OXFORD

APPLIED STATISTICS WITH R

A PRACTICAL GUIDE FOR THE LIFE SCIENCES

JUSTIN C. TOUCHON

Applied Statistics with R

Applied Statistics with R

A Practical Guide for the Life Sciences

JUSTIN C. TOUCHON

Department of Biology, Vassar College, USA

OXFORD
UNIVERSITY PRESS

OXFORD
UNIVERSITY PRESS

Great Clarendon Street, Oxford, OX2 6DP,
United Kingdom

Oxford University Press is a department of the University of Oxford.
It furthers the University's objective of excellence in research, scholarship,
and education by publishing worldwide. Oxford is a registered trade mark of
Oxford University Press in the UK and in certain other countries

First Edition published in 2021
Reprinted 2023 (with corrections)

Published in the United States of America by Oxford University Press
198 Madison Avenue, New York, NY 10016, United States of America

British Library Cataloguing in Publication Data
Data available

Library of Congress Control Number: 2021934831

ISBN 978–0–19–886997–9 (hbk.)
ISBN 978–0–19–886933–7 (pbk.)

DOI: 10.1093/oso/9780198869979.001.0001

Printed and bound by
CPI Group (UK) Ltd, Croydon, CR0 4YY

For Myra

Preface

Welcome!

The statistical analyses that life-scientists are being expected to perform are increasingly advanced and yet most graduate programs in the United States do not even offer a statistics course that teaches beyond Analysis of Variance (ANOVA) and linear regression. Undergraduate and graduate students are thus rarely provided with the opportunity to learn the types of analyses they need to know in order to publish and compete on the job market, much less simply analyze their data appropriately. Part of the reason for this is that the way statistics are traditionally taught can be frustratingly slow and tedious. When I was a graduate student, I remember excitedly enrolling in a statistics class with the hope of learning how to analyze the data I was collecting each summer in the field. Unfortunately, we spent the entire semester learning how to perform an analysis of variance and a linear regression, by hand. There has to be a better way!

This book is written with the belief that a comprehensive understanding of practical data analyses is not as daunting as it might seem. I have been teaching an annual statistics workshop at the Smithsonian Tropical Research Institute for more than 10 years and I know that my approach works. My teaching perspective is rooted in the idea that instead of spending time mired in statistical theory and learning data analysis by hand, the most important thing to understand is what kind of data you have. Once you know your data, you can then figure out how to analyze them

effectively. Whether at the undergraduate, graduate, or post-graduate level, this book will provide the tools needed to properly analyze your data in an efficient, accessible, plainspoken, frank, and (hopefully) humorous manner, ensuring that readers come away with the knowledge of which analyses they should use and when they should use them.

This book uses the statistical language R, which is the choice of ecologists worldwide and is rapidly becoming the "go-to" stats program throughout the life sciences. The examples in the book are rooted in a single, real dataset (published in the journal *Ecology* in 2013) and use actual analyses that I have conducted in my professional career as an ecologist. The dataset is admittedly somewhat messy, and early chapters are designed so that students "clean" the raw data as a way of learning basic data manipulation skills and building good habits. Moreover, using a single relatively large dataset (~2500 observations) allows students to get a good understanding of what they are analyzing from chapter to chapter, instead of jumping from one small pre-cleaned dataset to another throughout the book. It also allows readers to see how they can view the same data through different lenses and allows an easy and natural progression from linear and generalized linear models to mixed effects versions of those same analyses, given the hierarchically nested design of the example experiment.

Goals for the book

It is my sincere hope that you find this book useful and instructive. I have tried my hardest to distill down everything I know and think about data analysis into these pages. You will undoubtedly find that some of what I suggest may differ from what you read elsewhere, either on the web or in other books. Just about everyone these days happens to be rather opinionated, and statisticians and R users are certaintly no different. Wherever possible, I have tried to include the rationale behind my thinking.

Since you are reading this book, you evidently want to learn about data analysis. I applaud your initiative and to hope to reward you by teaching you how to do just that, efficiently and effectively. Here are the goals of this book.

