Getting Started with R

Notation

- Throughout these slides, text in red denotes commands that you can type in your R session (it is good to type them directly rather than copy and paste to get used to syntax of commands and expressions, at least to begin with)
- Note that commands are shown including the prompt sign > but ignore this when entering the command. For example:

```
> x <- 5
```

should be entered at the command line as:

```
x <- 5
```

- An extra > as well as the in-built R prompt will give an error:

```
> > x <- 5
Error: unexpected '>' in ">"
```

- Commands are shown in red, and output in blue (this is the convention in the standard Windows R GUI but differs in other interfaces)

R basics: command line

Getting used to the command line (in red):

Prompt [Whitespace] Command [Whitespace] Return

```
> 1+2

[1] 3
```

Results are output underneath (in blue). Elements of the output are indexed by the square brackets

- The prompt tells you that R is waiting for you to enter a command – type a valid command and press return or enter
- Some output may be printed to the console (as above) or a new variable may be created for example
- The prompt symbol will be displayed again afterwards
- Sometimes a + appears instead of the usual > prompt symbol. This means the previous line was incomplete while a command that is invalid will generate an error message

R basics: assignment

The following ommand assigns the value 1 to a new variable we create and name 'x'. The assignment operator is <-

```
> x <- 1
# Inspect the contents of the new variable named x
> x
[1] 1
# we can also use = as the assignment operator and
write the command as:
> x = 1
```

Note that the contents of any variable will be overwritten if later assigned something else:

```
> x <- 4
> x
[1] 4
```

Naming variables

- R is case sensitive so x and X are different variables:

```
> X
Error: object "X" not found
```

- Variable names are chosen by the programmer and should start with a letter followed by letters and numbers – informative names are helpful for several reasons
- You can use . or _ to separate parts of a variable name but not whitespace since this is used to detect the end of a token ('word'). A variable named data.norm or d.norm or dataNorm is fine, but trying to assign a value to a variable named 'data norm' will give an error:

```
Error: unexpected symbol in "data norm"
```

- Elsewhere R ignores whitespace so the following commands are equivalent:

```
> x<-4+3
> x <- 4 + 3
```

Command syntax

`> x <- c(1:5,8,9)` # this creates a vector named `'x'` containing some numeric values

- If we forget the closing bracket, a + sign indicates the command is incomplete:

```
> x <- c(1:5,8,9
+ )
>
```

- Some text editor programs highlight different types of syntax in different colours or automatically close brackets and quotation marks to help eliminate typing mistakes
- If we can't simply continue our command, use Esc or control-C to return to the prompt and start again
- There is also a useful command recall option – you can use the up/down arrows to scroll through previously entered commands, which can be edited or re-run to save typing again

Data Types and Structures

Common data structures (object types)

- To do anything useful in R, we need to use **objects** to hold data or information and perform various operations on them
- The terms 'object', 'variable' and 'data structure' can all refer generally to objects created in R. There are several different data structures in R including:
  - Vectors
  - Factors
  - Matrices
  - Dataframes
  - Lists

Common data structures (object types)

- Objects can be created in many different ways and hold different kinds of information
- Unlike other programming languages, there is no need to initialise a variable or object in R (define it before first use) – it can simply be created and used directly

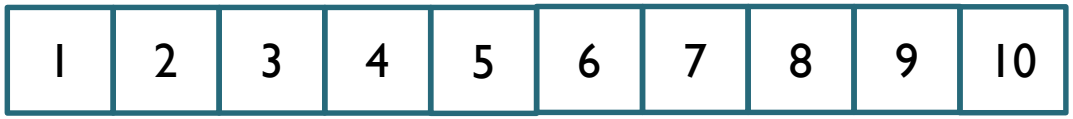
```
> x <- 1:10
> x*2
 [1]  2  4  6  8 10 12 14 16 18 20
```

- We'll work through some examples of each type and look at ways to access or manipulate the data contained within an object
- Be aware that the type (class) of an object and data it contains (numeric, character etc) can affect how it is treated by R

Vectors

- One-dimensional objects with length attribute

```
> x <- 1:10
```




```
> x <- c(1:5, 10:14)
```


```
> length(x)
[1] 10
> x[3]; x[7]; x[1:5]
[1] 3
[1] 11
[1] 1 2 3 4 5
```

Vectors

- Elements automatically added to create vector of appropriate length

```
> x <- 1:15
```

- Can contain numeric or character data

```
> class(x)
[1] "integer"


> x <- c("a", "b", "c")
```


```
> class(x)
[1] "character"
```

R basics: Factors

- Factors are useful for handling categorical data

```
> groups <- rep(c("WT", "MU"), each=3)
> groups
> class(groups) # character vector
> groups <- as.factor(groups) # coerce to factor
> class(groups) # factor
> groups
[1] WT WT WT MU MU MU
Levels: MU WT

> table(groups) # useful summary function giving the
number of entries for each level of the factor

> groups <- factor(groups, levels=c("WT", "MU")) # re-
order the levels (default alphabetical)
```

Matrices

- Two-dimensional object with row and column number attributes

```
> x <- 1:20
> y <- matrix(x, ncol=2, nrow=10)
> dim(y) # get the dimensions
[1] 10  2
> head(y) # shows the first 6 rows
> colnames(y) <- c("Col_1", "Col_2")
> y
```

- All columns of a matrix contain data of the same type. A dataframe is a similar structure but the columns can be a mixture of types – very useful for data with associated descriptive information

Accessing elements of an object

- Square brackets are used to refer to specific elements or subsets of a vector, factor, matrix or dataframe

```
> x <- 1:20 # vector again
> x[5] # print 5th element of x to screen
[1] 5
```

- For a 2 dimensional object need two indices separated by a comma [row, column]

```
> y[2,1] # specific element – second row, column 1
> y[1,] # all of first row
> y[1:5,1] # first 5 entries in column 1
```

Setting the working directory

- There are many useful functions in R for manipulating data stored in vectors, matrices or dataframes - best introduced through practical exercises of the online tutorial
- One important concept before we get started is the **working directory**. If we want to read in data from existing files or create new ones to save any plots or analysis results, R needs to know where to find/save them.

```
> getwd() # tells us the current working directory
> setwd() # allows us to set a different working directory
```

- For the latter, a path to the directory needs to be provided in quotes within the brackets. The path can be full or relative:

```
> setwd("C:/Users/jbloggs/data")
> setwd("./data") # assuming the current working directory is C:/Users/jbloggs (a level above the 'data' directory)
```

Setting the working directory

- The working directory can also be set and changed via the menu bar:
  - Using the R console you find 'Change dir…' under the File menu
  - In RStudio, the Session menu has an option 'Set working directory'.

- From these you can navigate to the desired directory and select OK to set it as the working directory. Note that this happens behind the scenes and nothing will appear to happen on your screen - you can use getwd() to confirm if you have set it as intended.