SLE712_Assignment_3

Jasmine Catague (Student ID:220525798), Daisy Ta (Student ID: 219495475), Mihiravi Arachchige (Student ID: 220045141)

21/05/2021

PART 1: Importing files, data wrangling, plots and saving code on GitHub

Repository link for our assignment 3: https://github.com/Ngocanhttna0609/Our-SLE712-Assessment3/tree/main (https://github.com/Ngocanhttna0609/Our-SLE712-Assessment3/tree/main)

Question 1

Here is the downloaded file from github gene_expression2.tsv dataset containing the RNA-seq count data for two samples of interest. Firstly, we download the file by using the download.file command and then using the destfile command to label the file in our folder. To read the gene_expression2.tsv we use read.table command selecting the “gene_expression.tsv” file and then using the command header=TRUE to read files with the labels in the first row. After that, we use [] to access a gene by a gene name. Finally, we output the first six rows using [1:6,].

```
# Download the file gene_expression.tsv
download.file ("https://raw.githubusercontent.com/markziemann/SLE712_files/master/assessment_task3/bioinfo_asst3_part1_files/gene_expression.tsv",
  destfile= "gene_expression.tsv")
# Read in the file
gene_exp <- read.table("gene_expression.tsv", header=TRUE, row.names=1)
str(gene_exp)
```

```
## 'data.frame':    58302 obs. of  2 variables:
##  $ SRR5150592: int  1 0 0 0 0 0 0 0 0 0 ...
##  $ SRR5150593: int  0 1 0 0 0 0 0 0 0 0 ...
```

```
# An example of trying to access a gene by gene name
gene_exp["ENSG00000227232", ]
```

```
##                 SRR5150592 SRR5150593
## ENSG00000227232          0          1
```

```
# a table of values for the first six genes.
gene_exp[1:6,]
```

```
##                 SRR5150592 SRR5150593
## ENSG00000223972          1          0
## ENSG00000227232          0          1
## ENSG00000278267          0          0
## ENSG00000243485          0          0
## ENSG00000284332          0          0
## ENSG00000237613          0          0
```

Question 2

We create a new column for the means of the other columns by using the rowMeans command. Then, we create a new column use the command c(name). Finally, we present it in a table with just the first six genes by using the command [1:6,] wherein [row,col].

```
# Calculate the mean of the rows
mean_row<-rowMeans(gene_exp)
# Make a new mean column
gene_exp$mean<-c(mean_row)
# Show the first six genes
gene_exp[1:6,]
```

```
##                 SRR5150592 SRR5150593 mean
## ENSG00000223972          1          0  0.5
## ENSG00000227232          0          1  0.5
## ENSG00000278267          0          0  0.0
## ENSG00000243485          0          0  0.0
## ENSG00000284332          0          0  0.0
## ENSG00000237613          0          0  0.0
```

Question 3

We use the order command to sort the genes first but this gives us the lowest to highest mean values of the genes. So then, we use the tail command which selects the last rows of the data set which corresponds to the top 10 highest gene expressions.

```
# Create sorted data frame from the ordered mean column
sorted_gene_exp<-gene_exp[order(gene_exp$mean),]
# Get the 10 highest mean values.
top10_genes<-row.names(tail(sorted_gene_exp,10))
top10_genes
```

```
##  [1] "ENSG00000108821" "ENSG00000198712" "ENSG00000196924" "ENSG00000198786"
##  [5] "ENSG00000198804" "ENSG00000137801" "ENSG00000198886" "ENSG00000075624"
##  [9] "ENSG00000210082" "ENSG00000115414"
```

Question 4

To determine the number of genes with a mean of more than 10 we use the command sum() and then select the mean value with <10. Then, using cat command we can separate character strings to print them properly.

```
# Calculate the numbers of genes with a mean < 10
number_genes <-sum(gene_exp$mean<10)
number_genes
```

```
## [1] 43124
```

