Assessment 1: Data Analysis (R Programming)
20 November 2024

DONALD PHILP

Health Data Science

7HMNT032W

UNIVERSITY OF WESTMINSTER

QUESTION 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.

```
a. var.a ← sum(c(1, 2, 3) + c(4, 5, “6”))
b. var_raq ← chartoRaw(“Welcome”)
c. .1a ← “I am learning R programming.”
d. 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).*
[3*4 = 12 pts.]

SOLUTION 1a

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.
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:

```r
# Non-numerical 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 1b

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.
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 if the change worked
var_raq
```

SOLUTION 1c

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.
Method(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 1b

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.
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:

```r
# 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
```

QUESTION 2

Write a program in R (for each sub-part) to explain the usage of following functions in R Programming:

a. length()
b. lapply()
c. summary()
d. read.csv()
e. rbind()

***NOTE:** Include comments in your program. You should not use any external R packages (libraries).*
[4*5 = 20 pts]

SOLUTION 2a

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.
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 2b

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.
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.

Question 2b (Continue)

R Code:

```r
# 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 2c

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.
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:

```r
# 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 2d

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:

```r
# 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 2e

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).

Question 2e (Continue)

R Code:

```r
# 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
```

QUESTION 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.

```
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).*
[8 pts.]

SOLUTION 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}`**

Question 3 (Continue)

R Code:

```r
# 1. 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

# 2. 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)
```

Question 3 (Continue)

R Code:

```
}
# func.1(var1) = 286
func.1(var1)
# Output
[1] 286
```

QUESTION 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:

a. *Allow the user to import the dataset and add the column (attributes/feature) names.*
b. *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.*
c. *Allow the user to calculate the Age for each Abalone and store these values in a new column.*
d. *Allow the user to discover whether the average age for male Abalone's is greater than female Abalone's or not.*
e. *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.*
f. *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).*

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

SOLUTION 4a

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.

R Code:

```r
# 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 4b

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.**

R Code:

```r
# 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”
```

Question 4b (Continue)

R Code:

```r
# 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 4c

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.

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 4d

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

```r
# 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))

```

Question 4d (Continue)

R Code:

```
# 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))

# Output
[1] "The male abalone average age, 12.2 ,is less than the female abelone
average age of 12.63"
```

SOLUTION 4e

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.

Question 4e (Continue)

R Code:

```r
# 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 4f

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.

R Code

```r
# 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
```

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

SOLUTION 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

Question 5 (Continue)

R Code:

```r
# 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)
```

QUESTION 6

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

a. *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:*
   i. *provide the number of dimensions, summarising the number of patients and the names of variables for any chosen dataset.*
   ii. *generate box plots for user-specified continuous variable(s).*

[2*3 = 6 pts.]

b. *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. Thereafter, based on user specified threshold value, highlight patients whose mean protein expression values are above this threshold.*

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

[14 pts.]

SOLUTION 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**, and File2 had a dimension of **426 instances and 14 features**.

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)

```

Question 6a(i) (Continue)

R Code:

```
# 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 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.
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

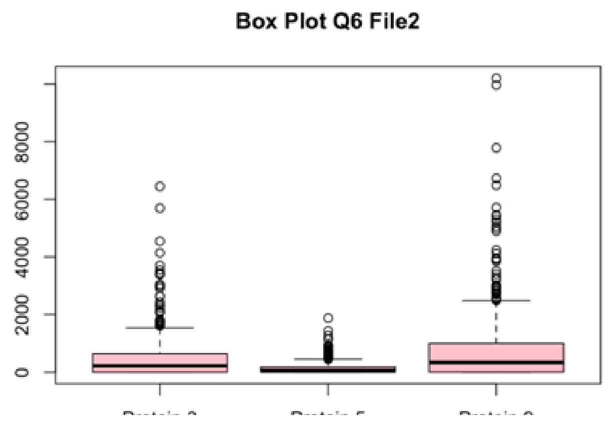
R Code:

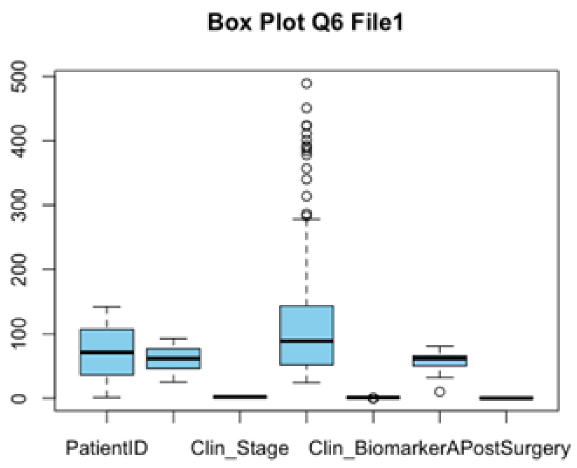
```
# sapply Data Frame all numeric columns
continuous_Q6_1 = sapply(Q6_File1, is.numeric)

# 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”)

```

Question 6a(ii) (Continue)

R Code:

```r
# 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 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

Question 6b (Continue)

R Code:

```r
# 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
}
```

Question 6b (Continue)

R Code:

```
# 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))
```

THANK YOU

DONALD PHILP
TEL +44 777 890 4717
EML W2107579@WESTMINSTER.AC.UK
WEB DONPHI.WORK

+