Differential Expression Analysis with Limma in R
by John Blischak and DataCamp

Analyze data generated by functional genomic experiements
example
cells treated by two separate drugs
the drug treatment is the variable of interest
this is an example of a phenotype
we put the samples through a high-throughput assay that can measure thousands of genes
genes here are called ‘features’
in other experiments ‘features’ can be proteins or other molecular structures
each ‘feature’ that is produced by the assay is a value that is a proxy relative to the abundance of that feature
for genes this number represents the number of RNA transcripts expressed
this measurement can be referred to as ‘expression levels’
our statistical models will test for differences in these measurements between samples with different phenotypes
if a feature has a higher expression level for one group relative to the other this is called ‘upregulated’
if lower this is called ‘down-regulated’
the goal of DE analysis is to identify genes that are associated with a phenotype of interest
examples
-identifying all the genes associated with a response to a stimulus like a drug, a developmental process, or a genetic mutation

Why DE?
why test thousands of genes (called a genome-wide DE analysis)?
- may find addtional genes of interest (unexpecting genes play a role)
- interpreting the relevance of any one gene is easier when comparing it to the behavior of other genes
- gain a systems-level understanding of the process

Steps to an experiment
1. design study
2. perform experiment
3. collect data
4. pre-process data
5. explore data
6. test data

7. interpret results
8. share results

Always remember
*measurements are relative, not absolute
impossible to directly convert these measurements to the total number of molecules
*stats cannot save a poorly designed study

Dataset from Bioconductor package “breastCancerVDX”
from Wang and Minn
344 patients: 209 estrogen receptor + and 135 ER-
analyze the differences between patients

2nd dataset from Bioconductor “CLL” chronic lymphocytic leukemia
from Chiaretti and Ritz
22 patients: 8 stable disease, 14 progressive disease
again goal is testing for differences between two groups of samples

Breaking down these experiments
each has 3 main data sets
1. expression matrix (x) - contains the expression measurements
2. feature data (f) - describes each of the measured features (often genes but can be protein or other molecular data)
3. phenotype data (p) - describes each of the samples in the study (ex. drug or no drug)

Expression matrix dataset
class matrix
each row is a feature that was measured
each column is one of the samples
Breast CA dataset measures 22,283 genes for 344 samples

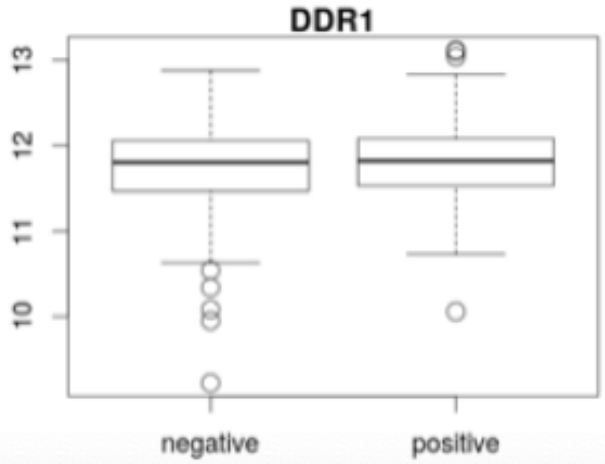
Feature dataset
a dataframe with one row per feature
each row is a gene
the columns describe the features
for the Breast CA dataset columns describe gene symbol, database identifier, and the chromosomal location in the genome

Phenotype dataset
a dataframe
each row is a sample

columns describe the samples
for this dataset - sample identifier, age of the subject, whether or not the tumor sample was positive or negative ER

Visualize gene expression with a boxplot
boxplot(<y-axis> ~ <x-axis>, main = "<title>")
this will create one boxplot for each value of the x-axis variable
example

```
boxplot(<gene expression> ~ <phenotype>, main = "<feature>")
boxplot(x[1, ] ~ p[, "er"], main = f[1, "symbol"])
```



gene expression identifies the single gene (expression matrix)
phenotype is ER neg vs ER pos (phenotype dataset)
y-axis vs x-axis respectively
we use the feature data to label the plot (feature dataset)
to plot the first gene in our breast CA dataset we subset the first row of the expression matrix
set 'er' which selects the 'er' column from the phenotype data frame
for the title, use the feature data column 'symbol'
remembering to subset to only include the first row, corresponding to the first gene
*what this output shows us
-this gene appears to be similar in both groups

Example
# Create a boxplot of the first gene in the expression matrix
# x for exp matrix, p for phenotype dataset, f for feature dataset
boxplot(x[1,] ~ p[, "Disease"], main = f[1, "symbol"])

Subsetting can get messy and prone to mistakes
Bioconductor helps with this problem
provides classes to store data for complex biological experiments
**this approach is known as object-oriented programming
a class defines a structure to hold complex data
a variable of a given class is referred to as an object of that class (also called an instance of a class)
every class has methods (or functions) that work in a special way for objects of that class
'getters' (also called accessors) - methods that retrieve data in an object
'setters' - modify the data
some methods can do both
core Bioconductor classes are in the package Biobase
install.packages("BiocManager")
BiocManager::install("Biobase")

Creating an object that contains all three datasets
called ExpressionSet object

```
# Create ExpressionSet object
eset <- ExpressionSet(assayData = x,
                      phenoData = AnnotatedDataFrame(p),
                      featureData = AnnotatedDataFrame(f))
# View the number of features (rows) and samples (columns)
dim(eset)
```

```
Features  Samples
   22283      344
```

expression matrix gets passed to assayData
*need to convert the phenotype and feature data frames into annotated data frames
this is a Bioconductor class that supports including descriptions of the columns of that data frame

Accessing data from an ExpressionSet object
using above example
x <- exprs(eset) #expression matrix
f <- fData(eset) #feature dataset

p <- pData(eset) #phenotype dataset

Why we do this?
when we subset without ExpressionSet object vs with:

- Subset with 3 separate objects:

```
x_sub <- x[1000, 1:10]
f_sub <- f[1000, ]
p_sub <- p[1:10, ]
```

- Subset with an ExpressionSet object:

```
eset_sub <- eset[1000, 1:10]
```

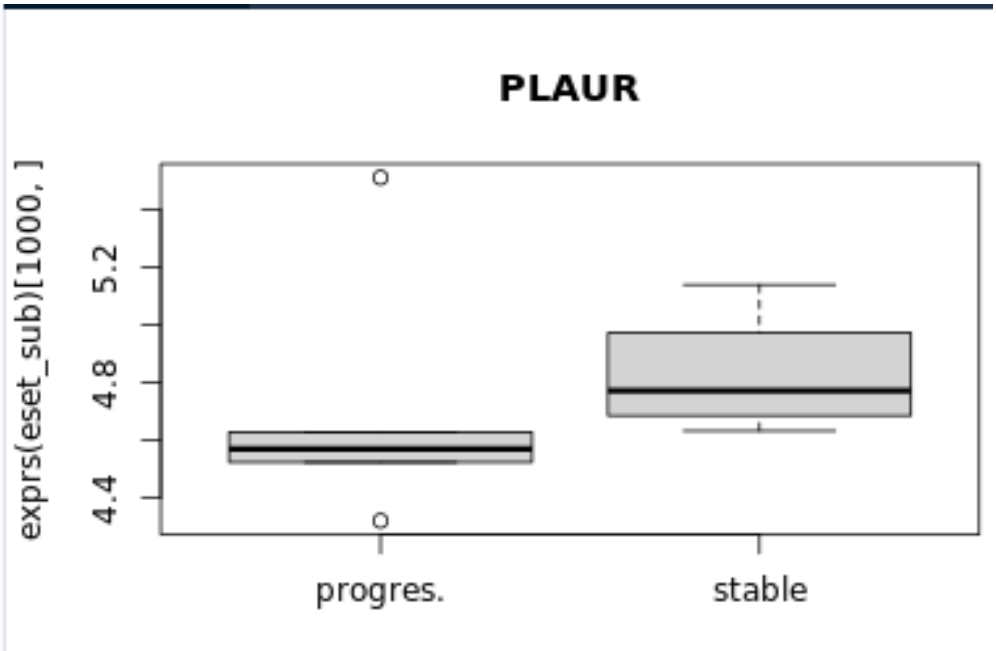
Visualizing with ExpressionSet object
boxplot(exprs(eset)[1,] ~ pData(eset)[, “er”], main = fData(eset)[1, “symbol”])

Example
# Subset to only include the first 10 samples (columns)
eset_sub <- eset[, 1:10]

# Check the dimensions of the subset
dim(eset_sub)

Create a boxplot of the 1000th gene in eset_sub
boxplot(exprs(eset_sub)[1000,] ~ pData(eset_sub)[, "Disease"],
main = fData(eset_sub)[1000, "symbol"])

output>



*shows higher expression of the gene PLAUR in stable leukemia compared to progressive leukemia

limma package
boiler plate code can get cumbersome
more importantly this type of code treats every gene as a completely independent analysis
the power of limma is it performs action on every gene in the data set
limma does this using a statistical technique known as empirical Bayes
which shares information across the genes
*this can be helpful for studies with smaller sample sizes

quick note on Bayes:
Bayes' theorem is a mathematical formula that calculates the probability of an event based on prior knowledge of conditions that might be related to the event. It's used to update probabilities based on new evidence or information. The formula is as follows:

$$ P(A | B) = \frac{P(B | A) \cdot P(A)}{P(B)} $$

Here's a breakdown:

- $P(A | B)$ is the posterior probability of event A given evidence B.
- $P(B | A)$ is the likelihood of evidence B given that event A has occurred.
- $P(A)$ is the prior probability of event A.
- $P(B)$ is the probability of evidence B.

In simpler terms, Bayes' theorem allows us to update our beliefs about the probability of an event (posterior probability) based on new evidence. It takes into account both our prior beliefs (prior probability) and the probability of observing the given evidence under different conditions.

