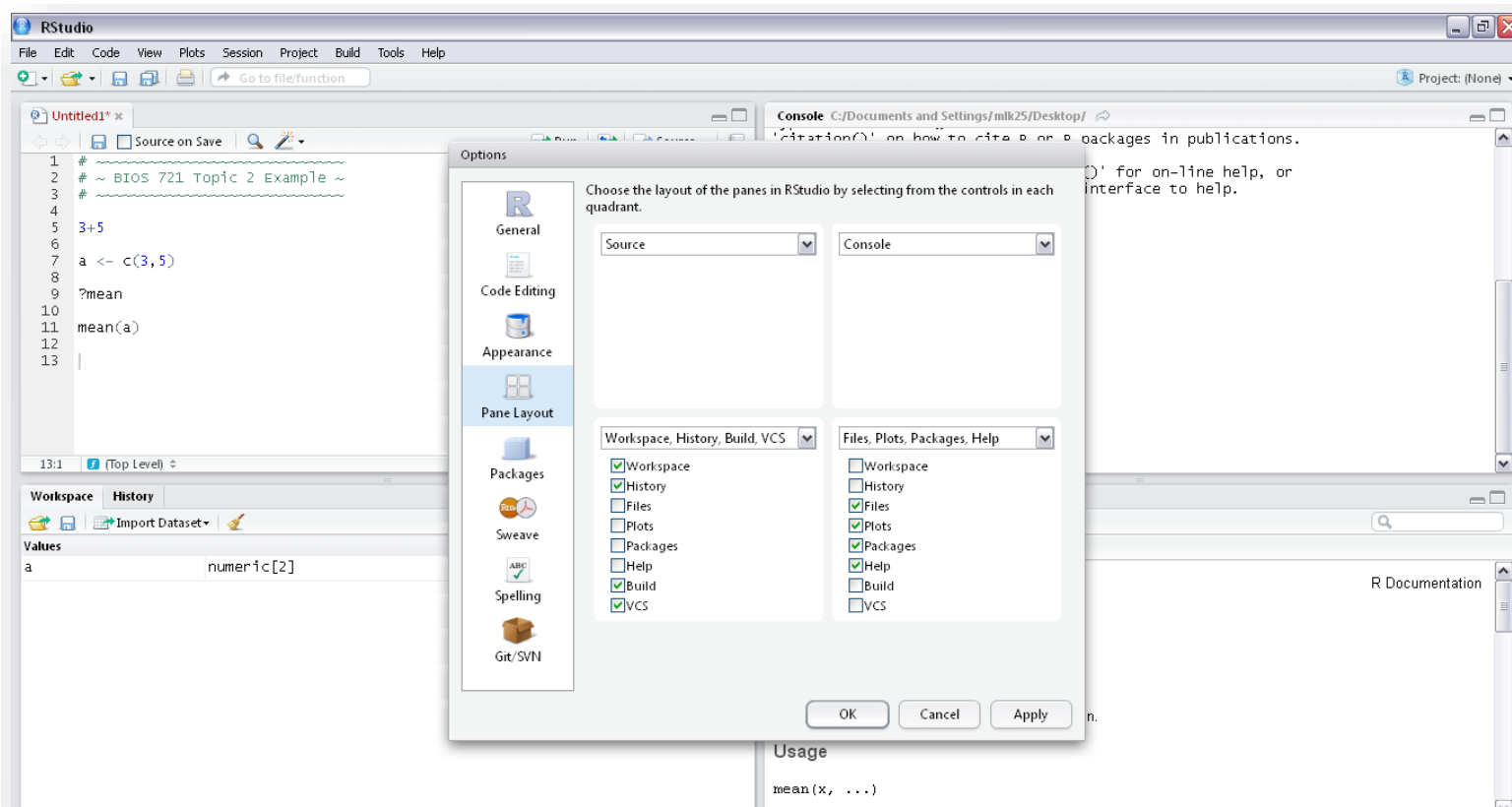
16

- Note: You MUST have R installed before using R Studio.
- After installing R Studio, there are a few options you should set to make working a little easier:
 - ▣ You may want to re-arrange the window layout (the layout on the previous screen is NOT the default, but is what I prefer).
 - ▣ You should set the default working directory (where input will be read from and where output will go by default – I like to use my desktop – you can always change this later)

Setting Up R Studio

17

- Re-arranging the window layout:
 - ▣ Tools > Global Options > Pane Layout > Apply/Ok



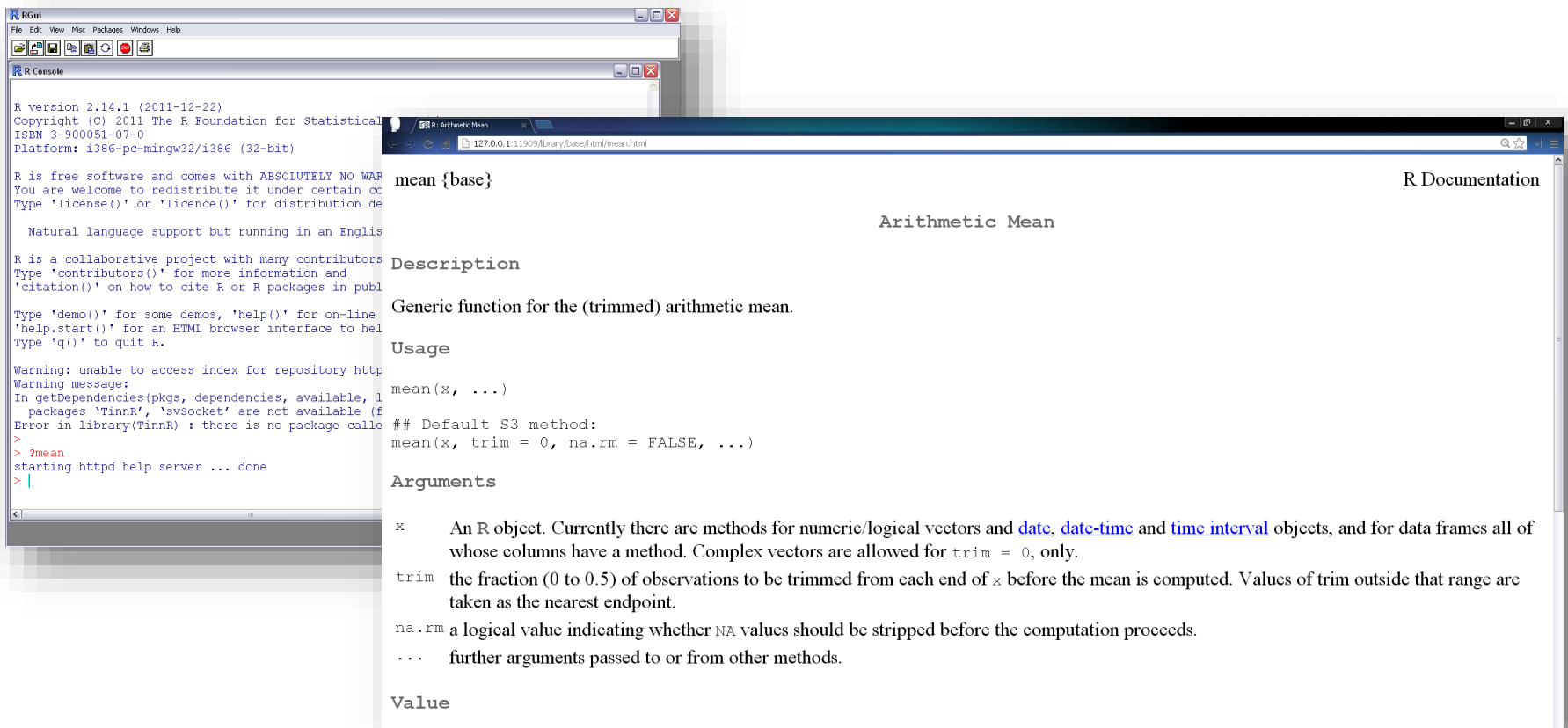
18

-
- The screenshot shows the RStudio Options dialog box, General tab. The 'Default working directory (when not in a project):' field is circled in blue and contains the path 'C:/Documents and Settings/mlk25/Desktop'. The 'R version:' field contains '[Default] C:\Program Files\R\R-2.14.1'. The 'Restore most recently opened project at startup' checkbox is checked. The 'Restore .RData into workspace at startup' checkbox is checked. The 'Save workspace to .RData on exit:' dropdown is set to 'Never'. The 'Always save history (even when not saving .RData)' checkbox is checked. The 'Remove duplicate entries in history' checkbox is checked. The 'Default text encoding:' field contains '[Ask]'. The 'Workspace' tab is selected in the bottom left pane, showing the variable 'a' with value 'numeric [2]'.

R Help Pages

19

- By typing `?name_of_function` in the R console, an R help page will be brought up in your internet browser.



The screenshot shows the R GUI with the R Console and a web browser displaying the R help page for the `mean` function.

R Console:

```
R version 2.14.1 (2011-12-22)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-pc-mingw32/i386 (32-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.

Warning: unable to access index for repository http://statlib.org/packages
Warning message:
In getDependencies(pkgs, dependencies, available, l) :
  packages 'TinnR', 'svSocket' are not available (if
Error in library(TinnR) : there is no package called 'TinnR'
> ?mean
starting http help server ... done
> |
```

