Managing Batch Effects in Microbiome Data

Yiwen Wang

2019-06-18

Contents

Chapter 1

Examples of microbiome studies with batch effects

This vignette provides all the analyses performed in the paper "Managing Batch Effects in Microbiome Data".

Packages installation and loading

First, you will need to install then load the following packages:

```
#cran.packages = c('knitr','mixOmics','xtable','ggplot2','vegan','cluster','gridExtra','pheatmap','ruv'
#install.packages(cran.packages)
#bioconductor.packages = c('sva','limma','AgiMicroRna','variancePartition','pvca')
#if (!requireNamespace("BiocManager", quietly = TRUE))
#    install.packages("BiocManager")
#BiocManager::install(bioconductor.packages, version = "3.8")

library(knitr)
library(xtable) # table
library(mixOmics)
library(sva) # ComBat
library(ggplot2) # PCA sample plot with density
library(gridExtra) # PCA sample plot with density
library(limma) # removeBatchEffect (LIMMA)
library(vegan) # RDA
library(AgiMicroRna) # RLE plot
library(cluster) # silhouette coefficient
library(variancePartition) # variance calculation
library(pvca) # PVCA
library(pheatmap) # heatmap
library(ruv) # RUVIII
library(lmerTest) #lmer
library(bapred) # FAbatch
```

1.1 Study description

1.1.1 Sponge *Aplysina aerophoba* study

Sacristán-Soriano *et al.* studied the potential involvement of bacterial communities from the sponge species *A. aerophoba* in the biosynthesis of brominated alkaloids (BAs) (Sacristán-Soriano et al., 2011). They compared the microbial composition and BA concentration in two different tissues (ectosome and choanosome) to investigate the relationship between bacterial composition and BA concentration. The authors concluded that differences in bacterial profiles were not only due to tissue variation (the main effect of interest), but also because the samples were run on two separate denaturing gradient gels during processing. Gel thus acted as a technical batch effect as described in Table 1.

1.1.2 Anaerobic digestion study

Anaerobic Digestion (AD) is a microbiological process of organic matter degradation that produces a biogas used in electrical and thermal energy production. However, the AD bioprocess undergoes inhibition during its developmental stage that is not well characterised: Chapleur *et al.* explored microbial indicators that could improve the AD bioprocess's efficacy and prevent its failure (Chapleur et al., 2016). They profiled the microbiota of 75 AD samples in various conditions. Here we consider two different ranges of phenol concentration as treatments. The experiment was conducted at different dates (5), which constitutes a technical source of unwanted variation (Table 1).

1.1.3 Huntington's disease study

In their study, Kong *et al.* reported differences in microbial composition between Huntington's disease (HD) and wild-type (WT) mice (Kong et al., 2018). However, the establishment of microbial communities was also driven by biological batch effects: the cage environment and sex. Here we consider only female mice to illustrate a special case of a batch $\times$ treatment unbalanced design. The HD data include 13 faecal mice samples hosted across 4 cages (Table 1).

We load the data and functions that are provided *outside* the packages.

```r
# load the data
load(file = './datasets/microbiome_datasets.RData')

# load the extra functions
source(file = './Functions.R')
dim(sponge.tss)
```

```
## [1] 32 24
```

```r
dim(ad.count)
```

```
## [1]  75 567
```

```r
dim(hd.count)
```

```
## [1]  13 368
```

Note: the AD data and HD data loaded are raw counts, while sponge data are total sum scaling (TSS) scaled data calculated on raw counts, with no offset.

1.2 Data processing

Data processing steps for microbiome data:

1. Prefilter the count data to remove features with excess zeroes across all samples
2. Add an offset of 1 to the whole data matrix
3. Total Sum Scaling transformation
4. Log-ratio transformation

1.2.1 Prefiltering

We use a prefiltering step to remove OTUs for which the sum of counts are below a set threshold (0.01%) compared to the total sum of all counts from (Arumugam et al., 2011).

```r
# ad data
ad.index.keep = which(colSums(ad.count)*100/(sum(colSums(ad.count))) > 0.01)
ad.count.keep = ad.count[,ad.index.keep]
dim(ad.count.keep)
```

```
## [1]  75 231
```

```r
# hd data
#hd.index.keep = which(colSums(hd.count)*100/(sum(colSums(hd.count))) > 0.001)
hd.count.keep = hd.count
dim(hd.count.keep)
```

```
## [1]  13 368
```

Compared with the raw counts, many OTUs below the threshold are removed. As HD data are small part of a big dataset and only have 13 samples, it is too strict to use the threshold 0.01% to filter the OTUs. We then maintained all the OTUs in HD data.

1.2.2 Total sum scaling

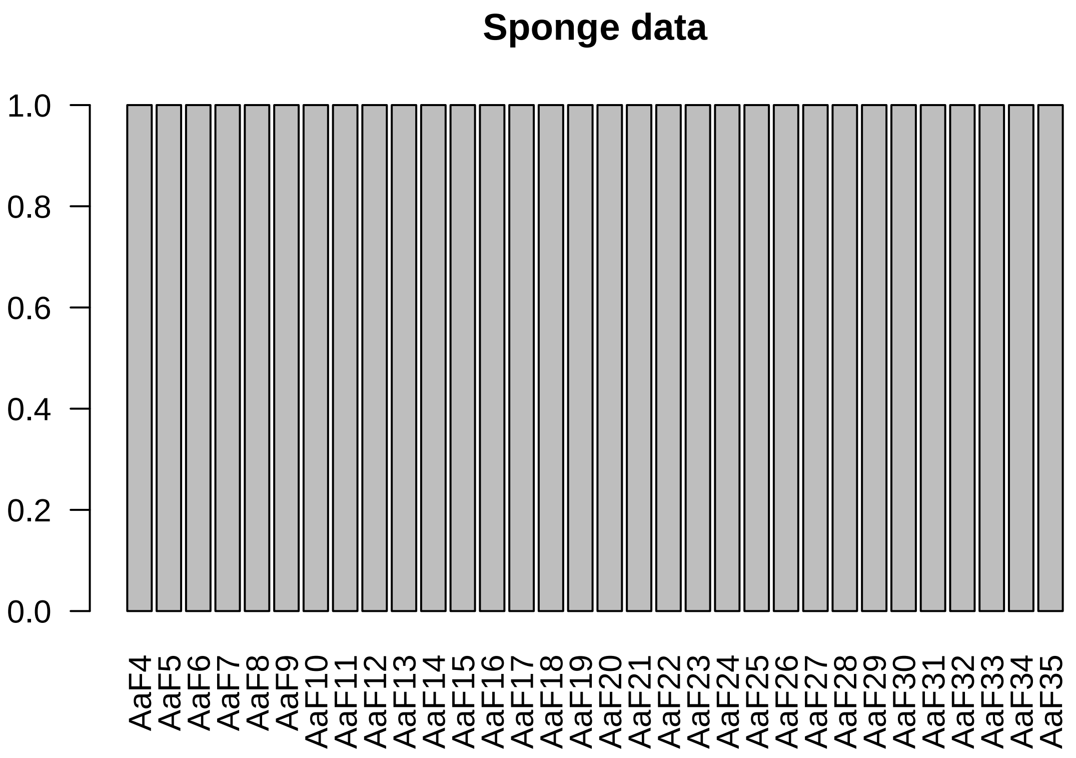
Now, let's apply a TSS scaling on the filtered data:

