OpenIntro Statistics: Labs for R

Andrew Bray
Mine Çetinkaya-Rundel
Arend Kuyper

2018

Contents

Preface

These lab exercises supplement the third edition of OpenIntro Statistics textbook. Each lab steps through the process of using the R programming language for collecting, analyzing, and using statistical data to make inferences and conclusions about real world phenomena.

This version of the labs have been modified by Arend Kuyper to include new datasets and examples for introductory statistics courses at Northwestern University. Visit Labs for R at OpenIntro for the original materials. The chapters in this book were adapted from labs originally written by Mark Hansen and adapted for OpenIntro by Andrew Bray and Mine Çetinkaya-Rundel.

0.1 Using these labs

Figure 1: Creative Commons Attribution ShareAlike

Chapter 1

Working Efficiently with RStudio

1.1 Why RStudio?

RStudio is an extremely powerful tool that is intended to optimize how we interact with the statistical software known as R. We could use R's basic interface, but RStudio is designed to streamline and organize statistical and analytic work with R. Like any tool we must learn how to use it properly, which is the focus of this lab.

While it might seem clunky or cumbersome at first, it is important to discipline yourself and adhere to sound workflow practices. Doing this from the very beginning will payoff immensely in later labs and beyond — whether or not you intend to work with RStudio in the future. Exercising and expanding your mind to preform analytic coding will make you a better critical thinker and problem solver.

1.2 Setting Up an R Project

It is important to recognize that quality analytic work requires that your work be easy to follow, replicate, and reference by others or by the future you. Therefore it is imperative that you strive to be as organized as possible. RStudio helps you organize all your work on a given data analysis/project through the creation of R projects.

There are several ways one could go about creating an R project, but we would suggest following the steps outlined below. These steps outline how to get setup for the first lab, but should, with obvious alterations, be followed for each lab.

Step 1

Create a folder somewhere on your computer, say on your desktop, and give it a descriptive name (e.g. STAT 202). This folder is where you will keep all of your work for each lab. You could save all your electronic notes here too.

Step 2

Next you will want to create a subfolder for an individual task or sub-project (e.g. **Lab 01**). The graphic below displays an example folder structure.

Step 3

Open RStudio.

Step 4

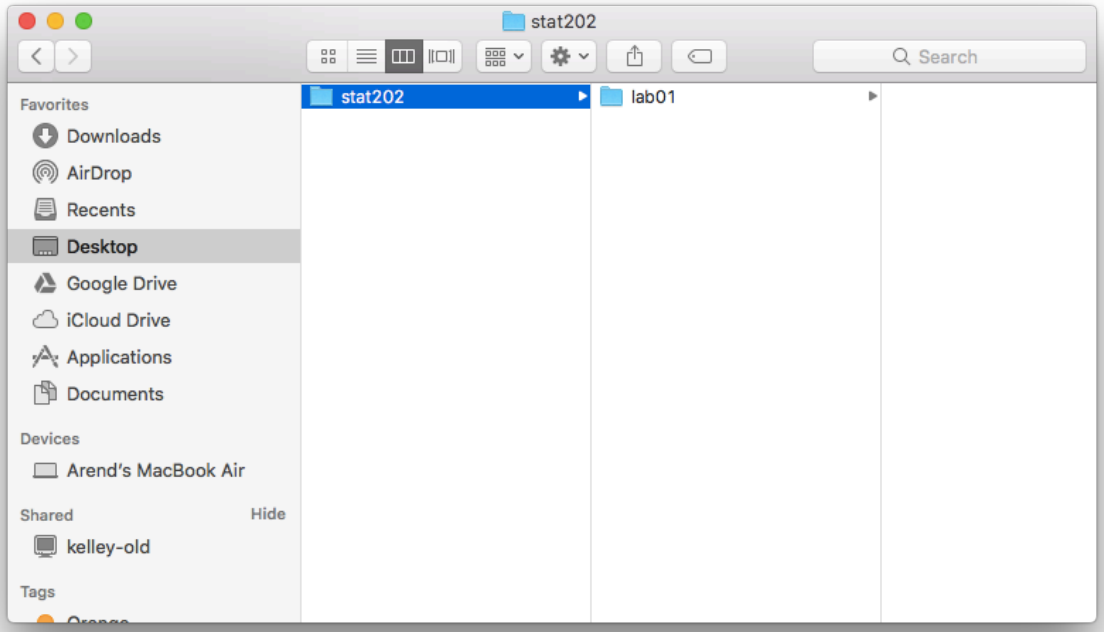

Figure 1.1:

4. Create a project by navigating to the upper right-hand corner of the program and clicking **Project (None) » New Project**.

Step 5

Select **Existing Directory**. Recall that in step 1, you created a file location for lab 1 — our data analysis project.

Step 6

Click **Browse** and navigate to where you created the **Lab 01** folder. Select this folder and then click **Create Project**.

Your R project has now been created. Note that in the upper right-hand corner, the program indicates that you are working on the project named **Lab 01**.

Step 7

Creating an R script is the next MAJOR step in this process. Using a script file is key to organizing your code for quick reference. You should think of an R script file as a thorough record of how to conduct an analysis or solve the problem at hand. You should strive to make your R script as organized as possible so that someone else could work through your code and reproduce the same output/answers/results.

To create an R script, go to the upper left-hand corner and click the white box icon then select **R Script**.

Step 8

Now you can proceed to write your code in the R Script. You can also save your progress by clicking on the save icon located in the icon bar. Notice that it is saved within your R project folder — **Lab 01** in this case. This is why we created an R project, so that all of our work for an analysis/project is kept in one central location.

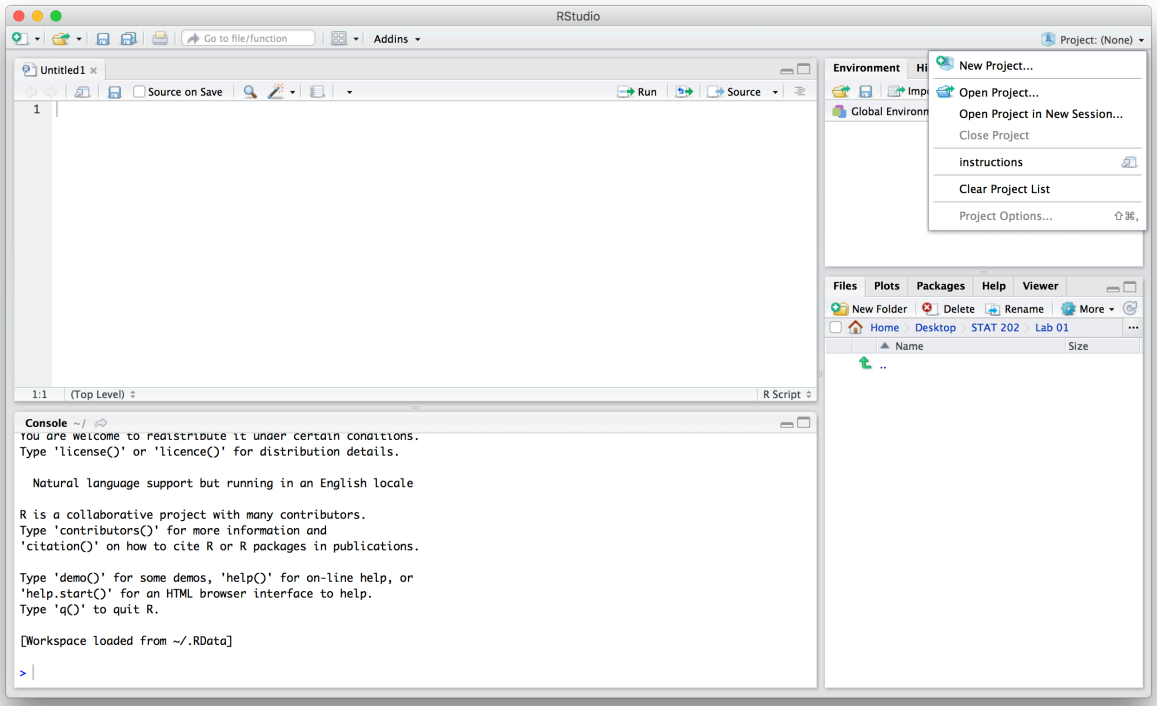

Figure 1.2:

Now, let's practice writing some R code. Good analytic code requires good comments. Comments are meant for human consumption and to explain what the executable code is doing. Therefore we need to let the program know that it is a comment and that it should not attempt to run it. In R, `#` is used to signal a comment. In RStudio, this will turn the line green, which indicates that it will be read as a comment — see the figure below. Also notice how we use white space (empty lines) to make it easier on our eyes to navigate and read the script file. **Practice by typying the code that is pictured.**

We can use R as a calculator, as shown below. To run the line of code `2+2`, you place your cursor anywhere on the that line of code and click **Run**, which is located in the top right corner of the script pane. Alternatively, you could have used the keyboard short cut of **Command + return** (Mac) or **ctrl + enter** (PC).

You also have the option of running multiple lines of code by highlighting the lines of code and clicking **Run** or using the keyboard short cut.

After running the command, the result will show up in your Console pane, which is located beneath the Script pane. In the screenshot below, you can see that our `2+2` command has generated an answer of `4`.

Continue to practice writting an R script by reproducing the code depicted below. As the comments in the pictured code indicate, we are loading/reading-in the `arbuthnot` dataset. Then we are taking a look at some of the observations from the dataset by using the functions `head()` and `tail()`. Make sure to run the lines of code in order, otherwise the the software will return error messages. We suggest running one line at a time to ensure your code is typed correctly and to see what each line is doing.

Notice the copious usage of comments in our analytic code. In general, analytic code should make liberal use of comments. The length and specificity of comments depends on a person's experience with a coding language. With experience comes the understanding and ability to write concise comments that cut directly to what information is absolutely necessary to communicate. It never hurts to have more comments than actual executable code, especially for those new to coding. Keep in mind that in the future you might want

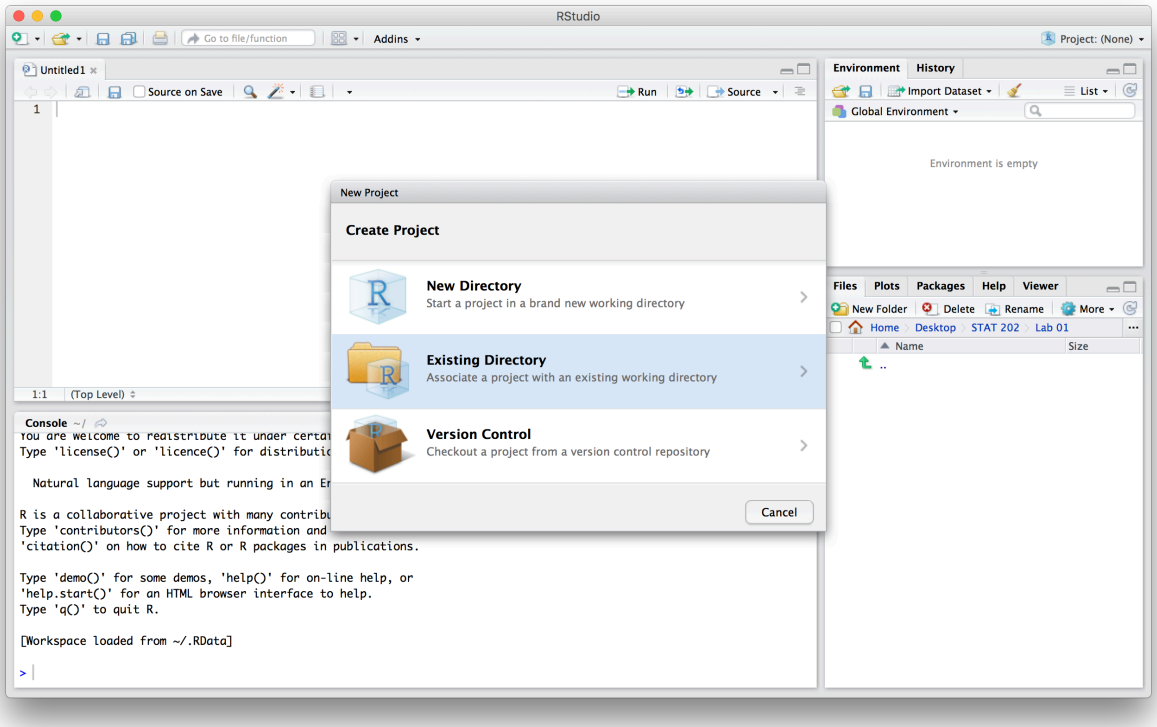

Figure 1.3:

to share analytic code with co-workers or peers, or go back to reference code months or years after you've written it.

1.3 Summary

The essential workflow that should be followed for each lab:

1. Create and work in an R project to ensure all work is kept in a single location.
2. Organize your work within an R script.
    - Use comments to clearly communicate what the code is doing.
    - Use white space (empty lines and spaces) to make the document easier to read and navigate.

There are many other features of RStudio that you'll find useful, but are not covered in this document. RStudio strives to provide help in many different ways:

1. The help tab within the lower right-hand pane.
2. Help automatically appears when typing a function (auto-complete).
3. You may begin by typing the name of function and it will suggest several options.

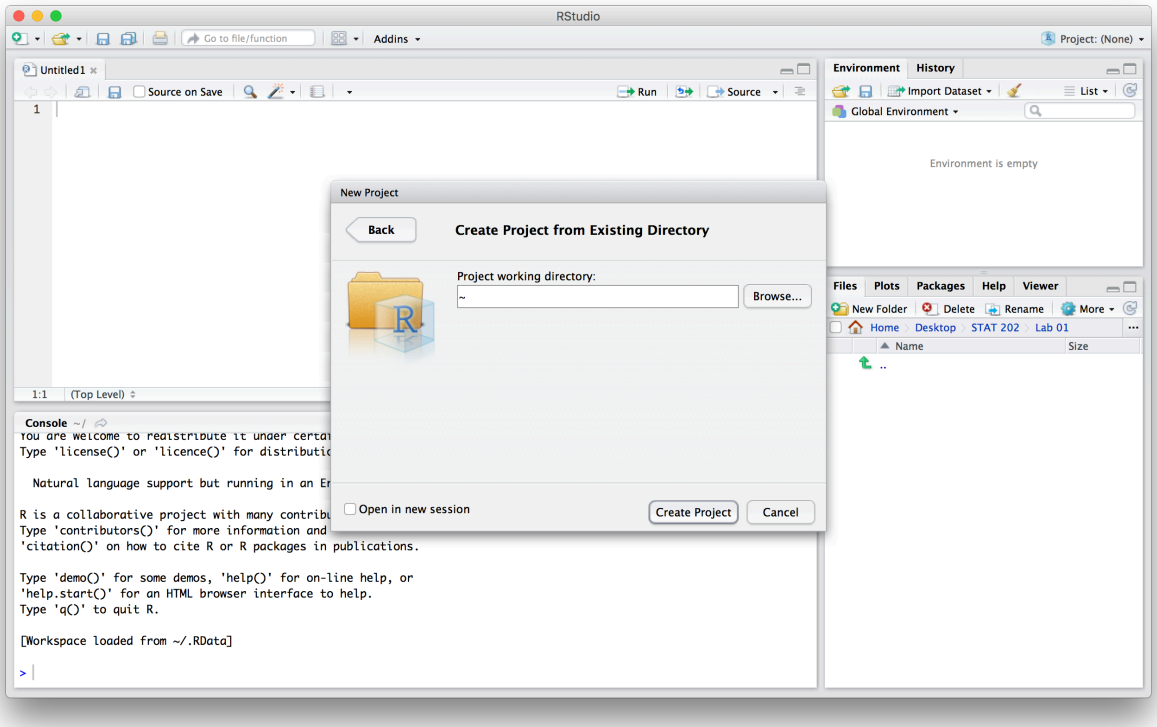
RStudio
Go to file/function
Addins
Project: (None)
Untitled1
Source on Save
Run
Source
Environment
History
Import Dataset
List
Global Environment
Environment is empty
New Project
Back
Create Project from Existing Directory
Project working directory:
~
Browse...
Files
Plots
Packages
Help
Viewer
New Folder
Delete
Rename
More
Home
Desktop
STAT 202
Lab 01
Name
Size
Open in new session
Create Project
Cancel
Console
Type 'license()' or 'licence()' for distributi
Natural language support but running in an E
R is a collaborative project with many contrib
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
[Workspace loaded from ~/.RData]

