**Assessment 1**

**Health Data Science (7HMNT032W.1)**

**Donald Philp | 21075797**

**2024**

***Q.1) When the given code snippet(s) are executed in R, will the given variable object(s) store the assigned values? If not, provide the amended code with the new output and explain your changes. [3\*4 = 12 pts.]***

1. ***var.a <- sum(c(1, 2, 3) + c(4, 5, “6”))***
2. ***var\_raq = chartoRaw(“Welcome”)***
3. ***.1a <- “I am learning R programming.”***
4. ***Var-z <- matrix(c(1:4, c(1, 2)), nrow = 2, byrow = TRUE)***

***NOTE: Include comments in your program. You should not use any external R packages (libraries).***

**Solution for Question 1 Part A**

**Approach:** The problem was assessed by quickly reviewing the question to identify any issues without running the code, as it was straightforward.

**Method(s):** A single variable, var.a, was defined using the sum function with two numeric vectors, each containing three elements. The issue arose because vectors are homogenous and cannot mix data types. The sum function cannot combine a string and a numeric value.

A screenshot of a computer

Description automatically generated

**Result with Key findings:** Changing the string "6" to a numeric value by removing the quotes resolved the issue, allowing the code to run as expected.

**R code:**

# Non-numeric argument to binary operator

var.a **=** sum**(**c**(**1,2,3**)** **+** c**(**4,5,”6”**))**

# Change the “6” to 6 as to ensure homogeneity

var.a **=** sum**(**c**(**1,2,3**)** **+** c**(**4,5,6**))**

# Confirm that the change worked

var.a

**Solution for Question 1 Part B**

**Approach:** This is similar to the first question in Part A, but since we could not spot the issue initially, copying the code into RStudio revealed the error.

**Method(s):** Running the code revealed an error stating that the function could not be found. A quick search in RStudio revealed the issue was a character-case error—the function was misspelt.

**A screenshot of a computer

Description automatically generated**

**Result with Key findings:** The key finding was that R is case-sensitive, so correct capitalisation is crucial. The function charToRaw required a capital "T" to execute correctly.

**R code:**

# Could not find function chartoRaw(“Welcome”)

var\_raq **=** chartoRaw**(**“Welcome”**)**

# CharToRaw converts a character string to raw bytes. Watch the case

var\_raq **=** charToRaw**(**“Welcome”**)**

# Confirm that the change worked

var\_raq

**Solution for Question 1 Part C**

**Approach:** Similar to the first two questions, we looked for errors before running the code and immediately noticed an issue discussed in Week 2 regarding variables starting with numbers.

**A screenshot of a computer program

Description automatically generatedMethod(s):** Running the code confirmed that while variables can start with full stops, they cannot start with a number or a full stop followed by a number.

**Result with Key findings:** The fix was simple: remove the number after the full stop or both the full stop and number. Variables starting with a full stop are valid but may confuse later. The key finding is that variables cannot start with a number.

**R code:**

# Unexpected symbol

.1a **=** “I am learning R programming.”

# Remove the number 1

.a **=** “I am learning R programming.”

# Confirm that the change worked

.a

**Solution for Question 1 Part D**

**Approach:** Unsure about the issue, we added the code to RStudio and ran it, which revealed the error: "Var not found."

**Method(s):** We noticed the dash (-) was interpreted as a subtraction operator, making the line read as "variable minus variable equals," which is invalid in R. Variables must first be defined before assigning an equation.

**A screenshot of a computer program

Description automatically generated**

**Result with Key findings:** The issue was that the code attempted to operate in reverse, starting with a subtraction instead of defining a variable. Additionally, the Var variable was not defined anywhere in the code. To fix this, we removed the minus sign after Var, ensuring the line correctly assigned Varz as a defined variable equal to an operation.

**R code:**

# Var-z is not defined, z is being subtracted by Var. This is incorrect

Var**-**z **=** matrix**(**c**(**1**:**4, c**(**1,2**))**, nrow **=** 2, byrow **=** **TRUE)**

# Remove the (-) minus sign after the first variable

Varz **=** matrix**(**c**(**1**:**4, c**(**1,2**))**, nrow **=** 2, byrow **=** **TRUE)**

# Confirm that the change worked

Varz

***Q.2) Write a program in R (for each sub-part) to explain the usage of following functions in R Programming: [4\*5 = 20 pts]***

1. ***length()***
2. ***lapply()***
3. ***summary()***
4. ***read.csv()***
5. ***rbind()***

***NOTE: Include comments in your program. You should not use any external R packages (libraries).***

**Solution for Question 2 Part A**

**Approach:** In question two, I knew most of the functions from A-E. We decided to apply as much as possible without looking at our notes. In the first question, we achieved this.

**Method(s):** We created a variable called var and assigned five data elements: five different colour strings. We knew that if we applied a length() function to the variable var, I would get the answer: five.

**A screenshot of a computer program

Description automatically generated**

**Result with Key findings:** The key finding is that you can apply the function length() to many objects in R. These include, but are not limited to, vectors, lists, data frames (number of columns), matrices, etc.

