Biological Data Science with R

Stephen D. Turner

2018-12-18

Table of contents

Preface

This book was written as a companion to a series of courses introducing the essentials of biological data science with R. While this book was written with the accompanying live instruction in mind, this book can be used as a self-contained self study guide for quickly learning the essentials need to get started with R. The BDSR book and accompanying course introduces methods, tools, and software for reproducibly managing, manipulating, analyzing, and visualizing large-scale biological data using the R statistical computing environment. This book also covers essential statistical analysis, and advanced topics including survival analysis, predictive modeling, forecasting, and text mining.

This is not a *"Tool X"* or *"Software Y"* book. I want you to take away from this book and accompanying course the ability to use an extremely powerful scientific computing environment (R) to do many of the things that you'll do *across study designs and disciplines* – managing, manipulating, visualizing, and analyzing large, sometimes high-dimensional data. Regardless of your specific discipline you'll need the same computational know-how and data literacy to do the same kinds of basic tasks in each. This book might show you how to use specific tools here and there (e.g., DESeq2 for RNA-seq analysis (Love, Huber, and Anders 2014), ggtree for drawing phylogenetic trees (Yu et al. 2017), etc.), but these are not important – you probably won't be using the same specific software or methods 10 years from now, but you'll still use the same underlying data and computational foundation. That is the point of this series – to arm you with a basic foundation, and more importantly, to enable you to figure out how to use *this tool* or *that tool* on your own, when you need to.

This is not a statistics book. There is a short lesson on essential statistics using R in Chapter 7 but this short chapter offers neither a comprehensive background on underlying theory nor in-depth coverage of implementation strategies using R. Some general knowledge of statistics and study design is helpful, but isn't required for going through this book or taking the accompanying course.

There are no prerequisites to this book or the accompanying course. However, each chapter involves lots of hands-on practice coding, and you'll need to download and install required softwar and download required data. See the setup instructions in Appendix A.

Acknowledgements

This book is partially adapted from material we developed for the University of Virginia BIMS8382 graduate course . The material for this course was adapted from and/or inspired by Jenny Bryan's STAT545 course at UBC (Bryan 2019), Software Carpentry (Wilson 2014) and Data Carpentry (Teal et al. 2015) courses, David Robinson's *Variance Explained* blog (Robinson 2015), the ggtree vignettes (Yu 2022) *Tidy Text Mining with R* (Silge and Robinson 2017), and likely many others.

Part I

Core lessons

1 Basics

This chapter introduces the R environment and some of the most basic functionality aspects of R that are used through the remainder of the book. This section assumes little to no experience with statistical computing with R. This chapter introduces the very basic functionality in R, including variables, functions, and importing/inspecting data frames (tibbles).

1.1 RStudio

Let's start by learning about RStudio. **R** is the underlying statistical computing environment. **RStudio** is a graphical integrated development environment (IDE) that makes using R much easier.

- **Options:** First, let's change a few options. We'll only have to do this once. Under *Tools… Global Options…*:
  - Under *General*: Uncheck "Restore most recently opened project at startup"
  - Under *General*: Uncheck "Restore .RData into workspace at startup"
  - Under *General*: Set "Save workspace to .RData on exit:" to Never.
  - Under *General*: Set "Save workspace to .RData on exit:" to Never.
  - Under *R Markdown*: Uncheck "Show output inline for all R Markdown documents"
- Projects: first, start a new project in a new folder somewhere easy to remember. When we start reading in data it'll be important that the *code and the data are in the same place.* Creating a project creates an Rproj file that opens R running *in that folder.* This way, when you want to read in dataset *whatever.txt*, you just tell it the filename rather than a full path. This is critical for reproducibility, and we'll talk about that more later.
- Code that you type into the console is code that R executes. From here forward we will use the editor window to write a script that we can save to a file and run it again whenever we want to. We usually give it a `.R` extension, but it's just a plain text file. If you want to send commands from your editor to the console, use `CMD`+`Enter` (`Ctrl`+`Enter` on Windows).
- Anything after a `#` sign is a comment. Use them liberally to *comment your code.*

1.2 Basic operations

R can be used as a glorified calculator. Try typing this in directly into the console. Make sure you're typing into into the editor, not the console, and save your script. Use the run button, or press `CMD`+`Enter` (`Ctrl`+`Enter` on Windows).

```
2+2
5*4
2^3
```

R Knows order of operations and scientific notation.

```
2+3*4/(5+3)*15/2^2+3*4^2
5e4
```

However, to do useful and interesting things, we need to assign *values* to *objects*. To create objects, we need to give it a name followed by the assignment operator `<-` and the value we want to give it:

```
weight_kg <- 55
```

`<-` is the assignment operator. Assigns values on the right to objects on the left, it is like an arrow that points from the value to the object. Mostly similar to `=` but not always. Learn to use `<-` as it is good programming practice. Using `=` in place of `<-` can lead to issues down the line. The keyboard shortcut for inserting the `<-` operator is `Alt-dash`.

