**Assignment 1 – Introduction to R, Bioconductor and biomaRt (worth 10% of MB6300)**

**Course co-ordinator: Dr Marcus Claesson (**[**m.claesson@ucc.ie**](mailto:m.claesson@ucc.ie)**)**

**Demonstrators: Jill O’Sullivan (**[**114303891@umail.ucc.ie**](mailto:114303891@umail.ucc.ie)**)**

**Student Name (ID): James Henry Hehir (120224791)**

For each question, **please record your answer in a Word document, and also what you typed on the command-line to get this answer. When finished, upload the document (with your name in the title) with your answers to Canvas under Assignments and MB6300 Assignment 1.**

**Deadline: End of Sunday 21st Feb**

If you cannot meet this deadline you will have to fill in and submit the Late Submission Form and provide any supporting documentation (e.g. medical cert)

All students are expected to work individually on assignments; those found collaborating with others will receive a score of zero for their work.

Include relevant steps that you used to get your answer. If you include intermediate steps, you will obtain marks even if your final answer is incorrect. Code should be as efficient and automated as possible.

Do not use the edit() function in any parts of this assignment.

All R code should include comments briefly describing how it works, and what it is doing.

**Q.1**

**(a) Create a matrix with 10 rows and 10 columns from the integers 1 to 100. Add column names and row names; the column names should be a to j (lower case) and the row names should also be A to J, but in upper case. (0.5%)**

> data <- 1:100 #setting up a vector of numbers from 1-100

> uppercase <-c(LETTERS[1:10]) #set uppercase letters for row names

> lowercase <-c(letters[1:10]) #set lowercase letters for column names

> mat <- matrix(data,nrow=10,ncol=10,byrow=TRUE, dimnames=list(uppercase,lowercase)) #create matrix with names of rows and columns

> mat

a b c d e f g h i j

A 1 2 3 4 5 6 7 8 9 10

B 11 12 13 14 15 16 17 18 19 20

C 21 22 23 24 25 26 27 28 29 30

D 31 32 33 34 35 36 37 38 39 40

E 41 42 43 44 45 46 47 48 49 50

F 51 52 53 54 55 56 57 58 59 60

G 61 62 63 64 65 66 67 68 69 70

H 71 72 73 74 75 76 77 78 79 80

I 81 82 83 84 85 86 87 88 89 90

J 91 92 93 94 95 96 97 98 99 100

**(b) From the matrix created in part (a), create a subset by extracting the first 5 rows and first 3 columns. In this subset replace the element in the 2nd row and 3rd column with 0. (0.5%)**

> new\_mat <- mat[c("A","B","C","D","E"),c("a","b","c")] create new matrix to extract criteria

> new\_mat[2,3] <- 0 #change the position 2,3 with value of 0

> new\_mat

a b c

A 1 2 3

B 11 12 0

C 21 22 23

D 31 32 33

E 41 42 43

**(c) In the matrix created in part (a), replace all values greater than 70, and less than 80 with 0. (0.5%)**

> mat[mat>70 & mat<80] <- 0 # this argument changes all values greater than 70 and less than 80 to 0

> mat

a b c d e f g h i j

A 1 2 3 4 5 6 7 8 9 10

B 11 12 13 14 15 16 17 18 19 20

C 21 22 23 24 25 26 27 28 29 30

D 31 32 33 34 35 36 37 38 39 40

E 41 42 43 44 45 46 47 48 49 50

F 51 52 53 54 55 56 57 58 59 60

G 61 62 63 64 65 66 67 68 69 70

H 0 0 0 0 0 0 0 0 0 80

I 81 82 83 84 85 86 87 88 89 90

J 91 92 93 94 95 96 97 98 99 100

**(d) In your own words and using examples in R, describe the differences between a data frame and a matrix, and a list and a vector. (0.5%)**

A matrix by definition is a rectangular array of data arranged in rows and columns. A data frame is a data table that stores multiple data types in columns called fields. Matrices are n dimensional and in comparison data frames are two dimensional. The negative of matrices is that all the data has to be the same while data frames can have different data types.

**Matrix Example**

> example <- matrix(1:9, nrow=3, ncol=3)

> example

[,1] [,2] [,3]

[1,] 1 4 7

[2,] 2 5 8

[3,] 3 6 9

**Data Frame example**

> df <- data.frame (first\_column = c("13/09/1995", "Jimmy Hehir", "25"),

+ second\_column = c("01/01/1982", "John Doe", "Old")

+ )

> df

first\_column second\_column

1 13/09/1995 01/01/1982

2 Jimmy Hehir John Doe

3 25 Old

Lists are recursive which means the can have values with different data types. Atomic vectors are not recursive which means all the values have to be the same data types which is good for manipulating data quickly.