So, you're correct in saying that it involves using past information to update probabilities rather than treating each event as completely independent.

limma by using Bayes improves inference by sharing information across genes
install:
biocManager::install("limma")

To identify differentially expressed genes between ER- and ER+
we need to fit to a linear model
Y = Bo + B1X1 + e
where
Y = expression level of gene
Bo = mean expression level of the gene in ER- tumors
B1 = mean difference in expression level of the gene in ER+ tumors compared to ER- tumors
X1 = ER status where 0 = neg and 1 = pos
e = epsilon and models the random noise

Putting it into R
model.matrix(~<explanatory>, data = <data frame>)
#only need to specify the explanatory variable
example with breast CA data

```
design <- model.matrix(~er, data = pData(eset))
```

```
head(design, 2)
```

```
      (Intercept) erpositive
VDX_3           1          0
VDX_5           1          1
```

er is the explanatory variable
not in quotes because I specify the source of the data is the phenotype data frame
the design matrix corresponds to a coefficient in the linear model

if the sample in a given row is modeled by this coefficient, then it has the value 1, 0 otherwise
above the Intercept represents an instance, here represent 1 for each sample, so Intercept is 1 for each data point
erpositive is 0 or 1 depending on if erpositive or ernegative
we can sanity check this and make sure it correlates with what we know
colSums(design)
output>

```
(Intercept)  erpositive
        344         209
```

Standard limma pipeline
library(limma)
#fit the coeffs of the model by passing it the ExpressionSet object and design matrix
fit <- lmFit(eset, design)
#calculate the t-statistics
fit <- eBayes(fit)
#summarize results > count the number of genes with higher or lower expression in ER+ tumors compared to ER-
results <- decideTests(fit[, “er”])
summary(results)
output>

```
   erpositive
-1       6276
0       11003
1        5004
```

Example
# Create design matrix for leukemia study
design <- model.matrix(~Disease, data = pData(eset))

# Count the number of samples modeled by each coefficient
colSums(design)

```
# Load package
library(limma)

# Fit the model
fit <- lmFit(eset, design)

# Calculate the t-statistics
fit <- eBayes(fit)

# Summarize results
results <- decideTests(fit[, "Diseasestable"])
summary(results)
```

Above works well for two groups but what about three groups or more
what if we had an additional group B2 (mean difference in group 3)
can use above to compare group 1 to group 2 and group 1 to group 3 but what about group 2 to group 3
this is problem is called treatment-contrasts parametrization
each coefficient represents a difference from a base condition

Dealing with this > use group-means parametrization
each coefficient models the mean expression level in a given group
*this model there is no intercept coefficient
the coefficients are no longer differences, you have to construct contrasts to test for differential expression
in the breast CA with two groups you test if the difference in the two coeffs is equal to 0
in 3 groups, test all pairwise comparisons by constructing 3 contrasts

$$Y = \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \epsilon$$

- β_1 - Mean in group 1
- β_2 - Mean in group 2
- β_3 - Mean in group 3
- Tests:
  - $\beta_2 - \beta_1 = 0$
  - $\beta_3 - \beta_1 = 0$
  - $\beta_3 - \beta_2 = 0$

*to do this we need to create a model without an intercept term
to do this we include a 0 in the formula argument
example with breast CA dataset
*we can do this with two groups as well
instead of having Intercept that represents an instance for each sample
we get two columns, one for erneg and one for erpos
colSums still equals out to the known 344 samples

```
design <- model.matrix(~0 + er, data = pData(eset))
head(design)
```

```
      ernegative erpositive
VDX_3          1          0
VDX_5          0          1
VDX_6          1          0
VDX_7          1          0
VDX_8          1          0
VDX_9          0          1
```

```
colSums(design)
```

```
ernegative erpositive
       135        209
```

Contrasts matrix
create contrasts to test specific hypotheses
*can make any number of contrasts by referring specifically to the column names of the design matrix
pass the design matrix object to the ‘levels’ argument
example

```
library(limma)
cm <- makeContrasts(status = erpositive - ernegative,
                    levels = design)
```

```
cm
```

```
            Contrasts
Levels       status
  ernegative     -1
  erpositive      1
```

here we have created a contrast called status
tests for a difference in the means of the erpos and erneg samples
the coeffs are the rows
the contrasts are the columns
*equation erpositive minus erneg is saved as multiplying the erpos coeff by 1 and erneg coeff by -1

Group-means parameterization requires an extra fit step
fit <- lmFit(eset, design)
#need to also fit the specific contrasts of interest
fit2 <- contrasts.fit(fit, contrasts = cm)

Results
#calculate the t-statistics
fit2 <- eBayes(fit2)
#count the number of DE genes
results <- decideTests(fit2)
summary(results)
**here the results are the same as our previous model (this simple model can use either parametrization)

Example
# Create design matrix with no intercept
design <- model.matrix(~0 + Disease, data = pData(eset))

```
# Count the number of samples modeled by each coefficient
colSums(design)

# Load package
library(limma)

# Create a contrasts matrix
cm <- makeContrasts(status = Diseaseprogres. - Diseasestable,
                    levels = design)

# View the contrasts matrix
cm

# Load package
library(limma)

# Fit the model
fit <- lmFit(eset, design)

# Fit the contrasts
fit2 <- contrasts.fit(fit, contrasts = cm)

# Calculate the t-statistics for the contrasts
fit2 <- eBayes(fit2)

# Summarize results
results <- decideTests(fit2)
summary(results)
```

Now doing true group parameterization with 3 groups
dataset = leukemiasEset
3 different types of leukemias > ALL, AML, CML
by Kohlmann and Haferlach
data
dim(eset)
20172 genes
36 samples (12 of each type)

Here is the model that we want to build

$$Y = \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \epsilon$$

- β_1 - Mean expression level in group ALL
- β_2 - Mean expression level in group AML
- β_3 - Mean expression level in group CML
- Tests:
  - AML v. ALL: $\beta_2 - \beta_1 = 0$
  - CML v. ALL: $\beta_3 - \beta_1 = 0$
  - CML v. AML: $\beta_3 - \beta_2 = 0$

to find the differentially expressed genes, test the above pairwise contrasts

process is similar to two groups
design <- model.matrix(~0 + type, data = pData(eset))
removes the intercept and gives us three coeffs
'type' is the column that represents the sample labels
next make our contrasts>

```
cm <- makeContrasts(AMLvALL = typeAML - typeALL,
                    CMLvALL = typeCML - typeALL,
                    CMLvAML = typeCML - typeAML,
                    levels = design)
```

run the rest of the limma pipeline

```
library(limma)
# Fit coefficients
fit <- lmFit(eset, design)

# Fit contrasts
fit2 <- contrasts.fit(fit, contrasts = cm)

# Calculate t-statistics
fit2 <- eBayes(fit2)

# Summarize results
results <- decideTests(fit2)
summary(results)
```

output>

```
   AMLvALL CMLvALL CMLvAML
-1     898    3401    1890
0    18323   13194   16408
1      951    3577    1874
```

what this tells us
show the number of upreguated and downregulated genes
CML vs ALL appear to have the biggest difference in cellular function

Practice dataset - stemHypoxia
3 different levels of oxygen 1%, 5%, 21%
by Prado-Lopez 2010
gene expression measurements of stem cells grown for 24 hours in 3 different levels of oxygen
the goal is to identify the genes that are affected by lower oxygen levels
15,325 genes
6 samples
2 replicates for each level of oxygen

```
# Create design matrix with no intercept
design <- model.matrix(~0 + oxygen, data = pData(eset))

# Count the number of samples modeled by each coefficient
colSums(design)
```

output>

```
oxygenox01 oxygenox05 oxygenox21
         2          2          2
```

```
# Load package
library(limma)

# Create a contrasts matrix
cm <- makeContrasts(ox05vox01 = oxygenox05 - oxygenox01,
                    ox21vox01 = oxygenox21 - oxygenox01,
                    ox21vox05 = oxygenox21 - oxygenox05,
                    levels = design)

# View the contrasts matrix
cm

# Load package
library(limma)

# Fit the model
fit <- lmFit(eset, design)

# Fit the contrasts
fit2 <- contrasts.fit(fit, contrasts = cm)

# Calculate the t-statistics for the contrasts
fit2 <- eBayes(fit2)

# Summarize results
results <- decideTests(fit2)
summary(results)
```

output>

	ox05vox01	ox21vox01	ox21vox05
Down	888	1854	214
NotSig	12671	10843	14972
Up	1766	2628	139

tells us as expected the largest difference is between ox21 vs ox01

Factorial experimental design
includes samples that experience each combination of experimental variables
example dataset > 2x2 design to study effect of low temperature in plants
2 types of Arabidopsis thaliana (col, vte2)
2 temps: normal, low
by Maeda
3 replicates for each combination of the two factors

Our design

$$Y = \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \epsilon$$

- β_1 - Mean expression level in `col` plants at `low` temperature
- β_2 - Mean expression level in `col` plants at `normal` temperature
- β_3 - Mean expression level in `vte2` plants at `low` temperature
- β_4 - Mean expression level in `vte2` plants at `normal` temperature

Building the group-means design matrix for 2x2 factorial
for this dataset we need to create a single variable that describes the 4 groups of samples
do this using the phenotype dataframe
use paste() to combine the variables
here the variables we are combining are type and temp into a single variable
sep = “.” creates a new variable name that combines both names separated by a .
also need to convert this character vector to a factor
a factor records the unique levels of a variable
replace the column names with the factor levels using the function levels()

```
group <- with(pData(eset),
              paste(type, temp, sep = "."))
group <- factor(group)
```

```
design <- model.matrix(~0 + group)
colnames(design) <- levels(group)
```

```
head(design, 3)
```

```
  col.low col.normal vte2.low vte2.normal
1       0          1        0           0
2       0          1        0           0
3       0          1        0           0
```

```
colSums(design)
```

```
    col.low  col.normal    vte2.low vte2.normal
          3           3           3           3
```

Our contrasts for a 2x2 factorial

	β_1	β_2	β_3	β_4
`type`	col	col	vte2	vte2
`temp`	low	normal	low	normal

- Differences of `type` in normal `temp` : $\beta_4 - \beta_2 = 0$
- Differences of `type` in low `temp` : $\beta_3 - \beta_1 = 0$
- Differences of `temp` in vte2 `type` : $\beta_3 - \beta_4 = 0$
- Effect of `temp` in col `type` : $\beta_1 - \beta_2 = 0$
- Differences of `temp` between col and vte2 `type` : $(\beta_3 - \beta_4) - (\beta_1 - \beta_2) = 0$

the last one is our interaction effect
we're asking how does the response to temperature differ between the 2 types of plants

Translate these 5 contrasts

```
library(limma)
cm <- makeContrasts(type_normal = vte2.normal - col.norma
                    type_low = vte2.low - col.low,
                    temp_vte2 = vte2.low - vte2.normal,
                    temp_col = col.low - col.normal,
                    interaction = (vte2.low - vte2.normal
                    levels = design)
```

```
cm
```

Levels	Contrasts				
	type_normal	type_low	temp_vte2	temp_col	interaction
col.low	0	-1	0	1	-1
col.normal	-1	0	0	-1	1
vte2.low	0	1	1	0	1
vte2.normal	1	0	-1	0	-1

we do this by referring to the column names of the design matrix
above interaction contrast is cut off > involves all four

Cont with expected pipeline

```
library(limma)

# Fit coefficients
fit <- lmFit(eset, design)

# Fit contrasts
fit2 <- contrasts.fit(fit, contrasts = cm)

# Calculate t-statistics
fit2 <- eBayes(fit2)

# Summarize results
results <- decideTests(fit2)
summary(results)
```

	type_normal	type_low	temp_vte2	temp_col	interaction
-1	0	466	1635	1885	128
0	11871	10915	7635	6989	11640
1	0	490	2601	2997	103

tells us the plants have similar function at normal temp but at low temp they start to diverge

