CS5702 W2 Worksheet Notebook

Martin Shepperd

10/10/2021

#Week 2 Worksheet

0. Worksheet Topics

1. Seminar: Analysing your questionnaire
2. Lab: Some More R practice
3. Selected Answers

1. Analysing the Joining Questionnaire

For this example we will take an *exploratory* approach (for more on EDA see Chapter 4 of the Modern Data Book).

1.1 Questionnaire overview

You were asked to complete this short, anonymous questionnaire and to critically reflect on the questions.

This questionnaire has been created using Google Forms and the link is https://forms.gle/ZzRAfZvQxwSgVxM28.

I used Google Forms for (i) simplicity, (ii) it's free and (iii) you can easily export the results as a csv file. We can then import this into R and do the analysis.

Exercise 1.1: Why bother to write R code at all, surely it will be easier to just use a spreadsheet? How would you answer this question?

1.2 Questionnaire Design

Exercise 1.2: What are the key design decisions?

Exercise 1.3: What possible ethical issues might arise?

Exercise 1.4: How could the questionnaire be improved?

1.3 A short story with a moral

For the Home Country question (Q4) I wanted a pull down menu so the should be constrained (e.g., to avoid UK, United Kingdom, GB, England, etc). But there are many countries and it feels like this would be a boring and error-prone task to manually copy and paste each country name into Google Forms.

I used StackExchange because I reasoned (i) many other people must want to get a country response and (ii) will use a drop-down list.

The moral is, for almost every task, someone will have solved it before you so it's worth a quick google before you implement something from scratch.

1.4 Fetch and clean the data

For simplicity the csv file name is pointed at a recent version I extracted from Google forms and stored on GitHub. Let's start by fetching the data from GitHub.

First we must fetch the raw data using a convenient built in function called `read.csv()` that imports the data into an R data frame. The options indicate the file has headers which can be used as variable names.

```
# Fetch the raw data CS5702JoiningSurvey_2021.csv from GitHub.
# Read the csv file into a new data frame called surveyDF
surveyFileName <- "https://raw.githubusercontent.com/mjshepperd/CS5702-Data/master/CS5702JoiningSurvey_
surveyDF <- read.csv(surveyFileName, header = TRUE, stringsAsFactors = FALSE, fileEncoding = 'UTF-8-BOM
```

A simple starting point is to *"eyeball"* the imported data. If you look in the Environment Pane in RStudio you can see a little table icon by the data frame surveyDF. (Alternatively you could run the R function `View()` directly.) As a starting point we should check to see if the values appear 'sensible'.

```
# Get a 'spreadsheet' type view of our data.
View(surveyDF)
```

Looking at the data frame surveyDF we can see Google has used the questions as column headers which have then been interpreted as variable names in the data frame. Unfortunately this means they're rather clunky so let's clean this up.

```
# Clean up surveyDF column names

names(surveyDF)  # The names() function returns all the dataframe column names
```

```
##  [1] "Timestamp"
##  [2] "X"
##  [3] "Q1..Which.MSc.will.you.be.studying."
##  [4] "Q2..What.is.your.study.mode."
##  [5] "Q3..What.subject..or.the.main.subject..did.you.study.for.your.first.degree..Please.choose.the.
##  [6] "Q4..What.would.you.consider.to.be.your.home.country."
##  [7] "Q5..Please.indicate.your..chief..motivation.for.undertaking.the.MSc.."
##  [8] "Q6..How.would.you.describe.your.understanding.of.statistics."
##  [9] "Q7..How.confident.are.you.at.programming."
## [10] "Q8..Please.indicate.any.programming.languages.with.which.you.have.basic.familiarity."
## [11] "Q9..What.are.you.most.excited.about.regarding.this.module..Modern.Data.."
## [12] "Q10..What.concerns.you.the.most.regarding.this.module."
```

```
# Choose shorter, clearer names
# For clarity I've chosen a risky approach of using column position
# rather than the very long original variable name.
names(surveyDF)[3] <- "MSc"
names(surveyDF)[4] <- "StudyMode"
names(surveyDF)[5] <- "Background"
names(surveyDF)[6] <- "Country"
names(surveyDF)[7] <- "Motivation"
names(surveyDF)[8] <- "StatsKnowl"
names(surveyDF)[9] <- "ProgKnowl"
names(surveyDF)[10] <- "ProgLangs"
names(surveyDF)[11] <- "Excite"
names(surveyDF)[12] <- "Concern"
```

Remove column X as this is redundant (in fact an error on my part, the lesson being check and double check your questionnaires!)

```
# Drop X using set difference with '-' operator
surveyDF = subset(surveyDF, select = -c(X))
```

We also note some values (as well as the variable names which we've already dealt with) are extremely long and a bit clunky. We can replace them with something a little more succinct.

```
# Shorten the MSc titles (for convenience)
surveyDF[surveyDF=="MSc Artificial Intelligence (AI)"] <- "AI"
surveyDF[surveyDF=="MSc Data Science Analytics (DSA)"] <- "DSA"
```