**R Code:**

# Define a variable called var in a vector

var **=** c**(**"Blue", "Red", "Green", "Purple", "Orange"**)**

# The length() function displays the elements within a vector.

length**(**var**)**

# Output

**[**1**]** 5

**Solution for Question 2 Part B**

**Approach:** Unsure of the answer, we reviewed our notes on lapply(). I created a list to hold numeric values, as lapply() works only on lists. I then defined a variable, sqrt\_numbers, and used lapply() to loop through the list, applying the square root function to each number and saving the results in a new list. Finally, we ran sqrt\_numbers to confirm that the function worked.

**Method(s):** We used a defined list and applied a function to each element, saving the results in a new list. The lapply() function requires two arguments: a list (or vector) and a function (built-in or user-defined). We used a user-defined function, function(x), where x represents each element in the list, applying the square root to x.

**A screenshot of a computer program

Description automatically generated**

**Result with Key findings:** The key findings are that I was able to apply the square root function to each object within the list and then add it to a new list called sqrt\_number. The output that I got was 1, 2, 3, 4, 5, 6, and 7.

**R Code:**

# Define a variable called numbers in a list

numbers **=** list**(**1, 4, 9, 16, 25, 36, 49**)**

# lapply is function used to apply a specific function to a list.

sqrt\_numbers **=** lapply**(**numbers, **function(**x**)** sqrt**(**x**))**

# Confirm that the change worked

sqrt\_numbers

**Solution for Question 2 Part C**

**Approach:** Applying a summary() function was also more straightforward, as we have used many summary questions for data frames. We decided to apply the summary() function to a data frame. I defined a data frame with df, including three features (x, y, z) and six instances (rows). These were random values I added all within a vector.

**Method(s):** The summary function provides statistical details for each feature. It includes the following statistics: minimum value, 1st Quartile value, Median, Mean, 3rd Quartile Value, and Max value.

**A screenshot of a computer

Description automatically generated**

**Result with Key findings:** The summary function give you the statistical details of each feature. It includes the following statistics: minimum value, 1st Quartile value, Median, Mean, 3rd Quartile Value and the Max value.

**R Code:**

# Define a data frame called df with 3 features and 6 instances

df **=** data.frame**(**

x **=** c**(**1, 2, 3, 4, 5, 6**)**,

y **=** c**(**21, 24, 7, 2, 19, 32**)**,

z **=** c**(**231, 112, 43, 432, 89, 23**)**

**)**

# The summary function give us general statistics on each variable defined within the data frame (x, y, z).

summary**(**df**)**

# Output

**x y z**

**Min.** :1.00 **Min.** : 2.00 **Min.** : 23.0

**1st Qu.**:2.25 **1st Qu.**:10.00 **1st Qu.**: 54.5

**Median** :3.50  **Median** :20.00 **Median** :100.5

**Mean**  :3.50 **Mean**  :17.50 **Mean** :155.0

**3rd Qu.**:4.75 **3rd Qu.**:23.25 **3rd Qu.**:201.2

**Max.** :6.00 **Max.** :32.00 **Max.** :432.0

**Solution for Question 2 Part D**

**Approach:** Applying the read.csv() function was possibly the easiest function to remember, as it is very often used at the beginning of a coding session to load data from a CSV file.

**Method(s):** The CSV file is a comma-separated file that splits sections of your data into rows and commas. This function has one mandatory parameter: the folder location and the file name of the CSV file. Based on the working directory, this location is usually relative to the location of the defined working directory. We used optional parameters, such as keeping the header, header = TRUE, and specifying the separator as ",".

**Result with Key findings:** As the data variable had no hard-coded location, as was specified by the brief, the output had no value but would be in a two-dimensional table form based on the data within the CSV file.

**R Code:**

# This functions opens a specific csv file in a specified location, you have a few options, in this case we kept the column heading and specified the separator as a ","

data **=** read.csv**(**'my/project/location/file.txt', header **=** **TRUE**, sep **=** ","**)**

**Solution for Question 2 Part E**

**Approach:** The rbind() function is another function we were unfamiliar with and had to look it up in our notes. I knew the best way to explain the rbind() function was to add rows to an existing data frame.

**Method(s):** We created two data frames to help me illustrate the rbind() function. The first data frame was defined with the name var1. This data frame had two features (x, y) and three instances (rows). The values were chosen at random. For the second data frame, I had to ensure that the features (x, y) had to be the same name so that easy rbinding could take place. I also created 4 instances (rows) for this data frame. I then utilised the rbind() function to add the second data frame's rows, var2, just below the var1's rows.

**Result with Key findings:** The findings and the rbind() functions are well-defined and can be seen below. We have created a third variable called var\_comb, which contains both the var1 and var2 data frames as one data frame. The final data frame has two features (x, y) and eight instances (rows).

