Worksheet 2 Importing data and some basics

This worksheet explains how to import data sets into R and perform some basic analysis on the data.

We start by installing the Tidyverse:

install.packages(“tidyverse”)

library(tidyverse)

**Organising Data**

It is important to organise your data in a way that is compatible with the data analysis functions used in **R**. You should ensure that:

* each variable has its own column
* each row is an observation
* the top of each column contains the name of the variable
* there are no blank columns or blank rows between data
* data are consistent (e.g. if a binary variable can take values ‘Yes’ or ‘No’ then only these two values are allowed, with no alternatives such as ‘Y’ and ‘N’)

**Dataframes/Tibbles**

It is possible to import data into **R** from a wide range of formats e.g.

* CSV files
* Excel files
* SPSS, SAS, Matlab and Stata files

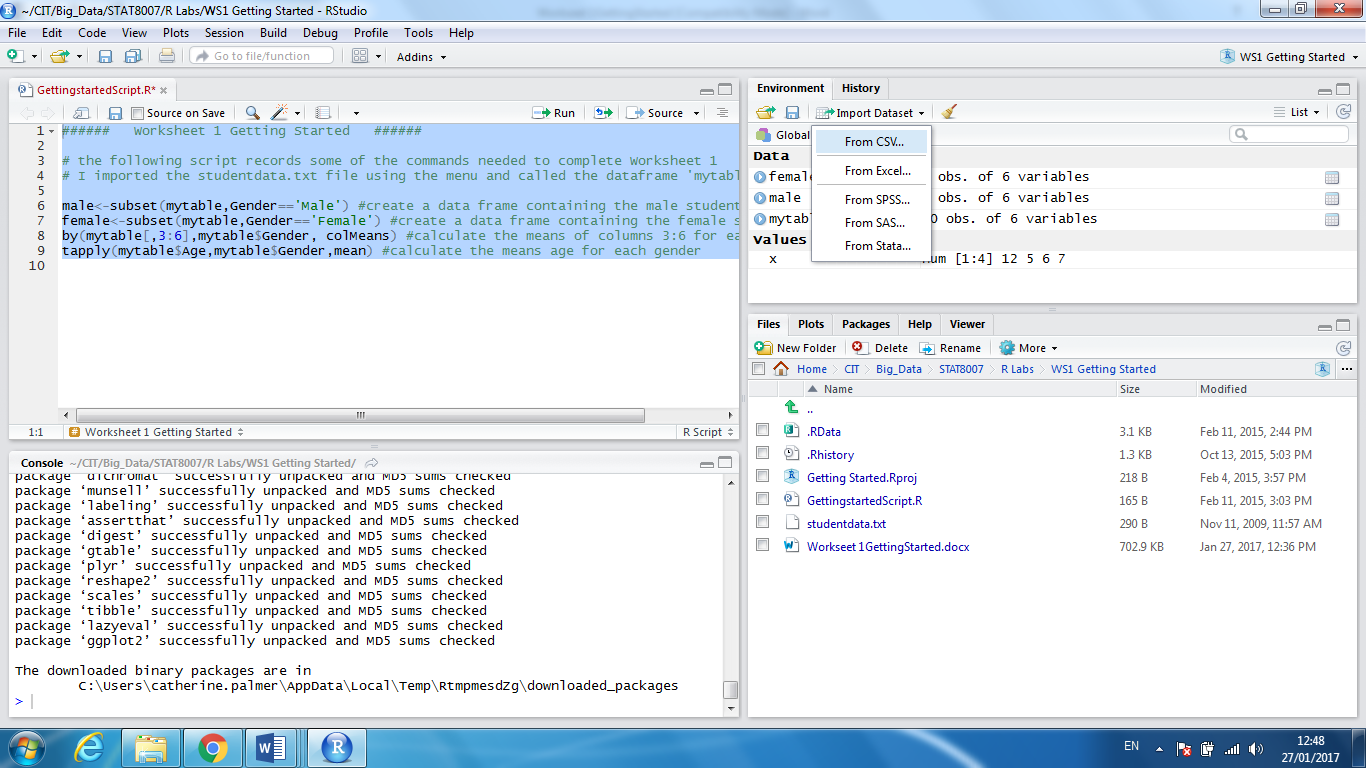
Up until recently, the most common way for **R** to handle data sets was in objects known as dataframes. A **dataframe** is an object with rows and columns. A more recent version of a dataframe is a **tibble**. Using a tibble avoids the inconsistencies that can occur when using dataframes. The rows of a dataframe/tibble contain different objects (or subjects) from the study. The columns contain values of different variables, often called fields. For example we might have the following table of data. Here we have information on 6 variables relating to 10 subjects. An Excel file containing this data is available on Blackboard.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Gender | Age | Maths | Programming | History |
| Amy | Female | 22 | 45 | 50 | 86 |
| Breda | Female | 25 | 65 | 55 | 65 |
| Colm | Male | 20 | 83 | 71 | 40 |
| Denise | Female | 23 | 46 | 38 | 69 |
| Edwin | Male | 27 | 85 | 69 | 50 |
| Frank | Male | 31 | 51 | 66 | 51 |
| Gina | Female | 25 | 75 | 54 | 58 |
| Harry | Male | 22 | 51 | 33 | 69 |
| Ivan | Male | 28 | 44 | 57 | 44 |
| Joseph | Male | 30 | 38 | 44 | 58 |

