Working with your RNA-Seq dataset

Princess Rodriguez

11/8/2023

Introduction

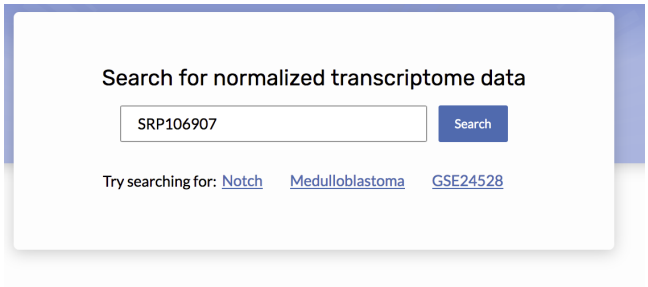
Refine.bio has harmonized 46,324 gene expression datasets from 203 organisms, from many different technologies into one universal repository. As this is **Survey to Bioinformatic Databases**, our goal is to access, process, visualize, and present data analysis to the class.

As a reminder, the dataset has already undergone processing with this following pipeline:

Refine.bio uses Salmon (https://combine-lab.github.io/salmon/) and tximport (https://bioconductor.org/packages/release/bioc/html/tximport.html) to process all RNA-seq data.

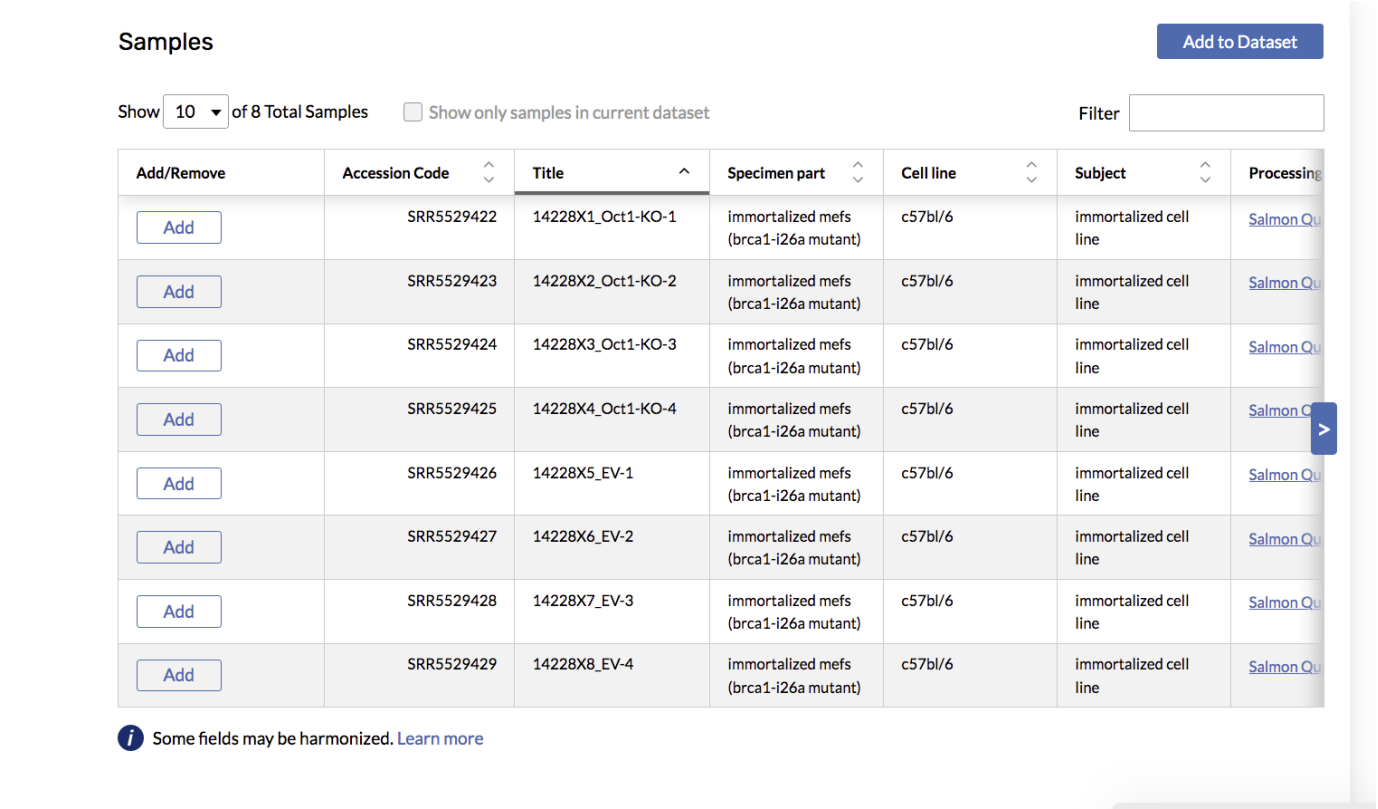
Instructions on downloading a dataset from refine.bio