Browser (R Documentation):

mean {base}

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)
```

Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)

Arguments

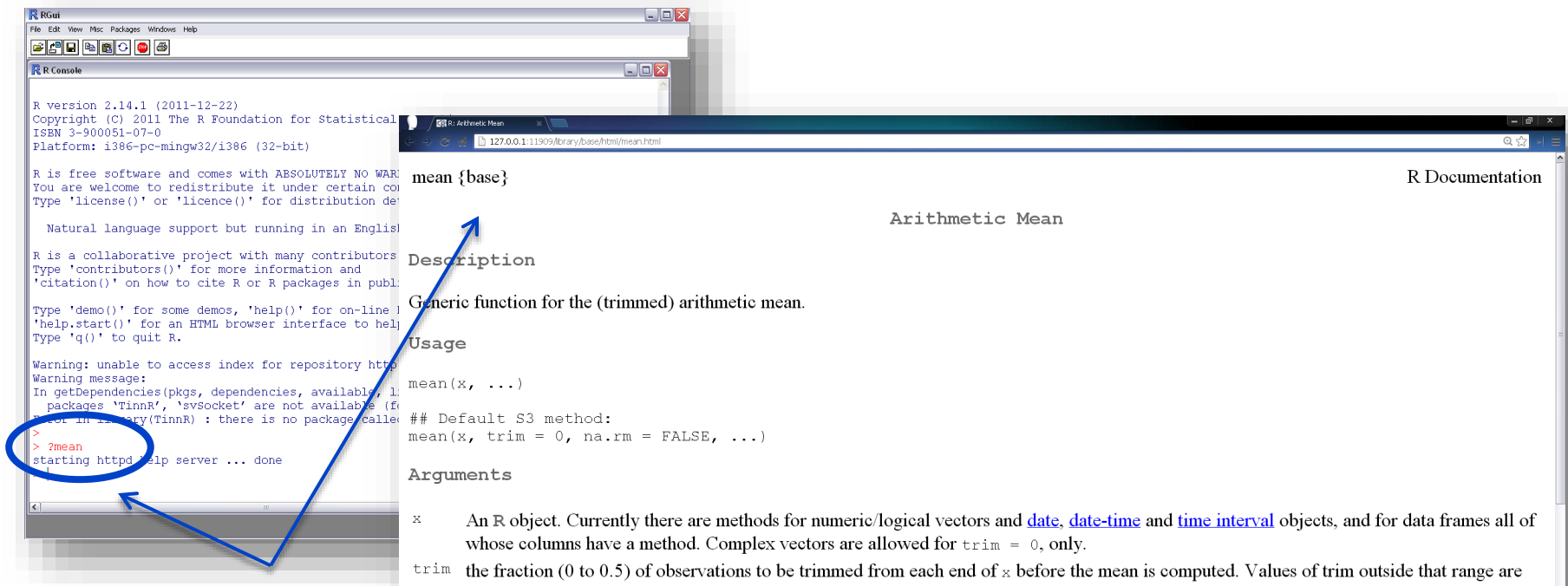
- `x` An R object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects, and for data frames all of whose columns have a method. Complex vectors are allowed for `trim = 0`, only.
- `trim` the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of `trim` outside that range are taken as the nearest endpoint.
- `na.rm` a logical value indicating whether NA values should be stripped before the computation proceeds.
- `...` further arguments passed to or from other methods.

Value

R Help Pages

20

- By typing `?name_of_function` in the R console, an R help page will be brought up in your internet browser. (TRY IT!)



Typing `?mean` into the R console brings up the R help page for the `mean()` function.

R Help Pages

21

- All R Help Pages have the same basic components:

The screenshot shows the R documentation page for the `mean` function. The title bar indicates the browser is viewing `R: Arithmetic Mean` at `127.0.0.1:11909/library/base/html/mean.html`. The page title is `mean {base}`, which is highlighted with a blue box and an arrow pointing to the `Description` section. The `Description` section contains the text: "Generic function for the (trimmed)". The `Usage` section shows the function signature: `mean(x, ...)` and the default S3 method: `## Default S3 method: mean(x, trim = 0, na.rm = FALSE)`. The `Arguments` section lists: `x` as "An R object. Currently the object must be a numeric vector, a matrix whose columns have a mean, or a data frame whose columns have a mean", `trim` as "the fraction (0 to 0.5) of the observations at each end of the distribution to be discarded", and `na.rm` as "a logical value indicating whether NA values should be stripped before the computation proceeds". The `Value` section is partially visible. A blue box with a blue border contains the following text: (1) Function Name and the {Package} it belongs to. - If you don't have the package installed and loaded you will not be able to use the function. - If the package listed is {Base} then the function is part of the base R installation and you do NOT have to install/load any extra packages to use the function during your session.

mean {base}

R Documentation

Description

Generic function for the (trimmed)

Usage

```
mean(x, ...)
```

Default S3 method:
mean(x, trim = 0, na.rm = FALSE)

Arguments

`x` An R object. Currently the object must be a numeric vector, a matrix whose columns have a mean, or a data frame whose columns have a mean.

`trim` the fraction (0 to 0.5) of the observations at each end of the distribution to be discarded. Values of trim outside that range are computed. Values of trim outside that range are

`na.rm` a logical value indicating whether NA values should be stripped before the computation proceeds.

`...` further arguments passed to or from other methods.

Value

(1) Function Name and the {Package} it belongs to.

- If you don't have the package installed and loaded you will not be able to use the function.
- If the package listed is {Base} then the function is part of the base R installation and you do NOT have to install/load any extra packages to use the function during your session.

R Help Pages

22

- All R Help Pages have the same basic components:

The screenshot shows the R help page for the `mean` function. The page is titled "mean {base}" and "Arithmetic Mean". It includes a "Description" section, a "Usage" section, an "Arguments" section, and a "Value" section. A blue box highlights the "Description" section, and a blue arrow points from a callout box to it. The callout box contains the text "(2) Description - Brief description of what the function does".

mean {base}

R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)
```

```
## Default S3 method:  
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x` An R object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects, and for data frames all of whose columns have a method. Complex vectors are allowed for `trim = 0`, only.

`trim` the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of `trim` outside that range are taken as the nearest endpoint.

`na.rm` a logical value indicating whether NA values should be stripped before the computation proceeds.

`...` further arguments passed to or from other methods.

Value

(2) Description
- Brief description of what the function does

R Help Pages

23

- All R Help Pages have the same basic components:

mean {base}

R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)
```

```
## Default S3 method:  
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

x An R object. Currently there are methods for numeric/logical vectors whose columns have a method. Complex vectors are allowed for `trim`.

trim the fraction (0 to 0.5) of observations to be trimmed from each end of the data. `0` indicates that no trimming is done. `0.5` indicates that all data are trimmed. `NA` is taken as the nearest endpoint.

na.rm a logical value indicating whether `NA` values should be stripped before the computation proceeds.

... further arguments passed to or from other methods.

Value

(3) Usage

- Example call of the function (syntax)
- Lists the required inputs (here, `x`)
- ... indicates that there are other optional inputs that have been set to DEFAULT values.

R Help Pages

24

- All R Help Pages have the same basic components:

mean {base}

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)
```

```
## Default S3 method:  
mean(x, trim = 0, na.rm = FALSE, ...)
```

(4) Arguments
- Detailed description of all possible inputs to the function

Arguments

x An R object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects, and for data frames all of whose columns have a method. Complex vectors are allowed for `trim = 0`, only.

trim the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of `trim` outside that range are taken as the nearest endpoint.

na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.

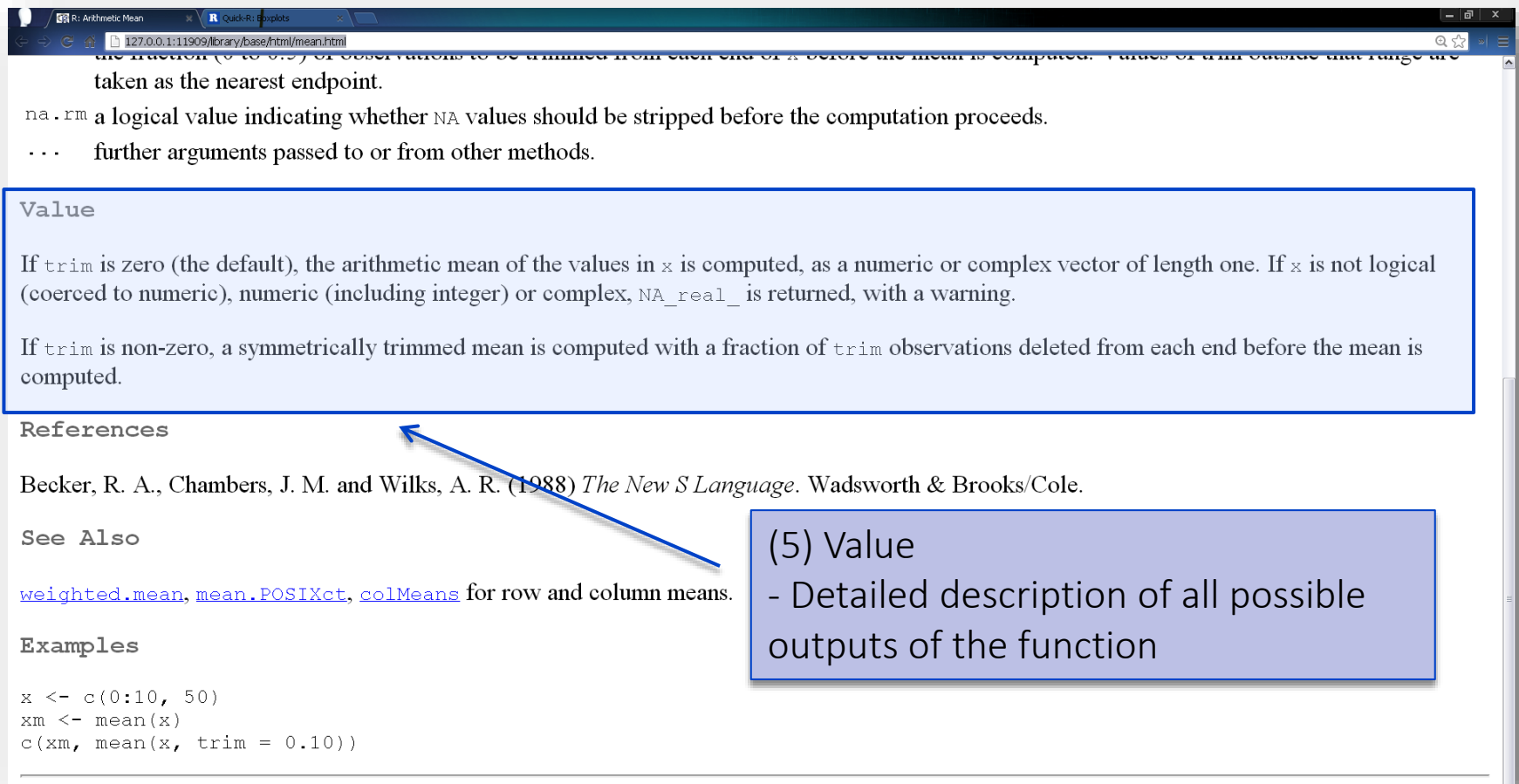
... further arguments passed to or from other methods.

Value

R Help Pages

25

- All R Help Pages have the same basic components:



the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of `trim` outside that range are taken as the nearest endpoint.

`na.rm` a logical value indicating whether NA values should be stripped before the computation proceeds.

... further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

[weighted.mean](#), [mean.POSIXct](#), [colMeans](#) for row and column means.

