

# The EMaC R Manual

Alex Sciuto

Last Updated Febuary 2020



# Contents

<b>1</b>	<b>How to Install R</b>	<b>5</b>
<b>2</b>	<b>Introduction to Programming in R</b>	<b>7</b>
2.1	Variables in R . . . . .	7
2.2	Logical Operators . . . . .	9
<b>3</b>	<b>Introduction to Data Wrangling</b>	<b>11</b>
3.1	The study . . . . .	11
3.2	Loading in Data . . . . .	12
3.3	Dplyr() Package . . . . .	22
<b>4</b>	<b>Plotting</b>	<b>35</b>



# 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 explaining 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, statistics, 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 arithmetic, **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:

Arithmetic	R-Input	R-Output
Addition:	<b>x + 2</b>	4
Subtraction:	<b>x - 2</b>	0
Division:	<b>x / 2</b>	1
Multiplication:	<b>x * 2</b>	4

Arithmetic	R-Input	R-Output
Exponent:	<code>x ^ 2</code>	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 variable type that you will need to assign variable values to. However, the logical type variable is extremely important because 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 variable, 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

```
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

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 specify 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



## 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`

### 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", skip_empty_rows = TRUE)
```

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\_Recognition  
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 detail.

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

### 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_Recognition`. 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, Participants, visual_field, constrained_to, zTOWRE_Word, zTOWRE_N
```

Participants

visual\_field

constrained\_to

zTOWRE\_Word

zTOWRE\_Nonword

zSpelling\_Dictation

zSpelling\_Recognition

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, zTOWRE_Score)
dplyr::select(Participants, visual_field, constrained_to)

```

```

Participants
visual__field
constrained_to
tg011
Fovea
A

```

```
tg011
Parafovea
N
tg011
Fovea
A
tg011
Fovea
A
tg011
Parafovea
T
tg011
Fovea
N
tg011
Parafovea
A
tg011
Fovea
A
tg011
Fovea
N
tg011
Parafovea
N
```

In this regard, the pipe follows simple logic. Once a dataset is manipulated by one function, you pass it to the next function.

### 3.3.2 group\_by() & summarise()

However, let us say we are particularly interested in the values of one variable. For example, word properties. Therefore, we would need to compile a new dataset that summarizes word properties, such as orthographic neighborhood size. So in this example we will create a dataframe summarized by the presented word.

```
mydata %>% dplyr::select(Participants, visual_field, constrained_to, presented_word, OrthoN_T)
  group_by(presented_word) %>%
  dplyr::summarise(OrthoN_T = mean(OrthoN_T))
```

presented\_\_word

OrthoN\_T

ache

3

acne

4

ants

7

arts

8

aunt

6

ball

20

bats

23

beef

6

bees

20

bell

15

### 3.3.3 arrange()

Let us say that we are still interested in neighborhood size of a particular target word. we can arrange the these variables so we can display the the words from lowest orthographic neighborhood size to higher orthographic neighborhood size. For this we can use the arrange function, which sorts a particular variable of interest in ascending or descending order.

```
mydata %>% dplyr::select(Participants, visual_field, constrained_to, presented_word, OrthoN_T, Fr
  group_by(presented_word) %>%
  dplyr::summarise(OrthoN_T = mean(OrthoN_T)) %>%
  dplyr::arrange(OrthoN_T)
```

presented\_\_word

OrthoN\_T

ache

3

film

3

gyms

3

wolf

3

womb

3

acne

4

clue

4

debt

4

drum

4

golf

4

### 3.3.4 filter()

Let us say that not only are we interested in orthographic neighborhood size, but we want to know orthographic neighborhood size for a particular condition in our study. We can then use the `filter()` function, and only keep the trails that are within the condition of interest. For this example, we are interested when the sentence is constraining toward the target word and when the target word is presented in the parafovea.

```
mydata %>% dplyr::select(Participants, visual_field, constrained_to, presented_word, OrthoN_T) %>%
  filter(visual_field == "Parafovea" & constrained_to == "T") %>%
  group_by(presented_word, visual_field, constrained_to) %>%
  dplyr::summarise(OrthoN_T = mean(OrthoN_T)) %>%
  dplyr::arrange(OrthoN_T)
```