1. Type the SRP# into refine.bio search engine

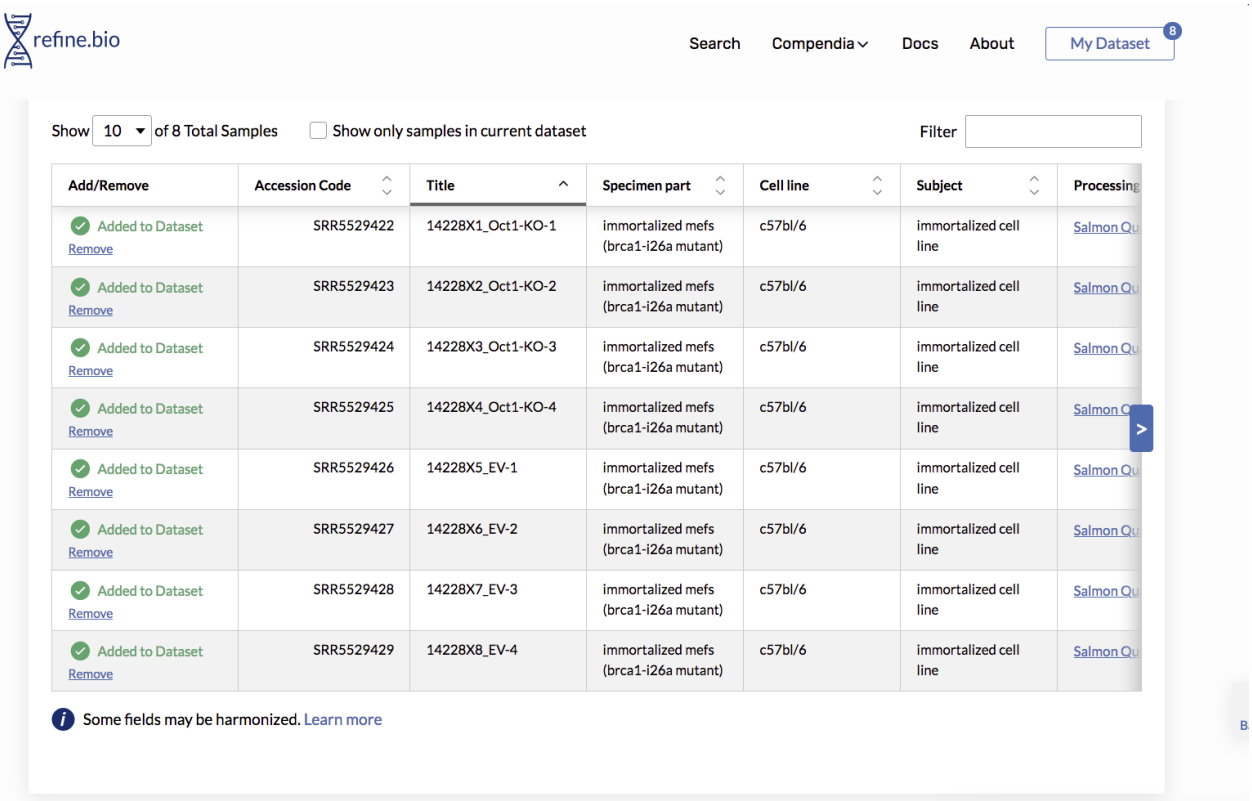


2. Verify that the dataset has been published and has an associated GEO accession number

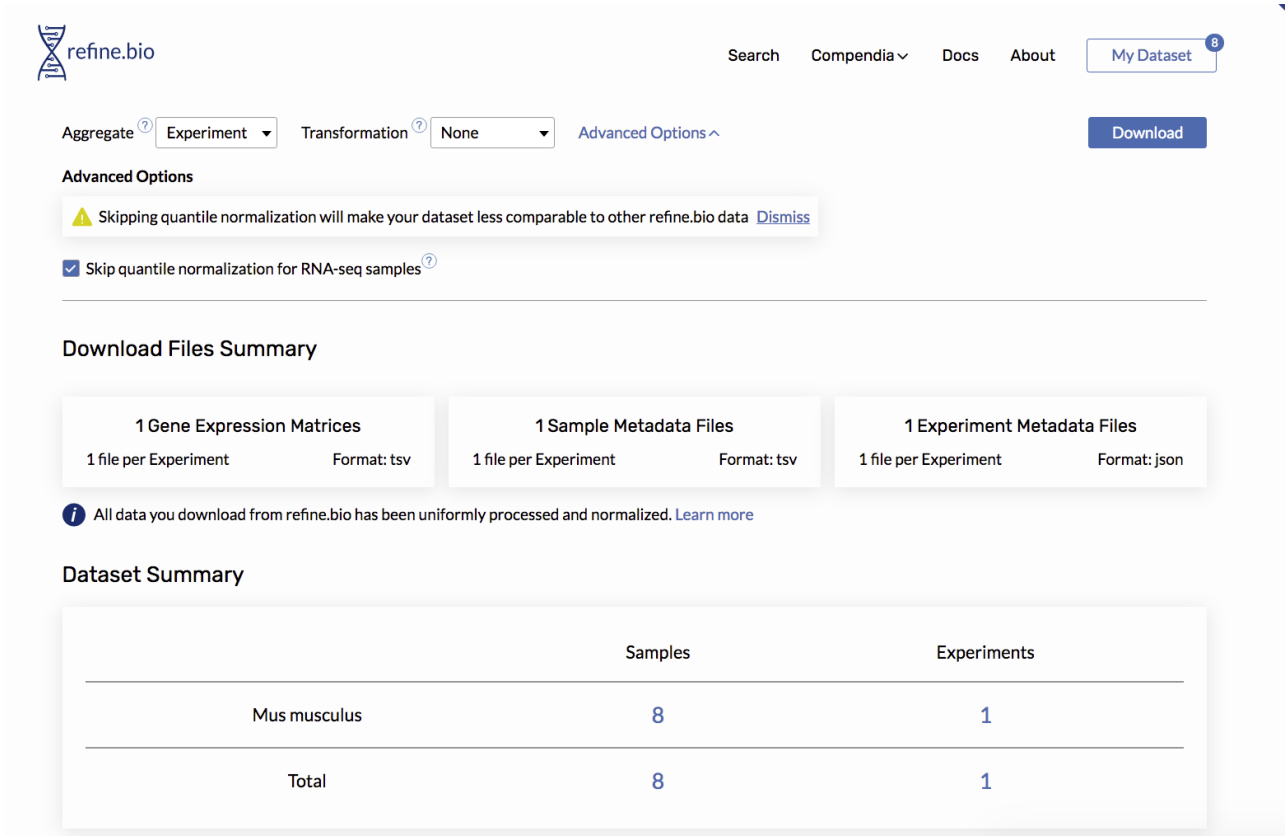
3. Carefully examine the dataset. You will need (3) biological replicates per treatment type as a minimum. If you have any questions, please reach out to your instructor **this is a crucial step**.



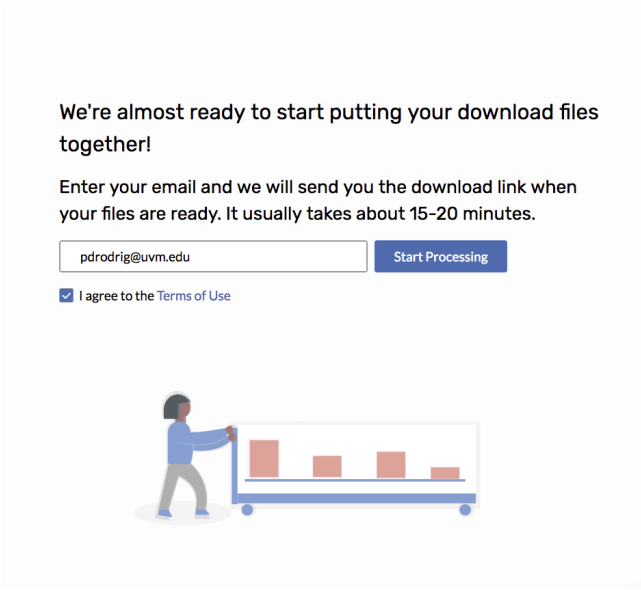
4. Add samples to `My Dataset`.



5. Click on `My Dataset`. **Be sure to SKIP quantile normalization**



6. Click on Download. The following panel will come up. Enter your email and hit start processing.



We will use the outputs from Refine.bio for the downstream analysis presented here.

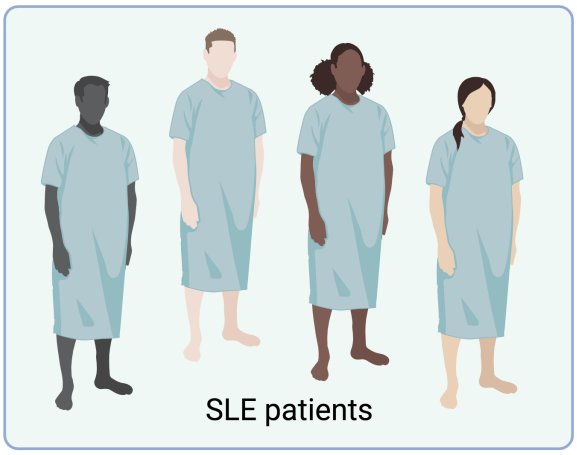
About the Dataset

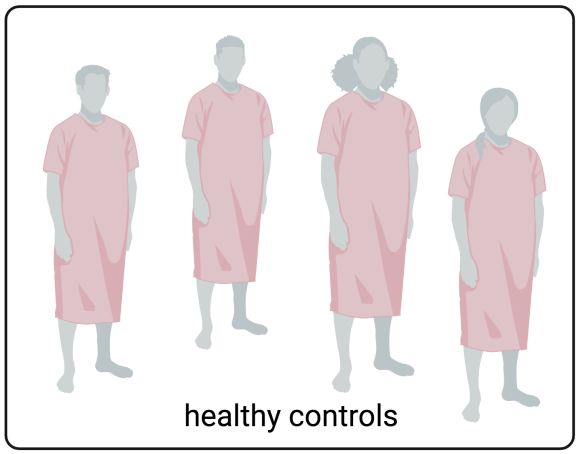
For this example analysis, we will use this systemic lupus erythematosus dataset (https://www.refine.bio/experiments/SRP136102/systemic-lupus-erythematosus-patient-blood-with-controls).

Design: Whole blood was collected in PaxGene tubes from 31 SLE and 29 healthy donors. RNA libraries were prepared for sequencing using standard Illumina protocols. Sequencing was performed on a Illumina HiSeq2000 platform. We will only be working with a subset of the dataset.

Goals:

- We would like to access if this is a “good” or “bad” dataset. We will make these decisions as we proceed through the analysis
- Overall our goal is to compare the transcriptional changes that occur in individuals with SLE versus healthy donors.
- To compare these two groups, we first need to determine how well the “biological replicates” for each group cluster together.





We will view sample clustering in two ways:

1. A dendogram
2. A principal component analysis plot

Data information

The `SRP136102` folder should contain the following:

- The `setting-up-with-refine-bio-data-edit.Rmd`
- The gene expression `SRP136102.tsv`
- The metadata TSV `metadata_SRP136102.tsv`

What is Metadata? Metadata is defined as the information that describes and explains data. It provides context with details such as the source, treatment, and relationships to other data sets. So, it can help you understand the relevance of a particular data set and guide you on how to use it.

Reading data into R

Regardless of the specific analysis in R we are performing, we usually need to bring data in for the analysis. The function in R we use will depend on the type of data file we are bringing in (e.g. text, Stata, SPSS, SAS, Excel, etc.) and how the data in that file are separated, or delimited. The table below lists functions that can be used to import data from common file formats.

Data type	Extension	Function	Package
Comma separated values	csv	`read.csv()`	utils (default)
		`read_csv()`	readr (tidyverse)

Data type	Extension	Function	Package
Tab separated values	tsv	`read_tsv()`	readr
Other delimited formats	txt	`read.table()`	utils
		`read_table()`	readr
		`read_delim()`	readr
Stata version 13-14	dta	`readdta()`	haven
Stata version 7-12	dta	`read.dta()`	foreign
SPSS	sav	`read.spss()`	foreign
SAS	sas7bdat	`read.sas7bdat()`	sas7bdat
Excel	xlsx, xls	`read_excel()`	readxl (tidyverse)

For example, if we have text file separated by commas (comma-separated values), we could use the function `read.csv`. However, if the data are separated by a different delimiter in a text file, we could use the generic `read.table` function and specify the delimiter as an argument in the function.

When working with genomic data, we often have a metadata file containing information on each sample in our dataset. Let's bring in the metadata file using the `read.delim` function. Check the arguments for the function to get an idea of the function options:

```
?read.delim
```

The `read.delim` function has *one required argument* (the file) and several *options* that can be specified.