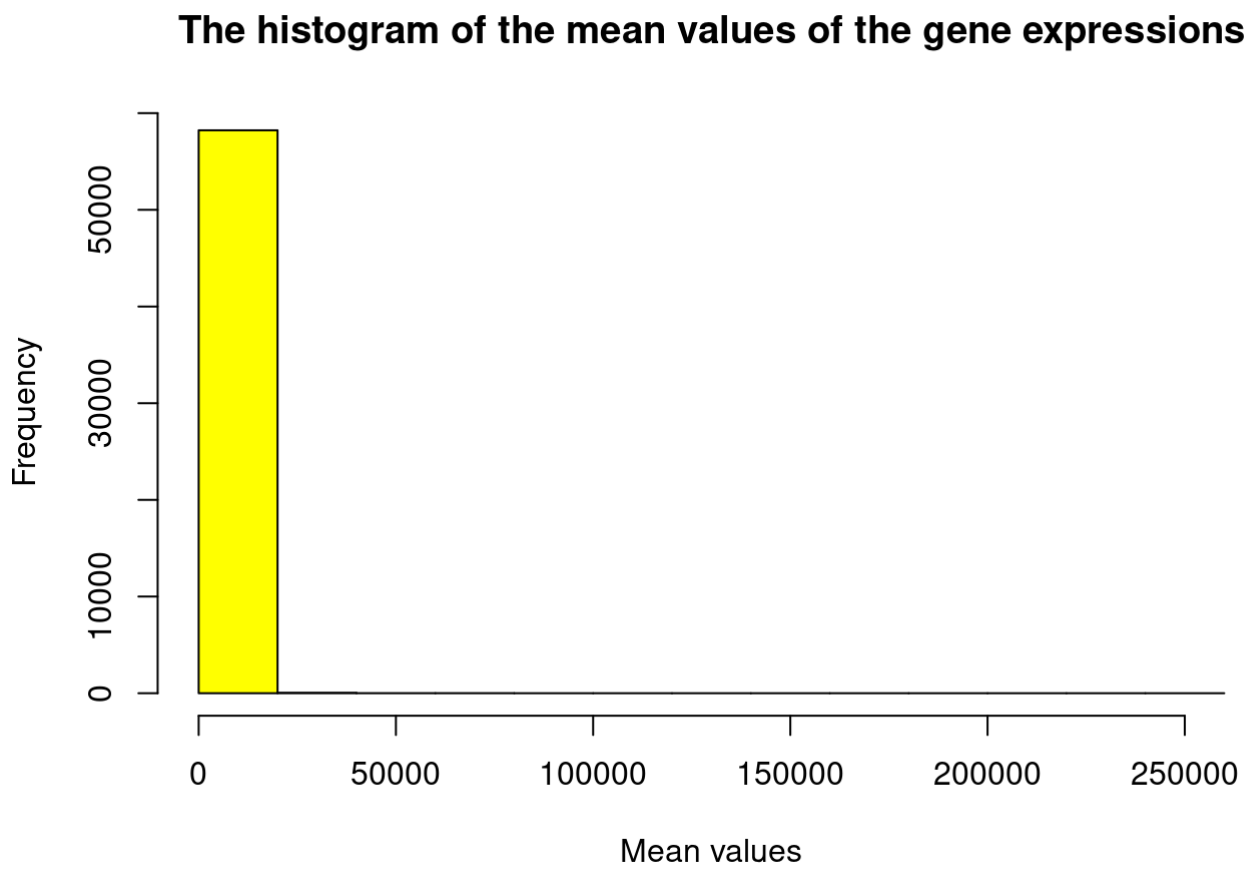
```
# Show the result
cat("The number of genes with a mean of < 10 is", number_genes)
```

```
## The number of genes with a mean of < 10 is 43124
```

Question 5

To create a histogram of the mean values we use the hist command. Then, to label the histogram main= title of the histogram, xlab=to label the x-axis and then ylab= to label the y-axis.

```
hist(gene_exp$mean, main= "The histogram of the mean values of the gene expressions",
xlab="Mean values", col="yellow")
```



Question 6

We download the growth_data2.csv by using the download.file command and then destfile command to save the file in the folder.To read the file we use the read.csv command and then use header=TRUE to read files with the labels in the first row. Then use the stringsAsFactors=FALSE command to re-encode strings. We use