Note: We need to add an offset of 1 to all count data (i.e. count data = count.keep + 1). Although sponge data are already TSS data, a 'tiny' offset is added and then TSS is redone. Because the presence of zeroes will make the next step – log-ratio transformation impossible.

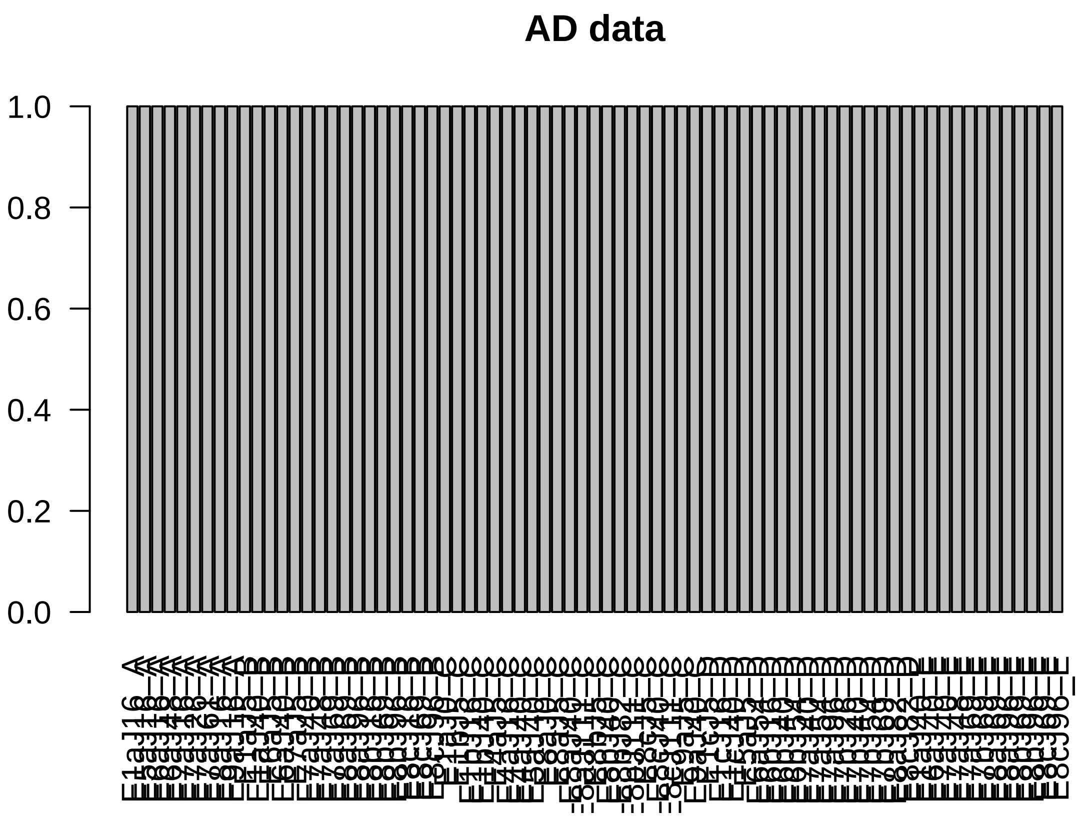
```r
# sponge data
sponge.tss = t(apply(sponge.tss + 0.01, 1, TSS.divide))

# ad data
ad.tss = t(apply(ad.count.keep + 1, 1, TSS.divide))

# hd data
hd.tss = t(apply(hd.count.keep + 1, 1, TSS.divide))
```

Check the library size sums to 1 for each sample. We now have compositional data.

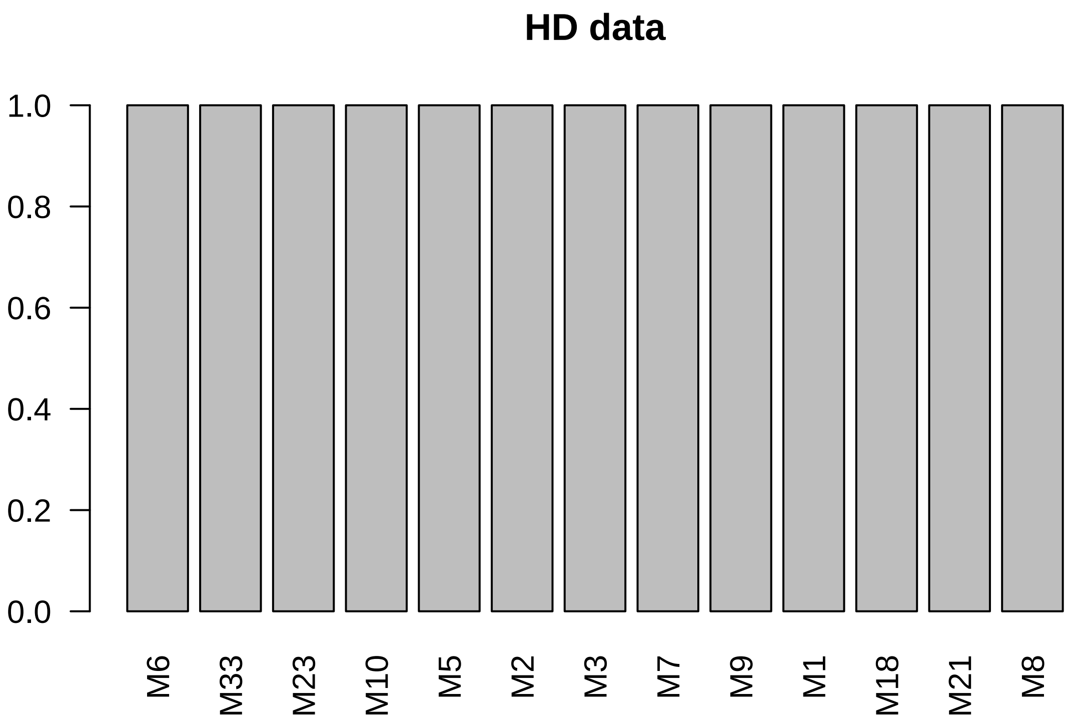
```r
# sponge data
sponge.lib.size.tss <- apply(sponge.tss, 1, sum)
barplot(sponge.lib.size.tss, main =  'Sponge data',las=2)
```



```r
# ad data
ad.lib.size.tss <- apply(ad.tss, 1, sum)
barplot(ad.lib.size.tss, main =  'AD data',las=2)
```



```
# hd data
hd.lib.size.tss <- apply(hd.tss, 1, sum)
barplot(hd.lib.size.tss, main =  'HD data',las=2)
```



