

Statistical Thinking: Week 1 Lab

Due 12noon Monday 10 August 2020

1. Introduction

This week's Lab provides a hands-on introduction to **R** and **RStudio**. As with many other labs you will do over the semester, you will need to think about how to manipulate a given set of data, how to best visualise it, and to think about what you can learn from it. At the beginning, we focus mostly on programming aspects, with the statistical concepts given less emphasis. (However, as we progress, this emphasis will shift.)

Learning Goals for Week 1

- Learn how to set up **R** and **RStudio** on your own device.
- Learn to install and load **R** packages.
- Learn what are **R Markdown** files and reproducible research.
- Learn what is 'the tidyverse'.
- Learn some basic **R** commands to manipulate and plot data.

If you are already a confident user of **R** and **RStudio**, this lab will be straightforward. You will finish quickly, and can use your time to expand your capabilities through a new DataCamp module, or investigate more deeply one of the many other resources noted in the Detailed Unit Information document.

If you are an **R** beginner, take your time to work through each part of this lab. Perhaps make a list of the different packages or commands as you learn them, and note on which lab or other reference on which they appear so that you can find them again when you need to do something similar later on.

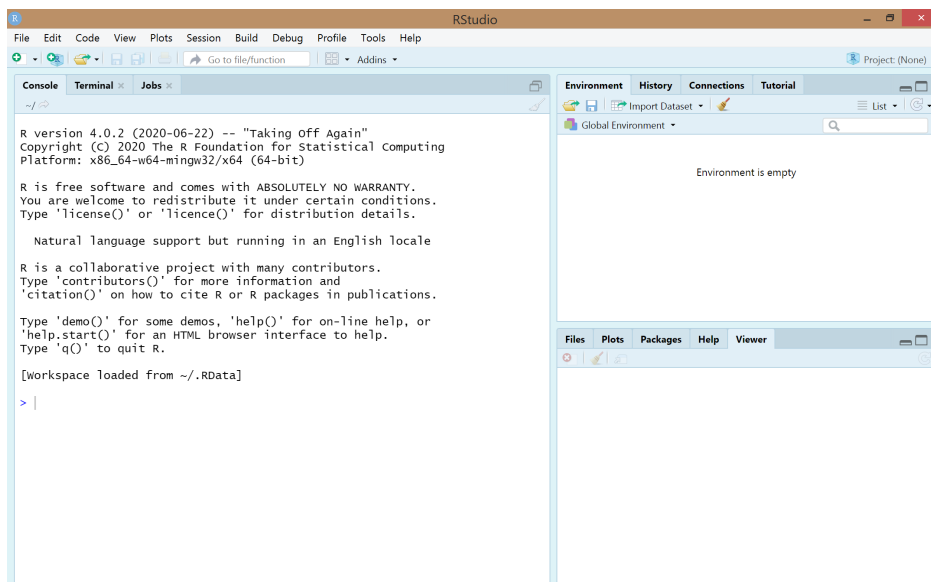
For everyone, as the labs progress, you are encouraged to explore beyond what the labs dictate; a willingness to experiment will make you a much better programmer. It will also give you the space to be able to think beyond the code, and to appreciate that having a computational perspective can really help you to *understand* data and the relevance (and limitations!) of modern statistics/econometrics/actuarial techniques.

Once you have both **R** and **RStudio** installed, you are ready to start the Lab activities.¹ So let's begin with the fundamental building blocks of **R** and **RStudio**: the interface, reading in data, and some basic commands.

2. Launch RStudio

Go ahead and launch RStudio. You should see a window that looks like the image shown below.

¹This portion of the lab is based on materials provided by the writers of the prescribed textbook, *Introductory Statistics with Randomization and Simulation*.



2.1 Packages

For this lab, we will use the following **R** packages, named *tidyverse* and *openintro*. Information about these two packages are given below.

The tidyverse package

The *tidyverse* package is actually a suite of packages, includes those that include function for data wrangling and data visualisation. These packages share common philosophies and are designed to work together. Over time the packages contained within the tidyverse has changed. You can find more about the packages in the tidyverse at <https://www.tidyverse.org/>.

