Advanced Statistics in EdgeR

Research Informatics Core

February 21, 2023

Contents

1 Morning

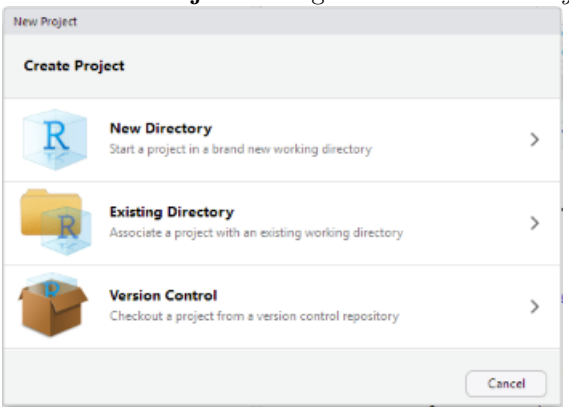
REFRESHER: Using R Studio

TIPS

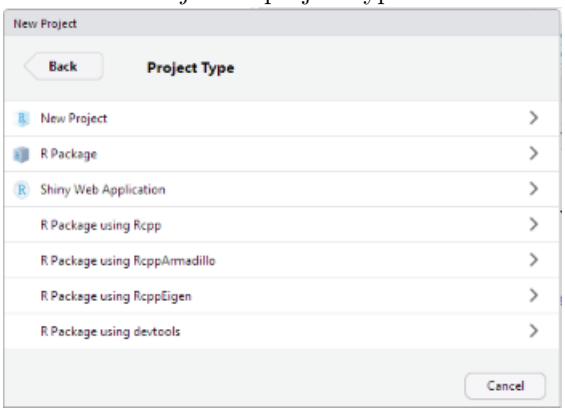
- Use Tab to auto complete
- Use up arrow to get previous command

1.0.1 Create a new project in R Studio

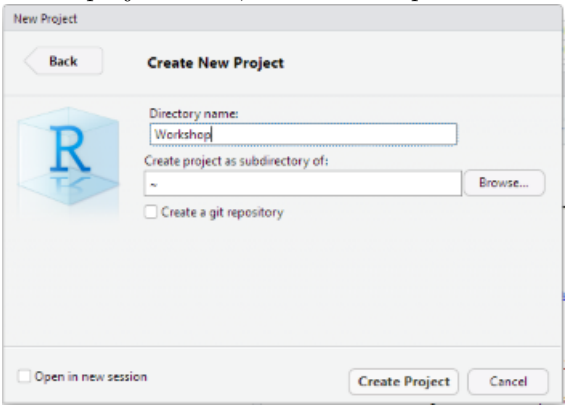
1. From *File* menu select *New Project...*
2. From **New Project** Dialog select *New Directory*



3. Select *New Project* as project type.



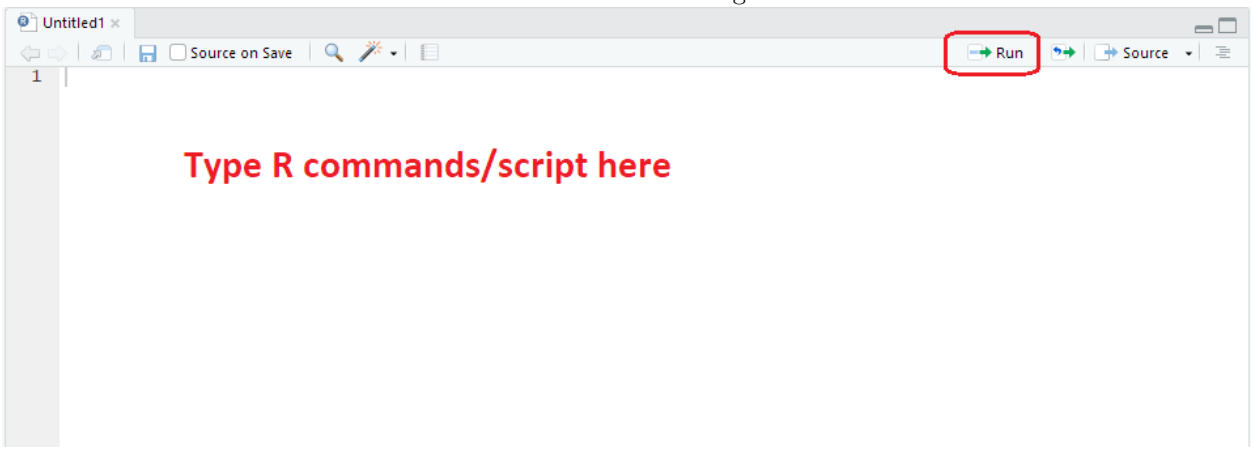
4. Give a project name, i.e. "Workshop" and click **Create Project** button.



1.0.2 Create a new script.

1. From the *File* menu select *New File* > *R Script*
2. Save file as "my_Rscript.R"

Write commands in code editor of R Studio and run them using icon **Run** in R Studio.



1.1 Prepare counts and metadata tables from RNA-seq results files

We'll download files gene counts per sample for an example RNA-seq project, and combine them into a single counts table. We'll also prepare a metadata file for this project.

First, download the table of zip counts at this URL:

https://uic-ric.github.io/workshop-data/edgeR/featurecounts.zip

And double-click to unzip it. There should be 6 files, which are from the featureCounts program. Each has gene expression counts for one sample. Here is a preview of the first one (note that long fields are truncated):

```
# Program:featureCounts v2.0.3; Command:"featureCounts" "-t" "exon" "-g" "gene_id" "-p"
    "-T" "1" "-s" "1" "-a" "mm9.gtf" "-o" "A.1" "A.1.bam"
Geneid   Chr          Start        End          Strand       Length  A.1.bam
Xkr4     chr1;chr...  3204563;...  3207049;...  -;-;-        3634    0
Rp1      chr1;chr...  4280927;...  4283093;...  -;-;-;-;...  9747    7
Sox17    chr1;chr...  4481009;...  4482749;...  -;-;-;-;...  3130    229
Mrpl15   chr1;chr...  4763279;...  4764597;...  -;-;-;-;...  4203    80
Lypla1   chr1;chr...  4797974;...  4798063;...  +;+;+;+;...  2433    184
Tcea1    chr1;chr...  4847775;...  4848057;...  +;+;+;+;...  2847    108
Rgs20    chr1;chr...  4899657;...  4900743;...  -;-;-;-;...  2241    0
Atp6v1h  chr1;chr...  5073254;...  5073359;...  +;+;+;+;...  1976    233
```

There's a comment line (first line starting with #) giving the command used to generate this file, and after that the fields in the table give the gene ID, all exon coordinates and strands (Chr, Start, End, and Strand), total gene length in bp, and read counts for sample A.1. The read counts (last column) is the data we want for each sample, along with the gene IDs.

What we need to do is:

- Make a list of files to combine
- Read each file into R
- Create a table of the gene ID and gene counts for each sample
- Write this to a new file

The steps below assume that the genes are in the same order in each file (we use cbind() without checking that the gene names match). In practice this is true for featureCounts files, but it's also something that's good to double-check in general.

```
# Make a list of all of the files ending in "counts.txt".
# This assumes you've downloaded all of the files to your "Downloads" folder
# and unpacked them there.
# full.names prints the entire directory structure, which we need to read them in.
fc.files <- list.files("~/Downloads/featurecounts",
   pattern="*.counts.txt",full.names=T)
# note that you file paths would reflect where the files are on YOUR computer
fc.files
```

```
## [1] "/Users/mmaiensc/Downloads/featurecounts/A.1.counts.txt"
## [2] "/Users/mmaiensc/Downloads/featurecounts/A.2.counts.txt"
## [3] "/Users/mmaiensc/Downloads/featurecounts/A.3.counts.txt"
## [4] "/Users/mmaiensc/Downloads/featurecounts/B.1.counts.txt"
## [5] "/Users/mmaiensc/Downloads/featurecounts/B.2.counts.txt"
## [6] "/Users/mmaiensc/Downloads/featurecounts/B.3.counts.txt"
```

```
# Read the files into R
# First read in the first file
# NOTE:
#  - R ignores comment lines (#), so the first line of the file will be skipped
```

```
#  - We're subsetting the data frame to just column 6 (gene counts)
#    This was column 7, but we set the first column as row names
#  - We set drop=F to keep it structured as a data frame (otherwise would become a vector)
counts.table <- read.table(fc.files[1],sep="\t",header=T,row.names=1)[,6,drop=F]
head(counts.table)
```

```
##        A.1.bam
## Xkr4         0
## Rp1          7
## Sox17      229
## Mrpl15      80
## Lypla1     184
## Tcea1      108
```

```
# Then read in the rest of the files and combine them
# the selection [-1] drops the first item from the list
for(f in fc.files[-1]){
  # here we read in the file and combine it with counts.table in one step
  counts.table <- cbind(counts.table, read.table(f,sep="\t",header=T,row.names=1)[,6,drop=F])
}
head(counts.table)
```

```
##        A.1.bam A.2.bam A.3.bam B.1.bam B.2.bam B.3.bam
## Xkr4         0       0       0       1       0       0
## Rp1          7      18      28      18       8       9
## Sox17      229     451     234     367     456     436
## Mrpl15      80     130     104     157     231     207
## Lypla1     184     215     189     292     344     377
## Tcea1      108     192     148     175     208     219
```

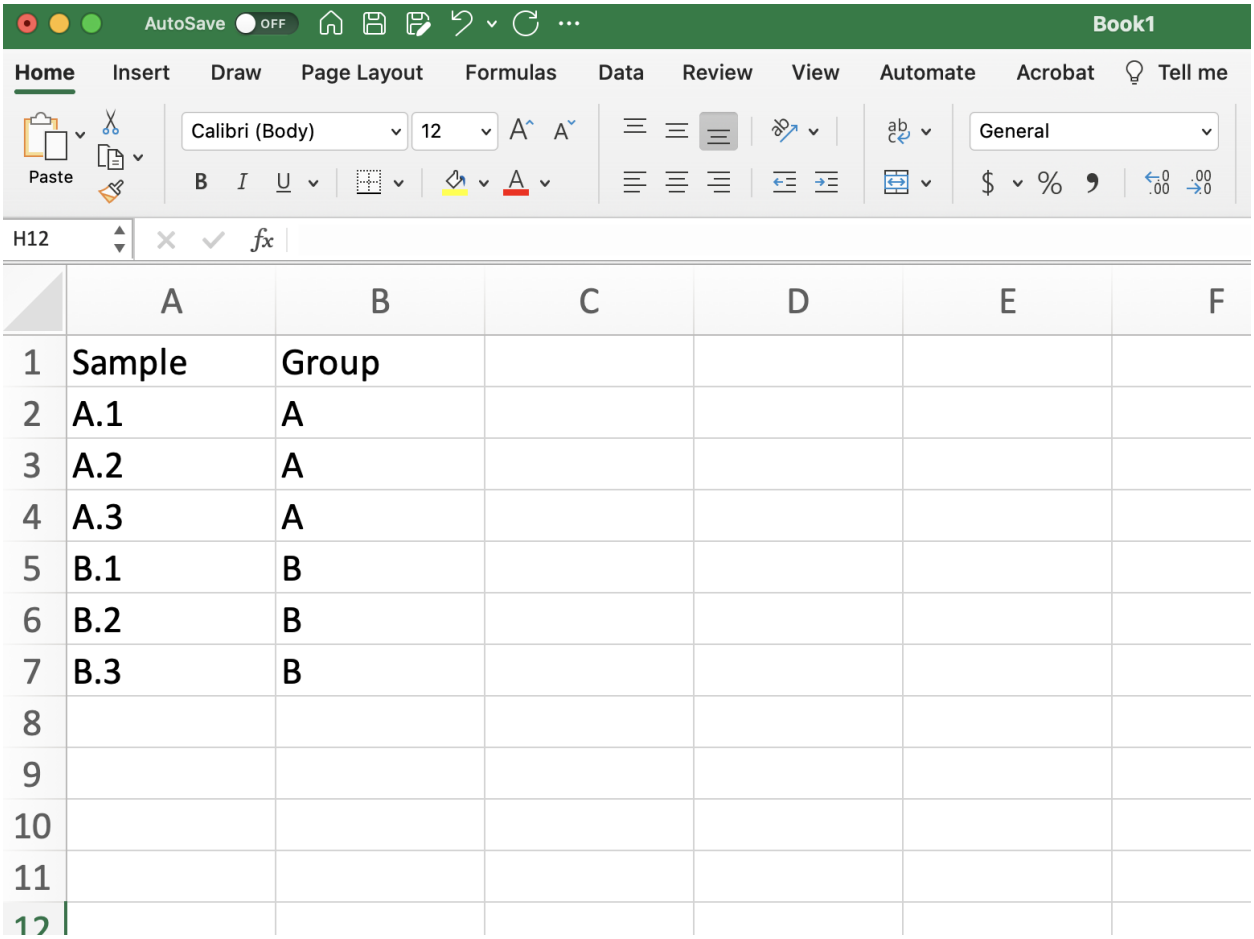
```
# Remove the ".bam" suffixes from the column names
colnames(counts.table) <- gsub(".bam$","",colnames(counts.table))
head(counts.table)
```

```
##        A.1 A.2 A.3 B.1 B.2 B.3
## Xkr4     0   0   0   1   0   0
## Rp1      7  18  28  18   8   9
## Sox17  229 451 234 367 456 436
## Mrpl15  80 130 104 157 231 207
## Lypla1 184 215 189 292 344 377
## Tcea1  108 192 148 175 208 219
```

```
# now write it to a file
write.table(counts.table,"combined_counts.txt",sep="\t",row.names=T,col.names=NA,quote=F)
```

You now have a counts table.

To make the metadata file, open up Microsoft Excel and create a file as follows:



	A	B	C	D	E	F
1	Sample	Group				
2	A.1	A				
3	A.2	A				
4	A.3	A				
5	B.1	B				
6	B.2	B				
7	B.3	B				
8						
9						
10						
11						
12						

Then choose Save As > Tab delimited Text (.txt). You now have a metadata table.

1.2 Pairwise differential analysis in edgeR

- Start a new R script and save it to your computer as "edgeR.R"
- Execute the edgeR commands one-at-a-time

ABOUT: These data are from an RNA-seq experiment with two experimental groups, A and B. There are three replicates per group.

```
library(edgeR)
```

```
## Loading required package: limma
```

```
# read in counts
data <- read.table("https://uic-ric.github.io/workshop-data/edgeR/pairwise_counts.txt",
   header=T,row.names=1)
dim(data)
```

```
## [1] 23420     6
```

```
head(data)
```

```
##        A.1 A.2 A.3 B.1 B.2 B.3
## Xkr4     0   0   0   1   0   0
## Rp1      7  18  28  18   8   9
## Sox17  229 451 234 367 456 436
## Mrpl15  80 130 104 157 231 207
## Lypla1 184 215 189 292 344 377
## Tcea1  108 192 148 175 208 219
```

```
# read in metadata table
metadata <- read.table("https://uic-ric.github.io/workshop-data/edgeR/pairwise_metadata.txt",
   header=T,row.names=1)
metadata
```

```
##     Group
## A.1     A
## A.2     A
## A.3     A
## B.1     B
## B.2     B
## B.3     B
```

```
# make sure the metadata row order matches the data column order
data <- data[,rownames(metadata)]
# check the data again to make sure we didn't drop any columns!
dim(data)
```

```
## [1] 23420     6
```

```
head(data)
```

```
##        A.1 A.2 A.3 B.1 B.2 B.3
## Xkr4     0   0   0   1   0   0
## Rp1      7  18  28  18   8   9
## Sox17  229 451 234 367 456 436
## Mrpl15  80 130 104 157 231 207
## Lypla1 184 215 189 292 344 377
## Tcea1  108 192 148 175 208 219
```

```
# subset counts to genes with >20 total counts
data_subset <- data[rowSums(data)>20,]
```

```
dim(data_subset)
```

```
## [1] 14193     6
```

```
# create edgeR object
genes <- DGEList(counts=data_subset, group=metadata[,1])
genes
```

```
## An object of class "DGEList"
## $counts
##        A.1 A.2 A.3 B.1 B.2 B.3
## Rp1      7  18  28  18   8   9
## Sox17  229 451 234 367 456 436
## Mrpl15  80 130 104 157 231 207
## Lypla1 184 215 189 292 344 377
## Tcea1  108 192 148 175 208 219
## 14188 more rows ...
##
## $samples
##     group lib.size norm.factors
## A.1     A  3326193            1
## A.2     A  5032893            1
## A.3     A  5352456            1
## B.1     B  4979670            1
## B.2     B  5620129            1
## B.3     B  5886895            1
```

```
# calculate TMM factors
genes <- calcNormFactors(genes)
genes
```

```
## An object of class "DGEList"
## $counts
##        A.1 A.2 A.3 B.1 B.2 B.3
## Rp1      7  18  28  18   8   9
## Sox17  229 451 234 367 456 436
## Mrpl15  80 130 104 157 231 207
## Lypla1 184 215 189 292 344 377
## Tcea1  108 192 148 175 208 219
## 14188 more rows ...
##
## $samples
##     group lib.size norm.factors
## A.1     A  3326193    1.0236147
## A.2     A  5032893    1.0200193
## A.3     A  5352456    0.8468997
## B.1     B  4979670    1.1083495
## B.2     B  5620129    0.9269172
## B.3     B  5886895    1.1007924
```

```
# estimate dispersion
genes <- estimateDisp(genes)
```

```
## Using classic mode.
```

