Week 1 Computer Lab Exercises

Put your group member names and ID numbers here

Due no later than 5pm Tuesday of Week 2 (31 July 2018)

Instructions

This assignment is due on Moodle 5pm Tuesday 31 July 2018. For this initial computer lab assignment, students can choose to either work in a group comprised of up to 4 people, or work individually. Only one submission for each group is required.**

Note that from week 2, all computer lab assignments will be group (only) assignments, and should contain 3 or 4 people. The tutors will finalise the group allocations in the week 2 tutorials.

Follow instructions provided in each section below. By the end, you will need to upload **three (3)** separate files to the Week 1 Lab 2018 Assignment link on Moodle for your group. These files will have names such as:

1. `GroupName_Lab1.Rmd`
2. `GroupName_Lab1.doc` (or `.docx`)
3. `GroupName_Trynew.html`

You must be sure to include all student names and ID numbers for every person in your group on every document or file submitted!

Assessment marks

The final mark for this assessment task will be based on

i. Completion of all tasks outlined in this file (50%);

ii. Completeness and clarity of responses (20%); and

iii. The ability of the submitted .Rmd files to compile immediately without error (reproducibility) (30%).

1. Install R and RStudio

If you haven't already done so, you should install `R` and then `RStudio` on your own machine, if you have one. Check out the **Install `R` and RStudio** instructions on Moodle under the *Computing Resources* section.

If you do not have your own machine, you can borrow one. Check out the **Need to borrow a laptop?** information on Moodle under the *Computing Resouces* section.

2. Use the Week_1_Lab_2018.Rmd file from the *Computer lab exercises* section on Moodle

The printed document you received was produced using the `Week_1_Lab_2018.Rmd` file. You will also be using your (renamed) copy of this .Rmd file to complete your analysis in RStudio!

a. Initiate the `RStudio` application on your computer, and check that all appears as described in the *Install R and RStudio* instruction sheet.

b. Download the `Week_1_Lab_2018.Rmd` file from the unit Moodle page. Save the file in the directory where you want to retain a copies of your work. We will refer to this directory as your *Working Directory*. (The PDF file was produced using the `Week_1_Lab_2018.Rmd` file, compiling it as a Word document first and from there printing as a PDF file. You will also be using this same (renamed) file to complete your Week 1 computer lab exercises in RStudio!)

c. In `RStudio`, go to `File/ Open file...` and select the `Week_1_Lab_2018.Rmd` file you just downloaded.

d. In `RStudio`, go to `Session/ Set working directory/ To source file location`.

e. In `RStudio`, go to `File/ Save as...` to create a new copy of the `Week_1_Lab_2018.Rmd` file and name it using your group name attached with an underscore to `Lab_1` (i.e. **GroupName_Lab_1.Rmd**). Use this new .Rmd file to complete the rest of this section.

f. Immediately put all group members name and ID numbers in the authoring section of your `GroupName_Lab1.Rmd` file.

Learn to install packages

To install an `R` package, use the *Packages* window in the bottom right hand corner of the `RStudio` environment. You can use the `default` location for the *Install to Library* location, and keep the box ticked for *Install dependencies*. If asks you to restart `R` prior to installing these packages, answer *Yes*.

g. **Install the three `R` packages** listed below:

```
knitr

tidyverse

gapminder
```

h. Now type each of the following commands into the `RStudio` *console* window, and hit Return/Enter after each:

```
library(knitr)
library(tidyverse)
library(gapminder)
```

The following `code chunk` will do the same thing, but will be **reproducible** (meaning you can automatically run the file in a new session, without any interference from you, and it will still compile fully and give you the same results!)

```
## Warning: package 'tidyverse' was built under R version 3.4.4

## -- Attaching packages ----------------------------------- tidyverse 1.2.1 --

## v ggplot2 2.2.1     v purrr   0.2.4
## v tibble  1.4.2     v dplyr   0.7.4
## v tidyr   0.8.0     v stringr 1.3.0
## v readr   1.1.1     v forcats 0.3.0

## Warning: package 'ggplot2' was built under R version 3.4.4

## Warning: package 'stringr' was built under R version 3.4.4

## Warning: package 'forcats' was built under R version 3.4.4

## -- Conflicts -------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

## Warning: package 'gapminder' was built under R version 3.4.4
```

Warning messages

You may notice some warning messages appear when you run code, particularly to install packages. Take a look to see what the messages say. Usually the problems are not too serious and you can ignore the issue.