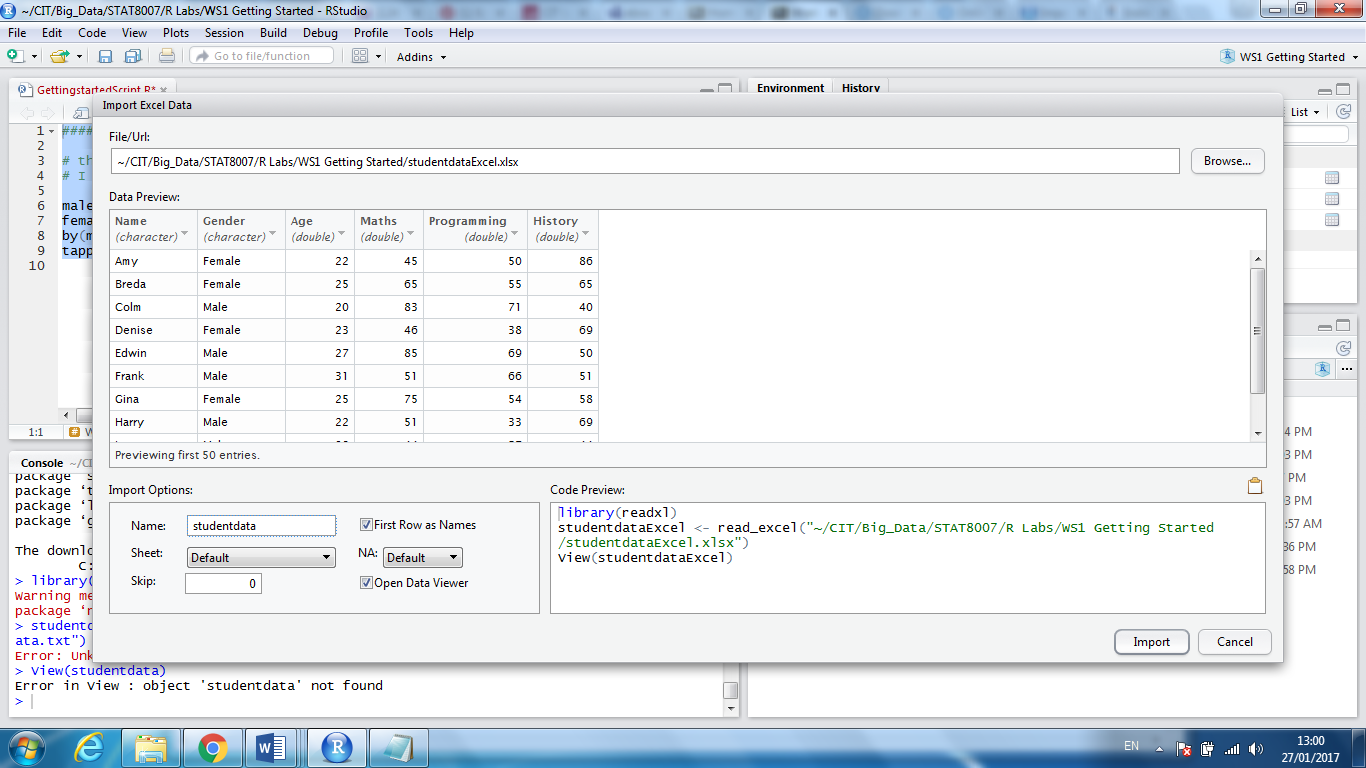
How can we import this data into **R**?

