Session 2

In this session, you will

a) be introduced to three types of data,
b) learn how data can be stored in R,
c) learn some useful functions for describing your data.

1. Types of Data:

Data comes in different types. This might seem like an odd idea at first, but the distinction between different data types is obvious with a moment of thought. The distinction is also important for R.

To illustrate different types of data, consider three examples.

1.a Numeric data

Numeric data are exactly as they sound: they are numbers, like 67, 2.3 or 1000, which might describe things (e.g, rates of neurons firing).

1.b Categorical data

Second, we have character, or **'categorical'** data. These are data that assign an observation to a category. Examples include colours ('Red', 'blue' and 'yellow'), or 'present' versus 'absent', or 'small', 'medium' or 'large'.

1.c Logical

Third, we have **'logical'** data, represented by the terms 'TRUE' and 'FALSE'. (Writing TRUE and FALSE in upper-case (i.e., big letters) is important to how R interprets these words). R deals with logical data in special ways that can be very useful, that you'll learn about later.

2.How do we organize data in R?

R can organize data in a variety of forms. We'll mention their names now:

- vectors
- dataframes
- arrays
- matrices
- lists

However, in this session we'll only consider the first two; these are the only types you need to know about for many of the analyses that might interest you.

Below, we'll describe what a vector is, and illustrate things you can do with vectors. We'll then do the same thing with dataframes.

Before we go any further, open RStudio and create a new project; save it, and save a script file for your work in this R Session. Refer to the first R Session if you need to remind yourself how to do this (or see this video: https://media.ed.ac.uk/media/1_507bgw71). For the work, below, we remind you that we **recommend submitting your commands from a script file**.

3. Vectors

3.a What is a vector?

You may recall coming across the term 'vector' in your studies of maths or physics, where the term is defined as a quantity having magnitude and direction. In R, however, the term 'vector' simply represents a line of data of the same type.

We'll explain what a vector is in R by looking at 3 examples, and then highlighting important features that the examples have in common.

To create a vector, we will use the function, **c()**, which ***c***ombines values into a vector.

First, let's look at a vector of numbers through an example.

In this session, and the two sessions to follow, we'll work with some data from a study by Beall et al. (2002). They measured hemoglobin in thousands of people in order to understand how hemoglobin levels change with the altitude at which people live. In Session 4 we'll look at all of their data, but in this Session and Session 3 we'll focus on a small subset of these data.

Let's examine a sub-set of these hemoglobin measurements, focusing on 12 people. We may do this by first assigning each of the data values, separated by commas, to an R object we call 'hemo', using the combine, c(), function:

```
hemo <- c(10.4,11.2,11.7,14.96,15.64,15.98,12.65,12.9,13.43,13.03,13.03,13.26)
```

The point here is that 'hemo' is an example of a vector.

Now, look at the content of the object, 'hemo', by typing the following into R:

```
hemo
```

You will see the 12 numbers listed; they are the content of 'hemo'. The combine function, c(), combined these values into a vector.

Let's look at a vector of categorical data. These data record the locations where hemoglobin was measured in the above 12 people:

```
places <-
c("USA","USA","USA","Andes","Andes","Andes","Ethiopia","Ethiopia","Ethiopia","Tib
et","Tibet","Tibet")
```

These location data are recorded in the same order as those in the vector, hemo. So, the first hemoglobin measurement in 'hemo' was made on someone in the USA, while the last measurement was made with someone in Tibet. We can see that hemoglobin levels were studied in 4 locations: USA (sea level), the Andes, Ethiopia and Tibet.

Again, look at the contents of 'places'. Note that with categorical data, the values must have quotation marks. Try reentering the values into the vector 'places', but without the quotation marks to see what happens.

Useful R tip: *Note the shape of the quotation marks in 'places': they are straight. In contrast, typical quotation marks produced in a word processor, like Word, often look like this: ", which have a different shape. As far as R is concerned, these different shapes of quotation marks represent very different things and you can confuse R if you use the wrong type. If you type quotation marks directly into RStudio, you should not have problems. However, if you copy and paste a command from a Word document, differences in the form of quotation marks may cause it to fail; if so, edit the quotation marks in Rstudio.*

And, our third example:

```
truefalse <- c(TRUE,FALSE,TRUE,FALSE,FALSE)
```

Again, look at the contents of 'truefalse'.