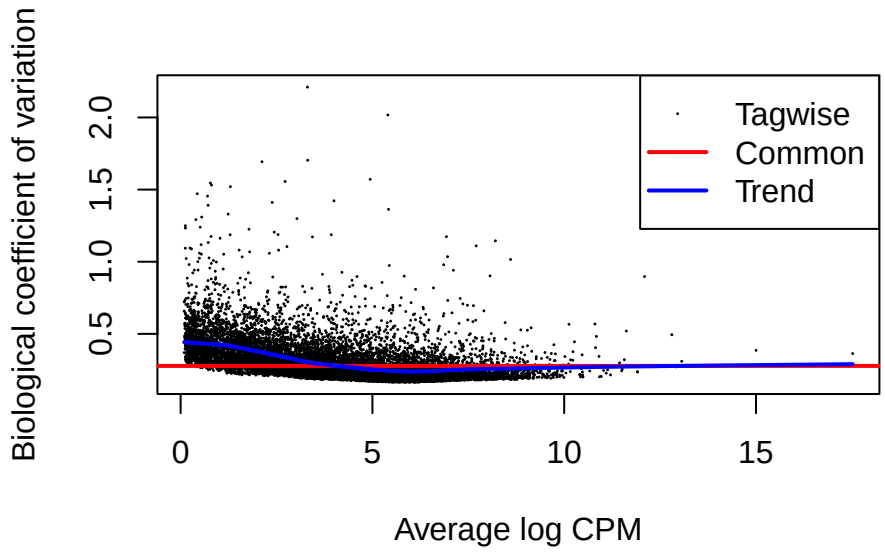
```
genes
```

```
## An object of class "DGEList"
## $counts
##        A.1 A.2 A.3 B.1 B.2 B.3
## Rp1      7  18  28  18   8   9
## Sox17  229 451 234 367 456 436
## Mrpl15  80 130 104 157 231 207
## Lypla1 184 215 189 292 344 377
## Tcea1  108 192 148 175 208 219
## 14188 more rows ...
##
## $samples
##     group lib.size norm.factors
## A.1     A  3326193    1.0236147
## A.2     A  5032893    1.0200193
## A.3     A  5352456    0.8468997
## B.1     B  4979670    1.1083495
## B.2     B  5620129    0.9269172
## B.3     B  5886895    1.1007924
##
## $common.dispersion
## [1] 0.07726973
##
## $trended.dispersion
## [1] 0.15541353 0.05724855 0.06490358 0.05832588 0.06245742
## 14188 more elements ...
##
## $tagwise.dispersion
## [1] 0.17099271 0.05072614 0.04405357 0.03575971 0.03376216
## 14188 more elements ...
##
## $AveLogCPM
## [1] 1.746894 6.165060 4.900612 5.724314 5.127011
## 14188 more elements ...
##
## $trend.method
## [1] "locfit"
##
## $prior.df
## [1] 4.134752
##
## $prior.n
## [1] 1.033688
##
## $span
## [1] 0.2945153
```

```
# calculate BCV from the common dispersion
sqrt(genes$common.dispersion)
```

```
## [1] 0.2779743
```

```
plotBCV(genes)
```




```
# run differential
stats <- exactTest(genes)
# NOTE: logFC will be computed as B/A: positive means higher in B
#       edgeR calculates this as the second group over the first
#       group from exactTest
stats
```

```
## An object of class "DGEExact"
## $table
##              logFC   logCPM     PValue
## Rp1    -0.93760389 1.746894 0.12038547
## Sox17   0.09658678 6.165060 0.72540551
## Mrpl15  0.54242240 4.900612 0.04323394
## Lypla1  0.36738965 5.724314 0.12314337
## Tcea1   0.04750527 5.127011 0.84813699
## 14188 more rows ...
##
## $comparison
## [1] "A" "B"
##
## $genes
## NULL
```

```
# run FDR correction
stats$table$QValue <- p.adjust(stats$table$PValue,method="BH")
head(stats$table)
```

```
##               logFC   logCPM     PValue    QValue
## Rp1     -0.93760389 1.746894 0.12038547 0.4167392
## Sox17    0.09658678 6.165060 0.72540551 0.9122603
## Mrpl15   0.54242240 4.900612 0.04323394 0.2537715
## Lypla1   0.36738965 5.724314 0.12314337 0.4207448
## Tcea1    0.04750527 5.127011 0.84813699 0.9637028
## Atp6v1h  0.21299802 6.416323 0.43648856 0.7456270
```

```
nrow(stats$table[stats$table$QValue<0.05,])
```

```
## [1] 684
```

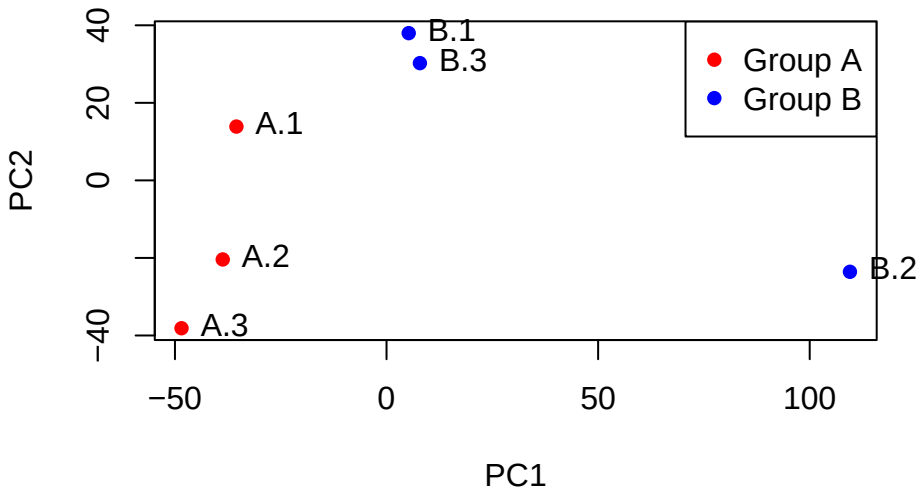
```
# calculate normalized expression
norm <- cpm(genes)
head(norm)
```

```
##               A.1       A.2       A.3       B.1        B.2       B.3
## Rp1      2.055957  3.506278  6.176934  3.261333   1.535687  1.388835
## Sox17   67.259172 87.851756 51.621518 66.494965  87.534172 67.281361
## Mrpl15  23.496654 25.323122 22.942897 28.446075  44.342969 31.943215
## Lypla1  54.042304 41.880549 41.694303 52.906076  66.034551 58.176773
## Tcea1   31.720483 37.400304 32.649507 31.707408  39.927868 33.794996
## Atp6v1h 68.434005 85.903823 81.623768 79.359114 121.895174 72.682388
```

1.3 PCA plot

- Use the normalized expression data from above

```
# log-scale the data first (this is good practice, as gene expression varies
#    over several orders of magnitude)
# add a pseudo-count (0.1) to avoid taking the log of 0
norm.log <- log2(norm + 0.1)
# run PCA on the transpose of cpm.log: we want to plot the SAMPLES
pca <- prcomp(t(norm.log))
# make colors for our plot
pca.colors <- c(rep("Red",3),rep("Blue",3))
plot(pca$x[,1],pca$x[,2],col=pca.colors,xlab="PC1",ylab="PC2",pch=16)
# add sample name labels to plot:
#    pos=4 makes them left-justified
#    xpd=NA lets the labels go off of the plot area
text(pca$x[,1],pca$x[,2],colnames(norm),pos=4,xpd=NA)
legend('topright',legend=c("Group A","Group B"),col=c("Red","Blue"),pch=16)
```



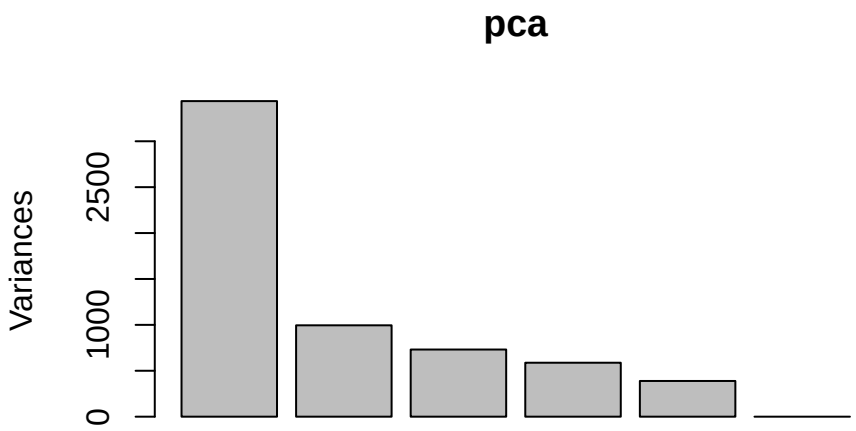

```
# check percent variation per PC
summary(pca)
```

```
## Importance of components:
##                             PC1     PC2     PC3      PC4      PC5        PC6
## Standard deviation     58.6293 31.5442 27.0428 24.24291 19.72479 1.234e-13
## Proportion of Variance  0.5598  0.1620  0.1191  0.09571  0.06336 0.000e+00
## Cumulative Proportion   0.5598  0.7218  0.8409  0.93664  1.00000 1.000e+00
```

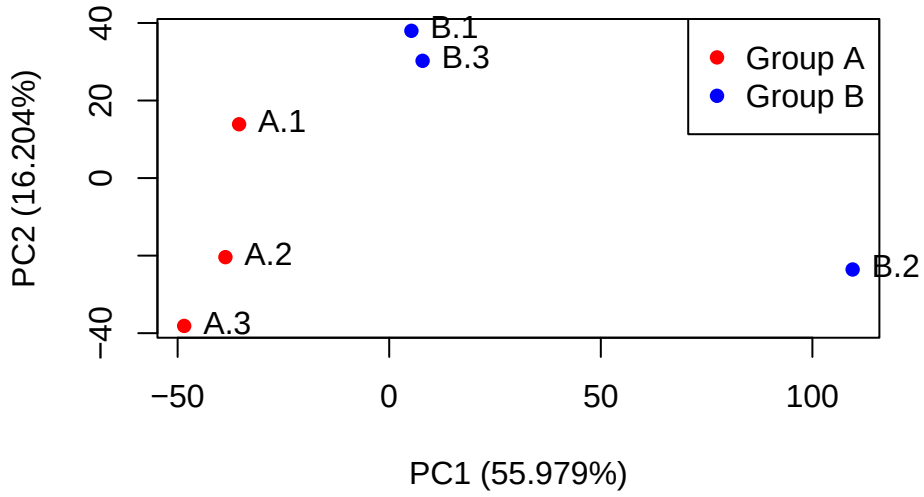
```
# this is the vector of % variance per PC
summary(pca)$importance[2,]
```

```
##     PC1     PC2     PC3     PC4     PC5     PC6
## 0.55979 0.16204 0.11910 0.09571 0.06336 0.00000
```

```
screeplot(pca)
```




```
# add the % variation to the axis labels
xlabel <- paste("PC1 (", 100*summary(pca)$importance[2,1], "%)",sep="")
ylabel <- paste("PC2 (", 100*summary(pca)$importance[2,2], "%)",sep="")
plot(pca$x[,1],pca$x[,2],col=pca.colors,xlab=xlabel,ylab=ylabel,pch=16)
text(pca$x[,1],pca$x[,2],colnames(norm),pos=4,xpd=NA)
legend('topright',legend=c("Group A","Group B"),col=c("Red","Blue"),pch=16)
```



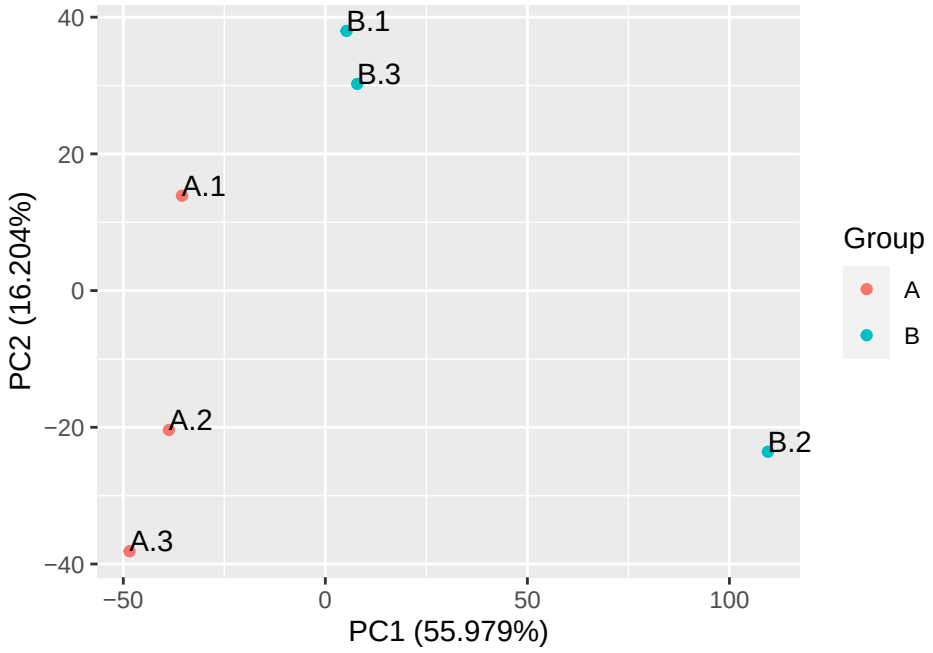
We can also make the plot with ggplot, which is a bit easier. For later exercises we will plot with ggplot2.

```
# combine coordinates with metadata and sample names
pca.data <- cbind(pca$x,metadata,Sample=rownames(metadata))
head(pca.data)
```

```
##            PC1       PC2        PC3        PC4          PC5           PC6 Group
## A.1 -35.477601  13.87805  18.720179 -41.274380   9.44146617 -7.929989e-14     A
## A.2 -38.707100 -20.39163 -48.826270  -4.993259  -2.02870077  2.319600e-14     A
## A.3 -48.452775 -38.13005  30.033762  20.884106  -4.27595253 -9.075600e-14     A
## B.1   5.241417  37.98072   1.380732   8.167031 -31.76147825  6.448961e-14     B
## B.2 109.511454 -23.57698   2.287999  -8.110552  -0.09805812 -2.160046e-14     B
## B.3   7.884604  30.23989  -3.596401  25.327054  28.72272351  1.396929e-13     B
##     Sample
## A.1    A.1
## A.2    A.2
## A.3    A.3
## B.1    B.1
```

```
## B.2    B.2
## B.3    B.3
```

```
library(ggplot2)
# geom_label adds the sample labels; hjust=0 and vjust=0 makes it left-justified
# coord_cartesian(clip='off') lets the labels go outside the plot boundary
ggplot(pca.data,aes(x=PC1,y=PC2,label=Sample,color=Group)) +
  geom_point() + geom_text(hjust=0, vjust=0, show.legend=F, color="black") +
  coord_cartesian(clip='off') + labs(x=xlabel, y=ylabel)
```



1.4 Heatmap

- Make a heatmap of z-scored log2 CPM values for all differentially expressed genes (FDR<0.05)
- We'll specify our own color scheme and limit the ranges of z-scores that are plotted

```
library(ComplexHeatmap)
```

```
## Loading required package: grid
```

```
## ========================================
## ComplexHeatmap version 2.12.1
## Bioconductor page: http://bioconductor.org/packages/ComplexHeatmap/
## Github page: https://github.com/jokergoo/ComplexHeatmap
## Documentation: http://jokergoo.github.io/ComplexHeatmap-reference
##
## If you use it in published research, please cite either one:
## - Gu, Z. Complex heatmaps reveal patterns and correlations in multidimensional
##     genomic data. Bioinformatics 2016.
## - Gu, Z. Complex Heatmap Visualization. iMeta 2022.
##
##
## The new InteractiveComplexHeatmap package can directly export static
## complex heatmaps into an interactive Shiny app with zero effort. Have a try!
##
## This message can be suppressed by:
##   suppressPackageStartupMessages(library(ComplexHeatmap))
## ========================================
```

```
library(circlize)
```

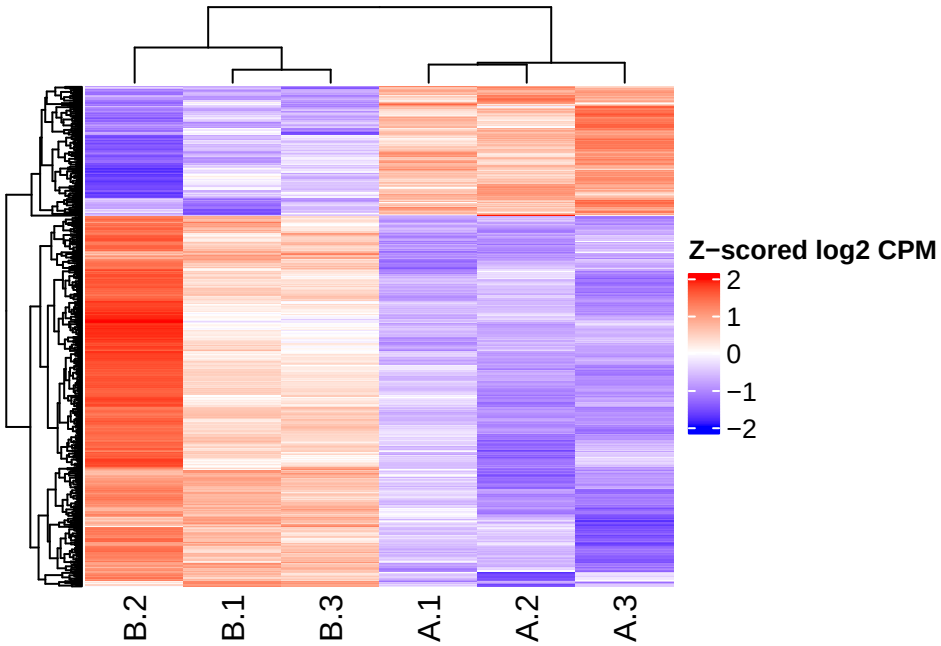
```
## ========================================
## circlize version 0.4.15
## CRAN page: https://cran.r-project.org/package=circlize
## Github page: https://github.com/jokergoo/circlize
## Documentation: https://jokergoo.github.io/circlize_book/book/
##
## If you use it in published research, please cite:
## Gu, Z. circlize implements and enhances circular visualization
##   in R. Bioinformatics 2014.
##
## This message can be suppressed by:
##   suppressPackageStartupMessages(library(circlize))
## ========================================
```

```
# Get a list of the differentially expressed genes, using a more stringent Q-value
degs.diff <- stats$table[stats$table$QValue < 0.05,]
dim(degs.diff)
```

```
## [1] 684   4
```

```
# subset the log-scaled CPMs to get the same genes
degs.norm <- norm.log[rownames(degs.diff),]
# we want to z-score across the genes, to account for gene-to-gene
#    differences in baseline expression
# need to transpose twice, as the scale function works on columns
degs.norm.z <- t(scale(t(degs.norm)))
# scale the colors from blue to white to red, with max/min at +/-2 at the middle at 0
col_fun <- colorRamp2(c(-2,0,2),c("blue","white","red"))
```

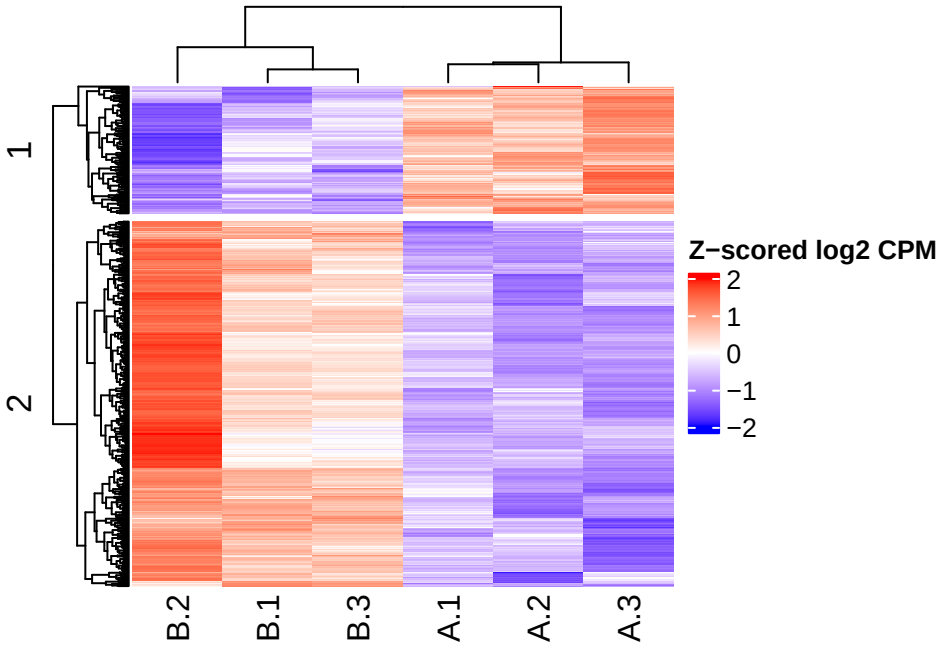
```
# basic heatmap, turning off the row names
Heatmap(degs.norm.z,show_row_names=F,name="Z-scored log2 CPM",col=col_fun)
```



Notice that sample B.2 is a little bit "different" from the other B samples, with more extreme colors. Sample B.2 is also the more "different" one in the PCA plot. Overall, B.2 is *more* different from the group A samples than the other B samples, which we can see in both the PCA plot and the heatmap.

