

RNA-Seq analysis

PhD Gamaliel López Leal

Max Planck Tandem Group in Computational Biology

Los Andes University

Mails: gamlopez@ccg.unam.mx, lgama1108@gmail.com

MSc Laura Carolina Camelo Valera

Max Planck Tandem Group in Computational Biology

Los Andes University

Mail: lc.camelo10@uniandes.edu.co

Create work directories

Before we start, we need to create all directories in where we going to work.

Each one of you have a home directory `\home` in which we are going to create all directories. To do this we use the comand `mkdir` followed by the name directory.

```
mkdir FASTQS
```

List all files in the FASTQS directory with the `ls` comand

```
phi11510-28a_R1.fastq  phi11510-28b_R1.fastq  phi11510-37a_R1.fastq  phi11510-37b_R1.fastq
phi11510-28a_R2.fastq  phi11510-28b_R2.fastq  phi11510-37a_R2.fastq  phi11510-37b_R2.fastq
```

Quality controls

First you have to remove sequencing adapters and those reads which have poor quality, before downstream analysis. To do this, you want to use TRIM_GALORE (https://www.bioinformatics.babraham.ac.uk/projects/trim_galore/).

To display the menu:

```
trim_galore --help
```

Run TRIM_GALORE using some default parameters.

```
trim_galore --fastqc --paired --retain_unpaired R1.fastq R2.fastq
```

If you want to look the results using strigment quality criteria you can use the `-q/--quality` option and or

```
--length_1 y --length_2 creating a new output directory
```

```
trim_galore --paired --retain_unpaired --length_1 40 R1.fastq R2.fastq --  
output_dir OutDir_paired_40
```

Read Mapping

Create a new work directory using `mkdir` named as **ReadMapping** First, you need to format the reference genome with the command `bowtie2-build`. To display the menu use the option `--help`

```
bowtie2-build --help
```

Formatting the reference genome.

```
bowtie2-build reference.fa REF
```

Mapping quality reads.

```
bowtie2 --no-unal -x REF -1 QualityReads_R1.fastq -2 QualityReads_R2.fastq -S  
outFile.sam
```

You can use the options `--sensitive` and `--very sensitive` to use strigment criteria.

The file `outFile.sam` contains only the mapped reads because we used the option `--no-unal`.

You can use the `grep -c` option to count the mapped reads or converter the sam file in a .bam file (binary) to count the mapped using the `-F 0x04` or `-F 4` flag

```
samtools view -bSo MappedReads.bam MappedReads.sam
samtools sort MappedReads.bam -o MappedReads_SORTED.bam
samtools view -F 0x04 -c MappedReads_SORTED.bam
samtools index MappedReads_SORTED.bam
samtools view -h -o MappedReads_SORTED.sam MappedReads_SORTED.bam
```

Read Count

An important aspect in RNA-Seq analyzes is to incorporate in a table the number of reads mapped for each gene. For this, we first we have to perform a gene calling and annotation of the reference genome. This time we are going to use prokka (<https://github.com/tseemann/prokka>)

PROKKA

```
prokka ReferenceGenome.fna
```

PROKKA Output Files

Extension	Description
.gff	This is the master annotation in GFF3 format, containing both sequences and annotations. It can be viewed directly in Artemis or IGV.
.gbk	This is a standard Genbank file derived from the master .gff. If the input to prokka was a multi-FASTA, then this will be a multi-Genbank, with one record for each sequence.
.fna	Nucleotide FASTA file of the input contig sequences.
.faa	Protein FASTA file of the translated CDS sequences.
.ffn	Nucleotide FASTA file of all the prediction transcripts (CDS, rRNA, tRNA, tmRNA, misc_RNA)
.sqn	An ASN1 format "Sequin" file for submission to Genbank. It needs to be edited to set the correct taxonomy, authors, related publication etc.
.fsa	Nucleotide FASTA file of the input contig sequences, used by "tbl2asn" to create the .sqn file. It is mostly the same as the .fna file, but with extra Sequin tags in the sequence description lines.
.tbl	Feature Table file, used by "tbl2asn" to create the .sqn file.
.err	Unacceptable annotations - the NCBI discrepancy report.
.log	Contains all the output that Prokka produced during its run. This is a record of what settings you used, even if the --quiet option was enabled.
.txt	Statistics relating to the annotated features found.
.tsv	Tab-separated file of all features: locus_tag,ftype,len_bp,gene,EC_number,COG,product

Vizualize the reads using Artemis and analyzed the read count

Open in Artemis your reference annotated genome (.gbk files from NCBI) and load mapped reads.

```
File -> Read BAM / VCF option
Select -> Select CDS Features -> click on the pink region rosa and select the
option Analyzed Read Count
```

Read Count using HTSeq

If you are working with a very large dataset (mouse genome) we recommend using HTSeq (<https://github.com/htseq>). Before use HTSeq, the prokka .gff file need to be formatted using the script `formatGFF.pl`

```
perl formatGFF.pl PROKKA.gff > new-formatted_prokka.gff
```

run `htseq-count`

```
/usr/local/bin/htseq-count -m union -a 2 -s no --idattr=gene_id aln-28a-SORTED.sam new-formatted_prokka.gff -o COUNTS > Counts-28a.txt
```

Practice 1.

Generate the expression tables for each condition. Feel free to use Artemis or HTSeq

Introduction to R

R is a free programming environment integrating tools to perform data manipulation, statistical computing and plot generation. [See More](#).

Getting Started

To start, please, create a directory in the shell called `IntroductionR`, then enter to it:

```
mkdir IntroductionR
cd IntroductionR
```

To open an R terminal, type `R` then press Enter. We strongly recommend to use R in a different working directory for each project. You can check the working directory which you are using with the `getwd()` command. For the development of this workshop we will be using the following working directory:

```
/home/users/your-login-id/IntroductionR
```

If you would like to change the working directory, you could use the command `setwd()` as follows

```
setwd('/newdirectorypath')
```

Data Types

In R you can create different objects of different data types ([vectors](#), [matrices](#), [data frames](#), functions, lists, etc...). To list the entities you create during an R session, you can use the commands `ls()` or `objects()`. All the objects created on a given environment can be saved after finishing each R session by answering `y` when it ask to save the workspace image or by using the command `save.image()`. The data will be stored at a file with a `.RData` extension.

To remove an object you could use the R command `rm()`:

```
rm(name-object1,name-object2, ..., name-object-n)
```

Vectors

The simplest data type in R are vectors, they are a collection of elements, all of them of the same type.

Numeric vectors

Vectors composed of numbers. With this kind of vectors you can perform arithmetic calculations

```
num_vector <- c(38, 66, 12, 105, 2, 17)
```

You could also create numeric vectors generating sequences or repetitions.

```
#Generating a regular sequence in ascending order from 1 to 30 incrementing by 2
sequence_vector<- seq(1,30, by = 2)
sequence_vector
[1] 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29

#Generating a vector of twenties of length 10 (parameter: times)
rep_constant<-rep(20, times=10)
rep_constant
[1] 20 20 20 20 20 20 20 20 20 20

#Generating a vector of three times num_vector (parameter:times)
rep_numvector<-rep(num_vector,times = 3)
rep_numvector
[1] 38 66 12 105 2 17 38 66 12 105 2 17 38 66 12 105 2 17

#Generating a vector repeating all the elements of num_vector three times each
one (parameter: each)
rep_each_numvector<-rep(num_vector,each = 3)
rep_each_numvector
[1] 38 38 38 66 66 66 12 12 12 105 105 105 2 2 2 17 17 17
```

Arithmetic Operators

To add, subtract, multiply or divide a constant value to all elements of a vector.

```

#Addition
add_constant<- num_vector + 3
add_constant
[1] 41 69 15 108 5 20

#Subtraction
sub_constant<- num_vector - 3
sub_constant
[1] 35 63 9 102 -1 14

#Multiplication
mult_constant<- num_vector * 3
mult_constant
[1] 114 198 36 315 6 51

#Division
div_constant<- num_vector / 3
div_constant
[1] 12.6666667 22.0000000 4.0000000 35.0000000 0.6666667 5.6666667

```

To add/subtract 2 vectors.

```

# Let's create another vector
num_vector1<- c(6,5,4,3,2,1)

#Addition
add_vectors <- num_vector + num_vector1
add_vectors
[1] 44 71 16 108 4 18

#Subtraction
sub_vectors <- num_vector - num_vector1
sub_vectors
[1] 32 61 8 102 0 16

#Multiplication
mult_vectors <- num_vector * num_vector1
mult_vectors
[1] 228 330 48 315 4 17

#Division
div_vectors <- num_vector / num_vector1
div_vectors
[1] 6.333333 13.200000 3.000000 35.000000 1.000000 17.000000

```

Character vectors

Vectors composed of characters

```
char_vector <- c("q", "z", "p", "a", "x", "mp")
```

Logical vectors

Vectors composed of logical values: True or False

```
logic_vector<-c(FALSE,TRUE,TRUE,FALSE,TRUE,FALSE) or logic_vector<-c(F,T,T,F,T,F)
```

You can also create them by using conditions

```
num_vector
[1] 38 66 12 105 2 17

# Creating a logical vector based on a condition
cond_vector<- (num_vector > 20)
cond_vector
[1] TRUE TRUE FALSE TRUE FALSE FALSE
```

To access a vector in any position you could use the following command:

```
vector_name[position]
```

Examples

```
#Obtaining the number at the third position of num_vector
num_vector[3]
[1] 12


#Obtaining the character at the fifth position of char_vector
char_vector[5]
[1] "x"

#Obtaining the logical value at the second position of cond_vector
cond_vector[2]
[1] TRUE
```

Matrices

Matrices, as vectors, are formed of elements of the same type. But matrices have more than 1 dimension. They have columns and rows, defined as shown below.

