The EMaC R Manual

Alex Sciuto

Last Updated Febuary 2020

Contents

Chapter 1

How to Install R

This is the EMaC manual for how to wrangle, analyze, and visualize data in **R**. I will review a few key packages, in addition to explaing their key functions, with real data examples.

There are two things you need to do to install R on your computer. First, you would need to install the latest version of R which you would download and install from this link: R-download. Once you run and install R onto your computer, you would need to install R-Studio. R studio is the graphic user interface (GUI) where you can place all of your R-code. Follow this link: R-Studio Desktop Download. Once you have these two programs installed, all you need to do is launch R-studio Desktop and you are ready to go.

Chapter 2

Introduction to Programming in R

Before you can do data wrangling, statiscs, and visualization using R for cognitive psychology research, you need to learn the basic syntax in R. This chapter will introduce the basics.

2.1 Variables in R

R can manipulate and wrangle all kinds of data, these data could be stored in a handful of variables that we can then work with. The first one we will be talking about is numeric.

2.1.1 Numeric

a numeric is a number (including decimals) that can be stored within a variable. Here is an example:

```
x = 2
```

Here, we assigned the numeric value `2` to the variable `x`. Thus, if we were to do arithemtic, `x` will be treated as `2`. Moreover, there are specific functions in R we can use in order to do arithmetic with numeric variables. Here are a few examples:

Arithemtic	R-Input	R-Output
Addition:	`x + 2`	`4`
Substraction:	`x - 2`	`0`

Arithemtic	R-Input	R-Output
Division:	`x / 2`	1
Multiplication:	`x * 2`	4
Exponent:	`x ^ 2`	4

2.1.2 Character

A character is a combination of characters (either letters and/or numbers) that can be stored within a variable. Moreover, it is very important that you place the character between quotation marks. Here is an example:

`x = "Cognitive Psychology"`

Here, we assigned the character value `"Cognitive Psychology"` to the variable `x`. There are ways to manipulate character type variables, and also modify them. However, we will touch on that later.

2.1.3 Logical

A logical can only have two possible values, `TRUE` and `FALSE` that can be stored within a variable. This is not typically a varaible type that you will need to assign variable values to. However, the logical type variable is extremely important becuase a lot of functions in R return a logical variable. We will dive into some of these functions

2.1.4 Vector

A vector is multiple variables stored into one data-set. Vectors can contain any variable type: character, numeric, string, and logical. When you are declaring a a vector type variable, you typically have to use the concatenate function `c()`. Here is an example:

`x = c(1,2,3,4)`

`x = c("I", "like", "Cognitive", "Psychology")`

`x = c(TRUE, FALSE, FALSE, TRUE)`

To reference a vector varaible, you need to use `[]` and inside the brackets, you have to specify the index of where the given variable in the vector is. Here is an example

```r
x = c("10", "20", "30", "40")
x[4]
```

```
## [1] "40"
```

Here, we assigned the numbers: `10, 20, 30, 40` to the vector `x`. `"Cognitive Psychology"` to the variable `x`. There are ways to manipulate character type variables, and also modify them. However, we will touch on that later.

2.2 Logical Operators & if-Statements

Now that you have a basic understanding of how variables work in R, the next step is to learn how to use logical operators in order to specifiy what are the specific conditions under which you want your variables to be manipulated. The main way to do this in R is by using logical operators.

```
x = 2
```

Here, we assigned the numeric value `2` to the variable `x`. Thus, if we were to apply logical operators to `x`, `x` will be treated as `2`. Here are examples of all of the logical operators:

Logical Operators	Description	R-Input	R-Output
<	less than	x < 10	TRUE
<=	less than or equal to	x <= 2	TRUE
>	greater than	x > 1	TRUE
>=	greater than or equal to	x >= 3	FALSE
==	exactly equal to	x == 9	FALSE
!=	not equal to	x != 10	FALSE
!x	Not x	!x	FALSE
x | y	x OR y	x == 10 | x != 10	TRUE
x & y	x AND y	x == 10 &x != 10	FALSE

Now that you have a basic understanding of logical operators, we can use them in order to specify code to manipulate variables in a particular way. For this example, we will be working with the function `if_else()`. However, one question you may have is what is a function in R? I will describe the structure of functions, starting with a simple one.