1.2.3 Centered log-ratio transformation

TSS results in compositional data (or proportions) that are restricted to a space where the sum of all OTU proportions for a given sample sums to 1. Using standard statistical methods on such data may lead to spurious results and therefore the data must be further transformed. The CLR is the transformation of choice.

```
# sponge data
sponge.tss.clr <- logratio.transfo(sponge.tss,logratio = 'CLR')
class(sponge.tss.clr) <- 'matrix'

# ad data
ad.tss.clr <- logratio.transfo(ad.tss,logratio = 'CLR')
class(ad.tss.clr) <- 'matrix'

# hd data
hd.tss.clr <- logratio.transfo(hd.tss,logratio = 'CLR')
class(hd.tss.clr) <- 'matrix'
```

The final TSS-CLR data of sponge study, AD study and HD study contain 32 samples and 24 OTUs, 75 samples and 231 OTUs, 13 samples and 368 OTUs, respectively as described in Table 1.

Table 1. Overview of exemplar datasets with batch effects. We considered microbiome studies from sponge *Aplysina aerophoba*; organic matter in anaerobic digestion (AD) and mice models with Huntington's disease (HD).

	Sponge data			**AD data**			**HD data**		
No. of OTUs	24			231			368		
No. of samples	32			75			13		
Design type	Balanced			Approx. balanced			Unbalanced		
Organism	Sponge samples			Organic matter			Fecal samples		
Batch sources	Gel (sample processing)			Date (sample processing and sequencing)			Cage (housing)		
Batch × treatment design		Ectosome	Choanosome		0-0.5	1-2		HD	WT
	Gel 1	8	8	09/04/2015	4	5	Cage F	0	3
				14/04/2016	4	12	Cage G	3	0
				01/07/2016	8	13	Cage H	3	0
	Gel 2	8	8	14/11/2016	8	9	Cage J	0	4
				21/09/2017	2	10			

Chapter 2

Batch effect detection

In this chapter, we apply qualitative methods to assess the presence of batch effects using visualisations.

2.1 Principal component analysis (PCA) with density plot per component

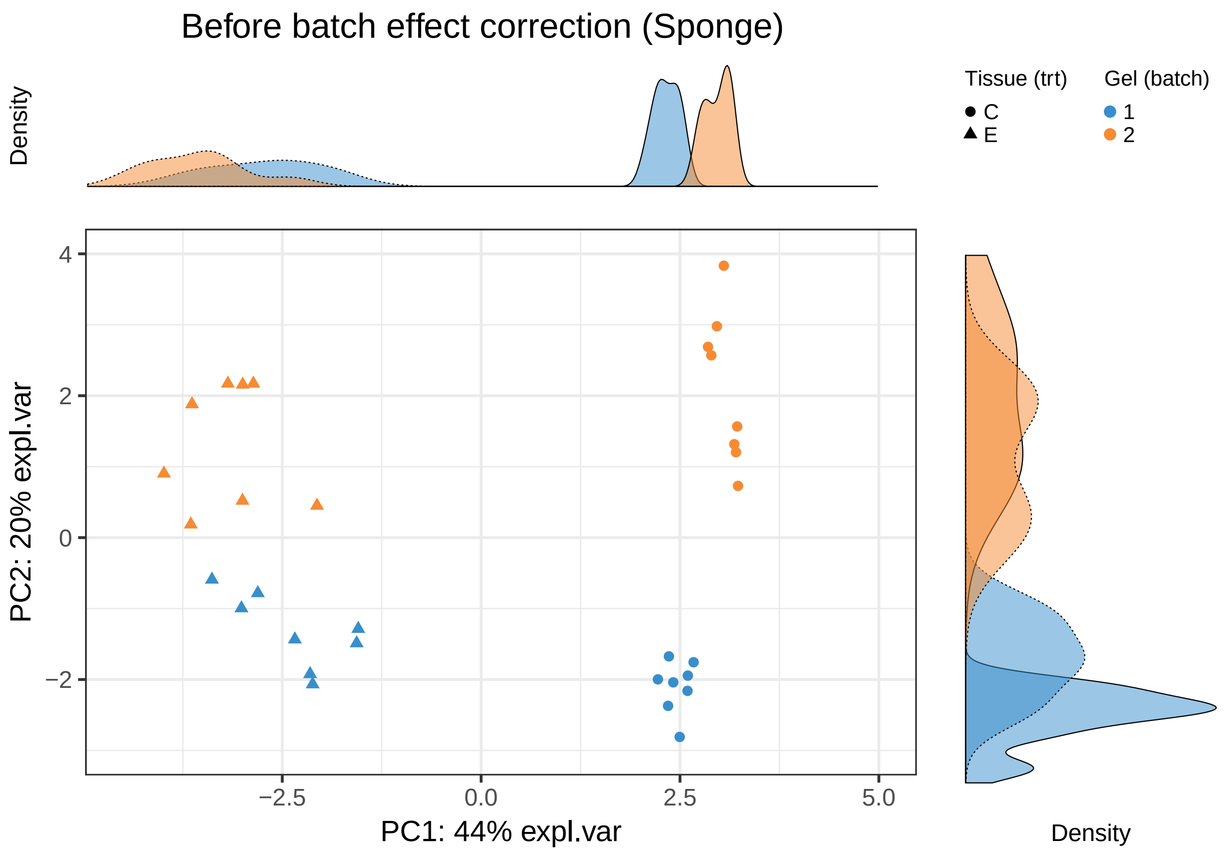
We first start with a PCA sample plot with denisty plot per PC.