		ROW			
C O L U M N					



Reference. (<https://www.differencebetween.com/wp-content/uploads/2018/09/Difference-Between-Row-and-Column-fig-2.png>)

To create a matrix you could use the following commands

```
#Creating a matrix of zeros having 10 rows and 5 columns
zero_matrix<-matrix(0,nrow=10,ncol=5)
zero_matrix
[,1] [,2] [,3] [,4] [,5]
[1,] 0 0 0 0 0
[2,] 0 0 0 0 0
[3,] 0 0 0 0 0
[4,] 0 0 0 0 0
[5,] 0 0 0 0 0
[6,] 0 0 0 0 0
[7,] 0 0 0 0 0
[8,] 0 0 0 0 0
[9,] 0 0 0 0 0
[10,] 0 0 0 0 0
```

```
#Creating a 2-dim matrix using the vector num_vector
fromvector_matrix<-matrix(num_vector, ncol=2, nrow=3)
fromvector_matrix
[,1] [,2]
[1,] 38 105
[2,] 66 2
[3,] 12 17
```

```
#Creating a matrix by joining two vectors (per columns)
bycolumns_matrix<-cbind(num_vector,num_vector1)
bycolumns_matrix
      num_vector num_vector1
[1,]         38          6
[2,]         66          5
[3,]         12          4
[4,]        105          3
[5,]          2          2
[6,]         17          1
```

```
#Creating a matrix by joining two vectors (per rows)
byrows_matrix<-rbind(num_vector,num_vector1)
byrows_matrix
      [,1] [,2] [,3] [,4] [,5] [,6]
num_vector  38  66  12 105  2  17
num_vector1  6   5   4  3  2   1
```

To enter elements in matrices you must specify two positions, the column number and the row number:

```
matrix_name[row_position,column_position] # ALWAYS in that order
```

Examples.

```
fromvector_matrix[2,1]
[1] 66

bycolumns_matrix[4,2]
num_vector1
      3

byrows_matrix[2,3]
num_vector1
      4
```

NOTE. Arithmetic operations can also be applied on matrices. [See More](#)

Data frames

Data frames are collections of elements of any type, i.e. numbers, character, logical values, vectors, matrices, even other data frames.

To create a data frame you could use the command `data.frame()`

```
fromvectors_dataframe<-
data.frame(num_vector,num_vector1,cond_vector,char_vector)
fromvectors_dataframe
  num_vector num_vector1 cond_vector char_vector
1         38          6         TRUE          q
2         66          5         TRUE          z
3         12          4        FALSE          p
4        105          3         TRUE          a
5          2          2        FALSE          x
6         17          1        FALSE         mp
```

It is also possible to upload a table as data frame in R, in this example we will use the file "", which you created in the previous session. To perform this step we are using the function `read.table()`, too visualize the first columns of the table you could use the R command `head()`.

```
geneExpression<-read.table("./RawCounts_phiAb11510.txt",h=T)
head(geneExpression)
      GeneId C28a C28b C37a C37b
1 phi-Ab11510_11551.fna_00009    38    75    48    75
2 phi-Ab11510_11551.fna_00008  1039   881   538   640
3 phi-Ab11510_11551.fna_00007   949   877   891  1025
4 phi-Ab11510_11551.fna_00006   353   274   424   476
5 phi-Ab11510_11551.fna_00005   280   266   376   421
6 phi-Ab11510_11551.fna_00004   543   553   673   821

#Converting the table from list to data.frame
geneExpression<- data.frame(geneExpression)
```

You could access data frames the same way as matrices.

To enter elements in matrices you must specify two positions, the column number and the row number:

```
dataframe_name[row_position,column_position] # ALWAYS in that order
```

Example.

```
geneExpression[5,3]
[1] 266

#Using ranges of positions, you can do this using vectors of positions. (The
following applies to matrices too)

geneExpression[c(1:6),(2:5)]
      C28a C28b C37a C37b
1     38    75    48    75
2  1039   881   538   640
3   949   877   891  1025
4   353   274   424   476
5   280   266   376   421
6   543   553   673   821
```

With data frames you could also perform calculations. For example if you want to compute the correlation between treatments, with the function `cor()` we can calculate this correlation between 2 columns of a data frame, the output of this function is a scalar for 2 vectors, a symmetric matrix instead. See below.

```
#Calculating correlation matrices between C28a and C28b
cor_C28<-cor(geneExpression$C28a,geneExpression$C28b)
cor_C28
[1] 0.9997915

#Calculating correlation matrices between C37a and C37b
cor_C37<-cor(geneExpression$C37a,geneExpression$C37b)
cor_C37
[1] 0.9997988

#You could also calculate the correlation among all treatments (C28a, C28b,
C37a, and C37b)
cor_all<-cor(geneExpression[c(1:6),(2:5)])
cor_all
```

	C28a	C28b	C37a	C37b
C28a	1.0000000	0.9916193	0.8056046	0.8086684
C28b	0.9916193	1.0000000	0.8420575	0.8497559
C37a	0.8056046	0.8420575	1.0000000	0.9976483
C37b	0.8086684	0.8497559	0.9976483	1.0000000

At this point if you use the command `objects()` or `ls()`, you could see all the elements that you have created throughout the workshop development.

Plots

Now, in this section we're going to use the package `ggplot2` (v.3.3.2) to graph some plots. To upload the package in the R session you have to use the following command

```
library(ggplot2)
```

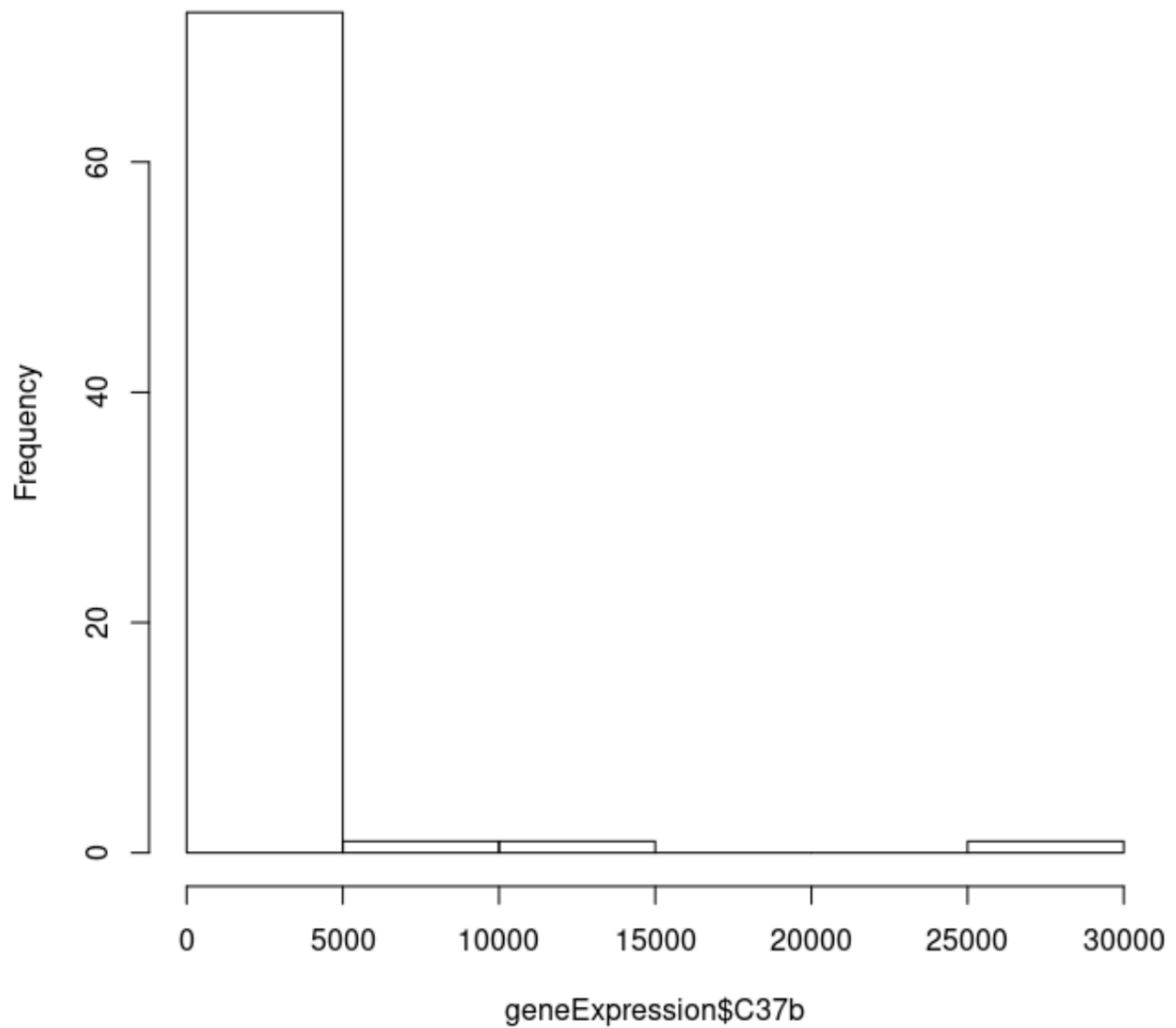
However, we can use R basic functions to create some plots, for example, histograms, boxplots, density plots, and others.