Practice dataset - effect of drought on Populus trees
2x2 design
2 types of Populus: DN34, NM6
2 conditions: normal water, drought
by Wilkins in 2009
16,172 genes
12 samples
3 replicates

Example
# Create single variable
group <- with(pData(eset), paste(type, water, sep = "."))

```
group <- factor(group)

# Create design matrix with no intercept
design <- model.matrix(~0 + group)
colnames(design) <- levels(group)

# Count the number of samples modeled by each coefficient
colSums(design)

# Load package
library(limma)

# Create a contrasts matrix
cm <- makeContrasts(type_normal = nm6.normal - dn34.normal,
                    type_drought = nm6.drought - dn34.drought,
                    water_nm6 = nm6.drought - nm6.normal,
                    water_dn34 = dn34.drought - dn34.normal,
                    interaction = (nm6.drought - nm6.normal) - (dn34.drought -
dn34.normal),
                    levels = design)

# View the contrasts matrix
cm
```

output>

```
             Contrasts
Levels        type_normal type_drought water_nm6 water_dn34 interaction
  dn34.drought           0           -1         0          1          -1
  dn34.normal           -1            0         0         -1           1
  nm6.drought            0            1         1          0           1
  nm6.normal             1            0        -1          0          -1
```

```
# Load package
library(limma)

# Fit the model
fit <- lmFit(eset, design)

# Fit the contrasts
fit2 <- contrasts.fit(fit, contrasts = cm)
```

```
# Calculate the t-statistics for the contrasts
fit2 <- eBayes(fit2)

# Summarize results
results <- decideTests(fit2)
summary(results)
```

output>

	type_normal	type_drought	water_nm6	water_dn34	interaction
Down	1829	3499	0	95	78
NotSig	12732	10151	16172	15846	16079
Up	1611	2522	0	231	15

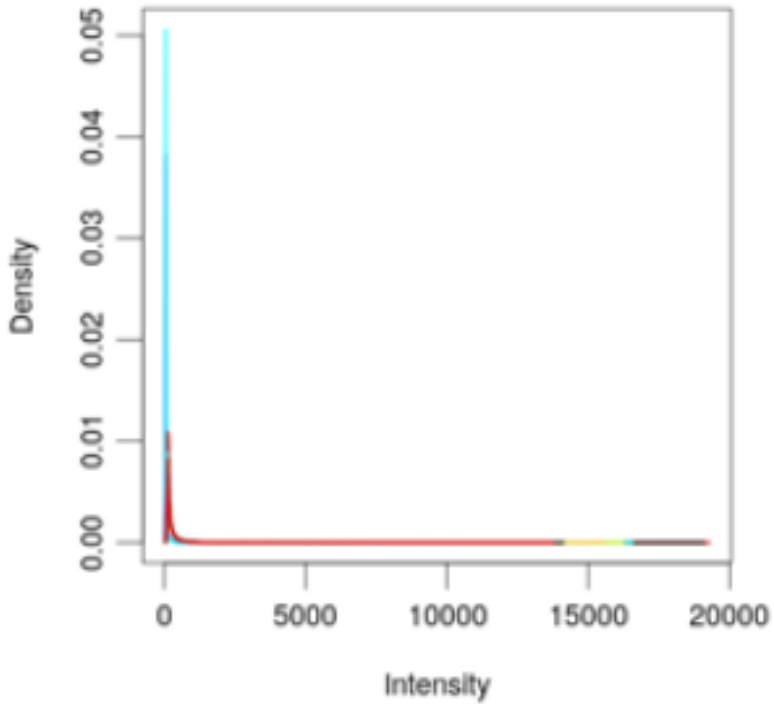
Normalizing and filtering
three steps

1. log transfrom
2. quantile normalize
3. filter

Before starting these three steps, we need to pre-process
to do this we need to visualize
we use a density plot
very similar to a histogram but the data is smoothed out to be continuous
histograms can be limiting with such large samples

```
library(limma)
plotDensities(eset, legend = FALSE)
```

the first argument is the ExpressionSet object
we remove the legend to ensure it doesn't block our viewing
example - above code using Arabidopsis dataset
output>



its smushed because all the densities lie near zero
what this means
here we have measured most of the genes in the genome
however only a subset of these genes are relevant to the system being studied
here we have most genes measured as zero and a subset with very high levels
*this gives us a very right-skewed distribution

We use log transform to unsmush
what log transform does?
increases the distance between the small measurement and decreases the distance between the large measurements
here is what log transforming does:

```
100 - 1
.1 - .001
```

```
99
0.099
```

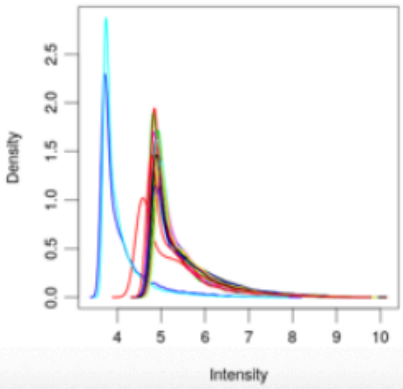
```
log(100) - log(1)
```

```
4.60517
```

```
log(.1) - log(.001)
```

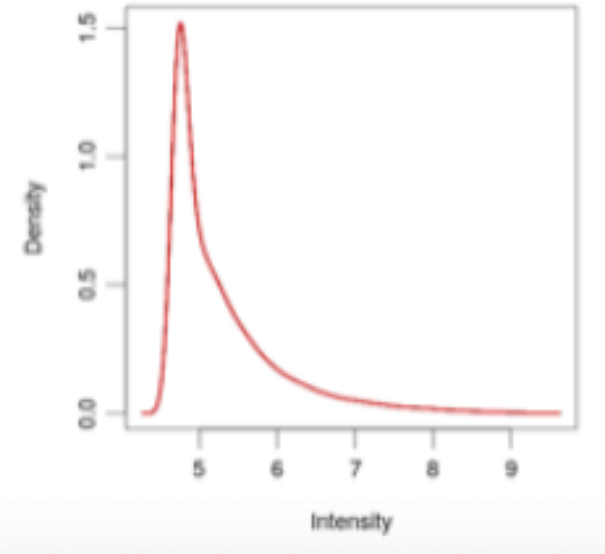
```
4.60517
```

```
# Log tranform
exprs(eset) <- log(exprs(eset))
plotDensities(eset, legend = FALSE)
```



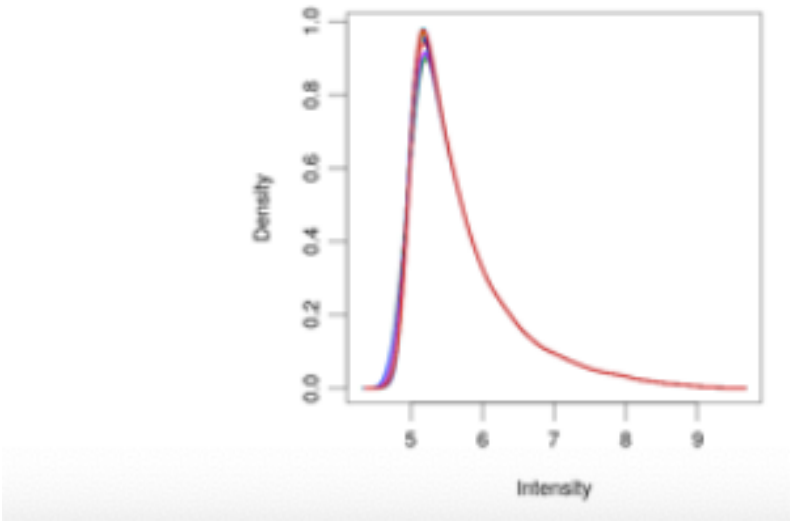
on the log scale the difference between 100 and 1 is identical to the difference between .1 and .001
R function log() by default computes the natural logarithm (base 10)
to use:
you pass it the expression matrix using exprs() and updat the ExpressionSet object by re-assigning the result to the expression matrix

Quantile normalize
distributions are not the same across the samples
*genomic techniques measure relative abundance
so these large scale differences are not meaningful, but arise from technical artifacts inherent to all genomic techniques
we can remedy this with quantile normalization
converts each sample to have the same distribution based on quantiles that are empirically computed as the average quantiles across all the samples
example:
exprs(eset) <- normalizeBetweenArrays(exprs(eset))
plotDensities(eset, legend = FALSE)



Filter genes
above lowly expressed genes are still creating a right-skew in the data
remove them by choosing a cutoff that excludes the peak of genes with low expression levels
figuring out that cutoff
calculate the mean of each row of the expression matrix
our example we estimate the line cutoff to be around 5

```
# Create logical vector
keep <- rowMeans(exprs(eset)) > 5
# Filter the genes
eset <- eset[keep, ]
plotDensities(eset, legend = FALSE)
```



we save as a logical vector that we have named ‘keep’
data is now ready for analysis

Example
library(limma)

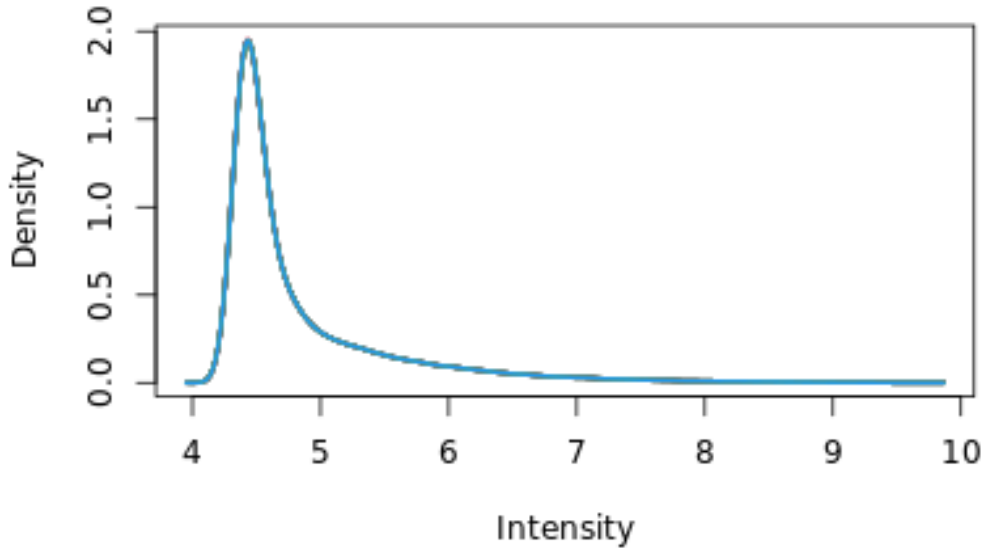
# Create new ExpressionSet to store normalized data
eset_norm <- eset_raw