- I hope to build your familiarity with R from the ground up via the chapters and assignments. Even if you have some experience with R, you will likely learn new ways to approach your data. If you are relatively new to R, I hope the hands on experience of typing along with the instructions will help you overcome "fear of the R prompt."
- I want to empower you to not only follow instructions carefully and analyze the data presented in these chapters, but hopefully to be able to analyze your own data and to think critically about data when you see them presented in research and in the public realm. As you may already know, science literacy is seriously lacking in the public sphere and increasing the number of people who can think critically about data presented in the news or elsewhere is extremly important.
- Lastly, I hope you can become a part of the global R community. R is so big there is no single repository of information about it nor is there a single manual that contains all the possible instructions you might need to execute. Thus, in addition to books like this one, you will need to become familiar with using the web to find answers to questions. I will provide examples in the later chapters of how you might seek out information to help yourself when (not *if*, mind you, but *when*) you get stuck or encounter an error.

Basic layout of the book

The materials presented in these chapters are set up as follows. There are ten topics, each an explanatory chapter which will allow you to teach yourself the code. I cannot stress enough that you really do want to type things in and you need to think about what the code means and what it is doing if you want to learn this stuff. If you have an electronic copy of the book, avoid any temptation to cut and paste. If you are reading this, you are interested in learning R, right? Trust me, if you cut and paste code you *will not* learn as well as if you type it in by hand.

The ten topics are:

1 Introduction to R
2 Before You Begin (aka Thoughts on Proper Data Analysis)
3 Exploratory Data Analysis
4 The Basics of Plotting
5 Basic Statistical Analyses using R
6 More Linear Models!
7 Generalized Linear Models
8 Linear and Generalized Linear Mixed Effects Models
9 Data Wrangling and Advanced Plotting with the ***tidyverse***
10 Writing Loops and Functions in R

Just a note about how each of the chapters will be formatted. Bits of code that you can/should type in are displayed in light grey boxes, and the output from that code is generally displayed directly below it. For example, check out the code below. What is shown in the grey box "*(2+2)*" is what you would type at the R prompt, and the bit of code below it is the output from executing that command.

```
2+2
```

```
## [1] 4
```

In general, if you type in exactly what is in the grey boxes you will get what is shown after it! Amazing, I know. Your mind is already blown, right?

The code that will be presented in this book is often written in a relatively "long" format in order to make it more readable. This might not exactly be how you type it to your computer though, which is perfectly fine.

At the end of each chapter is a short set of assignments to give you the opportunity to practice what you have just learned. You can find solutions to the assigments at the GitHub page for the book (https://github.com/jtouchon/Applied-Statistics-with-R) as well as other important information. Since R is an open source language it is likely that some of the code

needed to run the examples in this book may change over time, and I will post code updates on that site.

A little background about R

R is a statistical programming package and a powerful graphics engine. R is considered to be a dialect of the S and S+ language that was created by AT&T Bell Labs. S is commercially available while R is open source and freely available through the Comprehensive R Archive Network: (https://cran.r-project.org). R has many advantages besides being freely available. For example, a user might program loops to conduct many repetitive statistical analyses or simulate thousands of data sets with known parameters. In addition, in the fields of Ecology and Evolutionary Biology at least, R is now by far the most commonly used statistical program (see Touchon and McCoy 2016 *Ecosphere*). There is substantial evidence that similar shifts are occurring in Psychology and Neuroscience as well.

A little about how R works

Because R creates objects from analyses that are stored in its memory, new users often are surprised by the fact that the results of their analyses are not immediately displayed on the screen. When you run something successfully, all you generally see is the prompt, which is denoted by the '>' sign.

There are several reasons for this. First, R does exactly what you tell it to do. Thus, if you tell it to run an ANOVA and store that output as an object, it does that, but you have to tell it a separate function to show you the object you created. Second, printing stuff on the screen takes time and computer power. By not showing everything that is going on, R is being very effcient. For example, if you wanted to do 100 regressions on different data sets, R can do this without opening 100 separate windows. One can store only the regression coefficients and display all of them in a single line for comparision. It is this flexibility that makes R a fantastic statistical program. Also, it's free. Did I mention that it is free yet?

This book provides an introduction to using R in data analyses with practical examples designed to be readily accessible to all life scientists. Although the example dataset I will use is ecological in nature, the parallels will hopefully be easy to see with other disciplines. A more explicit discussion of this is at the end of Chapter 2. R is also a very powerful graphing tool and I will get you started on your way to making publication quality figures.

This book is not a comprehensive overview of all available statistical approaches and methods or experimental design. No single book could do that. I will of course touch on many different topics, but there are over 16,000 packages available to use in R (as of July 2020), a number which is growing by the day, so such an overview is impossible.