We can check the counts and values using the `table()` function which computes frequencies for each category (or level).

```
table(surveyDF$MSc)
```

```
##
##                  AI      Apprenticeship                 DSA                 DTS
##                   7                   1                  24                   1
## DTSS Apprenticeship
##                   1
```

Now we notice something slightly unexpected. In addition to AI and DSA there are three other categories (at the time of writing, though this might change as new responses are received). These have come about because (i) the survey as a catch-all allows other as a response (ii) other is unconstrained and (iii) I forgot that there is a new course in 2021-22.

```
# Fix the unconstrained names for the Apprenticeship route
surveyDF[surveyDF=="DTS"] <- "Apprenticeship"
surveyDF[surveyDF=="DTSS Apprenticeship"] <- "Apprenticeship"

# Re-check the frequencies and categories
table(surveyDF$MSc)
```

```
##
##             AI Apprenticeship            DSA
##              7              3             24
```

Exercise 1.5: Adapt the above code, or write your own, to simplify the values (known as Levels in R) for Motivation.

```
# Add your R code here
```

1.5 Data frame structure and data quality checking

Exercise 1.6: Make a list of things we should check for when trying to assure the quality of our survey data.

One common problem is **missing** data is frequently a problem for data analysis. It is so important we will revisit the topic in detail in Week 5 on Data Cleaning. In R missing observations should be denoted `na`. Note that this is different to `NaN` (or Not a Number) which refers to the result of a computation e.g., `sqrt(-1)`. Try it!

```
# An example computation that yields NaN i.e., an irrational number
sqrt(-1)
```

```
## Warning in sqrt(-1): NaNs produced
```

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

Whilst it might seem a technicality, the importing function read.csv() treats an empty string "" as such, i.e., it's still a value since the survey respondent has chosen not to enter any reply as opposed to they never saw the question. Remember, when a question isn't mandatory, the survey permits no answer as an answer instead of text!

```
# A *very* simple way to check for missing observations in our data frame
# using the is.na() function

is.na(surveyDF)
```

```
##    Timestamp   MSc StudyMode Background Country Motivation StatsKnowl ProgKnowl
## 1      FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 2      FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 3      FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 4      FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 5      FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 6      FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 7      FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 8      FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 9      FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 10     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 11     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 12     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 13     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 14     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 15     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 16     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 17     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 18     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 19     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 20     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
```

```
## 21     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 22     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 23     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 24     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 25     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 26     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 27     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 28     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 29     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 30     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 31     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 32     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 33     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
## 34     FALSE FALSE     FALSE      FALSE   FALSE      FALSE      FALSE     FALSE
##    ProgLangs Excite Concern
## 1      FALSE  FALSE   FALSE
## 2      FALSE  FALSE   FALSE
## 3      FALSE  FALSE   FALSE
## 4      FALSE  FALSE   FALSE
## 5      FALSE  FALSE   FALSE
## 6      FALSE  FALSE   FALSE
## 7      FALSE  FALSE   FALSE
## 8      FALSE  FALSE   FALSE
## 9      FALSE  FALSE   FALSE
## 10     FALSE  FALSE   FALSE
## 11     FALSE  FALSE   FALSE
## 12     FALSE  FALSE   FALSE
## 13     FALSE  FALSE   FALSE
## 14     FALSE  FALSE   FALSE
## 15     FALSE  FALSE   FALSE
## 16     FALSE  FALSE   FALSE
## 17     FALSE  FALSE   FALSE
## 18     FALSE  FALSE   FALSE
## 19     FALSE  FALSE   FALSE
## 20     FALSE  FALSE   FALSE
## 21     FALSE  FALSE   FALSE
## 22     FALSE  FALSE   FALSE
## 23     FALSE  FALSE   FALSE
## 24     FALSE  FALSE   FALSE
## 25     FALSE  FALSE   FALSE
## 26     FALSE  FALSE   FALSE
## 27     FALSE  FALSE   FALSE
## 28     FALSE  FALSE   FALSE
## 29     FALSE  FALSE   FALSE
## 30     FALSE  FALSE   FALSE
## 31     FALSE  FALSE   FALSE
## 32     FALSE  FALSE   FALSE
## 33     FALSE  FALSE   FALSE
## 34     FALSE  FALSE   FALSE
```

```
# We can also count as the above produces a large table is hard to read
paste("The count of missing observations is:",sum(is.na(surveyDF)))
```

```
## [1] "The count of missing observations is: 0"
```

The `is.na()` function returns True or False depending on whether the observation is present or not. In our case there are no missing values so we only see FALSE results, and the count is zero.

Note that almost always when we want to refer to a specific variable **in** a data frame we use the `$` notation e.g., `<data frame name>$<column name>`.

