Intro R Lab 1: Getting to know R

Simon Caton

Getting Help

```
help.start()
```

Start the HTML interface to online help (using a web browser available at your machine). This way of viewing help if often more useful and consumable that the default mechanism. You should briely explore the features of this facility then move on.

Similarly, if we want help on a specific function or to see an example of it in use, we use the **?** symbol. For example, let's look at the **mean** function.

```
?mean
```

Here, we see a brief overview of what the function does, the arguments it takes, and if we scroll to the bottom, we'll see an example and sometimes also some further reading materials.

Using R to generate and visualise data

Let's start nice and simple, with generating two pseudo-random normal vectors that will represent the x- and y-coordinates then plot them against each other.

```
x <- rnorm(50)
y <- rnorm(x)
```

Note here we are instantiating two variables, x and y using the assignment operator ('<-'), which consists of the two characters $<$ (less than) and - (minus) occurring strictly side-by-side and it *points* to the object receiving the value of the expression. In most contexts the $=$ operator can be used as an alternative.

Task Use help, to see what the *rnorm* function does.

Finally, we can make a simple plot of x against y

```
plot(x,y)
```

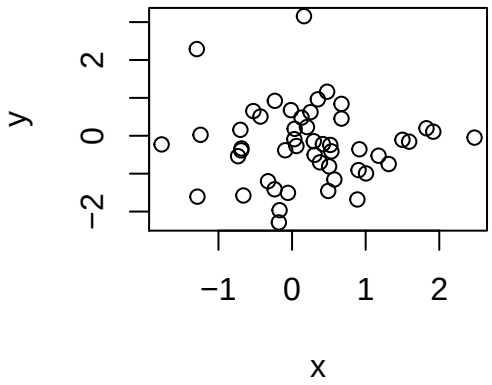

Figure 1: Plot of x against y

There are other simple ways to visualise these variables, try the following and see if you can work out what these functions do, not sure? Use help!

```
boxplot(x)
hist(x)
```

To remove our variables from the workspace we have a few different options:

```
x <- NULL
rm(y)
```

In the first case, the variable x is retained, but its value nullified, i.e. a variable x remains in memory, but has no value. In the second case y is deinstantiated.

Assigning numerical values to variables and manipulating them

Set up a vector named x, say, consisting of five numbers, namely 10.4, 5.6, 3.1, 6.4 and 21.7

```
x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
# this is a comment
# and by the way, c(10.4, 5.6, 3.1, 6.4, 21.7) -> x would give the same result
```

If an expression is used as a complete command, the value is printed and lost. So now if we were to use the command

```
1/x
```

the reciprocals of the five values in the vector x (0.0961538, 0.1785714, 0.3225806, 0.15625, 0.0460829) are printed to the terminal (and the value of x, of course, remains unchanged). To retain the result and store it in a variable, we need to assign the result to a new variable:

```
y <- 1/x
```

The elementary arithmetic operators are the usual +, -, *, / and ^ for raising to a power. In addition all of the common arithmetic functions are available. log, exp, sin, cos, tan, sqrt, and so on, all have their usual meaning. max and min select the largest and smallest elements of a vector respectively. range is a function whose value is a vector of length two, namely c(min(x), max(x)). length(x) is the number of elements in x, sum(x) gives the total of the elements in x, and prod(x) their product.

Task Have a go using some of these functions

Non-numeric data

As well as numerical vectors, R allows manipulation of logical quantities. The elements of a logical vector can have the values TRUE, FALSE, and NA (for "not available"). The first two are often abbreviated as T and F, respectively. Note however that T and F are just variables which are set to TRUE and FALSE by default, but are not reserved words and hence can be overwritten by the user. Hence, you should always use TRUE and FALSE, and **NEVER** instantiate a variable T or F.

Logical vectors are generated by conditions. For example

```
temp <- x > 13
```

sets temp as a vector of the same length as x with values FALSE corresponding to elements of x where the condition is not met and TRUE where it is. Or

```
## [1] FALSE FALSE FALSE FALSE  TRUE
```

The logical operators are <, <=, >, >=, == for exact equality and != for inequality. In addition if c1 and c2 are logical expressions, then c1 & c2 is their intersection (*and*), c1 | c2 is their union (*or*), and !c1 is the negation of c1.

Tasks

1. Identify the values of x between 3 and 7
2. Identify which values of x are greater than the mean of x
3. Identify which values of x are more than 1 standard deviation from the mean of x. Hint: ?sd

---

Logical vectors may be used in ordinary arithmetic, in which case they are *coerced* into numeric vectors, FALSE becoming 0 and TRUE becoming 1. However there are situations where logical vectors and their coerced numeric counterparts are not equivalent!

To represent Strings, we use a sequence of characters delimited by the quote characters, e.g., "x-values", 'New iteration results'. They also use standard escape sequences, using as the escape character, so \\ is entered and printed as \, and inside double quotes " is entered as \". Other useful escape sequences are \n, newline, \t, tab and \b, backspace—see ?Quotes for a full list.

Character vectors may be concatenated into a vector by the c() function; e.g.

```
c("Hello", "World")
```

```
## [1] "Hello" "World"
```

Similarly, the paste() function takes an arbitrary number of arguments and concatenates them one by one into character strings. E.g.

```
paste(c("X","Y"), 1:10, sep="")
```

```
##  [1] "X1"  "Y2"  "X3"  "Y4"  "X5"  "Y6"  "X7"  "Y8"  "X9"  "Y10"
```

```
#where sep defines the separator between each string
```

Missing Values

In some cases the components of a vector may not be completely known. When an element or value is "not available" or a "missing value", a place within a vector may be reserved for it by assigning it the special value *NA*.