How can we get the genes that correspond to the heatmap?

```
# use the row_split option to get 2 clusters
# save as an object also; draw() will also plot it for us
ht <- draw(Heatmap(degs.norm.z,show_row_names=F,name="Z-scored log2 CPM",
  col=col_fun,row_split=2))
```



```
# get the order of genes in the heatmap
genes_order <- row_order(ht)
```

NOTE: `genes_order` is a list of 2 vectors. Each vector is the order (indices) of the genes for each row cluster. To get the actual gene names for each cluster, we would need to select the names from the degs.norm.z based on the indices in the vectors.

```
# get gene names for the first cluster
cluster1_genes <- rownames(degs.norm.z)[ genes_order[[1]] ]
head(cluster1_genes)
```

```
## [1] "Eif2s3y" "Fbn2"    "Rbm20"   "Cst8"    "Kcnj2"   "Gtpbp2"
```

```
# use an apply statement to get the genes from all clusters
gene_clusters <- sapply(genes_order, function(x) rownames(degs.norm.z)[x])
# gene_clusters will also be a list
head(gene_clusters[[1]])
```

```
## [1] "Eif2s3y" "Fbn2"    "Rbm20"   "Cst8"    "Kcnj2"   "Gtpbp2"
```

```
head(gene_clusters[[2]])
```

```
## [1] "Ptafr"   "Fkbp10"  "Socs3"   "Lsm12"   "Cd200r1" "Wdr46"
```

You can save a really tall heatmap (with gene names turned ON) to a PDF to double-check the gene orders.

```
# make a really long PDF and inspect the gene order manually
pdf("pairwise_long_heatmap.pdf",height=100)
Heatmap(degs.norm.z,show_row_names=T,name="Z-scored log2 CPM",
  col=col_fun,row_split=2)
dev.off()
```

1.5 One-way ANOVA

- We will read in the data, set up our model, and estimate the dispersions pretty similarly to what we did before
- Then we use glmQLFit to fit the model, and glmQLFTest to test the model

ABOUT: These data are from an RNA-seq experiment comparing two different disease models (model1 and model2) to a healthy control group, so there are 3 expeirmental groups in total. There are 4 replicates per group.

```
# reload the edgeR library in case you closed R studio
library(edgeR)
# read in counts
data <- read.table("https://uic-ric.github.io/workshop-data/edgeR/anova_counts.txt",
   header=T,row.names=1)
dim(data)
```

```
## [1] 24421    12
```

```
head(data)
```

```
##        Control.1 Control.2 Control.3 Control.4 Model1.10 Model1.7 Model1.8
## Xkr4          26        33        34        24        59       41       29
## Rp1            3         1         1         3         2        0        0
## Sox17         75        54        86        94       169       82       66
## Mrpl15       537       566       633       740       905      684      717
## Lypla1       383       449       550       412       852      530      408
## Tcea1        593       557       619       476      1264      778      766
##        Model1.9 Model2.1 Model2.2 Model2.3 Model2.4
## Xkr4         38       43       29       21       21
## Rp1           0        1        0        2        1
## Sox17        87       90       65       70       77
## Mrpl15      579      470      560      697      646
## Lypla1      534      510      337      377      414
## Tcea1       887      849      619      609      504
```

```
# read in metadata table
metadata <- read.table("https://uic-ric.github.io/workshop-data/edgeR/anova_metadata.txt",
   header=T,row.names=1)
metadata
```

```
##           Condition
## Control.1   Control
## Control.2   Control
## Control.3   Control
## Control.4   Control
## Model1.10    Model1
## Model1.7     Model1
## Model1.8     Model1
## Model1.9     Model1
## Model2.1     Model2
## Model2.2     Model2
## Model2.3     Model2
## Model2.4     Model2
```

```
# make sure the metadata row order matches the data column order
data <- data[,rownames(metadata)]
# check the data again to make sure we didn't drop any columns!
```

```
dim(data)
```

```
## [1] 24421    12
```

```
# subset counts to genes with >20 total counts
data_subset <- data[rowSums(data)>20,]
dim(data_subset)
```

```
## [1] 16713    12
```

```
# define our factors and model matrix
treat <- factor(metadata[,1])
treat
```

```
##  [1] Control Control Control Control Model1  Model1  Model1  Model1  Model2
## [10] Model2  Model2  Model2
## Levels: Control Model1 Model2
```

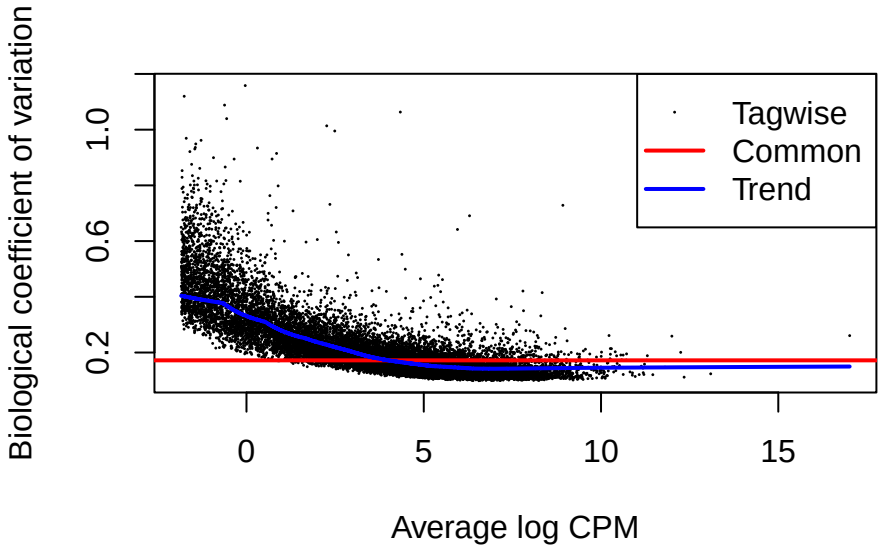
```
model <- model.matrix(~treat)
model
```

```
##    (Intercept) treatModel1 treatModel2
## 1            1           0           0
## 2            1           0           0
## 3            1           0           0
## 4            1           0           0
## 5            1           1           0
## 6            1           1           0
## 7            1           1           0
## 8            1           1           0
## 9            1           0           1
## 10           1           0           1
## 11           1           0           1
## 12           1           0           1
## attr(,"assign")
## [1] 0 1 1
## attr(,"contrasts")
## attr(,"contrasts")$treat
## [1] "contr.treatment"
```

```
# create edgeR object, without setting the groups this time
genes <- DGEList(counts=data_subset)
# calculate TMM factors
genes <- calcNormFactors(genes)
# estimate dispersion
genes <- estimateDisp(genes,model)
# calculate BCV from the common dispersion
sqrt(genes$common.dispersion)
```

```
## [1] 0.1719392
```

```
plotBCV(genes)
```




```
# fit the model
fit <- glmQLFit(genes,model)
# we are testing if either coefficient 2 or 3 has an effect
qlf <- glmQLFTest(fit, coef=2:3)
qlf
```

```
## An object of class "DGELRT"
## $coefficients
##        (Intercept)  treatModel1 treatModel2
## Xkr4    -12.905525  0.102388201 -0.10083802
## Sox17   -11.940529 -0.003795860 -0.09714796
## Mrpl15   -9.858317 -0.088182239 -0.11520116
## Lypla1  -10.181974  0.003604001 -0.16800056
## Tcea1    -9.953796  0.247642882  0.06200444
## 16708 more rows ...
##
## $fitted.values
##        Control.1 Control.2 Control.3 Control.4  Model1.10  Model1.7  Model1.8
## Xkr4     28.1162  27.83997  31.44391  29.64327   56.91949  36.68507  38.20549
## Sox17    73.9728  73.24605  82.72790  77.99047  134.61683  86.76163  90.35748
## Mrpl15  594.1755 588.33800 664.49950 626.44687  993.77219 640.49417 667.03958
## Lypla1  429.8554 425.63228 480.73121 453.20209  788.06930 507.91701 528.96773
## Tcea1   540.0582 534.75244 603.97719 569.39037 1263.82181 814.54333 848.30224
##         Model1.9  Model2.1  Model2.2  Model2.3  Model2.4
## Xkr4    34.97135  30.16443  25.02012  30.34803  28.77078
## Sox17   82.70862  79.67564  66.08758  80.16061  75.99450
## Mrpl15 610.57388 628.61220 521.40729 632.43842 599.56934
## Lypla1 484.18998 431.36985 357.80308 433.99550 411.43989
## Tcea1  776.49244 682.15466 565.81850 686.30678 650.63805
## 16708 more rows ...
##
## $deviance
##     Xkr4    Sox17   Mrpl15   Lypla1    Tcea1
## 5.444725 5.365753 6.818389 6.925917 8.804578
## 16708 more elements ...
##
## $method
```

```
## [1] "oneway"
##
## $unshrunk.coefficients
##        (Intercept)  treatModel1 treatModel2
## Xkr4    -12.909326  0.102759575 -0.10124567
## Sox17   -11.941974 -0.003797212 -0.09729660
## Mrpl15   -9.858497 -0.088198944 -0.11522322
## Lypla1  -10.182223  0.003604917 -0.16804610
## Tcea1    -9.953995  0.247686359  0.06201635
## 16708 more rows ...
##
## $df.residual
## [1] 9 9 9 9 9
## 16708 more elements ...
##
## $design
##   (Intercept) treatModel1 treatModel2
## 1           1           0           0
## 2           1           0           0
## 3           1           0           0
## 4           1           0           0
## 5           1           1           0
## 7 more rows ...
##
## $offset
##          [,1]    [,2]     [,3]     [,4]    [,5]     [,6]     [,7]    [,8]
## [1,] 16.24567 16.2358 16.35753 16.29856 16.8482 16.40894 16.44955 16.3611
##          [,9]    [,10]   [,11]    [,12]
## [1,] 16.41723 16.23025 16.4233 16.36993
## attr(,"class")
## [1] "CompressedMatrix"
## attr(,"Dims")
## [1]  5 12
## attr(,"repeat.row")
## [1] TRUE
## attr(,"repeat.col")
## [1] FALSE
## 16708 more rows ...
##
## $dispersion
## [1] 0.06742343 0.04528205 0.02207476 0.02321379 0.02182389
## 16708 more elements ...
##
## $prior.count
## [1] 0.125
##
## $AveLogCPM
## [1] 1.401471 2.691206 5.617022 5.173267 5.732831
## 16708 more elements ...
##
## $df.residual.zeros
## [1] 9 9 9 9 9
## 16708 more elements ...
##
```

```
## $df.prior
## [1] 8.589705
##
## $var.post
##      Xkr4     Sox17    Mrpl15    Lypla1     Tcea1
## 0.7549259 0.7015419 0.7345577 0.7452369 0.8463764
## 16708 more elements ...
##
## $var.prior
##      Xkr4     Sox17    Mrpl15    Lypla1     Tcea1
## 0.9120452 0.8119211 0.7104161 0.7197664 0.7081655
## 16708 more elements ...
##
## $samples
##           group lib.size norm.factors
## Control.1     1 11112335    1.0223515
## Control.2     1 10765117    1.0449583
## Control.3     1 12414349    1.0234380
## Control.4     1 12877683    0.9301165
## Model1.10     1 21699160    0.9563989
## 7 more rows ...
##
## $table
##        logFC.treatModel1 logFC.treatModel2   logCPM         F     PValue
## Xkr4         0.147714950        -0.1454785 1.401471 0.5676494 0.57690230
## Sox17       -0.005476268        -0.1401549 2.691206 0.2964354 0.74710527
## Mrpl15      -0.127220079        -0.1662001 5.617022 0.8403515 0.44816641
## Lypla1       0.005199475        -0.2423736 5.173267 1.9911074 0.16615373
## Tcea1        0.357273158         0.0894535 5.732831 3.4387713 0.05488631
## 16708 more rows ...
##
## $comparison
## [1] "treatModel1" "treatModel2"
##
## $df.test
## [1] 2 2 2 2 2
## 16708 more elements ...
##
## $df.total
## [1] 17.58971 17.58971 17.58971 17.58971 17.58971
## 16708 more elements ...
```

```
# do p-value adjustment
qlf$table$QValue <- p.adjust(qlf$table$PValue,method="BH")
head(qlf$table)
```

```
##        logFC.treatModel1 logFC.treatModel2   logCPM         F     PValue
## Xkr4         0.147714950        -0.1454785 1.401471 0.5676494 0.57690230
## Sox17       -0.005476268        -0.1401549 2.691206 0.2964354 0.74710527
## Mrpl15      -0.127220079        -0.1662001 5.617022 0.8403515 0.44816641
## Lypla1       0.005199475        -0.2423736 5.173267 1.9911074 0.16615373
## Tcea1        0.357273158         0.0894535 5.732831 3.4387713 0.05488631
## Rgs20       -0.212746891         0.2594904 5.170381 3.3717848 0.05760178
##           QValue
## Xkr4   0.6668351
```

```
## Sox17  0.8062485
## Mrpl15 0.5530580
## Lypla1 0.2658619
## Tcea1  0.1178462
## Rgs20  0.1219841
```

```
# check number of significant genes
nrow(qlf$table[qlf$table$QValue<0.05,])
```

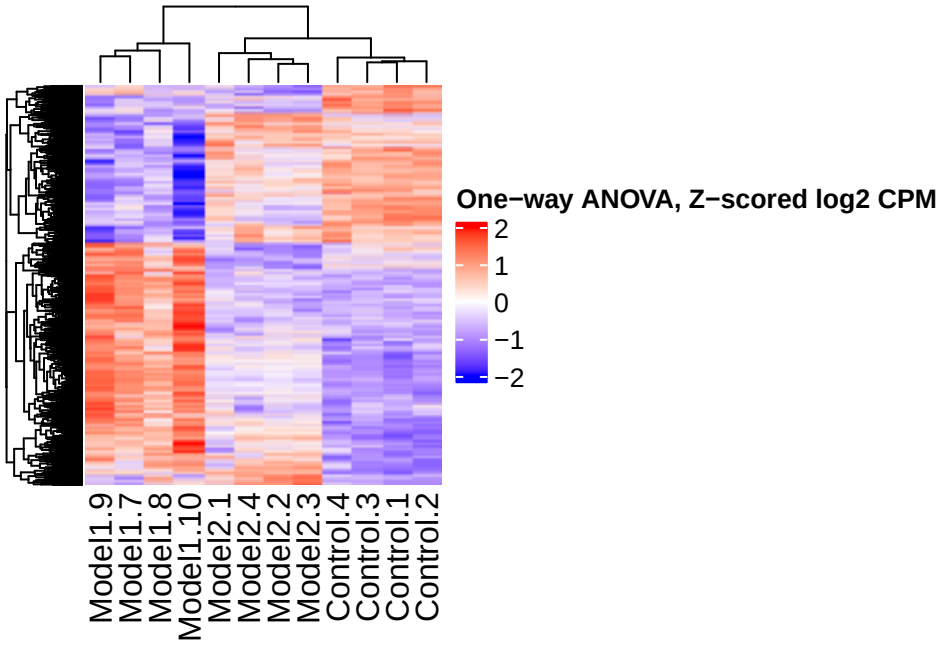
```
## [1] 5710
```

```
# make heatmap of differentially expressed genes
# directly generate log-scaled gene expression this time
norm <- cpm(genes, log=T)
# subset based on degs
degs.diff <- qlf$table[qlf$table$QValue < 0.05,]
dim(degs.diff)
```

```
## [1] 5710    6
```

```
# subset the (already) log-scaled CPMs to get the same genes
degs.norm <- norm[rownames(degs.diff),]
# z-score across the genes
degs.norm.z <- t(scale(t(degs.norm)))
```

```
# basic heatmap
Heatmap(degs.norm.z,show_row_names=F,name="One-way ANOVA, Z-scored log2 CPM",
        col=col_fun)
```



1.6 Contrasts

- Let's try a contrast where we compare the control group to the average of both treatments
- We'll remake the model matrix without an intercept for this, but using the same data as before

```
# make the model, this time without the intercept
model.alt <- model.matrix(~0+treat)
model.alt
```

```
##    treatControl treatModel1 treatModel2
## 1             1           0           0
## 2             1           0           0
## 3             1           0           0
## 4             1           0           0
## 5             0           1           0
## 6             0           1           0
## 7             0           1           0
## 8             0           1           0
## 9             0           0           1
## 10            0           0           1
## 11            0           0           1
## 12            0           0           1
## attr(,"assign")
## [1] 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$treat
## [1] "contr.treatment"
```

```
# create a new edgeR object
genes.alt <- DGEList(counts=data_subset)
# calculate TMM factors
genes.alt <- calcNormFactors(genes.alt)
# estimate dispersion
genes.alt <- estimateDisp(genes.alt,model.alt)
# calculate BCV from the common dispersion
# should be the same as last time: these models are ultimately equivalent in terms of how
# they model the data, but differ in how easily we can ask different questions of them
sqrt(genes.alt$common.dispersion)
```

```
## [1] 0.1719392
```

```
# fit the model
fit.alt <- glmQLFit(genes.alt,model.alt)
# make a couple contrasts to try
# pair-wise comparison between model1 and model2
comp1 <- makeContrasts(treatModel1 - treatModel2, levels=model.alt)
# comparison of control vs average of model1 and model2
comp2 <- makeContrasts(treatControl - (treatModel1 + treatModel2)/2, levels=model.alt)
# do qlf tests for these comparisons
qlf1 <- glmQLFTest(fit.alt, contrast=comp1)
qlf2 <- glmQLFTest(fit.alt, contrast=comp2)
# do p-value adjustment for each
qlf1$table$QValue <- p.adjust(qlf1$table$PValue,method="BH")
qlf2$table$QValue <- p.adjust(qlf2$table$PValue,method="BH")
# make heatmap of the DEGs
# we can use the same normalized expression that we got above, already in log-scale
# subset based on degs
```

```
degs.diff1 <- qlf1$table[qlf1$table$QValue < 0.05,]
degs.diff2 <- qlf2$table[qlf2$table$QValue < 0.05,]
dim(degs.diff1)
```

```
## [1] 3688    5
```

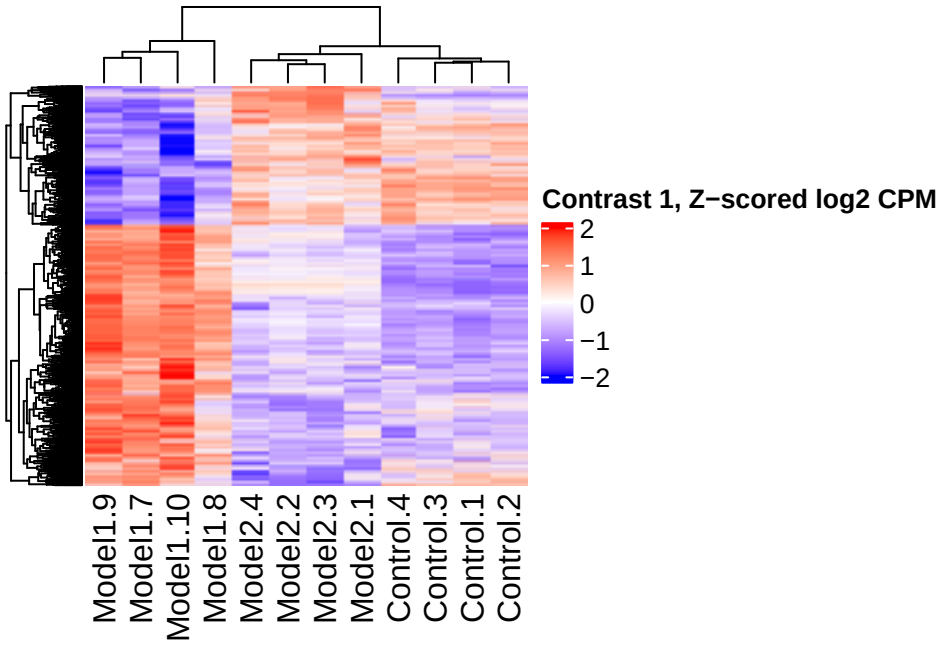
```
dim(degs.diff2)
```

```
## [1] 4030    5
```