It is particularly important to think about the data types of the different variables in our data frame. One quite easy way to look at the entire data frame is to use the `str()` function. (str is short for structure obvs!)

```
# Examine the data frame structure
str(surveyDF)
```

```
## 'data.frame':    34 obs. of  11 variables:
##  $ Timestamp : chr  "2021/09/22 11:44:56 am CET" "2021/09/22 2:24:38 pm CET" "2021/09/23 12:58:43 pm
##  $ MSc       : chr  "AI" "DSA" "AI" "DSA" ...
##  $ StudyMode : chr  "Full-time" "Full-time" "Part-time" "Full-time" ...
##  $ Background: chr  "Computer science" "Life Sciences BSc" "Computer science" "Natural sciences e.g.
##  $ Country   : chr  "United Kingdom" "United Kingdom" "United Kingdom" "Cameroon" ...
##  $ Motivation: chr  "As a step towards doing original research" "Enhance overall employability" "Enha
##  $ StatsKnowl: int  3 6 4 8 5 6 5 7 5 6 ...
##  $ ProgKnowl : int  3 8 8 6 1 7 2 7 9 9 ...
##  $ ProgLangs : chr  "Java or javascript;C or C++;Basic or Visual Basic" "Python;Java or javascript;R
##  $ Excite    : chr  "I am keen to know more about managing and retrieving data and explore industry
##  $ Concern   : chr  "My biggest concern would be the programming side as I'm not very great when it
```

Some of the variables are going to be used as categories e.g., `MSc` and `StudyMode`; we might want to compare FT and PT responses, etc. This is much easier if we make the categorical variable a **Factor**. These are categorical variables generally with relatively few distinct values or **Levels**.

```
# Changing MSc from a character type to a Factor
surveyDF$MSc <- as.factor(surveyDF$MSc)

# Changing StudyMode from a character type to a Factor
surveyDF$StudyMode <- as.factor(surveyDF$StudyMode)
```

Exercise 1.7: How can you check that your code has been successful? HINT: there are several ways you could accomplish this.

1.6 Some summary statistics

It's usually a good idea to start with some basic summary (descriptive) statistics to give a basic feel for the data.

```
# Using the summary() function on the numeric variables
summary(surveyDF$StatsKnowl)
```

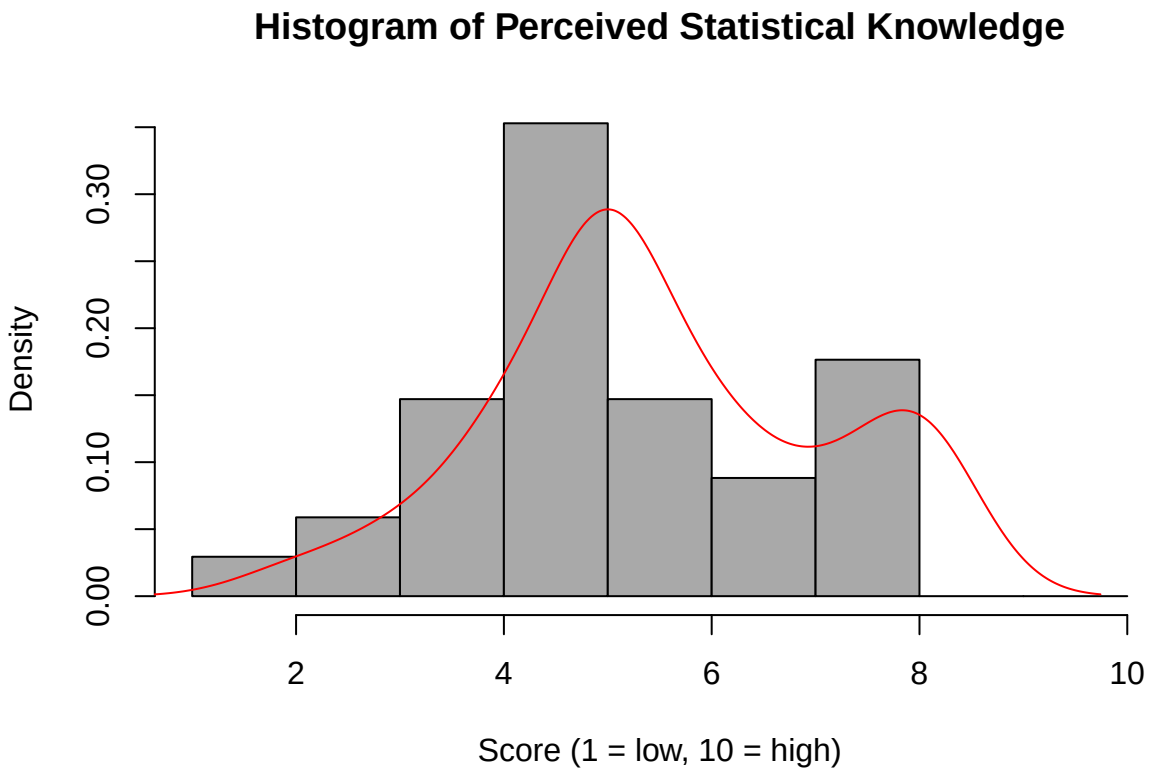
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00    5.00    5.00    5.50    6.75    8.00
```