**R Code:**

# Define a variable called var1 with two features (x, y) and 4 instances (rows)

var1 **=** data.frame**(**

x **=** c**(**1, 2, 3, 4**)**,

y **=** c**(**43, 11, 9, 17**)**

**)**

# Define another variable called var2 with two features (x, y) and 4 additional instances (rows)

var2 **=** data.frame**(**

x **=** c**(**46, 17, 2, 9**)**,

y **=** c**(**16, 19, 3, 11**)**

**)**

# rbind() function binds rows together "r" bind.

var\_comb **=** rbind**(**var1, var2**)**

var\_comb

**x y**

1 1 43

2 2 11

3 3 9

4 4 17

5 46 16

6 17 19

7 2 3

8 9 11

***Q.3) In the given code, a user enters the following values: var1 = 250, var2 = 35, var3 = 180, expecting the return value to be 286. However, it returns a value of 501. Modify this code to return the value as expected by the user. Explain your changes. [8 pts.]***

***var1 <- as.integer(readline("Enter a value:"))***

***var2 <- as.integer(readline("Enter a value:"))***

***var3 <- as.integer(readline("Enter a value:"))***

***func.1 <- function(var1) {***

***func.2 <- function(var2) {***

***var2 + var3***

***}***

***var3 = 1***

***var1 + func.2(var1)***

***}***

***func.1(var1)***

***NOTE: Include comments in your program. You should not use any external R packages (libraries).***

**Solution for Question 3**

**Approach:** To read the code better we decided to override the readline() functions. We defined three variables initially and adjusted the code. We reviewed functions 1 (func.1) and 2 (func.2) thoroughly. Func.1 contains three statements: Statement 1 results in 210, Statement 2 updates var3, and Statement 3 inputs var1 into var2 and adds var3 (equal to 1), leading to a total of 501.

**Method(s):** The only way to solve this complex code was to possibly swap some variables around to really see what was going on or how the code changed. The first value I swapped around was Statement 1's var2 value with var1. The first code change worked. I then evaluated the change in the next block of code called # Correct, where I explain how the changes of the var1 value with var2 change the outcome of the value from 501 to 286.

**Result with Key findings:** We changed the code to change the value from 501 to 286 in the finding. We established that one can change the code by changing the var variable. If one looks at Statement 3 of the original code, to change from number 501 to 286, one could also change the following line from: var1 **+** func.2**(**var1**)** to var1 **+** func.2**(**var2**).**This would have the same effect as my solution of changing Statement 1 from line:func.2 **=** **function(**var2**)** **{**var2 **+** var3**}** tofunc.2 **=** **function(**var1**)** **{**var2 **+** var3**}**

***A screenshot of a computer program

Description automatically generated***

**R Code:**

**# Re-frame Q3 to follow along easier**

**# Incorrect**

var1 **=** 250 # Re-frame

var2 **=** 35 # Re-frame

var3 **=** 180 # Re-frame

# Define the function

func.1 **=** **function(**var1**)** **{**

# Statement 1 = 35 + 180 = 210

func.2 **=** **function(**var2**)** **{**var2 **+** var3**}**

# Statement 2

var3 **=** 1

# This line determines the value of func.1(var1)

# Statement 3 = 250 + (250 + 1) = 501

var1 **+** func.2**(**var1**)**

**}**

func.1**(**var1**)**

# Output

**[**1**]** 501

**# However we expect func.1(var1) = 286**

**# Correct**

var1 **=** 250 # Re-frame

var2 **=** 35 # Re-frame

var3 **=** 180 # Re-frame

# Define the function

func.1 **=** **function(**var1**)** **{**

# Change var2 to var1

# Statement 1 = 35 + 180 = 210

func.2 **=** **function(**var1**)** **{**var2 **+** var3**}**

# Notice that var3 has been overridden by var3 = 1

# Statement 2

var3 **=** 1

# Statement 3 = 250 + (35 + 1) = 286

var1 **+** func.2**(**var1**)**

**}**

# func.1(var1) = 286

func.1**(**var1**)**

# Output

**[**1**]** 286

***Q.4) You are part of a Data Science team in a Marine Analytics Organisation. As a consultant, you are helping a client company (works in marine conservation and are presently focusing on Abalones) to analyse an Abalone dataset to gain new insights.***

***[Information: Abalone are marine snails, considered as white gold. They are in high demand in the Asian food markets, with a plate of it costing ~ £400-500. This is a leading cause of its poaching in Africa and various other countries. You may read more about it here and here.]***

***Write a program which helps in the following tasks:***

***[2 + 4 + 4 + 4 + 4 + 2 = 20 pts]***

1. ***Allow the user to import the dataset and add the column (attributes/feature) names.***
2. ***Allow the user to group the data based on gender and give the count of male and female Abalones. Give the data summary of continuous attributes for the two groups.***
3. ***Allow the user to calculate the Age for each Abalone and store these values in a new column.***
4. ***Allow the user to discover whether the average age for male Abalone’s is greater than female Abalone’s or not.***
5. ***Allow the user to discover all the Abalone’s for whom, the sum of Shucked weight, Viscera weight and Shell weight is less than the Whole weight of Abalone.***
6. ***Allow the user to find out all the Abalone’s whose Diameter is greater than 0.500mm and Height is less than 0.200mm.***

***NOTE: Use the Abalone dataset from the UCI repository (https://archive.ics.uci.edu/ml/datasets/Abalone). Include comments in your program. If required, you may use external R packages (libraries).***

**Solution for Question 4 Part A**

**Approach:** We downloaded the Abalone.data file and set the working directory to its location. After checking the file, we determined that the separator was ",". To ensure proper loading, we displayed the head and a summary of the dataset. Since the feature names weren't readable, we defined them in a variable called column\_names, reloaded the data with these names, and printed the head again to verify the column names were applied correctly.