str and head command to make sure the data has been successful imported. Finally, we use colnames command to get the column names.

```
download.file("https://raw.githubusercontent.com/markziemann/SLE712_files/master/asse
ssment_task3/bioinfo_asst3_part1_files/growth_data.csv",
              destfile = "growth_data2.csv")

growth_data <-read.csv("growth_data2.csv", header= TRUE, stringsAsFactors =FALSE)
str(growth_data)
```

```
## 'data.frame':    100 obs. of  6 variables:
##  $ Site           : chr  "northeast" "northeast" "northeast" "northeast" ...
##  $ TreeID         : chr  "A003" "A005" "A007" "A008" ...
##  $ Circumf_2004_cm: num  5.2 4.9 3.7 3.8 3.8 5.9 4.4 5.3 7.1 3.8 ...
##  $ Circumf_2009_cm: num  10.1 9.6 7.3 6.5 6.4 10 9.9 9 12 7.4 ...
##  $ Circumf_2014_cm: num  19.9 18.9 14.3 10.9 10.9 16.8 22.2 15.2 20.2 14.5 ...
##  $ Circumf_2019_cm: num  38.9 37 28.1 18.5 18.4 28.4 50 25.8 34.2 28.4 ...
```

```
head(growth_data)
```

```
##        Site TreeID Circumf_2004_cm Circumf_2009_cm Circumf_2014_cm
## 1 northeast   A003             5.2            10.1            19.9
## 2 northeast   A005             4.9             9.6            18.9
## 3 northeast   A007             3.7             7.3            14.3
## 4 northeast   A008             3.8             6.5            10.9
## 5 northeast   A011             3.8             6.4            10.9
## 6 northeast   A012             5.9            10.0            16.8
##   Circumf_2019_cm
## 1            38.9
## 2            37.0
## 3            28.1
## 4            18.5
## 5            18.4
## 6            28.4
```

```
# Get the column names
colnames(growth_data)
```

```
## [1] "Site"            "TreeID"          "Circumf_2004_cm" "Circumf_2009_cm"
## [5] "Circumf_2014_cm" "Circumf_2019_cm"
```

Question 7

We use the mean and sd command to calculate mean and standard deviation of tree circumference respectively in 2004 and 2019. Then, we use the cat command to show the results. Mean and sd of tree circumference in 2004 are 5.077 and 1.054462 respectively. Mean and sd of tree circumference in 2019 are 49.912 and 22.17979, respectively.

```
#The mean and standard deviation of tree circumference at the start (in 2004)
mean_2004 <- mean(growth_data$Circumf_2004_cm)
cat("Mean in 2004 is", mean_2004)
```

```
## Mean in 2004 is 5.077
```

```
sd_2004 <- sd(growth_data$Circumf_2004_cm)
cat("SD in 2004 is", sd_2004)
```

```
## SD in 2004 is 1.054462
```

```
#The mean and standard deviation of tree circumference at the end (in 2019)
mean_2019 <- mean(growth_data$Circumf_2019_cm)
cat("Mean in 2019 is", mean_2019)
```

```
## Mean in 2019 is 49.912
```