Examples

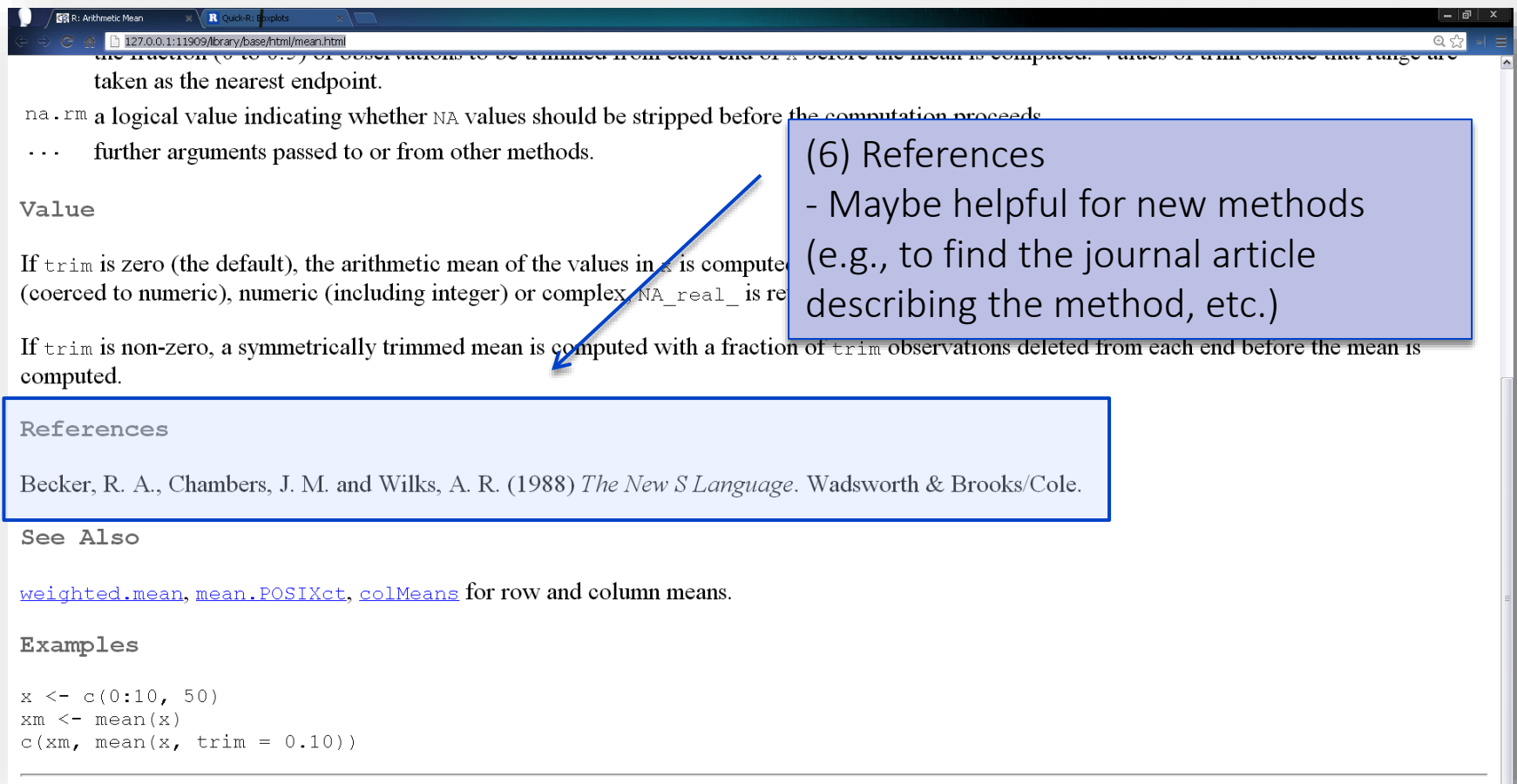
```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

(5) Value
- Detailed description of all possible outputs of the function

R Help Pages

26

- All R Help Pages have the same basic components:



the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of `trim` outside that range are taken as the nearest endpoint.

`na.rm` a logical value indicating whether NA values should be stripped before the computation proceeds

... further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed (coerced to numeric), numeric (including integer) or complex. `NA_real_` is returned if `x` contains only NA values.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

[weighted.mean](#), [mean.POSIXct](#), [colMeans](#) for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

(6) References

- Maybe helpful for new methods (e.g., to find the journal article describing the method, etc.)

R Help Pages

27

- All R Help Pages have the same basic components:

the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of `trim` outside that range are taken as the nearest endpoint.

`na.rm` a logical value indicating whether NA values should be stripped before the computation proceeds.

... further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` of the data removed from each end of the data and the mean computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

[weighted.mean](#), [mean.POSIXct](#), [colMeans](#) for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

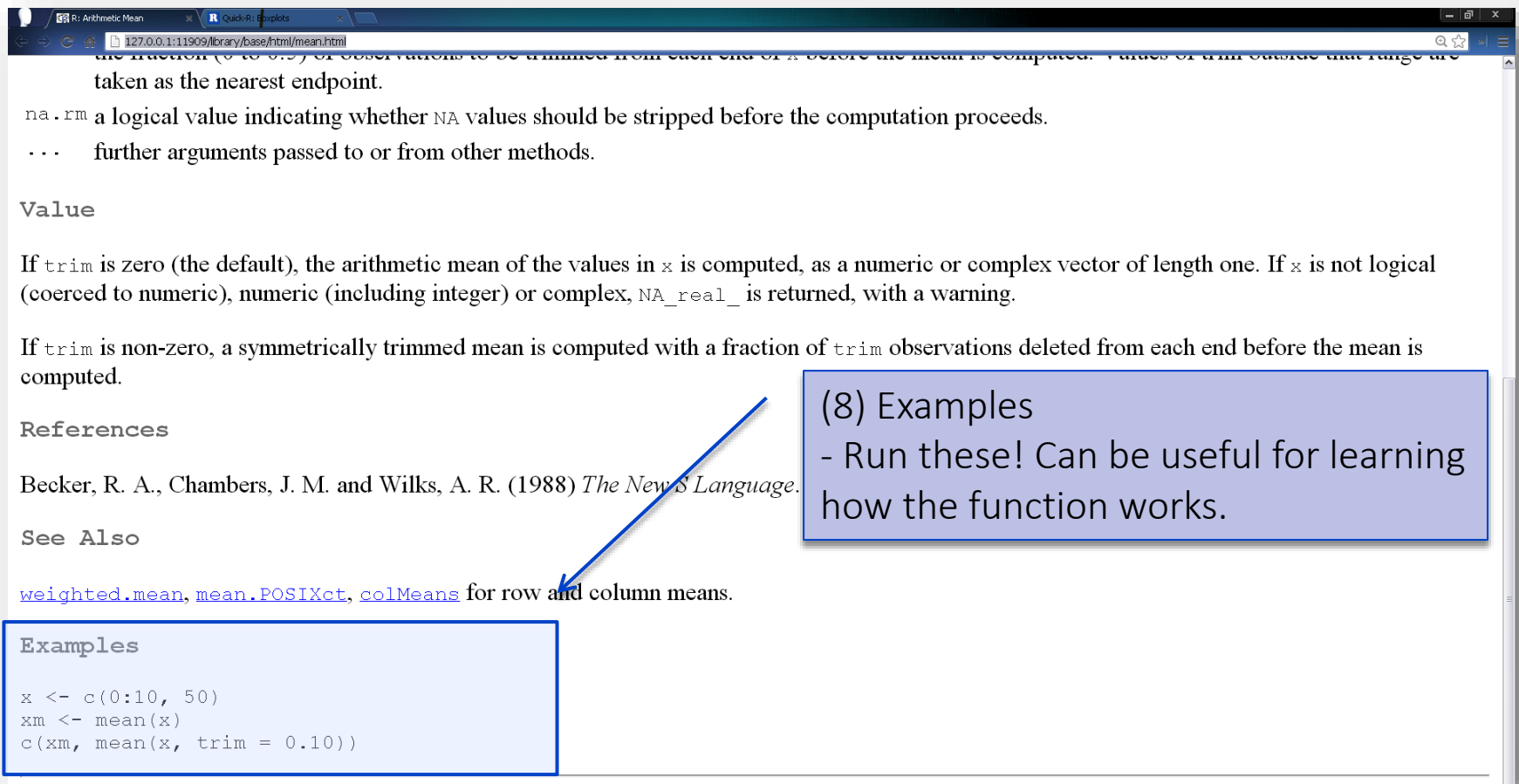
(7) See Also

- List of other R functions that may be helpful or are used by the current function

R Help Pages

28

- All R Help Pages have the same basic components:



the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.

`na.rm` a logical value indicating whether NA values should be stripped before the computation proceeds.

... further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*.

See Also

[weighted.mean](#), [mean.POSIXct](#), [colMeans](#) for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

(8) Examples
- Run these! Can be useful for learning how the function works.

R Help Pages

29

- All R Help Pages have the same basic components
- But not all R Help Pages are created equally
 - ▣ Some have way better documentation/examples than others
 - ▣ Google is your friend – chances are someone else has already done what you are trying to do!
- Another good R resource is the Quick R website
 - ▣ <http://www.statmethods.net/>
 - ▣ Maintained by the author of R in Action
 - ▣ Help pages are very complete and often have several examples for each function/topic
 - ▣ Search “boxplot” and check out what I mean!

R Objects

30

- R is an object oriented programming language
 - ▣ Makes it very flexible and a powerful programming platform
 - ▣ Objects can be very complex (e.g. output from a regression analysis) or very simple (e.g. a matrix).

- In their simplest form, you can think of objects as
 - ▣ Container + element
 - ▣ Example |
 - Container = Matrix
 - Element = Numbers
 - Object = Numeric Matrix

R Objects

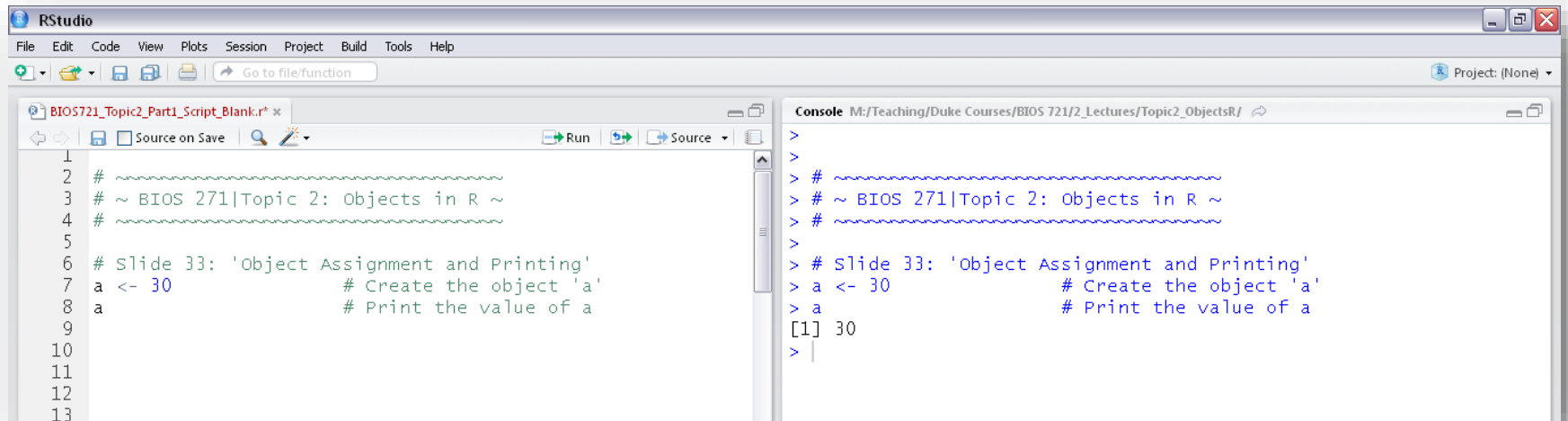
31

- Data Objects / Structures:
 - Scalars
 - Vectors
 - Matrices
 - Arrays
 - Data Frames
 - Lists
- To create an object, use the assignment command:
 - `<-` or `=`
- To print (i.e. display the value of) the object, enter the object's name into the console at hit enter.

R Objects

32

- In the example below,
 - ▣ The first line of code creates the object 'a' and assigns it to the scalar value 30.
 - ▣ The second line of code prints the value of 'a'.



The screenshot shows the RStudio interface. The script editor on the left contains the following code:

