Getting Started in R: Tinyverse Edition

Saghir Bashir and Dirk Eddelbuettel

This version was compiled on October 11, 2018

Are you curious to learn what R can do for you? Do you want to see how it works? Yes, then this “Getting Started” guide is for you. It uses realistic examples and a real life dataset to manipulate, visualise and summarise data. By the end of it you will have an overview of the key concepts of R.

R | Statistics | Data Science | Tinyverse

1. Preface

This “Getting Started” guide will give you a flavour of what R can do for you. To get the most out of this guide, read it whilst doing the examples and exercises using RStudio.

This note is a variant of the original document but stresses the use of Base R along with careful dependency management as discussed below.

Experiment Safely. Be brave and experiment with commands and options as it is an essential part of the learning process. Things can (and will) go “wrong”, like, getting error messages or deleting things that you create by using this guide. You can recover from most situations (e.g. by restarting R). To do this “safely” start with a fresh R session without any other data loaded (otherwise you could lose it).

2. Introduction

Before Starting. Make sure that:

1. R and RStudio are installed.
2. https://eddelbuettel.github.io/gsir-te/Getting-Started-in-R.zip has been downloaded and unzipped
3. Double click “Getting-Started-in-R.Rproj” to open RStudio with the setup for this guide.

Starting R & RStudio. R starts automatically when you open RStudio (see Figure 1). The console starts with information about the version number, license and contributors. The last line is a standard prompt “>” that indicates R is ready and expecting instructions to do something.

Fig. 1. RStudio Screenshot with Console on the left and Help tab in the bottom right

Quitting R & RStudio. When you quit RStudio you will be asked whether to Save workspace with two options:

- “Yes” – Your current R workspace (containing the work that you have done) will be restored next time you open RStudio.
- “No” – You will start with a fresh R session next time you open RStudio. For now select “No” to prevent errors being carried over from previous sessions.

3. R Help

We strongly recommend that you learn how to use R’s useful and extensive built-in help system which is an essential part of finding solutions to your R programming problems.

help() function. From the R “Console” you can use the help() function or ?. For example, try the following two commands (which give the same result):

```r
help(mean)
?mean
```

Keyword search. To do a keyword search use the function apropos() with the keyword in double quotes (“keyword”) or single quote (‘keyword’). For example:

```r
apropos("mean")
# [1] ".colMeans" ".rowMeans"
# [3] "colmens" "kmeans"
# [5] "mean" "mean_cl_boot"
# [7] "mean_cl_normal" "mean_sdl"
# [9] "mean_se" "mean.Date"
#[11] "mean.default" "mean.difftime"
#[13] "mean.POSIXct" "mean.POSIXlt"
# [15] "rowMeans" "weighted.mean"
```

Help Examples. Use the example() function to run the examples at the end of the help for a function:

```r
example(mean)
#
# mean> z <- c(0:10, 50)
# mean> zm <- mean(z)
# mean> c(zm, mean(z, trim = 0.10))
# [1] 8.75 5.50
```

RStudio Help. Rstudio provides search box in the “Help” tab to make your life easier (see Figure 1).

Searching On-line For R Help. There are a lot of on-line resources that can help. However you must understand that blindly copying and pasting could be harmful and further it won’t help you to learn and develop. When you search on-line use [R] in your search term (e.g. “[R] summary statistics by group”). Note that often there is more than one solution to your problem. It is good to investigate the different options.
Exercise. Try the following:

1. help(median)
2. ?sd
3. ?max

Warning. If an R command is not complete then R will show a plus sign (+) prompt on second and subsequent lines until the command syntax is correct.

To break out this, press the escape key (ESC).

Hint. To recall a previously typed commands use the up arrow key (↑). To go between previously typed commands use the up and down arrow (↓) keys. To modify or correct a command use the left (←) and right arrow (→) keys.