presented\_word

visual\_field

constrained\_to

OrthoN\_T

ache

Parafovea

T

3

film

Parafovea

T

3

gyms

Parafovea

T

3

wolf

Parafovea

T

3

womb

Parafovea

T

3

acne

Parafovea

T

4

clue

Parafovea

T

4

debt

Parafovea

T

4

drum

Parafovea

T

4

golf

Parafovea

T

4

### 3.3.5 `mutate()`

Let us say that we are not only interested in word properties like orthographic neighborhood size, but also the spelling ability of particular participants. In this study we have four spelling tests, which were measured through the following variables: `zTowre_Word`, `zTowre_Nonword`, `Spelling_Dictation`, `Spelling_Recognition`. However, these are a lot of measures. what if we wanted to know the average between all of these measures for each participant? we can use the `mutate` function to do this.

```
mydata %>% dplyr::select(Participants, zTOWRE_Word, zTOWRE_Nonword, zSpelling_Dictation)
  group_by(Participants) %>%
    dplyr::summarise(zTOWRE_Word = mean(zTOWRE_Word),
                     zTOWRE_Nonword = mean(zTOWRE_Nonword),
                     zSpelling_Dictation = mean(zSpelling_Dictation),
                     zSpelling_Recognition = mean(zSpelling_Recognition)) %>%
    dplyr::mutate(aggregated_spelling = ((zTOWRE_Word + zTOWRE_Nonword + zSpelling_Dictation + zSpelling_Recognition) / 4))
```

Participants

zTOWRE\_Word

zTOWRE\_Nonword

zSpelling\_Dictation

zSpelling\_Recognition

aggregated\_spelling

derp033

1.6354380

0.7156786

0.8383130

1.3936520

1.1457704

derp044

-0.6333223

0.5317050

-0.0918699

-0.7923070

-0.2464486

derp055

0.2251274

0.6236918

-0.7895071

0.2705651

0.0824693

derp066



-1.1483920

-1.1240570

-2.8824190

-1.4607150

-1.6538958

derp071

-0.3757874

-0.1122026

0.6057672

0.6449458

0.1906808



## Chapter 4

# Plotting

Now that we have a basic understanding of how to wrangle data, now we want to graphically explain the data we are wrangling. For this chapter we will be using `ggplot` package to further understand the data. However, before we go into plotting specific things about the data, we need to learn the grammar of plotting with `ggplot` in general

### 4.0.1 ggplot grammar

You can think of the grammar of graphics as a systematic approach for describing the components of a graph. It has seven components (the ones in bold are required to be specified explicitly in `ggplot2`):

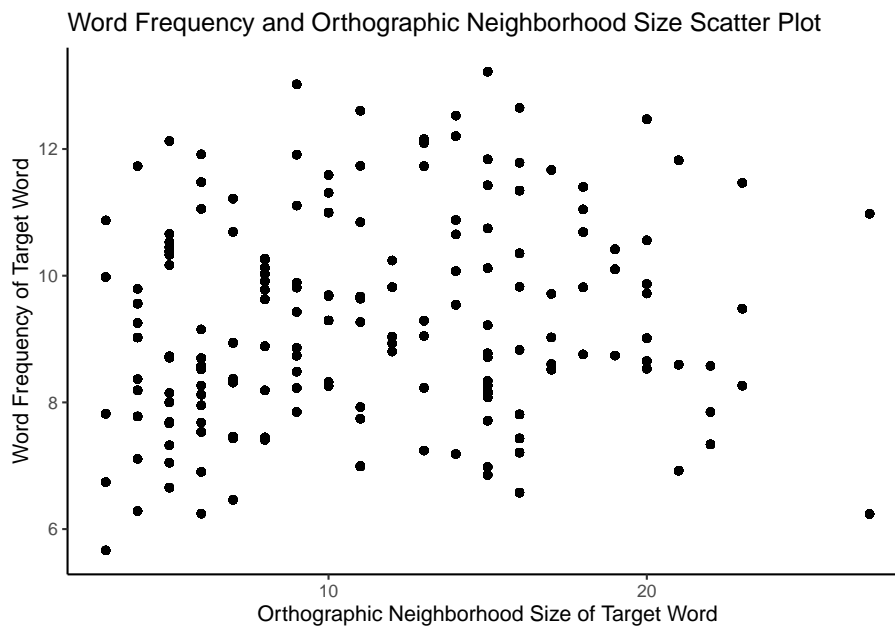
- ***Data***
  - data that you’re feeding into a plot.
- ***Aesthetic mappings***
  - How are variables (columns) from your data connect to a visual dimension? Horizontal positioning, vertical positioning, size, colour, shape, etc. These visual dimensions are called “aesthetics”
- ***Geometric objects***
  - What are the objects that are actually drawn on the plot? A point, a line, a bar, a histogram, a density, etc.
- Scales
  - How is a variable mapped to its aesthetic? Will it be mapped linearly? On a log scale? Something else? This includes things like the color scale e.g., `c(control, treatment_1, treatment_2) -> c(“blue”, “green”, “red”)`
- Statistical transformations
  - Whether and how the data are combined/transformed before being plotted e.g., in a bar chart, data are transformed into their frequencies; in a box-plot, data are transformed to a five-number summary.