Learning R is like learning any language. At times it will be diffcult and frustrating, but it is worth it and if you stick with it you will have breakthroughs that feel amazing (I call these "R-gasms"). Over time, you may grow to love working in R!

There is a quote I love from the musician, actor, author, poet, and all around amazing human Henry Rollins, which encapsulates a lot of how I think about doing statistical analyses and using R.

> *Numbers are perfect, infallible and everlasting. You aren't. Numbers are always right in the end. You may see an incorrect figure, but that's not the fault of the number, the fault lies in the person doing the calculating.*
>
> –Henry Rollins, *High Adventure in the Great Outdoors*

Why do I like that quote so much? It's because when you get an error in R, it is almost certainly your fault. R didn't mess up, you did. Sorry, but that's the honest truth. So check your code! :)

Why learn R?

You might be thinking to yourself "Why do I need to learn R?" or "Seriously, I have to type everything in by hand?!" or "Can't I do this easier in another program?" There are many answers to these questions.

- If you are an undergraduate thinking of going to graduate school, it is useful for you to learn R because you will almost certainly use R as a graduate student. Thus, you will have a leg up on everyone else! Get started now and be the best you can be.
- Yes, you have to type everything in, but that also helps you learn what you are doing. It is very easy to click some buttons and get an answer that you don't really understand. If you have to type in the code for the statistics you are doing, you will have a better understanding of what you are doing.
- Having some basic familiarity with "coding" is increasingly useful across a variety of disciplines. You don't need to be a pro, but being comfortable with a computer and with typing code to achieve a result is very useful.
- Because it is free and extremely powerful, R is the only statistics program you will ever really need to know. If you go on to graduate

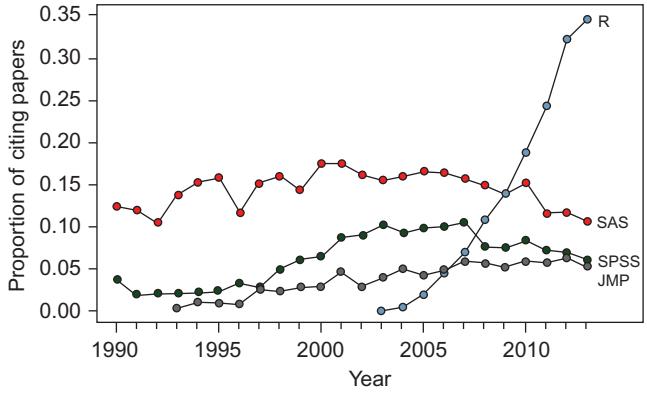

Figure 0.1 This figure, from Touchon and McCoy (2016), demonstrates the rise in usage of R as compared to SAS, SPSS, and JMP, in the field of ecology. R really is the go-to program, so it is in your best interest to learn it. Touchon, J.C. and McCoy, M.W. (2016). "The mismatch between current statistical practice and doctoral training in ecology." *Ecosphere*. 7(8):e01394. Reproduced under Creative Commons Attribution License (CC-BY)

school or into consulting or any field that deals with data, you will be able to use R. This book will teach you many of the basics you will need to know in R, but one of the best things about R is that it can be expanded to accomplish nearly any statistical (or, more generally, data analytic) needs you might have. The same cannot be said with other programs like JMP, SPSS, or SAS, which are very expensive and may not be available to you at another institution. Check out Figure 0.1 for evidence that R has become the program of choice (at least in Ecology, but the same is true in other fields as well).

Okay, shall we get started?

Acknowledgments

This book owes a tremendous debt to many people. First and foremost, thank you to Andy Jones and Stuart Dennis. The three of us took a germ of an idea—a desire to teach folks the practical tools they would need to analyze their data in R—and created the initial workshop at the Smithsonian Tropical Research Institute (STRI) that this material evolved from. Thank you to Owen McMillan, Adriana Bilgray, and Paola Gomez at STRI for their continued support of me and my desire to teach people how to use R. More generally, thank you to the amazing community of scientists at STRI for providing such an incredible environment to learn and conduct research. Many thanks to James Vonesh and Mike McCoy, two invaluable mentors, colleagues, and friends over the years. Your knowledge of R certainly eclipses mine, and I hope I've done justice to all that you have taught me. Thank you to my doctoral and post-doctoral advisor Karen Warkentin. Karen and James wrote the National Science Foundation grant that generated the data used throughout this book. Many thanks to Tim Thurman for opening my eyes to the world of ***ggplot2*** and ***dplyr***. Thank you to the hundreds of interns, undergraduate, and graduate students, postdocs, and professional scientists that I have had the pleasure of teaching over the past decade or so. The lessons in this book have been continually refined and improved based on your feedback, so thank you for making me a better teacher. In particular, thank you to the students in my 2020