Bringing in the `metadata_SRP136102.tsv` file

```
meta <- read.table(file = "metadata_SRP136102.tsv",
                   sep = "\t",
                   stringsAsFactors = FALSE,
                   header = TRUE)

#meta
```

Generation of "clean" sample annotation file

We are going to take a few moments to generate the sample annotation file using the `metadata_SRP136102.tsv`. By creating a "clean" metafile it will make for more easily, interpretable data plots later.

```
# select columns of interest
meta <- meta[, c("refinebio_accession_code", "refinebio_sex", "refinebio_subject")]
#meta
```

Dataframes (and matrices) have 2 dimensions (rows and columns), so if we want to select some specific data from it we need to specify the "coordinates" we want from it. We use the square bracket notation `[ ]` and provide *two indexes required*. Within the square bracket, **row numbers come first followed by column numbers (and the two are separated by a comma)**.

```
#rename columns
colnames(meta) <- c('Sample', 'Sex', 'Treatment')

#meta
```

Now the problem is that all treatment groups will be read as distinct in R because each sample has a unique number after the `_`. So sle_01 will be read as a *different* group than sle_02.

```
# adding a column
meta$Replicate <- c(1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6)
meta
```

```
##        Sample    Sex Treatment Replicate
## 1  SRR6870284 female    sle_01         1
## 2  SRR6870286 female    sle_02         2
## 3  SRR6870292 female    sle_07         3
## 4  SRR6870298 female    sle_10         4
## 5  SRR6870302 female    sle_13         5
## 6  SRR6870304 female    sle_14         6
## 7  SRR6870356 female   norm_06         1
## 8  SRR6870358 female   norm_07         2
## 9  SRR6870360 female   norm_08         3
## 10 SRR6870364 female   norm_11         4
## 11 SRR6870366 female   norm_13         5
## 12 SRR6870370   male   norm_15         6
```

```
# The `$` allows you to select a single column by name.
```

Class Exercise

Recreate the `meta` sample annotation file so that the first 6 rows are assigned to sle and the final 6 rows are assigned to normal.

```
# answer key
meta <- meta[, c("Sample", "Sex")]
meta$Treatment <- c("sle", "sle", "sle", "sle", "sle", "sle", "normal", "normal", "normal", "normal", "nor
mal", "normal")
meta$Replicate <- c(1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6)
meta
```

```
##        Sample    Sex Treatment Replicate
## 1  SRR6870284 female       sle         1
## 2  SRR6870286 female       sle         2
## 3  SRR6870292 female       sle         3
## 4  SRR6870298 female       sle         4
## 5  SRR6870302 female       sle         5
## 6  SRR6870304 female       sle         6
## 7  SRR6870356 female    normal         1
## 8  SRR6870358 female    normal         2
## 9  SRR6870360 female    normal         3
## 10 SRR6870364 female    normal         4
## 11 SRR6870366 female    normal         5
## 12 SRR6870370   male    normal         6
```

Designation of Factors

We discussed that a factor in R is a variable used to categorize and store the data. When stored as factor, the data is now being stored as a vector of integer values. This is useful during data analysis as we will see below.

Let's use the `str` function to look at our data now.

```
#structure function
str(meta)
```

```
## 'data.frame':    12 obs. of  4 variables:
##  $ Sample   : chr  "SRR6870284" "SRR6870286" "SRR6870292" "SRR6870298" ...
##  $ Sex      : chr  "female" "female" "female" "female" ...
##  $ Treatment: chr  "sle" "sle" "sle" "sle" ...
##  $ Replicate: num  1 2 3 4 5 6 1 2 3 4 ...
```

Now we will designate each of the variables we want to convert into a factor variable.

```
#designate as factor
meta$Treatment <- as.factor(meta$Treatment)
meta$Replicate <- as.factor(meta$Replicate)
meta$Sex <- as.factor(meta$Sex)
```

Let's take at the data again:

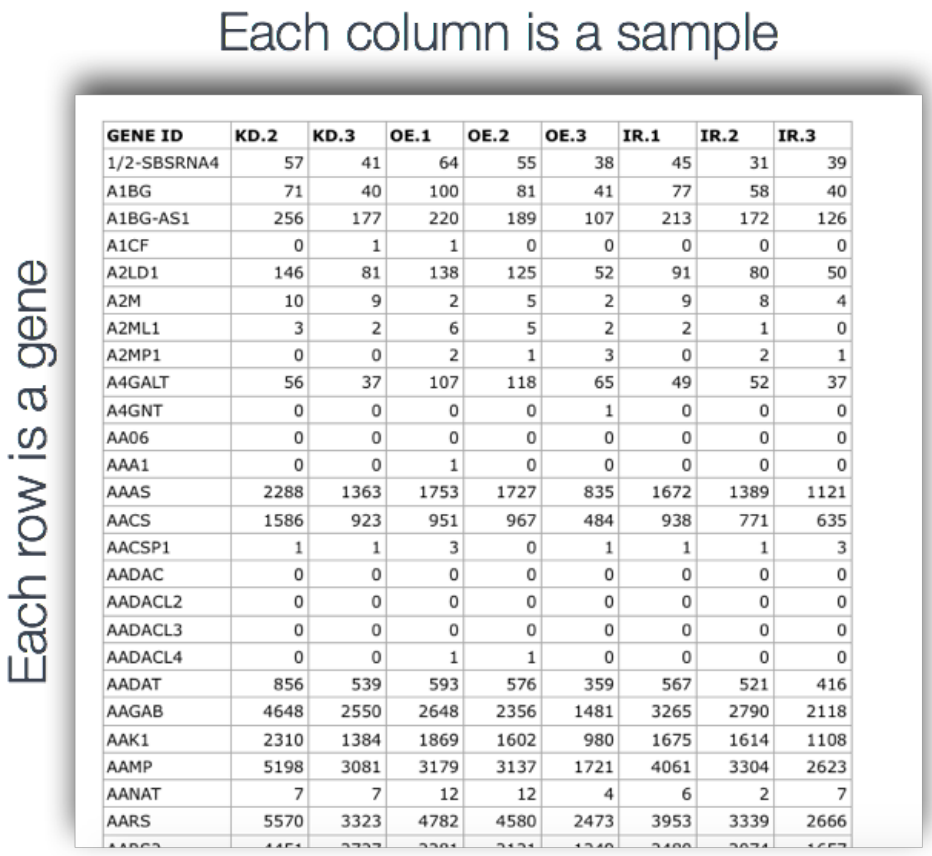
```
# check structure again
str(meta)
```

```
## 'data.frame':    12 obs. of  4 variables:
##  $ Sample   : chr  "SRR6870284" "SRR6870286" "SRR6870292" "SRR6870298" ...
##  $ Sex      : Factor w/ 2 levels "female","male": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Treatment: Factor w/ 2 levels "normal","sle": 2 2 2 2 2 2 1 1 1 1 ...
##  $ Replicate: Factor w/ 6 levels "1","2","3","4",..: 1 2 3 4 5 6 1 2 3 4 ...
```

Counts Matrix

So far, we've only added in our sample annotation file to the environment and told R/RStudio how we would like it to read the varying columns.

Now, let's tell RStudio to read in our counts matrix. This is the actual data output from an an RNA-seq experiment. After reads have been aligned to the genome, the next step is to *count* how many reads have mapped to each gene. A "raw" counts matrix is used in statistical programs for differential gene expression analysis. Below is the anatomy of a counts matrix.



GENE ID	KD.2	KD.3	OE.1	OE.2	OE.3	IR.1	IR.2	IR.3
1/2-SBSRNA4	57	41	64	55	38	45	31	39
A1BG	71	40	100	81	41	77	58	40
A1BG-AS1	256	177	220	189	107	213	172	126
A1CF	0	1	1	0	0	0	0	0
A2LD1	146	81	138	125	52	91	80	50
A2M	10	9	2	5	2	9	8	4
A2ML1	3	2	6	5	2	2	1	0
A2MP1	0	0	2	1	3	0	2	1
A4GALT	56	37	107	118	65	49	52	37
A4GNT	0	0	0	0	1	0	0	0
AA06	0	0	0	0	0	0	0	0
AAA1	0	0	1	0	0	0	0	0
AAAS	2288	1363	1753	1727	835	1672	1389	1121
AACS	1586	923	951	967	484	938	771	635
AACSP1	1	1	3	0	1	1	1	3
AADAC	0	0	0	0	0	0	0	0
AADACL2	0	0	0	0	0	0	0	0
AADACL3	0	0	0	0	0	0	0	0
AADACL4	0	0	1	1	0	0	0	0
AADAT	856	539	593	576	359	567	521	416
AAGAB	4648	2550	2648	2356	1481	3265	2790	2118
AAK1	2310	1384	1869	1602	980	1675	1614	1108
AAMP	5198	3081	3179	3137	1721	4061	3304	2623
AANAT	7	7	12	12	4	6	2	7
AARS	5570	3323	4782	4580	2473	3953	3339	2666

```
# read-in counts matrix
data <- read.table(file = "SRP136102.tsv", sep = "\t", stringsAsFactors = FALSE, header = TRUE, row.names
= 1)
head(data)
```

```
##                 SRR6870284 SRR6870286 SRR6870292 SRR6870298 SRR6870302
## ENSG00000000003   46.39945   24.51837    37.5936   44.98844   34.44015
## ENSG00000000005    0.00000    0.00000     0.0000    0.00000    0.00000
## ENSG00000000419  229.43710  213.17186   206.5260  145.13773  193.27979
## ENSG00000000457  529.40980  441.44977   401.6713  581.02700  472.23468
## ENSG00000000460   84.78243   78.22288   120.5951   95.55912  120.39508
## ENSG00000000938 3811.94040 3944.22310  6929.4775 2112.83840 4020.41260
##                 SRR6870304 SRR6870356 SRR6870358 SRR6870360 SRR6870364
## ENSG00000000003   39.04514   19.49657   37.34583   24.32008    15.3744
## ENSG00000000005    0.00000    0.00000    0.00000    0.00000     0.0000
## ENSG00000000419  237.87276  166.87498  142.88065  202.96245   204.9115
## ENSG00000000457  651.12090  405.19418  539.30914  605.55975   351.6534
## ENSG00000000460   77.85001   83.13609  129.71800  141.67328   100.7051
## ENSG00000000938 3335.41140 3803.87100 4677.96500 3465.53780  4443.0034
##                 SRR6870366  SRR6870370
## ENSG00000000003   17.38905   26.844183
## ENSG00000000005    0.00000    2.720015
## ENSG00000000419  103.15216  237.559280
## ENSG00000000457  265.86420  481.142120
## ENSG00000000460   50.99625  136.560960
## ENSG00000000938 1339.46720 3685.132600
```