**List Example**

> example <- list("Jimmy", c(1,2,3), TRUE, 12.3) # contains string, vector, logical values

> example

[[1]]

[1] "Jimmy"

[[2]]

[1] 1 2 3

[[3]]

[1] TRUE

[[4]]

[1] 12.3

**Vector Example**

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

> x

[1] 1 2 3 4

**Q.2**

**(a) Download the dataset (MB6300\_Assignment\_1\_data.txt) from Canvas. Read this into R. What are the dimensions of the table? What is the class? (0.5%)**

> data <-read.table("MB6300\_Assignment\_1\_data.txt") #read in data from text file

> dim(data) # This gives you the data’s dimensions

[1] 172745 13

> class(data) #This function gives you the class of the data

[1] "data.frame"

**(b) Using the apply() function and subsequent steps of your choice, find the row with the highest average expression over all samples. Which gene does this row correspond to? What is the expression in each individual sample for this row? (1%)**

> averages <- apply(data,1,mean) #this will give me averages of the rows. The apply function reads the data, selects row as given value 1 and calculates the mean

> which.max(averages) #the which.max function gives you the highest value of a declared data.

gene\_25722

155438

> data["gene\_25722",] #this extracts the row that contains gene\_25722

sample1 sample2 sample3 sample4 sample5 sample6 sample7 sample8 sample9 sample10 sample11 sample12 sample13

gene\_25722 3 115 1808074 141 3 2 0 404 112 46 10 0 0

**(c) Write a for-loop to take the place of the apply() function in part (b). Find the row with the highest average expression over all samples. Which method - apply() or for() - is more efficient and why? (1.5%)**

> averages1 <- NULL #set averages1 to null

> for(i in 1:nrow(data)) {

+ averages1[i] <- rowMeans(data[i ,])

+ } #this is the for loop that loops through the data and places the average at every loop at the ith position and storing it in averages1

> which.max(averages1)

[1] 155438

The apply() function is more efficient clearly by observing but I decided that I would time both functions just to see the difference.

**THE APPLY() FUCNTION**

> ptm <- proc.time()

> averages <- apply(data,1,mean)

> which.max(averages)

gene\_25722

155438

> # Stop the clock

> proc.time() - ptm

**user system elapsed**

**1.208 0.034 1.400**

**FOR LOOP**

> ptm <- proc.time()

> averages1 <- NULL

> for(i in 1:nrow(data)) {

+ averages1[i] <- rowMeans(data[i ,])

+ }

> which.max(averages1)

[1] 155438

> # Stop the clock

> proc.time() - ptm

**user system elapsed**

**41.026 0.214 41.678**

The apply function is more efficient as it is called once as compared to a for loop which must iterate through every time it is called in the loop. This makes it slower and should be avoided with higher level programming languages such as R and large amounts of data.

**(d) Using the method you find to be more efficient, write a function that will read in a given file, and output the row with the highest average expression (1%) (Output can be rowname, rowname and average, or the row itself).**

> highest\_avg\_exp <- function(input\_file, output\_file) {

+ data <- read.table(input\_file)

+ output <- apply(data,1,mean)

+ max1 <- which.max(output)

+ write.table(max1, output\_file)

+ } #this is a function that takes in an input file and exports an output file

>

> highest\_avg\_exp("MB6300\_Assignment\_1\_data.txt", "Q2D.txt")

**OUTPUT FILE:**

**Graphical user interface, application, Word

Description automatically generated**

**Q.3**

**(a) Extract all row names that have an ensembl gene identifier (i.e. row names that start with “ENS”) from the table given in Q.2. How many ensembl genes are there? (2%)**

> ensembl1 <- "ENS" #assign “ENS” to ensembl

> i <- substr(row.names(data), 0, nchar(ensembl1))==ensembl1 #the substr function checks the data and finds “ENS” at the start and assigns them to ensmbl1

> rows\_ens <- data[i,] #extracts the matched rows and assigns them to rows\_ens

> rows\_ens

sample1 sample2 sample3 sample4 sample5 sample6 sample7 sample8 sample9 sample10 sample11 sample12 sample13