**Method(s):** The most crucial method that we used to answer the question was utilising the following functions correctly. Setwd(), ensured that our data file would be located. Read.table() to ensure the data file is loaded correctly. vector c("column names") function to ensure that we have all the correct label data for the header. col.names = to ensure that we could apply the column names I specified earlier within the vector and apply it to the previously loaded data set of dataset\_csv.

**Result with Key findings:** The key findings were that the Abalone.data file matches the expected number of features given as the header names. The data was clean and clear of any missing values.

A screenshot of a computer

Description automatically generated

**R Code:**

# Set the working directory for your Code

setwd**(**"C:/myFiles/Health Data Science/Assignment"**)**

# Loads the table with including the header and "," separator

dataset\_csv **=** read.table**(**"abalone.data", header **=** **TRUE**, sep **=** ","**)**

# Prints the head of the table

head**(**dataset\_csv**)**

# Prints the first 5 entries

summary**(**dataset\_csv**)**

# Define the column names as specified in the details of the dataset website

column\_names **=** c**(**'Sex', 'Length', 'Diameter', 'Height', 'Whole\_weight', 'Shucked\_weight', 'Viscera\_weight', 'Shell\_weight', 'Rings'**)**

# Add the col.names called column\_names that we defined previously

dataset\_csv **=** read.table**(**"abalone.data", header **=** **TRUE**, col.names **=** column\_names, sep **=** ","**)**

# Confirm that the change worked

head**(**dataset\_csv**)**

**Solution for Question 4 Part B**

**Approach:** We created an empty "character" vector based on the length of the $Sex column in dataset\_csv. Then, we looped through the data to assign "male\_group," "female\_group," or "Infant" based on the values "M," "F," and "I" in the $Sex column. We included a check for any unexpected values, printing a warning if found. After categorizing the data, we created a new data frame, dataset\_csv.new, that combined the original dataset with the new group column. Finally, we used the subset.data.frame command to split the dataset into separate datasets for each group and checked the row counts to find the totals for each Sex group.

**Method(s):** We created an empty vector, specifying its type and length before the loop. Using a for loop from 1 to the end of the $Sex column, we assigned group names based on conditions (if, else if, else). We then used the subset.data.frame() function to create subsets from the original dataset and allocated them to respective groups. Finally, we counted the number of instances in each group using nrow(), which ignores the header.

**Result with Key findings:** I observed the following results from the code: The total number of abalone was 4176, of which we had **1527 males**, **1307 females** and **1342 infants**

A white background with blue text

Description automatically generated

**R Code:**

# Define a variable character vector with a length of the column 'Sex' in the dataset\_csv table

group **=** character(length**(**dataset\_csv**$**Sex**))**

# Create a for loop each value s from, number 1: all the way to the end of the data\_set\_csv table, which is the length of the table.

**for(**s **in** 1**:**length**(**dataset\_csv**$**Sex**))** **{**

# if dataset\_csv Sex column equals 'M' indicating male

**if(**dataset\_csv**$**Sex**[**s**]** **==** "M"**)** **{**

# if true, assigns "male\_group" tot the s-th position in group vector

group**[**s**]** **=** "male\_group"

# otherwise if dataset\_csv Sex column equals 'F' indicating female

**}else** **if** **(**dataset\_csv**$**Sex**[**s**]** **==** "F"**){**

# if true, assigns "female\_group" tot the s-th position in group vector

group**[**s**]** **=** "female\_group"

# if dataset\_csv Sex column equals 'I' indicating Infant

**}else** **if** **(**dataset\_csv**$**Sex**[**s**]** **==** "I"**){**

# if true, assigns "infant\_group" tot the s-th position in group vector

group**[**s**]** **=** "infant\_group"

# if there is any other value than 'M', 'F' or 'I', print out the line of this error

**}else** **{** print**(**paste**(**'There is something other than M, F or I on line:', s**))** **}**

**}**

# Create a new variable and bind the columns of dataset\_csv to each vector in the group variable

dataset\_csv.new **<-** cbind**(**dataset\_csv, group**)**

# Create a subset data frame for "female\_group" that was specified in the Vector

female\_group **=** subset.data.frame**(**dataset\_csv.new, group **==** "female\_group"**)**

# Create a subset data frame for "Male\_group" that was specified in the Vector

male\_group **=** subset.data.frame**(**dataset\_csv.new, group **==** "male\_group"**)**