What do these examples have in common? Two things. First, they all hold data that can be printed in a straight line and remain easily interpreted. In other words, vectors are 1-dimensional; you will see below that dataframes are 2-dimensional. Second, although our various vectors held different kinds of data, ***each example vector*** held only **one type** of data, be it numeric, categorical, or logical.

So, in R **a vector is a 1-dimensional array that holds data of only one type**.

3.b How do we know what type of data are stored in each of these vectors?

We might think that we could simply look at a vector's contents to know the type of

data being stored. But we'll see an example below where this is not the case.

To determine the type of data held in a vector, we can use the function, **mode()**. For example, if we type:

mode(hemo)

we get the output:

"numeric"

Similarly, for 'places' and 'truefalse' we get the following output:

"character"

and

"logical"

Try this for yourself!

But, what type of data are contained within the following vector:

onetofour <- c("1","2","3","4")

Will they be numeric?

If we enter **mode(**onetofour**)**, you'll see that the vector onetofour has character (i.e., categorical) data, because the numbers were contained within quotation marks.

This simple example with onetofour teaches us two lessons. **First,** just because data look like a particular data type, this does not mean that R will interpret the data in that way; it is useful to check what type of data R thinks it has. **Second,** we can make data categorical by putting it within quotation marks.

Useful R tip: *because of the confusion that can arise from using numbers to represent categories, it is best to code categorical data using characters (i.e., avoid using numbers for categorical data).*

3.c Can we do anything useful with the data in a vector?

Absolutely. We'll now see that you can describe the data in a vector, and that you can perform calculations on vectors.

We'll work with the first vector we made, 'hemo. We can use a variety of functions to tell us about the data in the object, 'hemo'. Consider a few examples…

- **How many observations (i.e., data points) are stored in hemo?**

Use the function **length()**:

length(hemo)

…and the answer is 12.

- **What is the value of hemo in position 12?**

Positional references in a vector are contained within square brackets. So the value contained in position 12 is

hemo[12]

The answer is 13.26

If we wish to know the values in more than one position, we might type

hemo[8:12]

which will respond with the values in positions 8 to 12. If we want to know only the values in positions 8 and 12, however, it is a little more tricky; type:

hemo[c(8,12)]

- **What are the largest and smallest numbers stored in hemo?**

Use the functions **max()** and **min()** (for maximum and minimum):

max(hemo)

min(hemo)

…and the answers are 15.98 and 10.4, respectively.

- **What do the numbers in 'hemo' sum to?**

Use **sum()**:

sum(hemo)

…and the answer is 158.18.

- **What is the mean (i.e., average) value in 'hemo'?**

Use **mean()**:

mean(hemo)

…equaling 13.18167

- **Can we use more than one function at a time?**

Yes! For instance, we could have calculated the mean 'the old fashioned way', which involves summing all of the numbers in 'hemo' together and then dividing this sum by the number of observations. This is done simply by:

sum(hemo)/length(hemo)

…and gives the same answer as the mean() function: 13.18167

If you haven't already, try all of these functions, now!

- **Can we also do calculations on the data in 'hemo'?**

Yes!

For example, let's say that we wanted to multiply all of the numbers in 'hemo' by 2. We could do this simply by entering:

2 * hemo

…which yields:

20.80 22.40 23.40 29.92 31.28 31.96 25.30 25.80 26.86 26.06 26.06 26.52

We can also perform calculations using more than one vector. For example, let's create a new vector, example:

example<- c(2,0,1,3,5,2.2,15.7,15,20.14,1,100,0)

Note that 'hemo' and 'example' both contain 12 observations.

We can now perform calculations that use both observations. For example,

addvectors <- hemo+ example

…and addvectors now yields:

12.40 11.20 12.70 17.96 20.64 18.18 28.35 27.90 33.57 14.03 113.03 13.26

Look back at the values in 'hemo' to see that the command,
addvectors <- hemo+ example
caused the first entry of the two vectors to be added together, the second entries of the two vectors to be added, and so on.

We can do a whole variety of calculations on vectors. Try some of your own, to get comfortable with this idea.

4. Dataframe

4.a What is a dataframe?

As we did with vectors, we'll first look at an example of a dataframe, and then discern the qualities of a dataframe.

Let's use the hemoglobin data we discussed above, as an example. Recall that, above, we had 12 observations (i.e., hemoglobin measurements) in total, with 3 from each of 4 countries. We could organize these data like this:

places	hemo
USA	10.40
USA	11.20
USA	11.70
Andes	14.96
Andes	15.64
Andes	15.98
Ethiopia	12.65

Ethiopia 12.90
Ethiopia 13.43
Tibet 13.03
Tibet 13.03
Tibet 13.26

We have 2 columns of data, one labeled 'places, which indicates where a hemoglobin measurement was made, and a second column labeled 'hemo, which is the hemoglobin level.