Examples.

We want to see the distribution of gene expression at the C37b condition. To visualize it, we could plot a histogram

```
#To print the image in .png format
hist(geneExpression$C37b)
```

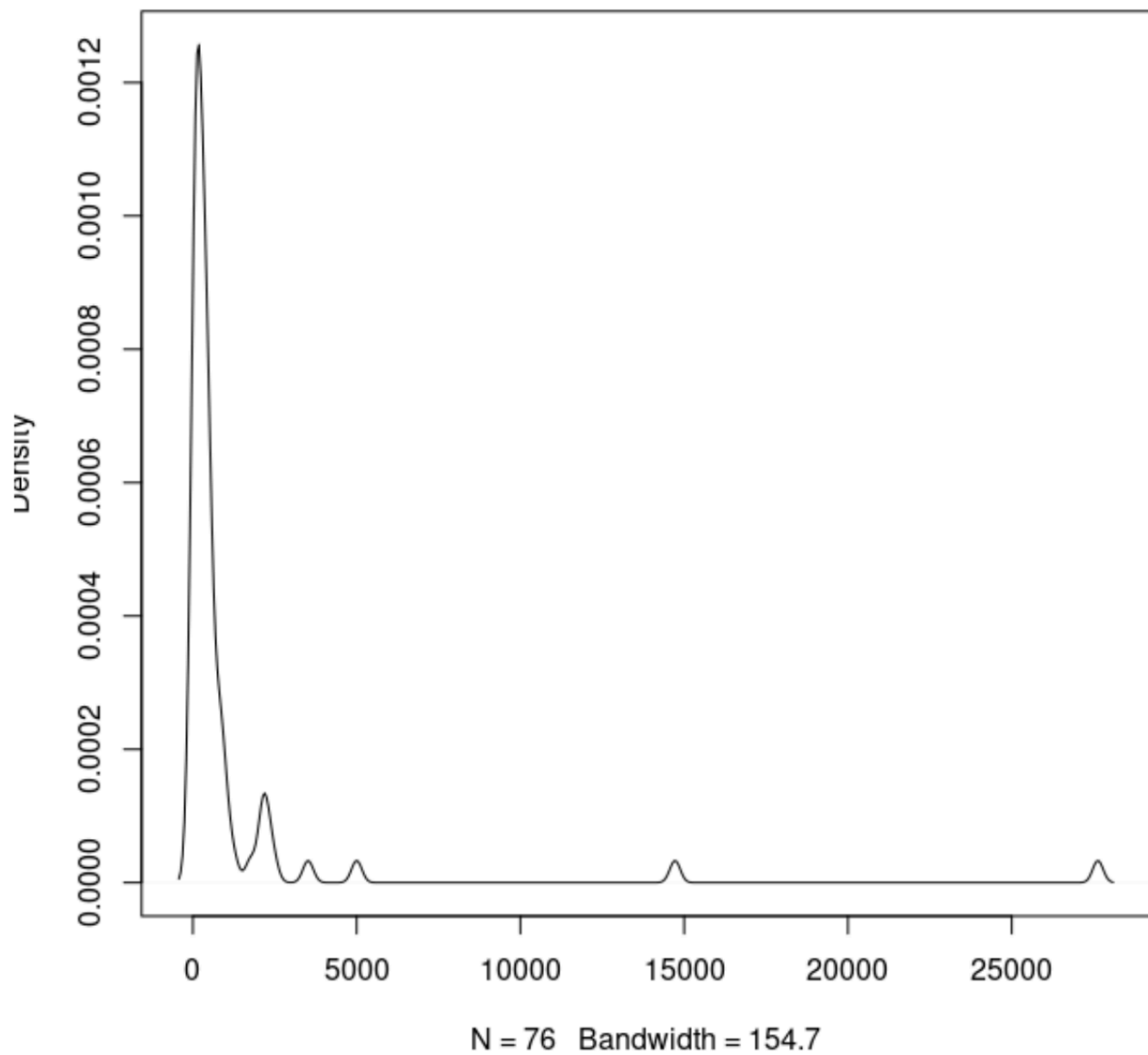
Histogram of geneExpression\$C37b



Now, to obtain the density plot you could use the following commands

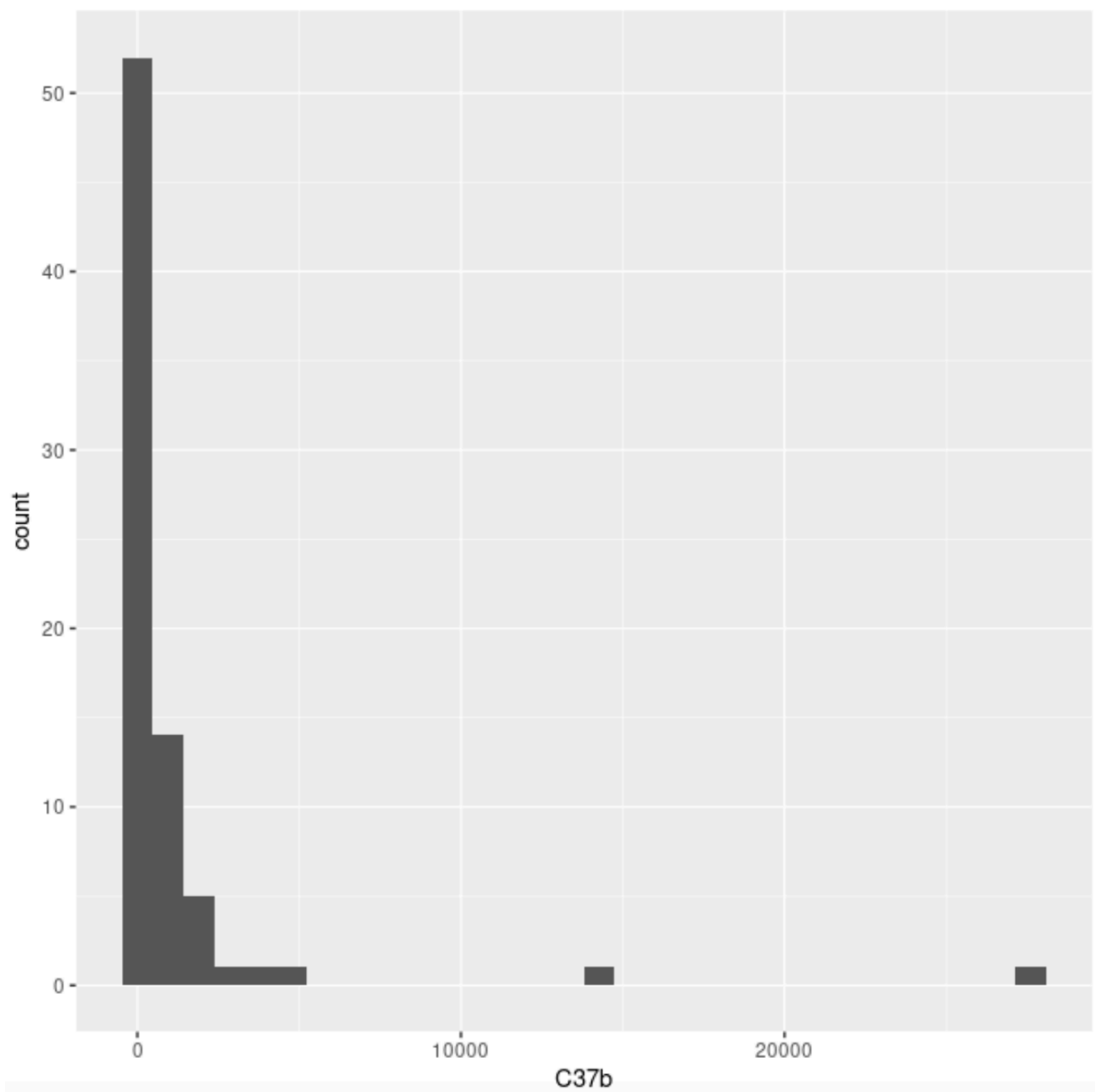
```
plot(density(geneExpression$C37b))
```

density.default(x = geneExpression\$C37b)