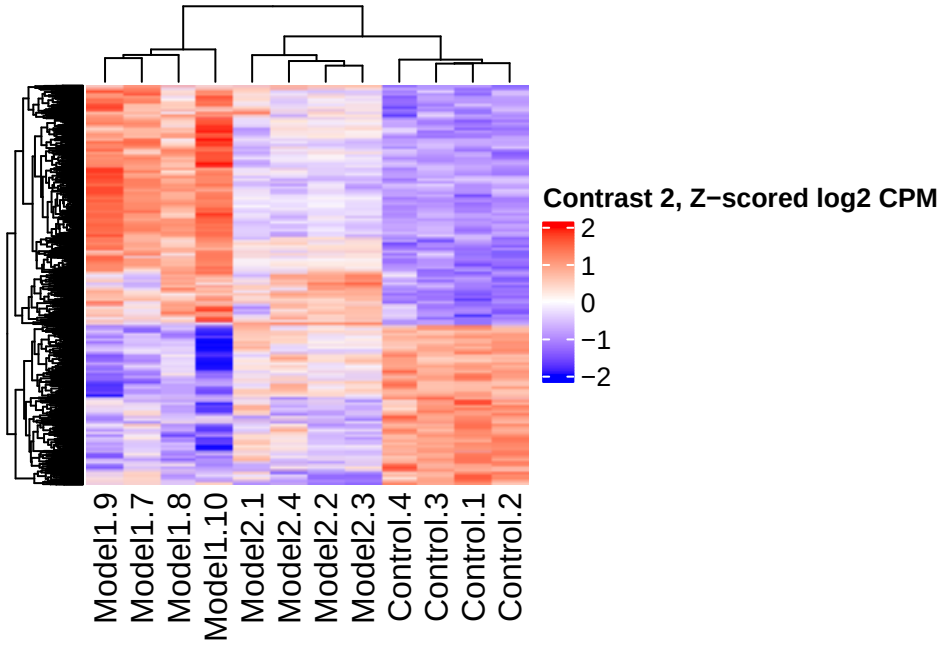
```
# how many genes are significant in both cases?
length(intersect(rownames(degs.diff1),rownames(degs.diff2)))
```

```
## [1] 1914
```

```
# next make heatmaps for DEGs for each factor
# subset the (already) log-scaled CPMs to get the same genes
degs.norm1 <- norm[rownames(degs.diff1),]
degs.norm2 <- norm[rownames(degs.diff2),]
# z-score across the genes
degs.norm.z1 <- t(scale(t(degs.norm1)))
degs.norm.z2 <- t(scale(t(degs.norm2)))
# heatmap 1: these are genes where model1 and model2 are different
Heatmap(degs.norm.z1,show_row_names=F,name="Contrast 1, Z-scored log2 CPM",
        col=col_fun)
```




```
# heatmap 2: these are genes where the average of model1 and model2 is dfferent from control
Heatmap(degs.norm.z2,show_row_names=F,name="Contrast 2, Z-scored log2 CPM",
        col=col_fun)
```


Contrast 2, Z−scored log2 CPM
2
1
0
−1
−2
Model1.9
Model1.7
Model1.8
Model1.10
Model2.1
Model2.4
Model2.2
Model2.3
Control.4
Control.3
Control.1
Control.2

1.7 Run all pairwise comparisons

- Loop over all pairs of levels and keep udpating our results
- With just 3 groups, we could write these out manually, but it's better to use a loop, since we can extend the same code to a bigger design.

```
# start a new data frame to store the results in
pw_stats <- data.frame(Gene = rownames(data_subset))
# get a list of the levels, and the number
levs <- levels(treat)
nlevs <- length(levs)
# loop over all pairs of levels
for(i in 1:(nlevs-1)){
    for(j in (i+1):nlevs){
        # names for these groups
        A <- levs[i]
        B <- levs[j]
        # get the logical vectors for these groups
        groupA <- A == treat
        groupB <- B == treat
        # subset the data for groupA or groupB data
        pw_metadata <- treat[as.logical(groupA+groupB)]
        pw_counts <- data_subset[,as.logical(groupA+groupB)]
        # run our edgeR commands
        pw_genes <- DGEList(counts=pw_counts, group=pw_metadata)
        pw_genes <- calcNormFactors(pw_genes)
        pw_genes <- estimateDisp(pw_genes)
        pw_test <- exactTest(pw_genes)
        pw_test$table$QValue <- p.adjust(pw_test$table$PValue,method="BH")
        # add the name of this comparison to the column names
        # edgeR does the comparisons as groupB / groupA, so
        # we list groupB first
        pw_name <- paste(B, A, sep="/")
        colnames(pw_test$table) = paste(pw_name, ":", colnames(pw_test$table))
        # add results to the pairwise list
        pw_stats <- cbind(pw_stats, pw_test$table)
    }
}
```

```
## Using classic mode.
## Using classic mode.
## Using classic mode.
```

```
head(pw_stats)
```

```
##          Gene Model1/Control : logFC Model1/Control : logCPM
## Xkr4     Xkr4            0.138984432                1.465855
## Sox17   Sox17           -0.013689113                2.732748
## Mrpl15 Mrpl15           -0.134684820                5.650387
## Lypla1 Lypla1           -0.004616124                5.248862
## Tcea1   Tcea1            0.347920072                5.763915
## Rgs20   Rgs20           -0.221133264                5.039171
##        Model1/Control : PValue Model1/Control : QValue Model2/Control : logFC
## Xkr4                0.64470648              0.76543151            -0.14476871
## Sox17               0.95738397              0.98514704            -0.13887247
## Mrpl15              0.33954120              0.49264277            -0.16392602
```

```
## Lypla1              0.97896788              0.99935806            -0.24083388
## Tcea1               0.01246574              0.03876022             0.09038471
## Rgs20               0.26518906              0.41282644             0.26060047
##        Model2/Control : logCPM Model2/Control : PValue Model2/Control : QValue
## Xkr4                  1.333340              0.66688795               0.8476461
## Sox17                 2.674548              0.49500181               0.7308919
## Mrpl15                5.636038              0.25514621               0.5071668
## Lypla1                5.135201              0.07745447               0.2453557
## Tcea1                 5.624921              0.55404798               0.7729386
## Rgs20                 5.281726              0.08114462               0.2531585
##        Model2/Model1 : logFC Model2/Model1 : logCPM Model2/Model1 : PValue
## Xkr4           -0.261474860               1.396063             0.41162303
## Sox17          -0.103249789               2.662185             0.67082146
## Mrpl15         -0.007752471               5.564003             0.96415608
## Lypla1         -0.214643681               5.129607             0.19795475
## Tcea1          -0.235448530               5.799491             0.16053919
## Rgs20           0.502914475               5.182926             0.01578798
##        Model2/Model1 : QValue
## Xkr4               0.6205134
## Sox17              0.8229533
## Mrpl15             0.9979526
## Lypla1             0.3969784
## Tcea1              0.3470350
## Rgs20              0.0694015
```

NOTE: you may find it useful to remake the above code as a *function*, which will allow you to rerun it again more easily on a new data set. For example:

```
pw_tests <- function( counts, groups ){
  # start a new data frame to store the results in
  pw_stats <- data.frame(Gene = rownames(counts))
  # get a list of the levels, and the number
  levs <- levels(groups)
  nlevs <- length(levs)
  # loop over all pairs of levels
  for (i in 1:(nlevs - 1)) {
    for (j in (i + 1):nlevs) {
      # names for these groups
      A <- levs[i]
      B <- levs[j]
      # get the logical vectors for these groups
      groupA <- A == groups
      groupB <- B == groups
      # subset the data for groupA or groupB data
      pw_metadata <- treat[as.logical(groupA + groupB)]
      pw_counts <- data_subset[, as.logical(groupA + groupB)]
      # run our edgeR commands
      pw_genes <- DGEList(counts = pw_counts, group = pw_metadata)
      pw_genes <- calcNormFactors(pw_genes)
      pw_genes <- estimateDisp(pw_genes)
      pw_test <- exactTest(pw_genes)
      pw_test$table$QValue <- p.adjust(pw_test$table$PValue, method = "BH")
      # add the name of this comparison to the column names
      # edgeR does the comparisons as groupB / groupA, so
      # we list groupB first
```

```
      pw_name <- paste(B, A, sep = "/")
      colnames(pw_test$table) = paste(pw_name, ":", colnames(pw_test$table))
      # add results to the pairwise list
      pw_stats <- cbind(pw_stats, pw_test$table)
    }
  }
  return(pw_stats)
}
# to run the function:
all_pairwise <- pw_tests( data_subset, treat )
```

```
## Using classic mode.
## Using classic mode.
## Using classic mode.
```

```
head(all_pairwise)
```

```
##          Gene Model1/Control : logFC Model1/Control : logCPM
## Xkr4     Xkr4            0.138984432                1.465855
## Sox17   Sox17           -0.013689113                2.732748
## Mrpl15 Mrpl15           -0.134684820                5.650387
## Lypla1 Lypla1           -0.004616124                5.248862
## Tcea1   Tcea1            0.347920072                5.763915
## Rgs20   Rgs20           -0.221133264                5.039171
##        Model1/Control : PValue Model1/Control : QValue Model2/Control : logFC
## Xkr4                0.64470648              0.76543151            -0.14476871
## Sox17               0.95738397              0.98514704            -0.13887247
## Mrpl15              0.33954120              0.49264277            -0.16392602
## Lypla1              0.97896788              0.99935806            -0.24083388
## Tcea1               0.01246574              0.03876022             0.09038471
## Rgs20               0.26518906              0.41282644             0.26060047
##        Model2/Control : logCPM Model2/Control : PValue Model2/Control : QValue
## Xkr4                  1.333340              0.66688795                0.8476461
## Sox17                 2.674548              0.49500181                0.7308919
## Mrpl15                5.636038              0.25514621                0.5071668
## Lypla1                5.135201              0.07745447                0.2453557
## Tcea1                 5.624921              0.55404798                0.7729386
## Rgs20                 5.281726              0.08114462                0.2531585
##        Model2/Model1 : logFC Model2/Model1 : logCPM Model2/Model1 : PValue
## Xkr4            -0.261474860               1.396063             0.41162303
## Sox17           -0.103249789               2.662185             0.67082146
## Mrpl15          -0.007752471               5.564003             0.96415608
## Lypla1          -0.214643681               5.129607             0.19795475
## Tcea1           -0.235448530               5.799491             0.16053919
## Rgs20            0.502914475               5.182926             0.01578798
##        Model2/Model1 : QValue
## Xkr4                0.6205134
## Sox17               0.8229533
## Mrpl15              0.9979526
## Lypla1              0.3969784
## Tcea1               0.3470350
## Rgs20               0.0694015
```

2 Afternoon

2.1 Two-way ANOVA

- Set up and run a two-way ANOVA-type model with an interaction term
- **NOTE**: the columns of the data and the rows of the metadata are not actually in the same order for this data set. We will use the same trick to reorder the data within R, but make sure you don't skip this step.

ABOUT: These data are from an RNA-seq experiment where cells were sorted according to three gating strategies (A/B/C). There are two genotypes in the data, WT and a mutant (KO). There are two replicates per group.

```
# load the edgeR library, if you've restarted R
library(edgeR)
# read in counts
data <- read.table("https://uic-ric.github.io/workshop-data/edgeR/twofactor_counts.txt",
   header=T,row.names=1)
dim(data)
```

```
## [1] 23762    12
```

```
head(data)
```

```
##        WT_A.1 WT_B.1 WT_C.1 WT_A.2 WT_B.2 WT_C.2 KO_A.1 KO_A.2 KO_C.1 KO_C.2
## Xkr4        0      0      0      0      0      0      5      0      0      0
## Rp1         0      0      0      0      0      0     89      0     17      3
## Sox17       0      1      0      0      0      0      0      0      0      0
## Mrpl15   2029   2731   4899   2301   3222   4842   1492   1527   3482   3548
## Lypla1    995   1418   2825    983   1224   2687   1444   1485   3682   3809
## Tcea1    1826   2029   1847   1979   2254   1921   1143   1164   1731   1601
##        KO_B.1 KO_B.2
## Xkr4        0      0
## Rp1         4     10
## Sox17       0      1
## Mrpl15   2109   2228
## Lypla1   2276   2523
## Tcea1    1304   1308
```

```
# read in metadata table
metadata <- read.table("https://uic-ric.github.io/workshop-data/edgeR/twofactor_metadata.txt",
   header=T,row.names=1)
metadata
```

```
##        Genotype Gating
## WT_A.1       WT      A
## WT_A.2       WT      A
## WT_B.1       WT      B
## WT_B.2       WT      B
## WT_C.1       WT      C
## WT_C.2       WT      C
## KO_A.1       KO      A
## KO_A.2       KO      A
## KO_B.1       KO      B
## KO_B.2       KO      B
## KO_C.1       KO      C
## KO_C.2       KO      C
```

```
# make sure the metadata row order matches the data column order
data <- data[,rownames(metadata)]
# check the data again to make sure we didn't drop any columns!
dim(data)
```

```
## [1] 23762    12
```

```
# subset counts to genes with >20 total counts
data_subset <- data[rowSums(data)>20,]
dim(data_subset)
```

```
## [1] 15051    12
```

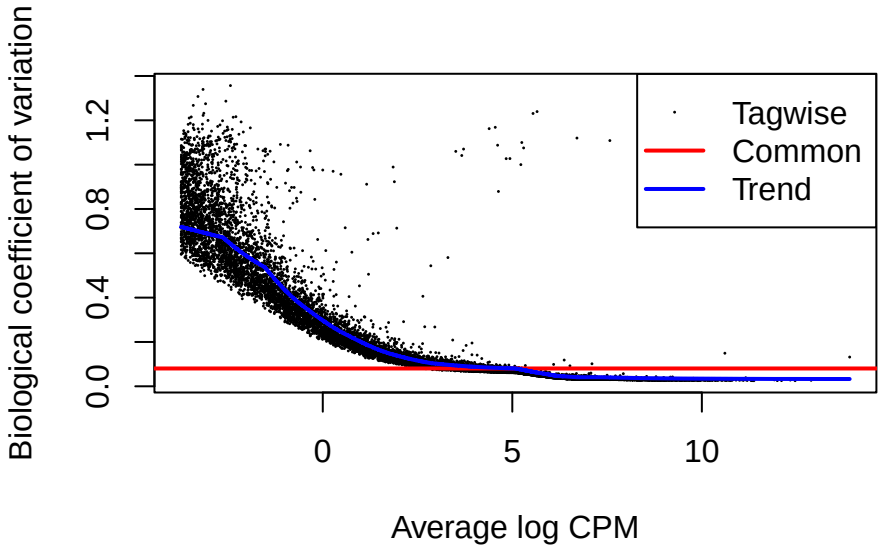
```
# define our factors and model matrix
geno <- factor(metadata[,1])
gate <- factor(metadata[,2])
model <- model.matrix(~ geno + gate + geno:gate)
model
```

```
##    (Intercept) genoWT gateB gateC genoWT:gateB genoWT:gateC
## 1            1      1     0     0            0            0
## 2            1      1     0     0            0            0
## 3            1      1     1     0            1            0
## 4            1      1     1     0            1            0
## 5            1      1     0     1            0            1
## 6            1      1     0     1            0            1
## 7            1      0     0     0            0            0
## 8            1      0     0     0            0            0
## 9            1      0     1     0            0            0
## 10           1      0     1     0            0            0
## 11           1      0     0     1            0            0
## 12           1      0     0     1            0            0
## attr(,"assign")
## [1] 0 1 2 2 3 3
## attr(,"contrasts")
## attr(,"contrasts")$geno
## [1] "contr.treatment"
##
## attr(,"contrasts")$gate
## [1] "contr.treatment"
```

```
# create edgeR object
genes <- DGEList(counts=data_subset)
# calculate TMM factors
genes <- calcNormFactors(genes)
# estimate dispersion
genes <- estimateDisp(genes,model)
# calculate BCV from the common dispersion
sqrt(genes$common.dispersion)
```

```
## [1] 0.08033678
```

```
plotBCV(genes)
```




```
# fit the model
fit <- glmQLFit(genes,model)
# now we test each factor in turn, keeping in mind
# which coefficients these correspond to
# first test for genotype (1 term)
qlf.geno <- glmQLFTest(fit, coef=2)
qlf.gate <- glmQLFTest(fit, coef=3:4)
qlf.inter <- glmQLFTest(fit, coef=5:6)
# do p-value adjustment
qlf.geno$table$QValue <- p.adjust(qlf.geno$table$PValue,method="BH")
qlf.gate$table$QValue <- p.adjust(qlf.gate$table$PValue,method="BH")
qlf.inter$table$QValue <- p.adjust(qlf.inter$table$PValue,method="BH")
# check number of significant genes
nrow(qlf.geno$table[qlf.geno$table$QValue<0.05,])
```

```
## [1] 8310
```

```
nrow(qlf.gate$table[qlf.gate$table$QValue<0.05,])
```

```
## [1] 9165
```

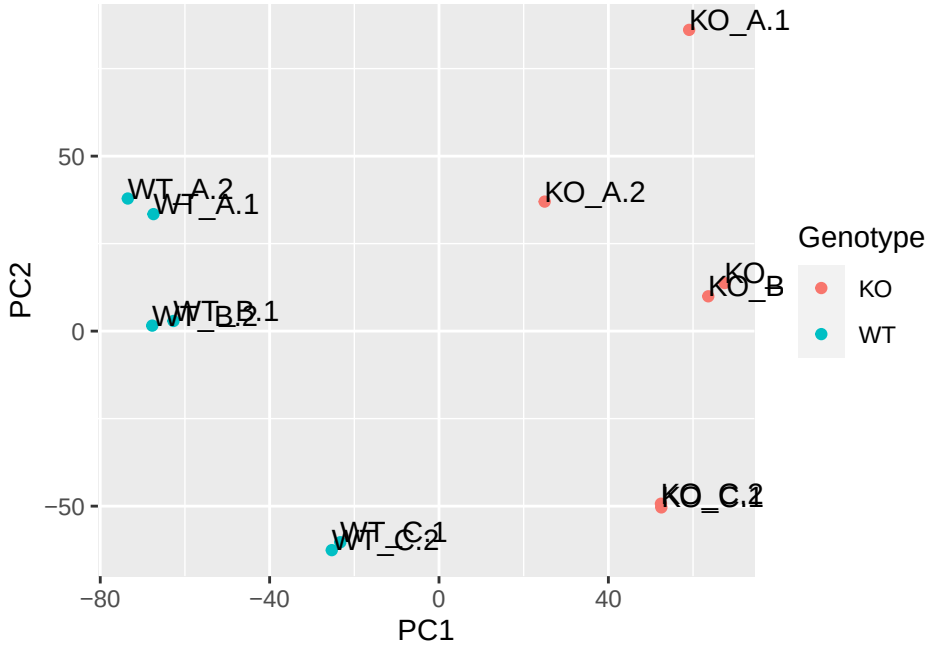
```
nrow(qlf.inter$table[qlf.inter$table$QValue<0.05,])
```

```
## [1] 4307
```