```
1  
2 # ~~~~~  
3 # ~ BIOS 271|Topic 2: Objects in R ~  
4 # ~~~~~  
5  
6 # Slide 33: 'Object Assignment and Printing'  
7 a <- 30           # Create the object 'a'  
8 a                # Print the value of a  
9  
10  
11  
12  
13
```

The console on the right shows the output of the code:

```
>  
>  
> # ~~~~~  
> # ~ BIOS 271|Topic 2: Objects in R ~  
> # ~~~~~  
>  
> # Slide 33: 'Object Assignment and Printing'  
> a <- 30           # Create the object 'a'  
> a                # Print the value of a  
[1] 30  
> |
```


R Objects

33

□ Scalars (a single element)

□ <code>s1 <- 2</code>	→	<code>[1] 2</code>	Numeric
□ <code>s2 <- "Hi!"</code>	→	<code>[1] "Hi!"</code>	Character
□ <code>s3 <- TRUE</code>	→	<code>[1] TRUE</code>	Logical

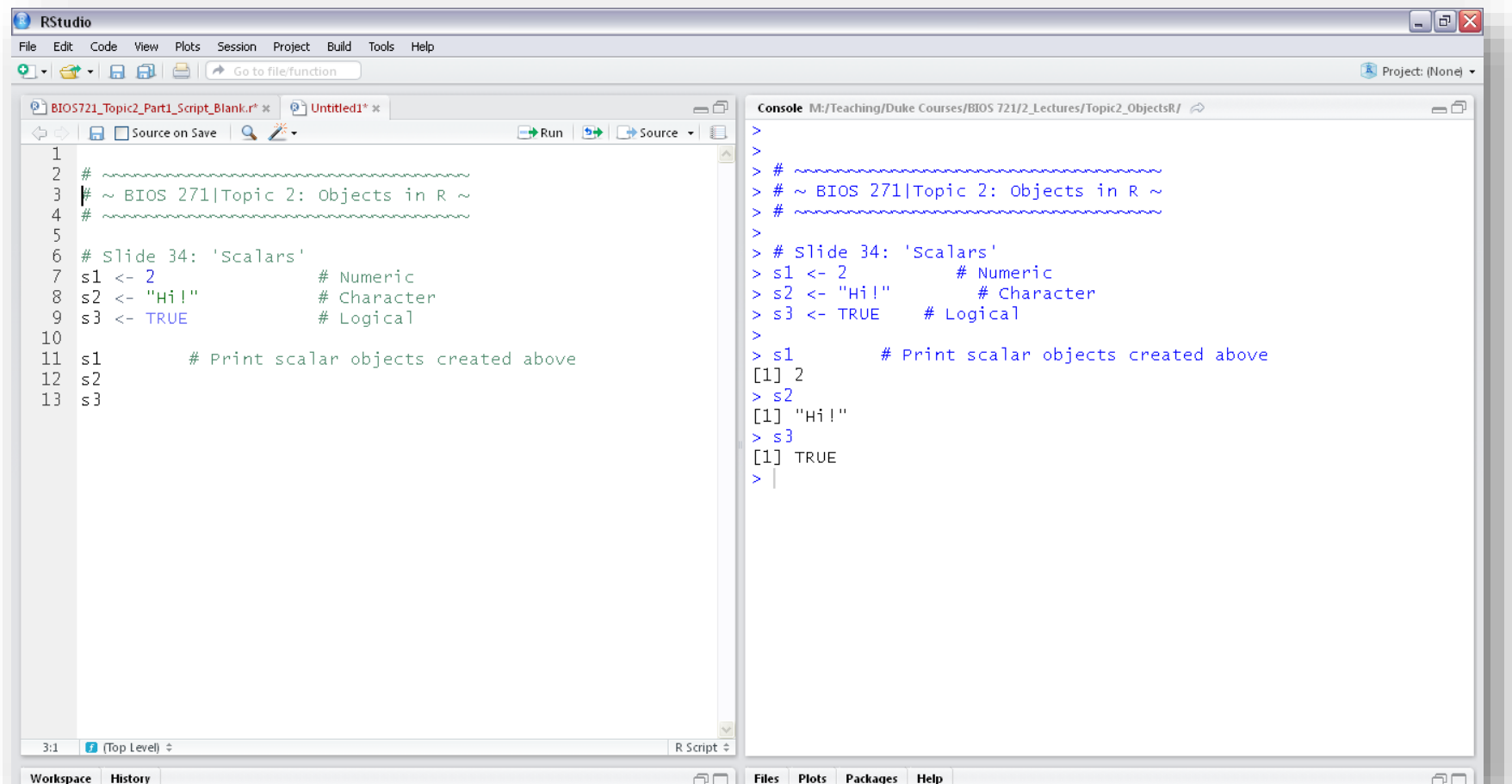
Mode
a.k.a
Type of
Data

□ Special Cases

- `NA` = missing value
- `NaN` = Not a number
- `-Inf/Inf` = Infinity (or at least close to it)

R Objects

34



The screenshot shows the RStudio interface with a script editor on the left and a console on the right. The script editor contains R code for creating scalar objects (numeric, character, and logical) and printing them. The console shows the output of the code, confirming the creation of the objects s1, s2, and s3.

```
1  
2 # ~~~~~  
3 # ~ BIOS 271|Topic 2: Objects in R ~  
4 # ~~~~~  
5  
6 # Slide 34: 'Scalars'  
7 s1 <- 2 # Numeric  
8 s2 <- "Hi!" # Character  
9 s3 <- TRUE # Logical  
10  
11 s1 # Print scalar objects created above  
12 s2  
13 s3
```

```
>  
>  
> # ~~~~~  
> # ~ BIOS 271|Topic 2: Objects in R ~  
> # ~~~~~  
>  
> # Slide 34: 'Scalars'  
> s1 <- 2 # Numeric  
> s2 <- "Hi!" # Character  
> s3 <- TRUE # Logical  
>  
> s1 # Print scalar objects created above  
[1] 2  
> s2  
[1] "Hi!"  
> s3  
[1] TRUE  
> |
```

R Objects

35

- A Note on Logical Objects:
 - ▣ TRUE and FALSE are reserved words (i.e. users cannot assign values to these words)
 - ▣ Although the values are “character” visually, they can be manipulated numerically (i.e. R easily allows TRUE and FALSE to be converted to 1 and 0, respectively).
- ▣ T and F are shortcuts that can be used in place of TRUE and FALSE, respectively.
 - But Be Careful! T and F are not reserved, so users can over write these values, which can lead to errors (see next slide for example).

R Objects

36

RStudio

File Edit Code View Plots Session Project Build Tools Help

Go to file/function

Project: (None)

BIOS721_Topic2_Part1_Script_Blank.r* x Untitled1* x

Source on Save Run Source

```
1
2 # ~~~~~
3 # ~ BIOS 271|Topic 2: Objects in R ~
4 # ~~~~~
5
6 # Slide 36: 'What is a logical object?'
7
8 logical <- FALSE
9 not.logical <- "FALSE"
10
11 logical
12 not.logical
13
14 logical*1
15 not.logical*1
16
17 logical2 <- F
18 logical2
19
20 F <- 2
21 logical2 <- F
22 logical2
```

Console M:/Teaching/Duke Courses/BIOS 721/2_Lectures/Topic2_ObjectsR/

```
>
> # ~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~
>
> # Slide 36: 'What is a logical object?'
>
> logical <- FALSE
> not.logical <- "FALSE"
>
> logical
[1] FALSE
> not.logical
[1] "FALSE"
>
> logical*1
[1] 0
> not.logical*1
Error in not.logical * 1 : non-numeric argument to binary operator
>
> logical2 <- F
> logical2
[1] FALSE
>
> F <- 2
> logical2 <- F
> logical2
[1] 2
> |
```

TRUE/FALSE (i.e. logicals) are special reserved words in R that convey certain properties (i.e. this is NOT true for character objects).

3:1 (Top Level) R Script

Workspace History Files Plots Packages Help

R Objects

37

The screenshot shows the RStudio interface with a script editor on the left and a console on the right. The script editor contains R code for creating and manipulating logical objects. The console shows the execution of this code, including an error message for an invalid operation. A blue box with a bracket points from the console output to a text box.

Script Editor Code:

```
1  
2 # ~~~~~  
3 # ~ BIOS 271|Topic 2: Objects in R ~  
4 # ~~~~~  
5  
6 # Slide 36: 'What is a logical object?'  
7  
8 logical <- FALSE  
9 not.logical <- "FALSE"  
10  
11 logical  
12 not.logical  
13  
14 logical*1  
15 not.logical*1  
16  
17 logical2 <-  
18 logical2  
19  
20 F <- 2  
21 logical2 <-  
22 logical2
```

Console Output:

```
>  
> # ~~~~~  
> # ~ BIOS 271|Topic 2: Objects in R ~  
> # ~~~~~  
>  
> # Slide 36: 'What is a logical object?'  
>  
> logical <- FALSE  
> not.logical <- "FALSE"  
>  
> logical  
[1] FALSE  
> not.logical  
[1] "FALSE"  
>  
> logical*1  
[1] 0  
> not.logical*1  
Error in not.logical * 1 : non-numeric argument to binary operator  
>  
> logical2 <- F           # Be careful when using logical  
> logical2                 # shortcuts (T and F)!  
[1] FALSE  
>  
> F <- 2  
> logical2 <- F  
> logical2  
[1] 2  
> |
```

Note: Logicals can be manipulated mathematically and can easily be converted to numeric values, usually a 0/1 binary indicator (NOT true for character objects).

R Objects

38

The screenshot shows the RStudio interface with a script editor on the left and a console on the right. The script editor contains R code for creating logical objects. The console shows the execution of this code, including an error message for a non-numeric argument to a binary operator. A blue callout box with a bracket points to the error message and contains the following text:

T/F are shortcuts for the typical logicals. But be careful when using T/F as logicals because these variables are NOT reserved (i.e. they can be over written by the user).