# View the distribution of the raw data
plotDensities(eset_norm, legend = FALSE)

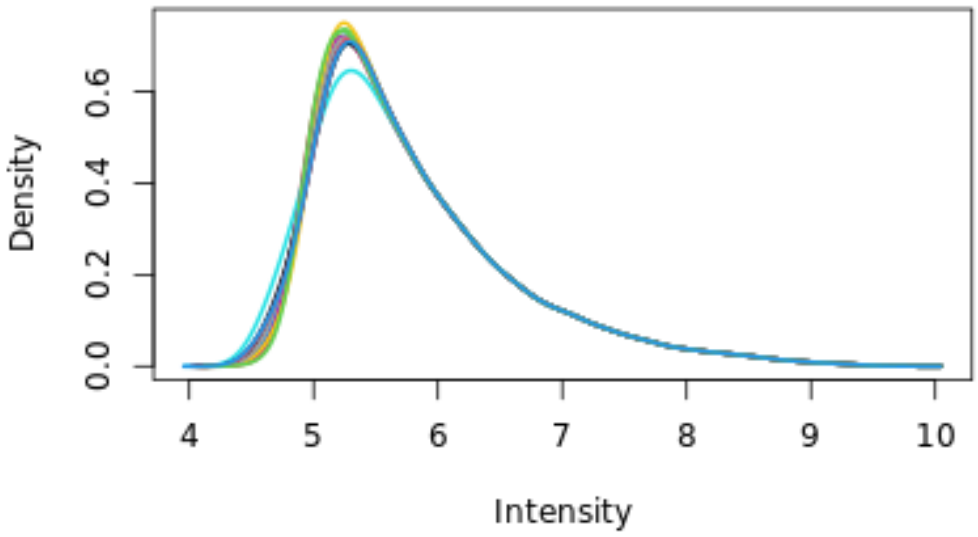
# Log tranform
exprs(eset_norm) <- log(exprs(eset_norm))
plotDensities(eset_norm, legend = FALSE)

# Quantile normalize
exprs(eset_norm) <- normalizeBetweenArrays(exprs(eset_norm))
plotDensities(eset_norm, legend = FALSE)

output>

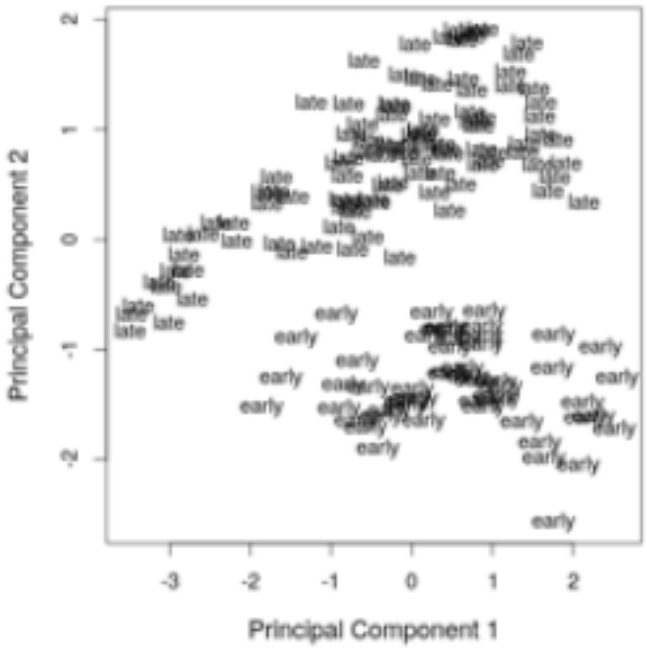



```
library(limma)

# Create new ExpressionSet to store filtered data
eset <- eset_norm

# View the normalized gene expression levels
plotDensities(eset, legend = FALSE); abline(v = 5)

# Determine the genes with mean expression level greater than 5
keep <- rowMeans(exprs(eset)) > 5
sum(keep)

# Filter the genes
eset <- eset[keep, ]
plotDensities(eset, legend = FALSE)

output>
```



Accounting for technical batch effects
every batch of an experiment is slightly different
need to balance variable of interest across batches
*if properly balanced, batch effects can be removed
to investigate batch effects you can use dimension techniques like PCA or multidimensional scaling (MDS)
these techniques reduce the representation of each sample from a vector of thousands of measurements to a vector the length of the number of samples
this reduced vector captures the largest sources of variation in the data
starting from the largest source
each one is orthogonal (independent) to the next
usually we focus on the first two dimensions becuase these are the largest sources of variation
this makes for easier visualization
want to determine if these sources of variation are correlated with the variables of interest or batch effects

example
plotMDS(eset, labels = pData(eset)[, “time”], gene.selection = “common”)
default subset to only include the 500 most variable genes in the data
gene.selection argument to ‘common’ performs PCA
output>

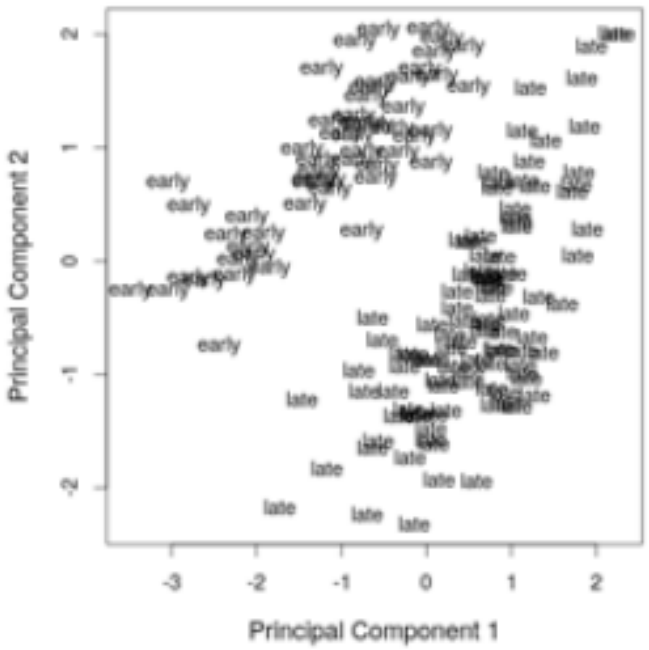


samples separated along the second PC on the y-axis
this suggested that the largest source of variation was due to a technical batch effect
if the samples were balanced across the experimental batches, we can remove the unwanted variation with removeBatchEffect()
this fits a linear model
and returns the residuals
our above example cont'd:

```
exprs(eset) <- removeBatchEffect(eset,
                  batch = pData(eset)[,"batch"],
                  covariates = pData(eset)[,"rin"])
```

```
plotMDS(eset, labels = pData(eset)[, "time"],
        gene.selection = "common")
```

pass the discrete variable batch to the argument 'batch'
continuous variable to argument 'covariates' #here 'rin' represents a measure of RNA quality
output>

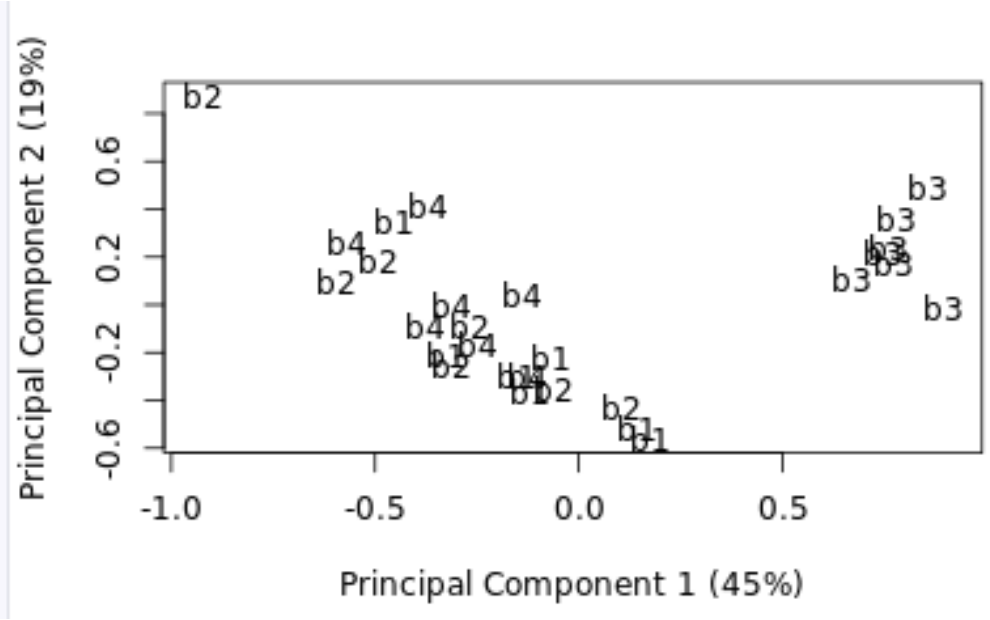


now the early and late samples are separated by PC1 on the x-axis
*this technique is good for visualization but should not be used in actual statistical analysis
include the batch variable as a coeff when constructing your design matrix

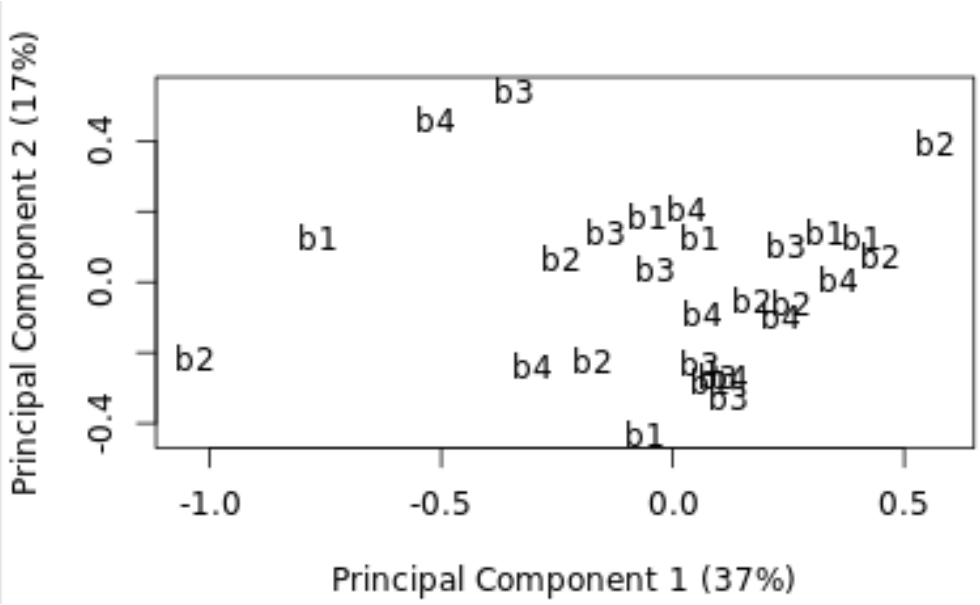
Olfactory stem cells dataset
7 treatments, 4 batches
via Bioconductor > HarmanData, Harman
by Osmond-McLeod and Oytam