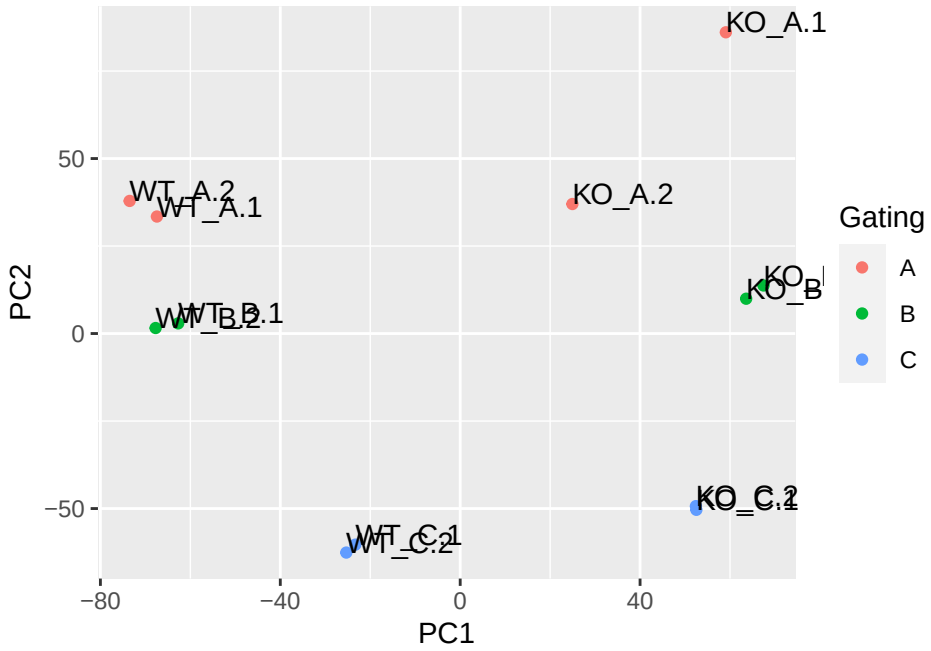
2.2 Filtering genes from two-way ANOVA

- Let's try a couple strategies for filtering genes from the two-way ANOVA

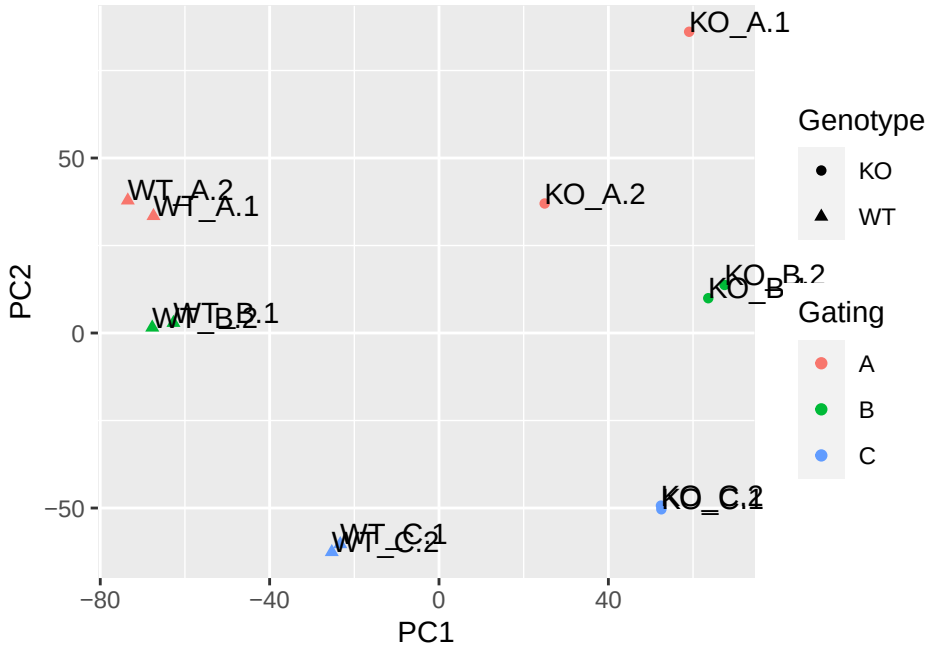
```
# load the ggplot2, heatmap, and circlize packages if you have restarted R
library(ggplot2)
library(ComplexHeatmap)
library(circlize)
# generate log-scaled normalized expression
norm <- cpm(genes, log=T)
# run PCA and plot with respect to each factor
pca <- prcomp(t(norm))
pca.data <- cbind(pca$x, metadata, Sample=rownames(metadata))
ggplot(pca.data,aes(x=PC1,y=PC2,label=Sample,color=Genotype)) +
  geom_point() + geom_text(hjust=0, vjust=0, show.legend=F, color="black") +
  coord_cartesian(clip='off')
```



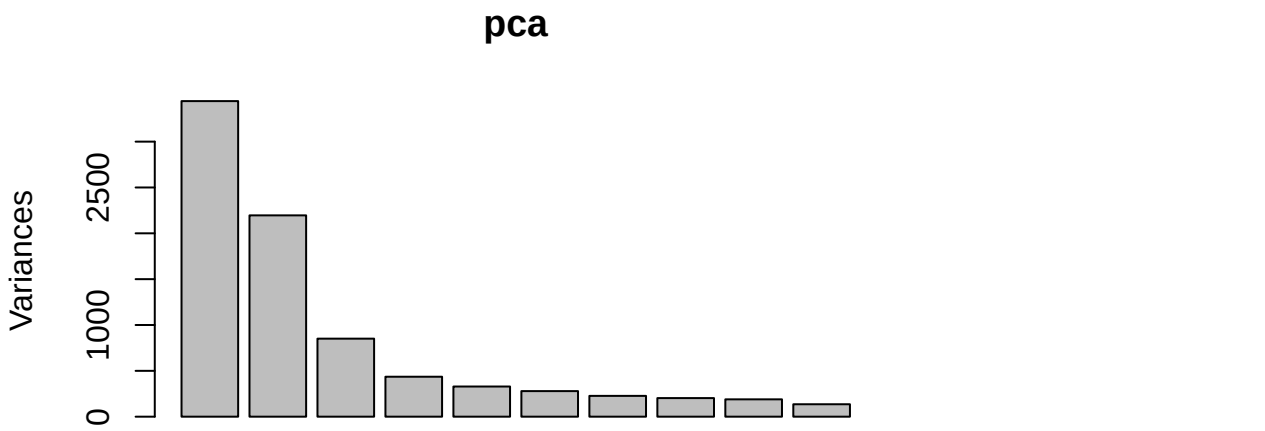

```
ggplot(pca.data,aes(x=PC1,y=PC2,label=Sample,color=Gating)) +
  geom_point() + geom_text(hjust=0, vjust=0, show.legend=F, color="black") +
  coord_cartesian(clip='off')
```




```
# Create combined PCA plot (Genotype as plot shape and Gating as color)
ggplot(pca.data,aes(x=PC1,y=PC2, label=Sample, color=Gating, shape=Genotype)) +
  geom_point() + geom_text(hjust=0, vjust=0, show.legend=F, color="black") +
  coord_cartesian(clip='off')
```




```
# check screeplot also
screeplot(pca)
```




```
# you can choose to add x and y labels with the % variance if you want
```

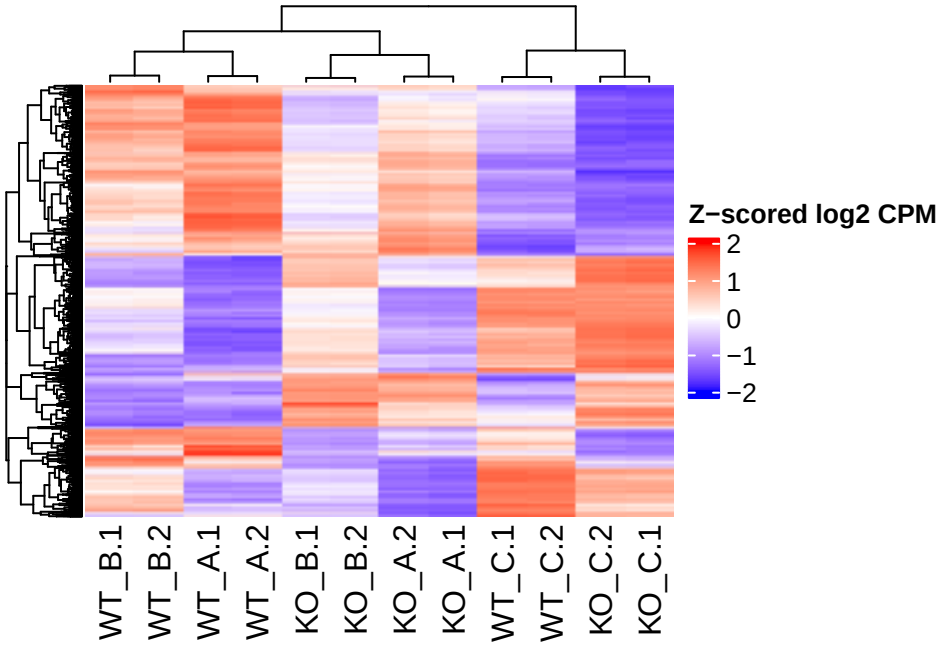
Now let's try some heatmaps, using a couple different filtering and normalization strategies.

First, we'll just make a plot of all of the DEGs, based on any of the tests (we use a very stringent q-value cutoff to make the number of results more manageable).

```
# First, let's plot all differentially expressed gene
# Use a very stringent q-value so that we don't have too many genes to plot
degs.diff <- qlf.geno$table[qlf.geno$table$QValue < 1e-7 |
   qlf.gate$table$QValue < 1e-7 | qlf.inter$table$QValue<1e-7,]
dim(degs.diff)
```

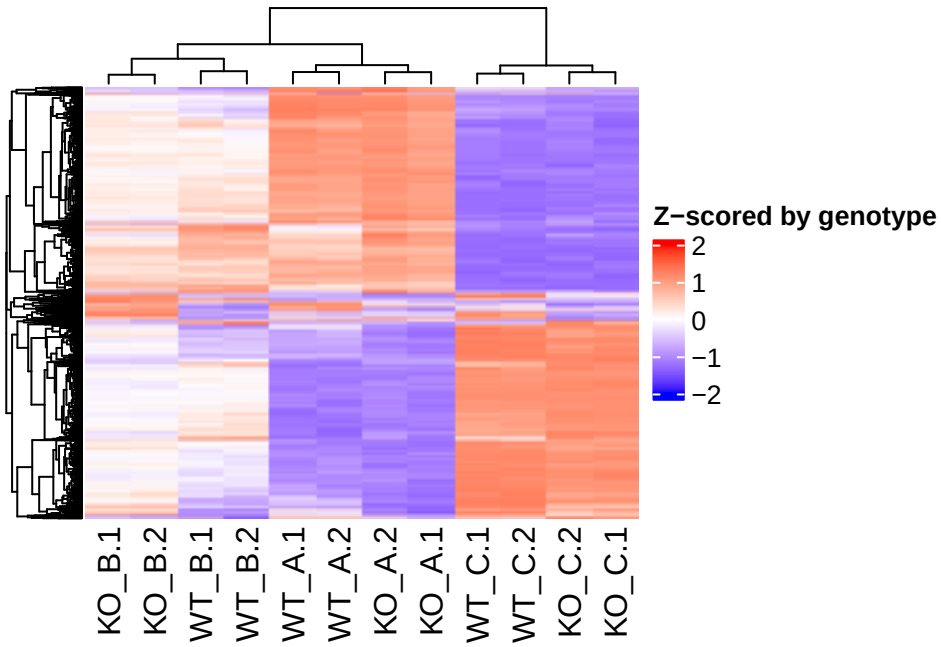
```
## [1] 3583    5
```

```
# subset the log-scaled CPMs to get the same genes
degs.norm <- norm[rownames(degs.diff),]
# z-score
degs.norm.z <- t(scale(t(degs.norm)))
# scale the colors from blue to white to red, with max/min at +/-2 at the middle at 0
col_fun <- colorRamp2(c(-2,0,2),c("blue","white","red"))
# heatmap of all differentially expressed genes
Heatmap(degs.norm.z,show_row_names=F,name="Z-scored log2 CPM",col=col_fun)
```



Now do a second heatmap of the **exact same genes**, but we're going to z-score separately by genotype. This means that the patterns we see have been controlled for baseline genotype differences.

```
# now let's z-score separately for each genotype
degs.WT <- degs.norm[,geno=="WT"]
degs.KO <- degs.norm[,geno=="KO"]
degs.WT.z <- t(scale(t(degs.WT)))
degs.KO.z <- t(scale(t(degs.KO)))
# combine the tables back together and make a new heatmap
degs.norm.z.alt <- cbind(degs.WT.z,degs.KO.z)
# the same genes as before, but z-scored separately for each genotype
# within each gating strategy, WT and KO shuold look pretty much the same
Heatmap(degs.norm.z.alt,show_row_names=F,name="Z-scored by genotype",col=col_fun)
```



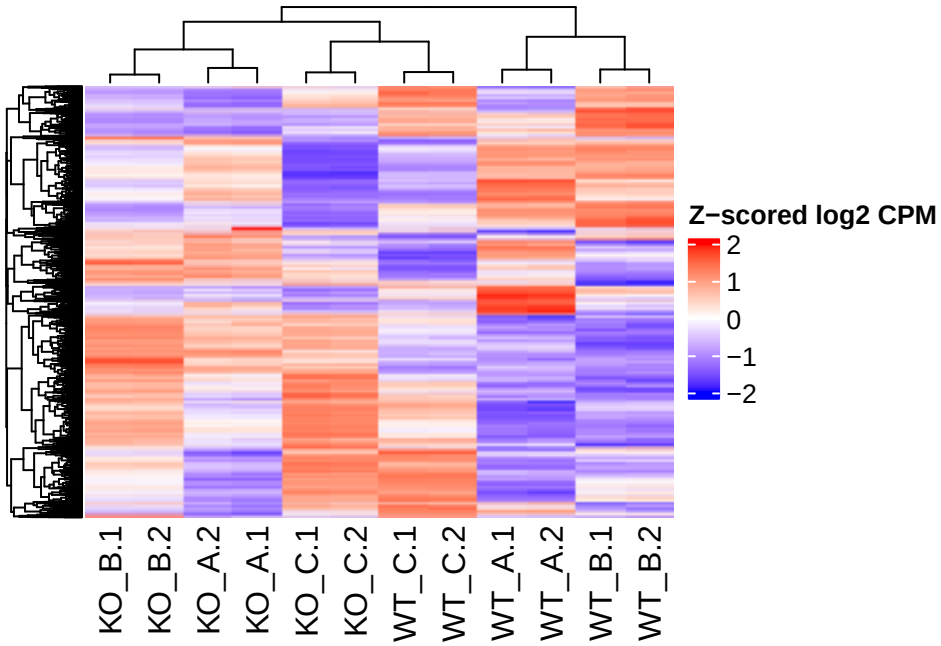
Note that now we primarly see the differences between gating strategies.

For our third heatmap, we will focus specfically on the genes with a significant interaction term. If we compare across gating strategies, we should see *different* patterns for WT and KO genotypes.

```
# finally, let's make a heatmap from just the interaction term
degs.diff.inter <- qlf.geno$table[qlf.inter$table$QValue<0.05,]
dim(degs.diff.inter)
```

```
## [1] 4307    5
```

```
# subset the log-scaled CPMs to get the same genes
degs.inter <- norm[rownames(degs.diff.inter),]
# z-score
degs.inter.z <- t(scale(t(degs.inter)))
# heatmap of genes with a significant interaction term
Heatmap(degs.inter.z,show_row_names=F,name="Z-scored log2 CPM",col=col_fun)
```



These are some examples of how we would use different filtering criteria and/or normalization strategies to address different questions.

2.3 Repeated measures differential

- **NOTE**: The columns of the data and the rows of the metadata are not in the same order for this data set. We will use the same trick to reorder the data within R.