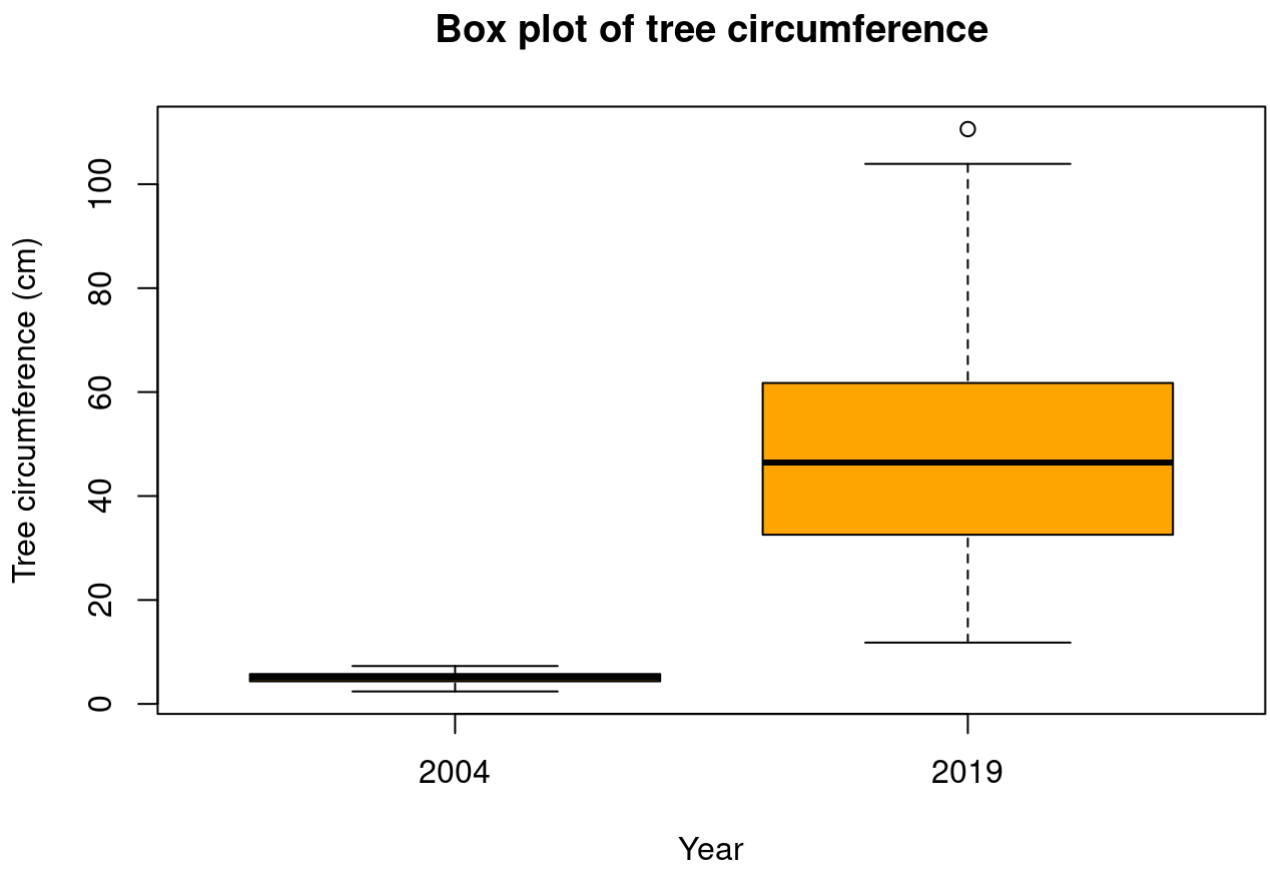
```
sd_2019 <- sd(growth_data$Circumf_2019_cm)
cat("SD in 2019 is", sd_2019)
```

```
## SD in 2019 is 22.17979
```

Question 8

Firstly, we get the tree circumference in 2004 and 2019 by using $ sign. Then, we use boxplot command to make a box plot of tree circumference at the start and end of the study at both sites. Then, to label the boxplot we used main for the title of the histogram, xlab=to label the x-axis and then ylab= to label the y-axis.

```
# Get the tree circumference in 2004 and 20009
x_vals <- growth_data$Circumf_2004_cm
y_vals <- growth_data$Circumf_2019_cm
# Make a box plot
boxplot(x_vals, y_vals,
main = "Box plot of tree circumference",
names = c("2004", "2019"),
xlab = "Year",
ylab = "Tree circumference (cm)",
col = "orange"
)
```



Question 9

Firstly, we get the tree circumference values of each site by using subset() command selecting the sites. Then, we use mean command to calculate the mean values in 2009 and 2019 of each site before we calculate the mean growth. The mean growth over the past 10 years of Northeast and Southwest are 30.076cm and 48.354cm respectively.

```
# Get the tree circumference values of each site
northeast <- subset(growth_data, Site=="northeast")
southwest <- subset(growth_data, Site=="southwest")
# Calculate the mean values of each site over past 10 years
na_2019 <- mean(northeast$Circumf_2019_cm)
sw_2019 <- mean(southwest$Circumf_2019_cm)
na_2009 <- mean(northeast$Circumf_2009_cm)
sw_2009 <- mean(southwest$Circumf_2009_cm)
# Calculate the mean growth of each site
na_mean_growth <- na_2019 - na_2009
sw_mean_growth <- sw_2019 - sw_2009
cat("The mean growth over the past 10 years of Northeast is", na_mean_growth)
```

```
## The mean growth over the past 10 years of Northeast is 30.076
```

```
cat("The mean growth over the past 10 years of Southwest is", sw_mean_growth)
```

```
## The mean growth over the past 10 years of Southwest is 48.354
```