Objects can be given any name such as `x`, `current_temperature`, or `subject_id`. You want your object names to be explicit and not too long. They cannot start with a number (`2x` is not valid but `x2` is). R is case sensitive (e.g., `weight_kg` is different from `Weight_kg`). There are some names that cannot be used because they represent the names of fundamental functions in R (e.g., `if`, `else`, `for`, see here for a complete list). In general, even if it's allowed, it's best to not use other function names, which we'll get into shortly (e.g., `c`, `T`, `mean`, `data`, `df`, `weights`). In doubt check the help to see if the name is already in use. It's also best to avoid dots (`.`) within a variable name as in `my.dataset`. It is also recommended to use nouns for variable names, and verbs for function names.

When assigning a value to an object, R does not print anything. You can force to print the value by typing the name:

```
weight_kg
```

Now that R has `weight_kg` in memory, we can do arithmetic with it. For instance, we may want to convert this weight in pounds (weight in pounds is 2.2 times the weight in kg).

```
2.2 * weight_kg
```

We can also change a variable's value by assigning it a new one:

```
weight_kg <- 57.5
2.2 * weight_kg
```

This means that assigning a value to one variable does not change the values of other variables. For example, let's store the animal's weight in pounds in a variable.

```
weight_lb <- 2.2 * weight_kg
```

and then change `weight_kg` to 100.

```
weight_kg <- 100
```

What do you think is the current content of the object `weight_lb`? 126.5 or 220?

You can see what objects (variables) are stored by viewing the Environment tab in Rstudio. You can also use the `ls()` function. You can remove objects (variables) with the `rm()` function. You can do this one at a time or remove several objects at once. You can also use the little broom button in your environment pane to remove everything from your environment.

```
ls()
rm(weight_lb, weight_kg)
ls()
weight_lb # oops! you should get an error because weight_lb no longer exists!
```

Exercise 1

What are the values after each statement in the following?

```
mass <- 50              # mass?
age  <- 30              # age?
mass <- mass * 2        # mass?
age  <- age - 10        # age?
mass_index <- mass/age  # massIndex?
```

1.3 Functions

R has built-in functions.

```
# Notice that this is a comment.
# Anything behind a # is "commented out" and is not run.
sqrt(144)
log(1000)
```

Get help by typing a question mark in front of the function's name, or `help(functionname)`:

```
help(log)
?log
```

Note syntax highlighting when typing this into the editor. Also note how we pass *arguments* to functions. The `base=` part inside the parentheses is called an argument, and most functions use arguments. Arguments modify the behavior of the function. Functions some input (e.g., some data, an object) and other options to change what the function will return, or how to treat the data provided. Finally, see how you can *next* one function inside of another (here taking the square root of the log-base-10 of 1000).

```
log(1000)
log(1000, base=10)
log(1000, 10)
sqrt(log(1000, base=10))
```

> Exercise 2
>
> See `?abs` and calculate the square root of the log-base-10 of the absolute value of `-4*(2550-50)`. Answer should be `2`.

1.4 Tibbles (data frames)

There are *lots* of different basic data structures in R. If you take any kind of longer introduction to R you'll probably learn about arrays, lists, matrices, etc. We are going to skip straight to the data structure you'll probably use most – the **tibble** (also known as the data frame). We use tibbles to store heterogeneous tabular data in R: tabular, meaning that individuals or observations are typically represented in rows, while variables or features are represented as columns; heterogeneous, meaning that columns/features/variables can be different classes (on variable, e.g. age, can be numeric, while another, e.g., cause of death, can be text).

We'll learn more about tibbles in Chapter 2.

2 Tibbles

There are *lots* of different basic data structures in R. If you take any kind of longer introduction to R you'll probably learn about arrays, lists, matrices, etc. Let's skip straight to the data structure you'll probably use most – the **data frame**. We use data frames to store heterogeneous tabular data in R: tabular, meaning that individuals or observations are typically represented in rows, while variables or features are represented as columns; heterogeneous, meaning that columns/features/variables can be different classes (on variable, e.g. age, can be numeric, while another, e.g., cause of death, can be text).

This lesson assumes a basic familiarity with R.

Recommended reading: Review the *Introduction* (10.1) and *Tibbles vs. data.frame* (10.3) sections of the ***R for Data Science* book**. We will initially be using the `read_*` functions from the **readr** package. These functions load data into a *tibble* instead of R's traditional data.frame. Tibbles are data frames, but they tweak some older behaviors to make life a little easier. These sections explain the few key small differences between traditional data.frames and tibbles.