Applied Biostatistics class at Vassar College for the countless typos they found in early drafts of these chapters. Lastly, thank you to my wife Myra Hughey for her patience, support, and editorial advice over the years. You are the best partner in research and life I could ever hope for.

Contents

Introduction to R

1 Introduction to R

1.1 OVERVIEW

The purpose of this first chapter is to introduce you to the basic workings of R and get you up to speed. Some of this material might be familiar to you if you've used R before, but the goal is to get anyone reading the book up to a basic level of familiarity. You will learn many of the basic and very important functions of R, such as:

- Creating objects
- Writing articulate R code
- Using functions
- Generating artificial data
- Entering data in a format that can be read and analyzed by R

This chapter does not intend to be an exhaustive introduction to all the basic workings of R. In other words, we'll move pretty quickly here. If you would like a greater introduction, I highly recommend checking out the

Applied Statistics with R: A Practical Guide for the Life Sciences. Justin C. Touchon, Oxford University Press (2021). © Justin C. Touchon. DOI: 10.1093/oso/9780198869979.003.0001

excellent book *Getting Started With R: An Introduction for Biologists* by Andrew Beckerman, Dylan Childs, and Owen Petchey.

1.2 GETTING STARTED

1.2.1 Obtaining R

If you are brand new to this, you will have to download R in order to do anything. Just navigate your web browser of choice to http://cran.r-project.org to download the appropriate version of R for your operating system. There is another program you may have heard of and may want to use called RStudio, which can be found at http://www.rstudio.com.

Box 1.1 - RStudio

Please remember this: RStudio is a program that uses R. It helps keep things organized and has some nice autocomplete functions, but R is the actual program that does *everything* that we will cover in this book. RStudio has plenty of great features, don't get me wrong. It's really great for writing in RMarkdown and LaTeX, if you choose to do that. But, like I said, RStudio is a program that uses R. R does all the heavy lifting. R is the statistics program. Personally, I use regular plain old R and not RStudio. To each their own …

1.2.2 Installing and loading packages

R is designed to be a small program (currently just about 80 mb) which makes it easy to download and install anywhere in the world. The base version of R contains a great number of functions for organizing and analyzing data, but the real strength comes in what are called ***packages***. Packages are freely downloadable additions to R that provide new functions and datasets for particular analyses. For example, the base version of R can conduct linear models and generalized linear models (Chapters 5 – 7) but cannot conduct mixed effects models (Chapter 8). To do mixed effects models, you need to download a specific package (of which there are several).

The only important thing to remember about packages is that adding them to R is a two-step process. First, you have to **install** a package, which (perhaps counterintuitively) just downloads the package to your computer.

Secondly, you have to **load** the package, which is when you have actively placed it in the current memory for use. You will generally obtain packages from the Comprehensive R Archive Network (https://cran.r-project.org/) (CRAN) directly through R.

Box 1.2 - Install some packages

Assuming you have installed R on your computer, you should run the following code to install the various packages you will need to have in order to execute the commands presented throughout this book. If you are using RStudio you can click on the packages tab and search for these one at time by using the little search window. Make sure to click the button to "Install Dependencies." We won't do anything with these right now, but they will be necessary later in the book.

```
install.packages(c("lme4","multcomp","car","ggplot2","gplots",
                   "MASS","tidyr","dplyr","broom","gridExtra",
                   "cowplot","emmeans","glmmTMB","lattice"),
                 dependencies = T, repos =
                   "http://cran.us.r-project.org")
```

1.3 WORKING FROM THE SCRIPT WINDOW

The biggest mistake that most new R users make is to just type commands into the command prompt. The problem with this is that once you hit enter the command is gone. If you hit the up-arrow, R will scroll through the previously executed commands, but aside from this what you typed is gone and *it cannot be edited*! It is of course reasonable to run lines from the command line from time to time, but it is much better to work from a script window.