Question 10

We get the growth values of tree circumference on each site by using $. Then, we use t.test() and wilcox.test() to run t.test and wilcox.test before getting the p-value. We use cat() command to show the result. The p-value of t.test is 1.712524e-06 and p-value of wilcox.test is 4.6264e-06 which are both p<0.01, meaning they are statistically significant at the level of 1%.

```
#get the growth values of each site
na_growth <- northeast$Circumf_2019_cm - northeast$Circumf_2009_cm
sw_growth <- southwest$Circumf_2019_cm - southwest$Circumf_2009_cm
#run t.test
t <- t.test(na_growth, sw_growth)
#get p-value of t.test
t_pvalue <- t$p.value
cat("p-value of t.test is", t_pvalue)
```

```
## p-value of t.test is 1.712524e-06
```

```
#run wilcox.test
wilcox <- wilcox.test(na_growth, sw_growth)
# get p-value of wilcox.test
wilcox_pvalue <- wilcox$p.value
cat("p-value of wilcox.test is", wilcox_pvalue)
```

```
## p-value of wilcox.test is 4.6264e-06
```

PART 2: Determine the limits of BLAST

```
# Loading the library
suppressPackageStartupMessages({
  library("seqinr")
  library("magrittr")
  library("kableExtra")
  library("R.utils")
  library("rBLAST")
  library("Biostrings")
  source("https://raw.githubusercontent.com/markziemann/SLE712_files/master/assessmen
t_task3/bioinfo_asst3_part2_files/mutblast_functions.R")
})
```

Question 1

We use download.file to download the whole set of E. coli gene DNA sequences and and use R.utils::gunzip to decompress. Then, we use makeblastdb() function to create a BLAST database with dbtype is nucleic. There are 4140 sequences which present in the E.coli set.

```
# Download the whole set of E. coli gene DNA sequences
download.file("ftp://ftp.ensemblgenomes.org/pub/bacteria/release-42/fasta/bacteria_0_
collection/escherichia_coli_str_k_12_substr_mg1655/cds/Escherichia_coli_str_k_12_subs
tr_mg1655.ASM584v2.cds.all.fa.gz",
              destfile = "Escherichia_coli_str_k_12_substr_mg1655.ASM584v2.cds.all.f
a.gz")
# Use gunzip to decompress
R.utils::gunzip("Escherichia_coli_str_k_12_substr_mg1655.ASM584v2.cds.all.fa.gz", ove
rwrite=TRUE)
# Create a BLAST database
makeblastdb("Escherichia_coli_str_k_12_substr_mg1655.ASM584v2.cds.all.fa", dbtype="nu
cl", "-parse_seqids")
```

Question2

We use download.file to download the sample fasta sequence. Then, we use seqinr::read.fasta to read them in. We use the command [] to choose our sequence of interest (number 1). After that, we use getLength() and seqinr::GC() to determine the length and the proportion of GC bases, respectively. Finally, we use cat() to show the results. The length of my selected sequence is 615. The proportion of GC bases in the sequence is about 55.6%.

```
# Download the sample fasta sequences
download.file("https://raw.githubusercontent.com/markziemann/SLE712_files/master/asse
ssment_task3/bioinfo_asst3_part2_files/sample.fa",
              destfile = "sample.fa")
# Read the sequences
SEQ <- seqinr::read.fasta("sample.fa")
# Select one sequence
my_seq <- SEQ[[1]]
# The length of my selected sequence
length_seq <- seqinr::getLength(my_seq)
# The proportion of GC bases
seqinr::GC(my_seq)
```

```
## [1] 0.5560976
```

```
# Show the results
cat(" The length of my selected sequence is", length_seq)
```

```
##  The length of my selected sequence is 615
```

```
cat("The proportion of GC bases is", seqinr::GC(my_seq))
```

```
## The proportion of GC bases is 0.5560976
```

Question 3

We use myblastn_tab to perform BLAST search. Then, using the str command to see the structure of the dataset. After that, we used as.character to identify which E.coli genes matches with the sequence of interest best. There is only one sequence that meets the demand because this gene does not share any matches with