To plot the histograms using the function `geom_histogram()` from the ggplot2 package

```
ggplot(geneExpression, aes(x=C37b)) + geom_histogram()
```



You can also compare between different conditions using different colors for each condition. To do this, it is necessary to change the dataset structure using the function `melt()`, as follows. This function is part of the package `reshape2` (v.1.4.4). You must load the library before using the function


```

#Loading library
library(reshape2)

#Change dataset structure
geneExp_melted<-melt(geneExpression)

#Visualize the first lines of the dataset
head(geneExp_melted)

```

	GeneId	variable	value
1	phi-Ab11510_11551.fna_00009	C28a	38
2	phi-Ab11510_11551.fna_00008	C28a	1039
3	phi-Ab11510_11551.fna_00007	C28a	949
4	phi-Ab11510_11551.fna_00006	C28a	353
5	phi-Ab11510_11551.fna_00005	C28a	280
6	phi-Ab11510_11551.fna_00004	C28a	543

```

#Changing column names
names(geneExp_melted) <-c("GeneId","Treatment","RawCounts")

#If you use the function head() to visualize the dataset, you will be able to
observe the variable names change.
head(geneExp_melted)

```

	GeneId	Treatment	RawCounts
1	phi-Ab11510_11551.fna_00009	C28a	38
2	phi-Ab11510_11551.fna_00008	C28a	1039
3	phi-Ab11510_11551.fna_00007	C28a	949
4	phi-Ab11510_11551.fna_00006	C28a	353
5	phi-Ab11510_11551.fna_00005	C28a	280
6	phi-Ab11510_11551.fna_00004	C28a	543

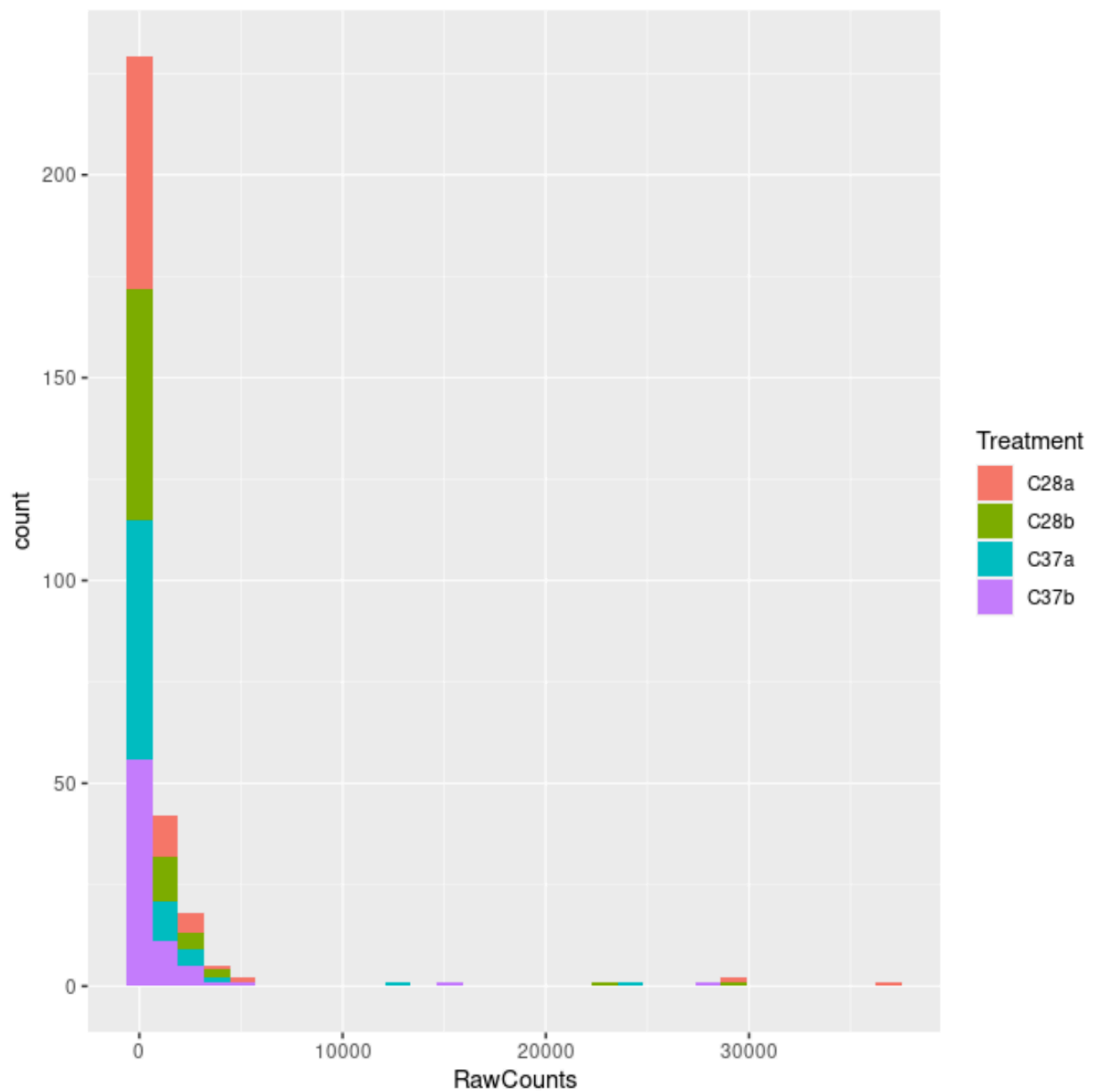
Notice that now, the dataset has changed, the columns have changed. Now each treatment is a value of the column "Treatment" and the counts per gene_id and treatment form the "RawCounts" column of the dataset. Now you could use `ggplot()` to graph the plot comparing the Raw Counts distribution between treatments.

```

#The geometry geom_histogram() just requires the x axis and fill (color)
definition. In this case the x axis are the RawCounts per treatment and to
differentiate each treatment by a color key we set the fill parameter based on
the Treatment column values.

ggplot(geneExp_melted,aes(x=RawCounts,fill=Treatment))+geom_histogram()

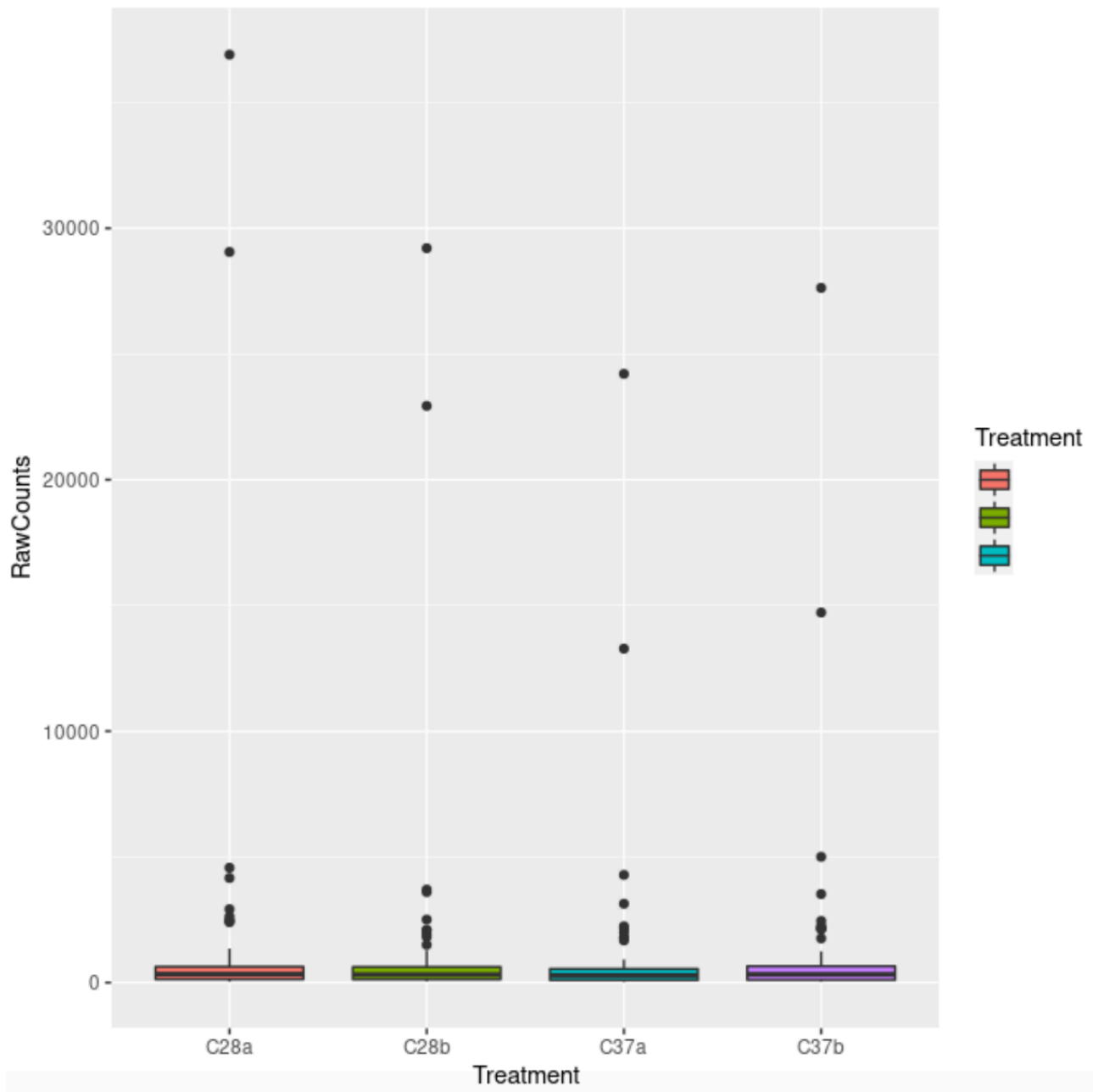
```