Example
```
# Load package
library(limma)

# Plot principal components labeled by treatment
plotMDS(eset, labels = pData(eset)[, "treatment"], gene.selection = "common")

# Plot principal components labeled by batch
plotMDS(eset, labels = pData(eset)[, "batch"], gene.selection = "common")
```

output>




```
# Load package
library(limma)

# Remove the batch effect
exprs(eset) <- removeBatchEffect(eset, batch = pData(eset)[,"batch"])

# Plot principal components labeled by treatment
plotMDS(eset, labels = pData(eset)[,"treatment"], gene.selection = "common")

# Plot principal components labeled by batch
plotMDS(eset, labels = pData(eset)[,"batch"], gene.selection = "common")

output>
```



Inspecting the results
results <- decideTests(fit2)
summary(results)

```
   status
-1   6276
0   11003
1    5004
```

topTable(fit2, number = 3)

```
           symbol entrez  chrom    logFC  AveExpr        t
205225_at     ESR1   2099 6q25.1 3.762901 11.37774 22.68392
209603_at    GATA3   2625  10p15 3.052348  9.94199 18.98154
209604_s_at  GATA3   2625  10p15 2.431309 13.18533 17.59968
                 P.Value    adj.P.Val        B
205225_at   2.001001e-70 4.458832e-66 149.1987
209603_at   1.486522e-55 1.656209e-51 115.4641
209604_s_at 5.839050e-50 4.337052e-46 102.7571
```

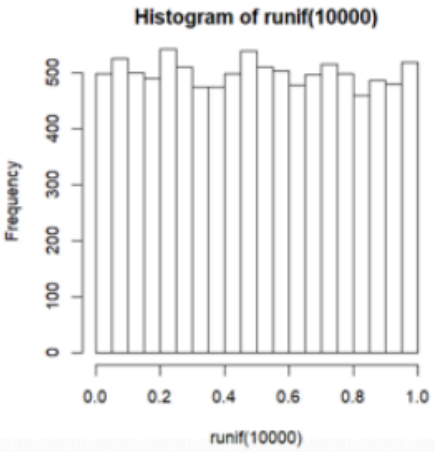
topTable gives the top differentially expressed genes
logFC > log-fold change in expression between the two groups
adjpval > uses the Benjamini-Hochberg false discovery rate (FDR)

B > is the log-odds, an alternative to the p-value for assessing if a gene is differentially expressed

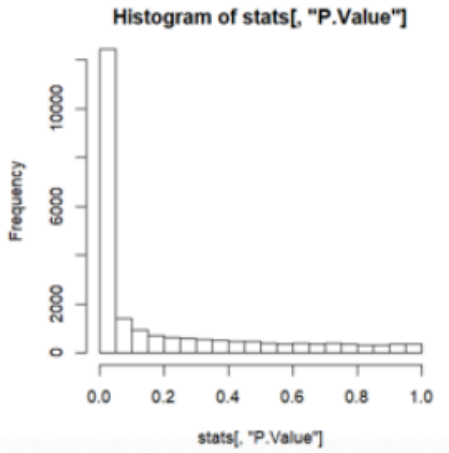
to get summary stats on all genes
stats <- toTable(fit2, number = nrow(fit2), sort.by = “none”)
pass the number of rows in the fit2 object
sort.by to none disables sorting by statistical significance
this can make it easier to compare and combine results

Histogram of p-values

```
hist(runif(10000))
```



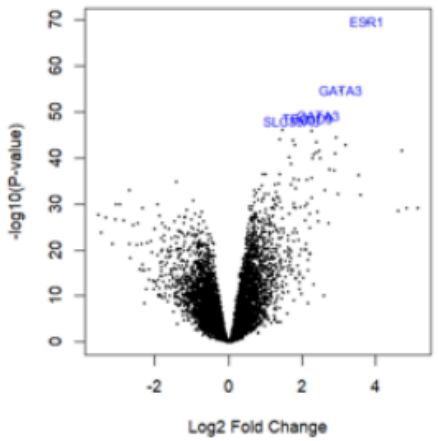

```
hist(stats[, "P.Value"])
```



expect left histogram for our null hypothesis of no differential expression
on the right for lots of differentially expressed genes you expect to see a right-skewed histogram due to many statistically significant p-values
deviations from these patterns may mean something is wrong with your code

Volcano plot

```
volcanoplot(fit2,
            highlight = 5,
            names = fit2$genes[, "symbol"])
```

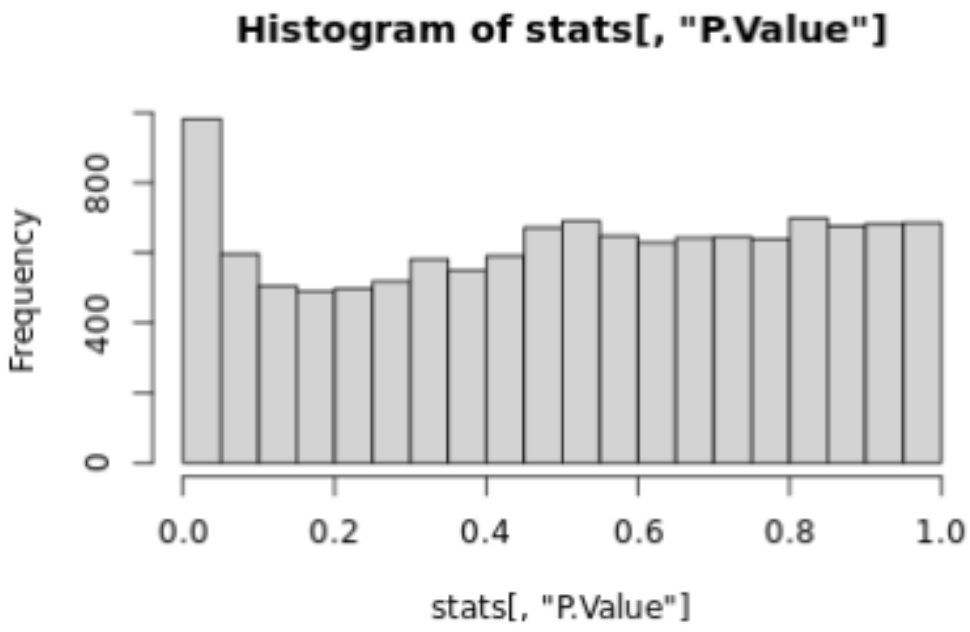


still using the breast CA dataset

first arg is the fitted model object
‘highlight’ argument allows us to highlight most significant genes
pass a vector of the labels to use for these genes
fit2 has a data framed ‘genes’ with the feature data
we use column ‘symbol’ to get the gene symbols
x-axis represents log-fold change between ER neg and ER pos
y-axis log-odds of differential expression
the higher the log-odds the more likely the gene is differentially expressed
this plot shape occurs because genes that have a larger log-fold change are more likely to be differentially expressed
*key note > this shape does not guarantee genes are differentially expressed

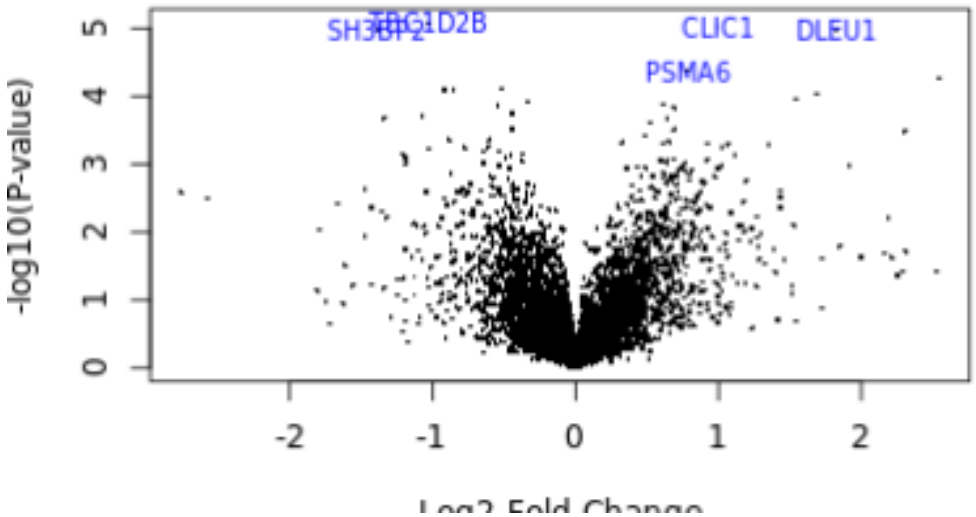
Example
```
# Obtain the summary statistics for every gene
stats <- topTable(fit2, number = nrow(fit2), sort.by = "none")

# Plot a histogram of the p-values
hist(stats[, "P.Value"])
```

output>




```
# Create a volcano plot. Highlight the top 5 genes
volcanoplot(fit2, highlight = 5, names = fit2$genes[, "symbol"])
```

output>



Enrichment testing
making sense of the results
biological databases out there that curate sets of related genes
examples
KEGG > photosynthesis, protein support
Gene Ontology Consortium > response to stress, developmental process
*enrichment is knowing if the differentially expressed genes in your experiment are overrepresented more than expected by chance in any known sets of genes
one way is via Fisher's exact test
tests for imbalances in a contingency table
example - 1000 genes with 100 DE genes

	In gene set	Not in gene set
DE	10	90
all	100	900

fisher.test(matrix(c(10,100, 90, 900), nrow = 2))

```
Fisher's Exact Test for Count Data

data:  matrix(c(10, 100, 90, 900), nrow = 2)
p-value = 1
alternative hypothesis: true odds ratio is
                        not equal to 1
95 percent confidence interval:
 0.4490765 2.0076377
sample estimates:
odds ratio
         1
```

this example shows no enrichment
because of the ratio
we have 10 out of the 100 DE genes
but also 100 out of the 900 all (non-DE) gene set

now this example

	In gene set	Not in gene set
DE	30	70
all	100	900

here we have enrichment
DE is 30%
which is higher than the 10% background rate
here the odds ratio is 3.85
and the p-value is very low 1.88e-07

Enrichment testing with limma
need to use a common id
in our example from the entrez database we use the gene IDs (column label 'entrez')
need the feature data from our prior ExpressionSet object
this is stored in the fitted mode object as the data frame 'genes'
can access like a list using the dollar sign notation

entrez <- fit2$genes[, “entrez”]
KEGG enrichment is performed with ‘kegga’
pass the fitted model object, the vector of gene IDs, and the species abbreviation
enrich_kegg <- kegga(fit2, geneid = entrez, species = “Hs”)
view the top enriched pathways with ‘togKEGG’
topKEGG(enrich_kegg, number = 3)
output>