There are many different ways to import data into R, one of the simplest methods is to use the Import Dataset icon in the top right. You can then browse windows to find your file.

Import data set > From Excel



Name the data frame ‘studentdata’



If you are going to work on the same data set frequently then it is more efficient to load it in using a function. One of the most flexible ways to import data into **R** is to use the readr package from the tidyverse. It contains different functions for importing data, including:

* read\_csv() reads comma delimited files
* read\_csv\_2 reads semicolon separated files
* read\_tsv() reads tab-delimited files
* read\_delim() reads in files with any delimiter (you need to specify the type)

To import data from Excel we need the separate readxl package.

install.packages(“readxl”)

library(readxl)

studentdata < -read\_excel(“filepath\unique\to\each\computer\studentdata.xlsx”)

Importing data into R using a function can be extremely frustrating since a single mistake in the file path will prevent R from finding the specified file. The here() function can be very useful when importing data as it automatically constructs the path to the projects files. If all the relevant data files are stored in the current project then we simply type the code below. This is extremely useful if you are using more than one computer to work on a project.

install.packages(here)

library(here)

studentdata <- readxl(“studentdata.txt”)

studentdata <- read\_excel(here("studentdataExcel.xlsx"))

Next we can import the data again but this time from a CSV file.

studentdata <- read\_csv(here("studentdata\_csv.csv"))

Parsed with column specification:

cols(

Name = col\_character(),

Gender = col\_character(),

Age = col\_integer(),

Maths = col\_integer(),

Programming = col\_integer(),

History = col\_integer()

)

When you run read\_csv() it prints out a column specification that gives the name and type of each column.

Sometimes there are a few lines of metadata at the top of the file. You can use skip = n to skip the first n lines. In the code below the first wo rows of the data are omitted.

studentdata <- read\_csv(here("studentdata\_csv.csv"), skip = 2)

If you would like to add column names, you can pass col\_names a character vector which will be used as the column names.

studentdata <- read\_csv(here("studentdata\_csv.csv"), col\_names = c("a","b","c","d","e","f"))

The command above will simply add a new row of names onto your data which is fine if there were no names to begin with. If there are names that you would like to replace, use this in conjunction with the skip argument.

studentdata <- read\_csv(here("studentdata\_csv.csv"), col\_names = c("a","b","c","d","e","f")skip = 2)

Another option that commonly needs tweaking is na, this specifies the value (or values) that are used to represent missing values in your file:

studentdata <- read\_csv(here("studentdata\_csv.csv"), na = “?”)

Note that all the values that had the value ? are now replaced by NA.