PCA is an unsupervised method used to explore the data variance structure by reducing its dimensions to a few principal components (PC) that explain the greatest variation in the data. Density plots are a complementary way to visualise batch effects per PC through examining the distributions of all samples.

```
# sponge data
sponge.pca.before = pca(sponge.tss.clr, ncomp = 3)

# ad data
ad.pca.before = pca(ad.tss.clr, ncomp = 3)

# hd data
hd.pca.before = pca(hd.tss.clr, ncomp = 3)
```

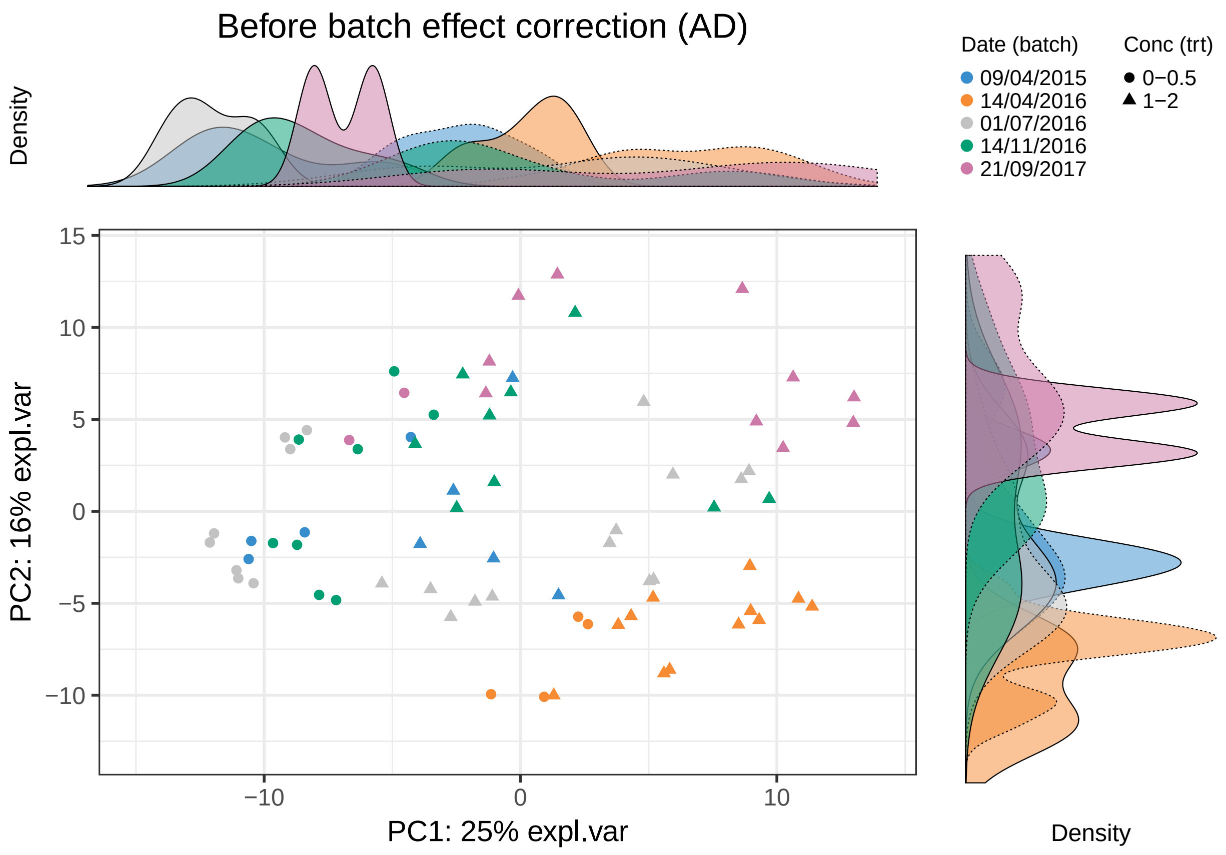
The PCA sample plots detect batch effects easily.

```
Scatter_Density(data = sponge.pca.before$variates$X, batch = sponge.batch, trt = sponge.trt, expl.var =
```



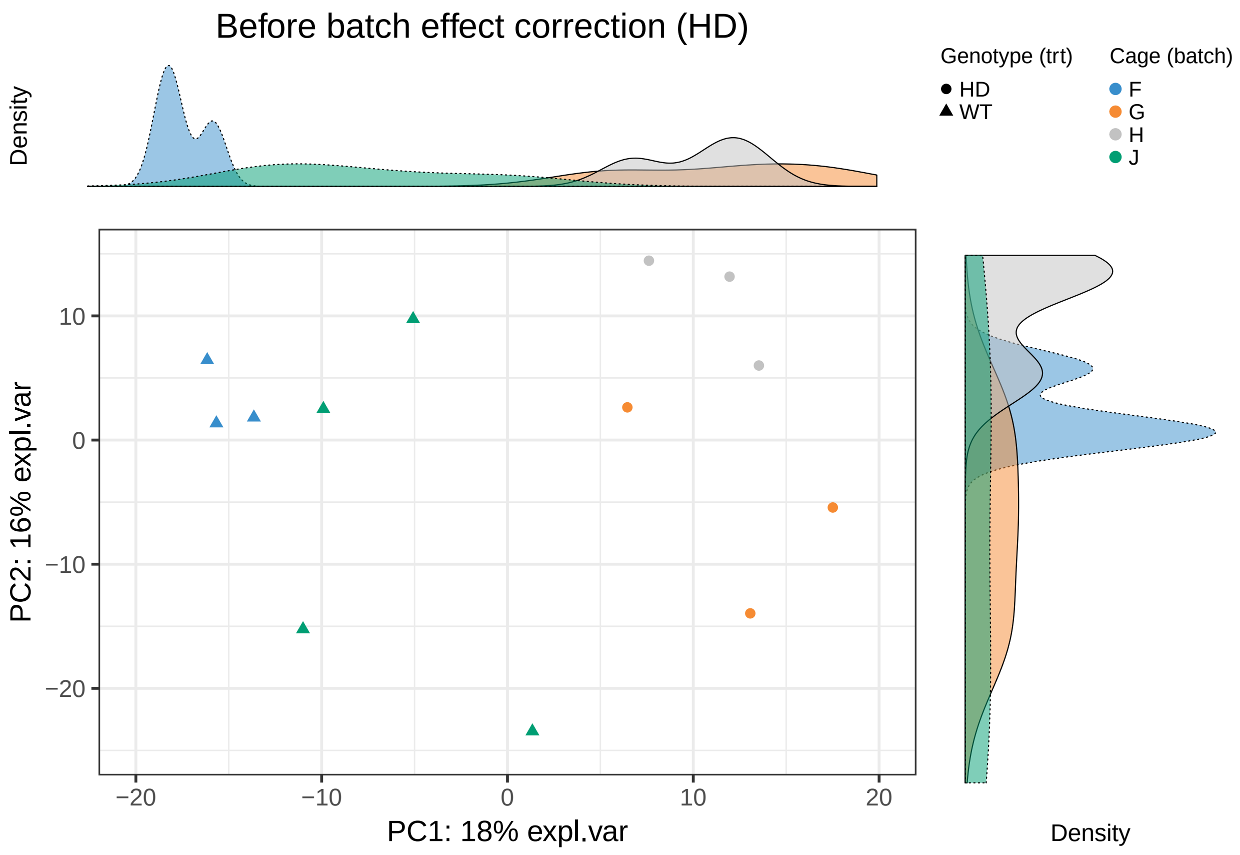
The first PC (explaining the largest source of variation) shows variation between samples from different tissues (the effect of interest), while the second PC (explaining the second largest source of variation) displays sample differences due to different batches, as also highlighted in the density plots per component. Therefore, PCA plots can inform not only of the presence of batch effects, but also which variation is the largest in the data. In this particular dataset, the effect of interest variation is larger than batch variation.

```
Scatter_Density(data = ad.pca.before$variates$X, batch = ad.batch, trt = ad.trt, expl.var = ad.pca.befo
```