any other genes in the database.

```
# Create BLAST databases and perform BLAST searches
myblastn_tab
```

```
## function (myseq, db)
## {
##     mytmpfile1 <- tempfile()
##     mytmpfile2 <- tempfile()
##     write.fasta(myseq, names = attr(myseq, "name"), file.out = mytmpfile1)
##     system2(command = "/usr/bin/blastn", args = paste("-db ",
##         db, " -query", mytmpfile1, "-outfmt 6 -evalue 0.05 -ungapped >",
##         mytmpfile2))
##     res <- NULL
##     if (file.info(mytmpfile2)$size > 0) {
##         res <- read.csv(mytmpfile2, sep = "\t", header = FALSE)
##         colnames(res) <- c("qseqid", "sseqid", "pident", "length",
##             "mismatch", "gapopen", "qstart", "qend", "sstart",
##             "send", "evalue", "bitscore")
##     }
##     unlink(c(mytmpfile1, mytmpfile2))
##     if (!is.null(res)) {
##         res <- res[order(-res$bitscore), ]
##     }
##     res
## }
```

```
res <- myblastn_tab(myseq= my_seq, db = "Escherichia_coli_str_k_12_substr_mg1655.ASM584v2.cds.all.fa")
str(res)
```

```
## 'data.frame':    1 obs. of  12 variables:
##  $ qseqid  : int 1
##  $ sseqid  : chr "AAC76851"
##  $ pident  : num 100
##  $ length  : int 615
##  $ mismatch: int 0
##  $ gapopen : int 0
##  $ qstart  : int 1
##  $ qend    : int 615
##  $ sstart  : int 1
##  $ send    : int 615
##  $ evalue  : num 0
##  $ bitscore: int 1183
```

```
#  Identify what E. coli gene my sequence matches best
top_hits <- as.character(res$sseqid[1:3])
top_hits
```

```
## [1] "AAC76851" NA         NA
```

```
# Show a table of the top 3 hits
head(res, 3)[, c("qseqid", "sseqid", "pident", "evalue","bitscore")]
```

```
##   qseqid   sseqid pident evalue bitscore
## 1      1 AAC76851    100      0     1183
```

Question 4

We use mutator() function to create mutated sequence with 100 point mutations and then compare with the original sequence by making a pairwise alignment by pairwiseAlignment from Biostrings library. Firstly, we read in my selected sequence. Then, we create a mutated copy with 100 substitutions. After that, we use DNAString() to convert to biostring. We use mmismatch() to get the number of mismatches. Finally, we show the result by using cat(). The number of mismatches between the original and mutated sequence is shown below.

```
# Read in
tophit <- seqinr::read.fasta("sample.fa")
tophit <- tophit[[1]]
str(tophit)
```

```
##  'SeqFastadna' chr [1:615] "a" "t" "g" "g" "a" "a" "a" "g" "c" "t" "g" "g" ...
##  - attr(*, "name")= chr "1"
##  - attr(*, "Annot")= chr ">1     "
```

```
# Mutate
tophit_mut <- mutator(myseq = tophit, nmut = 100)
## Perform pairwise alignment to prove that the mutation has worked as expected
# Convert to biostring
tophit_f <- DNAString(c2s(tophit))
tophit_mut_f <- DNAString(c2s(tophit_mut))
aln <- Biostrings::pairwiseAlignment(tophit_f,tophit_mut_f)
pid(aln)
```

```
## [1] 87.80488
```

```
# Get the number of mismatch
 nmismatch(aln)
```

```
## [1] 75
```

```
# Show the result
cat("The number of mismatches between the original and mutated sequence is", nmismatc
h(aln))
```

```
## The number of mismatches between the original and mutated sequence is 75
```

Question 5

Firstly, we have to write the blast index first using makeblastdb. Then, to determine the number and proportion of sites that need to be altered to prevent the BLAST search from matching the gene of origin we can set a number of mutations in the sequences first by using the mutator command.With this we have trial and error with different values and 200 mutaions gave us a null result meaning there is no more matches from BLAST.To