Let's check that our annotation file is accurate

```
# check identical
identical(colnames(data), meta$Sample)
```

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

We just asked are the column names in the data frame "data" identical to the meta column Sample. The answer is Yes!

Changing column names

Right now, we can all agree that the column names of the `data` counts matrix are not helpful in identifying which samples are which.

```
# change column names
meta$ColNames <- paste(meta$Treatment, meta$Replicate, sep = "_")
colnames(data) <- meta$ColNames
colnames(data)
```

```
##  [1] "sle_1"    "sle_2"    "sle_3"    "sle_4"    "sle_5"    "sle_6"
##  [7] "normal_1" "normal_2" "normal_3" "normal_4" "normal_5" "normal_6"
```

```
head(data)
```

```
##                     sle_1      sle_2     sle_3      sle_4      sle_5
## ENSG00000000003   46.39945   24.51837   37.5936   44.98844   34.44015
## ENSG00000000005    0.00000    0.00000    0.0000    0.00000    0.00000
## ENSG00000000419  229.43710  213.17186  206.5260  145.13773  193.27979
## ENSG00000000457  529.40980  441.44977  401.6713  581.02700  472.23468
## ENSG00000000460   84.78243   78.22288  120.5951   95.55912  120.39508
## ENSG00000000938 3811.94040 3944.22310 6929.4775 2112.83840 4020.41260
##                     sle_6   normal_1   normal_2   normal_3  normal_4
## ENSG00000000003   39.04514   19.49657   37.34583   24.32008   15.3744
## ENSG00000000005    0.00000    0.00000    0.00000    0.00000    0.0000
## ENSG00000000419  237.87276  166.87498  142.88065  202.96245  204.9115
## ENSG00000000457  651.12090  405.19418  539.30914  605.55975  351.6534
## ENSG00000000460   77.85001   83.13609  129.71800  141.67328  100.7051
## ENSG00000000938 3335.41140 3803.87100 4677.96500 3465.53780 4443.0034
##                   normal_5    normal_6
## ENSG00000000003   17.38905   26.844183
## ENSG00000000005    0.00000    2.720015
## ENSG00000000419  103.15216  237.559280
## ENSG00000000457  265.86420  481.142120
## ENSG00000000460   50.99625  136.560960
## ENSG00000000938 1339.46720 3685.132600
```

Let's look at the dimensions of 'data'

```
#dimensions
dim(data)
```

```
## [1] 43363    12
```

```
summary(data)
```

```
##      sle_1              sle_2              sle_3              sle_4
##  Min.   :     0.0   Min.   :     0.0   Min.   :     0.0   Min.   :     0.0
##  1st Qu.:     0.0   1st Qu.:     0.0   1st Qu.:     0.0   1st Qu.:     0.0
##  Median :     2.6   Median :     2.2   Median :     2.0   Median :     2.8
##  Mean   :   209.4   Mean   :   226.3   Mean   :   273.9   Mean   :   187.5
##  3rd Qu.:    53.4   3rd Qu.:    49.8   3rd Qu.:    46.3   3rd Qu.:    55.4
##  Max.   :473185.4   Max.   :796560.2   Max.   :912637.1   Max.   :472599.8
##      sle_5              sle_6             normal_1           normal_2
##  Min.   :     0.0   Min.   :     0.0   Min.   :      0.0   Min.   :     0.0
##  1st Qu.:     0.0   1st Qu.:     0.0   1st Qu.:      0.0   1st Qu.:     0.0
##  Median :     2.1   Median :     2.8   Median :      2.1   Median :     3.1
##  Mean   :   192.1   Mean   :   212.3   Mean   :    249.7   Mean   :   221.7
##  3rd Qu.:    46.2   3rd Qu.:    58.4   3rd Qu.:     51.1   3rd Qu.:    66.5
##  Max.   :491363.3   Max.   :340429.9   Max.   :1647740.4   Max.   :462483.7
##     normal_3           normal_4           normal_5           normal_6
##  Min.   :     0.0   Min.   :     0.0   Min.   :     0.0   Min.   :     0.0
##  1st Qu.:     0.0   1st Qu.:     0.0   1st Qu.:     0.0   1st Qu.:     0.0
##  Median :     2.9   Median :     2.2   Median :     1.1   Median :     2.8
##  Mean   :   237.5   Mean   :   202.6   Mean   :   145.0   Mean   :   214.5
##  3rd Qu.:    68.1   3rd Qu.:    51.0   3rd Qu.:    28.8   3rd Qu.:    60.6
##  Max.   :691516.5   Max.   :900123.2   Max.   :950292.4   Max.   :671115.1
```

Checking the quality of the dataset

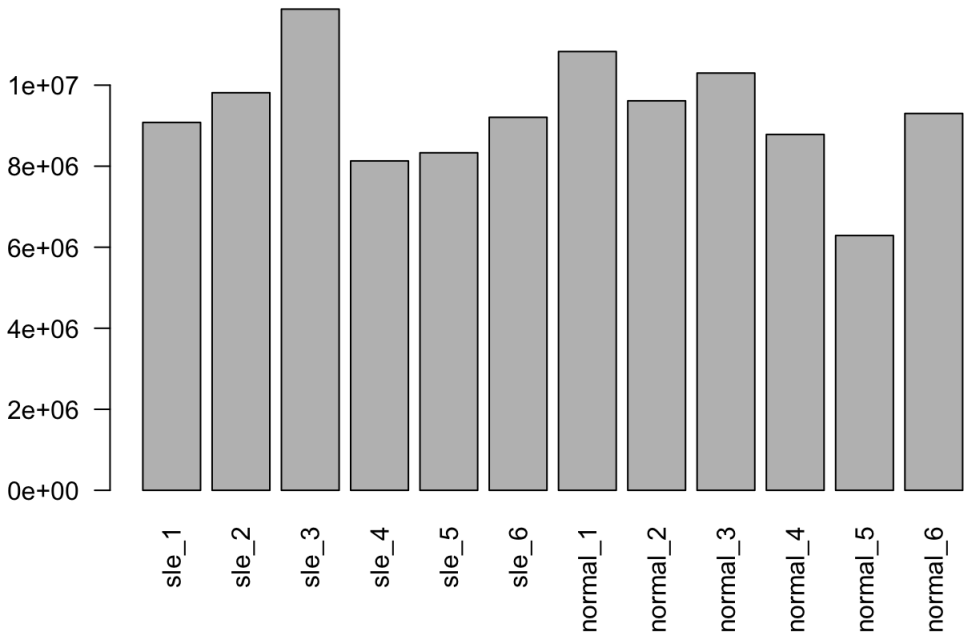
Question 1: How many total reads did each sample receive? For gene expression profiling experiments, we hope for 10 million - 25 million reads per sample.