# Create a subset data frame for "Infant\_group" that was specified in the Vector

infant\_group **=** subset.data.frame**(**dataset\_csv.new, group **==** "infant\_group"**)**

# We now have 3 data frames each for the different "Sex" categories.

# Display the amount of rows in the data frame infant\_group

nrow**(**infant\_group**)**

# Display the amount of rows in the data frame male\_group

nrow**(**male\_group**)**

# Display the amount of rows in the data frame female\_group

nrow**(**female\_group**)**

# 1527 males, 1307 females and 1342 infants

**Solution for Question 4 Part C**

**Approach:** We used a simple approach similar to the question before. We ran a for loop for the value a from the index number 1: all the way to the nrow of the dataset\_csv. We realise this was very similar to our previous code, but it didn't work to specify a specific column. We then specified a new column named $Age, and for each instance (row), I added the total adjacent $Rings + 1.5. According to the website, the age can be calculated in the following way, based on the description of the Rings feature:

|  |  |  |  |
| --- | --- | --- | --- |
| **Rings** | Target | Integer | **+1.5 gives the age in years** |

**Method(s):** The most important method was to add a new $Age column to the existing dataset\_csv data frame and calculate this appropriately. We had to run a for loop and ensure that for each instance, we took the $Ring value and added 1.5 years. This allowed us to calculate the age of each abalone instance and add it to the new $Age column.

**Result with Key findings:** The key finding was that we needed to find how the ages were calculated for the Abalone. The rest of the code was simple enough as we could use the previous code I knew to do a similar loop.

A screenshot of a graph

Description automatically generated

**R Code:**

# Run a for loop a from 1 to the number of rows in the dataset\_csv

**for** **(**a **in** 1**:**nrow**(**dataset\_csv**))** **{**

# Running though each line and changing the value of a as the new value of the Age column by Rings + 1.5

dataset\_csv**$**Age**[**a**]** **=** **(**dataset\_csv**$**Rings**[**a**]** **+** 1.5**)**

**}**

# This prints out the changed table with included Ages

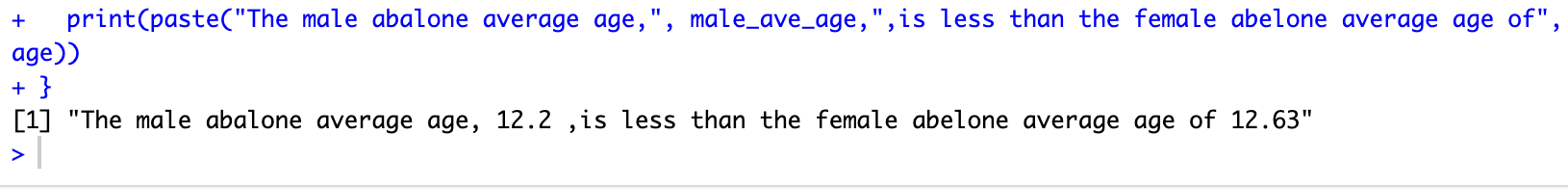
View**(**dataset\_csv**)**

**Solution for Question 4 Part D**

**Approach:** Similar to the previous question, we used the same for loop to add the ages for the male\_group data frame and the female\_group data frame, which we created earlier in question 4 Part B. Once again, we created two new variables to contain the mean value of each male and female group and then rounded it off to two decimal places. we then create an if statement that prints out the difference between the two average groups and which is older.

**Method(s):** The methods we used in calculating the average age were similar to the previous question in that we utilised the built-in function of mean() and applied it to both the male\_group$Age column and the female\_group$Age column, specifying variables for each of the values. I used the same loop technique as before and utilised the control structure and if command to print out the results of the differences in average age between the male\_group and female\_group.

**Result with Key findings:** The key findings were that the **average male abalone age was 12.2 years** which was **LESS** than the **average age of the female abalone group of 12.63 years.**

****

**R Code:**

# Create a for loop to place the $Age column in my existing male\_group data frame.

**for** **(**a **in** 1**:**nrow**(**male\_group**))** **{**

male\_group**$**Age**[**a**]** **=** **(**male\_group**$**Rings**[**a**]** **+** 1.5**)**

**}**

# Create a for loop to place the $Age column in my existing female\_group data frame.

**for** **(**a **in** 1**:**nrow**(**female\_group**))** **{**

female\_group**$**Age**[**a**]** **=** **(**female\_group**$**Rings**[**a**]** **+** 1.5**)**

**}**

# Create a variable and run a mean value for the $Age column in male\_group rounding to 2 decimals

male\_ave\_age **=** round**(**mean**(**male\_group**$**Age**)**, 2**)**

# Create a variable and run a mean value for the $Age column in female\_group rounding to 2 decimals

female\_ave\_age **=** round**(**mean**(**female\_group**$**Age**)**, 2**)**

# Create an if statement: if male\_ave\_age is older or equal to female\_ave\_age

**if** **(**male\_ave\_age **>=** female\_ave\_age**)** **{**

# Print the difference and state that male\_ave\_age is older than female\_ave\_age

print**(**paste**(**"The male abalone average age,", male\_ave\_age, ",is older than the female abelone average age of", female\_ave\_age**))**