The script window allows you to easily save and edit your code, and to execute one or multiple lines of code at once. To open a blank script window, go to the File menu and click on New Document, or just hit command-N (Mac) or control-N (PC) on your keyboard.

In the script window you can type in your commands and then execute them by hitting command-enter (Mac) or control-R (PC). This means you type code into the script window and then the program sends the line of

code to the command prompt for you. Do not cut and paste code from the script window to the command prompt; that is a waste of time. You can also highlight multiple lines of code and execute them all at once. To save your code simply go to the File menu and save as you would any other file (or just hit command-S or control-S on your keyboard).

A script allows you to edit, run, and tweak your code, save it, return to it later, send it collaborators or mentors, and so on. Anything you think will want to run more than once, or that you might want to edit, should be typed into a script window (which is pretty much everything).

1.4 CREATING WELL-DOCUMENTED AND ANNOTATED CODE

One of the most important things you can do is write orderly, well-annotated code that not only functions well but explains what is happening and why it is happening and does so in easy to read and understand language. This idea was first introduced by computer scientist Donald Knuth and is known as "literate programming." Literate programming is the process of interspersing your computer code, in this case R code, with plain-language descriptions of what the code is doing. This allows a reader to have a fully formed idea of what is going on. In R, you do this with annotation, which is simply the process of leaving notes within the code that are not actually code themselves. It's like you are Hansel and Gretel getting dragged into the woods: you want to leave plenty of clues for your future self (or others) to be able to discern the trail you took.

Box 1.3 - Write good code, for yourself and others

For any bit of R code you write, you should consider that you are writing for three audiences:

1) Your future self
2) Your collaborators
3) Everyone else that might look at your code one day

1. Writing for yourself

Seldom (never?) will you have the opportunity to sit down with a dataset and analyze it start to finish in a single sitting. It is rare that you even will have the opportunity to work on it on consecutive days where what you did yesterday is still fresh in your mind today. What is more realistic is that you work on something for some period of time (hours, days, maybe even weeks if you are really lucky!) then have to put it down for some time because you are distracted by other tasks (teaching, other research demands, manuscript revisions, parenting, a pandemic, etc.). By the time you come back to your code even a week later, you will likely have to invest some substantial time getting back to where you were. Writing good, clear code will reduce that restart time considerably.

2. Writing for collaborators

If you are, or are planning to be, a professional scientist, you are unlikely to work exclusively by yourself. There will be times when you collaborate with others. Maybe it's your graduate advisor, maybe a colleague at another institution. Whatever the scenario, it means you might be responsible for analyzing or organizing some set of data, then sharing it with others. If that's the case, you want to make sure when you send your code it is clear what you did and why you did it. Imagine the embarrassment of your collaborator sending you question after question trying to figure out what your code means!

3. Writing for folks in the future who might want to see your code

Increasingly, it is necessary to post both the data that go into a scientific article and also the code that was used to analyze it. This is a tremendous step towards increasing transparency in science and is to be applauded for sure. But it also means that some stranger might look at your code a month or a year or more down the road, even after you thought you were long done with it all. Thus, just like writing for your future self or your collaborators,

you want to make sure that your code is clean and organized and well-annotated.

Box 1.4 - How do you create good code?

There are a few basic rules of thumb that you should do your best to adhere to. These will help ensure you stay organized and help prevent embarrassment when you inevitably have to show your script file to someone else! It will also just make your life in R much, much easier.

1. Always start with your raw data
2. Annotate, annotate, annotate
3. Organize your script file into logical sections
4. Give your objects meaningful and unique names

1. Always start with your raw data

Import your raw file from Excel or whatever program you use for data entry but do whatever data processing is necessary (removing outliers, calculating new variables from old ones, etc.) within an R script. This way, you have a record that others can follow that leads them from the raw data to the final product. This also increases transparency in science: if you do a bunch of data filtration before you import your data, you are omitting potentially important information from the scientific record.