	SLE1	SLE2	SLE3	SLE4	SLE5	SLE6	NOR1	NOR2	NOR3	NOR4	NOR5	NOR6
Gene	SRR6870284	SRR6870286	SRR6870292	SRR6870298	SRR6870302	SRR6870304	SRR6870356	SRR6870358	SRR6870360	SRR6870364	SRR6870366	SRR6870370
ENSG00000	46.39945	24.518372	37.593605	44.98844	34.440155	39.04514	19.496567	37.345833	24.32008	15.374405	17.389055	26.844183
ENSG00000	0	0	0	0	0	0	0	0	0	0	0	2.720015
ENSG00000	229.4371	213.17186	206.52602	145.13773	193.27979	237.87276	166.87498	142.88065	202.96245	204.91154	103.15216	237.55928
ENSG00000	529.4098	441.44977	401.67133	581.027	472.23468	651.1209	405.19418	539.30914	605.55975	351.65338	265.8642	481.14212
ENSG00000	84.782425	78.22288	120.59514	95.55912	120.39508	77.850006	83.136086	129.718	141.67328	100.705055	50.996246	136.56096
ENSG00000	3811.9404	3944.2231	6929.4775	2112.8384	4020.4126	3335.4114	3803.871	4677.965	3465.5378	4443.0034	1339.4672	3685.1326
ENSG00000	24.05818	86.0677	10.924377	182.4188	61.897373	30.3192	67.46472	121.98081	87.64625	66.15537	39.508606	124.33941
ENSG00000	150.59164	145.47685	144.81953	101.119286	133.314	116.26222	211.79	185.7629	167.92206	128.47917	64.423546	202.2175
ENSG00000	164.50064	283.9439	911.2041	237.23932	195.17467	219.54022	334.52084	340.66434	530.53284	260.713	150.73477	290.39447
ENSG00000	677.6302	723.3635	778.9519	648.6704	654.75354	810.3993	463.08386	740.62585	634.70605	787.825	331.64746	549.5765
ENSG00000	113.302536	42.788414	21.369286	58.191277	42.022808	70.34963	36.99966	83.86802	74.15625	35.893425	21.676195	91.16115
ENSG00000	501.54877	339.92618	291.4867	708.7257	331.07486	720.51196	363.6663	692.7734	570.3952	378.81766	260.25204	546.9333
ENSG00000	198.18593	93.55468	88.91049	140.98128	109.664795	227.63315	175.29486	282.58344	247.23534	181.70479	120.930275	220.61317
ENSG00000	175.49452	171.69826	219.85947	184.18932	121.96013	249.82089	193.96674	313.0637	402.5816	186.16425	153.5487	304.20248
ENSG00000	6.706802	0	0	0	0	0	0	0	0.53838795	0	0	0
ENSG00000	9.609085	7.181365	6.1711364	10.5857935	5.757707	6.6149054	13.922814	15.214148	14.810676	12.512659	6.0250754	9.950732

What is the sum of the column?

Lets create a quick barplot to summarize the number of total reads for each sample.

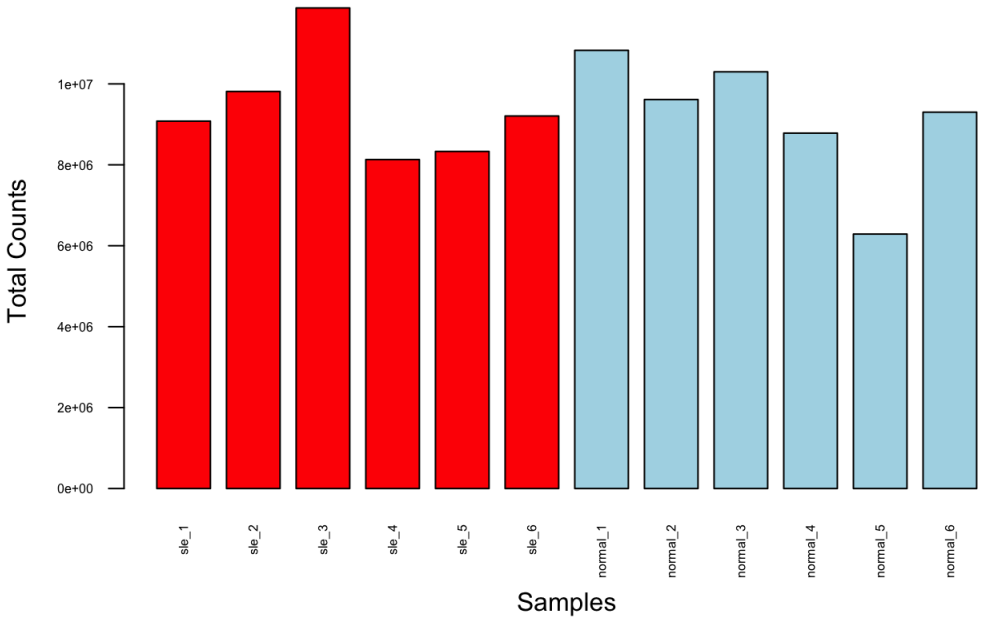
```
#barplot of total reads
barplot(colSums(data), las=2)
```



We can take a few moments to visualize this information better.

```
barcolors <- c("red", "red", "red", "red", "red", "red", "lightblue", "lightblue", "lightblue", "lightblu
e", "lightblue", "lightblue")
```

```
#formatted barplot of total reads
barplot(colSums(data), las=2, xlab = "Samples", ylab = "Total Counts", col = barcolors, cex.names = 0.5, c
ex.axis = 0.5, space = 0.3)
```

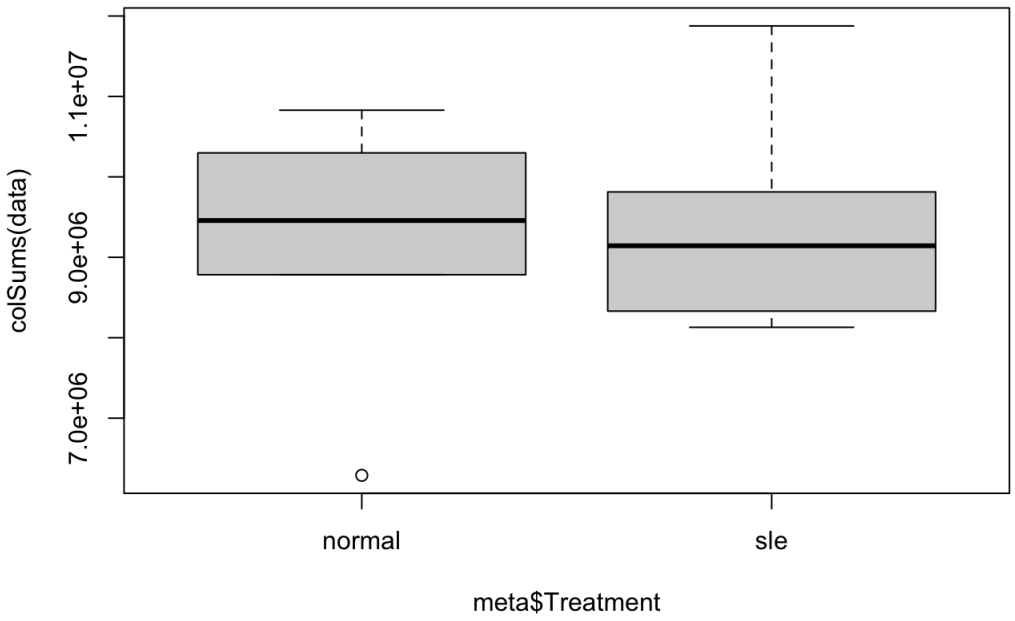


Question 2: Are there statistically significant differences in the total counts by treatment?

	SLE1	SLE2	SLE3	SLE4	SLE5	SLE6	NOR1	NOR2	NOR3	NOR4	NOR5	NOR6
Gene	SRR6870284	SRR6870286	SRR6870292	SRR6870298	SRR6870302	SRR6870304	SRR6870356	SRR6870358	SRR6870360	SRR6870364	SRR6870366	SRR6870370
ENSG00000	46.39945	24.518372	37.593605	44.98844	34.440155	39.04514	19.496567	37.345833	24.32008	15.374405	17.389055	26.844183
ENSG00000	0	0	0	0	0	0	0	0	0	0	0	2.720015
ENSG00000	229.4371	213.17186	206.52602	145.13773	193.27979	237.87276	166.87498	142.88065	202.96245	204.91154	103.15216	237.55928
ENSG00000	529.4098	441.44977	401.67133	581.027	472.23468	651.1209	405.19418	539.30914	605.55975	351.65338	265.8642	481.14212
ENSG00000	84.782425	78.22288	120.59514	95.55912	120.39508	77.850006	83.136086	129.718	141.67328	100.705055	50.996246	136.56096
ENSG00000	3811.9404	3944.2231	6929.4775	2112.8384	4020.4126	3335.4114	3803.871	4677.965	3465.5378	4443.0034	1339.4672	3685.1326
ENSG00000	24.05818	86.0677	10.924377	182.4188	61.897373	30.3192	67.46472	121.98081	87.64625	66.15537	39.508606	124.33941
ENSG00000	150.59164	145.47685	144.81953	101.119286	133.314	116.26222	211.79	185.7629	167.92206	128.47917	64.423546	202.2175
ENSG00000	164.50064	283.9439	911.2041	237.23932	195.17467	219.54022	334.52084	340.66434	530.53284	260.713	150.73477	290.39447
ENSG00000	677.6302	723.3635	778.9519	648.6704	654.75354	810.3993	463.08386	740.62585	634.70605	787.825	331.64746	549.5765
ENSG00000	113.302536	42.788414	21.369286	58.191277	42.022808	70.34963	36.99966	83.86802	74.15625	35.893425	21.676195	91.16115
ENSG00000	501.54877	339.92618	291.4867	708.7257	331.07486	720.51196	363.6663	692.7734	570.3952	378.81766	260.25204	546.9333
ENSG00000	198.18593	93.55468	88.91049	140.98128	109.664795	227.63315	175.29486	282.58344	247.23534	181.70479	120.930275	220.61317
ENSG00000	175.49452	171.69826	219.85947	184.18932	121.96013	249.82089	193.96674	313.0637	402.5816	186.16425	153.5487	304.20248
ENSG00000	6.706802	0	0	0	0	0	0	0	0.53838795	0	0	0
ENSG00000	9.609085	7.181365	6.1711364	10.5857935	5.757707	6.6149054	13.922814	15.214148	14.810676	12.512659	6.0250754	9.950732