Figure 1.4:

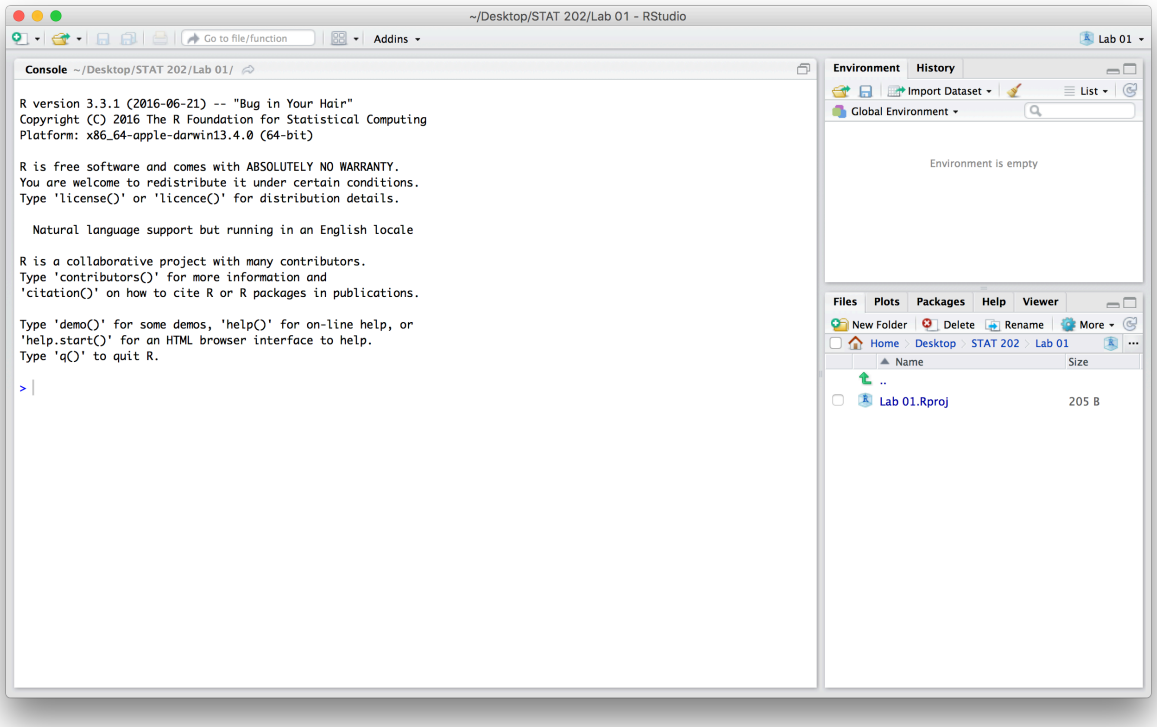
~/Desktop/STAT 202/Lab 01 - RStudio
Go to file/function
Addins
Lab 01
Console ~/Desktop/STAT 202/Lab 01/
R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Environment
History
Import Dataset
List
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
New Folder
Delete
Rename
More
Home
Desktop
STAT 202
Lab 01
Name
Size
..
Lab 01.Rproj
205 B

Figure 1.5:

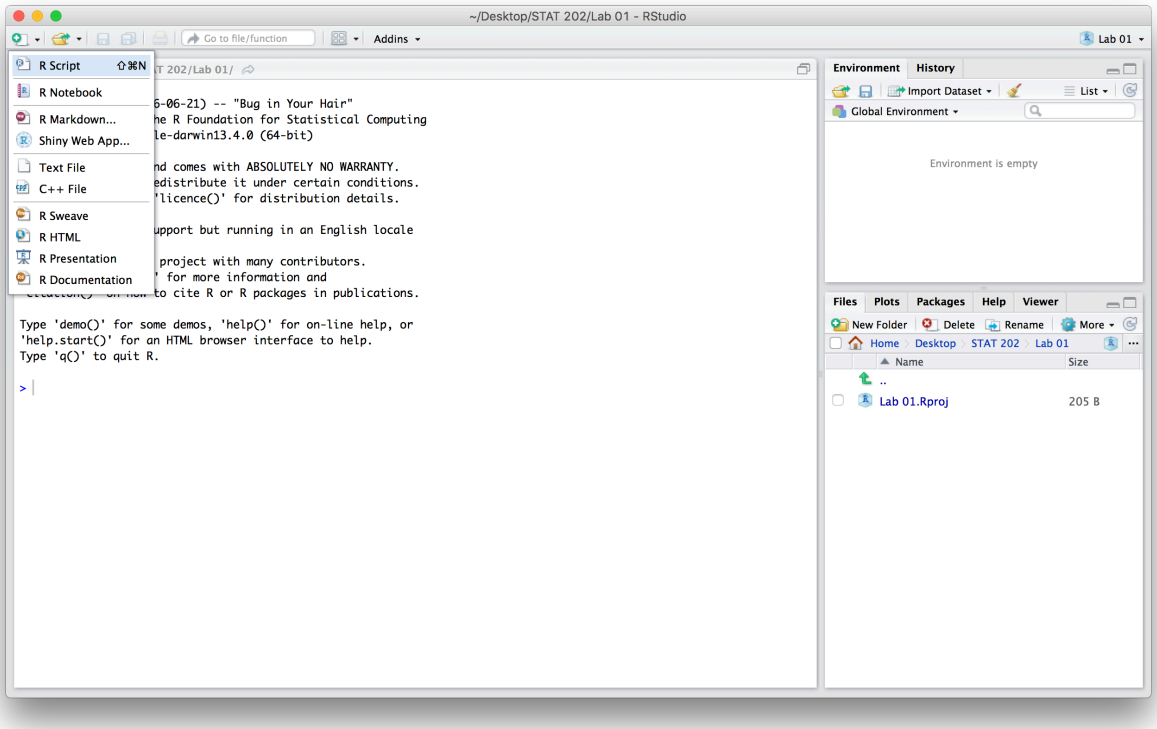
~/Desktop/STAT 202/Lab 01 - RStudio
Go to file/function
Addins
Lab 01
R Script ⇧⌘N
R Notebook
R Markdown...
Shiny Web App...
Text File
C++ File
R Sweave
R HTML
R Presentation
R Documentation
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Environment
History
Import Dataset
List
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
New Folder
Delete
Rename
More
Home
Desktop
STAT 202
Lab 01
Name
Size
Lab 01.Rproj
205 B

Figure 1.6:

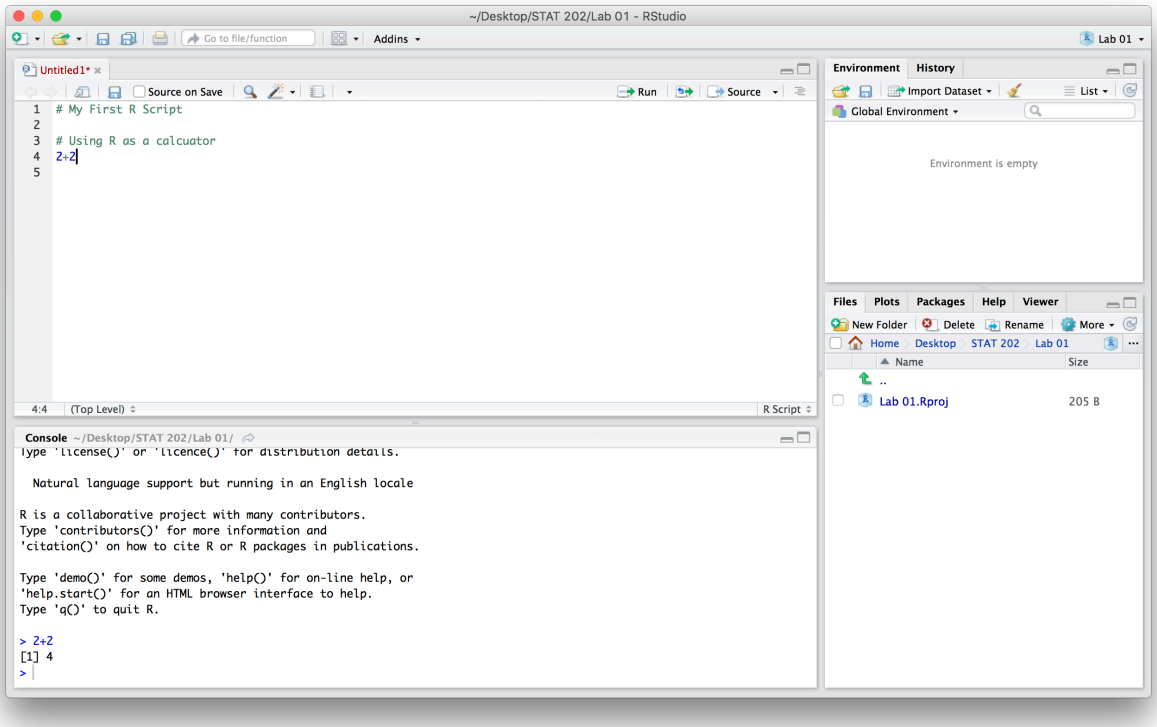
~/Desktop/STAT 202/Lab 01 - RStudio
Go to file/function
Addins
Lab 01
Untitled1*
Source on Save
Run
Source
1 # My First R Script
2
3 # Using R as a calcuator
4 2+2
5
4:4
(Top Level)
R Script
Console ~/Desktop/STAT 202/Lab 01/
type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
> 2+2
[1] 4
>
Environment
History
Import Dataset
List
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
New Folder
Delete
Rename
More
Home
Desktop
STAT 202
Lab 01
Name
Size
..
Lab 01.Rproj
205 B

Figure 1.7:

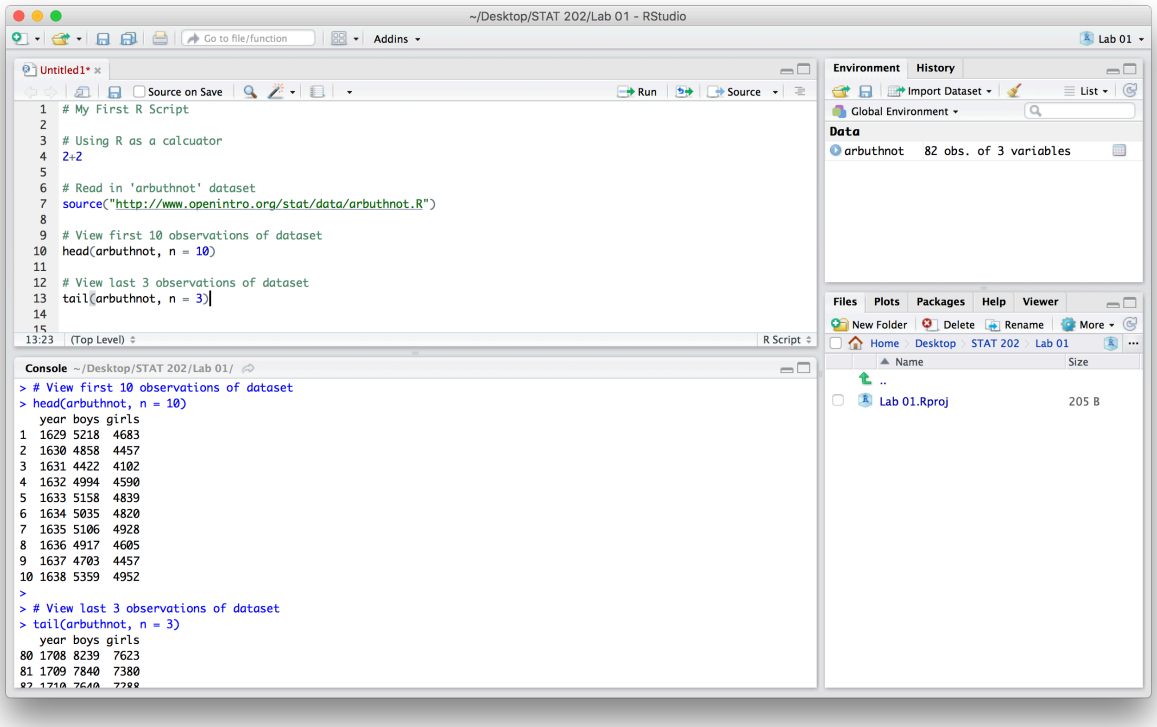
~/Desktop/STAT 202/Lab 01 - RStudio
Addins
Lab 01
Untitled1*
Source on Save
Run
Source
My First R Script
Using R as a calcuator
2+2
# Read in 'arbuthnot' dataset
source("http://www.openintro.org/stat/data/arbuthnot.R")
# View first 10 observations of dataset
head(arbuthnot, n = 10)
# View last 3 observations of dataset
tail(arbuthnot, n = 3)
13:23
(Top Level)
R Script
Console ~/Desktop/STAT 202/Lab 01/
> # View first 10 observations of dataset
> head(arbuthnot, n = 10)
year boys girls
1 1629 5218 4683
2 1630 4858 4457
3 1631 4422 4102
4 1632 4994 4590
5 1633 5158 4839
6 1634 5035 4820
7 1635 5106 4928
8 1636 4917 4605
9 1637 4703 4457
10 1638 5359 4952
> # View last 3 observations of dataset
> tail(arbuthnot, n = 3)
year boys girls
80 1708 8239 7623
81 1709 7840 7380
Environment
History
Import Dataset
List
Global Environment
Data
arbuthnot 82 obs. of 3 variables
Files
Plots
Packages
Help
Viewer
New Folder
Delete
Rename
More
Home
Desktop
STAT 202
Lab 01
Name
Size
..
Lab 01.Rproj
205 B

Figure 1.8:

Chapter 2

Introduction to R and RStudio

The goal of this lab is to introduce you to R and RStudio, which you'll be using throughout the course both to learn the statistical concepts discussed in the texbook and also to analyze real data and come to informed conclusions. To straighten out which is which: R is the name of *the programming language* itself and RStudio is a convenient *user interface* for working with R.

As the labs progress, you are encouraged to explore beyond what the labs dictate; a willingness to experiment will make you a much better programmer. Before we get to that stage, however, you need to build some basic fluency in R. Today we begin with the fundamental building blocks of R and RStudio: the interface, reading in data, and basic commands.

The panel in the upper right contains your *workspace* as well as a history of the commands that you've previously entered. Any plots that you generate will show up in the panel in the lower right corner.

The panel on the left is where the action happens. It's called the *console*. Everytime you launch RStudio, it will have the same text at the top of the console telling you the version of R that you're running. Below that information is the *prompt*. As its name suggests, this prompt is really a request, a request for a command. Initially, interacting with R is all about typing commands and interpreting the output. These commands and their syntax have evolved over decades (literally) and now provide what many users feel is a fairly natural way to access data and organize, describe, and invoke statistical computations.

To get you started, enter the following command at the R prompt (i.e. right after `>` on the console). You can either type it in manually or copy and paste it from this document.

```
source("http://www.openintro.org/stat/data/arbuthnot.R")
```

This command instructs R to access the OpenIntro website and fetch some data: the Arbuthnot baptism counts for boys and girls. You should see that the workspace area in the upper righthand corner of the RStudio window now lists a data set called `arbuthnot` that has 82 observations on 3 variables. As you interact with R, you will create a series of objects. Sometimes you load them as we have done here, and sometimes you create them yourself as the byproduct of a computation or some analysis you have performed. Note that because you are accessing data from the web, this command (and the entire assignment) will work in a computer lab, in the library, or in your dorm room; anywhere you have access to the Internet.

2.1 The Data: Dr. Arbuthnot's Baptism Records

The Arbuthnot data set refers to Dr. John Arbuthnot, an 18th century physician, writer, and mathematician. He was interested in the ratio of newborn boys to newborn girls, so he gathered the baptism records for children born in London for every year from 1629 to 1710. We can take a look at the data by typing its name into the console.