Or you could use another geometry, `geom_boxplot()`

#The geometry `geom_boxplot()` requires the x axis, y axis and fill (color) to be defined. In this case the x axis are the Treatments, the y axis are the RawCounts per treatment and to differentiate each treatment by a color key we set the fill parameter based on the Treatment column values.

```
ggplot(geneExp_melted,aes(y=RawCounts,x=Treatment,fill=Treatment))+geom_boxplot(
)
```

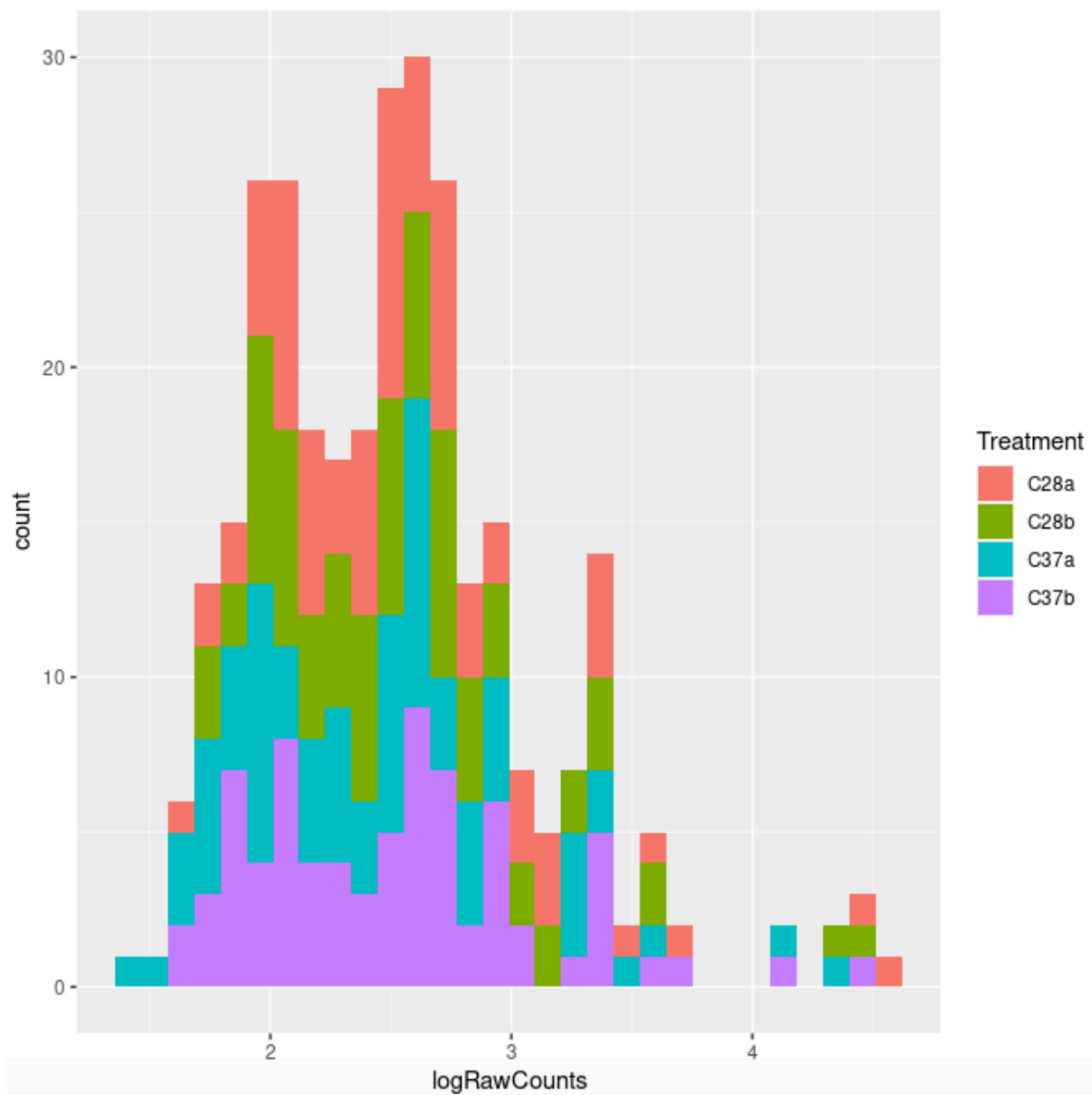


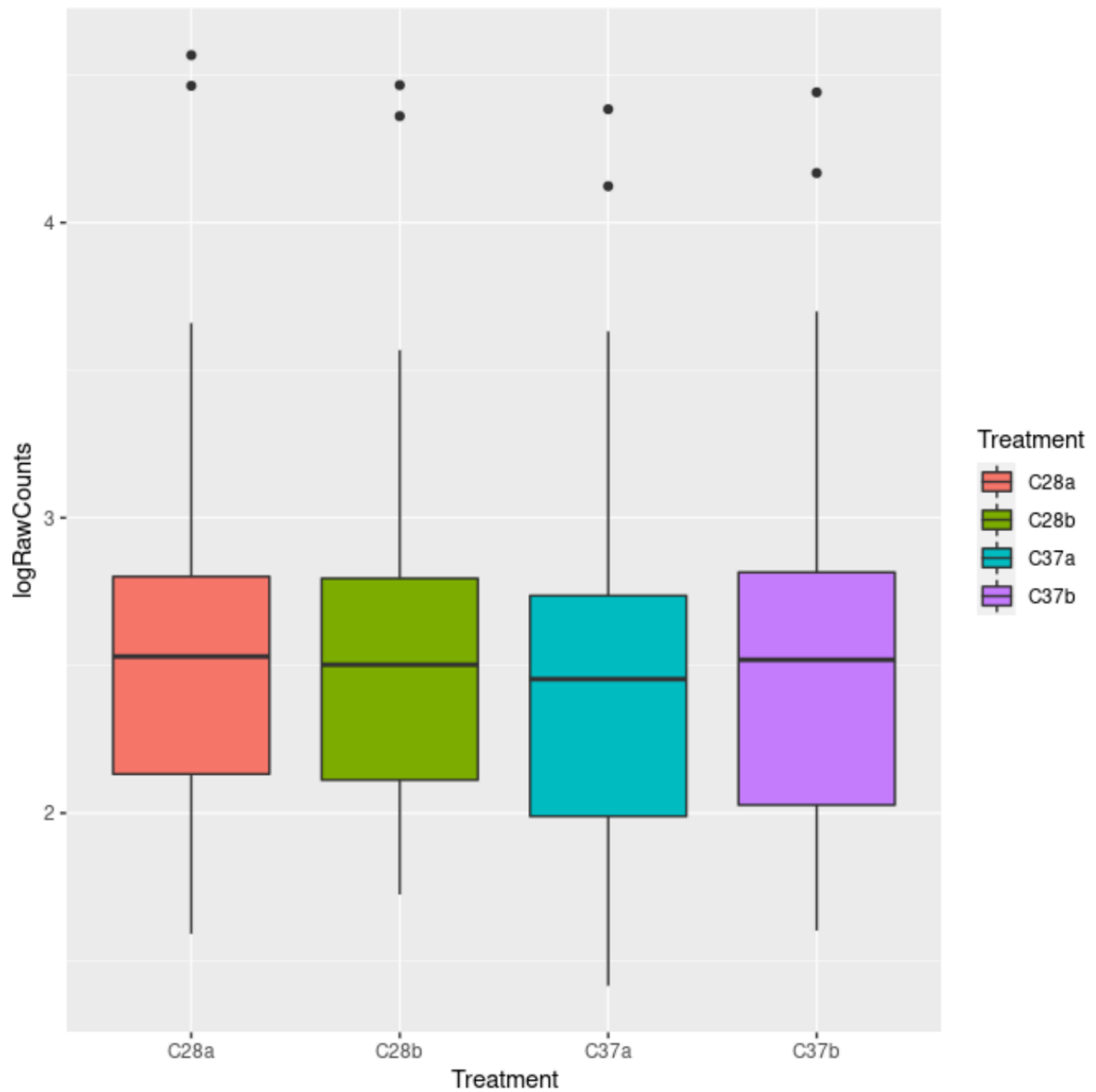
As you can see there are some outliers in the distributions, meaning genes that have a raw count bigger than the mean so it is impossible to see details. To solve this problem we are going to apply `log10()` transformation on RawCounts.

```
#Add a new column to the dataset, this column will be the log10(rowcounts+1)
geneExp_melted$logRowCounts<-log10(geneExp_melted$RawCounts+1)

#New Plots
#histogram
ggplot(geneExp_melted,aes(x=logRowCounts,fill=Treatment))+geom_histogram()
#boxplot
ggplot(geneExp_melted,aes(y=logRowCounts,x=Treatment,fill=Treatment))+geom_boxplot()
```

