Week 2: Introduction to R, part 2

Goals

• Learn how to increase the power of R with packages
• Use vectors
• Use data frames

Learning the Tools

Vectors

One useful feature of R is the ability to sometimes apply functions to an entire collection of numbers. The technical term for a set of numbers is “vector”. For example, the following code will create a vector of five numbers:

\[
\text{c()} \quad > \quad \text{c}(78, 85, 64, 54, 102, 98.6)
\]

This creates a vector containing the numbers 78, 85, 64, 54, 102, and 98.6.

\[
\text{c()} \quad \begin{bmatrix}
78.0 \\ 85.0 \\ 64.0 \\ 54.0 \\ 102.0 \\ 98.6
\end{bmatrix}
\]

c() is a function that creates a vector, containing the list of items given in its arguments. To help you remember, you could think of the function c() meaning to “combine” some elements into a vector.

Let’s add a little extra here to make the computer remember this vector. Let’s assign it to a variable, called \text{temperatureF} (because these numbers are actually a set of temperatures in degrees Fahrenheit):

\[
\text{temperatureF} \leftarrow \text{c}(78, 85, 64, 54, 102, 98.6)
\]

The combination of the less than sign and the hyphen makes an arrow pointing from right to left—this tells R to assign the stuff on the right to the name on the left. In this case we are assigning a vector to the variable \text{temperatureF}.

Inputting this to R causes no obvious output, but R will now remember this vector of temperatures under the name \text{temperatureF}. We can view the contents of the vector \text{temperatureF} by simply typing its name:

\[
\text{temperatureF}
\]
The power of vectors is that sometimes \( R \) can do the same calculation on all elements of a vector with one command. For example, to convert a temperature in Fahrenheit to Celsius, we would want to subtract 32 and multiply times 5/9. We can do that for all the numbers in this vector at once:

\[
> \text{temperatureC} <- (\text{temperatureF} - 32) \times \frac{5}{9}
\]

\[
> \text{temperatureC}
\]

\[
[1] \ 25.55556 \ 29.44444 \ 17.77778 \ 12.22222 \ 38.88889 \ 37.00000
\]

To pull out one of the numbers in this vector, we add square brackets after the vector name, and inside those brackets put the index of the element we want. (The “index” is just a number giving the relative location in the vector of the item we want. The first item has index 1, etc.) For example, the second element of the vector \( \text{temperatureC} \) is

\[
> \text{temperatureC}[2]
\]

\[
[1] \ 29.4444
\]

One of the ways to slip up in \( R \) is to confuse the [square brackets] which pull out an element of a vector, with the (parentheses) , which is used to enclose the arguments of a function.

Vectors can also operate mathematically with other vectors. For example, imagine you have a vector of the body weights of patients before entering hospital (\( \text{weight\_before\_hospital} \)) and another vector with the same patient's weights after leaving hospital (\( \text{weight\_after\_hospital} \)). You can calculate the change in weight for all these patients in one command, using vector subtraction:

\[
> \text{weight\_change\_during\_hospital} <- \text{weight\_before\_hospital} - \text{weight\_after\_hospital}
\]

The result will be a vector that has each patient’s change in weight.

### Basic calculation examples

**mean()**

In this course, we’ll learn how to use a few dozen functions, but let’s start with a couple of basic ones.

The function \( \text{mean()} \) does just what it sounds like: it calculates the sample mean (that is, the average) of the vector given to it as input. For example, the mean of the vector of the temperatures in degrees Fahrenheit:
Celsius from above is 26.81481:

```
> mean(temperatureC)
[1] 26.81481
```

**sum()** Another simple (and simply named) function calculates the sum of all numbers in a vector: `sum()`.

```
> sum(temperatureC)
[1] 160.8889
```

**length()** To count the number of elements in a vector, use `length()`.

```
> length(temperatureC)
[1] 6
```

This shows that there are 6 temperature values in the vector that make up the vector `temperatureC`. 
Extending R’s capabilities

R has a lot of power in its basic form, but one of the most important parts about R is that it is expandable by the work of other people. These expansions are usually released in “packages”.

Each package needs to be installed on your computer only once, but to be used it has to be loaded into R during each session.

To install a package in RStudio, click on the packages tab from the sub-window with tables for Files, Plots, Packages, Help, and Viewer. Immediately below that will be a button labeled “Install”—click that and a window will open.

In the second row (labeled “Packages”), type “ggplot2” (without the quotation marks). Make sure the box for “Install dependencies” near the bottom is clicked, and then click the “Install” button at bottom right. This will install the graphics package “ggplot2”.

This only needs to be done once on a given computer, and that package is permanently available.

There is another package that we will use in this course (and briefly today) called dplyr. While you are installing packages, go ahead and install dplyr as well.