The openintro package

The *openintro* package aligns with our prescribed textbook, and contains data and custom functions to enable you to, among other things, reproduce some of the analyses presented in the text. In this lab, we will use a special *OpenIntro Statistics R Markdown* template, and also will explore one of the datasets, named *arbuthnot*, contained in the *openintro* package. More information on these items is given below, but first let's install the packages and load them into the current **R** session.

2.2 Install the packages

If these two packages are not already available in your **R** environment, install them by typing the following two lines of code into the *Console* pane of your **RStudio** session, pressing the enter/return key after each one. (Note: You may need to select a server from which to download; any of the servers should work.)

```
> install.packages("tidyverse")
> install.packages("openintro")
```

Find these package names listed in the **RStudio Packages** pane. Notice a description of the package and the installed version number is also listed.

2.3 Load the package libraries

Next, you need to load these packages in your current session. We do this with the `library()` function. Run the following three lines in your *Console* pane:

```
> library(tidyverse)
> library(openintro)
```

Usually a package will only need to be installed on to your machine once, but you need to load them each time you start a new **RStudio** session.

3. R Markdown

Throughout the semester, we will use **R Markdown** files, which have an “.Rmd” extension, to create *reproducible* lab reports and assignments. Watch this short video (2:02 mins) to find out why:

Why use R Markdown for Lab Reports?

3.1 Use the OpenIntro Statistics R Markdown template

For this lab you will use a special **R Markdown** template from the *openintro* package. Follow these steps in **RStudio**:

- Go to: File -> New File -> R Markdown...
- Choose: Template
- Choose: Lab Report for OpenIntro Statistics Lab 1
- Click: OK

The **R Markdown** template should open in the *Editor* pane. The template provides spaces for you to answer a series of exercises (questions) we will soon get to below. We'll work with a data file named *arbuthnot* which is contained in the *openintro* package.

Before we get on to those exercises, let's be sure to change the title and save the template as your own file (and with a filename suitable for your lab submission), and learn how to *render* it to produce the final report.

3.2 Name and save your R Markdown file

To change the title of your report, replace the **Author name** inside the double quotation marks on line 3 of the template file with **your name** (both first and last) and your Monash **student ID** number, all on the same line.

Then save your file, this time using the format: **Lab01_IDnnnnnnnnn.Rmd**

with your Monash student ID number replacing the segment **nnnnnnnnn** in the file name.

3.3 How to complete the lab

Now continue working through the rest of this document. When you come across a numbered Exercise, put your answers in relevant place in the **Lab01_IDnnnnnnnn.Rmd** file.

Please be sure that the answers you provide to each question is clearly labelled, and that the corresponding code chunk is also clearly identified.


While you are working through the lab, you may want to compute quantities or create plots that are not directly related to an exercise. We suggest that you keep the full “working” version of your file first, with an alternative file name, such as **Lab01_mywork.Rmd**, then re-save it as **Lab01_IDnnnnnnnn.Rmd** to delete the element not required for submission.

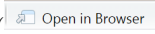
At the end there will be further instructions about how to complete your submission, which will include uploading your completed **Lab01_IDnnnnnnnn.Rmd** file.

3.4 Install the knitr package

You can render your **R Markdown** file and produce either a print (.docx or .pdf) or web browser (.html) version your report, using the *knitr* package. Install this package using the same approach you used for the *tidyverse* and *openintro* packages.



This time you may see a warning message, recommending that you let **RStudio** automatically restart **R** installing the package. Answer **Yes** to the question: *Do you want to restart R prior to installing?*

Once the installation is finished, click on the *Knit* button () , which you will see under the name of your **R Markdown** file at the top of the *Editor* pane.

The template from the *openintro* package will only permit² you to render your file to the .html format. So when you press the *Knit* button, an **R viewer** (a browser-like window) will pop-up to display your rendered report. You can open your rendered .html in your machine’s default web-browser by clicking on the *Open in browser* button () , shown below and at the top left corner of the **R Viewer** window.