```
1  
2 # ~~~~~  
3 # ~ BIOS 271|Topic 2: Objects in R ~  
4 # ~~~~~  
5  
6 # Slide 36: 'What is a logical object?'  
7  
8 logical <- FALSE  
9 not.logical <- "FALSE"  
10  
11 logical  
12 not.logical  
13  
14 logical*1  
15 not.logical*1  
16  
17 logical2 <-  
18 logical2  
19  
20 F <- 2  
21 logical2 <-  
22 logical2
```

```
>  
> # ~~~~~  
> # ~ BIOS 271|Topic 2: Objects in R ~  
> # ~~~~~  
> # Slide 36: 'What is a logical object?'  
>  
> logical <- FALSE  
> not.logical <- "FALSE"  
>  
> logical  
[1] FALSE  
> not.logical  
[1] "FALSE"  
>  
> logical*1  
[1] 0  
> not.logical*1  
Error in not.logical * 1 : non-numeric argument to binary operator  
>  
> logical2 <- F           # Be careful when using logical  
> logical2                # shortcuts (T and F)!  
[1] FALSE  
>  
> F <- 2  
> logical2 <- F  
> logical2  
[1] 2  
>
```

R Objects

39

- Vectors (stacks of scalars)

- ▣ Create using the R 'combine' function: `c (, ...)`

- ▣ Hard Code

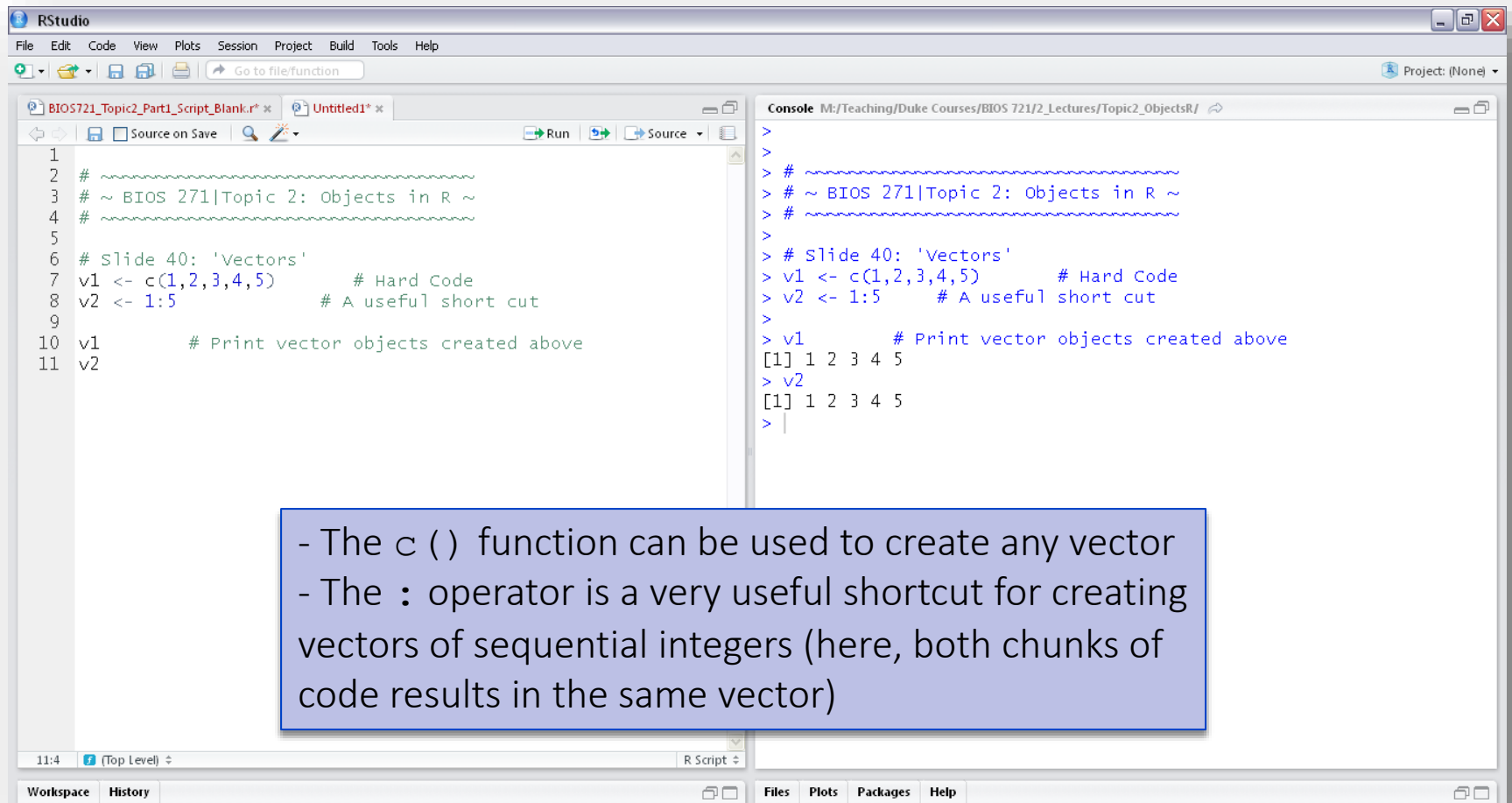
- `v1 <- c (1, 2, 3, 4, 5)` \rightarrow `[1] 1 2 3 4 5`

- ▣ But there are lots of shortcuts ... (more in the PPs)

- `v2 <- 1:5` \rightarrow `[1] 1 2 3 4 5`

R Objects

40



The screenshot shows the RStudio interface. The editor window on the left contains the following R code:

```
1  
2 # ~~~~~  
3 # ~ BIOS 271|Topic 2: Objects in R ~  
4 # ~~~~~  
5  
6 # Slide 40: 'Vectors'  
7 v1 <- c(1,2,3,4,5)      # Hard Code  
8 v2 <- 1:5               # A useful short cut  
9  
10 v1      # Print vector objects created above  
11 v2
```

The console window on the right shows the output of the code:

```
>  
>  
> # ~~~~~  
> # ~ BIOS 271|Topic 2: Objects in R ~  
> # ~~~~~  
>  
> # Slide 40: 'Vectors'  
> v1 <- c(1,2,3,4,5)      # Hard Code  
> v2 <- 1:5               # A useful short cut  
>  
> v1      # Print vector objects created above  
[1] 1 2 3 4 5  
> v2  
[1] 1 2 3 4 5  
> |
```

A blue box with a black border is overlaid on the bottom right of the screenshot, containing the following text:

- The `c()` function can be used to create any vector
- The `:` operator is a very useful shortcut for creating vectors of sequential integers (here, both chunks of code results in the same vector)

R Objects

41

- Matrices (stacks of vectors)

- Create using the R function 'matrix':

```
matrix(data, nrow, ncol,  
        byrow=FALSE, dimnames=NULL)   OR  
matrix(data, nrow, ncol, ...)
```

- Examples

- M1 <- matrix(1:6, 3, 2)
 - M2 <- matrix(1:6, 3, 2, byrow=TRUE)
 - M3 <- matrix(1:6, 3, 2, byrow=TRUE,
 dimnames=list(c("R1", "R2", "R3"),
 c("C1", "C2")))

R Objects

42

- Matrices (stacks of vectors)
 - Create using the R function 'matrix':

```
matrix(data, nrow, ncol,  
        byrow=FALSE, dimnames=NULL)   OR  
matrix(data, nrow, ncol, ...)
```

Recall:


- Any arguments NOT assigned a value are required input for the function to execute (here data, nrow, and ncol)
- Any arguments assigned to a value are optional; if not given a value during the function call, they will be set to the default value (here byrow and dimnames)
- To learn about a function, check out the R help page

R Objects

43

- Matrices (stacks of vectors)
 - Create using the R function 'matrix':

```
matrix(data, nrow, ncol,  
        byrow=FALSE, dimnames=NULL) OR
```



data = any VECTOR of values (numeric, character, or logical)

- Examples


- M1 <- matrix(1:6, 3, 2)
- M2 <- matrix(1:6, 3, 2, byrow=TRUE)
- M3 <- matrix(1:6, 3, 2, byrow=TRUE,
 dimnames=list(c("R1", "R2", "R3"),
 c("C1", "C2")))

R Objects

44

- Matrices (stacks of vectors)
 - Create using the R function 'matrix':

```
matrix(data, nrow, ncol,  
       byrow=FALSE, dimnames=NULL) OR
```



nrow = number of rows in the matrix

- Examples

- M1 <- matrix(1:6, 3, 2)
- M2 <- matrix(1:6, 3, 2, byrow=TRUE)
- M3 <- matrix(1:6, 3, 2, byrow=TRUE,
 dimnames=list(c("R1", "R2", "R3"),
 c("C1", "C2")))

R Objects

45

- Matrices (stacks of vectors)
 - Create using the R function 'matrix':

```
matrix(data, nrow, ncol,  
       byrow=FALSE, dimnames=NULL) OR
```



ncol = number of columns in the matrix


- Examples

- M1 <- matrix(1:6, 3, 2)
- M2 <- matrix(1:6, 3, 2, byrow=TRUE)
- M3 <- matrix(1:6, 3, 2, byrow=TRUE,
 dimnames=list(c("R1", "R2", "R3"),
 c("C1", "C2")))

R Objects

46

- Matrices (stacks of vectors)
 - Create using the R function 'matrix':

```
matrix(data, nrow, ncol,  
 byrow=FALSE, dimnames=NULL) OR
```

byrow = logical indicator for whether the vector should fill the matrix down columns (byrow=FALSE) or across rows (byrow=TRUE)

- Examples

```
■ M1 <- matrix(1:6, 3, 2)  
■ M2 <- matrix(1:6, 3, 2, byrow=TRUE)  
■ M3 <- matrix(1:6, 3, 2, byrow=TRUE,  
               dimnames=list(c("R1", "R2", "R3"),  
                             c("C1", "C2")))
```

R Objects

47

- Matrices (stacks of vectors)
 - Create using the R function 'matrix':

```
matrix(data, nrow, ncol,  
byrow=FALSE, dimnames=NULL) OR
```

dimnames = a list of two vectors containing the row and column names; row names should be listed first

- Examples

- M1 <- matrix(1:6, 3, 2)
- M2 <- matrix(1:6, 3, 2, byrow=TRUE)
- M3 <- matrix(1:6, 3, 2, byrow=TRUE,
dimnames=list(c("R1", "R2", "R3"),
c("C1", "C2")))

R Objects

48

The screenshot shows the RStudio interface with a script editor on the left and a console on the right. The script editor contains R code for creating three matrices (M1, M2, M3) and printing them. The console shows the output of these commands. A blue box highlights the comparison of results when `byrow=FALSE` vs. `byrow=TRUE`.