# Otherwise, Print the difference and state that male\_ave\_age is younger than female\_ave\_age

**}else{**

print**(**paste**(**"The male abalone average age,", male\_ave\_age,",is younger than the female abelone average age of", female\_ave\_age**))**

**}**

**Solution for Question 4 Part E**

**Approach:** We shortened the column names for each of the features. In this way the code is neatened to apply the subset.data.frame() function. Once these variable names were shortened, I created the following condition based on the question that was asked:

***Allow the user to discover all the Abalone’s for whom, the sum of Shucked weight, Viscera weight and Shell weight is less than the Whole weight of Abalone.***

**Condition:** whole\_w **>** shucked\_w **+** viscera\_w **+** shell\_w

We created a subset.data.frame() and added it to a new data frame called weight\_group. We then applied the View() function to see the condition's outcome. However, at first glance, the View() function did not guarantee that the condition was applied correctly. We then compared the dimensions of the original dataset\_csv data frame and the new weight\_group data frame.

**Method(s):** Our methods were similar to those previously used. We used the subset.data.frame() function to create a separate data frame with the new conditional parameters within it.

**Result with Key findings:** Because it was difficult to see if my code worked, I needed to ensure that it did work by running the dim() function of each of the data frames. The total instances for the dataset\_csv was **4176,** and the total instances for the new weight\_group was **4014**; **A difference of only 162.**

A screenshot of a computer code

Description automatically generated

**R Code:**

# Shorten the column names for the dataset\_csv data frame

whole\_w **=** dataset\_csv**$**Whole\_weight

shucked\_w **=** dataset\_csv**$**Shucked\_weight

viscera\_w **=** dataset\_csv**$**Viscera\_weight

shell\_w **=** dataset\_csv**$**Shell\_weight

# Create a new weight\_group variable and add the parameters as specified by the question. Only include the specified parameters in the new weight\_group

weight\_group **=** subset.data.frame**(**dataset\_csv, whole\_w **>** shell\_w **+** viscera\_w **+** shucked\_w**)**

# Confirm that the parameters in the new weight\_group worked

View**(**weight\_group**)**

# The View(weight\_group) table was not clear. Run the dimensions of the difference between the original dataset\_csv and the new weight\_group.

# dataset\_csv dimensions

dim**(**dataset\_csv**)**

# Output

**[**1**]** 4176 9

# weight\_group dimensions

dim**(**weight\_group**)**

# Output

**[**1**]** 4014 9

**Solution for Question 4 Part F**

**Approach:** I created the following condition based on the question that was asked:

***Allow the user to find out all the Abalone’s whose Diameter is greater than 0.500mm and Height is less than 0.200mm.***

**Condition:** dataset\_csv**$**Diameter **>** 0.5 **&** dataset\_csv**$**Height **<** 0.2

We created a subset.data.frame() and added it into a new data frame called size\_filter, which is very similar to the previous question. We then applied the View() function to see the outcome of the condition. However, at first glance, the View() function did not give me confidence that the condition was applied correctly. We then compared the dimensions of the original dataset\_csv data frame and the new size\_filter data frame.

**Method(s):** The methods used was similar to the methods I’ve used previously and almost identical to the previous question, Part E. We used the subset.data.frame() function to create a separate data frame with the new conditional parameters within the data frame.

**Result with Key findings:** Because it was difficult to see if our code worked, we needed to ensure that it did work by running the dim() function of each of the data frames. The total instances for the dataset\_csv was **4176,** and the total instances for the new size\_filter was **504**; **A difference of 3672.**

***A screenshot of a computer code

Description automatically generated***

**R Code:**

# Create a new size\_filter variable and add the parameters as the question specifies. Only include the specified parameters in the new size\_filter subset.data.frame

size\_filter **=** subset.data.frame**(**dataset\_csv, dataset\_csv**$**Diameter **>** 0.5 **&** dataset\_csv**$**Height **<** 0.2**)**

# Confirm that the parameters in the new size\_filter worked

View**(**size\_filter**)**

# The View(size\_filter) table was not clear. Run the dimensions of the difference between the original dataset\_csv and the new size\_filter.

# dataset\_csv dimensions

dim**(**dataset\_csv**)**

# Output

**[**1**]** 4176 9

# size\_filter dimensions

dim**(**size\_filter**)**

# Output

**[**1**]** 504 9

***Q.5) Write a function-based program in R, having an outer function with three arguments x, y and z. Create an inner function within the outer function, which has an argument with a constant value (3.56). The inner function should calculate product of arguments x and y, and sum the result with this constant value. The outer function should check whether the result of inner function is a prime number. If “TRUE” should multiply the result with argument z and return this value, else should return the result as derived from the inner function. [20 pts.]***

***NOTE: Include comments in your program. You should not use any external R packages (libraries).***

**Solution for Question 5**

**Approach:** Initially, it was easy to follow the function's outer structure. The Outer function and the Inner function of the question seem manageable. However, including a deeper function to validate prime numbers is currently too challenging. I did a search on understanding the principles of prime numbers. I know a prime number is a number that can only be divided by itself and one. The process for the code would work as follows:  
**Statement 1:** is the number smaller than or equal to 1?

Yes – send it to the output of the inner function

No – send it to the next statement

**Statement 2:** is the number odd or even?

If even – send it to the output of the inner function

If odd – send it to the next statement

**Statement 3:** divide the number by all integers from 2 to sqrt(n)

If divisible by any integer send it to the output of the inner function

If not divisible number is therefore prime

**Method(s):** Although I knew that I would need only need a total of the variables, x, y, z and could input them into outer\_function and then inner\_function I

**Result with Key findings:** No finding just learned the understanding of prime numbers

**R Code:**

# NOT COMPLETE

## Was not able to calculate this :(

outer\_function **=** **function(**x, y, z**)** **{**

inner\_function **=** **function(**x, y**)** **{**

return**((**x**\***y**)** **+** 3.56**)**

**}**

**if(**n **<=** 1**)** **{**

return**(FALSE)** # return the result to the outer function

**for(**i **in** 2**:(**sqrt**(**n**))**

result %% 2 **!=** 0# take the incoming number and if its greater than 1 then add a square root the number to check the total number of divisions

**}** #

**&** is\_prime **==** 2

**&** is\_prime %% 2 **!=** 0**)** **==** **TRUE)** **{**

return**(**result **\*** z**)**

is\_prime **=** inner\_function**(**x, y**)**

**}else{** return**(**result**)**

**}**

**}**

# Test the outer function

outer\_function**(**1,1,1**)**

***Q.6) Being a part of a Data Science team in a Health Analytics Organisation, you have been assigned the following tasks:***

1. ***Develop a R program which read two datasets (Q6-File1.csv and Q6-File2.csv): the first file includes clinical data, while the second file contains protein expression data. For each dataset, the program should do the following: [2\*3 = 6 pts.] provide the number of dimensions, summarising the number of patients and the names of variables for any chosen dataset.*** 
   1. ***generate box plots for user-specified continuous variable(s).***

***Develop a R program, which matches the patient-ids from the first dataset with the reference-ids from the second dataset (reference id in the second file is combination of patient id with sample number). For each patient, calculate the arithmetic mean of expression values for each protein.***

***whose mean protein expression values are above this threshold. [14 pts.]***

***NOTE: Include comments in your program. You should not use any external R packages (libraries).***

**Solution for Question 6a (i)**

**Approach:** Load the datasets from the csv file with read.csv(). Display the nrow total with dim() function. Display the column names with colnames() function.

**Method(s):** It was a simple method to apply to two datasets provided.

**Result with Key findings:** The two datasets were different in size: File1 had a dimension of **142 instances and 10 features,** andFile2 had a dimension of **426 instances and 14 features.**

A computer code on a white background

Description automatically generated

**R Code:**

# Dataset Q6-File1.csv, Load csv file, header = true and separator = ","

Q6\_File1 **=** read.csv**(**"Q6-File1.csv", header **=** **TRUE**, sep **=** ","**)**

# Display dimension of data frame Q6\_File1

dim**(**Q6\_File1**)**

# Output:

**[**1**]** 142 10

# Display Feature Names

colnames**(**Q6\_File1**)**

# Dataset Q6-File2.csv Load csv file, header = true and separator = ","

Q6\_File2 **=** read.csv**(**"Q6-File2.csv", header **=** **TRUE**, sep **=** ","**)**

# Display dimension of data frame Q6\_File2

dim**(**Q6\_File2**)**

# Output:

**[**1**]** 426 14

# Display Feature Names

colnames**(**Q6\_File2**)**

**Solution for Question 6a (ii)**

**Approach:** We knew we had to use sapply() to filter numeric data from a data frame to include only numeric features. We needed to know how to use a boxplot. We utilised our class notes for this.

**Method(s):** We first filtered out continuous numeric data from thee first Q6\_File1 data frame and place it in a new data frame that applied Boolean values to each feature. We then did the same for the Q6\_File2 data frame. With these two newly created Boolean values wee could apply the boxplot to this to ensure that it displayed correctly. We applied the main = for each boxplot name and gave the one boxplot a pink color and the other skyblue.