2.1 Our data

The data we're going to look at is cleaned up version of a gene expression dataset from Brauer et al. Coordination of Growth Rate, Cell Cycle, Stress Response, and Metabolic Activity in Yeast (2008) *Mol Biol Cell* 19:352-367. This data is from a gene expression microarray, and in this paper the authors are examining the relationship between growth rate and gene expression in yeast cultures limited by one of six different nutrients (glucose, leucine, ammonium, sulfate, phosphate, uracil). If you give yeast a rich media loaded with nutrients except restrict the supply of a *single* nutrient, you can control the growth rate to any rate you choose. By starving yeast of specific nutrients you can find genes that:

1. **Raise or lower their expression in response to growth rate**. Growth-rate dependent expression patterns can tell us a lot about cell cycle control, and how the cell responds to stress. The authors found that expression of >25% of all yeast genes is linearly correlated with growth rate, independent of the limiting nutrient. They also found that the subset of negatively growth-correlated genes is enriched for peroxisomal functions, and positively correlated genes mainly encode ribosomal functions.

2. **Respond differently when different nutrients are being limited**. If you see particular genes that respond very differently when a nutrient is sharply restricted, these genes might be involved in the transport or metabolism of that specific nutrient.

You can download the cleaned up version of the data at the link above. The file is called **brauer2007_tidy.csv**. Later on we'll actually start with the original raw data (minimally processed) and manipulate it so that we can make it more amenable for analysis.

2.2 Reading in data

2.2.1 dplyr and readr

There are some built-in functions for reading in data in text files. These functions are *read-dot-something* – for example, `read.csv()` reads in comma-delimited text data; `read.delim()` reads in tab-delimited text, etc. We're going to read in data a little bit differently here using the readr package. When you load the readr package, you'll have access to very similar looking functions, named *read-underscore-something* – e.g., `read_csv()`. You have to have the readr package installed to access these functions. Compared to the base functions, they're *much* faster, they're good at guessing the types of data in the columns, they don't do some of the other silly things that the base functions do. We're going to use another package later on called dplyr, and if you have the dplyr package loaded as well, and you read in the data with readr, the data will display nicely.

First let's load those packages.

```
library(readr)
library(dplyr)
```

If you see a warning that looks like this: `Error in library(packageName) : there is no package called 'packageName'`, then you don't have the package installed correctly. See the setup page.

2.2.2 `read_csv()`

Now, let's actually load the data. You can get help for the import function with `?read_csv`. When we load data we assign it to a variable just like any other, and we can choose a name for that data. Since we're going to be referring to this data a lot, let's give it a short easy name to type. I'm going to call it `ydat`. Once we've loaded it we can type the name of the object itself (`ydat`) to see it printed to the screen.

```
ydat <- read_csv(file="data/brauer2007_tidy.csv")
ydat
```

Take a look at that output. The nice thing about loading dplyr and reading in data with readr is that data frames are displayed in a much more friendly way. This dataset has nearly 200,000 rows and 7 columns. When you import data this way and try to display the object in the console, instead of trying to display all 200,000 rows, you'll only see about 10 by default. Also, if you have so many columns that the data would wrap off the edge of your screen, those columns will not be displayed, but you'll see at the bottom of the output which, if any, columns were hidden from view. If you want to see the whole dataset, there are two ways to do this. First, you can click on the name of the data.frame in the **Environment** panel in RStudio. Or you could use the `View()` function (*with a capital V*).

```
View(ydat)
```

2.3 Inspecting data.frame objects

2.3.1 Built-in functions

There are several built-in functions that are useful for working with data frames.

- Content:
  - `head()`: shows the first few rows
  - `tail()`: shows the last few rows
- Size:
  - `dim()`: returns a 2-element vector with the number of rows in the first element, and the number of columns as the second element (the dimensions of the object)
  - `nrow()`: returns the number of rows
  - `ncol()`: returns the number of columns
- Summary:
  - `colnames()` (or just `names()`): returns the column names
  - `str()`: structure of the object and information about the class, length and content of each column
  - `summary()`: works differently depending on what kind of object you pass to it. Passing a data frame to the `summary()` function prints out useful summary statistics about numeric column (min, max, median, mean, etc.)

```
head(ydat)
tail(ydat)
dim(ydat)
names(ydat)
str(ydat)
summary(ydat)
```

2.3.2 Other packages

The `glimpse()` function is available once you load the **dplyr** library, and it's like `str()` but its display is a little bit better.

```
glimpse(ydat)
```

The **skimr** package has a nice function, skim, that provides summary statistics the user can skim quickly to understand your data. You can install it with `install.packages("skimr")` if you don't have it already.

```
library(skimr)
skim(ydat)
```

2.4 Accessing variables & subsetting data frames

We can access individual variables within a data frame using the `$` operator, e.g., `mydataframe$specificVariable`. Let's print out all the gene names in the data. Then let's calculate the average expression across all conditions, all genes (using the built-in `mean()` function).

```
# display all gene symbols
ydat$symbol

#mean expression
mean(ydat$expression)
```

Now that's not too interesting. This is the average gene expression across all genes, across all conditions. The data is actually scaled/centered around zero:

We might be interested in the average expression of genes with a particular biological function, and how that changes over different growth rates restricted by particular nutrients. This is the kind of thing we're going to do in the next section.