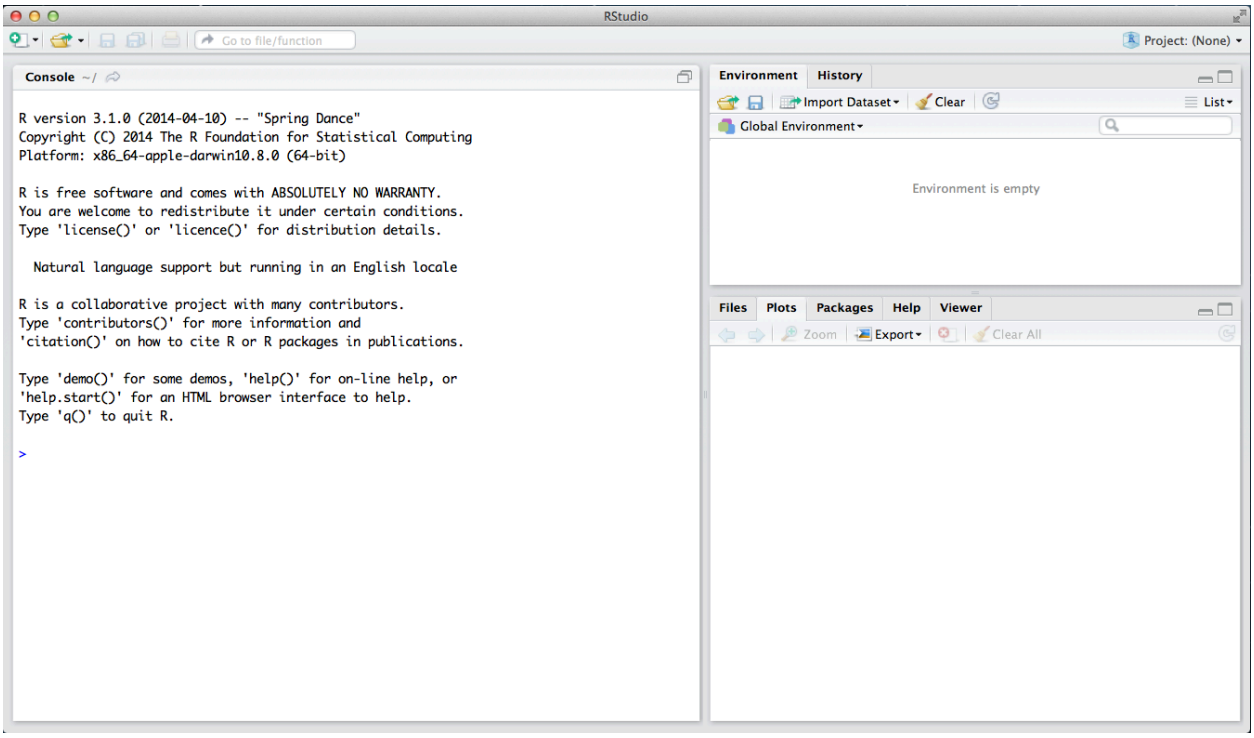

Figure 2.1: The RStudio Interface

```
arbuthnot
```

What you should see are four columns of numbers, each row representing a different year: the first entry in each row is simply the row number (an index we can use to access the data from individual years if we want), the second is the year, and the third and fourth are the numbers of boys and girls baptized that year, respectively. Use the scrollbar on the right side of the console window to examine the complete data set.

Note that the row numbers in the first column are not part of Arbuthnot's data. R adds them as part of its printout to help you make visual comparisons. You can think of them as the index that you see on the left side of a spreadsheet. In fact, the comparison to a spreadsheet will generally be helpful. R has stored Arbuthnot's data in a kind of spreadsheet or table called a *data frame*.

You can see the dimensions of this data frame by typing:

```
dim(arbuthnot)
```

```
## [1] 82  3
```

This command should output `[1] 82 3`, indicating that there are 82 rows and 3 columns (we'll get to what the `[1]` means in a bit), just as it says next to the object in your workspace. You can see the names of these columns (or variables) by typing:

```
names(arbuthnot)
```

```
## [1] "year"  "boys"  "girls"
```

You should see that the data frame contains the columns `year`, `boys`, and `girls`. At this point, you might notice that many of the commands in R look a lot like functions from math class; that is, invoking R commands means supplying a function with some number of arguments. The `dim` and `names` commands, for example, each took a single argument, the name of a data frame.

One advantage of RStudio is that it comes with a built-in data viewer. Click on the name `arbuthnot` in

the *Environment* pane (upper right window) that lists the objects in your workspace. This will bring up an alternative display of the data set in the *Data Viewer* (upper left window). You can close the data viewer by clicking on the x in the upper lefthand corner.

2.2 Some Exploration

Let's start to examine the data a little more closely. We can access the data in a single column of a data frame separately using a command like

```
arbuthnot$boys
```

This command will only show the number of boys baptized each year.

Excercise 1: What command would you use to extract just the counts of girls baptized? Try it!

Notice that the way R has printed these data is different. When we looked at the complete data frame, we saw 82 rows, one on each line of the display. These data are no longer structured in a table with other variables, so they are displayed one right after another. Objects that print out in this way are called *vectors*; they represent a set of numbers. R has added numbers in [brackets] along the left side of the printout to indicate locations within the vector. For example, `5218` follows `[1]`, indicating that `5218` is the first entry in the vector. And if `[43]` starts a line, then that would mean the first number on that line would represent the 43rd entry in the vector.

R has some powerful functions for making graphics. We can create a simple plot of the number of girls baptized per year with the command:

```
plot(x = arbuthnot$year, y = arbuthnot$girls)
```

By default, R creates a scatterplot with each x,y pair indicated by an open circle. The plot itself should appear under the *Plots* tab of the lower right panel of RStudio. Notice that the command above again looks like a function, this time with two arguments separated by a comma. The first argument in the plot function specifies the variable for the x-axis and the second for the y-axis. If we wanted to connect the data points with lines, we could add a third argument, the letter `l` for line.

```
plot(x = arbuthnot$year, y = arbuthnot$girls, type = "l")
```

You might wonder how you are supposed to know that it was possible to add that third argument. Thankfully, R documents all of its functions extensively. To read what a function does and learn the arguments that are available to you, just type in a question mark followed by the name of the function that you're interested in. Try the following.

```
?plot
```

Notice that the help file replaces the plot in the lower right panel. You can toggle between plots and help files using the tabs at the top of that panel.

Exercise 2: Is there an apparent trend in the number of girls baptized over the years? How would you describe it?

Now, suppose we want to plot the total number of baptisms. To compute this, we could use the fact that R is really just a big calculator. We can type in mathematical expressions like

```
5218 + 4683
```

to see the total number of baptisms in 1629. We could repeat this once for each year, but there is a faster way. If we add the vector for baptisms for boys and girls, R will compute all sums simultaneously.

```
arbuthnot$boys + arbuthnot$girls
```

What you will see are 82 numbers (in that packed display, because we aren't looking at a data frame here), each one representing the sum we're after. Take a look at a few of them and verify that they are right. Therefore, we can make a plot of the total number of baptisms per year with the command

```
plot(arbuthnot$year, arbuthnot$boys + arbuthnot$girls, type = "l")
```

This time, note that we left out the names of the first two arguments. We can do this because the help file shows that the default for `plot` is for the first argument to be the x-variable and the second argument to be the y-variable.

Similarly to how we computed the proportion of boys, we can compute the ratio of the number of boys to the number of girls baptized in 1629 with

```
5218 / 4683
```

or we can act on the complete vectors with the expression

```
arbuthnot$boys / arbuthnot$girls
```

The proportion of newborns that are boys

```
5218 / (5218 + 4683)
```

or this may also be computed for all years simultaneously:

```
arbuthnot$boys / (arbuthnot$boys + arbuthnot$girls)
```

Note that with R as with your calculator, you need to be conscious of the order of operations. Here, we want to divide the number of boys by the total number of newborns, so we have to use parentheses. Without them, R will first do the division, then the addition, giving you something that is not a proportion.

Exercise 3: Now, make a plot of the proportion of boys over time. What do you see? Tip: If you use the up and down arrow keys, you can scroll through your previous commands, your so-called command history. You can also access it by clicking on the history tab in the upper right panel. This will save you a lot of typing in the future.

Finally, in addition to simple mathematical operators like subtraction and division, you can ask R to make comparisons like greater than, `>`, less than, `<`, and equality, `==`. For example, we can ask if boys outnumber girls in each year with the expression

```
arbuthnot$boys > arbuthnot$girls
```

This command returns 82 values of either `TRUE` if that year had more boys than girls, or `FALSE` if that year did not (the answer may surprise you). This output shows a different kind of data than we have considered so far. In the `arbuthnot` data frame our values are numerical (the year, the number of boys and girls). Here, we've asked R to create *logical* data, data where the values are either `TRUE` or `FALSE`. In general, data analysis will involve many different kinds of data types, and one reason for using R is that it is able to represent and compute with many of them.

This seems like a fair bit for your first lab, so let's stop here. To exit RStudio you can click the x in the upper right corner of the whole window. You will be prompted to save your workspace. If you click *save*, RStudio will save the history of your commands and all the objects in your workspace so that the next time

you launch RStudio, you will see `arbuthnot` and you will have access to the commands you typed in your previous session. For now, click *save*, then start up RStudio again.

2.3 On Your Own

In the previous few pages, you recreated some of the displays and preliminary analysis of Arbuthnot's baptism data. Your assignment involves repeating these steps, but for present day birth records in the United States. Load up the present day data with the following command.

```
source("http://www.openintro.org/stat/data/present.R")
```

The data are stored in a data frame called `present`.

- What years are included in this data set? What are the dimensions of the data frame and what are the variable or column names?
- How do these counts compare to Arbuthnot's? Are they on a similar scale?
- Make a plot that displays the boy-to-girl ratio for every year in the data set. What do you see? Does Arbuthnot's observation about boys being born in greater proportion than girls hold up in the U.S.? Include the plot in your response.
- In what year did we see the most total number of births in the U.S.? You can refer to the help files or the R reference card http://cran.r-project.org/doc/contrib/Short-refcard.pdf to find helpful commands.

These data come from a report by the Centers for Disease Control http://www.cdc.gov/nchs/data/nvsr/nvsr53/nvsr53_20.pdf. Check it out if you would like to read more about an analysis of sex ratios at birth in the United States.

That was a short introduction to R and RStudio, but we will provide you with more functions and a more complete sense of the language as the course progresses. Feel free to browse around the websites for R and RStudio if you're interested in learning more, or find more labs for practice at http://openintro.org.

Chapter 3

Introduction to Data

This lab is structured to guide you through an organized process such that you could easily organize your code with comments – meaning your R script – into a lab report. I would suggest getting into the habit of writing an organized and commented R script that completes the tasks and answers the questions provided in the lab – including in the **Own Your Own** section.

Some define Statistics as the field that focuses on turning information into knowledge. The first step in that process is to summarize and describe the raw information - the data. In this lab, you will gain insight into public health by generating simple graphical and numerical summaries of a data set collected by the Centers for Disease Control and Prevention (CDC). As this is a large data set, along the way you'll also learn the indispensable skills of data processing and subsetting.

3.1 Getting started

The Behavioral Risk Factor Surveillance System (BRFSS) is an annual telephone survey of 350,000 people in the United States. As its name implies, the BRFSS is designed to identify risk factors in the adult population and report emerging health trends. For example, respondents are asked about their diet and weekly physical activity, their HIV/AIDS status, possible tobacco use, and even their level of healthcare coverage. The BRFSS Web site (http://www.cdc.gov/brfss) contains a complete description of the survey, including the research questions that motivate the study and many interesting results derived from the data.

We will focus on a random sample of 20,000 people from the BRFSS survey conducted in 2000. While there are over 200 variables in this data set, we will work with a small subset.

We begin by loading the data set of 20,000 observations into the R workspace. After launching RStudio, enter the following command.

```
source("http://www.openintro.org/stat/data/cdc.R")
```

The data set `cdc` that shows up in your workspace is a *data matrix*, with each row representing a *case* and each column representing a *variable*. R calls this data format a *data frame*, which is a term that will be used throughout the labs.

To view the names of the variables, type the command

```
names(cdc)
```

This returns the names `genhlth`, `exerany`, `hlthplan`, `smoke100`, `height`, `weight`, `wtdesire`, `age`, and `gender`. Each one of these variables corresponds to a question that was asked in the survey. For example, for `genhlth`, respondents were asked to evaluate their general health, responding either excellent, very good,

good, fair or poor. The `exerany` variable indicates whether the respondent exercised in the past month (1) or did not (0). Likewise, `hlthplan` indicates whether the respondent had some form of health coverage (1) or did not (0). The `smoke100` variable indicates whether the respondent had smoked at least 100 cigarettes in her lifetime. The other variables record the respondent's `height` in inches, `weight` in pounds as well as their desired weight, `wtdesire`, `age` in years, and `gender`.

Exercise 1: How many cases are there in this data set? How many variables? For each variable, identify its data type (e.g. categorical, discrete).

We can have a look at the first few entries (rows) of our data with the command

```
head(cdc)
```

and similarly we can look at the last few by typing

```
tail(cdc)
```

You could also look at *all* of the data frame at once by typing its name into the console, but that might be unwise here. We know `cdc` has 20,000 rows, so viewing the entire data set would mean flooding your screen. It's better to take small peeks at the data with `head`, `tail` or the subsetting techniques that you'll learn in a moment.

3.2 Summaries and tables

The BRFSS questionnaire is a massive trove of information. A good first step in any analysis is to distill all of that information into a few summary statistics and graphics. As a simple example, the function `summary` returns a numerical summary: minimum, first quartile, median, mean, second quartile, and maximum.

For `weight` this is

```
summary(cdc$weight)
```

R also functions like a very fancy calculator. If you wanted to compute the interquartile range for the respondents' weight, you would look at the output from the summary command above and then enter

```
190 - 140
```

R also has built-in functions to compute summary statistics one by one. For instance, to calculate the mean, median, and variance of `weight`, type

```
mean(cdc$weight)
var(cdc$weight)
median(cdc$weight)
```

While it makes sense to describe a quantitative variable like `weight` in terms of these statistics, what about categorical data? We would instead consider the sample frequency or relative frequency distribution. The function `table` does this for you by counting the number of times each kind of response was given. For example, to see the number of people who have smoked 100 cigarettes in their lifetime, type

```
table(cdc$smoke100)
```

or instead look at the relative frequency distribution by typing

```
table(cdc$smoke100)/20000
```

Notice how R automatically divides all entries in the table by 20,000 in the command above. This is similar to something we observed in the Introduction to R; when we multiplied or divided a vector with a number,

R applied that action across entries in the vectors. As we see above, this also works for tables. Next, we make a bar plot of the entries in the table by putting the table inside the `barplot` command.

```
barplot(table(cdc$smoke100))
```

Notice what we've done here! We've computed the table of `cdc$smoke100` and then immediately applied the graphical function, `barplot`. This is an important idea: R commands can be nested. You could also break this into two steps by typing the following:

```
smoke <- table(cdc$smoke100)

barplot(smoke)
```

Here, we've made a new object, a table, called `smoke` (the contents of which we can see by typing `smoke` into the console) and then used it in as the input for `barplot`. The special symbol `<-` performs an *assignment*, taking the output of one line of code and saving it into an object in your workspace. This is another important idea that we'll return to later.

Exercise 2: Create a numerical summary for `height` and `age`, and compute the interquartile range for each. Compute the relative frequency distribution for `gender` and `exerany`. How many males are in the sample? What proportion of the sample reports being in excellent health?

The `table` command can be used to tabulate any number of variables that you provide. For example, to examine which participants have smoked across each gender, we could use the following.

```
table(cdc$gender,cdc$smoke100)
```

Here, we see column labels of 0 and 1. Recall that 1 indicates a respondent has smoked at least 100 cigarettes. The rows refer to gender. To create a mosaic plot of this table, we would enter the following command.

```
mosaicplot(table(cdc$gender,cdc$smoke100))
```

We could have accomplished this in two steps by saving the table in one line and applying `mosaicplot` in the next (see the table/barplot example above).