	Pathway	N	Up	Down	P.Up	P.Down
path:hsa04110	Cell cycle	115	30	82	6.192773e-01	5.081518e-12
path:hsa05166	HTLV-I infection	233	55	135	8.959082e-01	9.285167e-09
path:hsa01100	Metabolic pathways	1033	350	373	3.175782e-08	9.969693e-01

‘N’ represents the number of genes that were in the set
‘Up’ number up regulated
‘Down’ number down regulated
then p values for enrichment of up and down regulated genes

Gene Ontology (GO) enrichment testing is similar as above
enrich_go <- goana(fit2, geneid = entrez, species = “Hs”)
topGO(enrich_go, otology = “BP”, number = 3)
“BP” is biological processes, there are many types of ontologies
can find them on the GO consortium website
‘number’ states how many we want in our output
*a note on GO > the same set of genes can be the underlying signal for many similar categories

Tips
don’t overinterpret
view as a tool to further investigate genes of interest
be skeptical of up vs down-regulated
only include the genes that were tested > if the background is all gene in the genome, this will bias the results

Example

```
# Extract the entrez gene IDs
entrez <- fit2$genes[, "entrez"]

# Test for enriched KEGG Pathways
enrich_kegg <- kegga(fit2, geneid = entrez, species = "Hs")

# View the top 20 enriched KEGG pathways
topKEGG(enrich_kegg, number = 20)
```

ouput>

	Pathway	N	Up	Down	P.Up	P.Down
path:hsa04650	<NA>	100	0	1	1	0.02204639
path:hsa00750	<NA>	2	0	0	1	1.00000000
path:hsa00780	<NA>	2	0	0	1	1.00000000
path:hsa00290	<NA>	3	0	0	1	1.00000000

+16 others

```
# Extract the entrez gene IDs
entrez <- fit2$genes[, "entrez"]

# Test for enriched GO categories
enrich_go <- goana(fit2[1:500, ], geneid = "entrez", species = "Hs")

# View the top 20 enriched GO Biological Processes
topGO(enrich_go, ontology = "BP", number = 20)
```

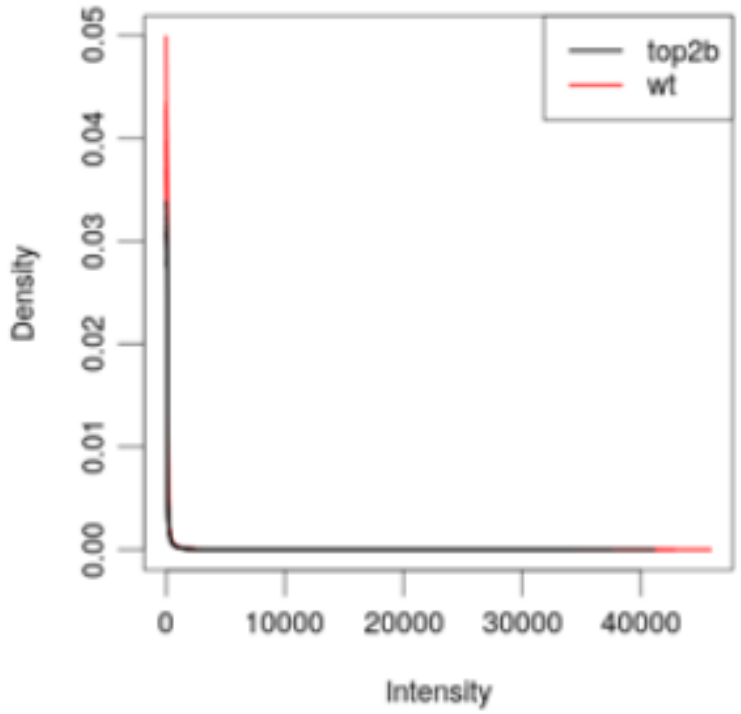
output>

	Term	Ont	N	Up	Down	P.Up
GO:0007165	signal transduction	BP	251	0	1	1
GO:0023052	signaling	BP	260	0	1	1
GO:0007154	cell communication	BP	264	0	1	1

Putting it all together
dataset studying the side effect of cardiotoxicity of cancer treatment drug doxorubicin
hypothesis doxo need top2b in order to cause cardiotoxicity
Does tox comes from doxo binding to protein topoisomerase-II beta (also called top2b)
2x2 study design
wt mice and top2b null mice
treated with either doxo or control solution (PBS)
29,532 genes, 12 mice, 3 replicates
Zhang 2012

First step - pre-process
to do this we visualize, go to is plotDensities
plotDensities(eset,

```
        group = pData(eset)[, “genotype”],
        legend = “topright”)
ouput>
```



most data lies near 0, this gives a long right tail
signifying many genes are not relevant in the mouse heart
we need to remove these

The three steps for pre-processing

1. log transformation to view the entire distribution
2. quantile normalization to transform each sample to the same empirical distribution
3. filtering to remove irrelevant genes

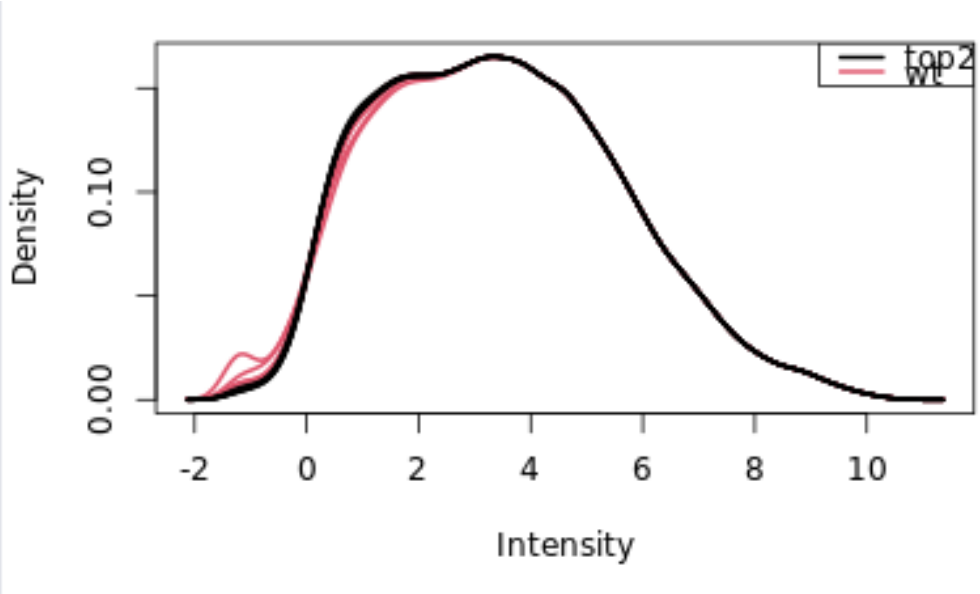
After pre-processing we do a sanity check
we know that top2b mice have top2b deleted from their heart cell
so we can confirm that they have lower expression of top2b
we do this sanity check by creating a boxplot
base code
boxplot(<y-axis> ~ <x-axis>, main = “<title>”)
for gene expression
boxplot(<gene expression> ~ <phenotype>, main = “<feature>”)
for our current example
boxplot(<Top2b expression> ~ <genotype>, main = “<Top2b info>”)

After boxplot, check for sources of variation
do this with PCA by using plotMDS
*ideally examples will cluster by their experimental group
if they didn't, we have to be concerned for batch effects
and we need to follow up with the experimentalists

Example
install and load limma
load raw data into an ExpressionSet object

```
# Create a new ExpressionSet to store the processed data
eset <- eset_raw
# Log transform
exprs(eset) <- log(exprs(eset))
plotDensities(eset,  group = pData(eset)[, "genotype"], legend = "topright")
# Quantile normalize
exprs(eset) <- normalizeBetweenArrays(exprs(eset))
plotDensities(eset,  group = pData(eset)[, "genotype"], legend = "topright")
# Determine the genes with mean expression level greater than 0
keep <- rowMeans(exprs(eset)) > 0
sum(keep)
# Filter the genes
eset <- eset[keep]
plotDensities(eset, group = pData(eset)[, "genotype"], legend = "topright")
```

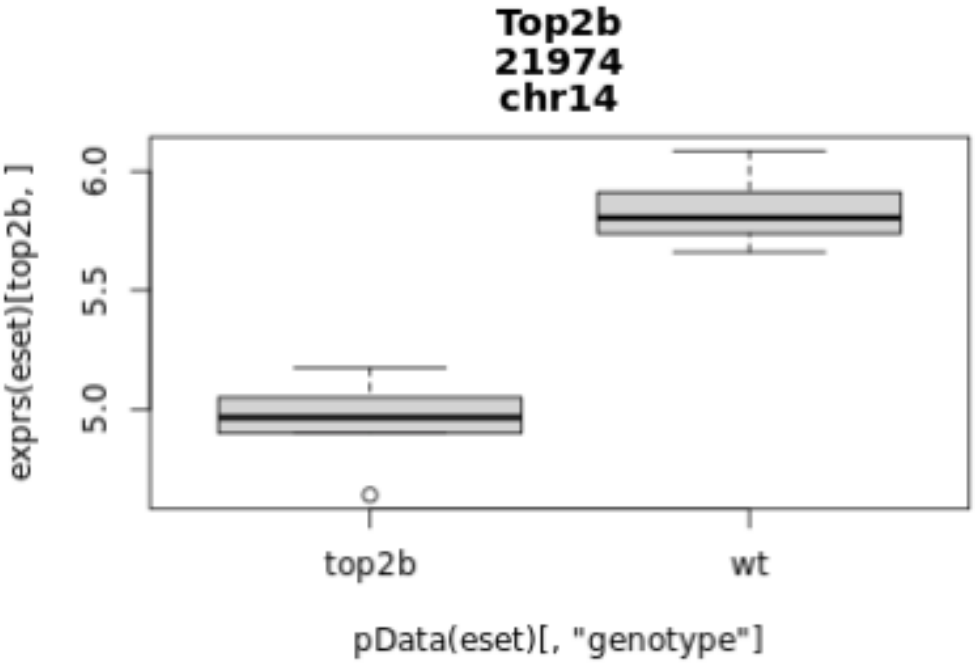
output>




```
# Find the row which contains Top2b expression data
top2b <- which(fData(eset)["symbol"] == "Top2b")
```