```
summary(surveyDF$ProgKnowl)
```

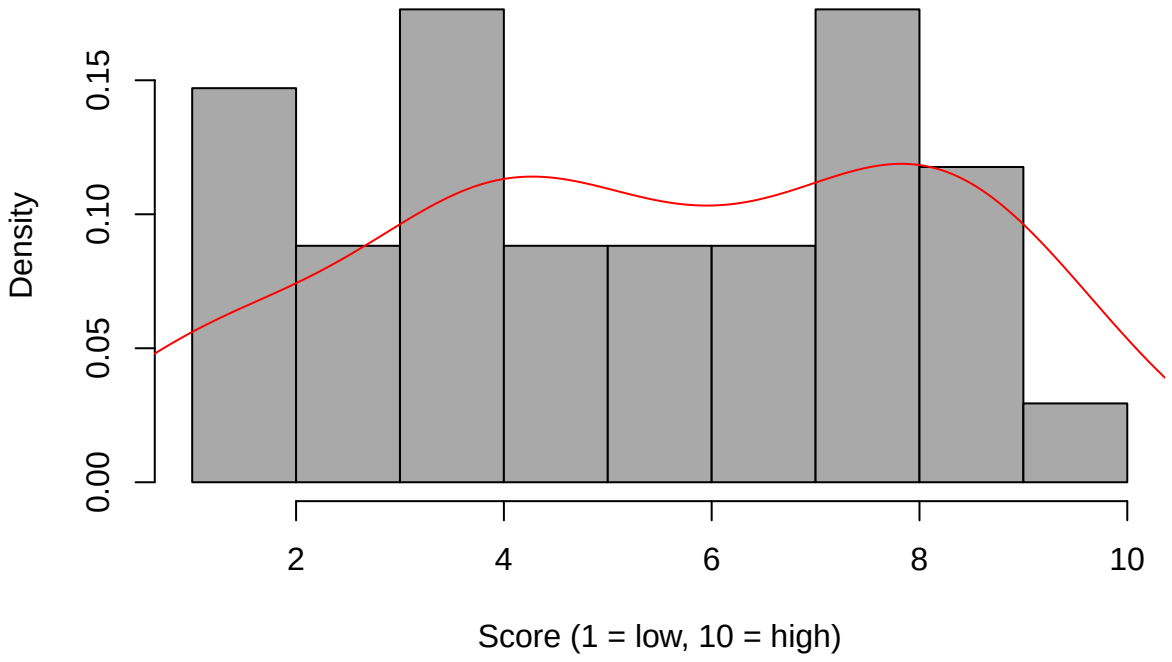
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0     4.0     5.5     5.5     8.0    10.0
```

```
# And easier to visualise
# I superimpose the a probability density plot over the histogram

hist(surveyDF$StatsKnowl,
     main="Histogram of Perceived Statistical Knowledge",
     xlab="Score (1 = low, 10 = high)",
     col="darkgray",
     xlim=c(1,10),
     breaks=1:10,
     prob = TRUE
     )
lines(density(surveyDF$StatsKnowl), col = "red")
```




```
hist(surveyDF$ProgKnowl,
     main="Histogram of Perceived Programming Knowledge",
     xlab="Score (1 = low, 10 = high)",
     col="darkgray",
     xlim=c(1,10),
     breaks=1:10,
     prob = TRUE
     )
lines(density(surveyDF$ProgKnowl), col = "red")
```



NB The above plots are generated using Base R. In subsequent weeks we will move to {ggplot} which is far more powerful, but also a little more complex.

For the character string variables we can use the `table()` function to produce frequency counts of categories. This makes no sense when we deal with longer text (since each input will almost certainly be unique in phrasing if not in sentiment). Similarly the `ProgLangs` variable is problematic because it is multi-valued i.e., it may contain zero or more programming languages. This is a challenge we will return to in Week 5 on Data Cleaning.

```
# Use the table() function to produce some tables of frequency counts
table(surveyDF$MSc)
```

```
##
##             AI Apprenticeship            DSA
##              7              3             24
```

```
table(surveyDF$StudyMode)
```

```
##
## Full-time Part-time
##        25         9
```

```
# A 2x2 contingency table of MSc by Study Mode
table(surveyDF$MSc,surveyDF$StudyMode)
```

```
##
##                  Full-time Part-time
##   AI                     5         2
##   Apprenticeship         0         3
##   DSA                   20         4
```

1.7 Multivariate analysis

Here we start exploring the relation ships between more than one variable.