4. Some R Concepts

In R speak, scalars, vectors/variables and datasets are called objects. To create objects (things) we have to use the assignment operator <-. For example, below, object height is assigned a value of 173 (typing height shows its value):

```r
height <- 173
height
# [1] 173
```

Warning: R is case sensitive. age and AgE are different:

```r
age <- 10
AgE <- 50

age
# [1] 10
AgE
# [1] 50
```

New lines. R commands are usually separated by a new line but they can also be separated by a semicolon (;)

```r
Name <- "Leo"; Age <- 25; City <- "Lisbon"
Name; Age; City
```

Comments. It is useful to put human readable comments in your programs. These comments could help the future you when you go back to your program. R comments start with a hash sign (#). Everything after the hash to the end of the line will be ignored by R.

```r
# This comment line will be ignored when run.
City  # Text after "#" is ignored.
# [1] "Lisbon"
```

5. R as a Calculator

You can use R as a calculator. Try the following:

```r
2 + 3
# [1] 5
(5*11)/4 - 7
# [1] 6.75
# ^ = "to the power of"
7^3
# [1] 343
```

Other math functions. You can also use standard mathematical functions that are typically found on a scientific calculator.

• Trigonometric: sin(), cos(), tan(), acos(), asin(), atan()
• Rounding: abs(), ceiling(), floor(), round(), sign(), signif(), sqrt(), trunc()
• Logarithms & Exponentials: exp(), log(), log10(), log2()

```r
# Square root
sqrt(2)
# [1] 1.41421
# Round down to nearest integer
floor(8.6178)
# [1] 8
# Round to 2 decimal places
round(8.6178, 2)
# [1] 8.62
```

Exercise. What do the following pairs of examples do?

1. ceiling(18.33) and signif(9488, 2)
2. exp(1) and log10(1000)
3. sign(-2.9) and sign(32)
4. abs(-27.9) and abs(11.9)

6. Some More R Concepts

You can do some clever and useful things with using the assignment operator "<-":

```r
roomLength <- 7.8
roomWidth <- 6.4
roomArea <- roomLength * roomWidth
roomArea
# [1] 49.92
```

Text objects. You can also assign text to an object.

```r
Greeting <- "Hello World!"
Greeting
# [1] "Hello World!"
```

Vectors. The objects presented so far have all been scalars (single values). Working with vectors is where R shines best as they are the basic building blocks of datasets. To create a vector we can use the c() (combine values into a vector) function.

```r
# A "numeric" vector
x1 <- c(26, 10, 4, 7, 41, 19)
x1
# [1] 26 10 4 7 41 19
# A "character" vector of country names
x2 <- c("Peru", "Italy", "Cuba", "Ghana")
```
There are many other ways to create vectors, for example, `rep()` (replicate elements) and `seq()` (create sequences):

```r
# Repeat vector (2, 6, 7, 4) three times
r1 <- rep(c(2, 6, 7, 4), times=3)
# [1] 2 6 7 4 2 6 7 4 2 6 7 4

# Vector from -2 to 3 incremented by half
s1 <- seq(from=-2, to=3, by=0.5)
# [1] -2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0
```

Vector operations. You can also do calculations on vectors, for example using `x1` from above:

```r
x1 * 2
# [1] 52 20 8 14 82 38
round(sqrt(x1*2.6), 2)
# [1] 8.22 5.10 3.22 4.27 10.32 7.03
```

Missing Values. Missing values are coded as `NA` in R. For example,

```r
x2 <- c(3, -7, NA, 5, 1, 1)
x2
# [1] 3 -7 NA 5 1 1
x3 <- c("Rat", NA, "Mouse", "Hamster")
x3
# [1] "Rat" NA "Mouse" "Hamster"
```

Managing Objects. Use function `ls()` to list the objects in your workspace. The `rm()` function removes (deletes) them.

```r
ls()
# [1] "age" "Age" "AgE"
# [4] "City" "Greeting" "height"
# [7] "Name" "op" "p"
# [10] "r1" "roomArea" "roomLength"
# [13] "roomWidth" "s1" "z1"
# [16] "x2" "x3"
rm(x1, x2, x3, r1, s1, AgE, age)
ls()
# [1] "Age" "City" "Greeting"
# [4] "height" "Name" "op"
# [7] "p" "roomArea" "roomLength"
# [10] "roomWidth"
```

Exercise. Calculate the gross by adding the tax to net amount.

```r
net <- c(108.99, 291.42, 16.28, 62.29, 31.77)
tax <- c(22.89, 17.49, 0.98, 13.08, 6.67)
```

7. R Functions and Packages

R Functions. We have already used some R functions (e.g. `c()`, `mean()`, `rep()`, `sqrt()`, `round()`). Most of the computations in R involves using functions. A function essentially has a name and a list of arguments separated by a comma. Let's have a look at an example:

```r
x2
# [1] "Peru" "Italy" "Cuba" "Ghana"
seq(from = 5, to = 8, by = 0.4)
# [1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8
```