**Creating Tibbles**

Recall, a Tibble is a modern version of a dataframe that is more compatible with the tidyverse and behaves in a more consistent manner. Older packages are designed to use dataframes so it can be useful to convert back and forth between a dataframe and a tibble:

as.data.frame(studentdata)

as\_tibble(studentdata)

To look at the tibble you can select it from the Global environment tab or type:

studentdata

**R** will print out the table as follows.

A tibble: 10 x 6

Name Gender Age Maths Programming History

*<chr>* *<chr>* *<int>* *<int>* *<int>* *<int>*

1 Amy Female 22 45 50 86

2 Breda Female 25 NA 55 NA

3 Colm Male 20 83 71 40

4 Denise Female 23 46 38 69

5 Edwin Male 27 85 69 50

6 Frank Male 31 51 66 51

7 Gina Female 25 75 54 58

8 Harry Male 22 51 33 69

9 Ivan Male 28 44 57 44

10 Joseph Male 30 38 44 58

The data is stored in the object studentdata and by typing its name at the prompt you are asking R to display the object.

We can create a new tibble from individual vectors with tibble()

x <- c(1,2,3,4,5)

y <- c(1,1,1,1,1)

z <- c(1,5,10,17,26)

tibble(x,y,z)

tibble() will automatically recycle inputs of length 1, and allows you to refer to variables that you just created, as shown below.

**Basic summary statistics on the studentdata object**

names(studentdata)

This requests **R** to print out the column names, the names of the attributes and the variables for the data and sure enough the labels are printed.

[1] "Name" "Gender" "Age" "Maths" "Programming"

[6] "History"

We might want to look at the Age variable and it would seem reasonable to try

Age

Error: object "Age" not found

Why did **R** not find the Age data for us? The reason is that the Age data is part of studentdata and the Age variable does not have an independent existence in **R**.

Instead try:

studentdata$Age

We obtain the ages of the 10 subjects.

[1] 22 25 20 23 27 31 25 22 28 30

Similarly

studentdata$Gender

will give us the genders of the subjects.

[1] Female Female Male Female Male Male Female Male Male Male

Levels: Female Male

We can perform basic statistical analysis on the Age variable.

mean(studentdata$Age) #Computes the mean age

[1] 25.3

median(studentdata$Age) #Computes the median

[1] 25

max(studentdata$Age) #Gives the maximum

[1] 31

min(studentdata$Age) #Gives the minimum

[1] 20

with(studentdata, range(Age)) #Gives the range

[1] 20 31

sd(studentdata$Age) #Computes the standard deviation

[1] 3.653005

summary(studentdata$Age)

Min. 1st Qu. Median Mean 3rd Qu. Max.

20.00 22.25 25.00 25.30 27.75 31.00

This last command gives summary statistics in one swoop.

We can look at some basic summary statistics of the data contained in studentdata using

summary(studentdata)

Name Gender Age Maths Programming

Length:10 Length:10 Min. :20.00 Min. :38.00 Min. :33.00

Class :char Class :char 1st Qu.:22.25 1st Qu.:45.00 1st Qu.:45.50

Mode :char Mode :char Median :25.00 Median :51.00 Median :54.50

Mean :25.30 Mean :57.56 Mean :53.70

3rd Qu.:27.75 3rd Qu.:75.00 3rd Qu.:63.75

Max. :31.00 Max. :85.00 Max. :71.00

NA's :1

History

Min. :40.00

1st Qu.:50.00

Median :58.00

Mean :58.33

3rd Qu.:69.00

Max. :86.00

NA's :1

Values of quantitative variables are summarized under six headings, the minimum, first quartile, median, mean, third quartile and maximum).