Exercise 3: What does the mosaic plot reveal about smoking habits and gender?

3.3 Interlude: How R thinks about data

We mentioned that R stores data in data frames, which you might think of as a type of spreadsheet. Each row is a different observation (a different respondent) and each column is a different variable (the first is `genhlth`, the second `exerany` and so on). We can see the size of the data frame next to the object name in the workspace or we can type

```
dim(cdc)
```

which will return the number of rows and columns. Now, if we want to access a subset of the full data frame, we can use row-and-column notation. For example, to see the sixth variable of the 567th respondent, use the format

```
cdc[567,6]
```

which means we want the element of our data set that is in the 567th row (meaning the 567th person or observation) and the 6th column (in this case, weight). We know that `weight` is the 6th variable because it is the 6th entry in the list of variable names

```
names(cdc)
```

To see the weights for the first 10 respondents we can type

```
cdc[1:10,6]
```

In this expression, we have asked just for rows in the range 1 through 10. R uses the `:` to create a range of values, so 1:10 expands to 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. You can see this by entering

```
1:10
```

Finally, if we want all of the data for the first 10 respondents, type

```
cdc[1:10,]
```

By leaving out an index or a range (we didn't type anything between the comma and the square bracket), we get all the columns. When starting out in R, this is a bit counterintuitive. As a rule, we omit the column number to see all columns in a data frame. Similarly, if we leave out an index or range for the rows, we would access all the observations, not just the 567th, or rows 1 through 10. Try the following to see the weights for all 20,000 respondents fly by on your screen

```
cdc[,6]
```

Recall that column 6 represents respondents' weight, so the command above reported all of the weights in the data set. An alternative method to access the weight data is by referring to the name. Previously, we typed `names(cdc)` to see all the variables contained in the cdc data set. We can use any of the variable names to select items in our data set.

```
cdc$weight
```

The dollar-sign tells R to look in data frame `cdc` for the column called `weight`. Since that's a single vector, we can subset it with just a single index inside square brackets. We see the weight for the 567th respondent by typing

```
cdc$weight[567]
```

Similarly, for just the first 10 respondents

```
cdc$weight[1:10]
```

The command above returns the same result as the `cdc[1:10,6]` command. Both row-and-column notation and dollar-sign notation are widely used, which one you choose to use depends on your personal preference.

3.4 A little more on subsetting

It's often useful to extract all individuals (cases) in a data set that have specific characteristics. We accomplish this through *conditioning* commands. First, consider expressions like

```
cdc$gender == "m"
```

or

```
cdc$age > 30
```

These commands produce a series of `TRUE` and `FALSE` values. There is one value for each respondent, where `TRUE` indicates that the person was male (via the first command) or older than 30 (second command).

Suppose we want to extract just the data for the men in the sample, or just for those over 30. We can use the R function `subset` to do that for us. For example, the command

```
mdata <- subset(cdc, cdc$gender == "m")
```

will create a new data set called `mdata` that contains only the men from the `cdc` data set. In addition to finding it in your workspace alongside its dimensions, you can take a peek at the first several rows as usual

```
head(mdata)
```

This new data set contains all the same variables but just under half the rows. It is also possible to tell R to keep only specific variables, which is a topic we'll discuss in a future lab. For now, the important thing is that we can carve up the data based on values of one or more variables.

As an aside, you can use several of these conditions together with `&` and `|`. The `&` is read "and" so that

```
m_and_over30 <- subset(cdc, gender == "m" & age > 30)
```

will give you the data for men over the age of 30. The `|` character is read "or" so that

```
m_or_over30 <- subset(cdc, gender == "m" | age > 30)
```

will take people who are men or over the age of 30 (why that's an interesting group is hard to say, but right now the mechanics of this are the important thing). In principle, you may use as many "and" and "or" clauses as you like when forming a subset.

Exercise 4: Create a new object called `under23_and_smoke` that contains all observations of respondents under the age of 23 that have smoked 100 cigarettes in their lifetime. Write the command you used to create the new object as the answer to this exercise.

3.5 Quantitative data

With our subsetting tools in hand, we'll now return to the task of the day: making basic summaries of the BRFSS questionnaire. We've already looked at categorical data such as `smoke` and `gender` so now let's turn our attention to quantitative data. Two common ways to visualize quantitative data are with box plots and histograms. We can construct a box plot for a single variable with the following command.

```
boxplot(cdc$height)
```

You can compare the locations of the components of the box by examining the summary statistics.

```
summary(cdc$height)
```

Confirm that the median and upper and lower quartiles reported in the numerical summary match those in the graph. The purpose of a boxplot is to provide a thumbnail sketch of a variable for the purpose of comparing across several categories. So we can, for example, compare the heights of men and women with

```
boxplot(cdc$height ~ cdc$gender)
```

The notation here is new. The `~` character can be read *versus* or *as a function of*. So we're asking R to give us a box plots of heights where the groups are defined by gender.

Next let's consider a new variable that doesn't show up directly in this data set: Body Mass Index (BMI) (http://en.wikipedia.org/wiki/Body_mass_index). BMI is a weight to height ratio and can be calculated as:

$$BMI = \frac{weight\ (lb)}{height\ (in)^2} * 703$$

703 is the approximate conversion factor to change units from metric (meters and kilograms) to imperial (inches and pounds).

The following two lines first make a new object called `bmi` and then creates box plots of these values, defining groups by the variable `cdc$genhlth`.

```
bmi <- (cdc$weight / cdc$height^2) * 703
boxplot(bmi ~ cdc$genhlth)
```

Notice that the first line above is just some arithmetic, but it's applied to all 20,000 numbers in the `cdc` data set. That is, for each of the 20,000 participants, we take their weight, divide by their height-squared and then multiply by 703. The result is 20,000 BMI values, one for each respondent. This is one reason why we like R: it lets us perform computations like this using very simple expressions.

Exercise 5: What does this box plot show? Pick another categorical variable from the data set and see how it relates to BMI. List the variable you chose, why you might think it would have a relationship to BMI, and indicate what the figure seems to suggest.

Finally, let's make some histograms. We can look at the histogram for the age of our respondents with the command

```
hist(cdc$age)
```

Histograms are generally a very good way to see the shape of a single distribution, but that shape can change depending on how the data is split between the different bins. You can control the number of bins by adding an argument to the command. In the next two lines, we first make a default histogram of `bmi` and then one with 50 breaks.

```
hist(bmi)
hist(bmi, breaks = 50)
```

Note that you can flip between plots that you've created by clicking the forward and backward arrows in the lower right region of RStudio, just above the plots. How do these two histograms compare?

At this point, we've done a good first pass at analyzing the information in the BRFSS questionnaire. We've found an interesting association between smoking and gender, and we can say something about the relationship between people's assessment of their general health and their own BMI. We've also picked up essential computing tools – summary statistics, subsetting, and plots – that will serve us well throughout this course.

3.6 On Your Own

- Make a scatterplot of weight versus desired weight. Describe the relationship between these two variables.
- Let's consider a new variable: the difference between desired weight (`wtdesire`) and current weight (`weight`). Create this new variable by subtracting the two columns in the data frame and assigning them to a new object called `wdiff`.
- What type of data is `wdiff`? If an observation `wdiff` is 0, what does this mean about the person's weight and desired weight. What if `wdiff` is positive or negative?

- Describe the distribution of `wdiff` in terms of its center, shape, and spread, including any plots you use. What does this tell us about how people feel about their current weight?
- Using numerical summaries and a side-by-side box plot, determine if men tend to view their weight differently than women.
- Now it's time to get creative. Find the mean and standard deviation of `weight` and determine what proportion of the weights that are within one standard deviation of the mean.

Chapter 4

The Normal Distribution

This lab is structured to guide you through an organized process such that you could easily organize your code with comments – meaning your R script – into a lab report. I would suggest getting into the habit of writing an organized and commented R script that completes the tasks and answers the questions provided in the lab – including in the **Own Your Own** section.

4.1 Getting started

In this lab we'll investigate the probability distribution that is most central to statistics: the normal distribution. If we are confident that our data are nearly normal, that opens the door to many powerful statistical methods. Here we'll use the graphical tools of R to assess the normality of our data and also learn how to generate random numbers from a normal distribution.

4.2 The Data

This week we'll be working with measurements of body dimensions. This data set contains measurements from 247 men and 260 women, most of whom were considered healthy young adults.

```
download.file("http://www.openintro.org/stat/data/bdims.RData", destfile = "bdims.RData")
load("bdims.RData")
```

Let's take a quick peek at the first few rows of the data.

```
head(bdims)
```

You'll see that for every observation we have 25 measurements, many of which are either diameters or girths. A key to the variable names can be found at http://www.openintro.org/stat/data/bdims.php, but we'll be focusing on just three columns to get started: weight in kg (`wgt`), height in cm (`hgt`), and `sex` (`1` indicates male, `0` indicates female).

Since males and females tend to have different body dimensions, it will be useful to create two additional data sets: one with only men and another with only women.

```
mdims <- subset(bdims, sex == 1)
fdims <- subset(bdims, sex == 0)
```

Exercise 1: Make a histogram of men's heights and a histogram of women's heights. After plotting each histogram, it might also be helpful to construct the histograms on the same plot/axes. Complete the code below to produce such a plot. Boxplots for each gender might also be helpful. After examining the plots, how would you compare the various aspects of the two distributions?

Complete the code chunk below, by replacing `????`, to constuct a plot containing a histogram of heights for each gender plotted on the same set of axes. Note the code alters the colors to distingish between the histograms.

```
### Constructing plot with histogtrams of heights by sex
# 1) Calculating & storing the x-axis limits to encompass all possible height values
# with a little extra "padding"
x_limits <- range(bdims$hgt) + c(-5,5)
# 2) Plot the first histogram, which initilizes the plotting space
hist( ???? , xlim = x_limits, col = rgb(1,0,0,.4), main = "Histograms of Heights by Sex", xlab = "Heigh
# 3) Add the other gender's histogram
hist( ???? , col = rgb(0,0,1,.4), add = TRUE)
```

4.3 Normal distribution

In your description of the distributions, did you use words like *bell-shaped* or *normal*? It's tempting to say so when faced with a unimodal symmetric distribution.

To see how accurate that description is, we can plot a normal distribution curve on top of a histogram to see how closely the data follow a normal distribution.

The overlaid normal curve should have the same mean and standard deviation as the data. We'll be working with the heights of women, so let's store them as a separate object and then calculate some statistics that will be used/referenced later.

```
fhgtmean <- mean(fdims$hgt)
fhgtsd   <- sd(fdims$hgt)
```

Next we make a density histogram to use as the backdrop and use the `lines` function to overlay a normal probability curve. The difference between a frequency histogram and a density histogram is that while in a frequency histogram the *heights* of the bars add up to the total number of observations, in a density histogram the *areas* of the bars add up to 1. The area of each bar can be calculated as simply the height *times* the width of the bar. Using a density histogram allows us to properly overlay a normal distribution curve over the histogram since the curve is a normal probability density function. Frequency and density histograms both display the same exact shape; they only differ in their y-axis. You can verify this by comparing the frequency histogram you constructed earlier and the density histogram created by the commands below.

```
hist(fdims$hgt, probability = TRUE)
x <- 140:190
y <- dnorm(x = x, mean = fhgtmean, sd = fhgtsd)
lines(x = x, y = y, col = "blue")
```

After plotting the density histogram with the first command, we create the x- and y-coordinates for the normal curve. We chose the `x` range as 140 to 190 in order to span the entire range of `fheight`. To create `y`, we use `dnorm` to calculate the density of each of those x-values in a distribution that is normal with mean `fhgtmean` and standard deviation `fhgtsd`. The final command draws a curve on the existing plot (the density histogram) by connecting each of the points specified by `x` and `y`. The argument `col` simply sets the color for the line to be drawn. If we left it out, the line would be drawn in black.

The top of the curve is cut off because the limits of the x- and y-axes are set to best fit the histogram. To

adjust the y-axis you can add a third argument to the histogram function: `ylim = c(0, 0.06)` – **go back to your code and add this**.

Exercise 2: Based on the this plot, does it appear that the data follow a nearly normal distribution?

4.4 Evaluating the normal distribution

Eyeballing the shape of the histogram is one way to determine if the data appear to be nearly normally distributed, but it can be frustrating to decide just how close the histogram is to the curve. An alternative approach involves constructing a normal probability plot, also called a normal Q-Q plot for "quantile-quantile".

```
qqnorm(fdims$hgt)
qqline(fdims$hgt)
```

A data set that is nearly normal will result in a probability plot where the points closely follow the line. Any deviations from normality leads to deviations of these points from the line. The plot for female heights shows points that tend to follow the line but with some errant points towards the tails. We're left with the same problem that we encountered with the histogram above: how close is close enough?

A useful way to address this question is to rephrase it as: what do probability plots look like for data that I *know* came from a normal distribution? We can answer this by simulating data from a normal distribution using `rnorm`.

```
sim_norm <- rnorm(n = length(fdims$hgt), mean = fhgtmean, sd = fhgtsd)
```

The first argument indicates how many numbers you'd like to generate, which we specify to be the same number of heights in the `fdims` data set using the `length` function. The last two arguments determine the mean and standard deviation of the normal distribution from which the simulated sample will be generated. We can take a look at the shape of our simulated data set, `sim_norm`, as well as its normal probability plot.

3. Make a normal probability plot of `sim_norm`. Do all of the points fall on the line? How does this plot compare to the probability plot for the real data?

Even better than comparing the original plot to a single plot generated from a normal distribution is to compare it to many more plots using the following function. It may be helpful to click the zoom button in the plot window.

```
qqnormsim(fdims$hgt)
```

Exercise 4: Does the normal probability plot for `fdims$hgt` look similar to the plots created for the simulated data? That is, do plots provide evidence that the female heights are nearly normal?

Exercise 5: Using the same technique, determine whether or not female weights appear to come from a normal distribution.

4.5 Normal probabilities

Okay, so now you have a slew of tools to judge whether or not a variable is normally distributed. Why should we care?

It turns out that statisticians know a lot about the normal distribution. Once we decide that a random variable is approximately normal, we can answer all sorts of questions about that variable related to probability. Take, for example, the question of, "What is the probability that a randomly chosen young adult female is taller than 6 feet (about 182 cm)?" (**The study that published this data set is clear to point out that the sample was not random and therefore inference to a general population is not suggested. We do so here only as an exercise.**)

If we assume that female heights are normally distributed (a very close approximation is also okay), we can find this probability by calculating a Z score and consulting a Z table (also called a normal probability table). In R, this is done in one step with the function `pnorm`.

```
1 - pnorm(q = 182, mean = fhgtmean, sd = fhgtsd)
```

Note that the function `pnorm` gives the area under the normal curve below a given value, `q`, with a given mean and standard deviation. Since we're interested in the probability that someone is taller than 182 cm, we have to take one minus that probability.

Assuming a normal distribution has allowed us to calculate a theoretical probability. If we want to calculate the probability empirically, we simply need to determine how many observations fall above 182 then divide this number by the total sample size.

```
sum(fdims$hgt > 182) / length(fdims$hgt)
```

Although the probabilities are not exactly the same, they are reasonably close. The closer that your distribution is to being normal, the more accurate the theoretical probabilities will be.

Exercise 6: Write out two probability questions that you would like to answer; one regarding female heights and one regarding female weights. Calculate the those probabilities using both the theoretical normal distribution as well as the empirical distribution (four probabilities in all). Which variable, height or weight, had a closer agreement between the two methods?

4.6 On Your Own

- Now let's consider some of the other variables in the body dimensions data set. Using the figures at the end of the exercises, match the histogram to its normal probability plot. All of the variables have been standardized (first subtract the mean, then divide by the standard deviation), so the units won't be of any help. If you are uncertain based on these figures, generate the plots in R to check.

  a. The histogram for female biiliac (pelvic) diameter (`bii.di`) belongs to normal probability plot letter _____.

  b. The histogram for female elbow diameter (`elb.di`) belongs to normal probability plot letter _____.

  c. The histogram for general age (`age`) belongs to normal probability plot letter _____.

  d. The histogram for female chest depth (`che.de`) belongs to normal probability plot letter _____.