In AD data, we observe a separation of samples from batch 14/04/2016.

```
Scatter_Density(data = hd.pca.before$variates$X, batch = hd.batch, trt = hd.trt, expl.var = hd.pca.befo
```



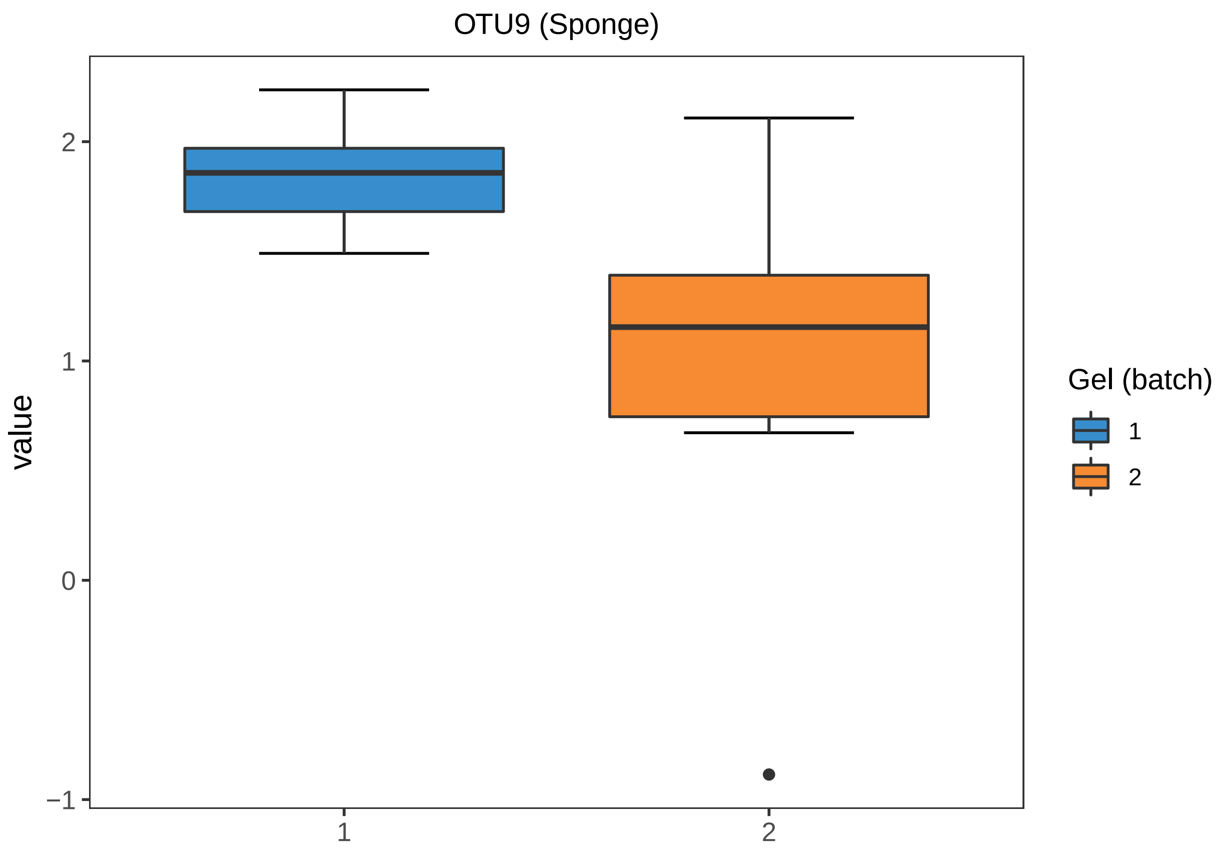
In HD data, batch effect is due to different cages and obvious.