2. Annotate, annotate, annotate

Leave lots of notes to yourself or others about what certain bits of code are doing and the rationale behind them. You annotate your code by using a number sign, also called a pound sign (or hashtag, as the kids might say). Whatever you call it, it's this thing: #. Anything that is written after a # on a given line won't be executed. For example, consider the following two bits of code (feel free to type this into your own R console and see for yourself what happens).

```
Howdy
```

```
## Error in eval(expr, envir, enclos): object 'Howdy' not found
```

Note that we got an error because R doesn't know what this word is supposed to mean. R assumes that "*Howdy*" must be an object in the memory, but it is not there, so it gives us an error. Now see what happens if we put a "#" in front of the word.

```
#Howdy
```

What happened? Nothing, absolutely nothing! The line was not executed because it came after a #. This simple coding trick allows you to leave yourself descriptive notes in your code that won't interrupt the executable code. Throughout this book, you will see notes embedded in the chunks of code that help describe what is going on.

3. Organize your script file into logical sections

I like to use long strings of #s to create large visible breaks in my script files that I can see when I'm quickly scrolling through it. At the top of the script file, I will put a header, each line starting of course with #s, that describes the basics of what the file is, where the data are from, the date I started working on it, that sort of thing. Then, I create a section where I load all the packages I will need for that script file. You can use a # and title it something like "*#Packages to load*." Then, create whatever other sections you might need for your purposes. These might include sections for analyzing different aspects of the data, or running different types of analyses, or to split apart analyses vs figure making.

Box 1.5 - An example of a script header

I like to give my code a nice big clear header at the top which describes what that particular file is for. I also like to put a block of code that loads whatever packages I will need for that particular set of analyses. Here is an example of what that might look like.

```
########################################
###Code for analyzing RxP experiment data
###Date created: Nov. 15, 2012

###Load the packages
library(dplyr)
library(ggplot2)
library(lme4)
library(emmeans)
library(MASS)
```

4. Give your objects meaningful and unique names

As my good friend Stu Dennis used to say, "You wouldn't call your dog 'dog,' so don't call your data 'data.' " I couldn't agree more. Make sure when you are importing your dataset, creating objects, running and saving models, etc., that you assign useful and meaningful names. The flip side of this is that you generally want those names to be short, since typing in a lengthy object name over and over gets a little laborious. The same logic applies to the names of variables in your data frame. For example, in the dataset we will work with in this book, there is a variable that represents the snout-vent length of red-eyed treefrog froglets at the start of metamorphosis, which is referred to as simply "SVL.initial" in the dataset. It's short enough that you don't mind typing it, but unique enough to let you know what it refers to.

You can customize all of this for your own purposes, and it is a good idea to try to be consistent throughout your coding. For example, in the previous paragraph I used a period (.) to take the place of a space, since R generally does not like spaces. Alternatively, I could have used an underscore (_) or a hyphen (-), thus creating the hypothetical variables "SVL_initial" or "SVL-initial." Any of these are acceptable, the key is just to be consistent with how you create spaces. Similarly, decide if you want to capitalize variable names or not and stick to it. Remember, R will distinguish between uppercase and lowercase characters (e.g., *SVL.initial* is different from *svl.initial*), so being consistent will help you prevent frustrating mistakes.

The basic point is to just try to be clear and organized and leave instructions for yourself and others. There are many excellent resources out there

that provide a much more exhaustive take on this subject than I have space to provide here, and I certainly encourage you to seek them out. One that I am quite fond of is the British Ecological Society's *A Guide to Reproducible Code in Ecology and Evolution*. A quick internet search will help you find it.

1.5 BEFORE WE GET STARTED

Before we begin, there are just a couple of things that are really useful to know about R and how it works. When you see the greater than sign (">") in the console, that means R is ready for you to type something in. If instead you ever see a plus sign ("+"), that means R is waiting for you to finish some command. R generally knows when a command has been finished or not. If you type into the console "*2+*", R is going to wonder what you are adding to that first 2. You can then type something at the "+" prompt to finish the command you are trying to execute. You can also hit the escape key to cancel whatever command R is waiting for you to finish. In the examples of the code in this book, you won't see the little greater than sign that you see in your console window.

```
#An example of an unfinished command
> 2+
+
```

This works to our advantage when we are writing out more complex things like statistical models. If we start a function, like **lm()** to run a linear model (Chapters 5 and 6), R will keep the command open until we close the parentheses and finish out the command. If that doesn't make sense just yet don't worry, it will.