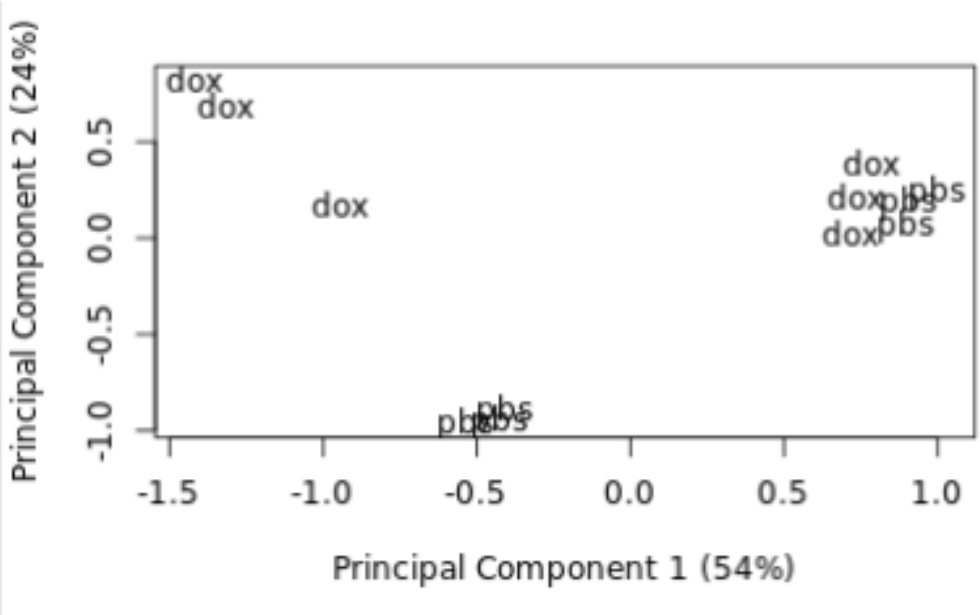
```
# Plot Top2b expression versus genotype
boxplot(exprs(eset)[top2b, ] ~ pData(eset)[, "genotype"],
        main = fData(eset)[top2b, ])
```

output>




```
# Plot principal components labeled by genotype
plotMDS(eset, labels = pData(eset)[, "genotype"], gene.selection = "common")

# Plot principal components labeled by treatment
plotMDS(eset, labels = pData(eset)[, "treatment"], gene.selection = "common")
```

output>



Model the data
our pre-processing steps appear to support our hypothesis
that null mice are resistant to the cardiotoxic effects of doxorubicin
we need to now test this with DE analysis
Steps for DE analysis:

1. build the design matrix with model.matrix
2. construct the contrasts matrix with makeContrasts
3. test the contrasts with lmFit, contrasts.fit, and eBayes

Using group-means parameterization, create a linear model with one coeff for each of the 4 groups

$$Y = \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \epsilon$$

- β_1 - Mean expression level in `top2b` mice treated with `dox`
- β_2 - Mean expression level in `top2b` treated with `pbs`
- β_3 - Mean expression level in `wt` mice treated with `dox`
- β_4 - Mean expression level in `wt` mice treated with `pbs`

each coeff models the mean expression level of the gene in the 3 replicates of that group

Contrasts for doxorubicin study

	β_1	β_2	β_3	β_4
`genotype`	top2b	top2b	wt	wt
`treatment`	dox	pbs	dox	pbs

- Response of wild type mice to dox treatment: $\beta_3 - \beta_4 = 0$
- Response of Top2b null mice to dox treatment: $\beta_1 - \beta_2 = 0$
- Differences between Top2b null and wild type mice in response to dox treatment: $(\beta_1 - \beta_2) - (\beta_3 - \beta_4) = 0$

for this study we are testing three contrasts
interaction term is constructed by contrasting the response to treatment in each genotype separately

Testing the doxorubicin study
perform the hypothesis testing using the limma pipeline
results <- decideTests(fit2)
vennDiagram(results)
the Venn Diagram reveals how many of the differentially expressed genes are shared between the 3 contrasts

Example

```
# Create single variable
group <- with(pData(eset), paste(genotype, treatment, sep = "."))
group <- factor(group)

# Create design matrix with no intercept
design <- model.matrix(~0 + group)
colnames(design) <- levels(group)

# Count the number of samples modeled by each coefficient
colSums(design)
```

output>

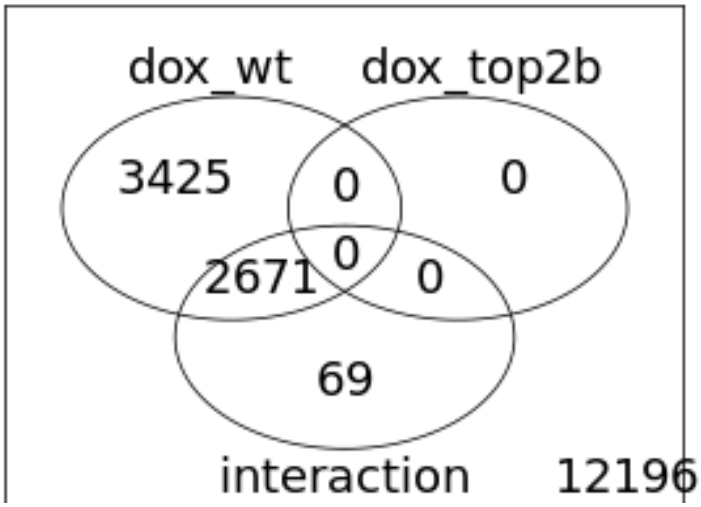
```
top2b.dox top2b.pbs    wt.dox    wt.pbs
        3         3         3         3
```

```
# Create a contrasts matrix
cm <- makeContrasts(dox_wt = wt.dox - wt.pbs,
                    dox_top2b = top2b.dox - top2b.pbs,
```

```
                interaction = (top2b.dox - top2b.pbs) - (wt.dox - wt.pbs),
                levels = design)

# View the contrasts matrix
cm
```

```
           Contrasts
Levels      dox_wt dox_top2b interaction
  top2b.dox      0         1           1
  top2b.pbs      0        -1          -1
  wt.dox         1         0          -1
  wt.pbs        -1         0           1
```

```
# Fit the model
fit <- lmFit(eset, design)

# Fit the contrasts
fit2 <- contrasts.fit(fit, contrasts = cm)

# Calculate the t-statistics for the contrasts
fit2 <- eBayes(fit2)

# Summarize results
results <- decideTests(fit2)
summary(results)

# Create a Venn diagram
vennDiagram(results)

output>
```



	dox_wt	dox_top2b	interaction
Down	3180	0	1254
NotSig	12265	18361	15621
Up	2916	0	1486

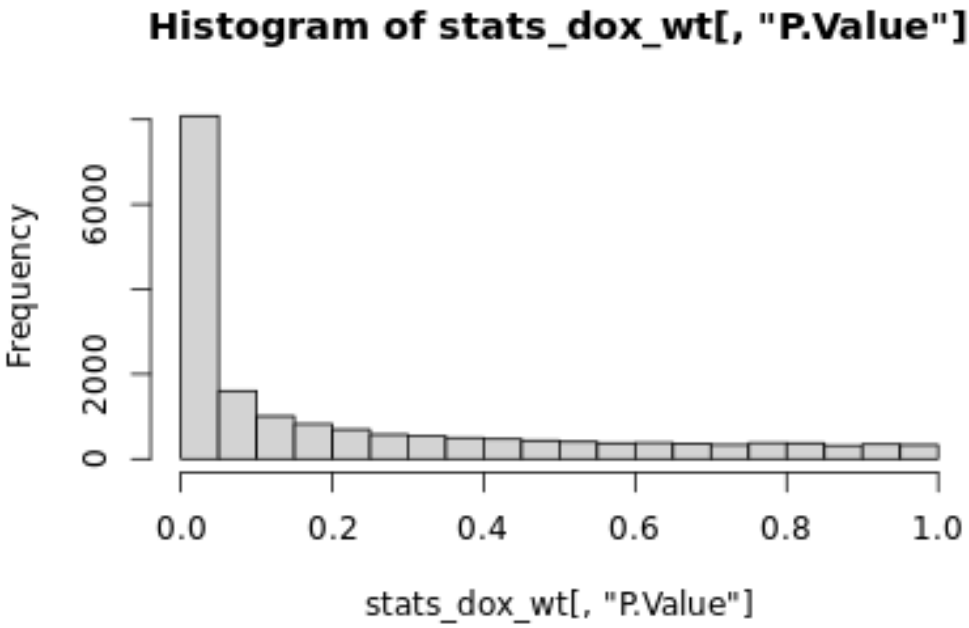
**the Venn diagram shows that doxorubicin disregulated thousands of genes on the WT mice
this supports the original hypothesis that doxorubicin exerts its effect viat Top2b
the genes in the interaction term mainly driven by the effect of doxorubicin in the WT mice
this is evidenced by the large overlap between the two contrasts in the Venn diagram

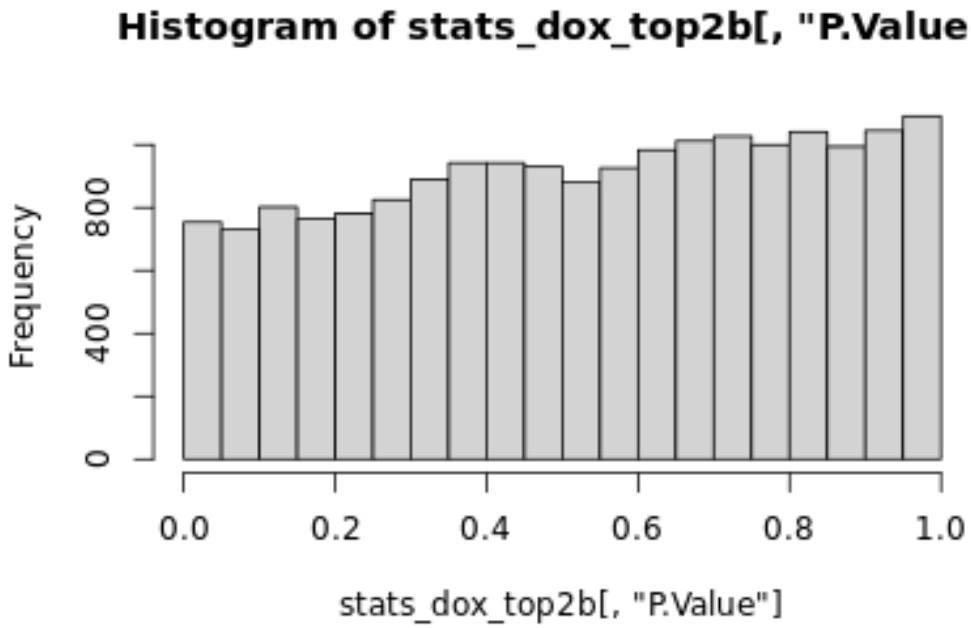
Inspecting the results
confirm that you properly modeled the experiment by plotting the histogram of p-values for every gene
do this with the topTable limma function
then visualize the magnitude of DE and highlight a few of the top genes using the limma function
do this with the volcanoplot
lastly test for pathway-level changes in response to doxorubicin treatment
do this with ‘kegga’ and ‘topKEGG’
*this identifies pathways that are enriched for DE genes more than expected by chance