2.2 Density plot and box plot

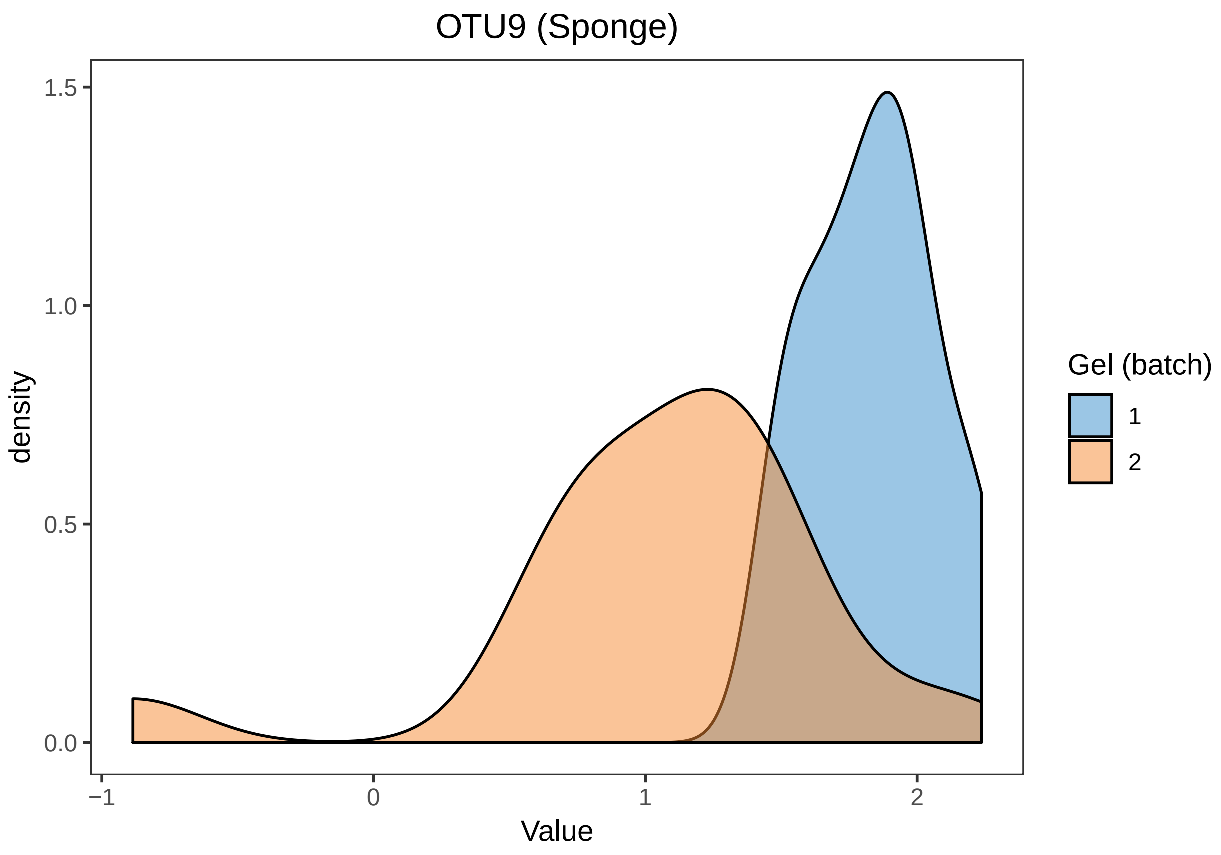
We also apply density plots and box plots on OTUs one at a time from each dataset to visualise batch effects. But only one OTU each dataset is selected as examples.

For each variable (e.g. OTU), density plots and box plots can be generated separately across samples within each batch to observe whether batch effects serve as a major source of variation.

```
# sponge data
sponge.before.df = data.frame(value = sponge.tss.clr[,9], batch = sponge.batch)

box_plot_fun(data = sponge.before.df,x=sponge.before.df$batch,
             y=sponge.before.df$value,title = 'OTU9 (Sponge)',
             batch.legend.title = 'Gel (batch)')
```



```
ggplot(sponge.before.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_fill_manual(
```



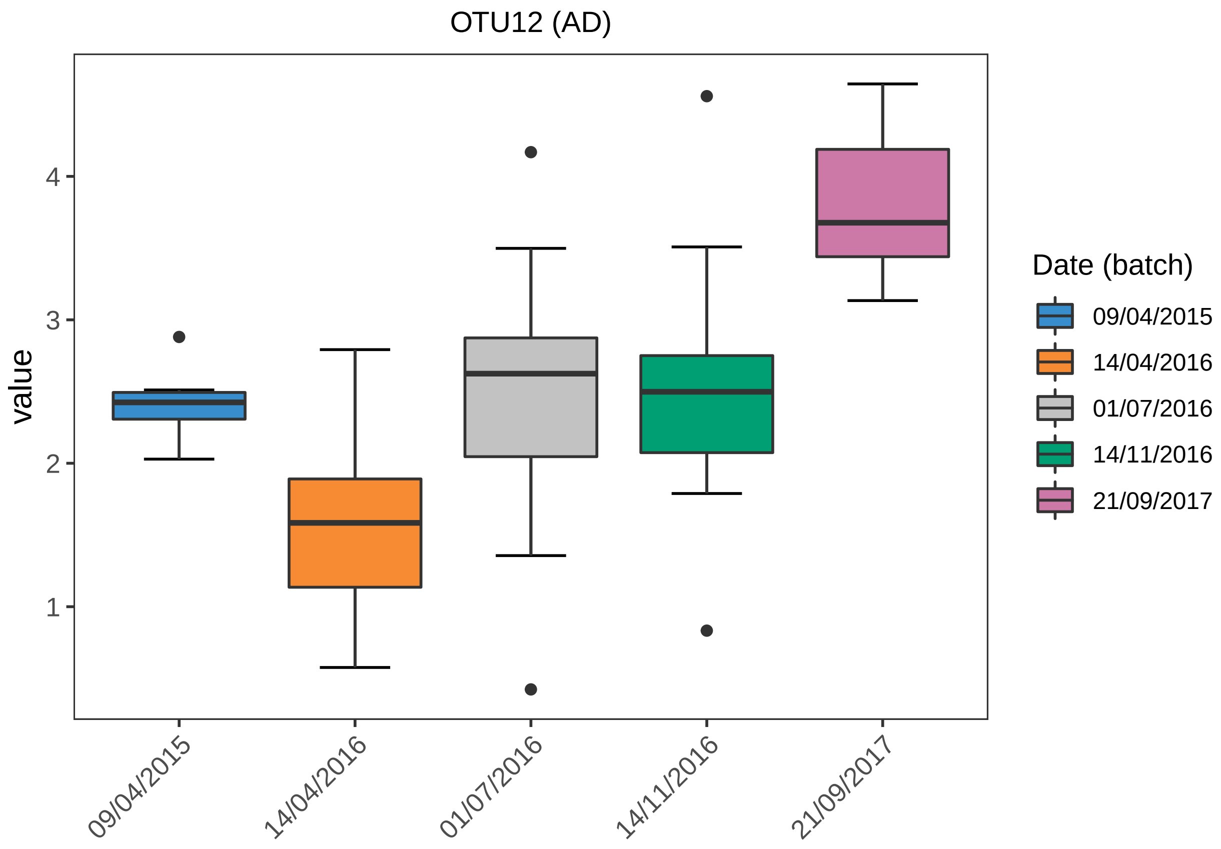
For the abundance of OTU9, the samples within different batches are very distinct, indicating a strong batch effect in sponge data.

```
sponge.lm = lm(sponge.tss.clr[,9]~ sponge.trt + sponge.batch)
summary(sponge.lm)
```