Now, it is easier to compare between each treatment distribution

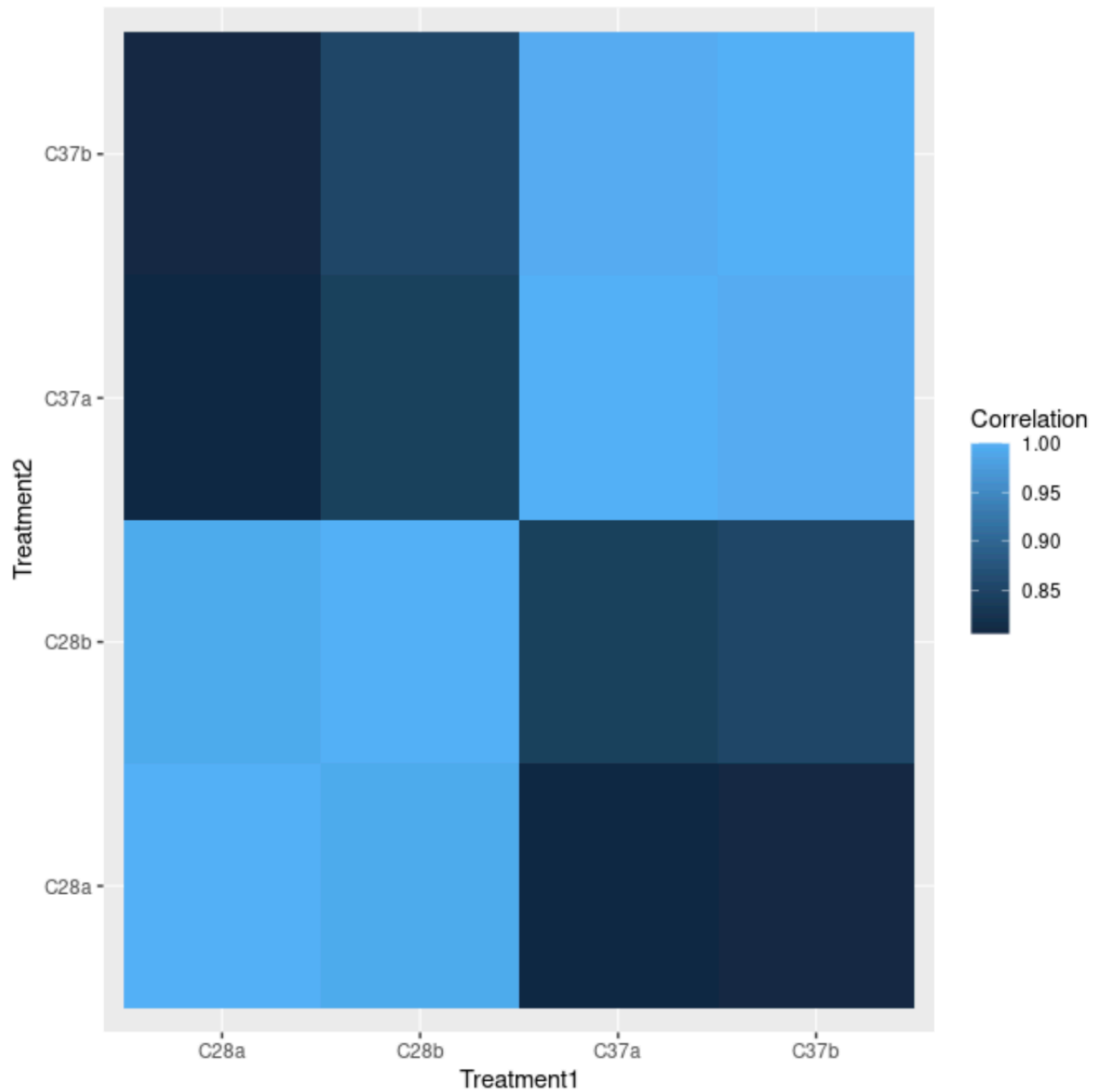




Using the melting strategy you could also obtain a correlation plot, using the correlation matrix computed in the previous section.

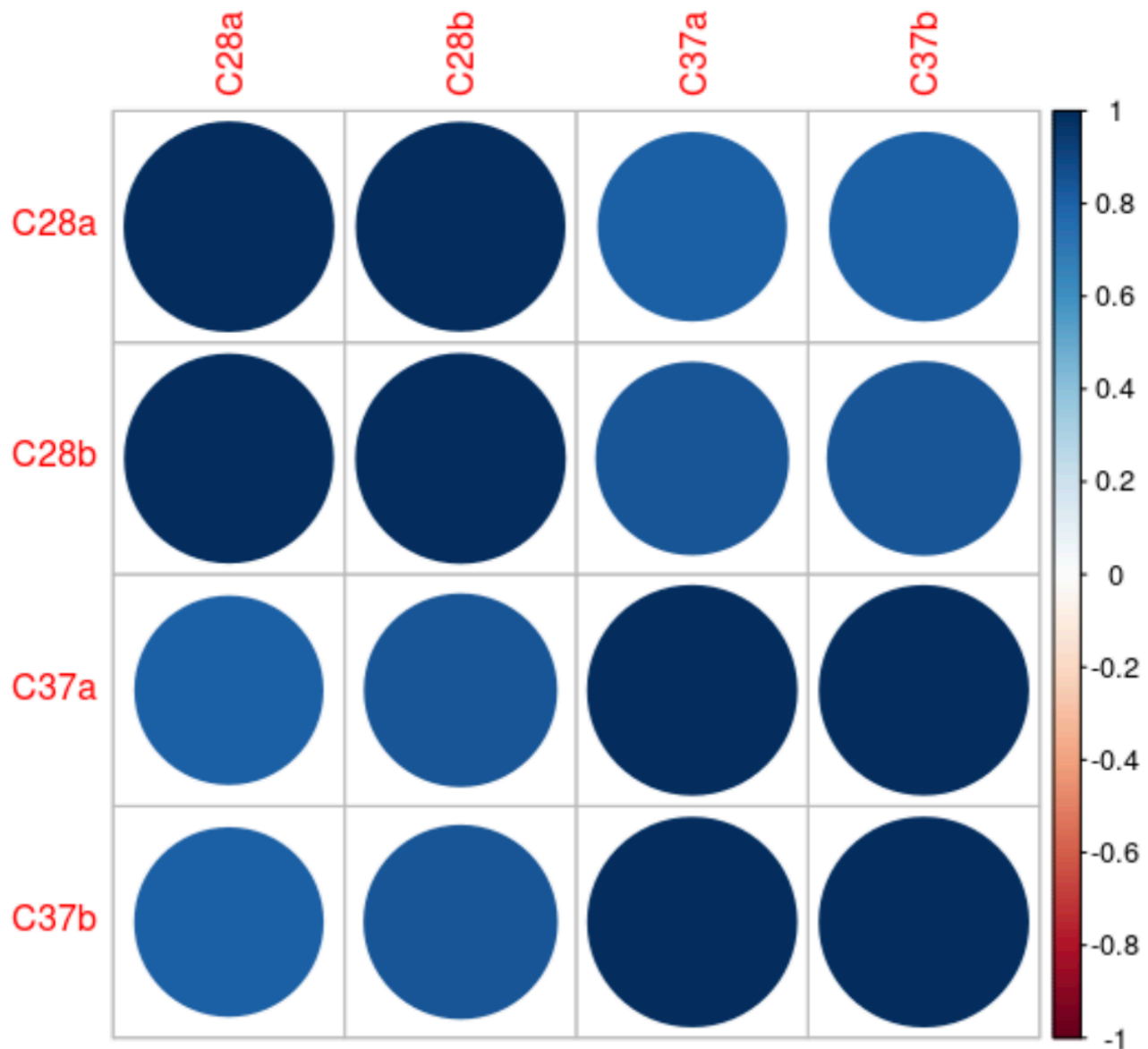
```
#Melting the dataset
cor_melted<-melt(cor_all)
names(cor_melted)<-c("Treatment1","Treatment2","Correlation")

ggplot(cor_melted, aes(x=Treatment1, y= Treatment2, fill= Correlation)) +
  geom_tile()
```



Besides, `corrplot` (v.0.84) is another package that could be used to plot the correlation matrix without need to melt it.

```
corrplot(cor_all)
```



Practice 2.