Loading a package

Once a package is installed, it needs to be loaded into R during a session if you want to use it. You do this with a function called library().

library() For this week’s labs, we will use the package dplyr, which allows for
easy modification of data frames. Before using the functions in this package, we need to load it. We do this with the `library()` function. In the console, enter this:

```r
> library(dplyr)
```

If the `ggplot2` package is installed on your computer, the computer will just give a new prompt and be ready to go. If the package is not installed it will give you an error message in red asking you to get the package installed. (See the section above.)

*R* has a lot of functionality for graphics in its basic form, but `ggplot2` has a more uniform syntax and is very widely used. Also, `ggplot2` gives nicer looking graphs than base *R* without a lot of extra commands.
Setting the working directory

The files on your computers are organized hierarchically into folders, or “directories.” It is convenient in RStudio to tell R which directory to look for files at the beginning of a session, to minimize typing later.

For these labs, the best way to set your working directory is to start R and RStudio by clicking on the “ABDLabs.Rproj” file in the ABDLabs directory. This will automatically load the needed packages and set the working directory to this folder.

You can also set the working directory for RStudio from RStudio’s menu. From the “Session” tab in the menu bar, choose “Set Working Directory”, and then “Choose Directory…” This will open a dialog box that will let you find and select the directory you want. For these labs, we will assume that you are using as your working directory the “ABDLabs” folder that is distributed with this lab manual.

Reading a file

In this course, we have saved the data in a “comma-separated variable” format. All files in this format ought to have “.csv” as the end of their file name. A CSV file is a plain text file, easily read by a wide variety of programs. Each row in the file (besides the first row) is the data for a given individual, and for each individual each variable is listed in the same order, separated by commas. It’s important to note that you can’t have commas anywhere else in the file, besides the separators.

The first row of a CSV file should be a “header” row, which gives the names of each variable, again separated by commas.

**read.csv()**

For examples in this tutorial, let’s use a data set about the passengers of the RMS Titanic. One of the data sets in the folder of data attached to this lab is called “titanic.csv”. This is a data set of 1313 passengers from the voyage of this ship, which contains information about some personal info about each passenger as well as whether they survived the accident or not.

To import a CSV file into R, use the `read.csv()` function as in the following command. (This assumes that you have set the working directory to ABDLabs, as we described above.)

```r
> titanicData <- read.csv("Data/titanic.csv")
```
This looks for the file called `titanic.csv` in the folder called `Data`. Here we have given the name `titanicData` to the object in R that contains all this passenger data. Of course, if you wanted to load a different data set, you would be better off giving it a more apt name than “titanicData”.

To see if the data loads appropriately, you might want to run the command

```
> summary(titanicData)
```

which will list all the variables and some summary statistics for each variable.

---

### Intro to data frames

A data frame is a way that R can store a data set on a number of individuals. A data frame is a collection of columns; each column contains the values of a single variable for all individuals. The values of each individual occur in the same order in all the columns, so the first value for one variable represents the same individual as the first value in the lists of all other variables.

The function `read.csv()` loads the data it reads into a data frame. The data frame is usually given a name, which is used to tell R’s functions which data set to use. For example, in the previous section we read in a data set to a data frame that we called `titanicData`. This data frame now contains information about each of the passengers on the Titanic. This data frame has seven variables, so it has seven columns (passenger_class, name, age, embarked, home_destination, sex, and survive).

Very importantly, we can grab one of the columns from a data frame by itself. We write the name of the data frame, followed by a $, and then the name of the variable.

For example, to show a list of the age of all the passengers on the Titanic, use

```
titanicData$age
```

This will show a vector that has all the values for this variable `age`, one for each individual in the data set.
Adding a new column

Sometimes we would like to add a new column to a data frame. The easiest way to do this is to simply assign a new vector to a new column name, using the $.

For example, to add the log of age as a column in the titanicData data frame, we can write

\[
titanicData$log\_age = \log(titanicData$age)
\]

You can run the command head(titanicData) to see that log_age is now a column in titanicData.

Choosing subsets of data

Sometimes we want to do an analysis only on some of the data that fit certain criteria. For example, we might want to analyze the data from the Titanic using only the information from females.

The easiest way to do this is to use the filter() function from the package dplyr. Make sure you have sourced the dplyr package as described above, and then load it into R using library():

\[
> \text{library(dplyr)}
\]

In the titanic data set there is a variable named sex, and an individual is female if that variable has value “female”. We can create a new data frame that includes only the data from females with the following command:

\[
> \text{filter(titanicData, sex == "female")}
\]

This new data frame will include all the same columns as the original titanicData, but it will only include the rows for which the sex was “female”.

