Session 3

Exploring Dataframes

In Session 2, you learned about different kinds of data, how those data might be organized in R, and you encountered some R functions that helped you to describe your data.

This session will build upon Session 2 by:

a) introducing a new method to get your data into R,
b) demonstrating how to view the data in a dataframe,
c) learning how to manipulate you data in dataframes.

1. How to input and organize your data

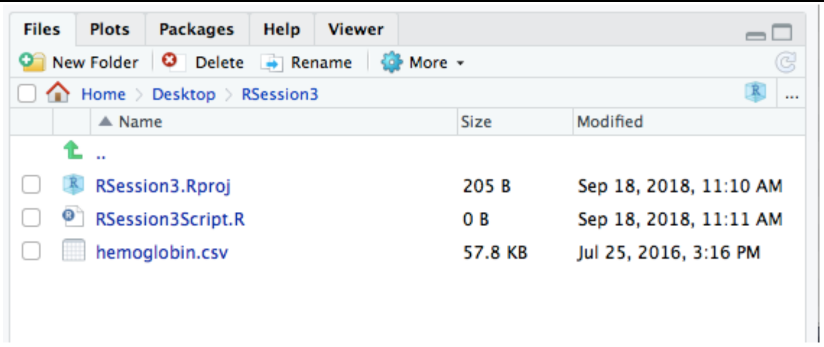
In Session 2, we created a dataframe by creating two vectors and using the command data.frame() to combine them into one object, the dataframe (remember the 'hemoglobin' example?). This method works fine when you have only a handful of data points. But imagine how difficult it would be to use this method if you had, say, 300 data points… This would be a nuisance.

Fortunately, there is a more convenient way to get data into R. Doing so requires two steps: (1) entering data into a spreadsheet, like Excel, and (2) getting that spreadsheet into R, which we can accomplish with the function, **read.table()**.

This is exactly what we will do to begin this session, but in reverse order: we will import a file, explain how this is done, and then examine how a spreadsheet is typically organized.

To develop Session 3, the course organizers created a spreadsheet in Excel, and saved it as 'hemoglobin.csv'. We'll explain how to import data from this file into RStudio, below. We recommend you watch this video as an alternative explanation of how to import data into RStudio: https://media.ed.ac.uk/media/1_3pnybd38

If you're using RStudio installed on your own computer , you can find the hemoglobin.csv dataset in the folder where you obtained this pdf file. Move it the folder into the RStudio Project Directory that you're currently using. For example, if you created a new project directory for this 3rd R-Session, move the file into this new directory (on some operating systems, you can do this by dragging the file between folders). When you have done so, you will see the name of the file appear in RStudio's menu. For example, I created a new project called RSession3 (and I saved my script file as RSession3Script); when I placed the hemoglobin.csv file in the project folder (i.e., the folder called RSession3) the csv file appeared in the menu as illustrated, below:



Name	Size	Modified
..		
RSession3.Rproj	205 B	Sep 18, 2018, 11:10 AM
RSession3Script.R	0 B	Sep 18, 2018, 11:11 AM
hemoglobin.csv	57.8 KB	Jul 25, 2016, 3:16 PM

Now that the file is visible, you can use the read.table() command to create a dataframe that holds the file's data:

```
hemo <- read.table("hemoglobin.csv", header=TRUE, sep=",")
```

If you're using **RStudio that is installed on a University of Edinburgh computer**, you need to move the file you wish to use into your project directory, just as we did above. As above, this will cause your file to appear in the RStudio menu. At this stage, you need to click on the file listed in the menu; two options will appear and you must choose "Import dataset". Now, a new window will appear with a variety of panels: find the panel labeled "Code Preview". It will include a command that uses a function like, "read_csv". Edit this function in three ways: 1) change the function from read_csv (or something similar) to read.table; 2) insert the options header=TRUE and sep=',' as seen in the command, above; 3) if you wish, change the name of the object that your data will be saved to. Now, press "Import", and your data will be saved in a dataframe with the name you chose.

Let's dissect the read.table() command to see how it works. Overall, we're using the function read.table() to assign data to the object, 'hemo. The term in quotes indicates the location and name of the file we wish to read.

So, what do the terms, **header=TRUE** and **sep=","** mean? These are options that you can specify to the function read.table() that allow it to do more precisely what you wish. The option **header=TRUE** tells the function that the first row of the data will contain the names of the columns, which are illustrated below. So, if you include columns names in the first row of your spreadsheet (you will see an example of this, below) and specify **header=TRUE**, read.table() will automatically use the your desired column names. The **sep=","** option tells the read.table() function that the data in the file hemoglobin.csv are separated by commas, which allows R to interpret the file correctly (see below about csv files).