ABOUT: These data are from an RNA-seq experiment where brain tissue was collected from four different sites from the same six animals. We'll run a repeated measures test controlling for animal-to-animal differences and testing for differences between tissues.

```
# read in counts
data <- read.table("https://uic-ric.github.io/workshop-data/edgeR/repeated_counts.txt",
   header=T,row.names=1)
dim(data)
```

```
## [1] 38445    24
```

```
head(data)
```

```
##           S19 S1 S20 S21 S22 S23 S24 S2 S37 S38 S39 S3 S40 S41 S42 S4 S55 S56
## 108349478   0  0   0   0   0   2   0  0   0   0   0  0   1   0   0  0   0   0
## 103690911   0  0   0   0   0   0   0  0   0   0   0  0   0   0   0  0   0   0
## 103690912  15  5   7   5  13  17   7  3   7   5   6  1  10   4   1  4   4   4
## 108349479 137 66 114 124 121 139 117 57 110  83  83 54 119 101  92 74  83  90
## 102556098  12  8   9  13  21  10  11 24  16  11   3  9  27  10  10 19  10   8
## 103690072   0  0   0   0   0   0   0  0   0   0   0  0   0   0   0  0   0   0
##           S57 S58 S59  S5 S60 S6
## 108349478   0   0   0   0   0  0
## 103690911   0   0   0   0   0  0
## 103690912   0   0   2   5   5  4
## 108349479  66  77  49 104  86 64
## 102556098   2  10   9  28   6 11
## 103690072   0   0   0   0   0  0
```

```
# read in metadata table
metadata <- read.table("https://uic-ric.github.io/workshop-data/edgeR/repeated_metadata.txt",
   header=T,row.names=1)
metadata
```

```
##      Tissue Individual
## S1  Tissue4  Animal-57
## S2  Tissue4  Animal-58
## S3  Tissue4  Animal-59
## S4  Tissue4  Animal-60
## S5  Tissue4  Animal-61
## S6  Tissue4  Animal-62
## S19 Tissue1  Animal-57
## S20 Tissue1  Animal-58
## S21 Tissue1  Animal-59
## S22 Tissue1  Animal-60
## S23 Tissue1  Animal-61
## S24 Tissue1  Animal-62
## S37 Tissue2  Animal-57
## S38 Tissue2  Animal-58
## S39 Tissue2  Animal-59
## S40 Tissue2  Animal-60
## S41 Tissue2  Animal-61
## S42 Tissue2  Animal-62
## S55 Tissue3  Animal-57
```

```
## S56 Tissue3  Animal-58
## S57 Tissue3  Animal-59
## S58 Tissue3  Animal-60
## S59 Tissue3  Animal-61
## S60 Tissue3  Animal-62
```

```
# confirm that we have each tissue in each animal
table(metadata)
```

```
##          Individual
## Tissue    Animal-57 Animal-58 Animal-59 Animal-60 Animal-61 Animal-62
##   Tissue1         1         1         1         1         1         1
##   Tissue2         1         1         1         1         1         1
##   Tissue3         1         1         1         1         1         1
##   Tissue4         1         1         1         1         1         1
```

```
# make sure the metadata row order matches the data column order
data <- data[,rownames(metadata)]
# check the data again to make sure we didn't drop any columns!
dim(data)
```

```
## [1] 38445    24
```

```
# subset counts to genes with >20 total counts
data_subset <- data[rowSums(data)>20,]
dim(data_subset)
```

```
## [1] 22865    24
```

```
# define our factors and model matrix
# don't include an interaction term
tissue <- factor(metadata[,1])
subject <- factor(metadata[,2])
# Try estimating dispersion with and without controlling for subject.
model1 <- model.matrix(~ tissue + subject)
model1
```

```
##    (Intercept) tissueTissue2 tissueTissue3 tissueTissue4 subjectAnimal-58
## 1            1             0             0             1                 0
## 2            1             0             0             1                 1
## 3            1             0             0             1                 0
## 4            1             0             0             1                 0
## 5            1             0             0             1                 0
## 6            1             0             0             1                 0
## 7            1             0             0             0                 0
## 8            1             0             0             0                 1
## 9            1             0             0             0                 0
## 10           1             0             0             0                 0
## 11           1             0             0             0                 0
## 12           1             0             0             0                 0
## 13           1             1             0             0                 0
## 14           1             1             0             0                 1
## 15           1             1             0             0                 0
## 16           1             1             0             0                 0
## 17           1             1             0             0                 0
## 18           1             1             0             0                 0
## 19           1             0             1             0                 0
## 20           1             0             1             0                 1
```

```
## 21           1             0             1             0                0
## 22           1             0             1             0                0
## 23           1             0             1             0                0
## 24           1             0             1             0                0
##    subjectAnimal-59 subjectAnimal-60 subjectAnimal-61 subjectAnimal-62
## 1                 0                0                0                0
## 2                 0                0                0                0
## 3                 1                0                0                0
## 4                 0                1                0                0
## 5                 0                0                1                0
## 6                 0                0                0                1
## 7                 0                0                0                0
## 8                 0                0                0                0
## 9                 1                0                0                0
## 10                0                1                0                0
## 11                0                0                1                0
## 12                0                0                0                1
## 13                0                0                0                0
## 14                0                0                0                0
## 15                1                0                0                0
## 16                0                1                0                0
## 17                0                0                1                0
## 18                0                0                0                1
## 19                0                0                0                0
## 20                0                0                0                0
## 21                1                0                0                0
## 22                0                1                0                0
## 23                0                0                1                0
## 24                0                0                0                1
## attr(,"assign")
## [1] 0 1 1 1 2 2 2 2 2
## attr(,"contrasts")
## attr(,"contrasts")$tissue
## [1] "contr.treatment"
##
## attr(,"contrasts")$subject
## [1] "contr.treatment"
```

```
model2 <- model.matrix(~ tissue)
model2
```

```
##    (Intercept) tissueTissue2 tissueTissue3 tissueTissue4
## 1            1             0             0             1
## 2            1             0             0             1
## 3            1             0             0             1
## 4            1             0             0             1
## 5            1             0             0             1
## 6            1             0             0             1
## 7            1             0             0             0
## 8            1             0             0             0
## 9            1             0             0             0
## 10           1             0             0             0
## 11           1             0             0             0
## 12           1             0             0             0
## 13           1             1             0             0
```

```
## 14          1             1             0             0
## 15          1             1             0             0
## 16          1             1             0             0
## 17          1             1             0             0
## 18          1             1             0             0
## 19          1             0             1             0
## 20          1             0             1             0
## 21          1             0             1             0
## 22          1             0             1             0
## 23          1             0             1             0
## 24          1             0             1             0
## attr(,"assign")
## [1] 0 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$tissue
## [1] "contr.treatment"
```

```
# create edgeR object
genes <- DGEList(counts=data_subset)
# calculate TMM factors
genes <- calcNormFactors(genes)
# estimate dispersion, for both models
genes1 <- estimateDisp(genes,model1)
genes2 <- estimateDisp(genes,model2)
# calculate BCV from the common dispersion
sqrt(genes1$common.dispersion)
```

```
## [1] 0.1219146
```

```
sqrt(genes2$common.dispersion)
```

```
## [1] 0.1479583
```

```
# proceed with model1 (controlling for subject)
# fit the model
fit <- glmQLFit(genes1,model1)
# just run a test for the tissue coefficients
qlf.tissue <- glmQLFTest(fit, coef=2:4)
# do p-value adjustment
qlf.tissue$table$QValue <- p.adjust(qlf.tissue$table$PValue,method="BH")
# check number of significant genes
nrow(qlf.tissue$table[qlf.tissue$table$QValue<0.05,])
```

```
## [1] 17939
```

2.4 Batch effect correction

ABOUT: These data are from an RNA-seq experiment comparing B cells from five mouse genotypes, with three replicates per group. However, the replicates were collected at different times, so there is a batch effect present, especially for replicate 1.

```
# reload the edgeR library in case you closed R studio
library(edgeR)
# read in counts
data <- read.table("https://uic-ric.github.io/workshop-data/edgeR/batch_counts.txt",
   header=T,row.names=1)
dim(data)
```

```
## [1] 23366    15
```

```
head(data)
```

```
##               A.1 A.2 A.3 B.1 B.2 B.3 C.1 C.2 C.3 D.1 D.2 D.3 E.1 E.2 E.3
## 0610005C13Rik   4   1  10   0   3  27  11  38  12   1   4  25   9   2  27
## 0610007N19Rik   0   0   0   2   0   0   0   0   0   0   0   0   0   0   0
## 0610007P14Rik 249  94 126 746 186 180 357 461 301 890 143 251 240 156 200
## 0610008F07Rik   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 0610009B14Rik   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
## 0610009B22Rik 145  45  93 272 131 141 161 256 200 388  77 155  87  93 137
```

```
# read in metadata table
metadata <- read.table("https://uic-ric.github.io/workshop-data/edgeR/batch_metadata.txt",
   header=T,row.names=1)
metadata
```

```
##     Genotype Batch
## A.1        A     1
## A.2        A     2
## A.3        A     3
## B.1        B     1
## B.2        B     2
## B.3        B     3
## C.1        C     1
## C.2        C     2
## C.3        C     3
## D.1        D     1
## D.2        D     2
## D.3        D     3
## E.1        E     1
## E.2        E     2
## E.3        E     3
```

```
# make sure the metadata row order matches the data column order
data <- data[,rownames(metadata)]
# check the data again to make sure we didn't drop any columns!
dim(data)
```

```
## [1] 23366    15
```

```
# subset counts to genes with >20 total counts
data_subset <- data[rowSums(data)>20,]
dim(data_subset)
```

```
## [1] 14037    15
```

```
# define our factors
geno <- factor(metadata[,1])
batch <- factor(metadata[,2])
```

First, let's try the analysis without the batch correction

```
# just model the genotype
model <- model.matrix(~ geno)
model
```

```
##    (Intercept) genoB genoC genoD genoE
## 1            1     0     0     0     0
## 2            1     0     0     0     0
## 3            1     0     0     0     0
## 4            1     1     0     0     0
## 5            1     1     0     0     0
## 6            1     1     0     0     0
## 7            1     0     1     0     0
## 8            1     0     1     0     0
## 9            1     0     1     0     0
## 10           1     0     0     1     0
## 11           1     0     0     1     0
## 12           1     0     0     1     0
## 13           1     0     0     0     1
## 14           1     0     0     0     1
## 15           1     0     0     0     1
## attr(,"assign")
## [1] 0 1 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$geno
## [1] "contr.treatment"
```

```
# create edgeR object
genes <- DGEList(counts=data_subset)
# calculate TMM factors
genes <- calcNormFactors(genes)
# estimate dispersion
genes <- estimateDisp(genes,model)
# calculate BCV from the common dispersion
sqrt(genes$common.dispersion)
```

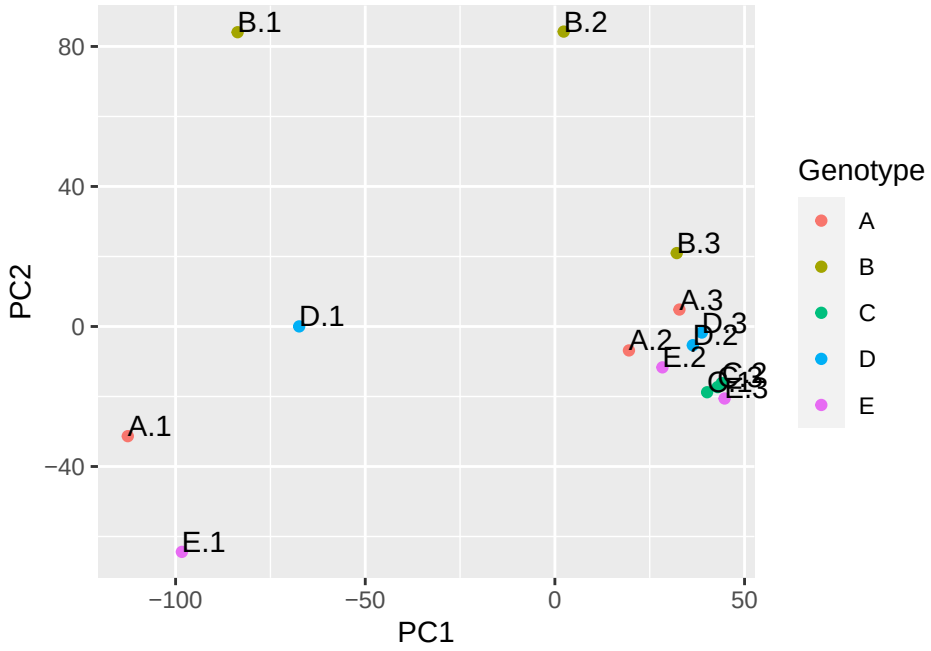
```
## [1] 0.3957431
```

```
# fit the model
fit <- glmQLFit(genes,model)
# run ANOVA: test all coefficients other than the intercept
qlf <- glmQLFTest(fit, coef=2:5)
# do p-value adjustment
qlf$table$QValue <- p.adjust(qlf$table$PValue,method="BH")
# check number of significant genes
nrow(qlf$table[qlf$table$QValue<0.05,])
```

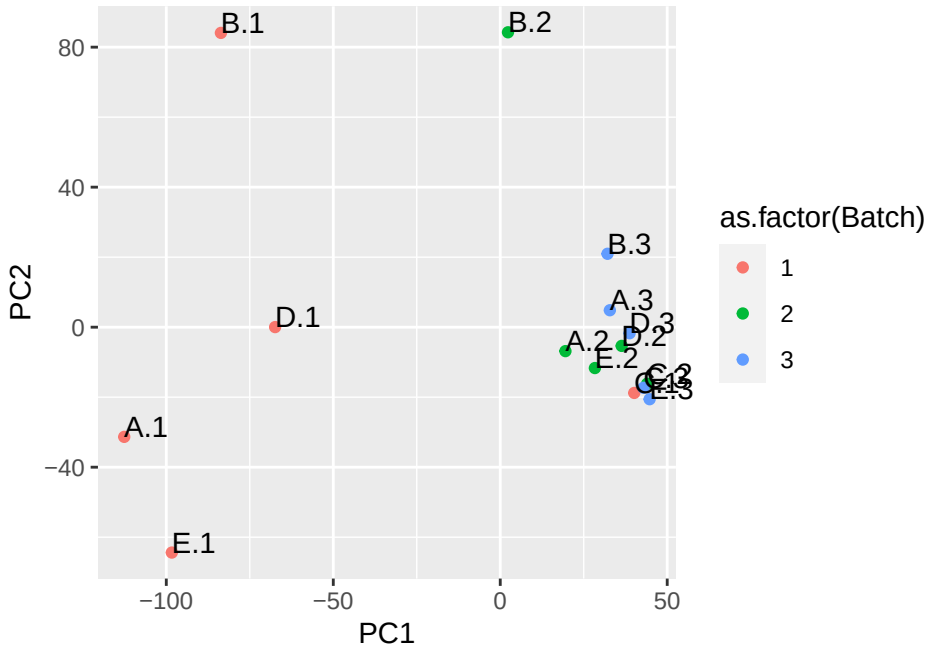
```
## [1] 1528
```

```
# generate log-scaled normalized expression
norm <- cpm(genes, log=T)
# make colors for each factor for plotting purposes
```

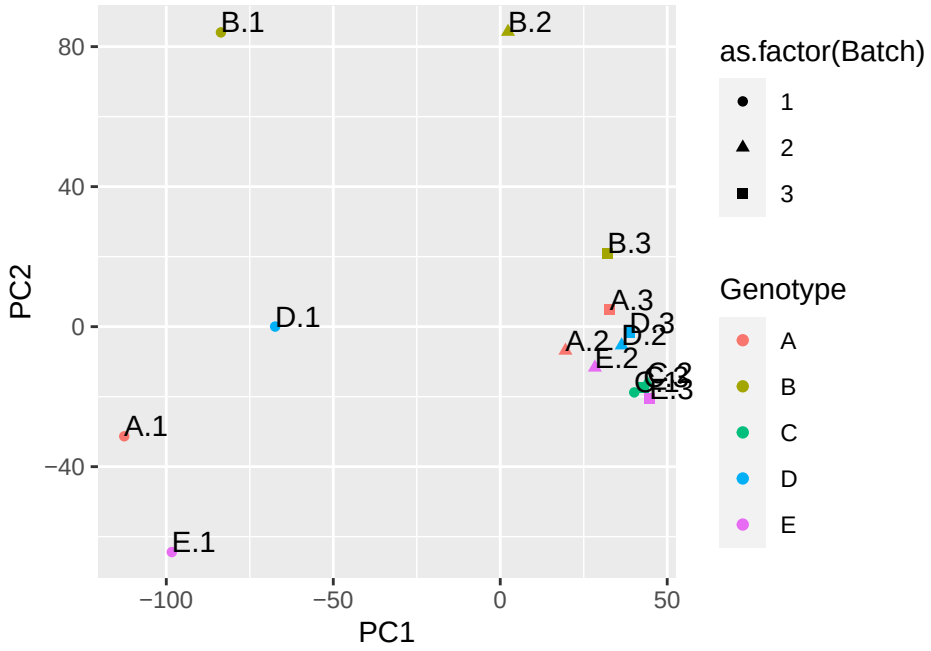
```
# we're going to use the trick again by renaming the factor levels
# genotype colors
geno.colors <- geno
levels(geno.colors) = rainbow(length(levels(geno.colors)))
# run PCA
pca <- prcomp(t(norm))
pca.data <- cbind(pca$x, metadata, Sample=rownames(metadata))
ggplot(pca.data,aes(x=PC1,y=PC2)) +
  geom_point(aes(color=Genotype)) +
  geom_text(aes(label=Sample), hjust=0, vjust=0, show.legend=F) +
  coord_cartesian(clip='off')
```




```
ggplot(pca.data,aes(x=PC1,y=PC2)) +
  geom_point(aes(color=as.factor(Batch))) +
  geom_text(aes(label=Sample), hjust=0, vjust=0, show.legend=F) +
  coord_cartesian(clip='off')
```




```
ggplot(pca.data,aes(x=PC1,y=PC2)) +
  geom_point(aes(shape=as.factor(Batch), color=Genotype)) +
  geom_text(aes(label=Sample), hjust=0, vjust=0, show.legend=F) +
  coord_cartesian(clip='off')
```



Note that replicate 1 samples are to the left, and other samples are to the right. This separation is along PC1, which is capturing the biggest portion of the variance. So, there is a batch effect, and it's pretty strong.

Now, with the batch effect correction

```
# new model matrix with batch factor; no interaction term
batch.model <- model.matrix(~ geno + batch)
batch.model
```

```
##    (Intercept) genoB genoC genoD genoE batch2 batch3
## 1            1     0     0     0     0      0      0
## 2            1     0     0     0     0      1      0
## 3            1     0     0     0     0      0      1
## 4            1     1     0     0     0      0      0
## 5            1     1     0     0     0      1      0
## 6            1     1     0     0     0      0      1
## 7            1     0     1     0     0      0      0
## 8            1     0     1     0     0      1      0
## 9            1     0     1     0     0      0      1
## 10           1     0     0     1     0      0      0
## 11           1     0     0     1     0      1      0
## 12           1     0     0     1     0      0      1
## 13           1     0     0     0     1      0      0
## 14           1     0     0     0     1      1      0
## 15           1     0     0     0     1      0      1
## attr(,"assign")
## [1] 0 1 1 1 1 2 2
## attr(,"contrasts")
## attr(,"contrasts")$geno
## [1] "contr.treatment"
##
## attr(,"contrasts")$batch
## [1] "contr.treatment"
```

```
# create edgeR object
batch.genes <- DGEList(counts=data_subset)
# calculate TMM factors
batch.genes <- calcNormFactors(batch.genes)
# estimate dispersion
batch.genes <- estimateDisp(batch.genes,batch.model)
# calculate BCV from the common dispersion
# should be much smaller!
sqrt(batch.genes$common.dispersion)
```

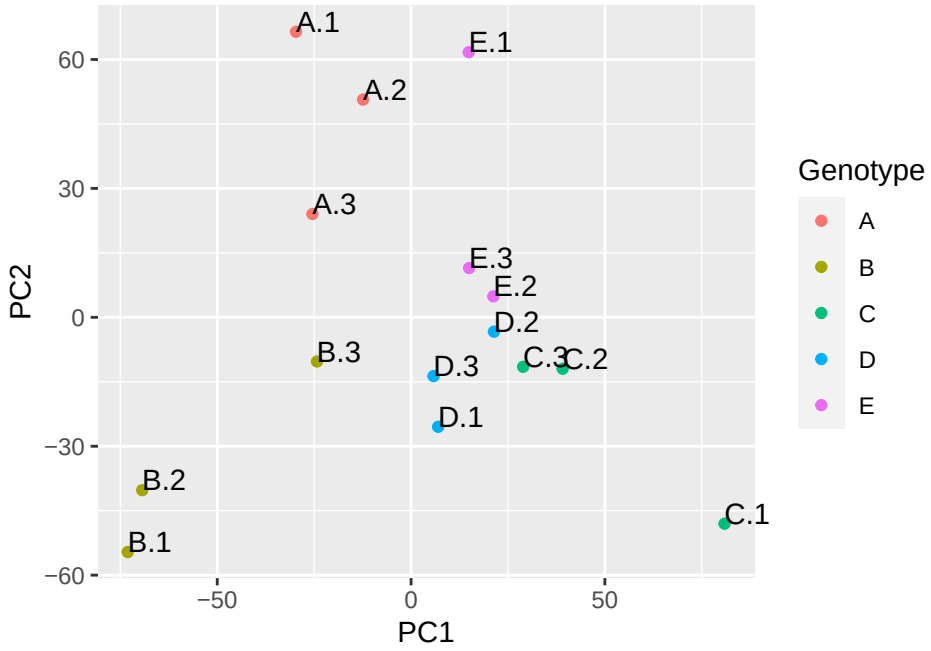
```
## [1] 0.2762293
```

```
# fit the model
batch.fit <- glmQLFit(batch.genes,batch.model)
# test again for the genotype effect - this test looks the same as before
# the batch terms are just "along for the ride"
batch.qlf <- glmQLFTest(batch.fit, coef=2:5)
# do p-value adjustment
batch.qlf$table$QValue <- p.adjust(batch.qlf$table$PValue,method="BH")
# check number of significant genes
# should be many more!
nrow(batch.qlf$table[batch.qlf$table$QValue<0.05,])
```

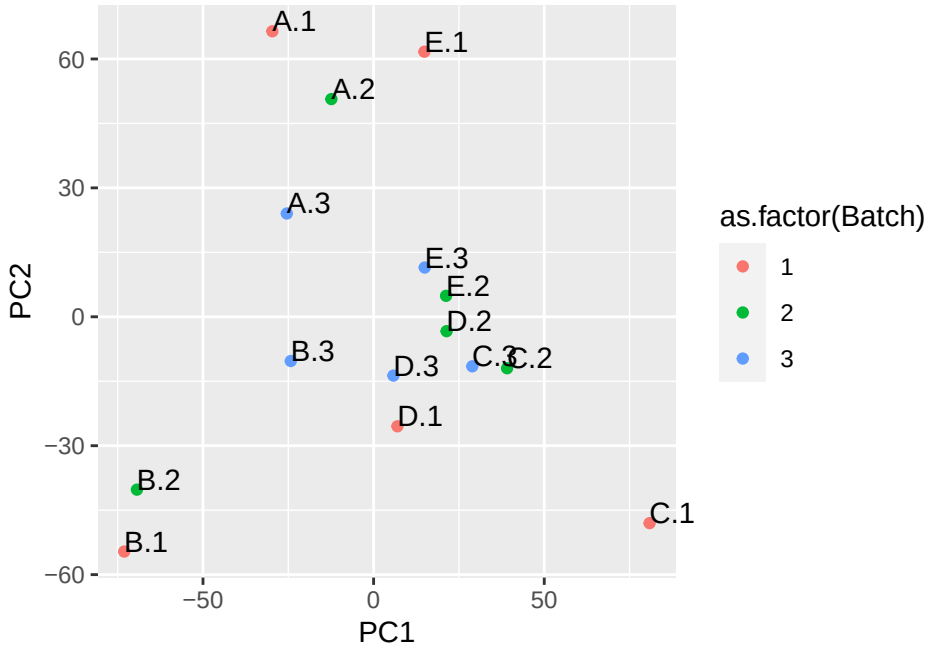
```
## [1] 2564
```

```
# correct for batch effect: we just need to give the batch factor vector
# we can start with the same normalized expression as above
batch.norm <- removeBatchEffect(norm, batch)
# run PCA
batch.pca <- prcomp(t(batch.norm))
batch.pca.data <- cbind(batch.pca$x, metadata, Sample=rownames(metadata))
```

```
# this looks better, separation based on batch is gone
ggplot(batch.pca.data,aes(x=PC1,y=PC2)) +
  geom_point(aes(color=Genotype)) +
  geom_text(aes(label=Sample), hjust=0, vjust=0, show.legend=F) +
  coord_cartesian(clip='off')
```




```
ggplot(batch.pca.data,aes(x=PC1,y=PC2)) +
  geom_point(aes(color=as.factor(Batch))) +
  geom_text(aes(label=Sample), hjust=0, vjust=0, show.legend=F) +
  coord_cartesian(clip='off')
```



2.5 Analysis with a continuous variable

ABOUT: This is a clinical data set (46 samples), where some patients had a disease and others did not.