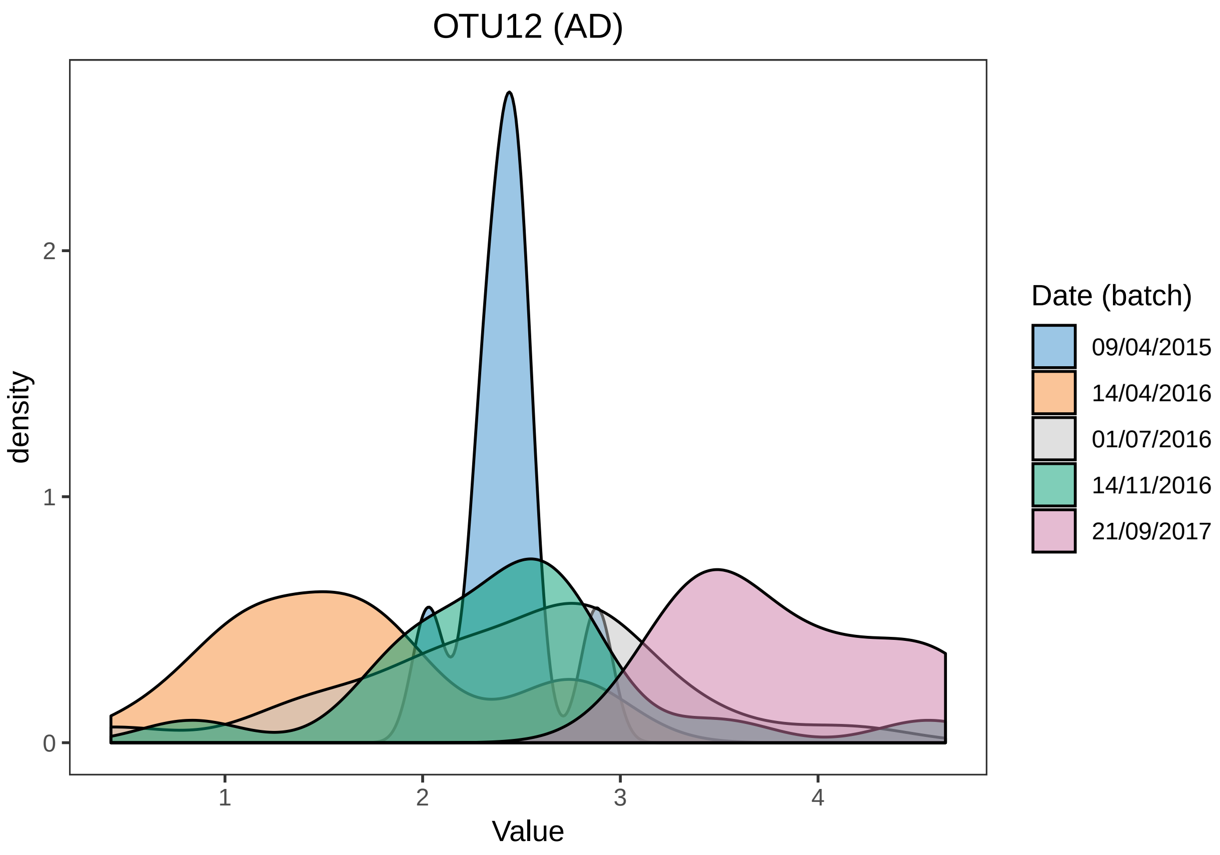
```
##
## Call:
## lm(formula = sponge.tss.clr[, 9] ~ sponge.trt + sponge.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.87967 -0.24705  0.04588  0.24492  1.00757
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)     1.7849     0.1497  11.922 1.06e-12 ***
## sponge.trtE     0.1065     0.1729   0.616    0.543
## sponge.batch2  -0.7910     0.1729  -4.575 8.24e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.489 on 29 degrees of freedom
## Multiple R-squared:  0.4236, Adjusted R-squared:  0.3839
## F-statistic: 10.66 on 2 and 29 DF,  p-value: 0.0003391
```

The batch effect observed with density and box plots was further confirmed with a linear regression model with tissue and batch effects ($P < 0.001$ for the regression coefficient associated with the batch effect in a linear model).

```
##################
# ad data
ad.before.df = data.frame(value = ad.tss.clr[,1], batch = ad.batch)

box_plot_fun(data = ad.before.df,x=ad.before.df$batch,
             y=ad.before.df$value,title = 'OTU12 (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)
```



```
ggplot(ad.before.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_fill_manual(valu
```



Batch effects in AD data are also visualised.

```
ad.lm = lm(ad.tss.clr[,1]~ ad.trt + ad.batch)
anova(ad.lm)
```

```
## Analysis of Variance Table
##
## Response: ad.tss.clr[, 1]
##           Df Sum Sq Mean Sq F value    Pr(>F)
## ad.trt     1  1.460  1.4605  3.1001   0.08272 .
## ad.batch   4 32.889  8.2222 17.4532 6.168e-10 ***
## Residuals 69 32.506  0.4711
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

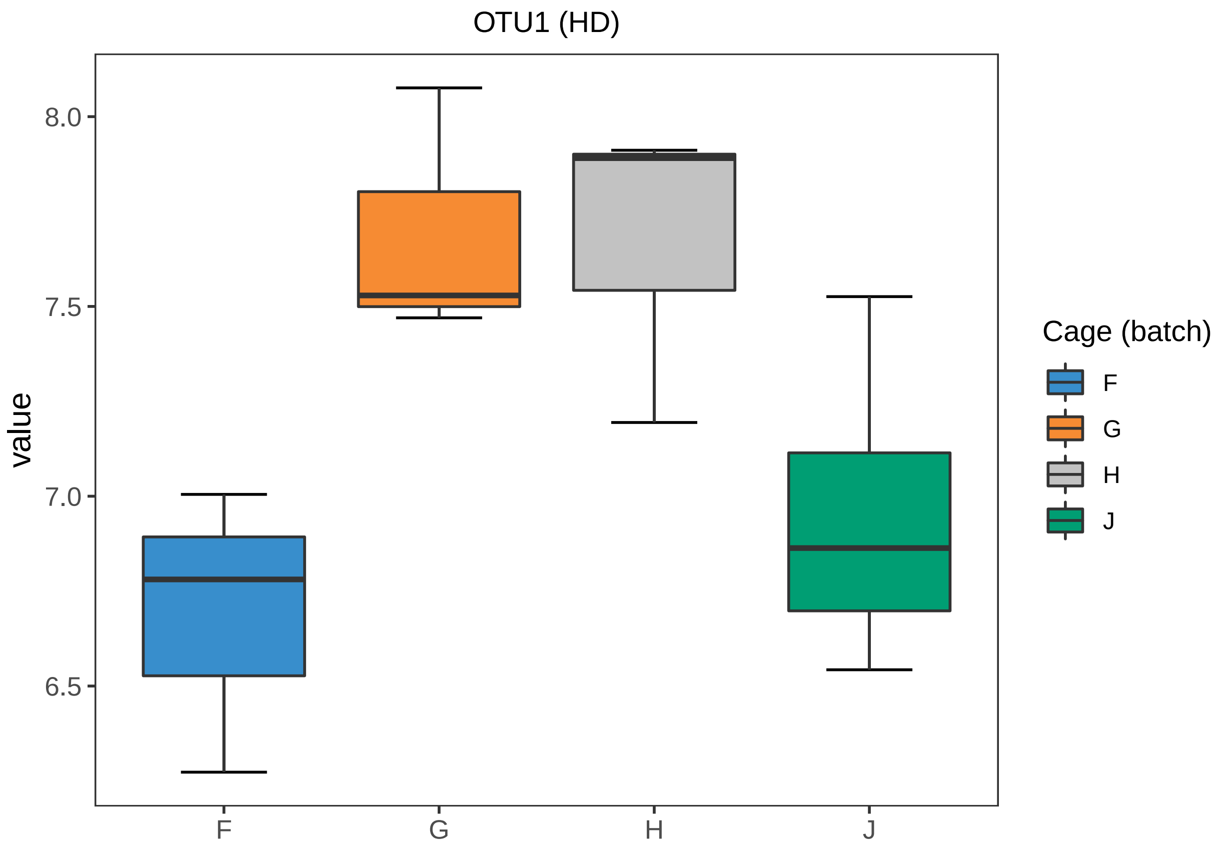
The difference between batches is statistically significant ($P < 0.001$, as tested with ANOVA). We can also obtain P values between each two batch categories.