**A diagram of a box plot

Description automatically generatedA diagram of a box plot

Description automatically generated**

**Result with Key findings:** We found that 7 out of the 10 features on the Q6\_File1 data frame had numerical values while only 3 out of 14 features had numerical values for Q6\_File2

**R Code:**

# sapply Data Frame all numeric columns

continuous\_Q6\_1 **=** sapply**(**Q6\_File1, is.numeric**)**

# sapply Data Frame all numeric columns

continuous\_Q6\_2 **=** sapply**(**Q6\_File2, is.numeric**)**

# Apply boxplot to Q6\_File1 CSV and use Continuous value by booleans specification

boxplot**(**Q6\_File1**[**,continuous\_Q6\_1**]**,

main **=** "Box Plot Q6 File1", col **=** "skyblue"**)**

# Apply boxplot to Q6\_File2 CSV and use Continuous value by booleans specification

boxplot**(**Q6\_File2**[**,continuous\_Q6\_2**]**,

main **=** "Box Plot Q6 File2", col **=** "pink"**)**

**Solution for Question 6b**

**Approach:** I had to do some research on firstly how to merge data frames and secondly how to break up values within specific columns. One of the challenge on this question was to merge the Patient ID with each other. The challenge was that one of the datasets had the Patient ID number stuck with the reference ID and need to be removed and added on a separate column. Besides this challenge it seemed that the data was not very clean and contained features that were suppose to be numeric but were categorical. We needed to convert many of the features to numeric before working on them. There was also missing data and we needed to ensure we worked around this problem.

**Method(s):** We used a few new methods. Merge() function two merge two data frames with a common feature, substr() to remove some characters of data within a specific column and applying a user-specific threshold to columns. We also needed to calculate the mean of entire rows of instances across more than on feature and calculated the mean across 10 proteins.

**Result with Key findings:** The key findings were to explore new unused features and to filter out data based on user-specific thresholds

A close-up of a computer code

Description automatically generated

**R Code:**

# For loop from index 1:last row of Q6\_File2 substr remove first character and adding to new $PatientID column

**for** **(**f **in** 1**:**nrow**(**Q6\_File2**))** **{**

Q6\_File2**$**PatientID**[**f**]** **=** substr**(**Q6\_File2**$**Reference\_ID**[**f**]**, 1, 1**)**

**}**

# Check new column $PatientID

head**(**Q6\_File2**)**

# Merge File1 with File2 by PatientID

data\_merge **=** merge**(**Q6\_File1, Q6\_File2, by **=** "PatientID"**)**

# Check new data frame data\_merge

head**(**data\_merge**)**

# Convert all Protein columns to Numeric

data\_merge**$**Protein.1 **=** as.numeric**(**data\_merge**$**Protein.1**)**

data\_merge**$**Protein.2 **=** as.numeric**(**data\_merge**$**Protein.2**)**

data\_merge**$**Protein.3 **=** as.numeric**(**data\_merge**$**Protein.3**)**

data\_merge**$**Protein.4 **=** as.numeric**(**data\_merge**$**Protein.4**)**

data\_merge**$**Protein.5 **=** as.numeric**(**data\_merge**$**Protein.5**)**

data\_merge**$**Protein.6 **=** as.numeric**(**data\_merge**$**Protein.6**)**

data\_merge**$**Protein.7 **=** as.numeric**(**data\_merge**$**Protein.7**)**

data\_merge**$**Protein.8 **=** as.numeric**(**data\_merge**$**Protein.8**)**

data\_merge**$**Protein.9 **=** as.numeric**(**data\_merge**$**Protein.9**)**

data\_merge**$**Protein.10 **=** as.numeric**(**data\_merge**$**Protein.10**)**

# Create an empty numeric column called mean\_protein

data\_merge**$**mean\_protein **=** numeric**(**nrow**(**data\_merge**))**

# Loop through each protein adding them and dividing by 10

**for** **(**p **in** 1**:**nrow**(**data\_merge**))** **{**

data\_merge**$**mean\_protein**[**p**]** **=**

**(**data\_merge**$**Protein.1**[**p**]** **+**

data\_merge**$**Protein.2**[**p**]** **+**

data\_merge**$**Protein.3**[**p**]** **+**

data\_merge**$**Protein.4**[**p**]** **+**

data\_merge**$**Protein.5**[**p**]** **+**

data\_merge**$**Protein.6**[**p**]** **+**

data\_merge**$**Protein.7**[**p**]** **+**

data\_merge**$**Protein.8**[**p**]** **+**

data\_merge**$**Protein.9**[**p**]** **+**

data\_merge**$**Protein.10**[**p**])** **/** 10

**}**

# Set threshold to 500

threshold **=** 500

# Create a blank character vector with a length the size of the data\_merge df

threshold\_group **=** character**(**nrow**(**data\_merge**))**

# Loop through the mean\_protein column and add NA to missing

**for** **(**t **in** 1**:**nrow**(**data\_merge**))** **{**

**if** **(**is.na**(**data\_merge**$**mean\_protein**[**t**]))** **{**

threshold\_group**[**t**]** **=** "missing"

# Add mean\_protein threshold to available values

**}else** **if** **(**data\_merge**$**mean\_protein**[**t**]** **<=** threshold**){**

threshold\_group**[**t**]** **=** "within\_threshold"

**}else{**

threshold\_group**[**t**]** **=** "outside\_threshold"

**}**

**}**

# add new column

data\_merge**$**threshold\_group **=** threshold\_group

# Create new data frame which only includes the "within\_threshold" range

thresh\_new **<-** subset**(**data\_merge, threshold\_group **==** "within\_threshold"**)**

# Print a table form listing the current groups

print**(**table**(**data\_merge**$**threshold\_group**))**