The function name is `seq` and it has three arguments from, to and by. The arguments from and to are the start and end values of a sequence that you want to create, and by is the increment of the sequence. The `seq()` functions has other arguments that you could use which are documented in the help page. For example, we could use the argument `length.out` (instead of by) to fix the length of the sequence as follows:

```r
seq(from = 5, to = 8, length.out = 16)
# [1] 5.0 5.2 5.4 5.6 5.8 6.0 6.2 6.4 6.6 6.8 7.0 7.2 7.4 7.6 7.8 8.0
```

Custom Functions. You can create your own functions (using the `function()` keyword) which is a very powerful way to extend R. Writing your own functions is outside the scope of this guide. As you get more and more familiar with R it is very likely that you will need to learn how to do so but for now you don't need to.

R Packages. You can already do many things with a standard R installation—but it can be extended using contributed packages. Packages are like apps for R. They can contain functions, data and documentation.

Extending Base R. Base R already comes with over two-thousand functions that have been proven to be versatile, reliable and stable. That is no small feat. When it is possible to solve a problem with fewer external dependencies, doing so follows time-honoured best practices. You want to think carefully before adding dependencies.

The tinyverse View. The philosophy of less is more is at the core of the tinyverse\(^1\). Fewer dependencies means a smaller footprint, faster installation, and most importantly fewer nodes in your dependency graph. Experience, as well as empirical and theoretical software engineering practice have demonstrated that failure increases with complexity.

So choosing when to rely on additional packages has to balance the increased functionality a package brings with both its history of development, its development model, maintenance status, and history of both changes and fixes. This is a complex topic, and there are no easy answers. But by adding another package, we always open a door to interface changes we no longer control. The added functionality is clearly valuable at times, yet one has to remain aware of the costs that may accrue as a consequence. So this document takes the view that fewer is better, and will rely on only two additional packages: `data.table`\(^2\) for data wrangling as well as input/output, and `ggplot2`\(^3\) for visualization.

Installation. If needed, install these two packages via the following command which should pick the suitable version for your installation:

```r
install.packages(c("data.table", "ggplot2"))
```

---

1. tinyverse: http://www.tinyverse.org/
2. data.table: http://r-datatype.com
3. ggplot2: https://ggplot2.tidyverse.org
8. Chick Weight Data

R comes with many datasets installed. We will use the ChickWeight dataset to learn about data manipulation. The help system gives a basic summary of the experiment from which the data was collect:

“The body weights of the chicks were measured at birth and every second day thereafter until day 20. They were also measured on day 21. There were four groups of chicks on different protein diets.”

You can get more information, including references by typing:

```
help("ChickWeight")
```

The Data. There are 578 observations (rows) and 4 variables:

- Chick – unique ID for each chick.
- Diet – one of four protein diets.
- Time – number of days since birth.

Note. weight has a lower case w (recall R is case sensitive).

Objective. Investigate the effect of diet on the weight over time.

9. Importing The Data

First we will import the data from a file called ChickWeight.csv using the fread() function from the data.table package which returns a data.table object (whereas the dataset built into R has a different format). The first thing to do, outside of R, is to open the file ChickWeight.csv to check what it contains and that it makes sense. Now we can import the data as follows:

```
suppressMessages(library(data.table))
cw <- fread("ChickWeight.csv")
```

Important Note. If all goes well then the data is now stored in an R object called cw. If you get the following error message then you need to change the working directory to where the data is stored.

Error: 'ChickWeight.csv' does not exist in current working directory ...

Change the working directory in RStudio. From the menu bar select “Session - Set Working Directory - Choose Directory...” then go to the directory where the data is stored. Alternatively, within in R, you could use the function setwd()\(^8\). You can also specify a full path, using - to denote your home directory.

10. Looking at the Dataset

To look at the data type just type the object (dataset) name:

```
cw
```

```
# 576: 48 4 18 261
```

Several base R functions help us inspect the data: str() compactly displays the structure, summary() provides a summary, and head() and tail() display the beginning and end of the data set.

```
summary(cw)
```

```
$ Chick : int [1:578] 18 18 16 16 16 ... 
$ Diet : int [1:578] 1 1 1 1 1 ... 
$ Time : int [1:578] 0 2 0 2 4 ... 
$ weight: int [1:578] 39 35 41 45 49 ... 
```

11. Chick Weight: Data Visualisation

ggplot2 Package. To visualise the chick weight data, we will use the ggplot2 package. Our interest is in seeing how the weight changes over time for the chicks by diet. For the moment don’t worry too much about the details just try to build your own understanding and logic. To learn more try different things even if you get an error messages.