- Coordinate system
  - This is a specification of how the position aesthetics (x and y) are depicted on the plot. For example, rectangular/cartesian, or polar coordinates.
- Facet
  - This is a specification of data variables that partition the data into smaller “sub plots”, or panels. These components are like parameters of statistical graphics, defining the “space” of statistical graphics. In theory, there is a one-to-one mapping between a plot and its grammar components, making this a useful way to specify graphics.

## 4.0.2 Example: Scatterplot Grammar

Let us say that we are interested in how specific word properties relate to one another, and how correlated they are. With this information we can learn specific details about the nature of our linguistic stimuli, which is very important in psycholinguistics. So for this example we are going to plot the orthographic neighborhood size of the target word and word frequency.

```
mydata %>% ggplot(aes(x = OrthoN_T, y = Freq_T)) + # this defines the x and y axes of
  geom_point() + # this adds geometric objects
  theme_update(plot.title = element_text(hjust = 0.5)) + # this line centers the title
  xlab("Orthographic Neighborhood Size of Target Word") + # this line sets the label for x-axis
  ylab("Word Frequency of Target Word") + # this line sets the label for y-axis
  ggtitle("Word Frequency and Orthographic Neighborhood Size Scatter Plot") + # this sets the title
  theme_classic() # this sets the theme for the plot
```



Grammar Component	Specification
data	mydata
aesthetic mapping	x: OrthoN_T, y: Freq_T
geometric object	points
scale	x: linear, y: linear
statistical transform	none
coordinate system	rectangular
facetting	none

### 4.0.3 Example: Histogram Grammar

Useful for depicting the distribution of a continuous random variable. Partitions the number line into bins of certain width, counts the number of observations falling into each bin, and erects a bar of that height for each bin.

Required aesthetics:

- **x**: A numeric vector.
- By default, a histogram plots the count on the y-axis. If you want to use proportion (i.e., “density”), specify the `y = ..density..` aesthetic.
- You can change the smoothness of the plot via two arguments (your choice):
- **bins**: the number of bins/bars shown in the plot.

- `binwidth`: the width of the bins shown on the plot.

Let us say that we want to make sure our linguistic stimuli is balanced across specific word properties such as word frequency. We want to know if there are too many high frequency words as compared to low ones, or vice versa. So here we are going to plot a histogram displaying the distribution of word frequency across items. However, what if we are also interested if this varies across sentence constraints? Then we can use the `fill` argument in the `aes()` call to do this.

```
mydata %>% ggplot(aes(x = OrthoN_T, fill = constrained_to)) +
  geom_histogram(bins = 20, alpha = .7) +
  theme_update(plot.title = element_text(hjust = 0.5)) +
  scale_fill_discrete(name = "Sentence Constraint", labels=c("Alternative", "Neutral",
  xlab("Word Frequency of Target Word") +
  ylab("Count") +
  ggtitle("Word Frequency Distribution for Target Words") +
  theme_classic()
```