Create a data frame with all gene expression values. Your data frame should look like:

Gene-Id	C28a	C28b	C37a	C37b
GeneA	200	220	500	545
GeneB	700	740	100	245

Differential gene expression analysis (DESeq2)

Open a terminal and access to 

use the commands `setwd` , `getwd` , `ls` and `list.files`

`getwd` print the path of the working directory.

`setwd` move to an other directory.

`ls` list all the elements created in R.

`list.files` list all the files in the working directory.

```
getwd()
setwd()
ls()
list.files()
```

Use DESeq2 from bioconductor package

(<http://bioconductor.org/packages/release/bioc/html/DESeq2.html>). Load the libraries **DESeq2**, **gplots** and **RColorBrewer**

```
library("DESeq2")
library("gplots")
library("RColorBrewer")
```

With the function `read.csv` read your gene expression table

```
countData <- read.csv("A_B.table.txt", header=T, row.names=1, sep="\t")
```

```
head (countData)
```

Assign the condition labels

```
colData <- DataFrame(condition=factor(c("C28", "C28", "C37", "C37")))

(condition <- factor(c(rep("C28", 2), rep("C37", 2))))
```

Create the expression file

```
dds <- DESeqDataSetFromMatrix(countData=countData, colData=colData,
design=~condition)
```

Run the analysis

```
dds <- DESeq(dds)
```

```
res <- results(dds)
```


You can sort the results using the adjusted *p*value

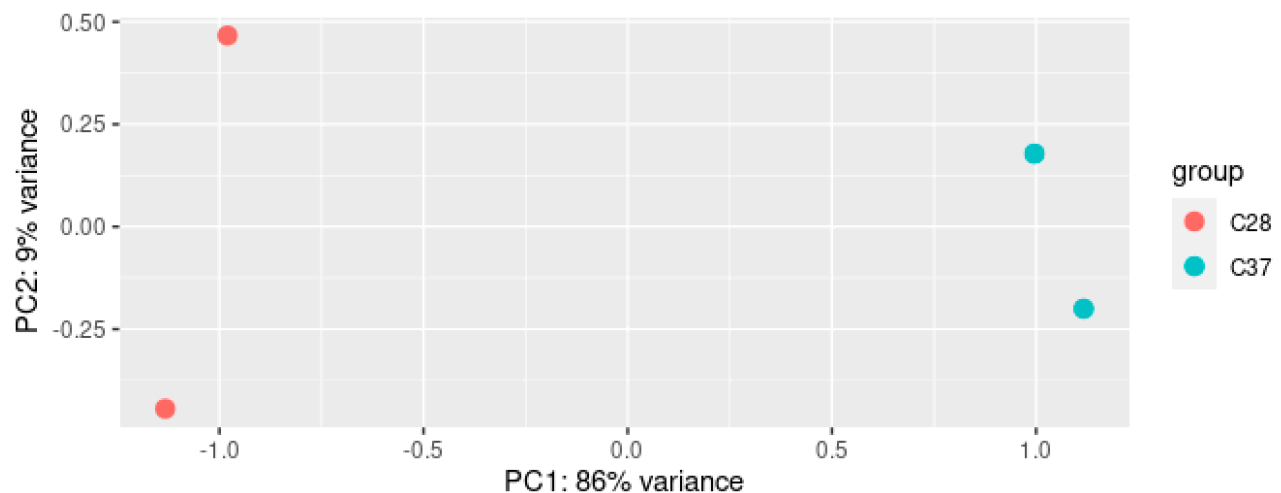
```
res <- res[order(res$padj), ]
resdata <- merge(as.data.frame(res), as.data.frame(counts(dds,
normalized=TRUE))), by="row.names", sort=FALSE)
names(resdata)[1] <- "Gene"
head (resdata)
```

Write and save the results as a .csv file

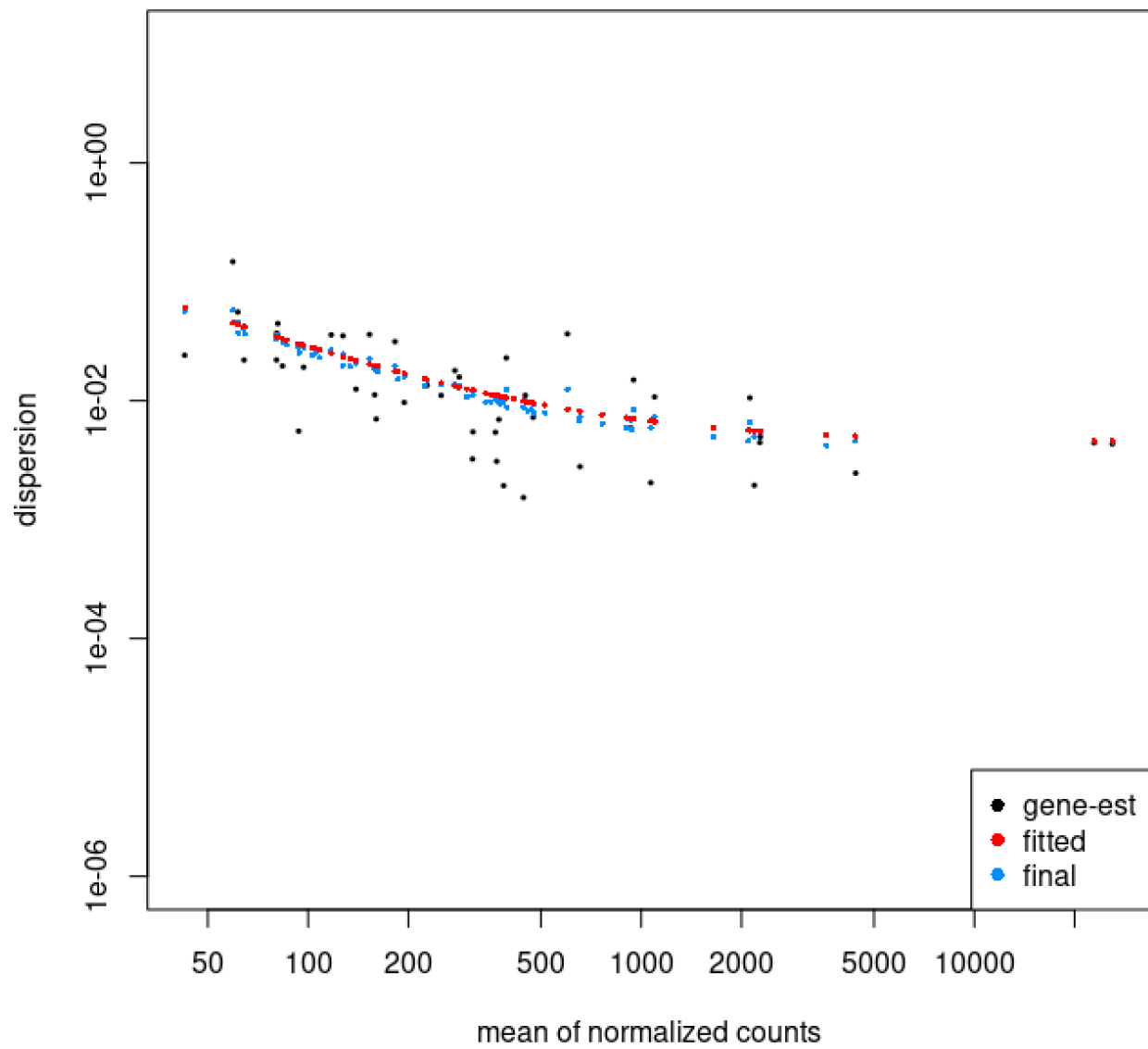
```
write.csv(resdata, file="diffexpr-results-deseq.csv")
```

An important aspect when analyzing the RNA-Seq data lies in knowing the dispersion of the data with respect to the experimental replicates.