First plot. Let’s plot the weight data (vertical axis) over time (horizontal axis).

```
# (Silently) load the plotting package
suppressMessages(library(ggplot2))
# An empty plot (the plot on the left)
ggplot(cw, aes(Time, weight))
# With data (the plot on the right)
ggplot(cw, aes(Time, weight)) + geom_point()
```
Exercise. Switch the variables Time and weight in code used for the plot on the right? What do you think of this new plot compared to the original?

Add colour for Diet. The graph above does not differentiate between the diets. Let’s use a different colour for each diet.

```
# Adding colour for diet
ggplot(cw, aes(Time, weight, colour=factor(Diet))) + geom_point()
```

Interpretation. It is difficult to conclude anything from this graph as the points are printed on top of one another (with diet 1 underneath and diet 4 at the top).

Factor Variables. Before we continue, we have to make an important change to the cw dataset by making Diet and Time factor variables. This means that R will treat them as categorical variables instead of continuous variables. It will simplify our coding.

```
cw[, Diet := factor(Diet)]
cw[, Time := factor(Time)]
summary(cw) # notice the difference ?
```

Notice that the := operator altered the variable “in-place”, and no explicit assignment was made. This is a key feature of data.table which operated “by reference”: changes are made in reference to one instance of the cw variable, rather than by creating updated copies. We will revisit this := assignment below.

**facet_wrap() function.** To plot each diet separately in a grid using facet_wrap():

```
# Adding jitter to the points
ggplot(cw, aes(Time, weight, colour=Diet)) + geom_point() +
  facet_wrap(~ Diet) +
  theme(legend.position = "bottom")
```

Exercise. To overcome the issue of overlapping points we can jitter the points using geom_jitter(). Replace the geom_point() above with geom_jitter(). What do you observe?

Interpretation. Diet 4 has the least variability but we can’t really say anything about the mean effect of each diet although diet 3 seems to have the highest.

Exercise. For the legend.position try using “top”, “left” and “none”. Do we really need a legend for this plot?

Mean line plot. Next we will plot the mean changes over time for each diet using the stat_summary() function:

```
# mean line plot
ggplot(cw, aes(Time, weight, group=Diet, colour=Diet)) +
  stat_summary(fun.y="mean", geom="line")
```
**Interpretation.** We can see that diet 3 has the highest mean weight gain by the end of the experiment but we don’t have any information about the variation (uncertainty) in the data.

**Exercise.** What happens when you add `geom_point()` to the plot above? Don’t forget the `+`. Does it make a difference if you put it before or after the `stat_summary(...)` line? Hint: Look very carefully at how the graph is plotted.

**Box-whisker plot.** To see variation between the different diets we use `geom_boxplot` to plot a box-whisker plot. A note of caution is that the number of chicks per diet is relatively low to produce this plot.

```r
ggplot(cw, aes(Time, weight, colour=Diet)) + 
  facet_wrap(~ Diet) + 
  geom_boxplot() + 
  theme(legend.position = "none") + 
  ggttitle("Chick Weight over Time by Diet")
```

**Interpretation.** Diet 3 seems to have the highest “average” weight gain but it has more variation than diet 4 which is consistent with our findings so far.

**Exercise.** Add the following information to the above plot:

- x-axis label (use `xlab()`)：“Time (days)”
- y-axis label (use `ylab()`)：“Weight (grams)”

**Final Plot.** Let’s finish with a plot that you might include in a publication.

```r
ggplot(cw, aes(Time, weight, group=Diet, colour=Diet)) + 
  facet_wrap(~ Diet) + 
  geom_jitter() + 
  stat_summary(fun.y="mean", geom="line", 
               colour="black") + 
  theme(legend.position = "none") + 
  ggtitle("Chick Weight over Time by Diet") + 
  xlab("Time (days)") + 
  ylab("Weight (grams)")
```

**12. data.table Data Wrangling Basics**

In this section we will learn how to wrangle (manipulate) datasets using the `data.table` package. Conceptually, `data.table` operations can be viewed as `dt[i, j, by]` with some intentional similarity to SQL. Here `i` can select (or subset) rows, `j` is used to select, summarise or mutate columns, and `by` is the grouping operator. Numerous examples follow.