Why is this important to ask? If we find there are differences, between the total number of reads between sle vs normal, this means that any downstream biological interpretation will due to lack of sequencing depth NOT biology.

```
boxplot(colSums(data) ~ meta$Treatment)
```




```
t.test(colSums(data) ~ meta$Treatment)
```

```
##
##  Welch Two Sample t-test
##
## data:  colSums(data) by meta$Treatment
## t = -0.25805, df = 9.7523, p-value = 0.8017
## alternative hypothesis: true difference in means between group normal and group sle is not equal to 0
## 95 percent confidence interval:
##  -2130360  1689482
## sample estimates:
## mean in group normal    mean in group sle
##              9185838              9406277
```

The p-value is 0.8017. There is not a significant difference between the total number of counts by treatment group. This is good news! We do not want the total number of counts to vary signficantly by treatment group because then we would not be able to compare the groups.

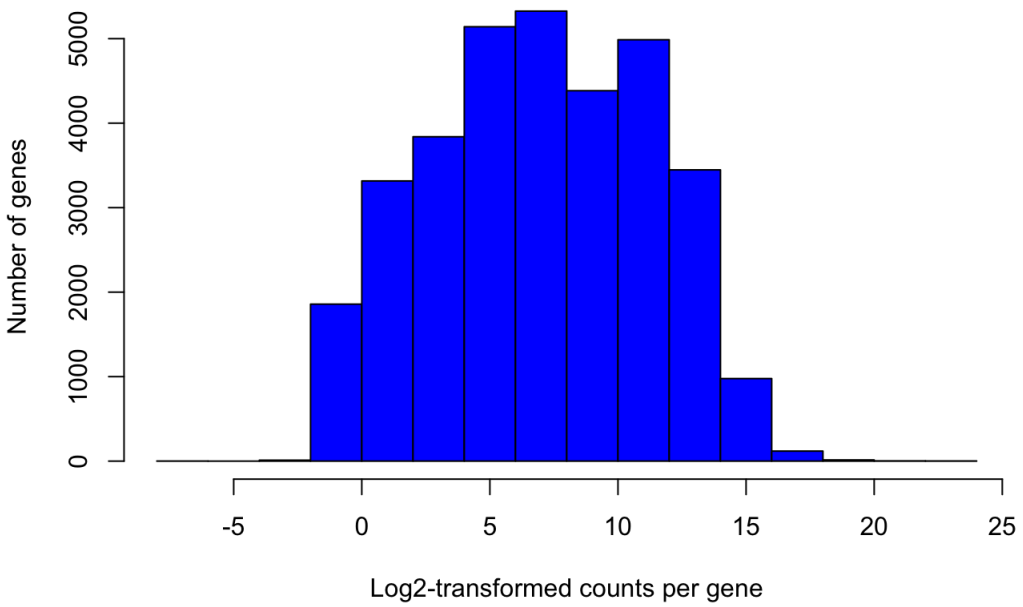
Excluding Low Abundance Genes

Question 3: What is the sum of read counts across each row?

	SLE1	SLE2	SLE3	SLE4	SLE5	SLE6	NOR1	NOR2	NOR3	NOR4	NOR5	NOR6
Gene	SRR6870284	SRR6870286	SRR6870292	SRR6870298	SRR6870302	SRR6870304	SRR6870356	SRR6870358	SRR6870360	SRR6870364	SRR6870366	SRR6870370
ENSG000000	46.39945	24.518372	37.593605	44.98844	34.440155	39.04514	19.496567	37.345833	24.32008	15.374405	17.389055	26.844183
ENSG000000	0	0	0	0	0	0	0	0	0	0	0	2.720015
ENSG000000	229.4371	213.17186	206.52602	145.13773	193.27979	237.87276	166.87498	142.88065	202.96245	204.91154	103.15216	237.55928
ENSG000000	529.4098	441.44977	401.67133	581.027	472.23468	651.1209	405.19418	539.30914	605.55975	351.65338	265.8642	481.14212
ENSG000000	84.782425	78.22288	120.59514	95.55912	120.39508	77.850006	83.136086	129.718	141.67328	100.705055	50.996246	136.56096
ENSG000000	3811.9404	3944.2231	6929.4775	2112.8384	4020.4126	3335.4114	3803.871	4677.965	3465.5378	4443.0034	1339.4672	3685.1326
ENSG000000	24.05818	86.0677	10.924377	182.4188	61.897373	30.3192	67.46472	121.98081	87.64625	66.15537	39.508606	124.33941
ENSG000000	150.59164	145.47685	144.81953	101.119286	133.314	116.26222	211.79	185.7629	167.92206	128.47917	64.423546	202.2175
ENSG000000	164.50064	283.9439	911.2041	237.23932	195.17467	219.54022	334.52084	340.66434	530.53284	260.713	150.73477	290.39447
ENSG000000	677.6302	723.3635	778.9519	648.6704	654.75354	810.3993	463.08386	740.62585	634.70605	787.825	331.64746	549.5765
ENSG000000	113.302536	42.788414	21.369286	58.191277	42.022808	70.34963	36.99966	83.86802	74.15625	35.893425	21.676195	91.16115
ENSG000000	501.54877	339.92618	291.4867	708.7257	331.07486	720.51196	363.6663	692.7734	570.3952	378.81766	260.25204	546.9333
ENSG000000	198.18593	93.55468	88.91049	140.98128	109.664795	227.63315	175.29486	282.58344	247.23534	181.70479	120.930275	220.61317
ENSG000000	173.49432	171.69826	219.83947	184.18932	121.96013	249.82089	193.96674	313.0637	402.5816	186.16423	133.5487	304.20248
ENSG000000	6.706802	0	0	0	0	0	0	0	0.53838795	0	0	0
ENSG000000	9.609085	7.181365	6.1711364	10.5857935	5.757707	6.6149054	13.922814	15.214148	14.810676	12.512659	6.0250754	9.950732

This gives us a quick picture of how many rows are empty or have 0 values. Remember that log2(500)=8.9658.

```
#histogram
hist(log2(rowSums(data)), xlab = "Log2-transformed counts per gene", ylab = "Number of genes", main = NUL
L, col = "blue")
```

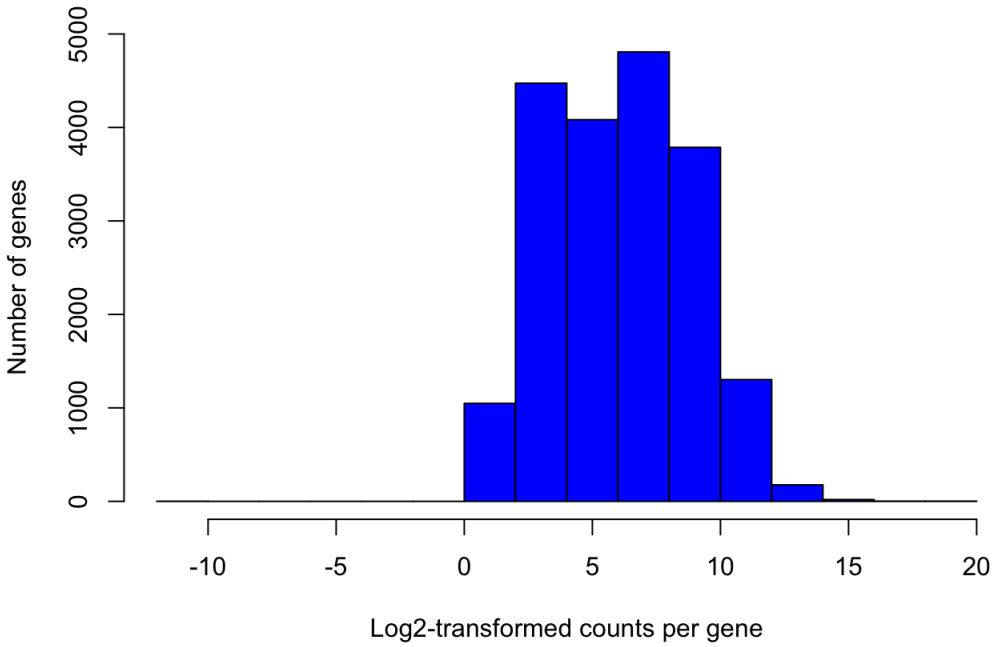



```
#find minimum value in each row
MinVals <- apply(data, 1, min)
sum(MinVals == 0)
```

```
## [1] 23662
```