find the proportion of sites in the sequence we can use the replicate command. This enables us to set a number of replicates giving proportions of successful BLASTs by increasing number of random bases. Then using 200 replicates choosing random sites of mutation in the sequence we used the sapply command performing the function command which gives an output in vector. In return, this gave us the proportion of successful BLASTs in the chosen sites in the sequence.

```
#We need to write the blast index first
makeblastdb(file="sample.fa", dbtype="nucl")

#We can test a random number of mutations first and see whether we can get any matche
s from BLAST.
tophit_mut<-mutator(myseq=tophit,nmut=80)

res<-myblastn_tab(myseq=tophit_mut,db="sample.fa")
res
```

```
##   qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue
## 1      1      1 89.379    612       65       0      1  612      1  612      0
##   bitscore
## 1      802
```

```
#We got matches when we tried to put 80 mutations. Hence,
!is.null(res)
```

```
## [1] TRUE
```

```
cat("The result for any matches of the mutated sequence is",!is.null(res) )
```

```
## The result for any matches of the mutated sequence is TRUE
```

```
#We must increase the number of mutations to see whether we can still find matches in
the BLAST. So now lets increase it by 200.
tophit_mut<-mutator(myseq=tophit,nmut=200)

res<-myblastn_tab(myseq=tophit_mut,db="sample.fa")
res
```

```
## NULL
```

```
!is.null(res)
```

```
## [1] FALSE
```

```
cat("The result for any matches of the mutated sequence is",!is.null(res) )
```

```
## The result for any matches of the mutated sequence is FALSE
```

```
#To find the proportion of sites in the sequence we can use the replicate command.
rep_mut<-function(NMUT){
  tophit_mut<-mutator(myseq=tophit,nmut=NMUT)
  res<-myblastn_tab(myseq=tophit_mut,db="sample.fa")
  as.numeric(!is.null(res))
}

replicate(n=200, rep_mut(200))
```

```
##   [1] 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
##  [38] 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
##  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
## [112] 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
## [149] 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
## [186] 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
```

```
#The sites that we have changed for our 615 bp sequence.
n_mut_range<-c(0,60,120,180,240)

replicator_function<-function(NMUT){
  mean(replicate(n=200,rep_mut(NMUT)))
}
myresult<-sapply(n_mut_range, replicator_function)
myresult
```

```
## [1] 1.00 1.00 0.65 0.11 0.01
```

Question 6

Now we can construct a chart of the increasing proportion of mutated bases by the decreasing proportion of succesful BLASTs to match the gene sequence. In here, we can conclude that from the orginal sequence with a length of 615 bp with 40% of the randomised bases gave us 0% successful BLAST. This highlights the limitations of BLASTs to give successful matches when sites in the sequence are randomised we can say that about 30-40% randomised bases BLAST becomes unsuccesful.

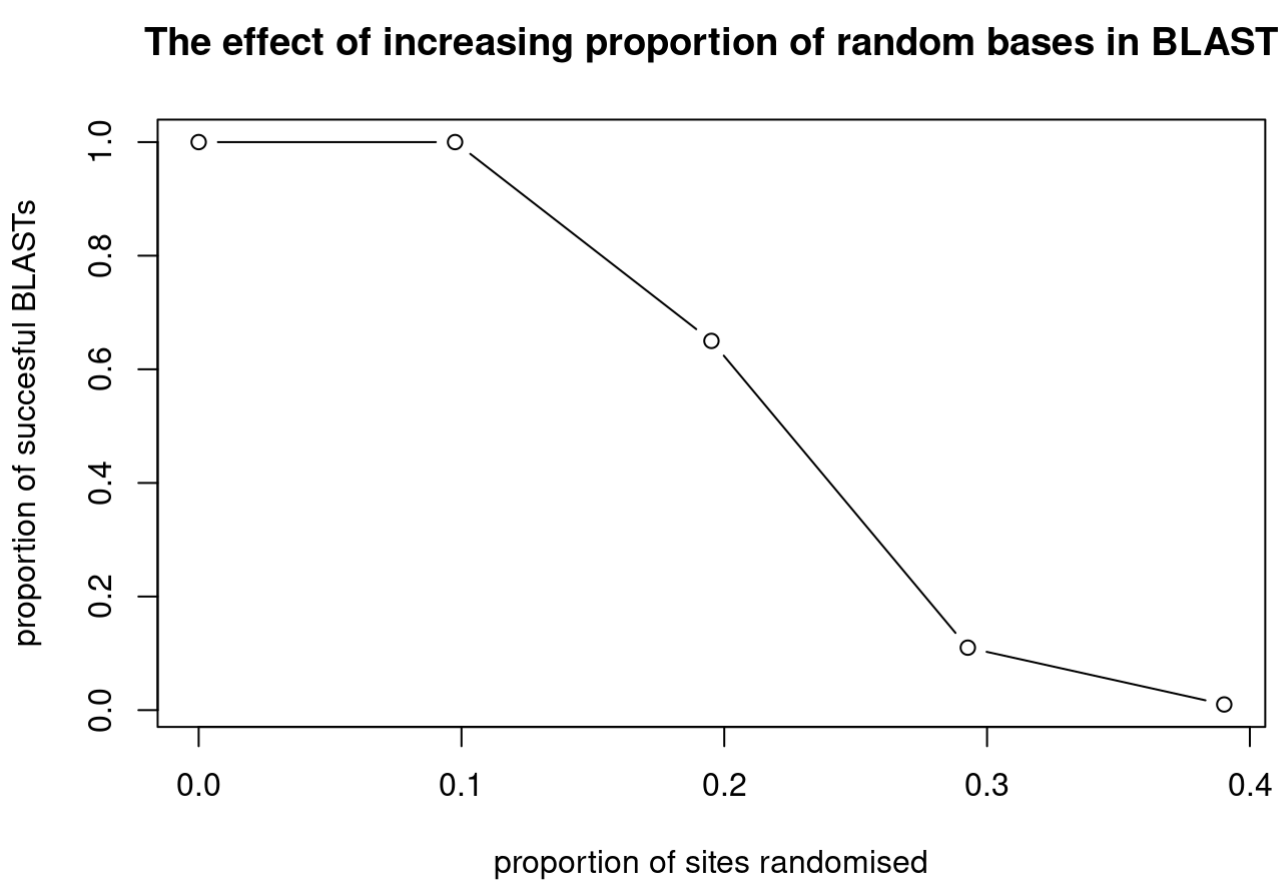
```
#To create a chart we can make a scatterplot of the proportions of sites and succesfu
l BLASTs.

#The proportion of sites randomised from the total sequence length of 615 bp, 200 rep
eats using gene 1.

prop_seq<-n_mut_range/length_seq

#The proportion of successful BLASTs from the previous result
prop_blast<-c(myresult)

plot(prop_seq,prop_blast,
     type='b',
     main="The effect of increasing proportion of random bases in BLAST",
     xlab="proportion of sites randomised",
     ylab="proportion of succesful BLASTs"
     )
```