This is an example of a dataframe. Note that, unlike vectors, a dataframe has 2 dimensions (rows and columns). Also note that the columns can contain different types of data (in this case, categorical and numerical).

Thus, **a dataframe is a 2-dimensional array that can contain different types of data**. Specifically, a dataframe is made up of more than one vector, where each vector contains its own data of a given type.

So, if a dataframe is made up of vectors, can we build a dataframe out of individual vectors? Yes, we can! We'll do this now, as this is one useful way to get data into R; we'll see another way of getting data into R in Session 3.

Recall that above, we had the two vectors:

```
places <-
c("USA","USA","USA","Andes","Andes","Andes","Ethiopia","Ethiopia","Ethiopia","Tib
et","Tibet","Tibet")

hemo <- c(10.4,11.2,11.7,14.96,15.64,15.98,12.65,12.9,13.43,13.03,13.03,13.26)
```

Thus, in each vector, the first entry has information about the first person studied (either place or hemoglobin level), the second entry has information about the second person observed, and so on.

We can combine these vectors to make a dataframe, which we'll imaginatively call 'datahemo', like this:

```
datahemo <- data.frame(places,hemo)
```

We can now look at the content of the object, datahemo, by entering, 'datahemo', which looks like this:

```
   places  hemo
1     USA 10.40
2     USA 11.20
3     USA 11.70
```

4 Andes 14.96

5 Andes 15.64

6 Andes 15.98
7 Ethiopia 12.65
8 Ethiopia 12.90
9 Ethiopia 13.43
10 Tibet 13.03
11 Tibet 13.03
12 Tibet 13.26

This looks just like what we had above, except that R also provides numbers 1 to 12 along the left. These numbers denote the number for a particular observation, which can be useful (later, we'll see how).

4.b How do we look at only one column of data at a time?
Sometimes, you might like to look at only one column of data. You can specify a column of data in a dataframe with the character, $.

Try entering the commands:

datahemo$places

datahemo$hemo

..and you will see the data in these columns

We can now perform the same calculations on the data in the 'hemo' column as we did with vectors. For example, we can calculate the mean hemoglobin level with:

mean(datahemo$hemo)

(You should get an answer of 13.18167)

Likewise, what are the maximum and minimum hemoglobin levels in the dataframe, 'datahemo'? to get the answers, type

max(datahemo$hemo)

min(datahemo$hemo)

Reference:

Beall, C. M., et al. 2002. An Ethiopian pattern of human adaptation to high-altitude hypoxia. *Proceedings of the National Academy of Sciences (USA)* 99: 17215–17218

5. Exercises

We recommend that you complete at least the first exercise below, for practice.

5.1. Imagine that you went to a restaurant with 3 friends. In fact, to help your imagination, Google the restaurant that you're imagining and look at their menu. In fact, imagine further that BMS2 will pay for your meal at any restaurant in the world… so go crazy! From their menu, choose 1 starter, 1 main, and 1 dessert for each of yourself and your three friends. Now, we're going to use R as a sophisticated calculator to analyze the bill!

a) Create a vector that stores the price of each of the four starters. For example, in my restaurant, my starter, and those of my 3 friends cost the following:

   starters <- c(5.99,4.50,7.98,3.50).

   Now, using prices from your restaurant choice, make similar vectors, which you can call 'starters', 'mains', and 'desserts'. Imagine that the first entry in each vector is the price of something ordered by the first person, the second entry is from something ordered by the second person, and so on (this detail comes in handy in question 1(f), and question 2(a)).

b) Use the vectors to determine the total cost of the starters? The mains? And the desserts?

c) What is the total cost of the meal?

d) How much will the total bill cost if you wish to add a 10% tip?

e) If everyone agreed to share the cost of the bill evenly, how much should each person pay?

f) What was the total cost of each person's individual meal?

5.2. Use the vectors created in question 1(a) to create a dataframe that holds all of the costs of the meal.

a) Look at the dataframe. What does each row in the dataframe represent? (Recall the order in which you filled the vectors, in question 1(a).)

b) Using the dataframe and functions you learned above, determine the cost of the most expensive main course that was ordered.

c) How much did the least expensive starter cost?

d) What was the average price of a dessert?