For example, is there any relationship between `Background` [UG course] and choice of MSc?

```
# This is an example of a 2-d table of frequencies.
table(surveyDF$Background, surveyDF$MSc)
```

```
##
##                                          AI Apprenticeship DSA
##   Arts BA, CIM level 6 diploma marketing  0              1   0
##   Biomedical Science                      0              0   1
##   Business studies                        0              0   3
##   Computer science                        6              0   6
##   Economics                               0              0   1
##   Engineering                             1              0   2
##   Humanities e.g., geography or history   0              1   0
##   ICT and MM                              0              0   1
##   Life Sciences BSc                       0              0   1
##   Maths                                   0              0   2
##   Media                                   0              0   1
##   Natural sciences e.g. chemistry or biology 0           0   3
##   Pharmacy                                0              0   1
##   Physics & Philosophy                    0              1   0
##   Statistics                              0              0   2
```

```
# and if we want marginal totals wrap table() with the addmargins() function
addmargins(table(surveyDF$Background, surveyDF$MSc))
```

```
##
##                                          AI Apprenticeship DSA Sum
##   Arts BA, CIM level 6 diploma marketing  0              1   0   1
##   Biomedical Science                      0              0   1   1
##   Business studies                        0              0   3   3
##   Computer science                        6              0   6  12
##   Economics                               0              0   1   1
##   Engineering                             1              0   2   3
##   Humanities e.g., geography or history   0              1   0   1
##   ICT and MM                              0              0   1   1
##   Life Sciences BSc                       0              0   1   1
##   Maths                                   0              0   2   2
##   Media                                   0              0   1   1
##   Natural sciences e.g. chemistry or biology 0           0   3   3
##   Pharmacy                                0              0   1   1
##   Physics & Philosophy                    0              1   0   1
##   Statistics                              0              0   2   2
##   Sum                                     7              3  24  34
```

```
# and if you want proportions then wrap with prop.table()
prop.table(table(surveyDF$Background, surveyDF$MSc))
```

```
##
```

```
##                                                        AI Apprenticeship
##   Arts BA, CIM level 6 diploma marketing     0.00000000     0.02941176
##   Biomedical Science                         0.00000000     0.00000000
##   Business studies                           0.00000000     0.00000000
##   Computer science                           0.17647059     0.00000000
##   Economics                                  0.00000000     0.00000000
##   Engineering                                0.02941176     0.00000000
##   Humanities e.g., geography or history      0.00000000     0.02941176
##   ICT and MM                                 0.00000000     0.00000000
##   Life Sciences BSc                          0.00000000     0.00000000
##   Maths                                      0.00000000     0.00000000
##   Media                                      0.00000000     0.00000000
##   Natural sciences e.g. chemistry or biology 0.00000000     0.00000000
##   Pharmacy                                   0.00000000     0.00000000
##   Physics & Philosophy                       0.00000000     0.02941176
##   Statistics                                 0.00000000     0.00000000
##
##                                                       DSA
##   Arts BA, CIM level 6 diploma marketing     0.00000000
##   Biomedical Science                         0.02941176
##   Business studies                           0.08823529
##   Computer science                           0.17647059
##   Economics                                  0.02941176
##   Engineering                                0.05882353
##   Humanities e.g., geography or history      0.00000000
##   ICT and MM                                 0.02941176
##   Life Sciences BSc                          0.02941176
##   Maths                                      0.05882353
##   Media                                      0.02941176
##   Natural sciences e.g. chemistry or biology 0.08823529
##   Pharmacy                                   0.02941176
##   Physics & Philosophy                       0.00000000
##   Statistics                                 0.05882353
```

Extension Exercise 1.9: Do AI and DSA students have different perceived levels of statistical and programming understanding?

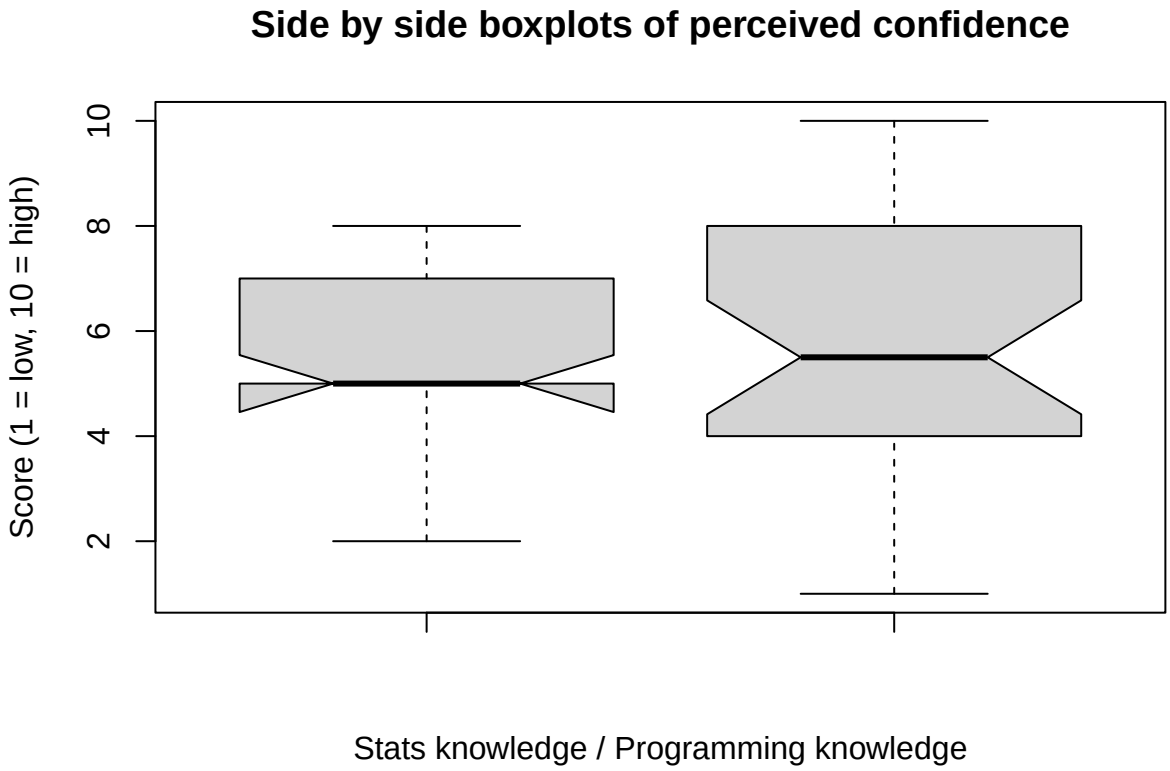
To answer this question having turned `MSc` into a **factor** is now useful. In R a factor has a special meaning, i.e., a character variable which takes on a limited number of different values (these are often referred to as categorical variables) e.g., StudyMode and MSc, whilst Concern would be a problematic choice because the text content is essentially unbounded. To make this change of type we use the `as.factor()` function.