Look at the dataframe by typing:

```
hemo
```

You should see a dataframe that looks like this:

```
    id                  hemoglobin   population
1   US.Sea.level1       10.40        USA
2   US.Sea.level2       11.20        USA
3   US.Sea.level3       11.70        USA
4   US.Sea.level4       11.80        USA
5   US.Sea.level5       11.90        USA
6   US.Sea.level6       12.05        USA
7   US.Sea.level7       12.10        USA
8   US.Sea.level8       12.15        USA
9   US.Sea.level9       12.20        USA
10  US.Sea.level10      12.35        USA
…
1953   Tibetan.male50     17.12     Tibet
1954   Tibetan.male51     17.12     Tibet
1955   Tibetan.male52     17.12     Tibet
1956   Tibetan.male53     17.12     Tibet
1957   Tibetan.male54     17.12     Tibet
1958   Tibetan.male55     17.69     Tibet
1959   Tibetan.male56     18.25     Tibet
1960   Tibetan.male57     19.39     Tibet
1961   Tibetan.male58     19.73     Tibet
1962   Tibetan.male59     19.96     Tibet
```

Let's examine this dataframe. From the numbers along the left, we see that there are 1962 observations. The column titles tell us that there are three columns, called 'id, 'hemoglobin', and 'population'. The 'id' column provides basic information about the person studied. Note that we added the ''…" to the middle of the dataframe printout to indicate that this document skips a lot of the data in the middle to save space; we've only presented the first and last 10 observations of the dataset, and the … indicates the data we've omitted.

2. How should data be organized in a spreadsheet?

Learning how to organize your data in a dataframe is a crucial skill for data analysis. The data in the dataframe, hemo, are organized in what's called the 'long' form, which is often an appropriate approach to organize data. In the long form format, rows contain information about a single unit that you have observed (e.g., one person), whereas each column contains one piece of information that you measured from that unit. For example, all of the measurements in the first row of 'hemo' were taken from a single person, whose id was 'US.Sea.level1' , who lived in the USA and had a hemoglobin level of 10.40. Likewise, the person that described in the second line of 'hemo' was given the id 'US.Sea.level2', they also lived in the USA and had a hemoglobin level of 11.20.

You can enter your data into Excel using the 'long format', and then save your data in an appropriate file type. We recommend the **csv** file type

2.a What is a csv file and how is a csv file created?

A csv (comma-separated values) file is an efficient type of file for storing data, in that it requires little computer memory.

2.b How do we create a csv file? If you were using Excel, you would enter your data into columns as you see above. Then, when you save your data, you would choose 'csv' as the 'file type'. It is that easy.

2.c So, what does a csv file look like? In short, it looks like the dataframe, but a comma separates the data from the different columns. For example, the first 3 lines of the csv file, hemoglobin.csv, look like this:

```
id,hemoglobin,population
US.Sea.level1,10.40,USA
US.Sea.level2,11.20,USA
```

Not very pretty, but efficient.

3. Looking at portions of the dataframe
Sometimes, you will have a dataframe that is big enough to make it inconvenient to examine the entire dataframe at once; this is true of 'hemo', above. Instead, you might only wish to look at the first 5 lines to see how the data are structured.

So, how can we look at only specific rows and columns of a dataframe? We saw in Session 2 how to use square brackets to examine or work with one or more numbers in a vector. When we work with dataframes there are two main things to know.

First, to tell R that you want to look at a portion of a dataframe, as with vectors, you need to use square brackets (i.e., [and]) in combination with the dataframe's name.

Second, you need to know that, within these square brackets, you specify **first the rows and then the columns** that you wish to focus upon.

For example, if you wished to know what measurement was stored in the 3rd row of the 2nd column of the dataframe, hemo, you would enter:

```
hemo[3,2]
```

Try this now, and compare it to the printout of the dataframe, above.

If, instead, you wished to look at the entire 3rd row of data, you'd enter:

```
hemo[3,]
```

This code tells R that you wish to see data from the 3rd row, but you're not fussy about which columns of data are presented. As a result, R will present all columns of data for row 3.

Likewise, if you wished to look at all rows of data in the 1st column, you would enter:

```
hemo[,1]
```

Following this train of thought, note that we can use this notation to call on functions to analyse portions of our data. For example, to find the mean value in the 2nd column, you can write:

```
mean(hemo[,2])
```

…which gives a result of 15.56137.