### j to select (or transform) columns

Add a new variable (column) or modifies an existing one. We already used this above to create factor variables.

```r
cw[, weightKg := weight/1000] # add a column
```

```r
cw[, weightKg := weight/1000] # add a column
```

```r
cw[, weightKg := weight/1000] # add a column
```

```r
cw[, weightKg := weight/1000] # add a column
```

```r
cw[, weightKg := weight/1000] # add a column
```
j to select (or transform) columns. Keeps, drops or reorders variables.

```r
cw[, .(Chick, Time, Diet, weightKg)]
```

j to summarise. It can be used to create aggregations, which is particularly handy with the grouping operator. The following example computes means and standard deviations of the ‘weight’ variable grouped by ‘Diet’.

```r
cw[, .(Mean = mean(weight), SDev = sd(weight)), by = Diet]
```

Exercise. What does the order() do? Try using order(Time) in the i column.

13. Chaining

You may want to do multiple data wrangling steps at once. This is where the ‘chaining’ of data.table operations (i.e., several sets of commands with square brackets) comes to the rescue:

```r
cw[Time %in% c(0, 21)]
cw[, .(N = .N, Mean = mean(weight)), by = .(Diet, Time)]
```

For comparing values in vectors use: < (less than), > (greater than), <= (less than and equal to), >= (greater than and equal to), == (equal to) and != (not equal to). These can be combined logically using & (and) and | (or).

Keying observations. Setting a key changes the order of the observations (rows), and also makes indexing faster.

```r
cw[order(Weight)]
```
by= argument. For each distinct combination of Diet and Time, the chick weight data is summarised into the number of observations (N, using the internal variable .N denoting current group size) and the mean (Mean) of weight.

Other summaries. We can calculate the standard deviation, median, minimum and maximum values—only at days 0 and 21.

cws <- cw[Time %in% c(0,21),
  .(N = .N,
   Mean = mean(weight),
   SDev = sd(weight),
   Median = median(weight),
   Min = min(weight),
   Max = max(weight) ),
  by=.(Diet, Time)]
cws

Finally, we can make the summaries “prettier” for a possible report or publication where we format the numeric values as text.

cws[, Mean_SD := paste0(format(Mean,digits=1),
  " (", format(SDev,digits=2), 
  ")")]
cws[, Range := paste(Min, "-", Max)]
prettySum <- cws[ , .(Diet, Time, N, Mean_SD, Median, Range)]
order(Diet, Time)]
prettySum

Interpretation. This summary table offers the same interpretation as before, namely that diet 3 has the highest mean and median weights at day 21 but a higher variation than group 4. However it should be noted that at day 21, diet 1 lost 4 chicks from 20 that started and diet 4 lost 1 from 10. This could be a sign of some issues (e.g. safety).

Limitations of data. Information on bias reduction measures is not given and is not available either. We don’t know if the chicks were fairly and appropriately randomised to the diets and whether the groups are comparable (e.g., same breed of chicks, sex (gender) balance). Hence we should be very cautious with drawing conclusion and taking actions with this data.

15. Conclusion
This “Getting Started in R” guide introduced you to some of the basic concepts underlying R and used a real life dataset to produce some graphs and summary statistics. It is only a flavour of what R can do but hopefully you have seen some of power of R and its potential.

What next. There are plenty of R courses, books and on-line resources that you can learn from. It is hard to recommend any in particular as it depends on how you learn best. Find things that work for you (paying attention to the quality) and don’t be afraid to make mistakes or ask questions. Most importantly have fun.

16. Acknowledgements
Special thanks to Saghir Bashir for publishing the initial version of *Getting Started with R*, Brodie Gaslam and Matt Dowle for feedback on this version, and to Joshua Ulrich for many discussions about The Tinyverse.

\[9\] (ie Saghir) contacted the source authors and kindly received the following reply: “They were mainly undergraduate projects, final-year, rather than theses, so, unfortunately, it’s unlikely that any record remains, particularly after so many years.”

---

Final Table. Eventually you should be able to produce a publication-ready version such as the following table. Its code uses the kable and kableExtra packages. While the code is not displayed here for compactness, full details are of course in the sources.