Chapter 3

Introduction to Data Wrangling

To introduce data wrangling, we will be working with a dataset from a study conducted in our lab. Here are the details of the study in order to understand the data manipulation more:

3.1 The study

On each trial in this experiment (n = 174), each participant (n = 84) sees a target word very briefly (e.g., word) and then is prompted to select which of two letters was in a particular position of that word (e.g., _ _ _ ↓) - one letter was in the presented word (e.g., D) and the other letter is in an orthographic neighbor of the word (e.g., K). This is a 2 (visual field) x 3 (sentence context) experimental design: the target word is either presented in the fovea (i.e., center of the screen) or the parafovea (i.e., 3 degrees to the right of fixation). Prior to the target word, they see a sentence context presented via Rapid Serial Visual Presentation (RSVP) that constrains to target (i.e., makes the presented word predictable and the orthographic neighbor implausible), constrains to the alternative (i.e., makes the orthographic neighbor predictable and the presented word implausible), or is neutral (i.e., makes neither word predictable but both of them plausible). In addition, we collect data about the individual subjects' language ability (i.e., their z-score on some test relative to the other participants), including spelling recognition (i.e., circle which words are spelled wrong), spelling dictation (i.e., write out words that they hear said), and phonological decoding (i.e., read aloud a list of words and a list of nonwords), and information about the lexical properties of the words (e.g., word frequency, cloze probability, orthographic neighborhood size, phonological neighborhood size, clustering coefficient, orthographic similarity to other words, which of the two words is higher frequency).

3.2 Loading in Data

3.2.1 Loading Packages

Before we start wrangling the data, we need to load in the packages we are using. In this lab, most of the data wrangling tools we will be using will be located in the `tidyverse()` package. What is a package? It is simply a group of functions that group of developers made that makes computations easier. To learn more about the `tidyverse` package, you may click on this link and read on it! For the data wrangling we will be doing in this section, we will be primarily using a package within tidyverse called `dplyr`. to load in the package, simply type in the following code. What this code is saying is IF you don't have the `tidyverse` installed, THEN install the package. After that, load the `tidyverse` with the `library()` function.

```
if(!require(tidyverse)){
    install.packages("tidyverse")
}
library(tidyverse)
```

3.2.2 setting the working directory

Now that we have the packages that we need, we need to load up our directory. What is a directory? It is essentially the folder we will be using to read in files, or export files into. In this directory, we have our data set of interest, `Topgown_Data`

```
setwd("C:/Users/alexs/Documents/Github/Emac_Rmanual/")
```

3.2.3 Reading in Data

Now that we have our working directory set we can use the `read_csv()` function which requires one minimum argument to work which is the name of your `.csv` file. Here we are setting the variable `mydata` to the contents within the `.csv` file. It stores the contents of `Topgown_Data.csv` in what is called a data frame. What is a dataframe? it is simply a matrix where each row constitutes a variable (which can be any type you want), and rows are observations.

```
mydata = read_csv("TopGown_Data.csv")
```

X1

Participants

visual_field

constrained_to

presented_word

predicted_word

question

correct_answer

item_pair

Items

probe_position

RT

accuracy

zTOWRE_Word

zTOWRE_Nonword

zSpelling_Dictation

zSpelling_Recogntion

zAll

zSpell

zTOWRE

OrthoN_T

OrthoCC_T

PhonoN_T

Freq_T

Freq_P

OrthoCC_P

OLD

Higher_Freq

1

tg011

Fovea

A

cage

cape

NA

NA

19

37

3

2986.73

1

-1.749307

0.071771

-0.0918699

1.155124

-0.1535705

0.531627

-0.838768

13

0.0000001

17

9.048

8.605

0.0000000

1.10

Target

2

tg011

Parafovea

N

ache

acne

Were they very frustrated?

Y

83

165

3

4146.67

1

-1.749307

0.071771

-0.0918699

1.155124

-0.1535705

0.531627

-0.838768

3

1.5000000

23

6.741

6.284

1.5000000

1.75

Target

3

tg011

Fovea

A

male

sale

NA

NA

37

73

1

2496.54

1

-1.749307

0.071771

-0.0918699

1.155124

-0.1535705

0.531627

-0.838768

27

0.0000000

59

10.978

11.466

0.0000000

1.00

Alternative

4

tg011

Fovea

A

leaf

loaf

NA

NA

70

139

2

1845.46

1

-1.749307

0.071771

-0.0918699

1.155124

-0.1535705

0.531627

-0.838768

9

0.0500000

29

8.861

7.109

0.5000000

1.75

Target

5

tg011

Parafovea

T

step

stop

NA

NA

7

14

3

2125.17

1

-1.749307

0.071771

-0.0918699

1.155124

-0.1535705

0.531627

-0.838768

7

0.1666667

11

10.691

11.478

0.0416667

1.50

Alternative

6

tg011

Fovea

N

mint

mind

NA

NA

24

48

4

1415.47

1

-1.749307

0.071771

-0.0918699

1.155124

-0.1535705

0.531627

-0.838768

16

0.0000005

17

11.344

11.784

0.0000001

1.25

Alternative

7

tg011

Parafovea

A

cast

case

NA

NA

20

40

4

3168.45

1

-1.749307

0.071771

-0.0918699

1.155124

-0.1535705

0.531627

-0.838768

19

0.0000000

28

10.102

12.204

0.0000000

1.00

Alternative

8

tg011

Fovea

A

golf

wolf

NA

NA

76

151

1

3765.47

1

-1.749307

0.071771

-0.0918699

1.155124

-0.1535705

0.531627

-0.838768

4

0.0000000

2

9.254

9.979

0.0000000

1.90

Alternative

9

tg011

Fovea

N

mane

maze

NA

NA

3

6

3

1503.80

1

-1.749307

0.071771

-0.0918699

1.155124

-0.1535705

0.531627

-0.838768

27

0.0000000

59

6.238

8.775

0.0000001

1.25

Alternative

10

tg011

Parafovea

N

name

fame

NA

NA

17

33

1

1565.79

1

-1.749307

0.071771

-0.0918699

1.155124

-0.1535705

0.531627

-0.838768

11

0.0041667

21

12.604

8.714

0.0000000

1.15

Target

3.3 Dplyr() Package

We will now be wrangling this new dataset we imported into R using several `dplyr` functions. These are the ones we will be covering. In the following sections I will explain what each functions does in detial.

Function	Description
select()	keeps only the variables you mention
arrange()	Order table rows by an expression involving its variables
filter()	choose rows/cases where conditions are true.
mutate()	adds new variables and preserves existing ones
summarize()	Create new variables summarizing the variables of an existing table
group_by()	takes an existing table and converts it into a grouped table where operations are performed "by group"

3.3.1 select()

For this particular experiment we have two independent variables of interest: `visual_field` and `constrained_to`; and four participant variables of interest: `zTOWRE_Word`,`zTOWRE_Nonword`, `zSpelling_Dictation`,`zSpelling_Recogntion`. What if we are only interested in these variables, and we don't particularly need word level variables such as `ortho_N`, or `phono_N`. We can then select these variables.

For this particular function, the first argument is going to be the dataset of interest. the dataset for we will be selecting from is `mydata`. The following arguments are simply the columns you would like to keep.

```
dplyr::select(mydata, File_Name, visual_field, constrained_to, zTOWRE_Word,zTOWRE_Nonword,  zSpel
```

Participants

visual_field

constrained_to

zTOWRE_Word

zTOWRE_Nonword

zSpelling_Dictation

zSpelling_Recogntion

tg011

Fovea

A

-1.749307

0.071771

-0.0918699

1.155124

tg011

Parafovea

N

-1.749307

0.071771

-0.0918699

1.155124

tg011

Fovea

A

-1.749307

0.071771

-0.0918699

1.155124

tg011

Fovea

A

-1.749307

0.071771

-0.0918699

1.155124

tg011

Parafovea

T

-1.749307

0.071771

-0.0918699

1.155124

tg011

Fovea

N

-1.749307

0.071771

-0.0918699

1.155124

tg011

Parafovea

A

-1.749307

0.071771

-0.0918699

1.155124

tg011

Fovea

A

-1.749307

0.071771

-0.0918699

1.155124

tg011

Fovea

N

-1.749307

0.071771

-0.0918699

1.155124

tg011

Parafovea

N

-1.749307

0.071771

-0.0918699

1.155124

3.3.1.1 The Pipe (%>*%)

However, another way I would suggest writing code like this is to use a function in the `tidyverse` called the pipe (`%>%`). Like the `select()` function, the first argument in all `tidyverse` functions will be your dataset of interest. The pipe simply gets a dataset, and pushes it into the first argument of a function. Here is an example.

```
mydata %>% dplyr::select(Participants, visual_field, constrained_to, zTOWRE_Word,zTOWR
  dplyr::select(Participants, visual_field, constrained_to)
```

Participants

visual_field

constrained_to

tg011

Fovea

A

tg011

Parafovea

N

tg011

Fovea