- The patients were also scored according to a separate phenotype.
- We are looking for genes that are different according to the disease, or the phenotype, or based on a combination (interaction) between these.

```
# read in counts
data <- read.table("https://uic-ric.github.io/workshop-data/edgeR/cont_counts.txt",
   header=T,row.names=1)
dim(data)
```

```
## [1] 57230    46
```

```
# read in metadata table
metadata <- read.table("https://uic-ric.github.io/workshop-data/edgeR/cont_metadata.txt",
   header=T,row.names=1)
head(metadata)
```

```
##          Phenotype_Score Disease
## sample50              82   FALSE
## sample51              90   FALSE
## sample52              65    TRUE
## sample53              90    TRUE
## sample54              49    TRUE
## sample55              43    TRUE
```

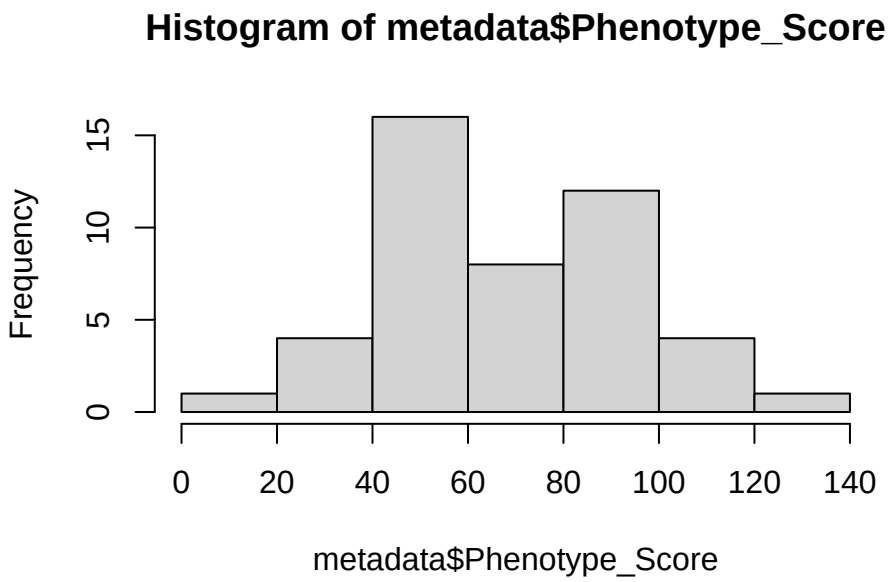
```
# make sure the metadata row order matches the data column order
data <- data[,rownames(metadata)]
# check the data again to make sure we didn't drop any columns!
dim(data)
```

```
## [1] 57230    46
```

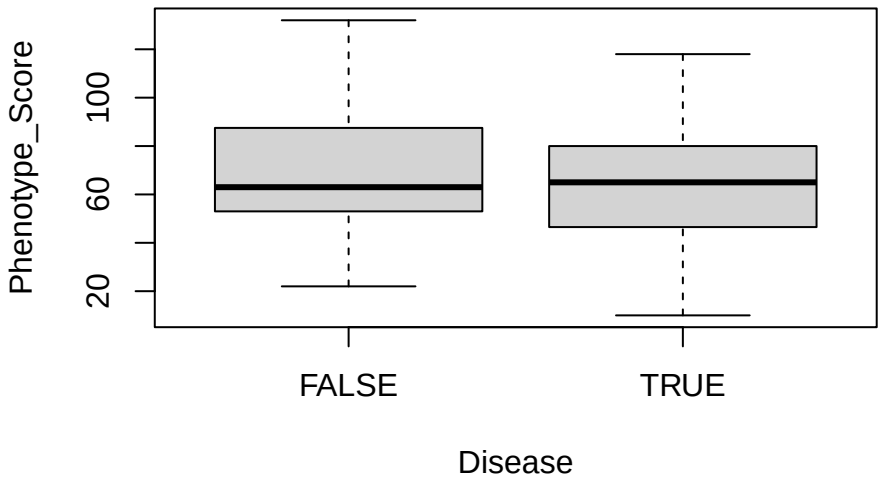
```
# subset counts to genes with >20 total counts
data_subset <- data[rowSums(data)>20,]
dim(data_subset)
```

```
## [1] 19827    46
```

```
# first let's take a look at the scores a bit more
# check a histogram of the scores
hist(metadata$Phenotype_Score)
```




```
# check if there's an association between score and disease
boxplot(Phenotype_Score ~ Disease, data=metadata)
```




```
kruskal.test(Phenotype_Score ~ Disease, data=metadata)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  Phenotype_Score by Disease
## Kruskal-Wallis chi-squared = 0.27836, df = 1, p-value = 0.5978
```

This looks OK: if there was a strong association (small p-value), then the variables would be confounded and we couldn't test them independently. So we can continue:

```
# define our variable/factor and model matrix
# use as.numeric for the score
score <- as.numeric(metadata[,1])
disease <- factor(metadata[,2])
model <- model.matrix(~ score + disease + score:disease)
head(model)
```

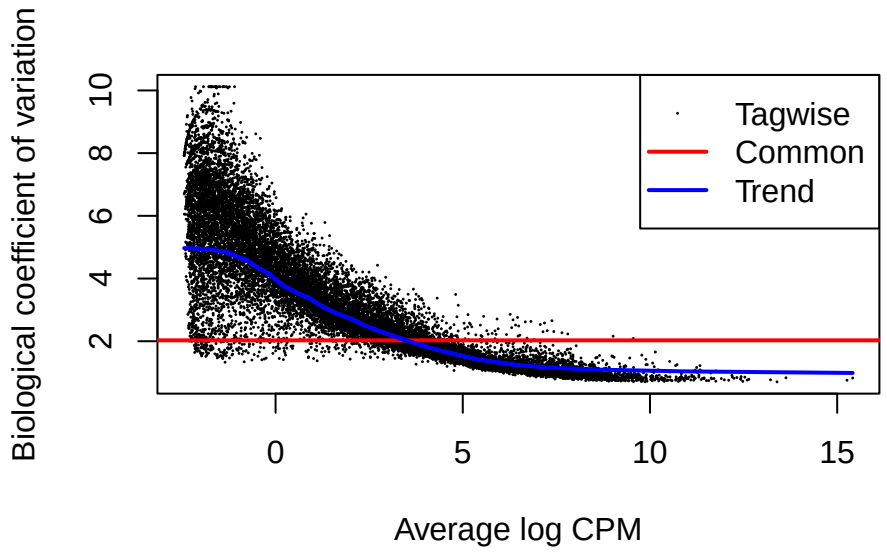
```
##   (Intercept) score diseaseTRUE score:diseaseTRUE
```

```
## 1           1    82           0                0
## 2           1    90           0                0
## 3           1    65           1               65
## 4           1    90           1               90
## 5           1    49           1               49
## 6           1    43           1               43
```

```
# create edgeR object
genes <- DGEList(counts=data_subset)
# calculate TMM factors
genes <- calcNormFactors(genes)
# estimate dispersion
# this may take a minute - there are a lot of samples!
genes <- estimateDisp(genes,model)
# calculate BCV from the common dispersion
# note the BCV is much higher in this data set
# clinical data sets are often highly variable!
sqrt(genes$common.dispersion)
```

```
## [1] 2.027726
```

```
plotBCV(genes)
```




```
# fit the model
fit <- glmQLFit(genes,model)
# now we test each variable/factor in turn
# the continuous variable is always just 1 term
qlf.score <- glmQLFTest(fit, coef=2)
qlf.disease <- glmQLFTest(fit, coef=3)
qlf.inter <- glmQLFTest(fit, coef=4)
# do p-value adjustment
qlf.score$table$QValue <- p.adjust(qlf.score$table$PValue,method="BH")
qlf.disease$table$QValue <- p.adjust(qlf.disease$table$PValue,method="BH")
qlf.inter$table$QValue <- p.adjust(qlf.inter$table$PValue,method="BH")
# check number of significant genes due to:
# score
nrow(qlf.score$table[qlf.score$table$QValue<0.05,])
```

```
## [1] 1964
```

```
# disease
nrow(qlf.disease$table[qlf.disease$table$QValue<0.05,])
```

```
## [1] 1114
```

```
# interaction term
nrow(qlf.inter$table[qlf.inter$table$QValue<0.05,])
```

```
## [1] 847
```

2.6 Analysis with no replicates

- **PLEASE AVOID NEEDING TO DO THIS:** you should **always** have biological replicates.
- But if you absolutely must analyze the data without replicates, at least do it by setting a fixed dispersion and computing approximate p-values, rather than simply looking at the fold-change. Using a fold-change threshold will bias you towards low-expressed genes.
- Note that the significance level of the q-values from this analysis are *very* dependent on the specific BCV you choose. However, it is fair to use the q-value to *prioritize* the genes based on which appear to be most strongly affected.

```
# read in counts
data <- read.table("https://uic-ric.github.io/workshop-data/edgeR/norep_counts.txt",
   header=T,row.names=1)
dim(data)
```

```
## [1] 24421     2
```

```
# read in metadata table
metadata <- read.table("https://uic-ric.github.io/workshop-data/edgeR/norep_metadata.txt",
   header=T,row.names=1)
metadata
```

```
##         Condition
## Control   Control
## Treat     Treated
```

```
# make sure the metadata row order matches the data column order
data <- data[,rownames(metadata)]
# check the data again to make sure we didn't drop any columns!
dim(data)
```

```
## [1] 24421     2
```

```
# subset counts to genes with >20 total counts
data_subset <- data[rowSums(data)>20,]
dim(data_subset)
```

```
## [1] 14947     2
```

```
# create edgeR object
genes <- DGEList(counts=data_subset, group=metadata[,1])
# calculate TMM factors
genes <- calcNormFactors(genes)
# instead of estimating dispersion, we'll just set it
# let's assume a BCV of 0.2 (20%)
bcv <- 0.2
# run differential, specifying the dispersion to use
stats <- exactTest(genes, dispersion=bcv^2)
# run FDR correction
stats$table$QValue <- p.adjust(stats$table$PValue,method="BH")
# sort by p-value to prioritize top genes
head(stats$table[order(stats$table$PValue),])
```

```
##          logFC   logCPM       PValue       QValue
## Ccl2  7.953027 8.449673 2.155760e-43 3.222214e-39
## Ly6c2 7.812661 8.354315 1.700712e-42 1.271027e-38
## Ccl5  7.711836 7.462217 4.876864e-40 2.429816e-36
## Mx1   8.615513 6.145622 5.191160e-38 1.637090e-34
## Gbp2  7.223603 7.725409 5.476317e-38 1.637090e-34
```

```
## Ccl7  8.580917 6.111356 9.357070e-38 2.331002e-34
```

3 Extra (Take Home) Exercises

3.1 Gene ID conversion with biomaRt

Use the biomaRt package in from bioconductor. For first time use we need to install it.

```
BiocManager::install("biomaRt")
```

```
library(biomaRt)
data <- read.table("https://uic-ric.github.io/workshop-data/edgeR/mouse_ensembl_rnaseq.txt",
   header=T,row.names=1)
head(data)
```

```
##                       Sample_001 Sample_002 Sample_003 Sample_004 Sample_005
## ENSMUSG00000051951.5          42         36         84         43         85
## ENSMUSG00000102851.1           2          1          7          1         10
## ENSMUSG00000103147.1          15         11         12         27         16
## ENSMUSG00000102331.1          18         14         33         18         24
## ENSMUSG00000102343.1          11         15         37         22         17
## ENSMUSG00000025900.12        118         82        186         92        172
##                       Sample_006 Sample_007 Sample_008 Sample_009
## ENSMUSG00000051951.5          11         60         76         39
## ENSMUSG00000102851.1           0          1          8          3
## ENSMUSG00000103147.1           5         11         22         10
## ENSMUSG00000102331.1          12          7         15          7
## ENSMUSG00000102343.1          14         15         26         13
## ENSMUSG00000025900.12         29        134        168         96
```

Note that the identifiers are:

- Ensembl IDs (start with `ENS`).
- For mouse (`MUS`). If no three-letter code is shown, then they're human IDs.
- For genes, rather than isoforms/transcripts or proteins, etc. (`G`).
- Have version numbers (end in `.[number]`). Will need to remove versions before looking up symbols.

```
# get rid of version numbers in transcript names
#    this is is also necessary if you're going to upload to a pathway database
# this is an example of REGULAR EXPRESSIONS, which we've seen a little
# of also in grep and sed commands in linux. Very powerful!
head(rownames(data))
```

```
## [1] "ENSMUSG00000051951.5"  "ENSMUSG00000102851.1"  "ENSMUSG00000103147.1"
## [4] "ENSMUSG00000102331.1"  "ENSMUSG00000102343.1"  "ENSMUSG00000025900.12"
```

```
newnames <- gsub("\\.[0-9]*","",rownames(data))
head(newnames)
```

```
## [1] "ENSMUSG00000051951" "ENSMUSG00000102851" "ENSMUSG00000103147"
## [4] "ENSMUSG00000102331" "ENSMUSG00000102343" "ENSMUSG00000025900"
```

Now convert the IDs to gene symbols. Check the manual for the `biomaRt` package for more details. In particular:

- The searchDatasets() function lets you search for available databases (for use in useDataset()).
- The listAttributes() function lets you see what fields you can map your IDs to (for use in getBM()).

For today, we already know the database (`mmusculus_gene_ensembl`) and attributes we want (`ensembl_gene_id` and gene symbol AKA `external_gene_name`). Note that both of these commands may take some time to run, since they are fetching data from an online database hosted by Ensembl.

```
# get the database of mouse gene annotations
mart <- useDataset("mmusculus_gene_ensembl",useMart("ensembl"))
gene_list <- getBM(filters="ensembl_gene_id",
   attributes=c("ensembl_gene_id","external_gene_name"),
   values=newnames, mart=mart)
```

```
head(gene_list)
```

```
##      ensembl_gene_id external_gene_name
## 1 ENSMUSG00000000194             Gpr107
## 2 ENSMUSG00000000247               Lhx2
## 3 ENSMUSG00000000308              Ckmt1
## 4 ENSMUSG00000000359               Rem1
## 5 ENSMUSG00000000392                Fap
## 6 ENSMUSG00000000394                Gcg
```

```
# note that not ALL IDs may get matched
length(newnames)
```

```
## [1] 39056
```

```
nrow(gene_list)
```

```
## [1] 38535
```

```
# also note that gene symbols are NOT unique relative to the gene IDs
length(unique(gene_list[,1]))
```

```
## [1] 38535
```

```
length(unique(gene_list[,2]))
```

```
## [1] 38492
```

Now that you have the list, you can combine with an existing data frame. In the example below, we combine with the counts table using `merge()`.

NOTE: usually you would want to keep the original IDs as the main identifiers throughout your analysis in edgeR, and then merge the symbols or other information with your normalized expression (CPM) or differential stats tables AFTER running the analysis. You can't have a mix of symbols and other strings in your counts table when processing through edgeR.

```
# replace the row names with the IDs after stripping off the version numbers
rownames(data) = newnames
# merge based on ensembl_gene_id and rownames
# keep all entries. unmatched IDs will have NA values for the gene symbol
data.named <- merge(data, gene_list,
   by.x="row.names", by.y="ensembl_gene_id", all=T)
head(data.named)
```

```
##            Row.names Sample_001 Sample_002 Sample_003 Sample_004 Sample_005
## 1 ENSMUSG00000000001       5669       3717       4019       5172       3460
## 2 ENSMUSG00000000003          3          5         18          4         13
## 3 ENSMUSG00000000028       3741       3486       3304       3720       2917
## 4 ENSMUSG00000000031       5125       4549       2065       3236       6456
## 5 ENSMUSG00000000037         16         34         40         14         28
## 6 ENSMUSG00000000049         29         24         31         34         46
##   Sample_006 Sample_007 Sample_008 Sample_009 external_gene_name
## 1       6072       4498       2741       3967              Gnai3
```

```
## 2          2          8          4          8             Pbsn
## 3       4836       4990       2807       3761            Cdc45
## 4       4417       8112       2954       9762              H19
## 5         18         20         29         22            Scml2
## 6         18         56         48         18             Apoh
```

3.2 Advanced PCA plots

For this exercise, we'll use a shotgun metagenomics data set, with taxonomic quantification summarized at a phylum level. These are data obtained from a mouse testing a surgery injury model, and a treatment for the injury. We have two experimental factors: (1) Injury, with levels Naive, Sham, and Surgery; and (2) Treatment, with levels treated and control.