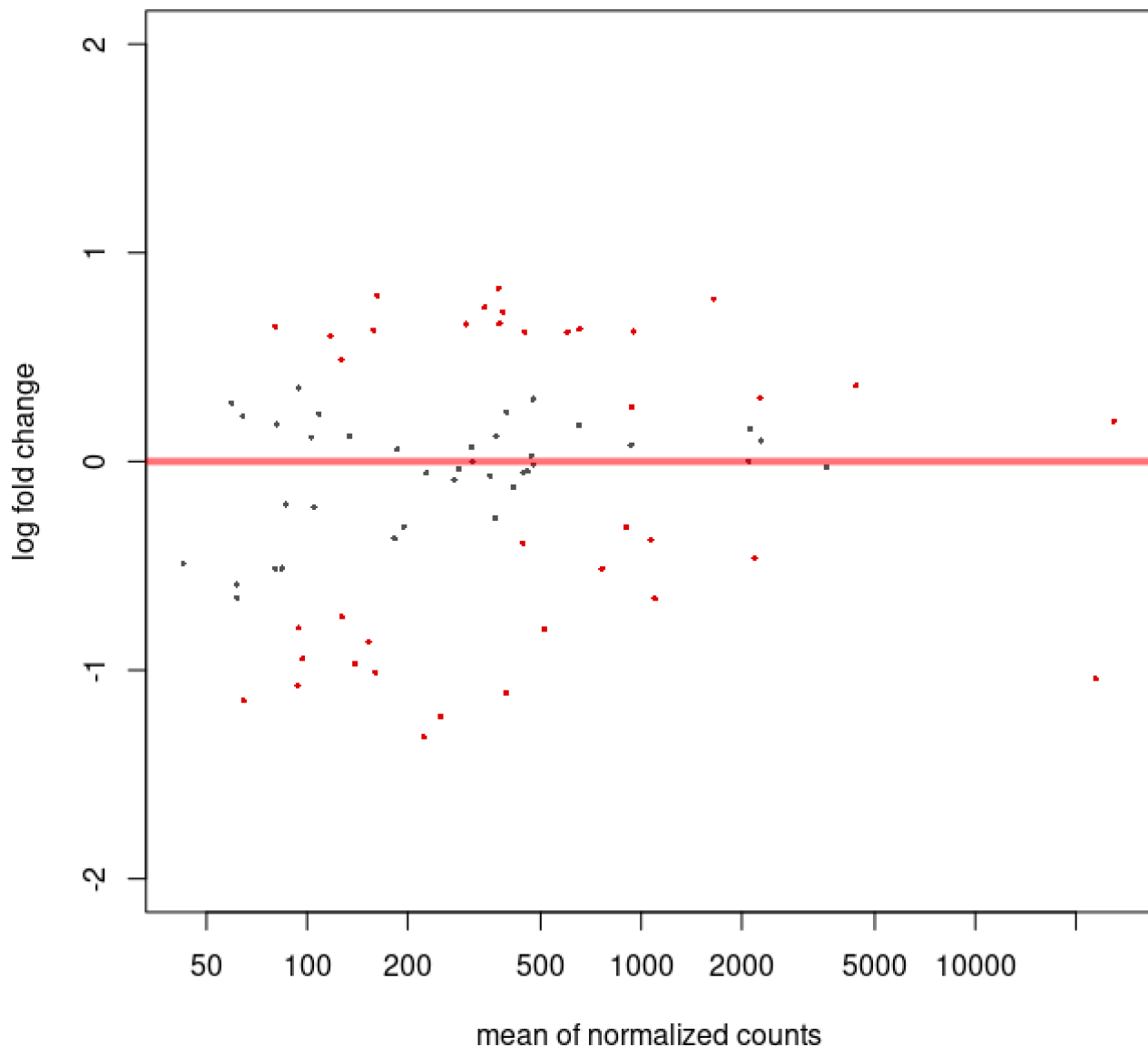
```
rld <- rlogTransformation(dds)
plotPCA(rld, intgroup = "condition")
```



```
plotDispEsts( dds, ylim = c(1e-6, 1e1))
```



```
plotMA(res, ylim=c(-2,2))
```



If you want to consult more approaches to analyzed your data you can go to:

<https://www.bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html>

Differential gene expression analysis (NOISeq)

Load the library

```
library("NOISeq")
```

Read the gene expression table

```
countData <- read.csv("B_A.table.txt", header=T, row.names=1, sep="\t")
head(countData)
```

Create the gene expression set

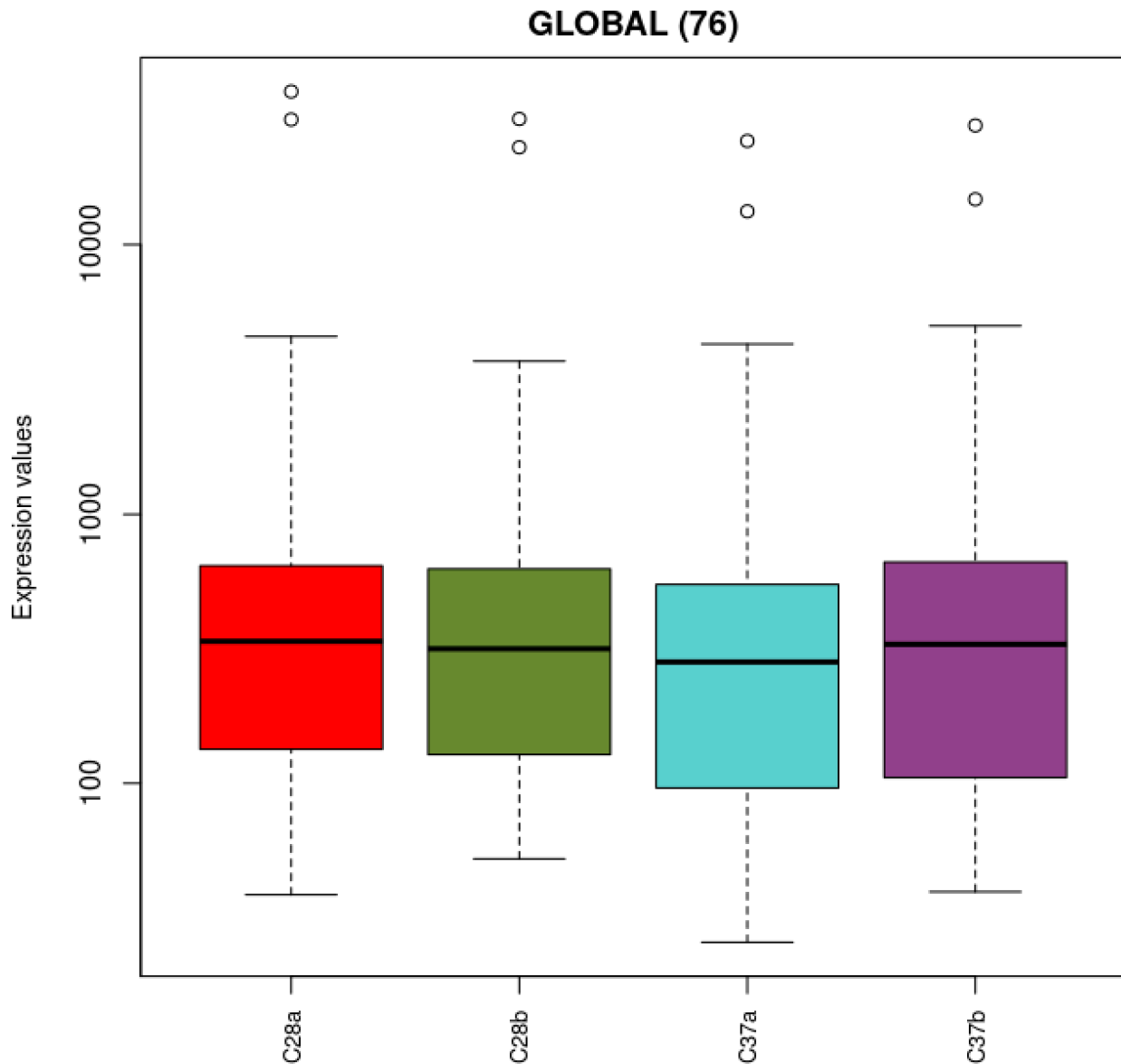
```
myfactors = data.frame(Strain= c("C28", "C28", "C37", "C37"))
mydata <- readData(data = countData, factors = myfactors)
head(assayData(mydata)$exprs)
head(pData(mydata))
```

Estimate the global saturation. **NOTE: This results could be not reliable if you are using the workshop dataset**

```
mysaturation = dat(mydata, k = 0, ndepth = 7, type = "saturation")
explo.plot(mysaturation, toplot = 1, samples = 1:6, yleftlim = NULL, yrightlim = NULL)
```

Plot the gene expression values distribution for each sample

```
mycountsbio = dat(mydata, factor = NULL, type = "countsbio")
explo.plot(mycountsbio, samples = NULL, plottype = "boxplot")
```



Plot the proportion of low expression genes

```
explo.plot(mycountsbio, toplot = 1, samples = NULL, plottype = "barplot")
```

Normalized the data

```
mycd = dat(mydata, type = "cd", norm = FALSE, refColumn = 1)  
explo.plot(mycd)
```

Differential gene expression

```
mynoiseqbio = noiseseqbio(mydata, k=0.5, norm="tmm", factor="Strain", lc=1, r=20,  
adj=1.5, plot=FALSE, a0per=0.9, random.seed=12345, filter=1)
```

Select the DE genes

```
mynoiseq.deg= degenes(mynoiseqbio, q=0.95, M=NULL)
mynoiseq.deg= degenes(mynoiseqbio, q=0.90, M=NULL)
mynoiseq.deg= degenes(mynoiseqbio, q=0.85, M=NULL)
mynoiseq.deg= degenes(mynoiseqbio, q=0.95, M="up")
```

"differentially expressed features (down in first condition)"

```
DE.plot(mynoiseqbio, q = 0.95, graphic = "MD")
```

Write and save the results as a .csv file

```
write.csv(mynoiseq.deg, file="diffexpr-results-noiseq.csv")
```

If you want to consult more approaches to analyzed your data you can go to:

<https://bioconductor.org/packages/devel/bioc/vignettes/NOISeq/inst/doc/NOISeq.pdf>