**Indexing into a tibble/dataframe**

A tibble is a two dimensional object consisting of rows and columns. The rows are referred to by the first (left-hand) subscript and the columns by the second (right-hand) subscript. Therefore:

studentdata[2,3]

[1] 25

is the value of Age (the variable in column 3) in row 2. To extract a range of values, say the 4th to 7th rows from Programming (the variable in the fifth column), we use the colon operator : to generate a series of subscripts (4, 5, 6 and 7):

studentdata[4:7, 5]

[1] 38 69 66 54

To extract a group of rows and a group of columns, you need to generate a series of subscripts for both the row and column subscripts. Suppose we want Maths and Programming from rows 1 to 5:

studentdata[1:5,4:5]

Maths Programming

1 45 50

2 65 55

3 83 71

4 46 38

5 85 69

To select all the entries in a row we leave the right-hand entry blank so to select the entire second row

studentdata[2,]

Name Gender Age Maths Programming History

2 Breda Female 25 65 55 65

Similarly to select all the entries in a column we leave the left hand entry blank. To select all the entries from the third column :

studentdata[,3]

[1] 22 25 20 23 27 31 25 22 28 30

**Subsetting a tibble/dataframe**

First we create a data frame:

ID <- c(1, 2, 3, 4, 5, 6)

Type <- c("A", "A", "C", "A", "B", "C")

Temp <- c(24, 67, 91, 56, 72, 80)

data <- data.frame(ID, Type, Temp) # data is a data frame

To select entries where temperature has a value greater than 70:

newdata<-subset(data, data$Temp>70)

To select only the variables ID and Type using the variable names we can use:

newdata <- subset(data,select=c(ID,Type))

In addition to the subset command we can also use [ ] as we did for indexing. To select only the variables ID and Type using the column numbers we can use:

newdata <- data[,1:2]

Note that the rows are indexed in the space before the comma, in this example we select all rows by leaving it blank. In the example below we extract the first three rows only and the first two columns only.

newdata <- data[1:3,1:2]

Excluding Data

newdata <- mydata[,-3] # exclude 3rd variable Type

mydata$Type <- NULL # delete 3rd variable Type

The **by** function

The function **by** enables us to summarize the data frame on the basis of the qualitative variables. For instance, to know the means of each of the numeric variables by gender we use:

by(studentdata[,3:6],studentdata$Gender,colMeans, na.rm = T)

studentdata$Gender: Female

Age Maths Programming History

23.75000 55.33333 49.25000 71.00000

-------------------------------------------------------------------------

studentdata$Gender: Male

Age Maths Programming History

26.33333 58.66667 56.66667 52.00000

The by command needs three arguments, first the data frame (here we only included the numeric data found in columns 3-6), second the qualitative variable (the factor) used to split the data and third the function to be applied to the data. Note that the na.rm =T argument omits NA values from the calculation.

**Sorting and Ranking**

We can sort, rank and order the studentdata using a vector of choice, for example we can sort, rank and order studentdata by the vector called Maths

sorted<-sort(studentdata$Maths)

ranks<-rank(studentdata$Maths)

we have created two new vectors containing the sorted and ranked data that we can turn into a dataframes

sorted<-data.frame(sorted)

ranks<-data.frame(ranks)

We can then join the data frames into a single data frame:

x<-cbind(studentdata$Maths, sorted,ranks)

#the command cbind joins columns together

studentdata.Maths sorted ranks

1 45 38 3.0

2 65 44 7.0

3 83 45 9.0

4 46 46 4.0

5 85 51 10.0

6 51 51 5.5

7 75 65 8.0

8 51 75 5.5

9 44 83 2.0

10 38 85 1.0

The sorted vector contains the values of Maths sorted in ascending order. If you want to sort into descending order, use the reverse order function rev

The values in Maths are in no particular sequence. The ranks column contains the value that is the rank of the particular data point (mark obtained in Maths), where 1 is assigned to the lowest data point and length(Maths) – here 10 – is assigned to the highest data point. So the first element Maths = 45 is the third lowest value in Maths. Fractional ranks indicate ties. There are two values of 51 in Maths and their ranks are 5 and 6 but because they are tied, each gets the average of their two ranks (5+6)/2 = 5.5

y <-rev(sort(studentdata$Maths))

Note that sort can cause problems because it uncouples values that might need to be in the same row of the dataframe. Always assign the sorted values to a new variable (we used the variable sorted) and never use:

x <-sort(x)

which assigns the sorted values to the original variable.