Finding Help

Remember how to find help in `R` and `RStudio` (information is available on the `Install R and RStudio` instruction sheet posted in the *Computing Resources* section on Moodle.) If you need more help, here are some additional suggestions for getting **help from the web**.

- Reading documentation only gets you so far. What about *finding* function(s) and/or package(s) to help solve a problem???
- Google! (I usually prefix "CRAN" to my search; others might suggest http://www.rseek.org/
- Ask your question on a relevant StackExchange outlet such as http://stackoverflow.com/ or http://stats.stackexchange.com/
- Other Resources
  - RStudio cheat sheets
  - Q/A site: http://stackoverflow.com
  - Dynamic Documents with R and knitr, Yihui Xie

Some questions you need to answer

i. What is an `R package`? Write your answers below in the space provided. (Hint: You can use the help pages to learn about these two specific packages. You can also use Google to search for answers!)

Answer here:

j. How does the `library()` function relate to a `package`?

Answer here:

k. **Save your `GroupName_Lab1.Rmd` file** (again, so that it retains your answers above) using File/Save (Ctrl-S) or File/Save As…. You will continue with this file a bit later on, in Section 5.

4. Start a new RMarkdown file

You can keep your `GroupName_Lab1.Rmd` file open in `RStudio` while you complete the next section. You will come back to this file again in Section 5.

a. Open a new Rmarkdown document with a default output format for html. (File -> New File -> `R` Markdown -> html -> OK) **Save and name** this new file **GroupName_Trynew** when you save it (it will automatically get the file extension `.Rmd`).

b. Immediately put all group members name and ID numbers in the authoring section of your `GroupName_Trynew.Rmd` file.

c. Using the **`Knit`** button (there is an icon of a ball of yarn with some knitting needles next to it!) on the top left side of the RMD window the , compile the document into an **html** document. Check that the resulting HTML file appears in your *working directory*.)

d. Next, look at the contents of the `GroupName_Trynew.Rmd` file. Do you know the answers to each of the following questions? (You do not need to write out answers to these questions here, but if you are unsure ask your tutor or a classmate!)

   i) How do you identify the lines of `R` code?
   ii) How does `knitr` know that this is code to be run?
   iii) What is a `code chunk'?
   iv) How do you run a `chunk` of `R` code?
   v) How do you run just one line of `R` code?
   vi) How do you highlight and run multiple lines of code?
   vii) What happens if you try to run a line that starts with ```` ```{r} ````? viii) What happens when you try to run a line of regular text from the document?

e. **Upload** your **GroupName_TryNew.html** file to Moodle as part of your Weekly Computer Lab 1 assignment submission.

5. Get some data

Return now to work with your **GroupName_Lab1.Rmd** file in `RStudio`.

a. Clear out the workspace by first identifying the *Environment* tab in the upper right hand corner window in `RStudio`, and click on the **broom** (yes, the broom icon at the top) icon to empty the objects there. Then re-run the `GroupName_Lab1.Rmd` file again to be sure that the Workspace is refreshed with the correct objects - since you worked in the same environment while wusing your `GroupName_TryNew.Rmd` file.

Data from packages

We are going to be working with **data**. Data can be found in many `R` packages, for example

```
## Observations: 574
## Variables: 6
## $ date     <date> 1967-07-01, 1967-08-01, 1967-09-01, 1967-10-01, 1967...
## $ pce      <dbl> 507.4, 510.5, 516.3, 512.9, 518.1, 525.8, 531.5, 534....
## $ pop      <int> 198712, 198911, 199113, 199311, 199498, 199657, 19980...
## $ psavert  <dbl> 12.5, 12.5, 11.7, 12.5, 12.5, 12.1, 11.7, 12.2, 11.6,...
## $ uempmed  <dbl> 4.5, 4.7, 4.6, 4.9, 4.7, 4.8, 5.1, 4.5, 4.1, 4.6, 4.4...
## $ unemploy <int> 2944, 2945, 2958, 3143, 3066, 3018, 2878, 3001, 2877,...
```

b. Run the above greyed-out code chunk (only) by clicking on the small green triangle (in the upper right hand corner of the code chunk).