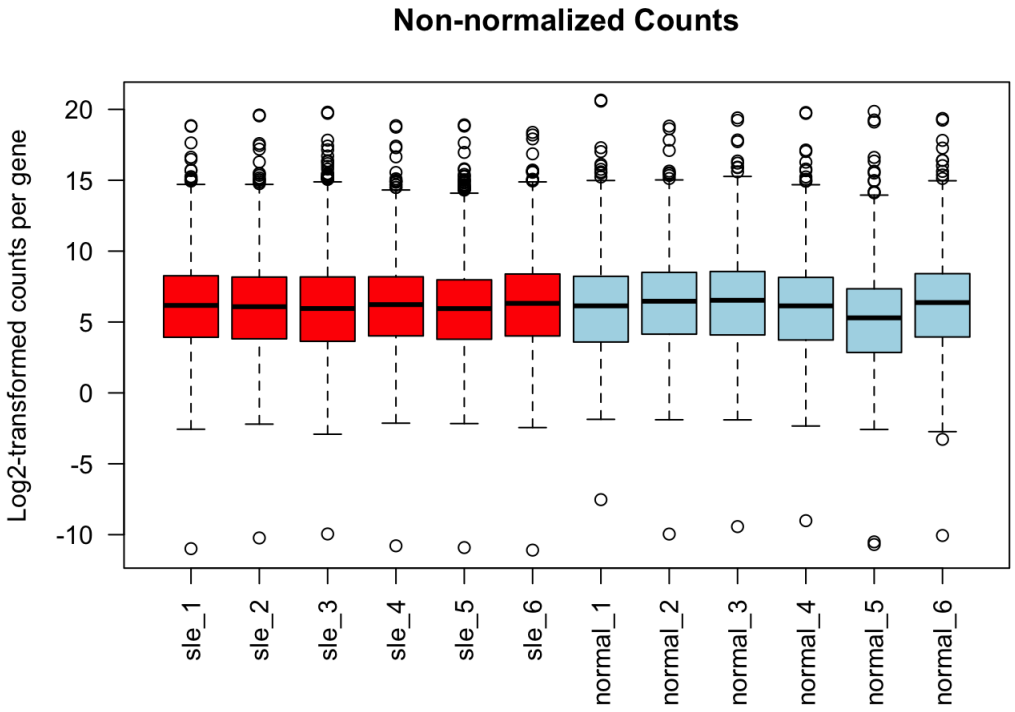
```
# removing rows with zero counts
data2 <- data[MinVals > 0, ]
```

```
# convert to matrix
data2_log <- as.matrix(log2(data2))
hist(rowMeans(data2_log),  xlab = "Log2-transformed counts per gene", ylab = "Number of genes", main = NUL
L, col = "blue")
```



To get better insights into the distribution per sample, boxplots offers a good perspective.

```
boxplot(data2_log, ylab="Log2-transformed counts per gene", main = "Non-normalized Counts",las=2, col = ba
rcolors)
```

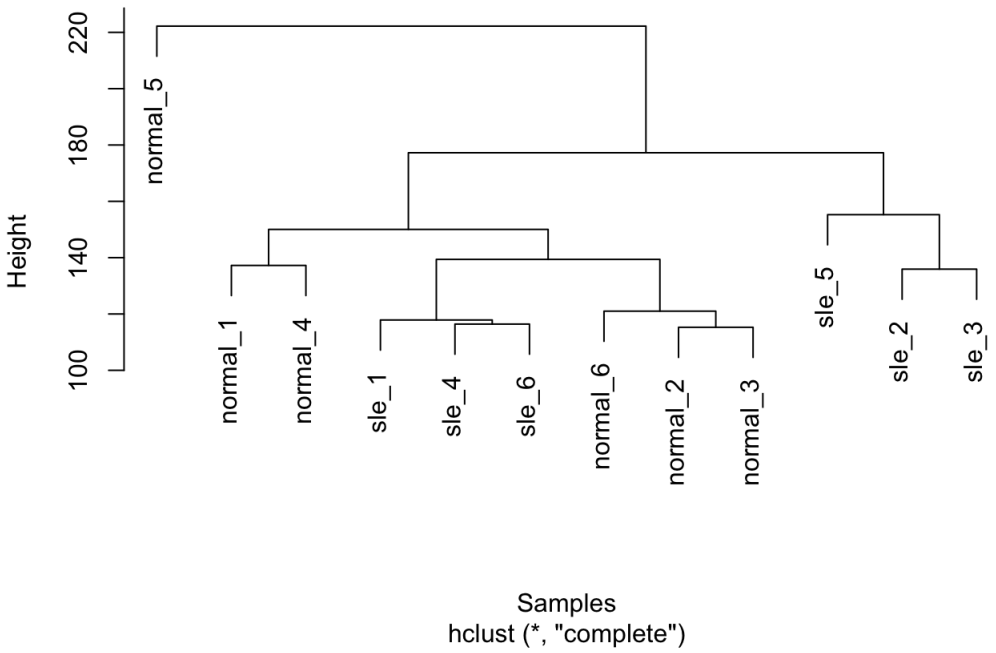


We can see that one sample **normal 5** has a lower total read count compared to the other samples.

Question 4: How variable are these samples compared to one another? What would you predict?

```
data_dist <- dist(t(data2_log), method = "euclidean")

plot(hclust(data_dist, method = "complete"), xlab = "Samples", main = NULL)
```



Below is a simple normalization where the median of each row is subtracted from the mean of the row. The outcome is “correction factor” that will be applied to each individual sample.

```
SampleMedians <- apply(data2_log, 2, median)
GrandMedian <- mean(SampleMedians)
CorrectionFactors <- GrandMedian - SampleMedians
CorrectionFactors
```

```
##        sle_1        sle_2        sle_3        sle_4        sle_5        sle_6
## -0.033709638  0.056259032  0.184132747 -0.091185129  0.189167082 -0.180139238
##     normal_1     normal_2     normal_3     normal_4     normal_5     normal_6
## -0.003349524 -0.329179379 -0.396310830 -0.002251669  0.842412612 -0.235846066
```

Now, the next step will loop through each column and fill in the medians adjusted with sample correction factor

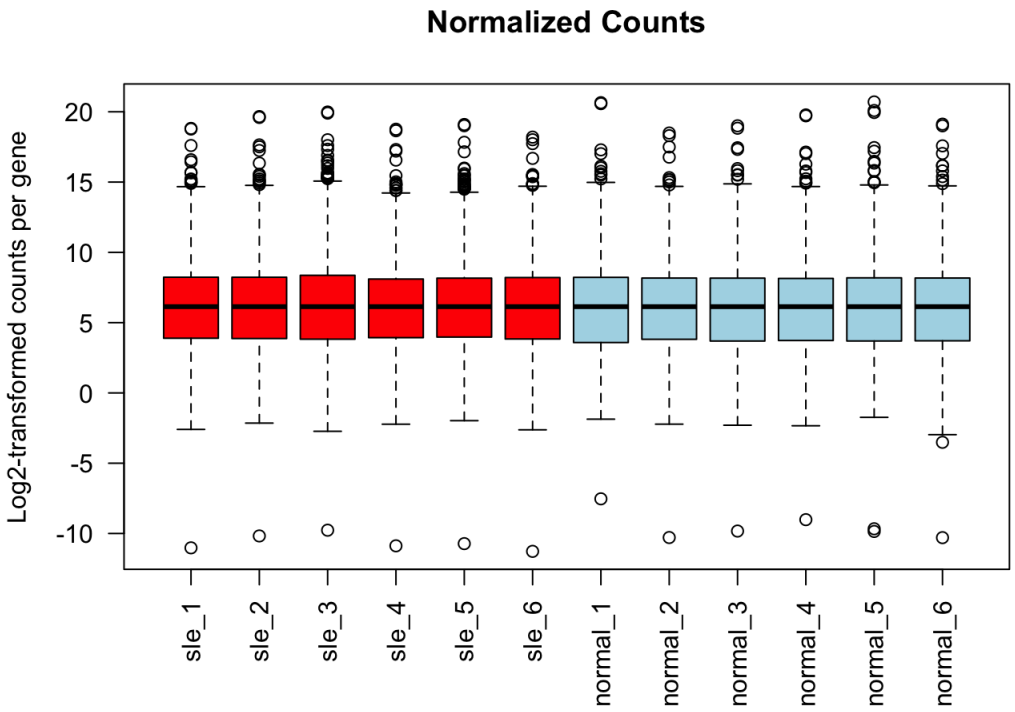
```
ExpNorm <- data2_log

for(col in colnames(ExpNorm)){
  ExpNorm[, col] <- data2_log[,col] + CorrectionFactors[col]
}
```

Class Exercise

Generate a boxplot of now “normalized” counts using `ExpNorm` as the required object (x).