EXERCISE 1

1. What's the standard deviation expression (hint: get help on the `sd` function with `?sd`).
2. What's the range of rate represented in the data? (hint: `range()`).

2.5 BONUS: Preview to advanced manipulation

What if we wanted show the mean expression, standard deviation, and correlation between growth rate and expression, separately for each limiting nutrient, separately for each gene, for all genes involved in the leucine biosynthesis pathway?

```
ydat |>
  filter(bp=="leucine biosynthesis") |>
  group_by(nutrient, symbol) |>
  summarize(mean=mean(expression), sd=sd(expression), r=cor(rate, expression))
```

Neat eh? We'll learn how to do that in the advanced manipulation with dplyr lesson.

3 Data Manipulation

Not much to see here...

4 Data Visualization

Not much to see here...

5 Tidy EDA

Not much to see here...

6 R Markdown

Not much to see here...

Part II

Electives

7 Essential Statistics

Not much to see here...

8 Survival Analysis

Not much to see here...

9 Predictive Modeling

Not much to see here...

10 Probabilistic Forecasting

Not much to see here...

11 Text Mining

Not much to see here...

12 Phylogenetic Trees

Not much to see here...

13 RNA-seq

Not much to see here...

Summary

In summary, this book has no content whatsoever.

References

Bryan, Jennifer. 2019. "STAT 545: Data Wrangling, Exploration, and Analysis with r." https://stat545.com/.

Love, Michael I., Wolfgang Huber, and Simon Anders. 2014. "Moderated Estimation of Fold Change and Dispersion for RNA-seq Data with DESeq2." *Genome Biology* 15 (12): 1–21.

Robinson, David. 2015. "Variance Explained." http://varianceexplained.org/.

Silge, Julia, and David Robinson. 2017. *Text Mining with R: A Tidy Approach*. 1st edition. Beijing ; Boston: O'Reilly Media.

Teal, Tracy K., Karen A. Cranston, Hilmar Lapp, Ethan White, Greg Wilson, Karthik Ram, and Aleksandra Pawlik. 2015. "Data Carpentry: Workshops to Increase Data Literacy for Researchers."

Wilson, Greg. 2014. "Software Carpentry: Lessons Learned." *F1000Research* 3.

Yu, Guangchuang. 2022. "Ggtree: An r Package for Visualization of Tree and Annotation Data." http://bioconductor.org/packages/ggtree/.

Yu, Guangchuang, David K. Smith, Huachen Zhu, Yi Guan, and Tommy Tsan-Yuk Lam. 2017. "Ggtree: An R Package for Visualization and Annotation of Phylogenetic Trees with Their Covariates and Other Associated Data." *Methods in Ecology and Evolution* 8 (1): 28–36.

A Setup

A.1 Software

A.2 Data

1. **Option 1: Download all the data**. Download and extract **this zip file** (NA Mb) with all the data for the entire workshop. This may include additional datasets that we won't use here.
2. **Option 2: Download individual datasets as needed.**
    - Create a new folder somewhere on your computer that's easy to get to (e.g., your Desktop). Name it `bds`. Inside that folder, make a folder called `data`, all lowercase.
    - Download individual data files as needed, saving them to the new `bdsr/data` folder you just made. Click to download. If data displays in your browser, right-click and select *Save link as…* (or similar) to save to the desired location.

- data/airway_metadata.csv
- data/airway_scaledcounts.csv
- data/annotables_grch38.csv
- data/austen.csv
- data/brauer2007_messy.csv
- data/brauer2007_sysname2go.csv
- data/brauer2007_tidy.csv
- data/dmd.csv
- data/flu_genotype.csv
- data/gapminder.csv
- data/grads_dd.csv
- data/grads.csv
- data/h7n9_analysisready.csv
- data/h7n9.csv
- data/heartrate2dose.csv
- data/ilinet.csv
- data/movies_dd.csv
- data/movies_imdb.csv
- data/movies.csv

- data/nhanes_dd.csv
- data/nhanes.csv
- data/SRP026387_metadata.csv
- data/SRP026387_scaledcounts.csv
- data/stressEcho.csv

B Additional Resources

Not much to see here...