The `economics` object is a dataframe. You can look directly at the dataframe by doubleclicking on the `economics` object in the *Environment window* in the upper right corner of RStudio.

Each of the rows of *economics* contains values for a range of available variables.

c. What are the variables in the `economics` dataframe? **List them and explain what they are below.** (Hint: you can use the **help** window to search for information about the **economics** object!)

Answer:

d. Here is another data set, from the `gapminder` package.

```
## Observations: 1,704
## Variables: 6
## $ country   <fct> Afghanistan, Afghanistan, Afghanistan, Afghanistan, ...
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia...
## $ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992...
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.8...
## $ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 1488...
## $ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 78...
```

e. Find out about the `glimpse` function - what does it do? What package it is from? How should we interpret what it produces? Describe each of these below.

Answer(s):

Read in data from a csv file

There are many different functions available for reading in data from data files saved outside of R. The one we'll use here (called `readr`) mimics the base `R` reading functions but is implemented in `C` so reads large files quickly. It also attempts to identify the types of variables in the file, which can be helpful when you are getting to know your data.

Data subdirectory

But first we need to do some housekeeping! It is useful to keep your data files in a subfolder (I call mine 'data') so that you can easily keep track of them.

f. Create a subdirectory named **`Data`** in your working directory, Then download the `Pedestrian_Counts.csv` file from the *Computer lab exercises* section of the Moodle page, and put it in your `Data` subdirectory.

You can pull a data file together yourself, or look at one compiled by someone else. Below is a data file compiled by Professor Di Cook, with original data obtained from the City of Melbourne. Run the code chunk and see how to import the `CSV` file to create a new dataframe in `R`.

g. Run the `code chunk` below. How many rows do you think it contains?

Answer here:

```
## Parsed with column specification:
## cols(
##   Date_Time = col_character(),
##   Sensor_ID = col_integer(),
##   Sensor_Name = col_character(),
##   Hourly_Counts = col_integer()
## )

## Observations: 1,392,618
## Variables: 4
## $ Date_Time     <chr> "01-MAY-2009 00:00", "01-MAY-2009 00:00", "01-MA...
## $ Sensor_ID     <int> 4, 17, 18, 16, 2, 1, 13, 15, 9, 10, 12, 11, 5, 6...
## $ Sensor_Name   <chr> "Town Hall (West)", "Collins Place (South)", "Co...
## $ Hourly_Counts <int> 209, 28, 36, 22, 52, 53, 17, 124, 5, 8, 2, 5, 15...
```

If you know your data file is going to be big, and you just want to look a bit closer at the different types of variables are, you can use the **`n_max=50`** option in the `read_csv` function so you extract the first 50 lines.

h. Run the code chunk below.

```
## Parsed with column specification:
## cols(
##   Date_Time = col_character(),
##   Sensor_ID = col_integer(),
##   Sensor_Name = col_character(),
```

```
##   Hourly_Counts = col_integer()
## )

## Observations: 50
## Variables: 4
## $ Date_Time     <chr> "01-MAY-2009 00:00", "01-MAY-2009 00:00", "01-MA...
## $ Sensor_ID     <int> 4, 17, 18, 16, 2, 1, 13, 15, 9, 10, 12, 11, 5, 6...
## $ Sensor_Name   <chr> "Town Hall (West)", "Collins Place (South)", "Co...
## $ Hourly_Counts <int> 209, 28, 36, 22, 52, 53, 17, 124, 5, 8, 2, 5, 15...

## [1] 1392618       4

## [1] 50  4

## # A tibble: 50 x 4
##    Date_Time         Sensor_ID Sensor_Name                Hourly_Counts
##    <chr>                <int> <chr>                             <int>
##  1 01-MAY-2009 00:00        4 Town Hall (West)                    209
##  2 01-MAY-2009 00:00       17 Collins Place (South)                28
##  3 01-MAY-2009 00:00       18 Collins Place (North)                36
##  4 01-MAY-2009 00:00       16 Australia on Collins                 22
##  5 01-MAY-2009 00:00        2 Bourke Street Mall (South)           52
##  6 01-MAY-2009 00:00        1 Bourke Street Mall (North)           53
##  7 01-MAY-2009 00:00       13 Flagstaff Station                    17
##  8 01-MAY-2009 00:00       15 State Library                       124
##  9 01-MAY-2009 00:00        9 Southern Cross Station                5
## 10 01-MAY-2009 00:00       10 Victoria Point                       8
## # ... with 40 more rows
```