```
# ~ BIOS 271|Topic 2: Objects in R ~  
# Slide 42: 'Matrices'  
M1 <- matrix(1:6,3,2)  
M2 <- matrix(1:6,3,2,byrow=TRUE)  
M3 <- matrix(1:6,3,2,byrow=TRUE,  
             dimnames=list(c("R1", "R2", "R3"),  
                           c("C1", "C2")))  
  
# Print matrix objects created above  
M1  
M2  
M3
```

Console output:

```
> # ~ BIOS 271|Topic 2: Objects in R ~  
> # Slide 42: 'Matrices'  
> M1 <- matrix(1:6,3,2)  
> M2 <- matrix(1:6,3,2,byrow=TRUE)  
> M3 <- matrix(1:6,3,2,byrow=TRUE,  
+             dimnames=list(c("R1", "R2", "R3"),  
+                           c("C1", "C2")))  
>  
> M1 # Print matrix  
[,1] [,2]  
[1,] 1 4  
[2,] 2 5  
[3,] 3 6  
> M2  
[,1] [,2]  
[1,] 1 2  
[2,] 3 4  
[3,] 5 6  
> M3  
  C1 C2  
R1 1 2  
R2 3 4  
R3 5 6  
>
```

Compare the results when `byrow=FALSE` vs. `byrow=TRUE`

R Objects

49

The screenshot shows the RStudio interface with a script editor on the left and a console on the right. The script editor contains R code for creating three matrices (M1, M2, M3) with dimension names. The console shows the output of the code, including the printed matrices and their dimensions.

Script Editor Code:

```
1 # ~~~~~  
2 # ~ BIOS 271|Topic 2: Objects in R ~  
3 # ~~~~~  
4 #  
5  
6 # Slide 42: 'Matrices'  
7 M1 <- matrix(1:6,3,2)  
8 M2 <- matrix(1:6,3,2,byrow=TRUE)  
9 M3 <- matrix(1:6,3,2,byrow=TRUE,  
10             dimnames=list(c("R1", "R2", "R3"),  
11                           c("C1", "C2")))  
12  
13  
14 M1      # Print matrix objects created above  
15 M2  
16 M3  
17  
18
```

Console Output:

```
> # ~~~~~  
> # ~ BIOS 271|Topic 2: Objects in R ~  
> # ~~~~~  
> # Slide 42: 'Matrices'  
> M1 <- matrix(1:6,3,2)  
> M2 <- matrix(1:6,3,2,byrow=TRUE)  
> M3 <- matrix(1:6,3,2,byrow=TRUE,  
+             dimnames=list(c("R1", "R2", "R3"),  
+                           c("C1", "C2")))  
>  
> M1      # Print matrix objects created above  
      [,1] [,2]  
[1,]    1    4  
[2,]    2    5  
[3,]    3    6  
> M2  
      [,1] [,2]  
[1,]    1    2  
[2,]    3    4  
[3,]    5    6  
> M3  
      C1 C2  
R1    1  2  
R2    3  4  
R3    5  6  
> |
```

Annotation: A blue box with a bracket points to the dimension names in the script editor code, stating: "When dimension names are specified, the row/column indices are replaced with the user specified row and column names."

R Objects

50

- Matrices (stacks of vectors)
 - Can also create matrices by combining vectors by rows or by columns (can be very useful in simulations)
 - Examples
 - `x <- rep(1, 5)`
 - `y <- rep(2, 5)`
 - `z <- rep(3, 5)`

 - `M.col <- cbind(x, y, z)`
 - `M.row <- rbind(x, y, z)`

R Objects

51

The `rep()` function will create a vector by replicating the scalar value (numeric, character, or logical) provided – another great shortcut!

```
1  
2 # ~~~~~  
3 # ~ BIOS 271|Topic 2: Objects in R ~  
4 # ~~~~~  
5  
6 # Slide 51: 'Matrices'  
7 x <- rep(1,5)  
8 y <- rep(2,5)  
9 z <- rep(3,5)  
10  
11 M.col <- cbind(x,y,z)  
12 M.row <- rbind(x,y,z)  
13  
14 M.col      # Print matrix objects created above  
15 M.row  
16 |  
17
```

```
> # ~~~~~  
> # ~ BIOS 271|Topic 2: Objects in R ~  
> # ~~~~~  
>  
> # Slide 51: 'Matrices'  
> x <- rep(1,5)  
> y <- rep(2,5)  
> z <- rep(3,5)  
>  
> M.col <- cbind(x,y,z)  
> M.row <- rbind(x,y,z)  
>  
> M.col      # Print matrix objects created above  
      x y z  
[1,] 1 2 3  
[2,] 1 2 3  
[3,] 1 2 3  
[4,] 1 2 3  
[5,] 1 2 3  
> M.row  
      [,1] [,2] [,3] [,4] [,5]  
x      1    1    1    1    1  
y      2    2    2    2    2  
z      3    3    3    3    3  
> |
```

R Objects

52

The screenshot shows the RStudio interface. The source editor on the left contains R code for creating vectors and matrices. The console on the right shows the execution of this code, including the output of the `cbind()` and `rbind()` functions. A blue box with a blue arrow points from the `cbind(x,y,z)` line in the code to the console output.

Source Editor Code:

```
1  
2 # ~~~~~  
3 # ~ BIOS 271|Topic 2: Objects in R ~  
4 # ~~~~~  
5  
6 # Slide 51: 'Matrices'  
7 x <- rep(1,5)  
8 y <- rep(2,5)  
9 z <- rep(3,5)  
10  
11 M.col <- cbind(x,y,z)  
12 M.row <- rbind(x,y,z)  
13  
14 M.col      # Print matrix objects created above  
15 M.row  
16 |  
17
```

Console Output:

```
>  
> # ~~~~~  
> # ~ BIOS 271|Topic 2: Objects in R ~  
> # ~~~~~  
>  
> # Slide 51: 'Matrices'  
> x <- rep(1,5)  
> y <- rep(2,5)  
> z <- rep(3,5)  
>  
> M.col <- cbind(x,y,z)  
> M.row <- rbind(x,y,z)  
>  
> M.col      # Print matrix objects created above  
      x y z  
[1,] 1 2 3  
[2,] 1 2 3  
[3,] 1 2 3  
[4,] 1 2 3  
[5,] 1 2 3  
> M.row  
      [,1] [,2] [,3] [,4] [,5]  
x      1   1   1   1   1  
y      2   2   2   2   2  
z      3   3   3   3   3  
> |
```

Annotation:

The `cbind()` function will combine vectors as the columns of a matrix

R Objects

53

The screenshot shows the RStudio interface with a script editor on the left and a console on the right. The script editor contains R code for creating and printing matrices. The console shows the output of the code. A blue box with a double-lined arrow points from the `rbind()` function call in the script to the console output.

Script Editor Code:

```
1  
2 # ~~~~~  
3 # ~ BIOS 271|Topic 2: Objects in R ~  
4 # ~~~~~  
5  
6 # Slide 51: 'Matrices'  
7 x <- rep(1,5)  
8 y <- rep(2,5)  
9 z <- rep(3,5)  
10  
11 M.col <- cbind(x,y,z)  
12 M.row <- rbind(x,y,z)  
13  
14 M.col      # Print matrix objects created above  
15 M.row  
16 |  
17
```

Console Output:

```
> # ~~~~~  
> # ~ BIOS 271|Topic 2: Objects in R ~  
> # ~~~~~  
>  
> # Slide 51: 'Matrices'  
> x <- rep(1,5)  
> y <- rep(2,5)  
> z <- rep(3,5)  
>  
> M.col <- cbind(x,y,z)  
> M.row <- rbind(x,y,z)  
>  
> M.col      # Print matrix objects created above  
      x y z  
[1,] 1 2 3  
[2,] 1 2 3  
[3,] 1 2 3  
[4,] 1 2 3  
[5,] 1 2 3  
> M.row  
      [,1] [,2] [,3] [,4] [,5]  
x      1   1   1   1   1  
y      2   2   2   2   2  
z      3   3   3   3   3  
> |
```

Annotation:

The `rbind()` function will combine vectors as the rows of a matrix

R Objects

54

- Arrays (stacks of matrices)
 - Create using the R function 'array':
`array(data, dim, ...)`
 - Examples
 - `A1 <- array(1:16, dim=c(4, 2, 2))`
 - `A2 <- array(c(rep(1, 4), rep(2, 4), rep(3, 4)),
dim=c(2, 2, 3),
dimnames=list(c("R1", "R2"),
c("C1", "C2"),
c("ones", "twos", "threes")))`

R Objects

55

- Arrays (stacks of matrices)
 - Create using the R function 'array':