- Note that normal probability plot D has a slight stepwise pattern. Why do you think this is the case?

- As you can see, normal probability plots can be used both to assess normality and visualize skewness. Make a normal probability plot for female knee diameter (`kne.di`). Based on this normal probability plot, is this variable left skewed, symmetric, or right skewed? Use a histogram to confirm your findings.

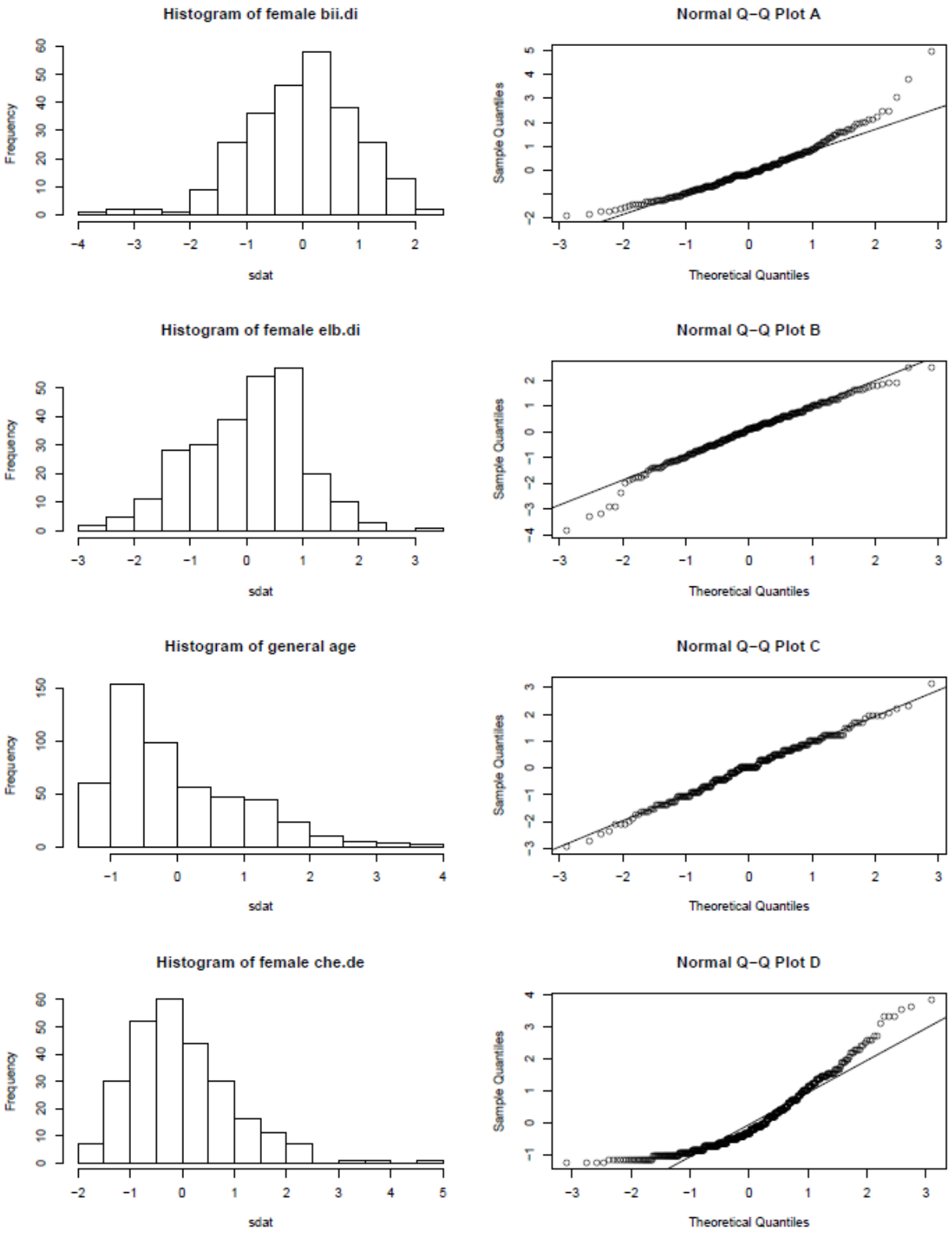

Figure 4.1: HistQMatch

Chapter 5

Foundations for Statistical Inference - Sampling Distributions

This lab is structured to guide you through an organized process such that you could easily organize your code with comments — meaning your R script — into a lab report. I would suggest getting into the habit of writing an organized and commented R script that completes the tasks and answers the questions provided in the lab — including in the **Own Your Own** section.

5.1 Getting started

In this lab, we investigate the ways in which the statistics from a random sample of data can serve as point estimates for population parameters. We're interested in formulating a *sampling distribution* of our estimate in order to learn about the properties of the estimate, such as its distribution.

5.2 The data

We consider real estate data from the city of Ames, Iowa. The details of every real estate transaction in Ames is recorded by the City Assessor's office. Our particular focus for this lab will be all residential home sales in Ames between 2006 and 2010. This collection represents our population of interest. In this lab we would like to learn about these home sales by taking smaller samples from the full population. Let's load the data.

```
download.file("http://www.openintro.org/stat/data/ames.RData", destfile = "ames.RData")
load("ames.RData")
```

We see that there are quite a few variables in the data set, enough to do a very in-depth analysis. For this lab, we'll restrict our attention to just two of the variables: the above ground living area of the house in square feet (`Gr.Liv.Area`) and the sale price (`SalePrice`). To save some effort throughout the lab, create two variables with short names that represent these two variables.

```
area <- ames$Gr.Liv.Area
price <- ames$SalePrice
```

Let's look at the distribution of area in our population of home sales by calculating a few summary statistics and making a histogram.

```
summary(area)
hist(area)
boxplot(area,horizontal = TRUE)
```

Exercise 1: Describe this population distribution.

5.3 The unknown sampling distribution

In this lab we have access to the entire population, but this is rarely the case in real life. Gathering information on an entire population is often extremely costly or impossible. Because of this, we often take a sample of the population and use that to understand the properties of the population.

If we were interested in estimating the mean living area in Ames based on a sample, we can use the following command to survey the population.

```
samp1 <- sample(area, 50)
```

This command collects a simple random sample of size 50 from the vector `area`, which is assigned to `samp1`. This is like going into the City Assessor's database and pulling up the files on 50 random home sales.Working with these 50 files would be considerably simpler than working with all 2930 home sales.

Exercise 2: Describe the distribution of this sample. How does it compare to the distribution of the population?

If we're interested in estimating the average living area in homes in Ames using the sample, our best single guess is the sample mean.

```
mean(samp1)
```

Depending on which 50 homes you selected, your estimate could be a bit above or a bit below the true population mean of 1499.69 square feet. In general, though, the sample mean turns out to be a pretty good estimate of the average living area, and we were able to get it by sampling less than 3% of the population.

Exercise 3: Take a second sample, also of size 50, and call it `samp2`. How does the mean of `samp2` compare with the mean of `samp1`? Suppose we took two more samples, one of size 100 and one of size 1000. Which would you think would provide a more accurate estimate of the population mean?

Not surprisingly, every time we take another random sample, we get a different sample mean. It's useful to get a sense of just how much variability we should expect when estimating the population mean this way. The distribution of sample means, called the *sampling distribution*, can help us understand this variability. In this lab, because we have access to the population, we can build up the sampling distribution for the sample mean by repeating the above steps many times. Here we will generate 5000 samples and compute the sample mean of each.

```
sample_means50 <- rep(NA, 5000)

for(i in 1:5000){
   samp <- sample(area, 50)
   sample_means50[i] <- mean(samp)
```

```
   }

hist(sample_means50)
```

If you would like to adjust the bin width of your histogram to show a little more detail, you can do so by changing the `breaks` argument.

```
hist(sample_means50, breaks = 25)
```

Here we use R to take 5000 samples of size 50 from the population, calculate the mean of each sample, and store each result in a vector called `sample_means50`. On the next page, we'll review how this set of code works.

Exercise 4: How many elements are there in `sample_means50`? Describe the sampling distribution, and be sure to specifically note its center. Would you expect the distribution to change if we instead collected 50,000 sample means?

5.4 Interlude: The `for` loop

Let's take a break from the statistics for a moment to let that last block of code sink in. You have just run your first `for` loop, a cornerstone of computer programming. The idea behind the for loop is *iteration*: it allows you to execute code as many times as you want without having to type out every iteration. In the case above, we wanted to iterate the two lines of code inside the curly braces that take a random sample of size 50 from `area` then save the mean of that sample into the `sample_means50` vector. Without the `for` loop, this would be painful:

```
sample_means50 <- rep(NA, 5000)

samp <- sample(area, 50)
sample_means50[1] <- mean(samp)

samp <- sample(area, 50)
sample_means50[2] <- mean(samp)

samp <- sample(area, 50)
sample_means50[3] <- mean(samp)

samp <- sample(area, 50)
sample_means50[4] <- mean(samp)
```

and so on…

With the for loop, these thousands of lines of code are compressed into a handful of lines. We've added one extra line to the code below, which prints the variable `i` during each iteration of the `for` loop. Run this code.

```
sample_means50 <- rep(NA, 5000)

for(i in 1:5000){
   samp <- sample(area, 50)
   sample_means50[i] <- mean(samp)
   print(i)
   }
```

Let's consider this code line by line to figure out what it does. In the first line we *initialized a vector*. In this case, we created a vector of 5000 zeros called `sample_means50`. This vector will will store values generated within the `for` loop.

The second line calls the `for` loop itself. The syntax can be loosely read as, "for every element `i` from 1 to 5000, run the following lines of code". You can think of `i` as the counter that keeps track of which loop you're on. Therefore, more precisely, the loop will run once when `i = 1`, then once when `i = 2`, and so on up to `i = 5000`.

The body of the `for` loop is the part inside the curly braces, and this set of code is run for each value of `i`. Here, on every loop, we take a random sample of size 50 from `area`, take its mean, and store it as the ith element of `sample_means50`.

In order to display that this is really happening, we asked R to print `i` at each iteration. This line of code is optional and is only used for displaying what's going on while the `for` loop is running.

The `for` loop allows us to not just run the code 5000 times, but to neatly package the results, element by element, into the empty vector that we initialized at the outset.

Exercise 5: To make sure you understand what you've done in this loop, try running a smaller version. Initialize a vector of 100 zeros called `sample_means_small`. Run a loop that takes a sample of size 50 from `area` and stores the sample mean in `sample_means_small`, but only iterate from 1 to 100. Print the output to your screen (type `sample_means_small` into the console and press enter). How many elements are there in this object called `sample_means_small`? What does each element represent?

5.5 Sample size and the sampling distribution

Mechanics aside, let's return to the reason we used a `for` loop: to compute a sampling distribution, specifically, this one.

```
hist(sample_means50)
```

The sampling distribution that we computed tells us much about estimating the average living area in homes in Ames. Because the sample mean is an unbiased estimator, the sampling distribution is centered at the true average living area of the the population, and the spread of the sampling distribution indicates how much variability is induced by sampling only 50 home sales.

To get a sense of the effect that sample size has on our distribution, let's build up two more sampling distributions: one based on a sample size of 10 and another based on a sample size of 100.

```
sample_means10 <- rep(NA, 5000)
sample_means100 <- rep(NA, 5000)

for(i in 1:5000){
  samp <- sample(area, 10)
  sample_means10[i] <- mean(samp)
  samp <- sample(area, 100)
  sample_means100[i] <- mean(samp)
}
```

Here we're able to use a single `for` loop to build two distributions by adding additional lines inside the curly braces. Don't worry about the fact that `samp` is used for the name of two different objects. In the second command of the `for` loop, the mean of `samp` is saved to the relevant place in the vector `sample_means10`. With the mean saved, we're now free to overwrite the object `samp` with a new sample, this time of size 100.

In general, anytime you create an object using a name that is already in use, the old object will get replaced with the new one.

To see the effect that different sample sizes have on the sampling distribution, plot the three distributions on top of one another.

```
par(mfrow = c(3, 1))

xlimits <- range(sample_means10)

hist(sample_means10, breaks = 20, xlim = xlimits)
hist(sample_means50, breaks = 20, xlim = xlimits)
hist(sample_means100, breaks = 20, xlim = xlimits)
```

The first command specifies that you'd like to divide the plotting area into 3 rows and 1 column of plots (to return to the default setting of plotting one at a time, use `par(mfrow = c(1, 1))`). The `breaks` argument specifies the number of bins used in constructing the histogram. The `xlim` argument specifies the range of the x-axis of the histogram, and by setting it equal to `xlimits` for each histogram, we ensure that all three histograms will be plotted with the same limits on the x-axis.

Exercise 6: When the sample size is larger, what happens to the center? What about the spread?

5.6 On your own

So far, we have only focused on estimating the mean living area in homes in Ames. Now you'll try to estimate the mean home price.

- Take a random sample of size 50 from `price`. Using this sample, what is your best point estimate of the population mean?
- Since you have access to the population, simulate the sampling distribution for $\bar{x}_{price}$ by taking 5000 samples from the population of size 50 and computing 5000 sample means. Store these means in a vector called `sample_means50`. Plot the data, then describe the shape of this sampling distribution. Based on this sampling distribution, what would you guess the mean home price of the population to be? Finally, calculate and report the population mean.
- Change your sample size from 50 to 150, then compute the sampling distribution using the same method as above, and store these means in a new vector called `sample_means150`. Describe the shape of this sampling distribution, and compare it to the sampling distribution for a sample size of 50. Based on this sampling distribution, what would you guess to be the mean sale price of homes in Ames?
- Of the sampling distributions from 2 and 3, which has a smaller spread? If we're concerned with making estimates that are more often close to the true value, would we prefer a distribution with a large or small spread?

Chapter 6

Confidence Intervals - Foundations for Statistical Inference

This lab is structured to guide you through an organized process such that you could easily organize your code with comments — meaning your R script — into a lab report. I would suggest getting into the habit of writing an organized and commented R script that completes the tasks and answers the questions provided in the lab – including in the **Own Your Own** section.

6.1 Sampling from Ames, Iowa

If you have access to data on an entire population, say the size of every house in Ames, Iowa, it's straight forward to answer questions like, "How big is the typical house in Ames?" and "How much variation is there in sizes of houses?". If you have access to only a sample of the population, as is often the case, the task becomes more complicated. What is your best guess for the typical size if you only know the sizes of several dozen houses? This sort of situation requires that you use your sample to make inference on what your population looks like.

6.2 The data

In the previous lab, "Sampling Distributions", we looked at the population data of houses from Ames, Iowa. Let's start by loading that data set.

```
download.file("http://www.openintro.org/stat/data/ames.RData", destfile = "ames.RData")
load("ames.RData")
```

In this lab we'll start with a simple random sample of size 60 from the population. Specifically, this is a simple random sample of size 60. Note that the data set has information on many housing variables, but for the first portion of the lab we'll focus on the size of the house, represented by the variable `Gr.Liv.Area`.

```
population <- ames$Gr.Liv.Area
samp <- sample(population, 60)
```

Exercise 1: Describe the distribution of your sample. What would you say is the "typical" size within your sample? Also state precisely what you interpreted "typical" to mean.

Exercise 2: Would you expect another student's distribution to be identical to yours? Would you expect it to be similar? Why or why not?

6.3 Confidence intervals

One of the most common ways to describe the typical or central value of a distribution is to use the mean. In this case we can calculate the mean of the sample using,

```
sample_mean <- mean(samp)
```

Return for a moment to the question that first motivated this lab: based on this sample, what can we infer about the population? Based only on this single sample, the best estimate of the average living area of houses sold in Ames would be the sample mean, usually denoted as $\bar{x}$ (here we're calling it `sample_mean`). That serves as a good *point estimate* but it would be useful to also communicate how uncertain we are of that estimate. This can be captured by using a *confidence interval*.

We can calculate a 95% confidence interval for a sample mean by adding and subtracting 1.96 standard errors to the point estimate (See Section 4.2.3 if you are unfamiliar with this formula). Note that the 1.96 is the result of rounding and we could ues R to find a more precise value which is provided in the code below.

```
qnorm(0.975) # or
qnorm(0.025) # which is the negative version
```

Note that if we take 0.975 - 0.025 we get 0.95. Each tail is set to have area 0.025. Usually two decimal places of accuracy is suffcient when determining the appropriate z^* value for a given confidence level. Therefore we will continue using 1.96, but keep the function above in mind when you desire to use a different confidence level or you need more precision.

```
se <- sd(samp) / sqrt(60)
lower <- sample_mean - 1.96 * se
upper <- sample_mean + 1.96 * se
c(lower, upper)
```

This is an important inference that we've just made: even though we don't know what the full population looks like, we're 95% confident that the true average size of houses in Ames lies between the values *lower* and *upper*. There are a few conditions that must be met for this interval to be valid.

Exercise 3: For the confidence interval to be valid, the sample mean must be normally distributed and have standard error $s/\sqrt{n}$. What conditions must be met for this to be true?

6.4 Confidence levels

Exercise 4: What does "95% confidence" mean?

In this case we have the luxury of knowing the true population mean since we have data on the entire population. This value can be calculated using the following command:

```
mean(population)
```

Exercise 5: Does your confidence interval capture the true average size of houses in Ames? If you are working on this lab in a classroom, does your neighbor's interval capture this value?

Exercise 6: Each student in your class should have gotten a slightly different confidence interval. What proportion of those intervals would you expect to capture the true population mean? Why? If you are working in this lab in a classroom, collect data on the intervals created by other students in the class and calculate the proportion of intervals that capture the true population mean.

Using R, we're going to recreate many samples to learn more about how sample means and confidence intervals vary from one sample to another. *Loops* come in handy here (If you are unfamiliar with loops, review the **Foundations for Statistical Inference (Lab 04)**).

Here is the rough outline:

- Obtain a random sample.
- Calculate and store the sample's mean and standard deviation.
- Repeat steps (1) and (2) 500 times.
- Use these stored statistics to calculate many confidence intervals.

But before we do all of this, we need to first create empty vectors where we can save the means and standard deviations that will be calculated from each sample. And while we're at it, let's also store the desired sample size as `n`.

```
samp_mean <- rep(NA, 500)
samp_sd <- rep(NA, 500)
n <- 60
```

Now we're ready for the loop where we calculate the means and standard deviations of 500 random samples.

```
for(i in 1:500){
  samp <- sample(population, n) # obtain a sample of size n = 60 from the population
  samp_mean[i] <- mean(samp)    # save sample mean in ith element of samp_mean
  samp_sd[i] <- sd(samp)        # save sample sd in ith element of samp_sd
}
```

Lastly, we construct the confidence intervals.

```
lower_vector <- samp_mean - 1.96 * samp_sd / sqrt(n)
upper_vector <- samp_mean + 1.96 * samp_sd / sqrt(n)
```

Lower bounds of these 500 confidence intervals are stored in `lower_vector`, and the upper bounds are in `upper_vector`. Let's view the first interval.

```
c(lower_vector[1], upper_vector[1])
```

6.5 On your own

- Using the function `plot_ci()` (which was downloaded with the data set), we are able to plot the first fifity 95% confidence intervals of our 500. What proportion of the 50 plotted confidence intervals include the true population mean? Is this proportion exactly equal to the confidence level? If not, explain why. Devise and implement a process to calculate the number (and proportion) of confidence intervals that include the true population for all 500 95% confidence intervals - you don't want to have to plot and count by hand.

```
plot_ci(lower_vector, upper_vector, mean(population))
```

- Suppose we want 90% confidence intervals instead of 95% confidence intervals. What is the appropriate critical value?
- Construct 500 90% confidence intervals. You do not need to obtain new samples, simply calculate new intervals based on the sample means and standard deviations you have already collected. Using the `plot_ci` function, plot the first 50 90% confidence intervals and calculate the proportion of intervals that include the true population mean. How does this percentage compare to the confidence level selected for the intervals? Using the method which you implemented in question 1 of the **On Your Own** section, determine the number (and proportion) of the 500 randomly generated 90% confidence intervals that include the true population mean.

Chapter 7

Inference for Numerical Data

The lab is structured to guide you through an organized process such that you could easily organize your code with comments – meaning your R script – into a lab report. I would suggest getting into the habit of writing an organized and commented R script that completes the tasks and answers the questions provided in the lab – including in the **Own Your Own** section.

7.1 Overview

We will be conducting hypothesis tests (HTs) and constructing confidence intervals (CIs) for means and difference of means throughout this lab. We will calculate them by "hand" and through the use of a built in function in R called `t.test()`, which is an extremely useful and flexible function when given the raw sample. Sometimes we are only given access to sample statistics (e.g. $\bar{x}$, s_x, n), which necessitates that we perform calculations by "hand" – the function `t.test()` requires the raw data.

7.2 North Carolina births

In 2004, the state of North Carolina released a large data set containing information on all births recorded in their state. This data set is useful to researchers studying the relation between habits and practices of expectant mothers and the birth of their children. We will work with a random sample of observations from this data set.

7.3 Exploratory analysis

Load the `nc` data set into our workspace.

```
download.file("http://www.openintro.org/stat/data/nc.RData", destfile = "nc.RData")
load("nc.RData")
```

We observations of 13 different variables, some categorical and some numerical. The meaning of each variable is as follows:

variable	description
`fage`	father's age in years.
`mage`	mother's age in years.
`mature`	maturity status of mother.
`weeks`	length of pregnancy in weeks.
`premie`	whether the birth was classified as premature (premie) or full-term.
`visits`	number of hospital visits during pregnancy.
`marital`	whether mother is `married` or `not married` at birth.
`gained`	weight gained by mother during pregnancy in pounds.
`weight`	weight of the baby at birth in pounds.
`lowbirthweight`	whether baby was classified as low birthweight (`low`) or not (`not low`).
`gender`	gender of the baby, `female` or `male`.
`habit`	status of the mother as a `nonsmoker` or a `smoker`.
`whitemom`	whether mom is `white` or `not white`.

Exercise 1: What are the cases in this data set? How many cases are there in our sample?

As a first step in the analysis, we should consider summaries of the data. This can be done using the `summary` command:

```
summary(nc)
```

As you review the variable summaries, consider which variables are categorical and which are numerical. For

numerical variables, are there outliers? If you aren't sure or want to take a closer look at the data, make a graph.

Suppose we want to investigate the typical age for mothers and fathers in North Carolina. Begin by constructing histograms, box plots, and calculating summary statistics.

```
# Mother's age
hist(nc$mage)
boxplot(nc$mage, horizontal = TRUE)
summary(nc$mage)
sd(nc$mage)
IQR(nc$mage)

# Father's age
hist(nc$fage)
boxplot(nc$fage, horizontal = TRUE)
summary(nc$fage)
sd(nc$fage)
IQR(nc$fage)
```

Note that `sd(nc$fage)` or `IQR(nc$fage)` do not return valid output/values. The summary output indicated that there were 171 births where a father's age was missing or not reported. By default most R function will not return valid output when data is missing. This can be fixed by adding the argument `na.rm = TRUE` in the function call – see below.

```
# Father's age continued
sd(nc$fage, na.rm = TRUE)
IQR(nc$fage, na.rm = TRUE)
```

Suppose we want to test the hypothesis that the mean age for women giving birth in North Carolina is 26.5 years. Calculating by "hand":

```
# Sample Data
xbar <- mean(nc$mage)
std <- sd(nc$mage)
samp_size <- length(nc$mage)

# Hypothesis Test -- Calculating p-value
dof <- samp_size - 1
std_err <- std/sqrt(samp_size)
test_stat <- (xbar - 26.5)/std_err
p_value <- 2*pt(-abs(test_stat),df = dof)
p_value

# Constructing 95% CI for the mean
t_star <- qt(p = .975, df = dof)
LB <- xbar - t_star*std_err
UB <- xbar + t_star*std_err
c(LB,UB)
```

Since our p-value is less than significance level α (0.05), we have sufficient evidence to reject that the mean age of women giving birth in North Carolina is 26.5 years old. Note that 26.5 is not in the 95% confidence interval for the mean age of birthing women in NC. Two-tailed hypothesis tests for the mean with significance level α are logically equivalent to $100(1-\alpha)\%$ confidence interval for the mean – **this is a big deal**.

Let's make use of the `t.test()` function now. The function has several inputs that you should become familiar with and work to understand.

```
# Mother's mean age -- HT and 95% CI
t.test(x = nc$mage, alternative = "two.sided", mu = 26.5, conf.level = 0.95)
```

Exercise 2: Suppose now we want test whether the mean age of NC fathers is 30 years. Use $\alpha = 0.01$. Also construct a 99% confidence interval for the mean. Calculate by "hand" and by using `t.test()`. *Hint: When calculating by "hand", missing values can be an issue so first extract and store the useful observations as shown below. The `t.test()` takes care of missingness automatically.*

```
# Extract and store useful data
keep_condition <- !is.na(nc$fage)
f_age <- nc$fage[keep_condition]
```

Suppose a researcher wants to test the hypothesis that in NC the mean age of fathers is different than the mean age of mothers at birth – assume an $\alpha = 0.01$. This is a test for the difference in two population means, which requires us to ask whether the data for the two groups (mothers and fathers) is paired or not. Clearly, the data is paired since each mother and father can be reasonably matched together. First by "hand",

```
# Paired -- only need the differences in age
age_diff <- nc$mage - nc$fage
summary(age_diff)

# Diff. sample statistics -- note their are missing values
xbar <- mean(age_diff, na.rm = TRUE)
std <- sd(age_diff, na.rm = TRUE)
samp_size <- sum(!is.na(age_diff))

# Hypothesis Test -- Calculating p-value
dof <- samp_size - 1
std_err <- std/sqrt(samp_size)
test_stat <- (xbar - 0)/std_err
p_value <- 2*pt(-abs(test_stat),df = dof)
p_value

# Constructing 99% CI for the mean
t_star <- qt(p = .995, df = dof)
LB <- xbar - t_star*std_err
UB <- xbar + t_star*std_err
c(LB,UB)
```

Using `t.test()` – a few coding options.

```
# Option 1: Calculate the differences externally and feed
# them into the t.test function
t.test(x = age_diff, alternative = "two.sided", mu = 0, conf.level = 0.99)

# Option 2: Let the function calculate the differences by
# setting paired = TRUE
t.test(x = nc$mage, y = nc$fage, alternative = "two.sided", mu = 0, paired = TRUE)
```

Now consider the possible relationship between a mother's smoking habit and the weight of her baby. Plotting the data is a useful first step because it helps us quickly visualize trends, identify strong associations, and develop research questions.

Exercise 3: Make a side-by-side box plot of `habit` and `weight`. What does the plot highlight about the relationship between these two variables?

The box plots show how the medians of the two distributions compare, but we can also compare the means

of the distributions using the following function to split the `weight` variable into the `habit` groups, then take the mean of each using the `mean` function.

```
by(nc$weight, nc$habit, mean)
```

There is an observed difference, but is this difference statistically significant? In order to answer this question we will conduct a hypothesis test.

> **Exercise 4:** Check if the conditions necessary for inference are satisfied. Note that you will need to obtain sample sizes to check the conditions. You can compute the group size using the same `by` command above but replacing `mean` with `length`.
>
> **Exercise 5:** Write the hypotheses for testing if the average weights of babies born to smoking and non-smoking mothers are different.

Again, this a hypothesis test for a difference of two means, but in this case the data is not paired – there is no reasonable way of matching members from one group (smokers) to the other (non-smokers). Are the two groups independent from one another? There is no reason to believe that the groups are dependent since the records were randomly sampled. First by "hand",

```
# Sample statistics for baby weights for smoking mothers
grp1 <- nc$weight[nc$habit == "smoker"]
xbar1 <- mean(grp1, na.rm = TRUE)
std1 <- sd(grp1, na.rm = TRUE)
samp_size1 <- sum(!is.na(grp1))

# Sample statistics for baby weights for nonsmoking mothers
grp2 <- nc$weight[nc$habit == "nonsmoker"]
xbar2 <- mean(grp2, na.rm = TRUE)
std2 <- sd(grp2, na.rm = TRUE)
samp_size2 <- sum(!is.na(grp2))

# Hypothesis Test -- Calculating p-value
dof <- min(c(samp_size1,samp_size2)) - 1
std_err <- sqrt(std1^2/samp_size1 + std2^2/samp_size2)
test_stat <- (xbar1-xbar2 - 0)/std_err
p_value <- 2*pt(-abs(test_stat),df = dof)
p_value

# Constructing 95% CI for the mean
t_star <- qt(p = .975, df = dof)
LB <- (xbar1 - xbar2) - t_star*std_err
UB <- (xbar1 - xbar2) + t_star*std_err
c(LB,UB)
```

Using `t.test()`.

```
# Note taht grp1 and grp2 come from above
t.test(x = grp1, y = grp2, alternative = "two.sided", mu = 0, var.equal = FALSE, conf.level = .95)
```

Notice that the by "hand" calculations and the results from `t.test()` do not match. The difference is caused by the use of different degrees of freedom calculations. The software is utilizing the exact calculation for the degrees of freedom while we are utilizing a *conservative* estimate to the degrees of freedom.

Also note `var.equal =` is an indicator/flag for whether we are willing to make the assumption that the two groups have equal variance (i.e. spread/variability). In most cases it is safer not to make this assumption, thus the default is `FALSE`. Although, some software and researchers will make this assumption. Set `var.equal`

`= TRUE` and see what happens. **What affect did this assumption have on the p-value and confidence interval?**

7.4 On your own

- Calculate a 95% confidence interval for the average length of pregnancies (`weeks`) and interpret it in context.
- Calculate a new confidence interval for the same parameter at the 90% confidence level.
- Conduct a hypothesis test evaluating whether the average weight gained by younger mothers is different than the average weight gained by mature mothers.
- Now, a non-inference task: Determine the age cutoff for younger and mature mothers. Use a method of your choice, and explain how your method works.
- Pick a pair of numerical and categorical variables and come up with a research question evaluating the relationship between these variables. Formulate the question in a way that it can be answered using a hypothesis test and/or a confidence interval. Answer your question using the `t.test()` function, report the statistical results, and also provide an explanation in plain language.

Chapter 8

Inference for Categorical Data

This lab is structured to guide you through an organized process such that you could easily organize your code with comments — meaning your R script — into a lab report. I would suggest getting into the habit of writing an organized and commented R script that completes the tasks and answers the questions provided in the lab — including in the **Own Your Own** section.

8.0.1 Getting Started

In August of 2012, news outlets ranging from the Washington Post to the Huffington Post ran a story about the rise of atheism in America. The source for the story was a poll that asked people, "Irrespective of whether you attend a place of worship or not, would you say you are a religious person, not a religious person or a convinced atheist?" This type of question, which asks people to classify themselves in one way or another, is common in polling and generates categorical data. In this lab we take a look at the atheism survey and explore what's at play when making inference about population proportions using categorical data.

8.1 The survey

To download a copy of the press release for the poll, conducted by WIN-Gallup International, click on the following link:

Download the Press Release

Take a moment to review the report, then address the following questions.

Exercise 1: In the first paragraph, several key findings are reported. Do these percentages appear to be *sample statistics* (derived from the sample data) or *population parameters*?

Exercise 2: The title of the report is "Global Index of Religiosity and Atheism". To generalize the report's findings to the global human population, what must we assume about the sampling method? Does that seem like a reasonable assumption?

8.2 The data

Turn your attention to Table 6 (pages 15 and 16), which reports the sample size and response percentages for all 57 countries. While this is a useful format to summarize the data, we will base our analysis on the original data set of individual responses to the survey. Load this dataset into R with the following command.

```
download.file("http://www.openintro.org/stat/data/atheism.RData", destfile = "atheism.RData")
load("atheism.RData")
```

Exercise 3: What does each row of Table 6 correspond to? What does each row of `atheism` correspond to?

To investigate the link between these two ways of organizing this data, take a look at the estimated proportion of atheists in the United States. Towards the bottom of Table 6, we see that this is 5%. We should be able to come to the same number using the `atheism` data.

Exercise 4: Using the command below, create a new dataframe called `us12` that contains only the rows in `atheism` associated with respondents to the 2012 survey from the United States. Next, calculate the proportion of atheist responses. Does it agree with the percentage in Table 6? If not, why?

```
us12 <- subset(atheism, nationality == "United States" & year == "2012")
```

8.3 Inference on proportions

As was hinted at in Exercise 1, Table 6 provides *statistics*, that is, calculations made from the sample of 51,927 people. What we'd like, though, is insight into the population *parameters*. You answer the question, "What proportion of people in your sample reported being atheists?" with a statistic; while the question "What proportion of people on earth would report being atheists" is answered with an estimate of the parameter.

The inferential tools for estimating population proportion are analogous to those used for means in the last chapter: the confidence interval and the hypothesis test.

> **Exercise 5:** Write out the conditions for inference to construct a 95% confidence interval for the proportion of atheists in the United States in 2012. Are you confident all conditions are met?

If the conditions for inference are reasonable, we can calculate the standard error and construct the interval by "hand" as outlined in our book. Note that since the goal is to construct an interval estimate for a proportion, it's necessary to specify what constitutes a "success", which here is a response of `"atheist"`.

```
# First we need to identify the successes and failures
# We can find out what our options with the following
levels(us12$response)

# Since we are looking for atheist we define a success as
# "atheist" and a failure as not atheist (regarless of other
# categories)
# -- code returns TRUE for atheist and FALSE otherwise
atheist_yes_no <- us12$response == "atheist"

# Checking how many atheist and non-atheist are in the sample
# Also a check to verify we have at least 10 successes and failures
table(atheist_yes_no)
```

```
# We need to calculate p_hat (number of success/total observations)
# Option 1 (My preferred option - most efficient)
p_hat <- mean(atheist_yes_no)
p_hat
# Option 2
sum(atheist_yes_no)/length(atheist_yes_no)

# Construct the 95% CI
std_err <- sqrt(p_hat*(1-p_hat)/length(atheist_yes_no))
z_star <- qnorm(.975)
lb <- p_hat - z_star*std_err
ub <- p_hat + z_star*std_err
c(lb,ub)
```

Although formal confidence intervals and hypothesis tests don't show up in the report, suggestions of inference appear at the bottom of page 7: "In general, the error margin for surveys of this kind is ± 3-5% at 95% confidence".

Exercise 6: Based on the work above, what is the margin of error for the estimate of the proportion of atheists in US in 2012?

Exercise 7: Calculate confidence intervals for the proportion of atheists in 2012 in two other countries of your choice, and report the associated margins of error. Be sure to note whether the conditions for inference are met. It may be helpful to create new data sets for each of the two countries first, and then use these data sets during calculations.

8.4 How does the proportion affect the margin of error?

Imagine you've set out to survey 1000 people on two questions: are you female? and are you left-handed? Since both of these sample proportions were calculated from the same sample size, they should have the same margin of error, right? Wrong! While the margin of error does change with sample size, it is also affected by the proportion.

Think back to the formula for the standard error: $SE = \sqrt{p(1-p)/n}$. This is then used in the formula for the margin of error for a 95% confidence interval: $ME = 1.96 \times SE = 1.96 \times \sqrt{p(1-p)/n}$. Since the population proportion p is in this ME formula, it should make sense that the margin of error is in some way dependent on the population proportion. We can visualize this relationship by creating a plot of ME vs. p.

The first step is to make a vector `p` that is a sequence from 0 to 1 with each number separated by 0.01. We can then create a vector of the margin of error (`me`) associated with each of these values of `p` using the familiar approximate formula ($ME = 2 \times SE$). Lastly, we plot the two vectors against each other to reveal their relationship.

```
n <- 1000
p <- seq(0, 1, 0.01)
me <- 2 * sqrt(p * (1 - p)/n)
plot(me ~ p, ylab = "Margin of Error", xlab = "Population Proportion")
```

8. Describe the relationship between `p` and `me`.

8.5 Success-failure condition

The textbook emphasizes that you must always check conditions before making inference. For inference on proportions, the sample proportion can be assumed to be nearly normal if it is based upon a random sample of independent observations and if both $np \geq 10$ and $n(1-p) \geq 10$. This rule of thumb is easy enough to follow, but it makes one wonder: what's so special about the number 10?

The short answer is: nothing. You could argue that we would be fine with 9 or that we really should be using 11. What is the "best" value for such a rule of thumb is, at least to some degree, arbitrary. However, when np and $n(1-p)$ reaches 10 the sampling distribution is sufficiently normal to use confidence intervals and hypothesis tests that are based on that approximation.

We can investigate the interplay between n and p and the shape of the sampling distribution by using simulations. To start off, we simulate the process of drawing 5000 samples of size 1040 from a population with a true atheist proportion of 0.1. For each of the 5000 samples we compute $\hat{p}$ and then plot a histogram to visualize their distribution.

```
p <- 0.1
n <- 1040
p_hats <- rep(0, 5000)

for(i in 1:5000){
  samp <- sample(c("atheist", "non_atheist"), n, replace = TRUE, prob = c(p, 1-p))
  p_hats[i] <- sum(samp == "atheist")/n
}

hist(p_hats, main = "p = 0.1, n = 1040", xlim = c(0, 0.18))
```

These commands build up the sampling distribution of $\hat{p}$ using the familiar `for` loop. You can read the sampling procedure for the first line of code inside the `for` loop as, "take a sample of size n with replacement from the choices of atheist and non-atheist with probabilities p and $1-p$, respectively." The second line in the loop says, "calculate the proportion of atheists in this sample and record this value." The loop allows us to repeat this process 5,000 times to build a good representation of the sampling distribution.

Exercise 9: Describe the sampling distribution of sample proportions for $n = 1040$ and $p = 0.1$. Be sure to note the center, spread, and shape. *Hint:* Remember that R has functions such as `mean` to calculate summary statistics.

Exercise 10: Repeat the above simulation three more times but with modified sample sizes and proportions: for $n = 400$ and $p = 0.1$, $n = 1040$ and $p = 0.02$, and $n = 400$ and $p = 0.02$. Plot all four histograms together by running the `par(mfrow = c(2, 2))` command before creating the histograms. You may need to expand the plot window to accommodate the larger two-by-two plot. Describe the three new sampling distributions. Based on these limited plots, how does n appear to affect the distribution of $\hat{p}$? How does p affect the sampling distribution?

Once you're done, you can reset the layout of the plotting window by using the command `par(mfrow = c(1, 1))` command or clicking on "Clear All" above the plotting window (if using RStudio). Note that the latter will get rid of all your previous plots.

Exercise 11: If you refer to Table 6, you'll find that Australia has a sample proportion of 0.1 on a sample size of 1040, and that Ecuador has a sample proportion of 0.02 on 400 subjects. Let's suppose for this exercise that these point estimates are actually the truth. Then given the shape of their respective sampling distributions, do you think it is sensible to proceed with inference and report margin of errors, as the report does? – Checking our condition for at least 10 observed successes and failures.

8.6 On your own

The question of atheism was asked by WIN-Gallup International in a similar survey that was conducted in 2005. (We assume here that sample sizes have remained the same - or close enough so it doesn't really matter.) Table 4 on page 13 of the report summarizes survey results from 2005 and 2012 for 39 countries.

- Answer the following two questions using confidence intervals calculated by "hand" or by using the built in functions described in the final section of the lab. As always, write out the hypotheses for any tests you conduct and outline the status of the conditions for inference.

  a. Is there convincing evidence that Spain saw a change in its atheism index between 2005 and 2012? *Hint: Create a new datasets for respondents from Spain for 2005 and 2012. Form confidence intervals for the true proportion of athiests in both years, and determine whether they overlap.*

  b. Is there convincing evidence that the United States saw a change in its atheism index between 2005 and 2012?

- If in fact there has been no change in the atheism index in the countries listed in Table 4, in how many of those countries would you expect to detect a change (at a significance level of 0.05) simply by chance? *Hint: Look in the textbook index under Type 1 error.*

- Suppose you're hired by the local government to estimate the proportion of residents that attend a religious service on a weekly basis. According to the guidelines, the estimate must have a margin of error no greater than 1% with 95% confidence. You have no idea what to expect for p. How many people would you have to sample to ensure that you are within the guidelines? *Hint: Refer to your plot of the relationship between p and margin of error. Do not use the data set to answer this question.*

8.7 Options for using built in functions in R: `prop.test()` & `binom.test()`

As mentioned during lecture, the by "hand" calculations represent a great start for conducting inferences for proportions and difference of proportions (i.e. confidence intervals and hypothesis tests). The methods presented in the book depend upon the normal approximation to the binomial distribution – thus the reason for requiring 10 success and 10 failures (Chapter 3). In the case of a **1-sample proportion**, the true distribution is the binomial distribution and we can calculate the exact confidence interval and hypothesis test p-value using the `binom.test()` function. Computational limitations made this impractical in the past. This method is sometimes referred to as the exact test.

Another option is to utilize an alternative approximation method that works just as good and better in some instances than those presented in the book. This method is known as a goodness-of-fit test which readily extends to many situations and can be implemented using the `prop.test()` function.

The code below demonstrates appling these methods/functions to construct a 95% confidence interval for the proportion of atheists in the United States in 2012 (Exercise 5).

```
# First we need to identify the successes and failures
# We can find out what our options with the following
levels(us12$response)

# Next we define a success as "atheist" and a failure as not
# being an atheist -- code returns TRUE for atheist and FALSE
# otherwise
```

```r
atheist_yes_no <- us12$response == "atheist"

# We require the number of successess/atheists
num_success <- sum(atheist_yes_no)
# We reuire the number of total observations
num_obs <- length(atheist_yes_no)

# Exact Test
binom.test(x = num_success, n = num_obs, alternative = "two.sided")
# Goodness-of-Fit test
prop.test(x = num_success, n = num_obs, alternative = "two.sided",)
```

These methods, in most cases, **do not** result in confidence intervals that are symmetic about the point estimate. This makes defining a margin of error slightly more difficult and beyond the scope of this course.

- Compare and contrast the 95% confidence intervals for the proportion of atheists in the United States in 2012 constructed using `binom.test()`, `prop.test()`, and by "hand" calculations.

Chapter 9

Introduction to Linear Regression

This lab is structured to guide you through an organized process such that you could easily organize your code with comments — meaning your R script — into a lab report. I would suggest getting into the habit of writing an organized and commented R script that completes the tasks and answers the questions provided in the lab — including in the **Own Your Own** section.

9.1 Batter up (Getting Started)

The movie Moneyball focuses on the "quest for the secret of success in baseball". It follows a low-budget team, the Oakland Athletics, who believed that underused statistics, such as a player's ability to get on base, betterpredict the ability to score runs than typical statistics like home runs, RBIs (runs batted in), and batting average. Obtaining players who excelled in these underused statistics turned out to be much more affordable for the team.

In this lab we'll be looking at data from all 30 Major League Baseball teams and examining the linear relationship between runs scored in a season and a number of other player statistics. Our aim will be to summarize these relationships both graphically and numerically in order to find which variable, if any, helps us best predict a team's runs scored in a season.

9.2 The data

Let's load up the data for the 2011 season.

```
download.file("http://www.openintro.org/stat/data/mlb11.RData", destfile = "mlb11.RData")
load("mlb11.RData")
```

In addition to runs scored, there are seven traditionally used variables in the data set: at-bats, hits, home runs, batting average, strikeouts, stolen bases, and wins. There are also three newer variables: on-base percentage, slugging percentage, and on-base plus slugging. For the first portion of the analysis we'll consider the seven traditional variables. At the end of the lab, you'll work with the newer variables on your own.

Exercise 1: What type of plot would you use to display the relationship between `runs` and one of the other numerical variables? Plot this relationship using the variable `at_bats` as the predictor. Does the relationship look linear? If you knew a team's `at_bats`, would you be comfortable using a linear model to predict the number of runs?

If the relationship looks linear, we can quantify the strength of the relationship with the correlation coefficient.

```
cor(mlb11$runs, mlb11$at_bats)
```

9.3 Sum of squared residuals

Think back to the way that we described the distribution of a single variable. Recall that we discussed characteristics such as center, spread, and shape. It's also useful to be able to describe the relationship of two numerical variables, such as `runs` and `at_bats` above.

Exercise 2: Looking at your plot from the previous exercise, describe the relationship between these two variables. Make sure to discuss the form, direction, and strength of the relationship as well as any unusual observations.

Just as we used the mean and standard deviation to summarize a single variable, we can summarize the relationship between these two variables by finding the line that best follows their association. Use the following interactive function to select the line that you think does the best job of going through the cloud of points.

```
plot_ss(x = mlb11$at_bats, y = mlb11$runs)
```

After running this command, you'll be prompted to click two points on the plot to define a line. Once you've done that, the line you specified will be shown in black and the residuals in blue. Note that there are 30 residuals, one for each of the 30 observations. Recall that the residuals are the difference between the observed values and the values predicted by the line:

$$e_i = y_i - \hat{y}_i$$

The most common way to do linear regression is to select the line that minimizes the sum of squared residuals. To visualize the squared residuals, you can rerun the plot command and add the argument `showSquares = TRUE`.

```
plot_ss(x = mlb11$at_bats, y = mlb11$runs, showSquares = TRUE)
```

Note that the output from the `plot_ss` function provides you with the slope and intercept of your line as well as the sum of squares.

Exercise 3: Using `plot_ss`, choose a line that does a good job of minimizing the sum of squares. Run the function several times. What was the smallest sum of squares that you got? How does it compare to your neighbors?

9.4 The linear model

It is rather cumbersome to try to get the correct least squares line, i.e. the line that minimizes the sum of squared residuals, through trial and error. Instead we can use the `lm` function in R to fit the linear model (a.k.a. regression line).

```
m1 <- lm(runs ~ at_bats, data = mlb11)
```

The first argument in the function `lm` is a formula that takes the form `y ~ x`. Here it can be read that we want to make a linear model of `runs` as a function of `at_bats`. The second argument specifies that R should look in the `mlb11` data frame to find the `runs` and `at_bats` variables.

The output of `lm` is an object that contains all of the information we need about the linear model that was just fit. We can access this information using the summary function.

```
summary(m1)
```

Let's consider this output piece by piece. First, the formula used to describe the model is shown at the top. After the formula you find the five-number summary of the residuals. The "Coefficients" table shown next is key; its first column displays the linear model's y-intercept and the coefficient of `at_bats`. With this table, we can write down the least squares regression line for the linear model:

$$\widehat{runs} = -2789.2429 + 0.6305 * (at\_bats)$$

One last piece of information we will discuss from the summary output is the Multiple R-squared, or more simply, R^2. The R^2 value represents the proportion of variability in the response variable that is explained by the explanatory variable. For this model, 37.3% of the variability in runs is explained by at-bats.

Exercise 4: Fit a new model that uses `homeruns` to predict `runs`. Using the estimates from the R output, write the equation of the regression line. What does the slope tell us in the context of the relationship between success of a team and its home runs?

9.5 Prediction and prediction errors

Let's create a scatterplot with the least squares line laid on top.

```
plot(mlb11$runs ~ mlb11$at_bats)
abline(m1)
```

The function `abline` plots a line based on its slope and intercept. Here, we used a shortcut by providing the model `m1`, which contains both parameter estimates. This line can be used to predict y at any value of x. When predictions are made for values of x that are beyond the range of the observed data, it is referred to as *extrapolation* and is not usually recommended. However, predictions made within the range of the data are more reliable. They're also used to compute the residuals.

Exercise 5: If a team manager saw the least squares regression line and not the actual data, how many runs would he or she predict for a team with 5,578 at-bats? Is this an overestimate or an underestimate, and by how much? In other words, what is the residual for this prediction?

9.6 Model diagnostics

To assess whether the linear model is reliable, we need to check for (1) linearity, (2) nearly normal residuals, and (3) constant variability.

Linearity: You already checked if the relationship between runs and at-bats is linear using a scatterplot. We should also verify this condition with a plot of the residuals vs. at-bats. Recall that any code following a # is intended to be a comment that helps understand the code but is ignored by R.

```
plot(m1$residuals ~ mlb11$at_bats)
abline(h = 0, lty = 3)  # adds a horizontal dashed line at y = 0
```

Exercise 6: Is there any apparent pattern in the residuals plot? What does this indicate about the linearity of the relationship between runs and at-bats?

Nearly normal residuals: To check this condition, we can look at a histogram

```
hist(m1$residuals)
```

or a normal probability plot of the residuals. Consider simulating normal probability plots for an adequate benchmark for determining suitability of the normality assumption.

```
qqnorm(m1$residuals)
qqline(m1$residuals)  # adds diagonal line to the normal prob plot
```

Exercise 7: Based on the histogram and the normal probability plot, does the nearly normal residuals condition appear to be met?

Constant variability: We can check the constant variability assumption by plotting the residuals (standardized) vs the model's fitted-values. If there should be no clear relationship concerning the disbursement of points around a horizontal line at 0 (since residuals are scaled), then we conclude that the constant variability assumption is appropriate.

```
std_residuals <- scale(m1$residuals) # standardizes residuals
plot(x = m1$fitted.values, y = std_residuals)
abline(h = 0, lty = "dashed") # adds dashed horizontal line
```

Exercise 8: Based on the plot, does the constant variability condition appear to be met?

9.7 On Your Own

- Choose another traditional variable from `mlb11` that you think might be a good predictor of `runs`. Produce a scatterplot of the two variables and fit a linear model. At a glance, does there seem to be a linear relationship?
- How does this relationship compare to the relationship between `runs` and `at_bats`? Use the R^2 values from the two model summaries to compare. Does your variable seem to predict `runs` better than `at_bats`? How can you tell?
- Now that you can summarize the linear relationship between two variables, investigate the relationships between `runs` and each of the other five traditional variables. Which variable best predicts `runs`? Support your conclusion using the graphical and numerical methods we've discussed (for the sake of conciseness, only include output for the best variable, not all five).

- Now examine the three newer variables. These are the statistics used by the author of *Moneyball* to predict a teams success. In general, are they more or less effective at predicting runs that the old variables? Explain using appropriate graphical and numerical evidence. Of all ten variables we've analyzed, which seems to be the best predictor of `runs`? Using the limited (or not so limited) information you know about these baseball statistics, does your result make sense?
- Check the model diagnostics for the regression model with the variable you decided was the best predictor for runs.

Chapter 10

Multiple Linear Regression

This lab is structured to guide you through an organized process such that you could easily organize your code with comments — meaning your R script — into a lab report. We would suggest getting into the habit of writing an organized and commented R script that completes the tasks and answers the questions provided in the lab — including in the **Own Your Own** section.

10.1 Getting Started

Recall that we explored simple linear regression by examining baseball data from the 2011 Major League Baseball (MLB) season. We will also use this data to explore multiple regression. Our inspiration for exploring this data stems from the movie Moneyball, which focused on the "quest for the secret of success in baseball". It follows a low-budget team, the Oakland Athletics, who believed that underused statistics, such as a player's ability to get on base, better predict the ability to score runs than typical statistics like home runs, RBIs (runs batted in), and batting average. Obtaining players who excelled in these underused statistics turned out to be much more affordable for the team.

In this lab we'll be looking at data from all 30 Major League Baseball teams and examining the linear relationship between runs scored in a season and a number of other player statistics. Our aim will be to find the model that best predicts a team's runs scored in a season. We also aim to find the model that best predicts a team's total wins in a season. The first model would tell us which player statistics we should pay attention to if we wish to purchase runs and the second model would indicate which player statistics we should utilize when we wish to purchase wins.

10.2 The data

Let's load up the data for the 2011 season.

```
download.file("http://www.openintro.org/stat/data/mlb11.RData", destfile = "mlb11.RData")
load("mlb11.RData")
```

In addition to runs scored, there are seven traditionally used variables in the data set: at-bats, hits, home runs, batting average, strikeouts, stolen bases, and wins. There are also three newer variables: on-base percentage, slugging percentage, and on-base plus slugging. For the first portion of the analysis we'll consider the seven traditional variables. At the end of the lab, you'll work with the newer variables on your own.

We also would like to modify the data so that it easier to work with during model selection. We remove the variable team from the dataset and store the updated verison in `mlb11_wins`.

```
mlb11_wins <- mlb11[,-1] # since team is in the first column we can us -1 to remove it
```

Since `wins` is not a player level statistic - at least for non-pitchers - we do not want to use it when predicting `runs`. Therefore we are going to create another modified dataset to utilize when attempting to find the best model for predicting the total number of `runs` for a team during a season. The reverse is not an issue when attempting to predict a team's number of wins for a season - `runs` can be used to predict `wins`.

```
mlb11_runs <- mlb11_wins[,-8] # since wins is in the 8th column we can us -8 to remove it
```

As discussed in class there are many ways to go about model selection. We will look at both forward and backward selection methods that utilize

10.3 The search for the best model

As discussed in class there are many ways to go about model selection. We will look at both forward and backward selection methods that utilize different criterions (R^2_{adj}, p-values, or AIC).

10.3.1 Predicting runs with backward selection

The first step in backward selection is to define a full model. Since we created a modified dataset for predicting runs we can use a shortcut, `runs ~ .`, for telling R to use all remaining variables to predict runs.

```
runs_full <- lm(runs ~ ., data = mlb11_runs)
summary(runs_full)
```

Exercise 1: How many variables are being used to predict `runs` in the full model? How many parameters are being estimated in the full model? How many of the parameters are significantly different than 0 at the 0.05 level? What is the full model's R^2_{adj}?

Now that we have a full model defined we can go about backwards model selection. The `step()` function in R makes it extremely easy to use AIC (Akiake's Information Criterion) for model selection. Similar to R^2_{adj}, AIC applies a penalty to models using more predictor variables. Run the following code to determine the best model for predicting a team's runs in a season using backward selection with AIC as the criterion (note that lower AIC indicates a better model).

```
runs_backAIC <- step(runs_full, direction = "backward")
summary(runs_backAIC)
```

Exercise 2: How many steps did the backward selection using AIC conduct before selecting a model? Which variable was the first to be removed? Which variables ended up in the final model? How many parameters are being estimated in this final model? How many of the parameters in this final model are significantly different than 0 at the 0.05 level? Does this final model have a higher R^2_{adj} than the full model for runs?

Instead of AIC, let's use R^2_{adj} as our criterion when conducting backward selection. Remember that R^2_{adj} indicates a better model.

```
## Step 1
summary(lm(runs ~ . , mlb11_runs))$adj.r
summary(lm(runs ~ . - at_bats, mlb11_runs))$adj.r
summary(lm(runs ~ . - hits, mlb11_runs))$adj.r
```

```
summary(lm(runs ~ . - homeruns, mlb11_runs))$adj.r
summary(lm(runs ~ . - bat_avg, mlb11_runs))$adj.r
summary(lm(runs ~ . - strikeouts, mlb11_runs))$adj.r
summary(lm(runs ~ . - stolen_bases, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_slug, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_obs, mlb11_runs))$adj.r
```

Since the model which removed `new_onbase` has the highest R^2_{adj} we move onto step 2 using that model and continue by removing one variable at a time and calculate the new R^2_{adj} for each model.

```
## Step 2
summary(lm(runs ~ . - new_onbase, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - at_bats, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - hits, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - homeruns, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - bat_avg, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - strikeouts, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - stolen_bases, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - new_slug, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - new_obs, mlb11_runs))$adj.r
```

Since the model in step 2 that removed `strikeouts` has the highest R^2_{adj} we move onto step 3 using the model that now has both `new_onbase` and `strikeouts` removed and continue by removing one variable at a time and calculating the new R^2_{adj} for each model.

```
## Step 3
summary(lm(runs ~ . - new_onbase - strikeouts, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - strikeouts - at_bats, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - strikeouts - hits, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - strikeouts - homeruns, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - strikeouts - bat_avg, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - strikeouts - stolen_bases, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - strikeouts - new_slug, mlb11_runs))$adj.r
summary(lm(runs ~ . - new_onbase - strikeouts - new_obs, mlb11_runs))$adj.r
```

Since none of the models which remove one more additional variable from the model that already excludes both `new_onbase` and `strikeouts` have larger R^2_{adj}, we stop the process and now have a final model. In the code below we store the final model and look at a summary of the final model.

```
runs_backADJr <- lm(runs ~ . - new_onbase - strikeouts, mlb11_runs)
summary(runs_backADJr)
```

3. Which variables ended up in the final model when using backward selection with R^2_{adj}? How many parameters are being estimated in this final model? How many of the parameters in this final model are significantly different than 0 at the 0.05 level? Does this final model have a higher R^2_{adj} than the full model for runs? Higher than the final model when using backward selection with AIC?

Finally let's use a p-value method with a 0.05 significance level as our criterion when conducting backward selection. Remember that higher p-values are bad so we remove a variable when if it has the highest p-value which is greater than 0.05. If all p-values are less than 0.05 then we stop and we have arrived at our final model. Fortunately, R does have a function that makes this easier whish is `drop1()` - `add1()` is for forward selection. We must input the model fit and then indicate that `test = "F"` so p-values are printed.

```
## Step 1
drop1(lm(runs ~ . , mlb11_runs), test = "F")
## Step 2
```

```
drop1(lm(runs ~ . - new_onbase, mlb11_runs), test = "F")
## Step 3
drop1(lm(runs ~ . - new_onbase - strikeouts, mlb11_runs), test = "F")
## Step 4
drop1(lm(runs ~ . - new_onbase - strikeouts - homeruns, mlb11_runs), test = "F")
## Step 5
drop1(lm(runs ~ . - new_onbase - strikeouts - homeruns - at_bats, mlb11_runs), test = "F")
## Step 6
drop1(lm(runs ~ . - new_onbase - strikeouts - homeruns - at_bats - hits, mlb11_runs), test = "F")
## Step 7
drop1(lm(runs ~ . - new_onbase - strikeouts - homeruns - at_bats - hits  - bat_avg, mlb11_runs), test =
## Step 8
drop1(lm(runs ~ . - new_onbase - strikeouts - homeruns - at_bats - hits  - bat_avg - new_slug, mlb11_ru
```

Below we store the final model selected under this method and examinie it using `summary()`.

```
## Final model using backward selection with p-value criterion
runs_backPval <- lm(runs ~ . - new_onbase - strikeouts - homeruns - at_bats - hits  - bat_avg - new_slu
summary(runs_backPval)
```

Exercise 4: Which variables ended up in the final model using a p-value method with a 0.05 significance level as our criterion when conducting backward selection? How many parameters are being estimated in this final model? How many of the parameters in this final model are significantly different than 0 at the 0.05 level? Does this final model have a higher R^2_{adj} than the full model for runs? Why might someone prefer this final model over all of the models thus far?

10.3.2 Predicting runs with forward selection

The first step in forward selection is to set up a null/base model to build up from. This model could include variables that researchers stipulate a model must have for theoretical reasons. No such variables exisit in our case which means our null model will only have the intercept in it. We must also specify the full model so the preocedure knows which models to attempt. Note that the full model will be athe same as in backward selection.

```
## Creating the null model for runs
runs_null <- lm(runs ~ 1, data = mlb11_runs)
summary(runs_null)
```

Now that we have a null model defined we can go about forward model selection. Once again we will us the `step()` function in R to use AIC (Akiake's Information Criterion) for model selection - remember that lower AIC indicates a better model.

```
runs_forwardAIC <- step(runs_null, direction = "forward", scope = formula(runs_full))
summary(runs_forwardAIC)
```

Exercise 5: How many steps did the forward selection using AIC conduct before selecting a model? Which variable was the first to be added? Which previous selection method does this agree with - this doesn't always happen?

Instead of AIC, let's use R^2_{adj} as our criterion when conducting forward selection. Remember that R^2_{adj} indicates a better model.

```
## Step 1
summary(lm(runs ~ 1 , mlb11_runs))$adj.r
summary(lm(runs ~ + at_bats, mlb11_runs))$adj.r
summary(lm(runs ~ + hits, mlb11_runs))$adj.r
summary(lm(runs ~ + homeruns, mlb11_runs))$adj.r
summary(lm(runs ~ + bat_avg, mlb11_runs))$adj.r
summary(lm(runs ~ + strikeouts, mlb11_runs))$adj.r
summary(lm(runs ~ + stolen_bases, mlb11_runs))$adj.r
summary(lm(runs ~ + new_onbase, mlb11_runs))$adj.r
summary(lm(runs ~ + new_slug, mlb11_runs))$adj.r
summary(lm(runs ~ + new_obs, mlb11_runs))$adj.r
```

Since the model that added `new_obs` has the highest R^2_{adj} we move onto step 2 using that model and continue by adding one variable at a time and calculate the new R^2_{adj} for each model.

```
## Step 2
summary(lm(runs ~ new_obs, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + at_bats, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + hits, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + homeruns, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + bat_avg, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + strikeouts, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + stolen_bases, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + new_onbase, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + new_slug, mlb11_runs))$adj.r
```

Since the model is step 2 that added `stolen_bases` has the highest R^2_{adj} we move onto step 3 using that model and continue by adding one variable at a time and calculate the new R^2_{adj} for each model.

```
## Step 3
summary(lm(runs ~ new_obs + stolen_bases, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + stolen_bases + at_bats, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + stolen_bases + hits, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + stolen_bases + homeruns, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + stolen_bases + bat_avg, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + stolen_bases + strikeouts, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + stolen_bases + new_onbase, mlb11_runs))$adj.r
summary(lm(runs ~ new_obs + stolen_bases + new_slug, mlb11_runs))$adj.r
```

Since none of the models that added one more variable in step 3 resulted in an increased R^2_{adj}, we stop the process and now have a final model. In the code below we store the final model and look at a summary of the final model.

```
runs_forwardADJr <- lm(runs ~ new_obs + stolen_bases, mlb11_runs)
summary(runs_forwardADJr)
```

Exercise 6: Does the final model selected using forward selection with R^2_{adj} differ from the final model using forward selection with AIC?

Finally let's use a p-value method with a 0.05 significance level as our criterion when conducting forward selection. Remember that hlower p-values are consider better so we add a variable when it has the lowest p-value and is less than 0.05. If newly added variable is not less than 0.05 then we stop and conclude that we have arrived at our final model. Fortunately, R does have a function that makes this easier which is `add1()`.

We must input model fit, the possible full model, and then indicate that the `test = "F"` so p-values are printed.

```
## Step 1
add1(lm(runs ~ 1 , mlb11_runs), test = "F", scope = formula(runs_full))
## Step 2
add1(lm(runs ~ new_obs , mlb11_runs), test = "F", scope = formula(runs_full))
## Step 3
add1(lm(runs ~ new_obs + stolen_bases, mlb11_runs), test = "F", scope = formula(runs_full))
```

Below we store the final model selected under this method and examinie it using `summary()`.

```
## Final model using backward selection with p-value criterion
runs_forwardPval <- lm(runs ~ new_obs + stolen_bases, mlb11_runs)
summary(runs_forwardPval)
```

Exercise 7: What do you note about the final model selected using forward selection with p-values? This does not always occur.

10.4 Assessing the conditions

After conducting a model selection procedures we should conduct graphical checks to explore wheather our conditions for multiple regression are being met. R has built in command for basic diagnostic plots. Simply use the `plot()` function and input the model that you desire diagnostic plots for as demonstrated in the code below.

```
plot(runs_backAIC)
```

None of the models have and severe departures from our necessary conditions for multiple regression. Diagnostic methods can be tricky and to really get a good understanding of them will require either further self-study or taking a regression course.

10.5 On Your Own

- Using all the available variables in our dataset, conduct backward selection using AIC to select the best model for predicting `wins` for a team in a single season. **Hints: make sure you are using the `mlb11_wins` dataset.** How many variables are being used to predict `wins` in the full model? Which variables are included in the this final model?
- Construct a 95% confidence interval for one of the slopes in the final model from part 1.
- Conduct backward selection using R^2_{adj} and then p-value method to select the best model for predicting `wins`. Do either of the final models selected using these criterions match the final model selected using backward selection with AIC? Do they match eachother?
- Conduct forward selection with the three different criterions we have been using to select the best model for predicting `wins` for each. Are they all the same? Different?