Running code chunks within the Editor pane

If you you want to check whether a line of **R** code in a code chunk produces the desired result, you can:

- Run the entire chunk, by clicking on the green sideways triangle button () in the top right corner of the code chunk, or
- Run selected code in the chunk, by highlighting the relevant lines of code and then clicking the button marked **Run** () on the top right corner of the *Editor* pane.³

There are other options for running parts or all of your **R Markdown** code, which can see under the

We’re all set up now and ready to start looking at some data!

²In most other instances throughout the semester, you will be creating your own **R Markdown** files, without starting from this *OpenIntro* template. In that case, you will have a choice to render to an alternative format.

³If at any point you need to refresh all of your calculations to that point, you can *Run All Chunks Above* the chunk you’re working in by clicking on the down arrow in the code chunk.

4 Dr. Arbuthnot's Baptism Records

John Arbuthnot was a Scottish physician. In 1710 he published a paper⁴ that examined birth⁵ records in London for each of the 82 years from 1629 to 1710. The data from Arbuthnot's study is contained in a data frame⁶ named *arbuthnot*. This tibble contains three series, named *Year*, *Males* and *Females*. As the names suggest, the “Year” variable indicates the year corresponding to counts of the other two variables, with *Males* being the number of males born in London in the given year and *Females* being the number of females born in the same year.

4.1 Load Arbuthnot's dataset

Typically we would need to *load the dataset* into the active session. To do this, we can add the following line `data("arbuthnot")`

to the bottom of the “load-packages” code chunk. This can be found line 11 of your **Lab01_IDnnnnnnnnn.Rmd** file, above the final line of the code chunk that contains the three backtick marks, as follows:

```
8 {r load-packages, message=FALSE}
9 library(tidyverse)
10 library(openintro)
11 data("arbuthnot")
12 }
```

Note that Arbuthnot's data is already loaded in the template

However, it should be noted that the *openintro* package template for Lab 1 already has the *arbuthnot* data loaded, though it does not make this explicit. Similarly, the second data file you will need for the Exercises, named *present*, is also already loaded into the template.

4.2 The Viewer pane

If you want to take a quick peek at the *arbuthnot* dataset, you can run the “load-packages” code chunk. This will cause the dataset to load into your current **R** session. View the *arbuthnot* tibble object by clicking on it in the *Environment* pane. The first time you click on it, the first ten rows will print in your *Console* window. Click on the *arbuthnot* tibble name a second time and the data frame will load into a *Viewer* pane in the upper left portion of the **RStudio** environment, next to the *Editor* pane(s).

The *arbuthnot* tibble contains 82 rows and 3 columns. While you will be able to see four columns of numbers in the *Viewer* pane, the first column is not actually part of the tibble. Each row represents a different year, shown in the first column of the tibble, and the second and third columns contain the numbers of boys and girls, respectively, baptised the year corresponding to the entry in column 1. Use the scrollbar on the right-hand side of the *Viewer* pane to examine the complete data set.

The format of the tibble is important. Each of the column entries in a given row relate to the same year, so the 82 rows correspond to 82 distinct *cases*. The three columns represent three different attributes of a case, or *variables*, namely the *year*, the number of *boys* baptised and the number of *girls* baptised.

You can close the data viewer by clicking on the “×” in the upper left-hand corner.

⁴Arbuthnot, J. (1710). “An argument for divine providence, taken from the constant regularity observed in the births of both sexes,” *Philosophical Transactions of the Royal Society*, Vol 27, pp 186–190.

⁵Actually the data contain counts of baptisms (by gender and by year), but these are used as a proxy for the number of births.

⁶The *arbuthnot* data frame is a *tibble* (`tbl_df`) object. This type of data frame comes from the *tibble* package, which is part of the “tidyverse”. For information about the tibble package, see Chapter 10, R for Data Science.

4.3 The broom::glimpse command

Another way to take a look at this tibble is to use the *glimpse* function:

```
glimpse(arbuthnot)
```

This function comes from the *broom* package, which is included in the tidyverse. To try it out, just type the command above into your *Console* pane.

The *glimpse* function does not change the format of the tibble, it merely uses the tibble as an input, and prints out an alternative format without modifying the tibble itself.

You could also include the above line of code in your **Lab01_IDnnnnnnnnn.Rmd** file, inside a code chunk and positioned below the command that loads the dataset. If you decide later not to execute the command when you render your Lab01.Rmd file, simply comment out the line by inserting a hashtag symbol (#) in front of it, for example:

```
library(tidyverse)
```

5. More questions

Let's now do some more questions using the *arbuthnot* tibble.

5.1 Print a variable

We can access a variable contained in a single column of a tibble (or any type of data frame) using command such as:

```
arbuthnot$boys
```

```
## [1] 5218 4858 4422 4994 5158 5035 5106 4917 4703 5359 5366 5518 5470 5460 4793
## [16] 4107 4047 3768 3796 3363 3079 2890 3231 3220 3196 3441 3655 3668 3396 3157
## [31] 3209 3724 4748 5216 5411 6041 5114 4678 5616 6073 6506 6278 6449 6443 6073
## [46] 6113 6058 6552 6423 6568 6247 6548 6822 6909 7577 7575 7484 7575 7737 7487
## [61] 7604 7909 7662 7602 7676 6985 7263 7632 8062 8426 7911 7578 8102 8031 7765
## [76] 6113 8366 7952 8379 8239 7840 7640
```

This command will only show the number of boys baptised each year. The dollar sign (\$) selects the column of the data frame according to the name that follows it. It basically says “go to the data frame that comes before me, and find the variable that comes after me”. That is, the dollar sign before a variable name selects the column of the data frame corresponding to the variable name.

Notice the way **R** has printed these data. 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, and instead 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-hand 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 [46] starts a line, then that would mean the first number on that line would represent the 46th entry in the vector.

Exercise 1

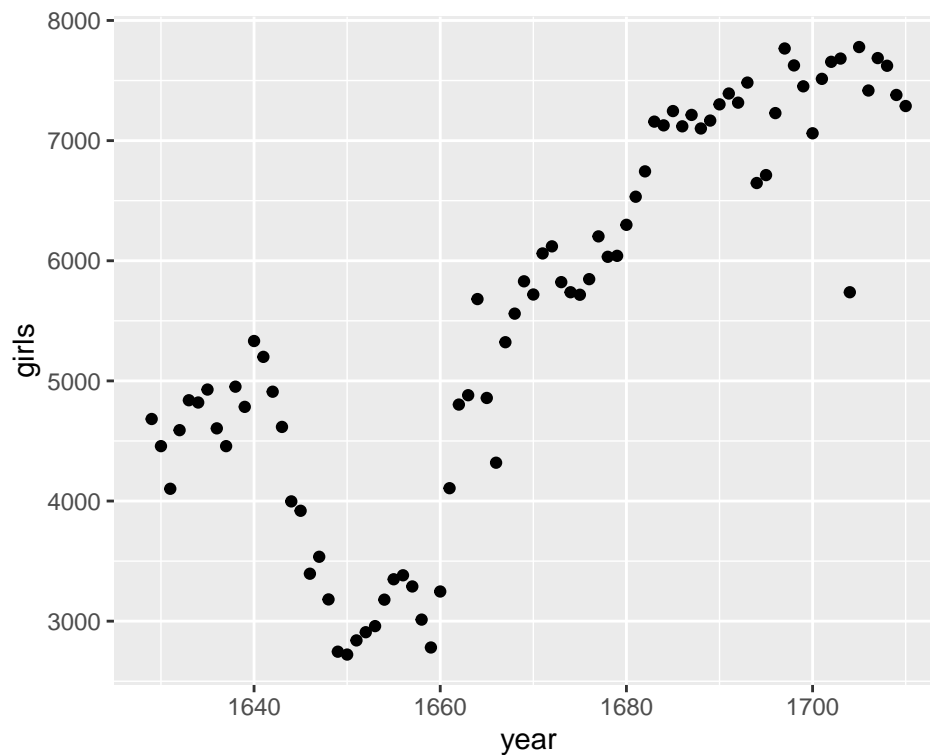
1. What line of **R** code would you use to extract the vector containing just the counts of girls baptised?

(You probably noticed the template completed the code chunk for you! You can add some text to say what has been printed. Then you will need to complete the remaining questions.)

5.2 Data visualisation

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

```
ggplot(data = arbutnot, aes(x = year, y = girls)) +  
  geom_point()
```



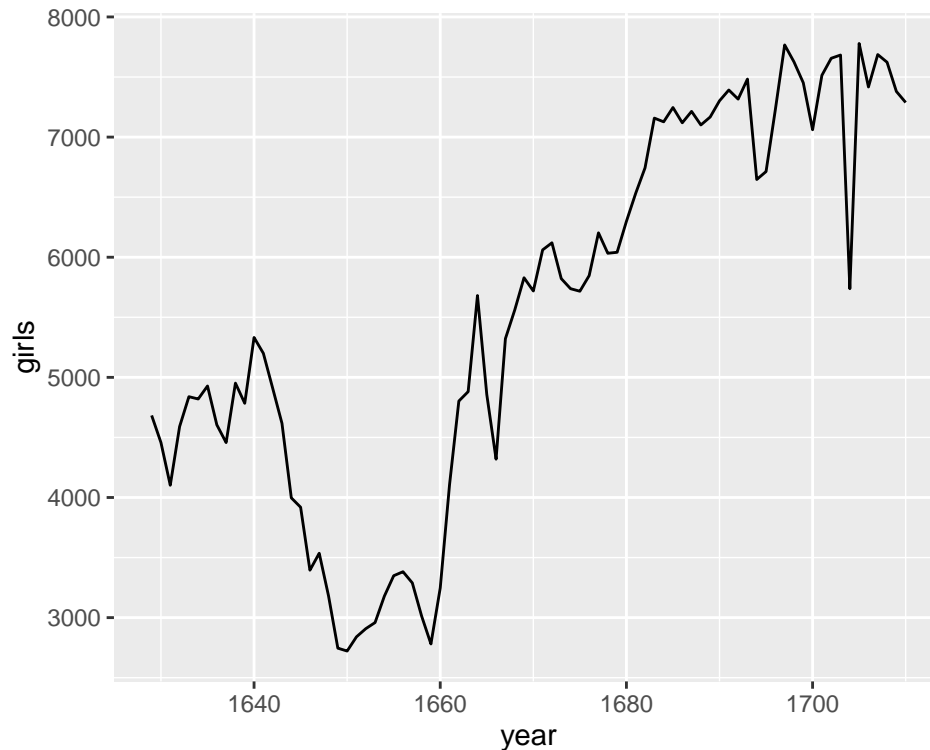
We use the `ggplot()` function to build plots. If you run the plotting code in your *Console*, you should see the plot appear under the Plots tab of the lower right panel of RStudio. Notice that the command above again looks like a function, this time with arguments separated by commas.

With `ggplot()`:

- The first argument is always the dataset.
- Next, you provide the variables from the dataset to be assigned to aesthetic elements of the plot, e.g. the x and the y axes.
- Finally, you use another layer, separated by a plus symbol (+), to specify the geometric object for the plot. In this case, since we want to produce a scatterplot, we used `geom_point()`.

For instance, if you wanted to visualize the above plot using a line graph, you would replace `geom_point()` with `geom_line()`, as shown below.

```
ggplot(data = arbuthnot, aes(x = year, y = girls)) +  
  geom_line()
```



You might wonder how you are supposed to know the syntax for the `ggplot()` function. Thankfully, **R** documents all of its functions extensively. To learn what a function does and its 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 in your *Console*:

```
?ggplot
```

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

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

5.3 R as a big calculator

Now, suppose we want to plot the total number of baptisms, over time. To compute the *total* series, 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 to that of girls, **R** will compute all sums simultaneously.

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

When you run this command, you will see it prints a vector containing 82 numbers, each one representing the sum we're after for each year in the data set. (Take a look at a few of them and verify that they are right.)

However, the command above doesn't save the *total* result. We could define a new object, using the command:

```
total <- arbuthnot$boys + arbuthnot$girls
```

However, the vector named *total* created in the above code chunk would be completely separate from the original *arbuthnot* tibble. It would be more convenient to have the constructed *total* variable as another column in *arbuthnot* tibble.

5.4 Adding a new variable to the tibble

As we'll be using this new *total* vector to generate some plots, we'll want to save it as a permanent column in our *arbuthnot* tibble. We do this using the `mutate()` command, as follows:

```
arbuthnot <- arbuthnot %>%  
  mutate(total = boys + girls)
```

The `%>%` operator used in the code chunk above is called the “piping operator”. It takes the output of the previous expression (the expression to the left of `%>%`) and “pipes” it into the first argument of the function that follows it.

To continue our analogy with mathematical functions, $x \%>\% f(y)$ is equivalent to $f(x, y)$.

A note on piping

Note that we can read these two lines of code as the following:

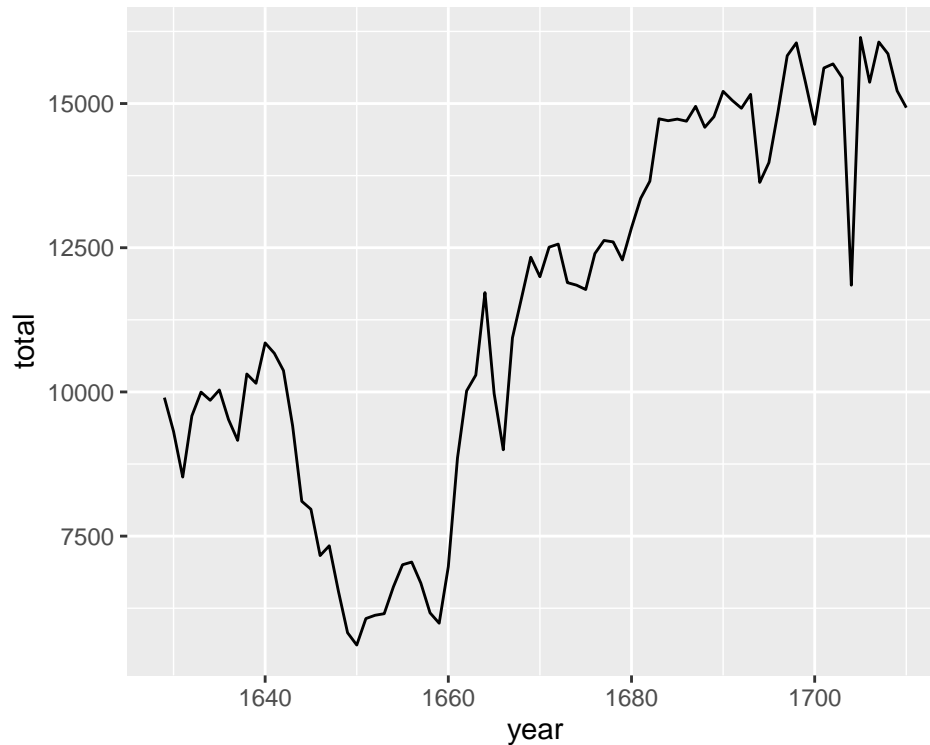
“Take the *arbuthnot* dataset and pipe it into the *mutate* function. Mutate the *arbuthnot* tibble by creating a new variable, called *total*, that is the sum of the variables called *boys* and *girls*. Then assign the resulting tibble to the object called *arbuthnot*, i.e. overwrite the old *arbuthnot* dataset with the new one containing the new variable.”

This is equivalent to going through each row and adding up the boys and girls counts for that year and recording that value in a new column called *total*.

You'll see that there is now a new column called *total* that has been tacked onto the data frame. The special left-arrow symbol “`<-`” performs an assignment, taking the output of one line of code and saving it into an object in your environment. In this case, you already have an object called *arbuthnot*, so this command updates the original data set to one having the additional, new mutated column.

You can make a line plot of the total number of baptisms per year with the command:

```
ggplot(data = arbuthnot, aes(x = year, y = total)) +  
  geom_line()
```



Similarly, as we now have stored the *total* number of births each year, we can compute the ratio of the number of *boys* to the number of *girls* baptized in 1629, given by

5218 / 4683

or you can act on the complete columns with the expression

```
arbuthnot <- arbuthnot %>%  
  mutate(boy_to_girl_ratio = boys / girls)
```

You can also compute the proportion of all newborns that are *boys* in 1629

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

```
## [1] 0.5270175
```

or you can compute this quantity for all years simultaneously and append it as a column in the original tibble, using the `mutate()` function:

```
arbuthnot <- arbuthnot %>%  
  mutate(boy_ratio = boys / total)
```

Note that we are using the new *total* variable, contained in the *arbuthnot* tibble, that we created earlier.

Exercise 3

3. Now, generate a plot of the proportion of boys born over time. What do you see?

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 the number of births of boys outnumber that of girls in each year with the expression

```
arbuthnot <- arbuthnot %>%  
  mutate(more_boys = boys > girls)
```

This command adds a new variable to the *arbuthnot* dataframe containing the 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 variable contains a different kind of data than we have encountered so far. All other columns in the *arbuthnot* data frame have values that 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.

6. More Practice

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

Load the present data frame

To find the minimum and maximum values of columns, you can use the functions `min()` and `max()` together with the `summarize()` function, operating on the *arbuthnot* tibble. For example, the next code chunk shows how to find the minimum and maximum amount of boy births in a year:

```
arbuthnot %>%  
  summarize(min = min(boys), max = max(boys))
```

Exercise 4

4. What years are included in the *present* data set? What are the dimensions of the data frame? What are the variable (column) names?

Exercise 5

5. How do these counts compare to Arbuthnot's? Are they of a similar magnitude?

Exercise 6

6. Make a plot that displays the proportion of boys born over time. 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.

Exercise 7

7. In what year did we see the most total number of births in the U.S.?

Hint: First calculate the totals and save it as a new variable. Then, sort your dataset in descending order based on the total column. You can do this interactively in the data viewer by clicking on the arrows next to the variable names. To include the sorted result in your report you will need to use two new functions: `arrange()`, for sorting. We can arrange the data in a descending order with another function: `desc()`, for descending order. The sample code is provided below.

```
present %>%  
  arrange(desc(total))
```

The data contained in the *present* dataset come from reports by the Centers for Disease Control. You can learn more about them by bringing up the help file using the command:

```
?present
```

Resources for learning R and working in RStudio

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 unit progresses.

Remember there are many resources available - refer to the Detailed Unit Information document in the **Week 0** section on the unit Moodle site.

7. Complete your Lab 1 Submission

You need to complete a this Lab by completing the Moodle activity (a “quiz”) named “Lab 1 Submissions”. This can be found in the **Week 1** section on Moodle.

To receive the full 3% credit for the Week 1 Lab, before 12noon on Monday 10 August 2020, students must:

- i. Complete Exercises 1-7 in your **Lab01_IDnnnnnnnn.Rmd** file. Remember to include both code chunks and relevant text, for each question. (You can also delete the instructional text, e.g. “Insert any text here.” or `# Insert code for Exercise 2 here.`)
- ii. Render your **Lab01_IDnnnnnnnn.Rmd** file to an .html format. (This is your only option using this template.)
- iii. You need to convert the **Lab01_IDnnnnnnnn.html** to a .pdf document, named **Lab01_IDnnnnnnnn.pdf**. This can be done as follows: Open the .html file in the web-browser **Google Chrome**, and then print the page to a .pdf file.
- iv. Complete the Week 1 Lab Submission “quiz” in Moodle (with correct answers), following the instructions provided on Moodle.
- v. Submit the “quiz” before 12noon on Monday 10 August 2020.

GOOD LUCK and HAVE FUN!

(Remember to post questions or clarifications on the Discussion Forum!)