```
array(data, dim, ...)
```

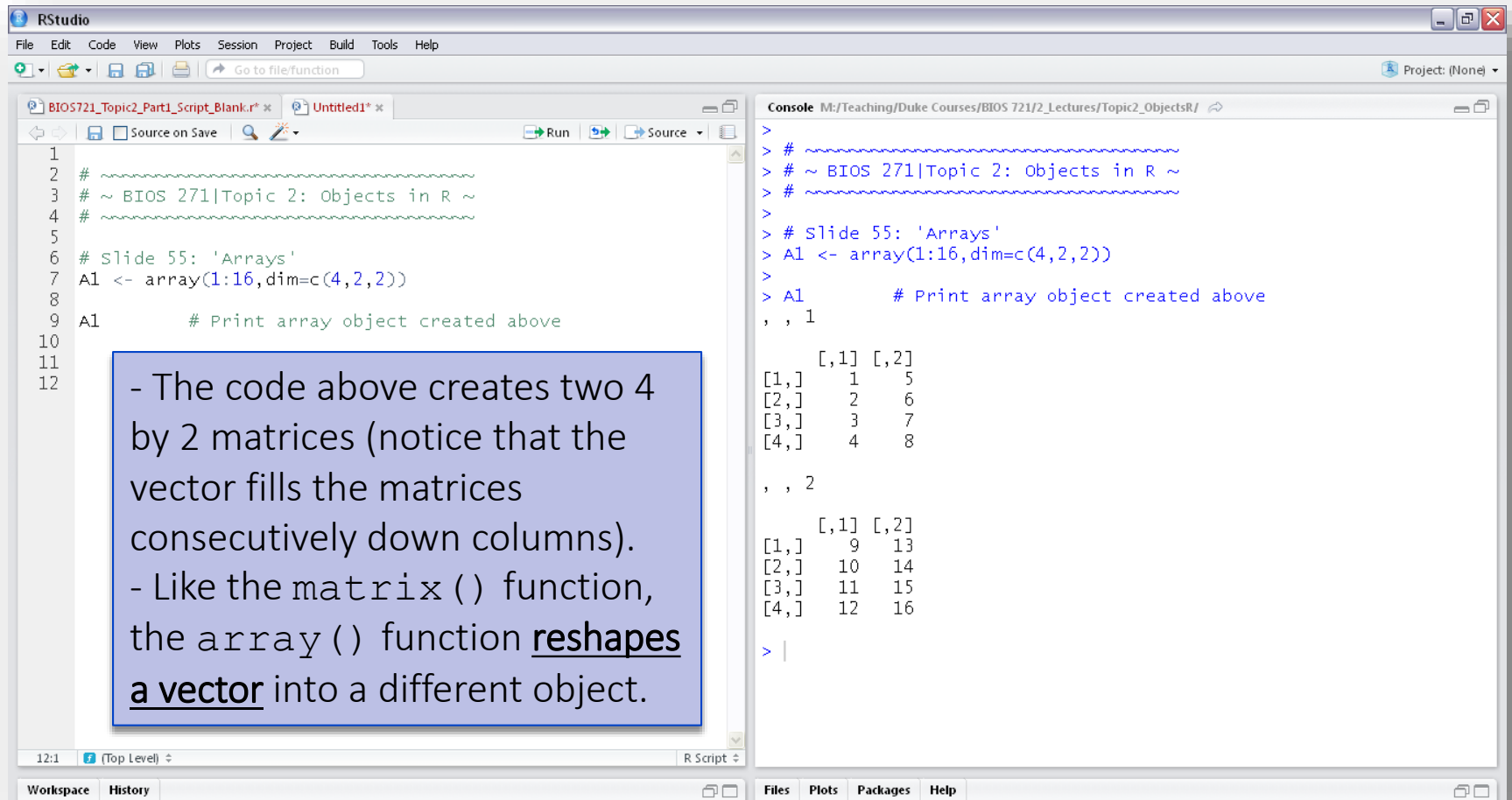
data = any VECTOR of values (numeric, character, or logical)
dim = a vector of length 3 giving the no. of rows in each matrix, no. of columns in each matrix, and the no. of matrices

- Examples

- `A1 <- array(1:16, dim=c(4, 2, 2))`
- `A2 <- array(c(rep(1, 4), rep(2, 4), rep(3, 4)),
dim=c(2, 2, 3), dimnames=list(c("R1", "R2"),
c("C1", "C2"),
c("ones", "twos", "threes")))`

R Objects

56



The screenshot shows the RStudio interface. The script editor on the left contains the following code:

```
1  
2 # ~~~~~  
3 # ~ BIOS 271|Topic 2: Objects in R ~  
4 # ~~~~~  
5  
6 # Slide 55: 'Arrays'  
7 A1 <- array(1:16,dim=c(4,2,2))  
8  
9 A1      # Print array object created above  
10  
11  
12
```

A blue box highlights the following text:

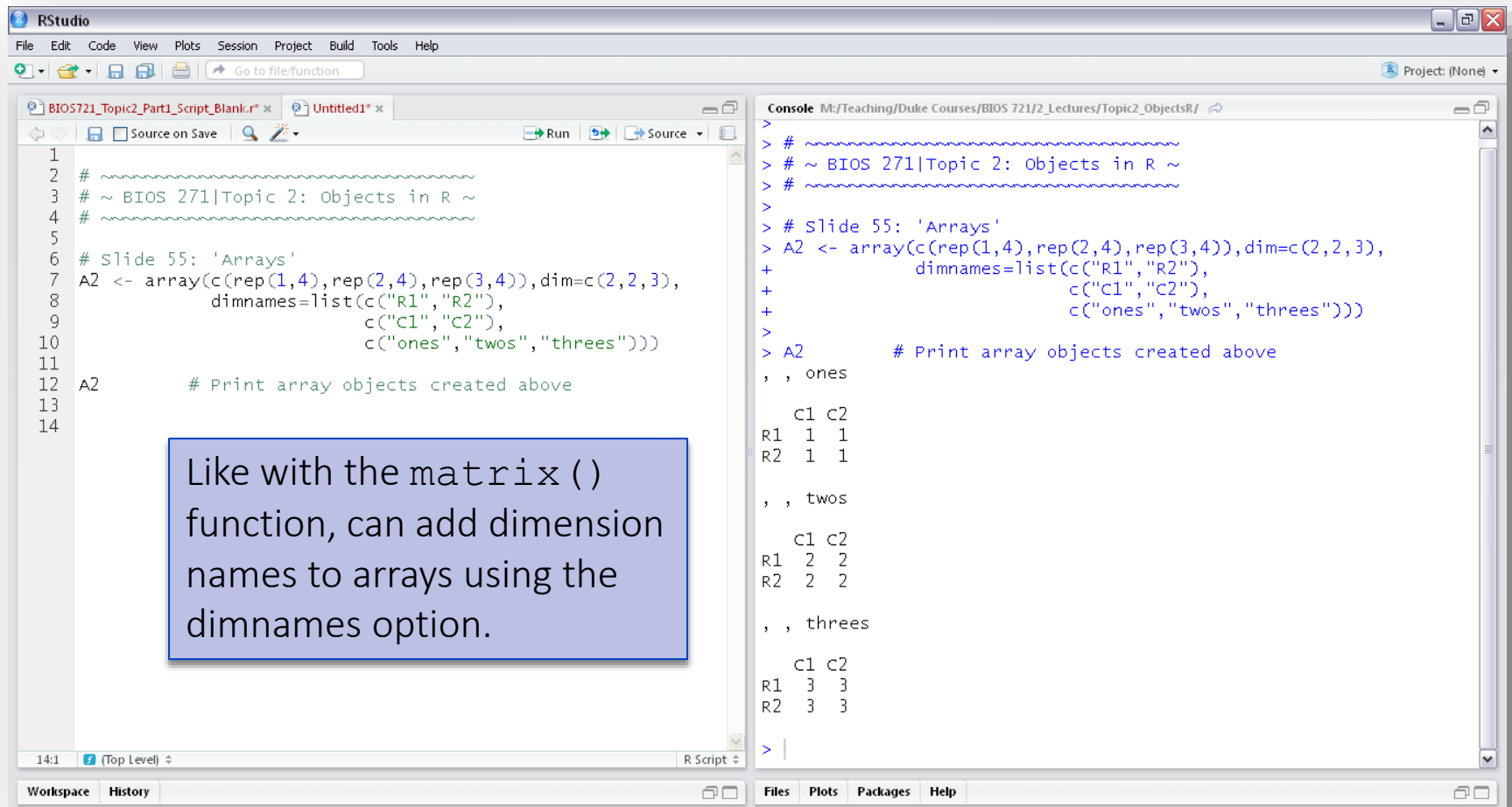
- The code above creates two 4 by 2 matrices (notice that the vector fills the matrices consecutively down columns).
- Like the `matrix()` function, the `array()` function reshapes a vector into a different object.

The console on the right shows the output of the code:

```
>  
> # ~~~~~  
> # ~ BIOS 271|Topic 2: Objects in R ~  
> # ~~~~~  
>  
> # Slide 55: 'Arrays'  
> A1 <- array(1:16,dim=c(4,2,2))  
>  
> A1      # Print array object created above  
, , 1  
  
      [,1] [,2]  
[1,]    1    5  
[2,]    2    6  
[3,]    3    7  
[4,]    4    8  
  
, , 2  
  
      [,1] [,2]  
[1,]    9   13  
[2,]   10   14  
[3,]   11   15  
[4,]   12   16  
  
> |
```


R Objects

57



The screenshot shows the RStudio interface. The source editor on the left contains R code for creating an array. The console on the right shows the execution of the code, including the output of the array object.

```
1  
2 # ~~~~~  
3 # ~ BIOS 271|Topic 2: Objects in R ~  
4 # ~~~~~  
5  
6 # Slide 55: 'Arrays'  
7 A2 <- array(c(rep(1,4),rep(2,4),rep(3,4)),dim=c(2,2,3),  
8           dimnames=list(c("R1","R2"),  
9                         c("C1","C2"),  
10                        c("ones","twos","threes")))  
11  
12 A2      # Print array objects created above  
13  
14
```

Console output:

```
> # ~~~~~  
> # ~ BIOS 271|Topic 2: Objects in R ~  
> # ~~~~~  
>  
> # Slide 55: 'Arrays'  
> A2 <- array(c(rep(1,4),rep(2,4),rep(3,4)),dim=c(2,2,3),  
+           dimnames=list(c("R1","R2"),  
+                         c("C1","C2"),  
+                        c("ones","twos","threes")))  
>  
> A2      # Print array objects created above  
, , ones  
  
  C1 C2  
R1  1  1  
R2  1  1  
  
, , twos  
  
  C1 C2  
R1  2  2  
R2  2  2  
  
, , threes  
  
  C1 C2  
R1  3  3  
R2  3  3  
  
> |
```

Like with the `matrix()` function, can add dimension names to arrays using the `dimnames` option.

R Objects

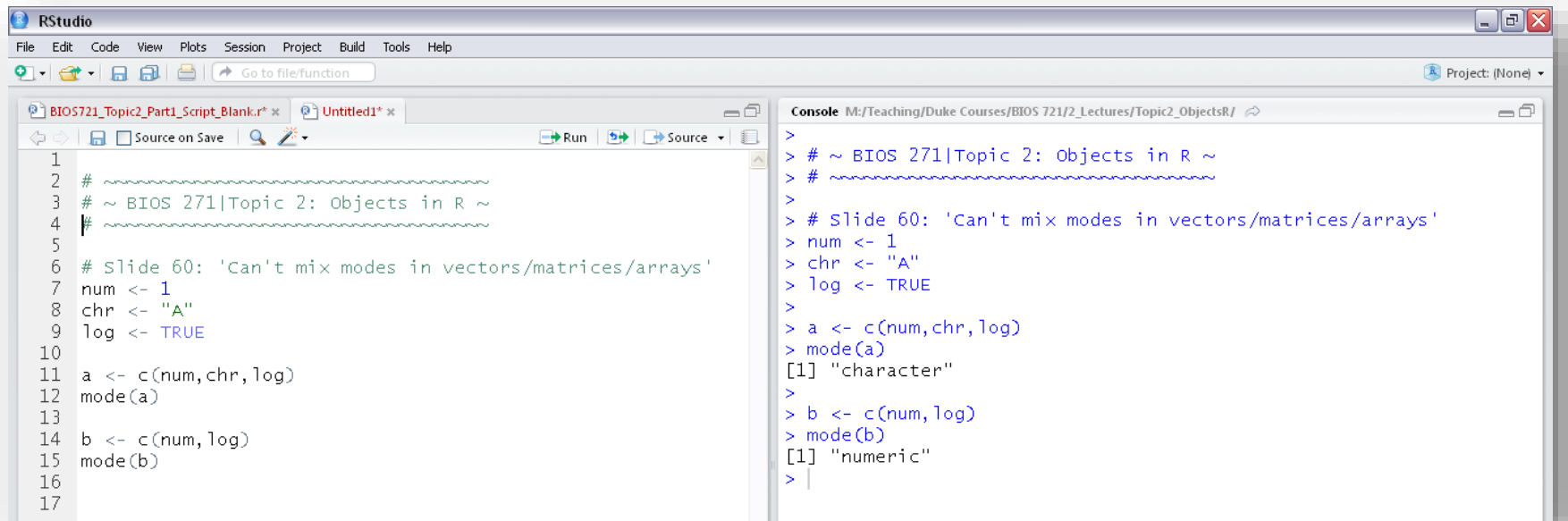
58

- For all of the R objects that we have talked about so far, all elements have to be of the SAME mode!
 - ▣ That is, the following objects have to have all numeric, character, or logical elements:
 - Vectors
 - Matrices
 - Arrays
 - ▣ No mixing!

R Objects

59

- If there is mixing, R will coerce the elements of these objects to all be of the same mode.



The screenshot shows the RStudio interface with a script editor on the left and a console on the right. The script editor contains R code that demonstrates coercion of different data types into a single mode (character) when they are combined into a vector.