1.6 CREATING OBJECTS

Let's start with creating simple **objects** in R. Objects allow the user to create very simple symbolic representations of simple or complex datasets or other information which allows the user to then create and run elaborate analyses from seemingly simple code with the object stored in the computer's memory. However, creating and manipulating objects can be hard to get used to for those that are accustomed to drag and drop menus or

using spreadsheets to manually manipulate and analyze data by selecting columns or individual cells. In the end, however, creating and manipulating objects is far more efficient than manually manipulating data, as I hope you will see by the end of the chapter.

Objects are created with the **assign** operator, which is an arrow made from a less than sign and a minus sign and looks like "<-". For example, if we want to create an object called **n** and assign the value of 10 to it, we would type the following:

```
n <- 10
```

In R lingo, you can say this as "n gets 10" or "10 is assigned to n." It can go the other way as well, although this is less common.

```
10 -> n
```

In both of these examples, you created an object called **n** and assigned it the value of 10. You may have noticed that nothing happened after you hit return. In general, that is good. You told R to do something and it did it. You did not tell R to display what **n** was. To do that, you should just type **n** at the command prompt.

```
n
```

```
## [1] 10
```

Remember earlier when we typed the word "*Howdy*" at the prompt and got an error? Well, if we assign something to be called "*Howdy*" then it will be stored in the memory and we can access it.

```
#Assign a value to be stored as Howdy
Howdy <- 23
#See what happens when you type Howdy into the prompt
Howdy
```

```
## [1] 23
```

You can also use the = sign to do this (e.g., $n = 10$). Some folks prefer the equal sign because it is one less key stroke, others prefer the little arrow. I recommend the arrow because it separates assigning objects from other uses of the equal sign, and that is what you will see in these chapters. Once again, to each their own.

Names of objects *MUST* start with a letter and can include letters, numbers, dots (.), and underscores (_). R is case sensitive, so "x" and "X" are different objects. Remember, spelling always counts and spelling mistakes are among the most frequent "bugs" in R code. Whenever you get an error, the first thing you should always look for is a typo in your code.

Box 1.6 - Always remember the following!

R does exactly what you tell it to do and only what you tell it to do.
There is no spell check and no autocorrect.
Spelling counts.
R doesn't make mistakes, we make mistakes.

One thing to be careful of is that it is very easy to write over and replace existing objects. For example, earlier we made an object called **n** and assigned it the value 10. We can easily write over the first **n** with a new version assigned a different value.

```
n <- 15
n
```

```
## [1] 15
```

Maybe it is obvious, but this is a really important thing to realize. Note that when we overwrite **n** with a new value R doesn't give us any warning or anything to say "Hey, you already have something called n in the memory! Are you sure you want to do this?" You could easily overwrite your whole

dataset with a few keystrokes and R won't blink an eye (not that R has an eye to blink, because it is a computer program).

We can also use R as a calculator.

```
2+2
```

```
## [1] 4
```

```
2*2+1
```

```
## [1] 5
```

```
2*(2+1)
```

```
## [1] 6
```

```
2^4/2+1
```

```
## [1] 9
```

```
2^(4/2)+1
```

```
## [1] 5
```

```
2^(4/(2+1))
```

```
## [1] 2.519842
```

Objects stored in the memory can of course also be used in math calculations. Since we have assigned numerical values to both **n** and **Howdy**, we can add them together.

```
n + Howdy
```

```
## [1] 38
```

1.7 FUNCTIONS

Most of the work done by R is going to be done by functions. In Chapter 10 you will learn how to write your own functions, but for now let's just discuss what they are. Functions are pre-written sets of code that we use to do something to a numerical value or set of values organized into an object. Functions can be very simple, like calculating the average of a set of numbers, or very complicated. Functions have two basic parts: a name and a set of parentheses where you specify the objects or values you want to the function to work on. Everything you type between the parentheses are called **arguments**.

For example, the **rnorm()** function calculates a random number selected from a normal distribution. In its simplest form, we can just pass it a single number and see what happens.

```
rnorm(1)
```

```
## [1] 0.565837
```

That produced a single value, which might be positive or negative and is not too far from zero. What happens if you change the 1 to a different number? What happens if you execute the function a bunch of times?

If you run that function over and over, you will notice a pattern and you might start to wonder why the random values being generated are so close to zero. There are a number of hidden default values for the **rnorm()** function. How do we know what they are? This is a good time to introduce the **help** function, which is just a simple **?**. If you are using RStudio you can click on the help tab (lower right screen) and type in your search term there, but the quickest way is to just type a **?** at the prompt followed by the thing you are looking for, like so: