Lab 1: Introduction to Data

Overview

In this lab, we begin working with a data set. We identify fundamental attributes of the data, such as its size and variable names, and we also calculate new variables based on ones in the data set. Lastly, we produce basic plots of the data.

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 every year from 1629 to 1710.

Getting Started with Data

You will need to download the .csv file "arbuthnot.csv" from Canvas to begin this lab. You should set up a QTM 100 folder for this class, as well as a QTM 100 datasets sub-folder within that folder. Store all of your datasets into this folder, including arbuthnot.csv.

To import the dataset, set your working directory to the folder that contains it, either using the RStudio interface (More -> Set as Working Directory) or setwd function adn then use the read.csv function as follows.

```
setwd("~/Desktop/QTM_100_S21/Lab_Datasets/")# change this to your folder

arbuthnot <- read.csv("arbuthnot.csv", header = TRUE)
```

Some points to remember:

1. The variable's name (data frame) that contains your data need not necessarily be `arbuthnot`. You can choose any convenient name.
2. You need to use forward slashes "/" when providing folder or file paths. You must specify the paths in quotations as any other `string` variable.
3. To obtain the file path on Windows, locate the file, hold the `Shift` key, right-click the file, select `Copy as path`, paste (`Ctrl + V`) it into your script, replace backward slashes with forward slashes.
4. To get the file path on Mac, use 'Get Info. More details are available here.

We can look at the data by typing its name into the console.

```
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 scroll bar 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. The comparison to a spreadsheet will generally be helpful. R has stored Arbuthnot's data in a spreadsheet or table called a data frame.

An alternative way to view the data is to use RStudio's built-in data viewer. Click on the name Arbuthnot in the upper right window that lists the objects in your workspace. It will bring up an alternative display of the Arbuthnot counts in the upper left window. You can close the data viewer by clicking on the "x" in the upper lefthand corner.

Instead of viewing the entire data set, you can view the first six entries with the command *head* or the last six entries with the command *tail*:

```
head(arbuthnot)
tail(arbuthnot)
```

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

```
dim(arbuthnot)
```

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)
```

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.

To get a quick overview of the values in your data, the `summary` command will produce a summary of each variable in the data set according to its type (we'll discuss different types of variables more).

```
summary(arbuthnot)
```

The summary allows us to see various data artifacts quickly; for example, `year` ranges from 1629 to 1710, the fewest number of boy births in a given year was 2890, and the largest number of girl births in a given year was 7779.

Working with Data

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
```

The dollar sign accesses the variable `boys` in the data set `arbuthnot.` This command will only show the number of boys baptized each year.

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 display line. These data are no longer structured in a table with other variables, so they are displayed 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, that would mean the first number on that line would represent the 43rd entry in the vector.

Now, suppose we want to examine the total number of baptisms each year. We can use R as a calculator to compute the total number of baptisms in 1629. Type the mathematical expression:

```
5218 + 4683
```

And you see that there were 9,901 baptisms in 1629. We could repeat this once 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
```

The 82 numbers here represent the sum of girl and boy baptisms for each year. Take a look at a few of them and verify that they are right.

Similarly to how we computed the total number of baptisms, we can calculate the proportion of baptisms that are boys as the number of boy baptisms divided by the total number of baptisms for any given year:

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

Instead of calculating the proportion for just the year 1629, let's simultaneously compute this for all years.

Using the assignment command `<-`, we can save this result to our arbuthnot data set under the name propBoys.

```
arbuthnot$propBoys <- arbuthnot$boys / (arbuthnot$boys + arbuthnot$girls)
```

It creates a new variable called propBoys in the arbuthnot data set. View the data set to verify that it has an additional variable stored, and then view the values of this variable with:

```
arbuthnot
arbuthnot$propBoys
```

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 number of newborns, so we must use parentheses. Without them, R will first divide, then add, giving you something, not a proportion.

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

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

This command would return 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. Our values are numerical in the `arbuthnot` data frame (the year, the number of boys and girls). Here, we've asked R to create logical data, where the values are either `TRUE` or `FALSE`. In general, data analysis will involve many different kinds of data types that need to be considered and treated differently.

With a logical data result, `TRUE` has an internal value of 1, and `FALSE` has 0. Summing a logical will evaluate the total number of TRUE results.

```
sum(arbuthnot$boys > arbuthnot$girls)
```

Here, we see that all 82 results are TRUE, meaning that for all years, the number of boy baptisms exceeded the number of girl baptisms.

Plotting Data

What is the proportion of male baptisms, and does it vary by year? We can assess this by summarizing the data numerically or looking at it graphically. A numeric summary of the propBoys variable reveals that the proportion of boy baptisms ranged from 0.5027 to 0.5362 with an average of 0.5170.

R has some powerful functions for making graphics. A plot can provide even further insight. We can create a simple plot of the proportion of boys baptized per year with the command:

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

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 the line.

```
plot(x = arbuthnot$year, y = arbuthnot$propBoys, 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 available arguments, type in a question mark followed by the name of the function 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. The help file shows you that the plot command can take an additional argument called type, which can have a variety of specifications.

The help file for the plot only provides the most basic items that you can manipulate in the plot. View the graphical parameter settings for a list of all items that you can control.

```
?par
```

For example, here, we can see that we can also change plotting color with the argument col. Colors can take on numeric values 1-10, and they can also take on character values. For example, try plotting in red with the numeric assignment or in the color plum with the character assignment. Note that the character assignment must go in quotations.

```
plot(x = arbuthnot$year, y = arbuthnot$propBoys, type = "l",col=2)
plot(x = arbuthnot$year, y = arbuthnot$propBoys, type = "l",col="plum")
```

For a list of all colors available, submit:

```
colors()
```