```
1  
2 # ~~~~~  
3 # ~ BIOS 271|Topic 2: Objects in R ~  
4 # ~~~~~  
5  
6 # Slide 60: 'Can't mix modes in vectors/matrices/arrays'  
7 num <- 1  
8 chr <- "A"  
9 log <- TRUE  
10  
11 a <- c(num,chr,log)  
12 mode(a)  
13  
14 b <- c(num,log)  
15 mode(b)  
16  
17
```

The console shows the output of the code, confirming that the mode of the vector 'a' is 'character' and the mode of the vector 'b' is 'numeric'.

```
> # ~ BIOS 271|Topic 2: Objects in R ~  
> # ~~~~~  
>  
> # Slide 60: 'Can't mix modes in vectors/matrices/arrays'  
> num <- 1  
> chr <- "A"  
> log <- TRUE  
>  
> a <- c(num,chr,log)  
> mode(a)  
[1] "character"  
>  
> b <- c(num,log)  
> mode(b)  
[1] "numeric"  
> |
```

R Objects

60

- But in consulting and research that is not always practical or feasible.
 - ▣ Most data sets will have a mixture of (at least) numeric and character data values.
 - ▣ The last two R data objects allow you to mix modes or even object types.
 - Data Frames
 - Lists

R Objects

61

□ Data Frames

- Allow you to combine vectors of different modes together to create a data set (essentially a matrix)
- Create using the R function 'data.frame':

```
data.frame(data, ...)
```

■ Example

- `num <- 1:4`
- `chr <- c("M", "F", "F", "M")`
- `log <- c(TRUE, TRUE, FALSE, TRUE)`
- `D1 <- data.frame(num, chr, log)` compare to
- `D2 <- cbind(num, chr, log)`

R Objects

62

The data frame object maintains the original mode of each vector while the matrix object coerces all columns to be character vectors.

```
1 # ~~~~~
2 # ~ BIOS 271|Topic 2: Objects in R ~
3 # ~~~~~
4
5
6 # Slide 62: 'Data Frames'
7 num <- 1:4
8 chr <- c("M","F","F","M")
9 log <- c(TRUE,TRUE,FALSE,TRUE)
10 D1 <- data.frame(num,chr,log) # compared to ...
11 D2 <- cbind(num,chr,log)
12
13 D1      # Print data.frame objects created above
14 D2
15
16
```

```
> # ~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~
>
> # Slide 62: 'Data Frames'
> num <- 1:4
> chr <- c("M","F","F","M")
> log <- c(TRUE,TRUE,FALSE,TRUE)
> D1 <- data.frame(num,chr,log) # compared to ...
> D2 <- cbind(num,chr,log)
>
> D1      # Print data.frame objects created above
  num chr  log
1  1  M TRUE
2  2  F TRUE
3  3  F FALSE
4  4  M TRUE
> D2
  num chr log
[1,] "1" "M" "TRUE"
[2,] "2" "F" "TRUE"
[3,] "3" "F" "FALSE"
[4,] "4" "M" "TRUE"
> |
```

R Objects

63

□ Lists

- Allow you to combine objects of different modes and objects of different classes (types) into a single object
- Create using the R function 'list':

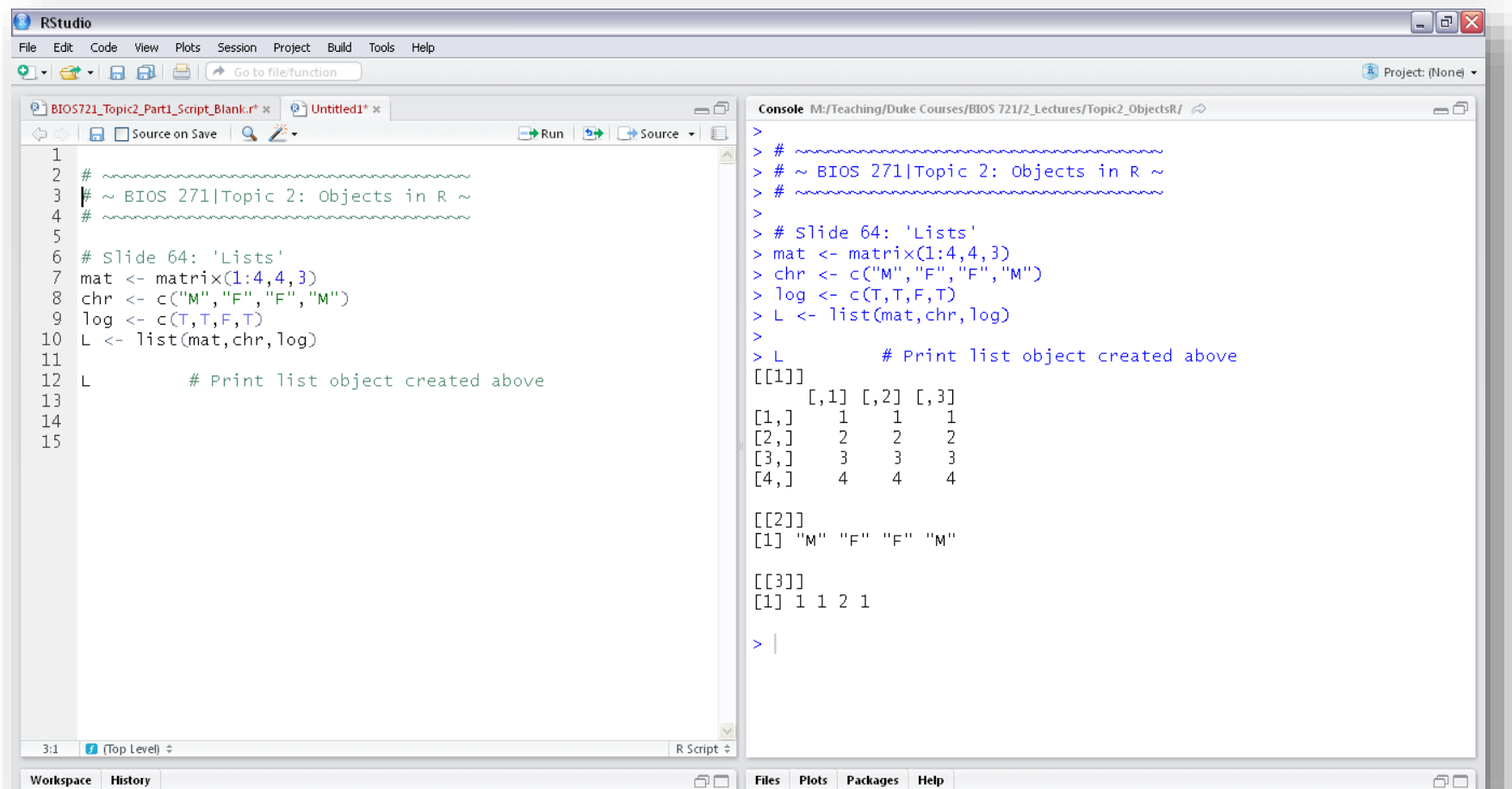
```
list(data, ...)
```

■ Example

- `mat <- matrix(1:4, 4, 3)`
- `chr <- c("M", "F", "F", "M")`
- `log <- c(TRUE, TRUE, FALSE, TRUE)`
- `L <- list(mat, chr, log)`

R Objects

64



The screenshot shows the RStudio interface with a script editor on the left and a console on the right. The script editor contains R code for creating a list object. The console shows the output of the code, including the creation of a matrix, a character vector, a logical vector, and a list containing these objects. The list is printed in a structured format showing its components.

```
1  
2 # ~~~~~  
3 # ~ BIOS 271|Topic 2: Objects in R ~  
4 # ~~~~~  
5  
6 # Slide 64: 'Lists'  
7 mat <- matrix(1:4,4,3)  
8 chr <- c("M","F","F","M")  
9 log <- c(T,T,F,T)  
10 L <- list(mat,chr,log)  
11  
12 L          # Print list object created above  
13  
14  
15
```

```
>  
> # ~~~~~  
> # ~ BIOS 271|Topic 2: Objects in R ~  
> # ~~~~~  
>  
> # Slide 64: 'Lists'  
> mat <- matrix(1:4,4,3)  
> chr <- c("M","F","F","M")  
> log <- c(T,T,F,T)  
> L <- list(mat,chr,log)  
>  
> L          # Print list object created above  
[[1]]  
      [,1] [,2] [,3]  
[1,]    1    1    1  
[2,]    2    2    2  
[3,]    3    3    3  
[4,]    4    4    4  
  
[[2]]  
[1] "M" "F" "F" "M"  
  
[[3]]  
[1] 1 1 2 1  
  
> |
```


R Objects

65

- If you want to know the mode of the elements in an object (that does not allow mixing) you can use
 - ▣ `mode(object)`
 - ▣ Returns one of the following
 - `"numeric"`
 - `"character"`
 - `"logical"`

R Objects

66

- If you want to know the type of object that you are dealing with you can use
 - ▣ `class(object)`
 - ▣ Returns one of the following
 - `"numeric"`
 - `"integer"`
 - `"character"`
 - `"logical"`
 - `"matrix"`
 - `"array"`
 - `"data.frame"`
 - `"list"`
- a.k.a. scalar/vector of a particular mode

For Next Time

67

- Next Time Topic 1 continued (Part 2)
 - ▣ Operations performed on objects