For reproducibility, session info is added at the end

```
sessionInfo()
```

```
## R version 4.1.0 (2021-05-18)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.2 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0
##
## locale:
##  [1] LC_CTYPE=en_AU.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_AU.UTF-8        LC_COLLATE=en_AU.UTF-8
##  [5] LC_MONETARY=en_AU.UTF-8    LC_MESSAGES=en_AU.UTF-8
##  [7] LC_PAPER=en_AU.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_AU.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] rBLAST_0.99.2       Biostrings_2.60.0   GenomeInfoDb_1.28.0
##  [4] XVector_0.32.0      IRanges_2.26.0      S4Vectors_0.30.0
##  [7] BiocGenerics_0.38.0 R.utils_2.10.1      R.oo_1.24.0
## [10] R.methodsS3_1.8.1   kableExtra_1.3.4    magrittr_2.0.1
## [13] seqinr_4.2-5
##
## loaded via a namespace (and not attached):
##  [1] bslib_0.2.5.1          compiler_4.1.0         jquerylib_0.1.4
##  [4] highr_0.9              bitops_1.0-7           zlibbioc_1.38.0
##  [7] prettyunits_1.1.1      tools_4.1.0            progress_1.2.2
## [10] digest_0.6.27          jsonlite_1.7.2         evaluate_0.14
## [13] lifecycle_1.0.0        viridisLite_0.4.0      pkgconfig_2.0.3
## [16] rlang_0.4.11           rstudioapi_0.13        yaml_2.2.1
## [19] xfun_0.23              GenomeInfoDbData_1.2.6 stringr_1.4.0
## [22] httr_1.4.2             knitr_1.33             xml2_1.3.2
## [25] vctrs_0.3.8            sass_0.4.0             systemfonts_1.0.2
## [28] hms_1.1.0              ade4_1.7-16            webshot_0.5.2
## [31] svglite_2.0.0          glue_1.4.2             R6_2.5.0
## [34] rmarkdown_2.8          scales_1.1.1           ellipsis_0.3.2
## [37] htmltools_0.5.1.1      MASS_7.3-54            rvest_1.0.0
## [40] colorspace_2.0-1       stringi_1.6.2          RCurl_1.98-1.3
## [43] munsell_0.5.0          crayon_1.4.1
```