A common way to visualise the distribution of values for a variable is a **boxplot**. Boxplots can be used to compare distributions graphically based on five pieces of information, the minimum, Q1, Q2, Q3 and the maximum (they were proposed by the statistician John Tukey, see wikipedia).

In R we can use the boxplot() function to produce boxplots.

```
# We can use boxplots to show similar(ish) information to histograms
# for example ...
boxplot(surveyDF$StatsKnowl, surveyDF$ProgKnowl,
        xlab = "Stats knowledge / Programming knowledge",
        ylab = "Score (1 = low, 10 = high)",
        notch = TRUE,  # This shows the 95% CI for the medians
        main = "Side by side boxplots of perceived confidence"
        )
```

```
## Warning in (function (z, notch = FALSE, width = NULL, varwidth = FALSE, : some
## notches went outside hinges ('box'): maybe set notch=FALSE
```

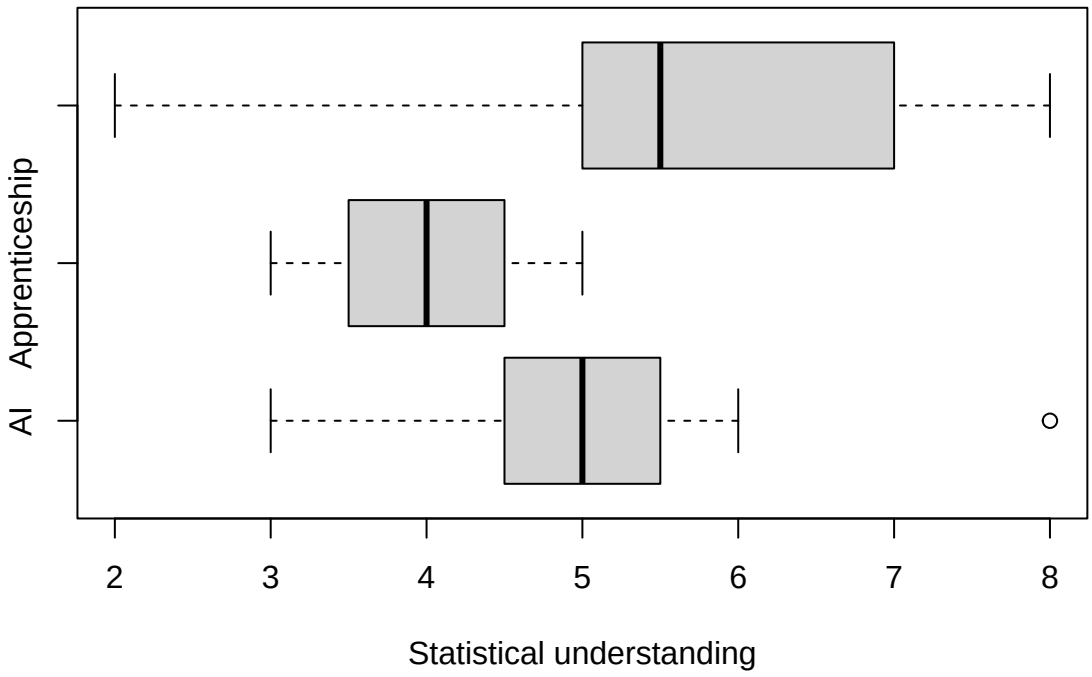


Where we want to separate and compare the data by category e.g., MSc we can use a factor. We tell R how to do this by using a tilde:

`<variable_name> ~ <factor_name>`

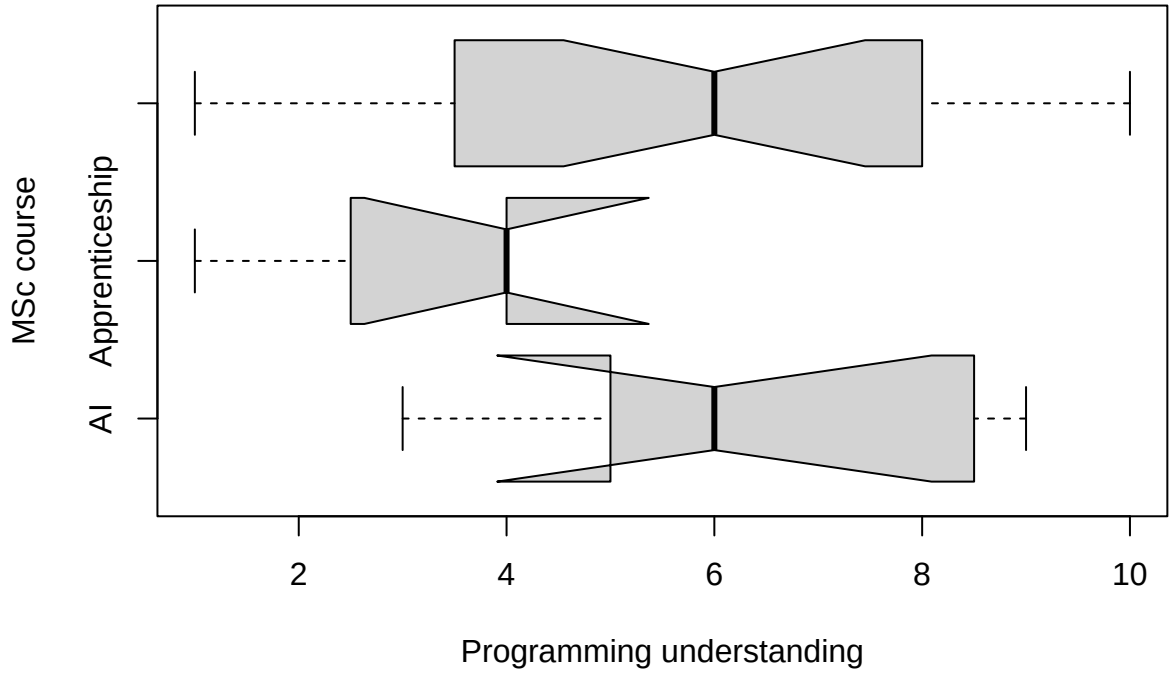
for instance, `surveyDF$StatsKnowl ~ surveyDF$MSc` which then forms the argument we pass to the `boxplot()` function in parentheses (see below).

```
# Compare the different distributions of values using a boxplot for each value of the factor MSc.
# Since there are 2 courses (AI and DSA) this produces 2 boxplots of StatsKnowl and we can see
# if there are any differences
boxplot(surveyDF$StatsKnowl ~ surveyDF$MSc,
        horizontal = TRUE,
        xlab = "Statistical understanding",
        ylab = "")
```




```
# Side by side boxplots of ProgKnowl separated by the factor MSc.
boxplot(surveyDF$ProgKnowl ~ surveyDF$MSc,
        notch = TRUE,             # Shows the 95% confidence intervals
        horizontal = TRUE,        # Rotate plots
        xlab = "Programming understanding",
        ylab = "MSc course")
```

```
## Warning in (function (z, notch = FALSE, width = NULL, varwidth = FALSE, : some
## notches went outside hinges ('box'): maybe set notch=FALSE
```



**Extension exercise 1.9:* To learn more about boxplots see Chapter 4 of the Modern Data book for an introductory

discussion or for a little more depth: Williamson, D., et al. (1989). The Box Plot: A Simple Visual Method to Interpret Data. Annals of Internal Medicine, 110(11), pp916-921.

Exercise 1.10: Now we've generated the boxplots we need to consider whether there are any differences. What do you think? How large are the differences? Do they matter? Sometimes statisticians test whether the differences are important by means of Inferential Statistics and you will learn about these in CS5701 (Quantitative Data Analysis).

1.8 Text analysis

Some of the variables are free format text which can also be analysed in R. This is a little more complex so at this juncture we will simply generate some word clouds.

```
# To extract all the words into a single character string you need the
# paste() function with the collapse option.
words <- paste(surveyDF$Excite, collapse = " ")

# Display the words
words
```

```
## [1] "I am keen to know more about managing and retrieving data and explore industry best practices. C
```

```
# Save the words in a text file as input to a word cloud generator
# You can change the file name
# NB This will overwrite the previous contents (if any)
fileName <- file("MyWords.txt")
writeLines(words, fileName)
close(fileName)
```

You can then copy and paste into a word cloud generator. At a later stage we will do this programmatically but to keep things simple for now you can use my word cloud generator which is written using R and shiny.

Exercise 1.11: What are the dominant themes for (i) `Excite` and (ii) `Concern`? Does the word cloud also find unimportant words? What about similar words but with different endings e.g., singular and plural?