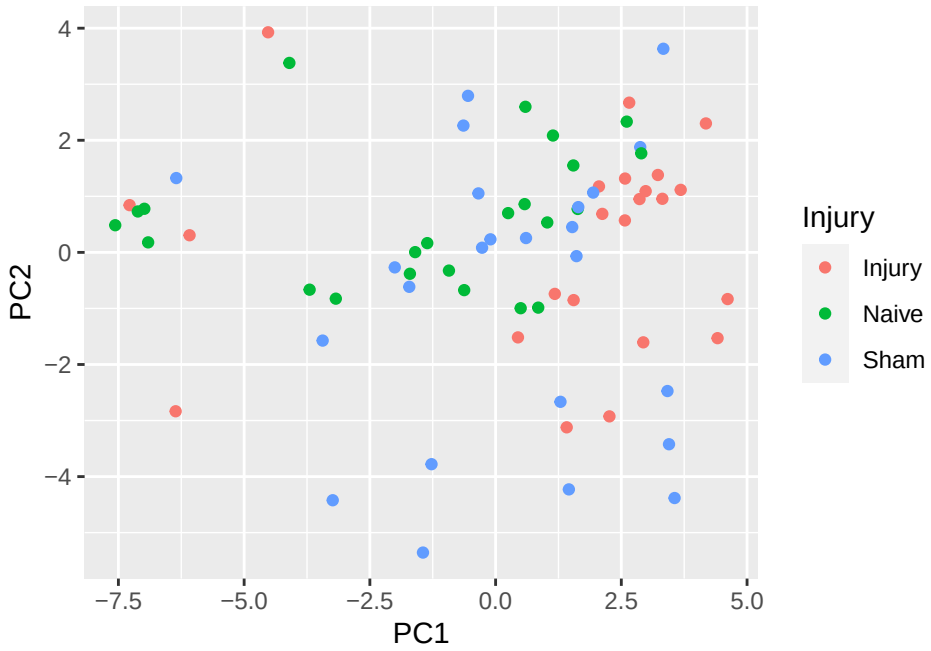
```
# load the edgeR and ggplot2 libraries
library(edgeR)
library(ggplot2)
# read in the counts data
data <- read.table("https://uic-ric.github.io/workshop-data/edgeR/metagenomics_counts.txt",
   sep="\t",header=T,row.names=1)
# observe that the features are taxonomic names, down to phylum level
# all are for bacteria
head(rownames(data))
```

```
## [1] "sk_Bacteria;k__Bacteria incertae sedis;p__Caldiserica"
## [2] "sk_Bacteria;k__Bacteria incertae sedis;p__Candidatus Levybacteria"
## [3] "sk_Bacteria;k__Bacteria incertae sedis;p__Candidatus Magasanikbacteria"
## [4] "sk_Bacteria;k__Bacteria incertae sedis;p__Thermotogae"
## [5] "sk_Bacteria;k__Bacteria incertae sedis;p__Candidatus Azambacteria"
## [6] "sk_Bacteria;k__Bacteria incertae sedis;p__Thermodesulfobacteria"
```

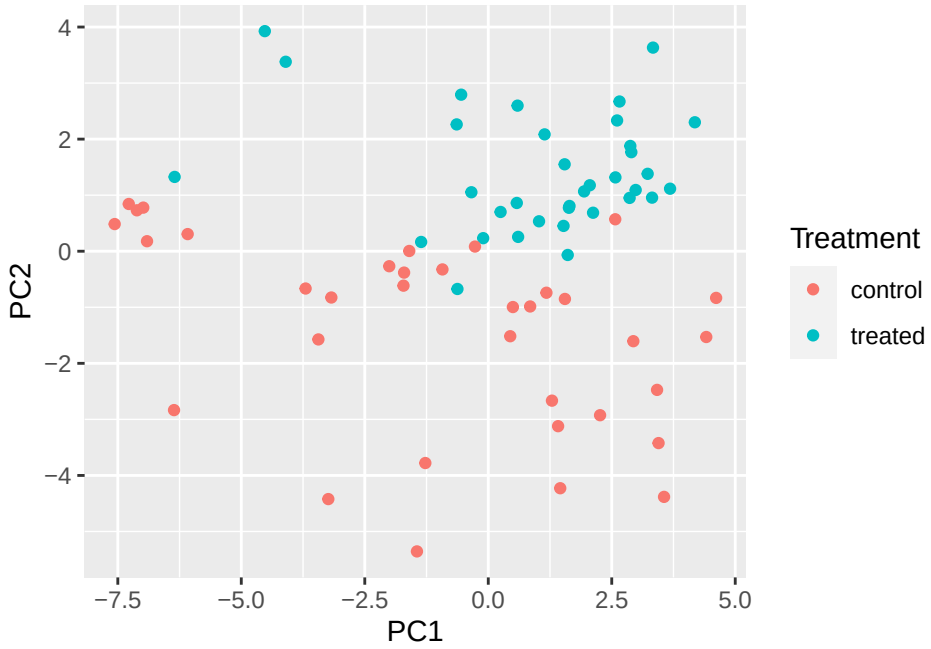
```
# simplify taxonomic names to just the phylum
rownames(data) <- gsub("^.*;p__","",rownames(data))
head(rownames(data))
```

```
## [1] "Caldiserica"                  "Candidatus Levybacteria"
## [3] "Candidatus Magasanikbacteria" "Thermotogae"
## [5] "Candidatus Azambacteria"      "Thermodesulfobacteria"
```

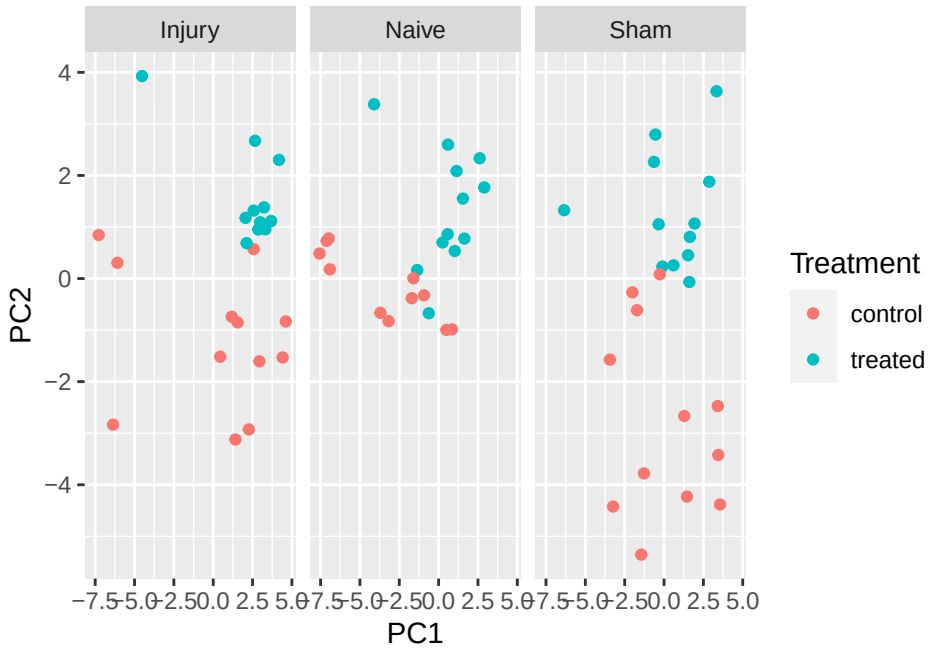
```
# also read in metadata
metadata <- read.table("https://uic-ric.github.io/workshop-data/edgeR/metagenomics_metadata.txt",
   header=T,row.names=1)
# match metadata and data, filter the taxa
data <- data[,rownames(metadata)]
data_subset <- data[rowSums(data)>20,]
# run just a few steps in edgeR to get normalized abundance
# you can run differential stats also, but you should be an expert on that by now
# so we omit those steps here
taxa <- DGEList(counts = data_subset)
taxa <- calcNormFactors(taxa)
taxa.norm <- cpm(taxa,log=T)
# run PCA analysis and make a data frame with metadata
pca <- prcomp(t(taxa.norm))
pca.data <- cbind(pca$x, metadata)
# colored by Injury and Treatment
ggplot(pca.data,aes(x=PC1,y=PC2,color=Injury)) + geom_point()
```



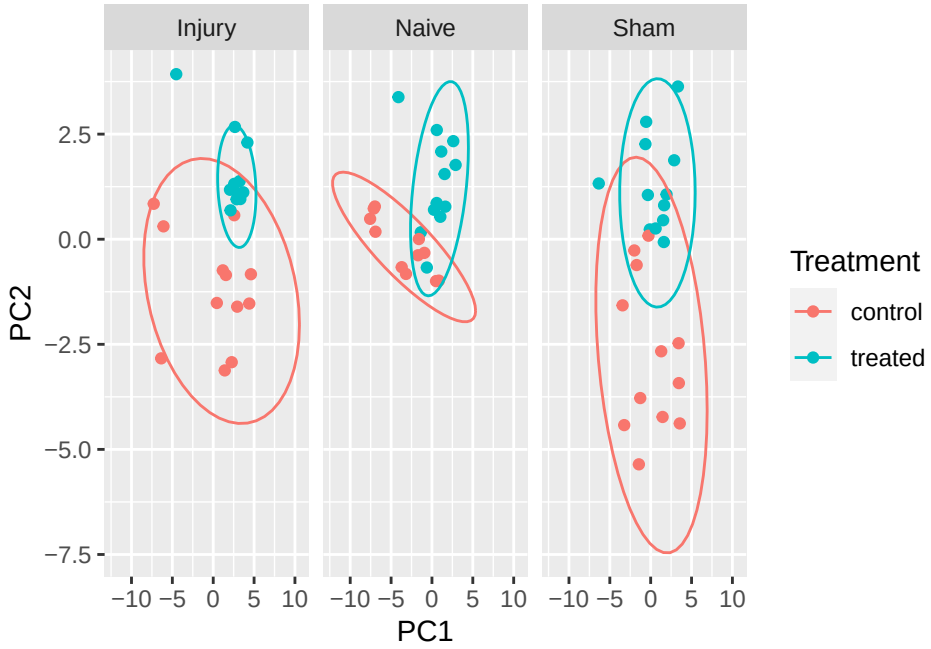

```
ggplot(pca.data,aes(x=PC1,y=PC2,color=Treatment)) + geom_point()
```




```
# facet by Treatment
ggplot(pca.data,aes(x=PC1,y=PC2,color=Treatment)) + geom_point() +
   facet_grid(~Injury)
```



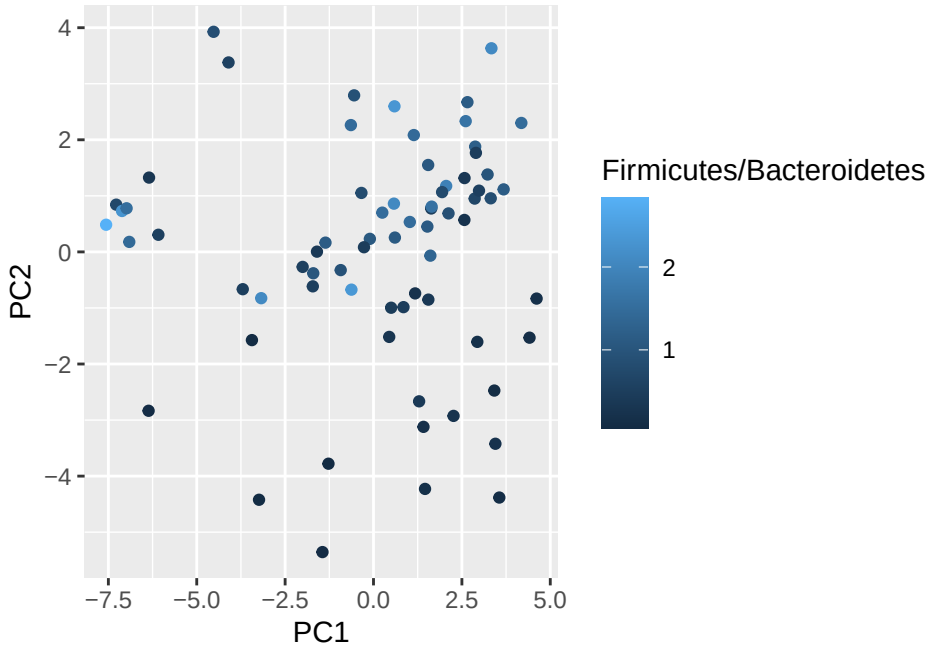

```
# plot each factor, add confidence ellipse
ggplot(pca.data,aes(x=PC1,y=PC2,color=Treatment)) + geom_point() +
  facet_grid(~Injury) + stat_ellipse()
```



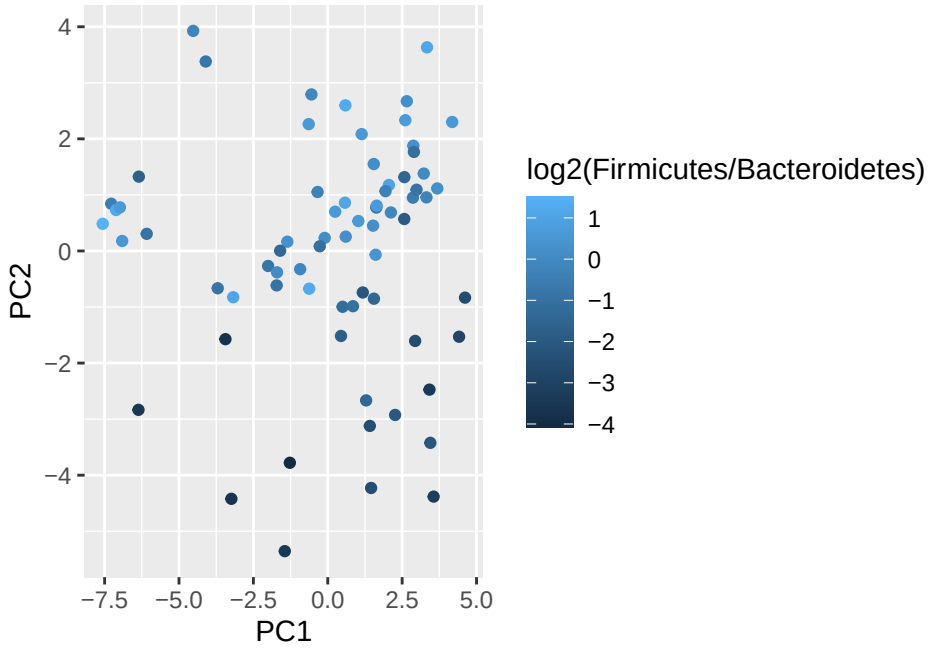
The Firmicutes/Bacteroidetes ratio is a common measure of the health of a gut microbiome. We can add that value to the PCA plot as the dot colors.

```
# add counts for Bacteroidetes and Firmicutes
pca.data.expanded <- cbind(pca.data,
   Bacteroidetes = t(data["Bacteroidetes",]),
   Firmicutes = t(data["Firmicutes",]))
# color by Firmicutes/Bacteriodetes ratio
ggplot(pca.data.expanded,aes(x=PC1,y=PC2,color=Firmicutes/Bacteroidetes)) +
```

```
  geom_point()
```

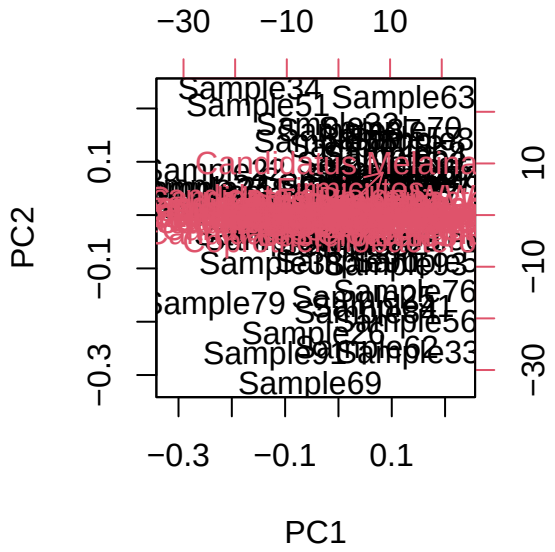



```
# color by the log2-ratio
ggplot(pca.data.expanded,aes(x=PC1,y=PC2,color=log2(Firmicutes/Bacteroidetes))) +
  geom_point()
```



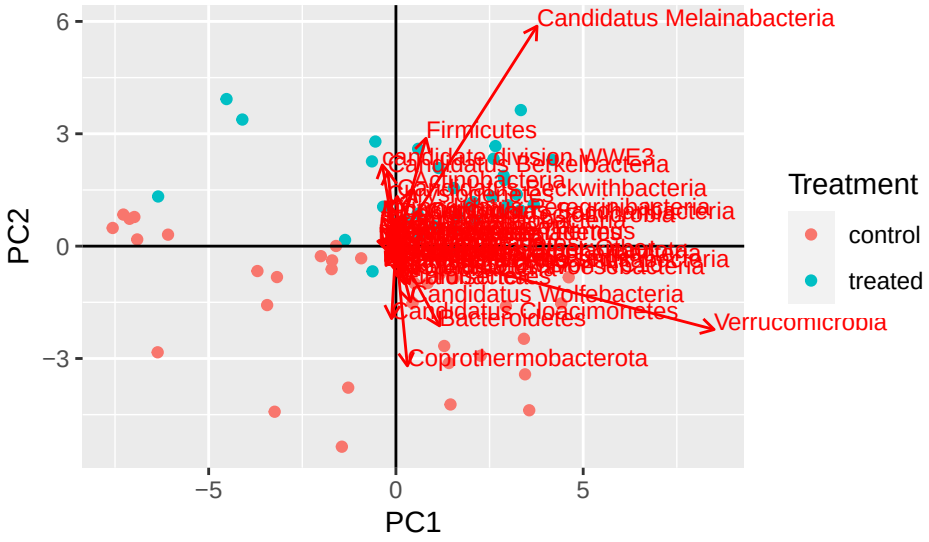
Make add a biplot, which allows us to visualize the contribution of each feature on the principle components. Note that you may want to be careful with this plot if you have a huge number of features, but with a relatively small number of taxa it will work.

```
# using the built-in biplot function
biplot(pca)
```




```
# the setup is more manual with ggplot2, but
# we have more control over the plotting aesthetics

# first, get the rotation matrix as a data frame
rotation <- data.frame(taxa=rownames(pca$rotation), pca$rotation)
# determine a scalar to multiply the vector lengths by
# so that the scales line up better
mult <- min(
  (max(pca.data[,"PC2"]) - min(pca.data[,"PC2"])/(max(rotation[,"PC2"])-min(rotation[,"PC2"]))),
  (max(pca.data[,"PC1"]) - min(pca.data[,"PC1"])/(max(rotation[,"PC1"])-min(rotation[,"PC1"])))
)
rotation$PC1 = mult * rotation$PC1
rotation$PC2 = mult * rotation$PC2
# start making the plot: first the main PCA scatterplot
plot <- ggplot(pca.data, aes(x=PC1, y=PC2,color=Treatment)) + geom_point()
# add vertical and horizontal lines for the middle of the biplot
plot <- plot + geom_hline(yintercept=0) + geom_vline(xintercept=0)
# add text labels for each taxa
# set clip="off" to let text labels go outside the plot area
plot <- plot + coord_equal(clip='off') +
   geom_text(data=rotation, aes(x=PC1, y=PC2, label=taxa),
       size=3, hjust=0, vjust=0, color="red")
# add arrows for the biplot: from the origin (0,0)
# to the PC coordinate in the rotation matrix
plot <- plot + geom_segment(data=rotation, aes(x=0, y=0, xend=PC1, yend=PC2),
   arrow=arrow(length=unit(0.2,"cm")), color="red")
plot
```


Candidatus Melainabacteria
Firmicutes
Candidatus Berkelbacteria
Verrucomicrobia
Coprothermobacterota
Bacteroidetes
Candidatus Wolfebacteria
Candidatus Cloacimonetes
Treatment
control
treated
PC1
PC2
6
3
0
−3
−5
0
5