ENSMUSG00000102057 0 0 0 0 0 0 1 0 0 1 0 1 0

ENSMUSG00000104527 0 0 0 0 0 0 0 8 0 1 0 0 0

ENSMUSG00000019868 1 5 4 22 0 0 9 1 5 3 1 7 0

ENSMUSG00000032572 3 2 0 7 0 0 3 3 0 1 2 1 0

ENSMUSG00000055128 3 0 0 1 0 0 4 1 0 0 0 4 1

ENSMUSG00000054446 4 0 6 11 0 2 2 4 6 3 1 24 1

ENSMUSG00000054452 3 0 1 4 1 0 1 0 0 5 1 3 1

ENSMUSG00000037369 0 1 1 4 2 1 0 0 1 0 1 2 1

ENSMUSG00000054453 1 2 5 2 2 2 0 1 3 2 2 5 0

ENSMUSG00000054455 3 0 4 6 2 2 3 3 0 3 2 8 2

ENSMUSG00000087830 7 9 14 16 2 5 13 7 10 19 8 26 2

ENSMUSG00000102076 0 0 0 0 43 0 0 0 0 0 0 0 0

ENSMUSG00000054523 0 0 0 0 1251 0 0 0 0 0 0 0 0

> nrow(rows\_ens)

[1] 13

**(b) Using biomaRt, download the transcript IDs for these genes. How many transcripts are there for each gene? (1%)**

> mart = useMart("ENSEMBL\_MART\_ENSEMBL")

> mart\_mouse <- useDataset("mmusculus\_gene\_ensembl", mart= mart)

> qb <- getBM(attributes = c("ensembl\_gene\_id", "description"), filters = "ensembl\_gene\_id", values = (rownames(rows\_ens)), mart = mart\_mouse)

> qb

ensembl\_gene\_id description

1 ENSMUSG00000019868 vesicle (multivesicular body) trafficking 1 [Source:MGI Symbol;Acc:MGI:1913451]

2 ENSMUSG00000032572 collagen, type VI, alpha 4 [Source:MGI Symbol;Acc:MGI:1915803]

3 ENSMUSG00000037369 lysine (K)-specific demethylase 6A [Source:MGI Symbol;Acc:MGI:1095419]

4 ENSMUSG00000054446 carboxypeptidase A1, pancreatic [Source:MGI Symbol;Acc:MGI:88478]

5 ENSMUSG00000054452 TLE family member 5, transcriptional modulator [Source:MGI Symbol;Acc:MGI:95806]

6 ENSMUSG00000054453 synaptotagmin-like 5 [Source:MGI Symbol;Acc:MGI:2668451]

7 ENSMUSG00000054455 vesicle-associated membrane protein, associated protein B and C [Source:MGI Symbol;Acc:MGI:1928744]

8 ENSMUSG00000054523 membrane-spanning 4-domains, subfamily A, member 5 [Source:MGI Symbol;Acc:MGI:2670985]

9 ENSMUSG00000055128 cell growth regulator with ring finger domain 1 [Source:MGI Symbol;Acc:MGI:1916368]

10 ENSMUSG00000102057 predicted gene 29594 [Source:MGI Symbol;Acc:MGI:5580300]

11 ENSMUSG00000102076 predicted gene, 20561 [Source:MGI Symbol;Acc:MGI:5295668]

12 ENSMUSG00000104527 predicted gene, 37332 [Source:MGI Symbol;Acc:MGI:5610560]

**(c) What is the amino acid sequence for ENSMUSG00000000120? (0.5%)**

> x <- getSequence(id="ENSMUSG00000000120", type = "ensembl\_gene\_id", seqType = "peptide", mart= mart\_mouse)

> x$peptide

[1] "MRRAGAACSAMDRLRLLLLLLLLLGVSFGGAKETCSTGMYTHSGECCKACNLGEGVAQPCGANQTVCEPCLDSVTFSDVVSATEPCKPCTECLGLQSMSAPCVEADDAVCRCSYGYYQDEETGRCEACSVCGVGSGLVFSCQDKQNTVCEECPEGTYSDEANHVDPCLPCTVCEDTERQLRECTPWADAECEEIPGRWITRSTPPEGSDVTTPSTQEPEAPPERDLIASTVADTVTTVMGSSQPVVTRGTADNLIPVYCSILAAVVVGLVAYIAFKRWNSCKQNKQGANSRPVNQTPPPEGEKLHSDSGISVDSQSLHDQQTHTQTASGQALKGDGNLYSSLPLTKREEVEKLLNGDTWRHLAGELGYQPEHIDSFTHEACPVRALLASWGAQDSATLDALLAALRRIQRADIVESLCSESTATSPV\*"

**(d) What is the length of the amino acid sequence? (0.5%)**

> nchar(x$peptide)

[1] 428