We've now seen how to examine one row or one column at a time. How can we look at multiple rows or columns? To do this, we make use of the colon, which tells R to look at a range of rows or columns. For example, to look at rows 1 to 4, we enter:

```
hemo[1:4,]
```

or likewise, to look at the columns 2 to 3, we enter:

```
hemo[,2:3]
```

So, what would we enter to look at only the data in columns 2 to 3 of rows 1 to 4? Try to figure it out now!

4. Using functions with dataframes

As noted above, we can apply functions to the data in dataframes. On that note, we'll now discuss how to perform mathematical operations on data. For example, let's imagine that the instruments used to take measurements in the hemoglobin study were poorly calibrated, and consistently provided measures that were too small by 2 units (this did not actually happen in the study). In this case, we could 'correct' the data by adding 2 to all the hemoglobin measures. We can do this with a single command:

```
hemo$hemoglobinCorrected <- hemo$hemoglobin + 2
```

How does this command work? Recall from Session 2 that we can select an individual column in a dataframe using the $ operator. On the left side of the arrow, we are creating a new column called, hemoglobinCorrected, which becomes a part of the 'hemo' dataframe. On the right side of the arrow, we are adding 2 to each value in the original data in the ' hemoglobin' column of the dataframe, 'hemo'.

We can now perform analyses on our new column, hemogloginCorrected, if we wish

5. Exercises

We recommend that you complete two of the following exercises to practice what you learned, above. Any additional work will provide extra practice.

5.1. The hemoglobin data in the dataset, above, refers to a concentration of hemoglobin (in units of gm / dL (or, grams per deci-litre)). Each molecule of hemoglobin has 4 sites to bind oxygen.

a) Given the above information, create a new column in the dataframe, hemo, which contains the concentration of oxygen binding sites.

b) What is the average concentration of oxygen binding sites for this dataset? [*Note*: when you list the contents of hemo, the first colum displayed is the row number, and not a column of data in the dataframe. So if you enter 'hemo' then the second column displayed is the first column of data in the hemo dataframe]

5.2. Sometimes it is useful to work with only a portion of a dataset. For example, let's imagine that we were only interested in hemoglobin levels collected in the Tibet. In this exercise, we'll walk through one approach to focus on these data, only; Session 4 will consider this task in more detail.

a) Look at the dataframe, hemo, and determine which rows contain data from Tibet.

b) Using the square brackets approach you learned above, have R display only the rows of data from Tibet.

c) Now, we can go one step further. Create a new object that will hold only the rows of data from Tibet (e.g., TibetData <- *SomethingThatCallsOnlyRowsFromTibet*)

d) Look at the new object with data from Tibet to check that it contains the data that you expect.

e) Determine the minimum, mean and maximum hemoglobin levels in the Tibet data.

5.3. In this question, you will make your own dataframe from scratch, using your own data. We're going to analyze your taste in movies.

a) Open an Excel spreadsheet, and save it with a meaningful name. At the top of the left-most column, type "Movie" (omit the quotation marks). Then, below this, enter the names of 5 movies that you enjoy or would like to see. When you enter their names, enter them in a way that does not use spaces. For example, you might have a column with the movies: StarWarsForceAwakens, Untouchables, GoodBadAndUgly, PrincessBride, GoneWithTheWind. You should now have one column with 6 rows of information: the column name, followed by 5 movies.

b) Find a website that provides movie ratings. For example, one option is:

https://www.rottentomatoes.com

At the top of a second column, enter "Rating" (again, omitting the quotation marks). Find ratings for each of your 5 movies, and record each movie's rating in that column. For example, according to rottentomatoes.com, Gone With The Wind has an Audience Score of 93% (i.e., 93% of people liked it), so you would enter 0.93.

c) When your spreadsheet with its 2 columns and 6 rows of information (5 of which have data) is complete, save it as a csv file. To do this, select File, then "Save As", and then choose csv as the "Format".

d) Use read.table() to import your csv file into R, and adding it to an object (name it whatever you want). Remember to use the appropriate options for the header and 'sep' in the read.table() function.

e) Look at the dataframe you created. Is it as you expected? (We hope so, but it is always wise to check!)

f) Use the square brackets (i.e., [and]) to look only at the first 3 rows (this is just for practice).

g) Now, look at only the second column (again, just for practice).

h) Now, use functions in R to determine the highest rating among your selected movies.

i) What was the lowest rating?

j) What was the average rating among your movies? For fun, you can compare this average rating with those from other members of your class.