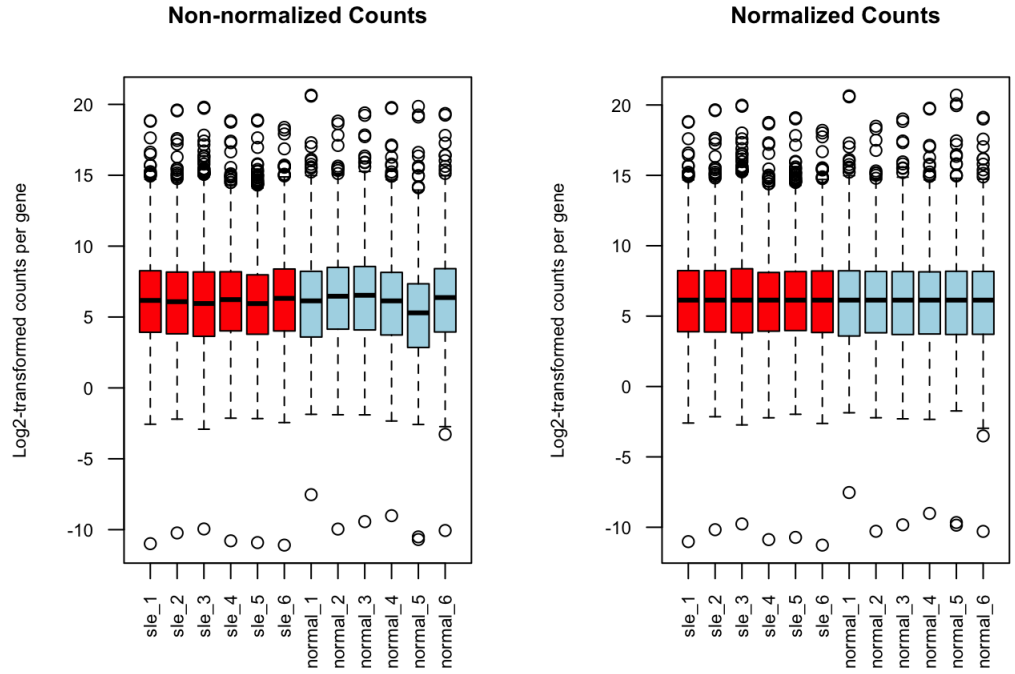
```
boxplot(ExpNorm, ylab = "Log2-transformed counts per gene", main = "Normalized Counts", las = 2, col = bar
colors)
```



The `par(mfrow)` parameter allows you to arrange multiple plots in the same plotting space.

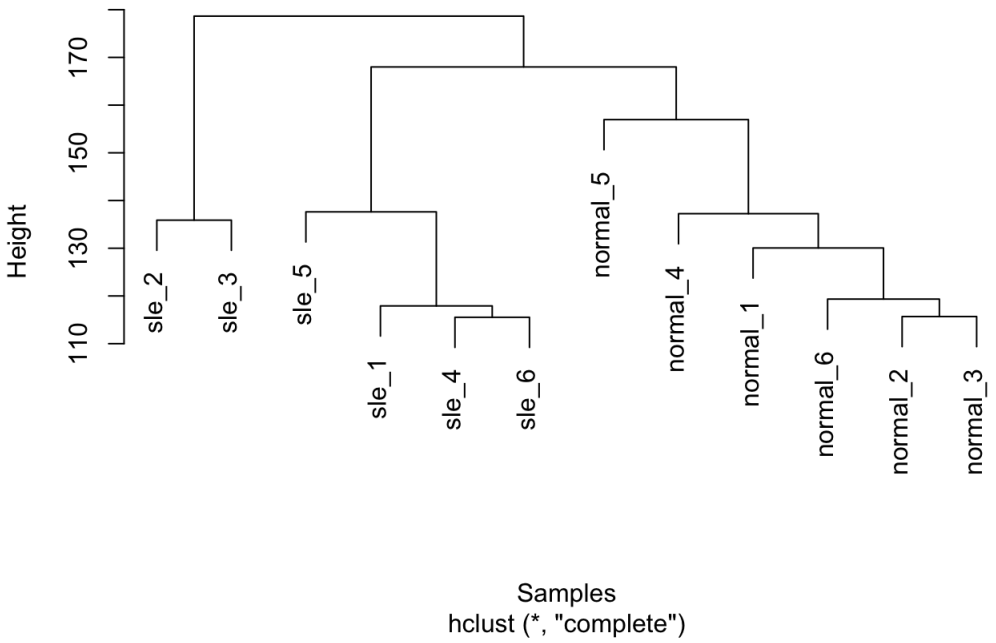
```
par(mfrow=c(1,2),cex.lab=0.7, cex.main = 0.9, cex.axis = 0.7)

boxplot(data2_log, ylab="Log2-transformed counts per gene", main = "Non-normalized Counts",las=2, col = ba
rcolors)
boxplot(ExpNorm, ylab = "Log2-transformed counts per gene", main = "Normalized Counts", las = 2, col = bar
colors)
```



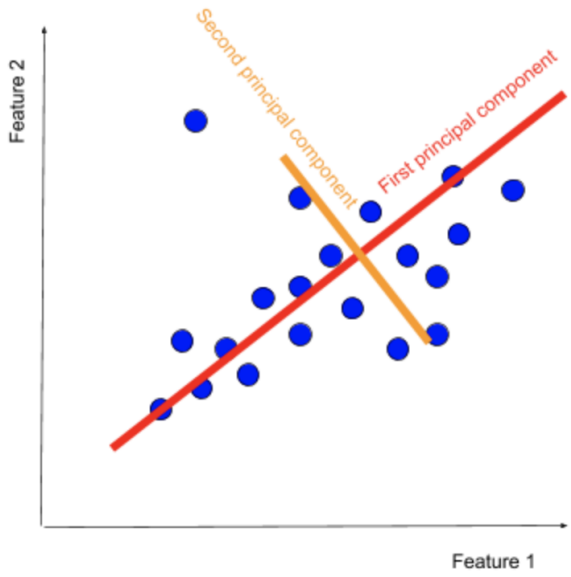
Cluster dendrogram of normalized data

```
NormDist <- dist(t(ExpNorm), method = "euclidean")
plot(hclust(NormDist, method = "complete"), xlab = "Samples", main = NULL)
```



Principal Component Analysis

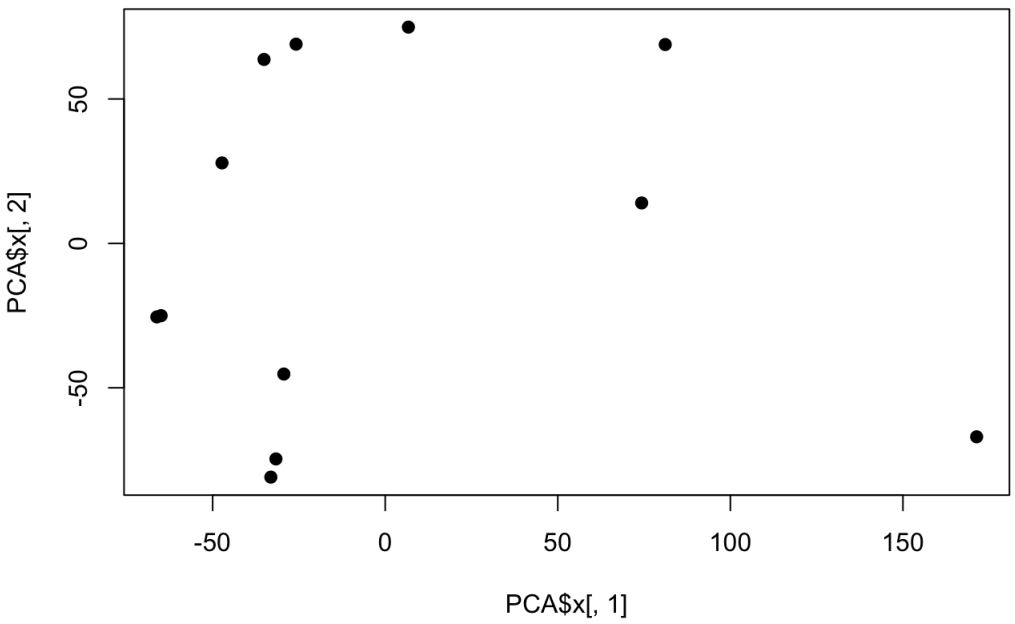
Principal component analysis is a popular technique for analyzing large datasets containing a high number of dimensions/features per observation. PCA increases the interpretability of data while preserving the maximum amount of information, and enables the visualization of multidimensional data.



The first principal component is computed so that it explains the greatest amount of variance in the original features. The second component is orthogonal to the first, and it explains the second greatest amount of variance.

Why is this useful? With large datasets (where there are many different variables), principal components allows for the interrogation of sample clustering and its cause.

```
PCA <- prcomp(t(NormDist))
plot(PCA$x[ , 1], PCA$x[ , 2], pch = 19)
```



Let's use colors to highlight groups SLE versus normal

```
plot(PCA$x[ ,1], PCA$x[ , 2], pch = 19, col = meta$Treatment, main = "Colored by Treatment", xlab = "PCA
1", ylab = "PCA2", xlim=c(-100,200), ylim = c(-100,100))

legend("topright", legend = c("SLE", "normal"), pch = 16, col = c("#FF6666", "black"), bty = "n")
```