2. Lab: Some More R practice

2.1 Recap Vectors

```
# This R code generates random height data using the function rnorm()
# and assigns it to the numeric vector height
set.seed(42)     # Set the random number generator seed
                 # so your results are repeatable.

# The results are rounded for readability
height <- round(rnorm(n = 20, mean = 1.65, sd = 0.15), 3)

# Generate a dummy vector weight so rmd can be knitted to html/pdf
# You will need to update in Exercise 2.2
weight <- seq(1,length.out=20)
```

Exercise 2.1: What is the value of 18th observation of `height`?

Exercise 2.2: Generate a new vector `weight` that is the same length, contains random values with a mean of 65 and a sd=10. Then compute a new vector called `BMI`. HINT

$$BMI = \frac{weight}{height^2}$$

. SECOND HINT you can apply operations to vectors e.g.,

```
# Example vector operation
height2 <- height * 2
```

2.2 More data frames

A very useful feature of R is the ability to combine columns (vectors) to make data frames. This is accomplished with the `cbind()` function.

```
# Make a new data frame from the vectors height and weight.
# NB They must be the same length!
newDF <- as.data.frame(cbind(height,weight))
```

Exercise 2.3: Add `bmi` as a third column to the dataframe `newDF`.

There is also an analogous row binding function `rbind()` which can be used to add extra rows. We make a new row using the combine function `c()`.

```
# Create a 21st row ready to add to newDF
newRow <- c(10, 20, 30)
```

Exercise 2.4: Now add `newRow` to the end of dataframe `newDF`.

2.3 Using files in R

Very often, the data we need to analyse is stored in external files. So the ability of R to access files is very important. Fortunately R provides a number of very convenient ways to import data. One of the simplest is from a comma-separated variable (CSV) file. This is also very flexible since a very common way that data is stored is in Excel spreadsheets. Users can save their spreadsheets in comma-separated values files (CSV) and then use R's built in functionality to read and manipulate the data.

```
# R code to read a remote file (on GitHub) into a data frame pubsDataFrame
# Because the path url is quite long I've first copied it to a character string fname
# to improve readability

fname <- "https://raw.githubusercontent.com/mjshepperd/CS5702-Data/master/pubs.csv"
pubsDataFrame <- read.csv(fname, header = TRUE, stringsAsFactors = FALSE, fileEncoding = 'UTF-8-BOM')

head(pubsDataFrame)    # This function defaults to showing the first 6 rows
```

```
##              pubName  open        town weeklySales foodSales
## 1 The Dead Albatross  TRUE    Uxbridge        2735      1209
## 2   The Island Queen  TRUE   Islington        3644         0
## 3        Johnnys Bar FALSE Vladivostok           0         0
```

```
## 4           Red Lion  TRUE    Habrough        3263        NA
## 5         The Crown FALSE    Haccombe           0         0
## 6         Royal Oak FALSE      Haceby           0         0
```

In the above R code note that we use the function `read.csv()`; it has several arguments. The first argument is the path name to the csv file we wish to import. In this example we have nested another function `file.choose()` which will prompt the user to select the relevant file. Sometimes this can be very flexible. Next, typically the first row of a spreadsheet contains column descriptions so this can form a good basis for naming each column in R. Next, we have the argument `stringsAsFactors = FALSE`. This is something of a technicality but the default behaviour is to convert string character variables into factors (i.e., categories such as female and male) which isn't generally the desired behaviour. Finally, we specify the encoding to avoid Mac/PC/excel differences.

Exercise 2.5: Read in a local csv file into a new data frame. If you don't have any suitable file you can create a file using Excel and populate it with some dummy data. HINT There are two useful functions `getwd()` and `setwd()` to manage your working directory or alternatively you can set it via RStudio and the Session > Set Working Directory menu option.

So far we have reviewed some methods for importing data into R from external files, but there are occasions when we'll want to go the other way, i.e., exporting data from R so that the data can be archived or used by external applications.

Analogous to the `read.csv()` function is the `write.csv()` which outputs an R object to a delimited text file.

Exercise 2.6: Write a data frame to a local csv file. Check you can view it with either a text editor or Excel. If you need help using this function, type the function name into the Help Tab of the bottom right pane in RStudio.

Extension Exercise 2.7: Read the database section of the Modern Data book to get an overview of manipulating more complex files e.g., databases in R.

Selected Answers

Exercise 1.7: You can you check that your code has changed the variable type in several ways you could accomplish this. Simplest might be to re-run the data frame structure function `str(surveyDF)`. Another possibility would be the class() function i.e., `class(surveyDF$MSc)`.

Exercise 2.1: To find the value of 18th observation try `height[18]`.

Exercise 2.2: `weight <- round(rnorm(n = 20, mean = 65, sd = 10), 3)`.

Exercise 2.3:

```
bmi <- weight / height * height  # create a new vector bmi
newDF <- cbind(newDF,bmi)        # add bmi as a 3rd column
```

NB When we `cbind()` to `newDF`, since it's already a data frame, we don't need to coerce the data type using `as.data.frame()`. Also be careful to recreate the vectors and data frame from scratch, if you want to re-run the code because otherwise you will keep appending colmns and rows.

Exercise 2.6: To write a data frame to a local csv file you need `write.csv(surveyDF,"TestFile1.csv")`.