It's **REALLY** important to recognise that there is a **HUGE** difference between a value that is missing, i.e. not known, and 0. For example, consider that we ask 5 people how much they have in their wallet/purse right now. There is a pretty fundamental difference, between recording 0 and recording nothing (or NA). For example, the individual may have refused to reveal the amount of money in their wallet, they may have lost it, or not had it with them when asked. In such cases, NA provides us with more information and is less ambiguous than 0 in that it allows us to differentiate between 0 and unknown.

The function is.na(x) gives a logical vector of the same size as x with value TRUE if and only if the corresponding element in x is NA. This is a very useful means to get an idea of how clean/complete a data set is.

```
z <- c(1:3,NA)
is.na(z)
```

```
## [1] FALSE FALSE FALSE  TRUE
```

Notice that the logical expression x == NA is quite different from is.na(x) since NA is not really a value but a marker for a quantity that is not available. Thus x == NA is a vector of the same length as x all of whose values are NA as the logical expression itself is incomplete and hence undecidable. In general, any operation on an NA becomes an NA.

Task Try it on the vector z produced above

Where NA is one kind of missing value, there is a second kind of missing values which are produced by numerical computations: Not a Number, NaN, values. For example

```
0/0
```

```
## [1] NaN
```

and

```
Inf - Inf
```

```
## [1] NaN
```

which both give NaN since the result cannot be defined sensibly. In summary, is.na(xx) is TRUE both for NA and NaN values. To differentiate these, is.nan(xx) is only TRUE for NaNs.

For strings, missing values are sometimes printed as when character vectors are printed without quotes.

Indexes, selecting and modifying subsets of a data set

Let's start with a simple vector

```
x <- c(-5:-1, NA, NA, 1:3)
```

So we have the values -5, -4, -3, -2, -1, NA, NA, 1, 2, 3. Let's look at a few ways to select, subset and modify our data.

Subsets of the elements of a vector may be selected by appending to the name of the vector an index vector in square brackets. More generally any expression that evaluates to a vector may have subsets of its elements similarly selected by appending an index vector in square brackets immediately after the expression. There are four distinct types of index vectors.

1. Logical Vectors: Values corresponding to TRUE in the index vector are selected and those corresponding to FALSE are omitted. For example

```
y <- x[!is.na(x)]
```

creates (or re-creates) an object y which will contain the non-missing values of x, in the same order. Note that if x has missing values, which is does in this case, y will be shorter than x. Note: we can place any logical expression in the square brackets. A more complex example is:

```
y <- x[(!is.na(x)) & x>0]
```

2. A vector of positive integral quantities: put simply, the values in the index vector within a given set, e.g. the first 5 values of x:

```
y <- x[1:5]
```

3. A vector of negative integral quantities: the opposite of the above, e.g. everything except the first 5 values of x:

```
y <- x[-(1:5)]
```

At this point, a valid question would be, ok so what? and why should I care? Well, put simply if we change 1:5 to the following, we have a really convenient ways to form subsets of our data. Consider our how much money do you have in your wallet/purse right now survey.

```
money <- c(10,100000,-10,NA,15)
strangeAnswers <- money[(is.na(x)) & money<0]
normalPeople <- money[(!is.na(x)) & money>=0 & money < 200]
toffs <- money[money > 10000]
```

4. A vector of character strings. This only applies where an object has a names attribute to identify its components. In this case a sub-vector of the names vector may be used in the same way as the positive integral labels in item 2 further above.

```
fruit <- c(5, 10, 1, 20)
names(fruit) <- c("orange", "banana", "apple", "peach")
lunch <- fruit[c("apple","orange")]
```

Task Check the contents of lunch.

The advantage is that alphanumeric names are often easier to remember than numeric indices. This option is particularly useful in connection with data frames, as we shall see later.

An indexed expression can also appear on the receiving end of an assignment, in which case the assignment operation is performed only on those elements of the vector. E.g.

```
x[is.na(x)] <- 0
```

which replaces NAs with a 0.

Task Instead of replacing NAs with 0 replace them instead with the mean of x. Note you will need to reassign the original values to x, as right now the NAs are 0s.

Whilst this may seem **extremely trivial** replacing and/or identifying missing values in this way is very useful. Now whilst there are many ways to decide what to replace a missing value with, the general process that we have just done is something called *imputation* where we derive missing values using some property or properties of the data to hand. In this case we assign the mean, but another common mechanism is to use the median as it is less influenced by values a long way from the mean (or outliers).

A final note on some key R data structures (or objects)

Vectors are the most important type of object in R, almost all operations we will do in R are performed on one or more vectors, but there are several others which we will meet more formally in later sessions.

- *matrices* or more generally arrays are multi-dimensional generalizations of vectors. In fact, they are vectors that can be indexed by two or more indices and will be printed in special ways.
- *factors* provide compact ways to handle categorical data. An example of categorical data is Gender, i.e. Male and Female. It is often very useful to have specific ways to interact and represent with data in this format.

- *lists* are a general form of vector in which the various elements need not be of the same type, and are often themselves vectors or lists. Lists provide a convenient way to return the results of a (statistical) computation.
- *data frames* are matrix-like structures, in which the columns can be of different types.Think of data frames as 'data matrices' with one row representing an observation. Imagine standing at a set of traffic lights, and recording every car that passes, each row of the data frame is a car, each column are the properties of the observed car (e.g. colour, no. of doors, make, registered year, registered location, etc.).
- *functions* are themselves objects in R which can be stored in the project's workspace.