Note that the syntax here requires a double == sign. In R (and many other computer languages), the double equal sign creates a statement that can be evaluated as true or false, while a single equal sign may change the value of the first variable to the value on the right-hand side of the equal sign. Here we are asking, for each individual, whether sex is “female”, not assigning the value “female” to the variable sex. So we use a double equal sign ==.
**R commands summary**

### Making a vector
```
c(78, 85, 64, 54, 102, 98.6)
```
c() combines all its arguments into a vector
Elements of the vector

### Mean of a vector
```
mean(temperatureC)
```
Vector of numerical data

### Sum of a vector
```
sum(temperatureC)
```

### Length of a vector
```
length(temperatureC)
```

### Loading packages
```
library(ggplot2)
```
Name of the package

### Reading a data file
```
titanicData <- read.csv("Data/titanic.csv")
```
Name for the data frame
Name of the folder (if not in working directory)
Name of the data file

### Summary
```
summary(titanicData)
```
Name of the data frame

### Choosing one variable from a data frame
```
titanicData$age
```
Name of the data frame
Name of the variable

### Subsetting the data
```
filter(titanicData, sex == "female")
```
Name of the source data frame
Name of the variable to filter by
Desired value of variable
Questions

For the answers the following questions, create a script that captures each of these calculations, and use comments to record your verbal answers. Also, use comments to indicate the questions numbers. Make sure to include your name and the date in a comment at the top.

1. People are notoriously dishonest about revealing how often they perform antisocial behaviors like peeing in swimming pools. (In addition to be disgusting, the nitrogenous chemicals in urine combine with the pool’s chlorine to produce some toxic chemicals like trichloramine, the source of most skin irritations for swimmers.) A group of researchers (Jmaiff Blackstock et al. 2017) recently realized that an artificial sweetener called ACE passes out in urine unmetabolized and in known average quantities, and therefore by measuring ACE concentrations we can measure the amount of urine in a pool.

Here is a list of measurements, each from a different pool, of the concentration of ACE (measured in ng/L) for 23 different pools in Canada.

640, 1070, 780, 70, 160, 130, 60, 50, 2110, 70, 350, 30, 210, 90, 470, 580, 250, 310, 460, 430, 140, 1070, 130

a. In R, create a vector of these data, and name it appropriately.
b. What is the mean ACE concentration of these 23 pools?
c. Urine on average has 4000 ng ACE/ml. Therefore to convert these measurements of ng ACE / L pool water to ml urine / L pool water we need to divide each by 4000.

Make a new vector showing the concentration of urine per liter in these 23 pools. Give it a suitable name.
d. What is the mean concentration of urine per liter? How did this change relative to the mean measurement of ng ACE / L?
e. The arithmetic mean is calculated by adding up all the numbers and dividing by how many numbers there are. Calculate the mean of these numbers using `sum()` and `length()`. Did you get the same answer as with using `mean()`?
f. Use R to calculate the average amount of urine (in ml) in a 500,000 L pool.

2. Weddell seals live in Antarctic waters and take long strenuous dives in order to find fish to feed upon. Researchers (Williams et al. 2004) wanted to know whether these feeding dives were more energetically expensive than regular dives (perhaps because they are deeper, or the seal has to swim further or faster). They measured the metabolic costs of dives using the oxygen consumption of 10 animals (in ml O₂ / kg) during a feeding dive. Here are the data:
For the same 10 animals, they also measured the oxygen consumption in non-feeding dives. With the 10 animals in the same order as before, here are those data:

42.2, 51.7, 59.8, 66.5, 81.9, 82.0, 81.3, 81.3, 96.0, 104.1

a. Make a vector for each of these lists, and give them appropriate names.
b. Confirm (using R) that both of your vectors have the same number of individuals in them.
c. Create a vector called MetabolismDifference by calculating the difference in oxygen consumption between feeding dives and nonfeeding dives for each animal.
d. What is the average difference between feeding dives and nonfeeding dives in oxygen consumption?
e. Another appropriate way to represent the relationship between these two numbers would be to take the ratio of O\textsubscript{2} consumption for feeding dives over the O\textsubscript{2} consumption of nonfeeding dives. Make a vector which gives this ratio for each seal.
f. Sometimes ratios are easier to analyze when we look at the log of the ratio. Create a vector which gives the log of the ratios from the previous step. (Use the natural log.) What is the mean of this log-ratio?

3. The data file called “countries.csv” in the Data folder contains information about all the countries on Earth\(^1\). Each row is a country, and each column contains a variable.

a. Use read.csv() to read the data from this file into a data frame called countries.
b. Use summary() to get a quick description of this data set. What are the first three variables?

\(^1\) These data mainly come from the World Health Organization, but the Continent list comes from https://datahub.io/ and the ecological footprint and cell phone data come from http://www.nationmaster.com.
c. Using the output of `summary()`, how many countries are from Africa in this data set?

d. What kinds of variables (i.e., categorical or numerical) are `continents`, `cell_phone_subscriptions_per_100_people_2012`, `total_population_in_thousands_2015` and `fines_for_tobacco_advertising_2014`? (Don’t go by their variable names – look at the data in the summary results to decide.)

e. Add a new column to your countries data frame that has the difference in ecological footprint between 2012 and 2000. What is the mean of this difference?

4. Using the countries data again, create a new data frame called `AfricaData`, that only includes data for countries in Africa. What is the sum of the `total_population_in_thousands_2015` for this new data frame?