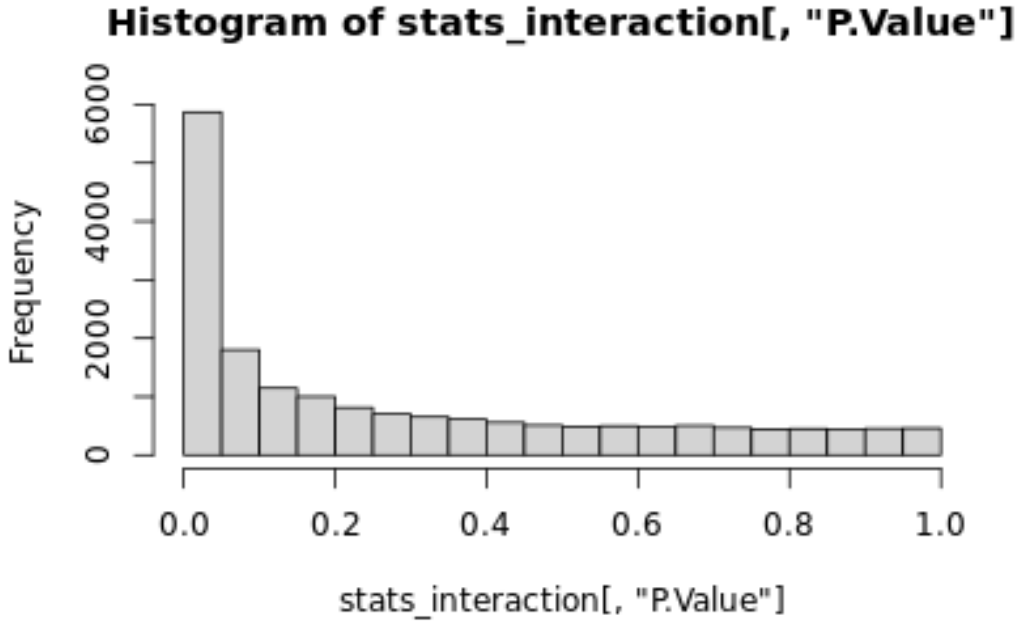
Example
Obtain the summary statistics for the contrast dox_wt

```
stats_dox_wt <- topTable(fit2, coef = "dox_wt", number = nrow(fit2),
                         sort.by = "none")
# Obtain the summary statistics for the contrast dox_top2b
stats_dox_top2b <- topTable(fit2, coef = "dox_top2b", number = nrow(fit2),
                            sort.by = "none")
# Obtain the summary statistics for the contrast interaction
stats_interaction <- topTable(fit2, coef = "interaction", number = nrow(fit2),
                              sort.by = "none")

# Create histograms of the p-values for each contrast
hist(stats_dox_wt[, "P.Value"])
hist(stats_dox_top2b[, "P.Value"])
hist(stats_interaction[, "P.Value"])
```

output>



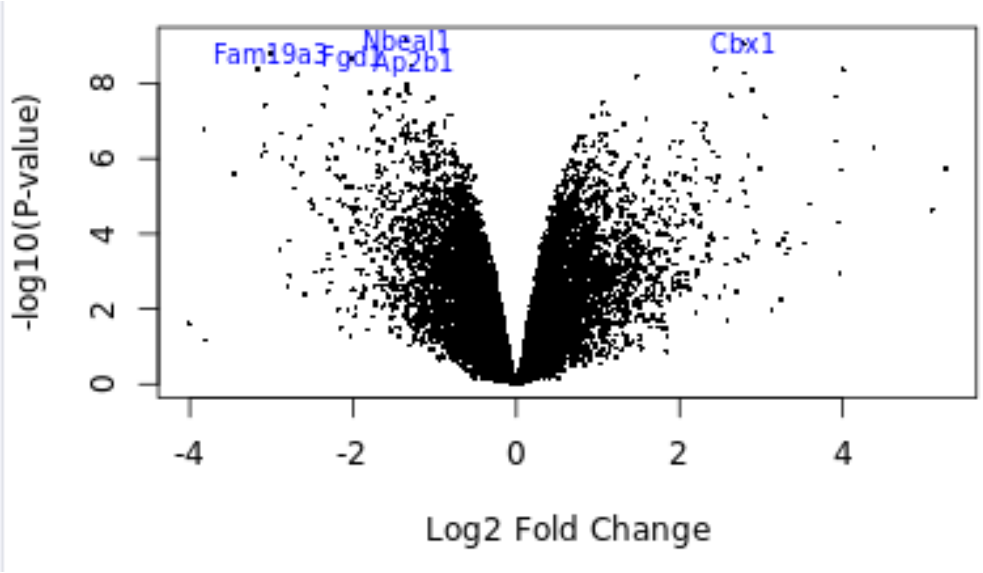


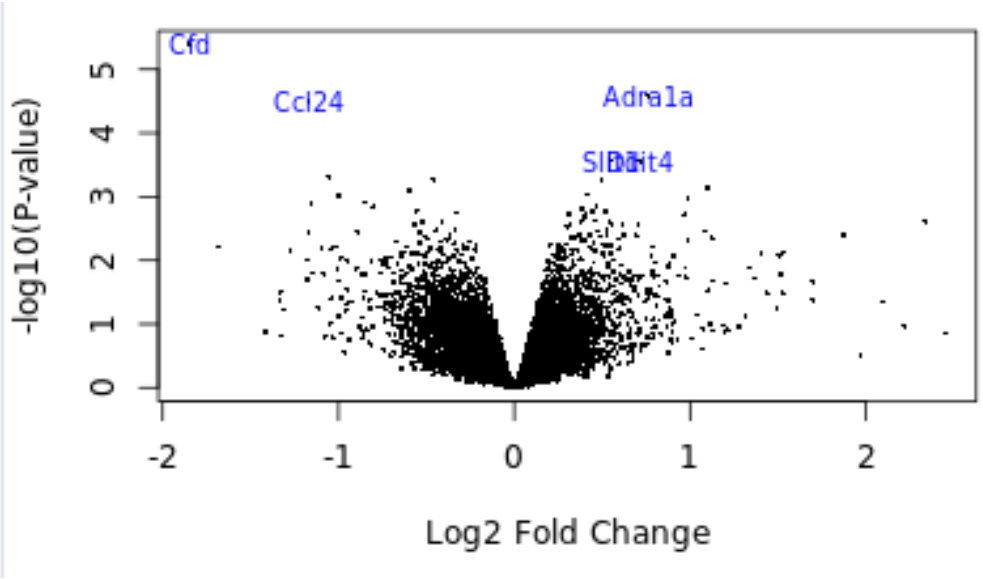


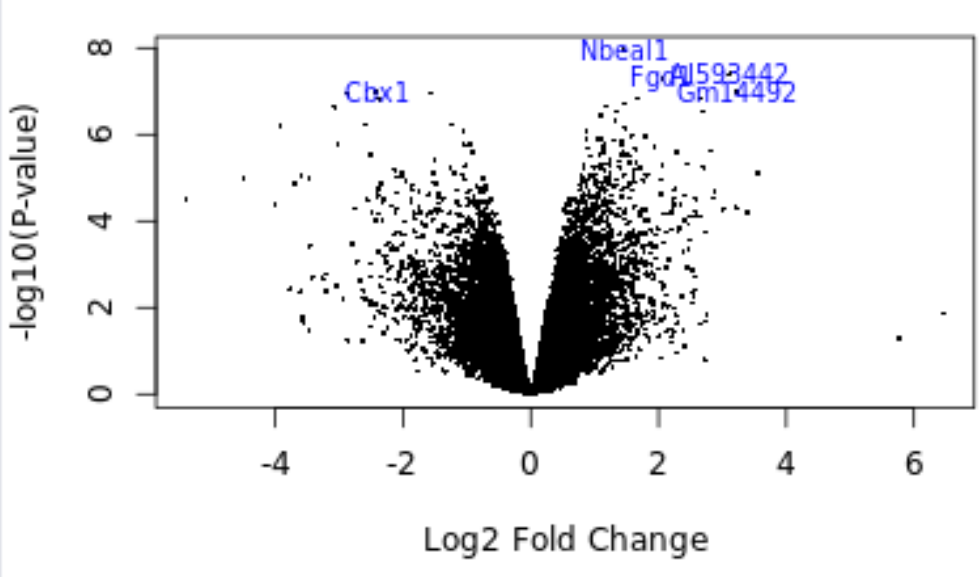

```
# Extract the gene symbols
gene_symbols <- fit2$genes[, "symbol"]

# Create a volcano plot for the contrast dox_wt
volcanoplot(fit2, coef = "dox_wt", highlight = 5, names = gene_symbols)

# Create a volcano plot for the contrast dox_top2b
volcanoplot(fit2, coef = "dox_top2b", highlight = 5, names = gene_symbols)
```

# Create a volcano plot for the contrast interaction
volcanoplot(fit2, coef = "interaction", highlight = 5, names = gene_symbols)

ouput>








```
# Extract the entrez gene IDs
entrez <- fit2$genes[, "entrez"]

# Test for enriched KEGG Pathways for contrast dox_wt
enrich_dox_wt <- kegga(fit2, coef = "dox_wt", geneid = entrez, species = "Mm")

# View the top 5 enriched KEGG pathways
topKEGG(enrich_dox_wt, number = 5)

# Test for enriched KEGG Pathways for contrast interaction
enrich_interaction <- kegga(fit2, coef = "interaction", geneid = entrez, species =
"Mm")

# View the top 5 enriched KEGG pathways
topKEGG(enrich_interaction, number = 5)

output>
```

```
topKEGG(enrich_dox_wt, number = 5)

              Pathway   N Up Down         P.Up         P.Down
path:mmu05322    <NA>  77 37    1 8.678388e-10 9.999999e-01
path:mmu03008    <NA>  71 34    4 4.661802e-09 9.996444e-01
path:mmu05034    <NA> 133 47   17 7.679832e-07 9.727465e-01
path:mmu05412    <NA>  62  3   28 9.994243e-01 1.493108e-06
path:mmu04613    <NA> 128 44   22 3.968210e-06 6.984633e-01
```

```
topKEGG(enrich_interaction, number = 5)

              Pathway   N Up Down          P.Up         P.Down
path:mmu05322    <NA>  77  0   27 1.0000000000 8.289360e-12
path:mmu04613    <NA> 128  9   31 0.8309868479 8.514544e-09
path:mmu03008    <NA>  71  1   19 0.9988453741 1.277807e-06
path:mmu05034    <NA> 133  9   27 0.8619949817 3.234677e-06
path:mmu05412    <NA>  62 17    2 0.0000242473 9.592574e-01
```