6. Basic arithmetic and other simple operations

The rest of this computer lab is meant to help you to learn how to code in `R`. Work through each topic, and if you have questions use the `help` facility first, then discuss with your group to get their insights and to ensure everyone understands each component.

An `expression` simply requests the evaluation of a mathematical or logical formula.

The `R` *assignment operator* (`<-`) assigns a value to a name, and stores the named object in the `workspace`. You can then retrieve the object when you need it.

a. Run the `code chunk` below.

```
## [1] 12

## [1] 12
```

b. Before you move on, take a look in the upper right-hand pane of the RStudio. Do you see the various objects that you have already created?

Answer here:

c. Write a `code chunk` (`Code/Insert Chunk` or use Ctrl-Alt-i) to delete the object `z` using the **remove** (rm) function. Once completed, check that the object `z` no longer appears in your `workspace`.

Put code chunk below here:

Vectors

To create a vector of any length, you just need to collect together the desired components in order, using the **combine** (c) command. For example:

```
## [1] 1 3 5 7
```

You can use the same `combine` function with non-numeric data, such as text values, as shown below.

d. What happens when you use the `print` function on the object `cool`? Try it!

Answer here:

Sequences

An alternative way to have constructed the numeric vector **a** is to use the **sequence** (seq) function. In this case, as we want **a** to be comprised of the integer numbers starting from 1 and ending with 9, skipping every second integer value. So we can write

```
## [1] 1 3 5 7 9
```

e. Do you get the same result with `a2 <- seq(1, 9, 2)`? **Show by creating a new code chunk** that checks to see if by subtracting **a1** from **a2** you get exactly zero. (Hint: You can check if the subtraction result is **exactly zero** using the relational operator given by **==**.)

Put code chunk below here:

So far the vectors have been relatively short, and have only needed a single line to print all components. Now suppose you wanted to put the integers from 1 to 100 in an object. You can, of course, use `seq(1,100,1)`, but an alternative is to use the **colon** (:) operator to indicate any integer range.

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

f. Use the `colon` operator to list all integers between 5 and 15, including these end points.

Answer here:

Length of a vector

The length of a vector or list can be found using the **length** function.

g. Write a code chunk that uses the `length` function to find the lengths of each of the objects **a**, **b** and `cool`.

Put code chunk below here:

Extracting part of a vector

To extract a portion of an existing vector, supply the index numbers associated with the elements you want to keep inside square brackets following the name of the original vector. For example, to extract only the last ten elements of the vector **b** defined above.

```
##  [1]  91  92  93  94  95  96  97  98  99 100
```

Or, for example, if you wish to extract every element of **b** with an odd index, you can use

```
##  [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45
## [24] 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91
## [47] 93 95 97 99
```

It is also possible to extract elements of a vector to correspond to a condition involving another object. For example, suppose that in addition to **b** we have the vector **d** below.

```
##   [1]  37.5  37.0  36.5  36.0  35.5  35.0  34.5  34.0  33.5  33.0  32.5
##  [12]  32.0  31.5  31.0  30.5  30.0  29.5  29.0  28.5  28.0  27.5  27.0
##  [23]  26.5  26.0  25.5  25.0  24.5  24.0  23.5  23.0  22.5  22.0  21.5
##  [34]  21.0  20.5  20.0  19.5  19.0  18.5  18.0  17.5  17.0  16.5  16.0
##  [45]  15.5  15.0  14.5  14.0  13.5  13.0  12.5  12.0  11.5  11.0  10.5
##  [56]  10.0   9.5   9.0   8.5   8.0   7.5   7.0   6.5   6.0   5.5   5.0
##  [67]   4.5   4.0   3.5   3.0   2.5   2.0   1.5   1.0   0.5   0.0  -0.5
##  [78]  -1.0  -1.5  -2.0  -2.5  -3.0  -3.5  -4.0  -4.5  -5.0  -5.5  -6.0
##  [89]  -6.5  -7.0  -7.5  -8.0  -8.5  -9.0  -9.5 -10.0 -10.5 -11.0 -11.5
## [100] -12.0
```

Now we can select values of **b** for which the corresponding values of **d** satisfy **`a < 0`**.

```
##  [1]  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93
## [18]  94  95  96  97  98  99 100
```

h. Find all months (dates) where the median unemployment is less than 12 months, by writing a suitable `code chunk` below.

Operations

In the previous example we used **subtraction** (-) and **multiplication** (*) operators. In `R` there are five basic arithmetic operators: **addition** (+), subtraction, multiplication, **division** (/) and **exponentiation** (^).

Try out the operators in the following examples.

```
## [1]  5 10 15

## [1] 3 6 1

## [1]  2  2 24

## [1]  1.0 -2.0  0.6

## [1]   1.0   0.5 243.0
```

Note that R is case-sensitive! You should have both objects **A** and **a** currently saved in your workspace. Do you?

Answer:

Matrices

The **matrix** function (`matrix(Y, nrow=r, ncol=c)`) enables you to construct a matrix, putting the elements of a vector **Y** into a matrix having `r` rows and `c` columns. For example

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

Note that the elements of the vector `1:6` enter the matrix **Y** filling in the first column ([,1]) before continuing on to fill in the second column ([,2]). This is because the default is to fill the matrix by column, as indicated by the `byrow = FALSE` default in the `matrix` command. (You may want to review the `help file` for `matrix`.) To change this to have the elements of the vector fill in the matrix by row rather than by column, we need to include `byrow=TRUE` (or simply `byrow = T`) as a fourth argument to the `matrix` function. In this case a different matrix results

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
```

Matrix rows are indicated by `[r,]`, where `r` indicates the relevant row number, and matrix columns are indicated by `[,c]`, where `c` indicates the relevant column number.

i. Write a code chunk to find the 3rd row of the matrix **Y**, by selecting only the 3rd row with the subsetting indicator `[3,]`.

Put code chunk below here:

Similar to the case for vectors, if you wanted to extract a particular submatrix, you just need to express which rows and columns you want to extract by providing the relevant index sequence. To extract particular rows, insert the row indices, as a vector, in the `r` place inside the square brackets `[r,c]` following the name of the original matrix, and and leave the column specification in the `c` place blank.

j. Write a code chunk to extract all columns of the second and third rows of **Y** only.

Put code chunk below here:

Dimension of an object

The dimension of an object, such as a matrix, is found using the **dimensions** (dim) command. In the case of a matrix, the result of`dim(matrix)` is a two dimensional vector containing the number of rows and columns, respectively, of the matrix in question.

k. Write a code chunk to find the dimensions of the `economics` dataframe, and of the matrix `Y`.

Put code chunk below here:

l. Once you have completed answering all questions, **Knit to Word** the entire .Rmd file. This will automatically save the .Rmd and also will *hopefully* produce your `GroupName_Lab1.doc` (or .docx) Word document - look for it in your `working directory`.

m. Save your `GroupName_Lab1.doc` (or `.docx`) file.

Turn in your work!

Then, upload to the Week 1 Lab 2018 Assignment link on Moodle each of the following:

1. Your `GroupName_Trynew.html` file
2. Your `GroupName_Lab1.Rmd` file
3. Your `GroupName_Lab1.doc` (or `.docx`) file

PLEASE NOTE!!

- **Make sure ALL of your group members are listed as authors on all files.** If a name and/or ID number is not listed on the submitted files, the **whole group** may lose credit.
- Make sure there are **no typos** in either your Group Name OR in any of the student ID numbers provided in each file.
- Turn in ONLY **one set of files per group**!!

DUE: No later than 5pm on Tuesday of Week 2 (31 July 2018)