```
summary(ad.lm)
```

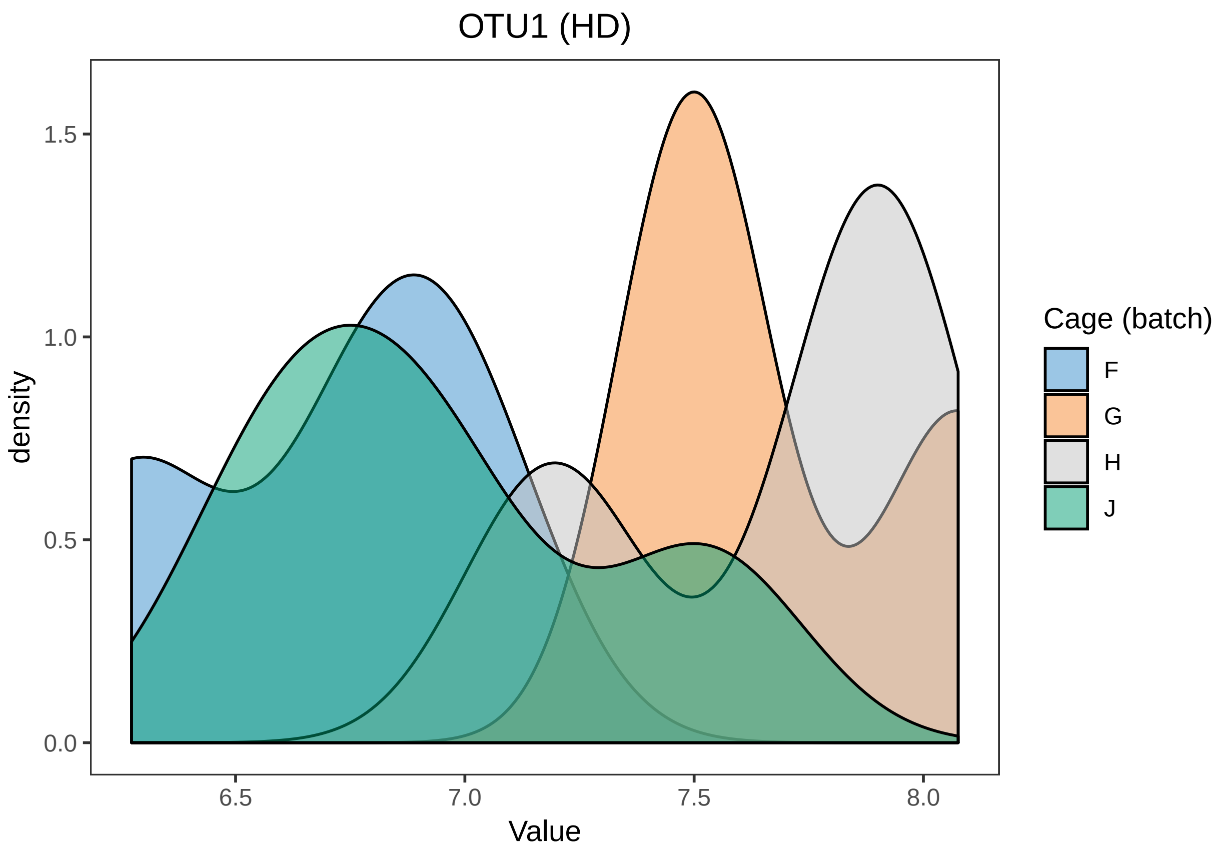
```
##
## Call:
## lm(formula = ad.tss.clr[, 1] ~ ad.trt + ad.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.09885 -0.39613 -0.00381  0.36645  1.98185
##
## Coefficients:
```

```
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)            2.311213   0.247768   9.328 7.57e-14 ***
## ad.trt1-2              0.203619   0.171183   1.189  0.23833
## ad.batch14/04/2016 -0.828100   0.287918  -2.876  0.00535 **
## ad.batch01/07/2016  0.007239   0.273672   0.026  0.97897
## ad.batch14/11/2016  0.062689   0.282978   0.222  0.82533
## ad.batch21/09/2017  1.361132   0.306373   4.443 3.30e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6864 on 69 degrees of freedom
## Multiple R-squared:  0.5138, Adjusted R-squared:  0.4786
## F-statistic: 14.58 on 5 and 69 DF,  p-value: 9.665e-10
```

```
##################
# hd data
hd.before.df = data.frame(value = hd.tss.clr[,1], batch = hd.batch)

box_plot_fun(data = hd.before.df,x=hd.before.df$batch,
             y=hd.before.df$value,title = 'OTU1 (HD)',
             batch.legend.title = 'Cage (batch)')
```



```
ggplot(hd.before.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_fill_manual(valu
```



It is easy to detect the differences between samples within different batches.

```
hd.lm = lm(hd.tss.clr[,1]~ hd.batch)
anova(hd.lm)
```

```
## Analysis of Variance Table
##
## Response: hd.tss.clr[, 1]
##             Df Sum Sq Mean Sq F value  Pr(>F)
## hd.batch    3 2.4108 0.80359  5.2569 0.02276 *
## Residuals   9 1.3758 0.15286
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

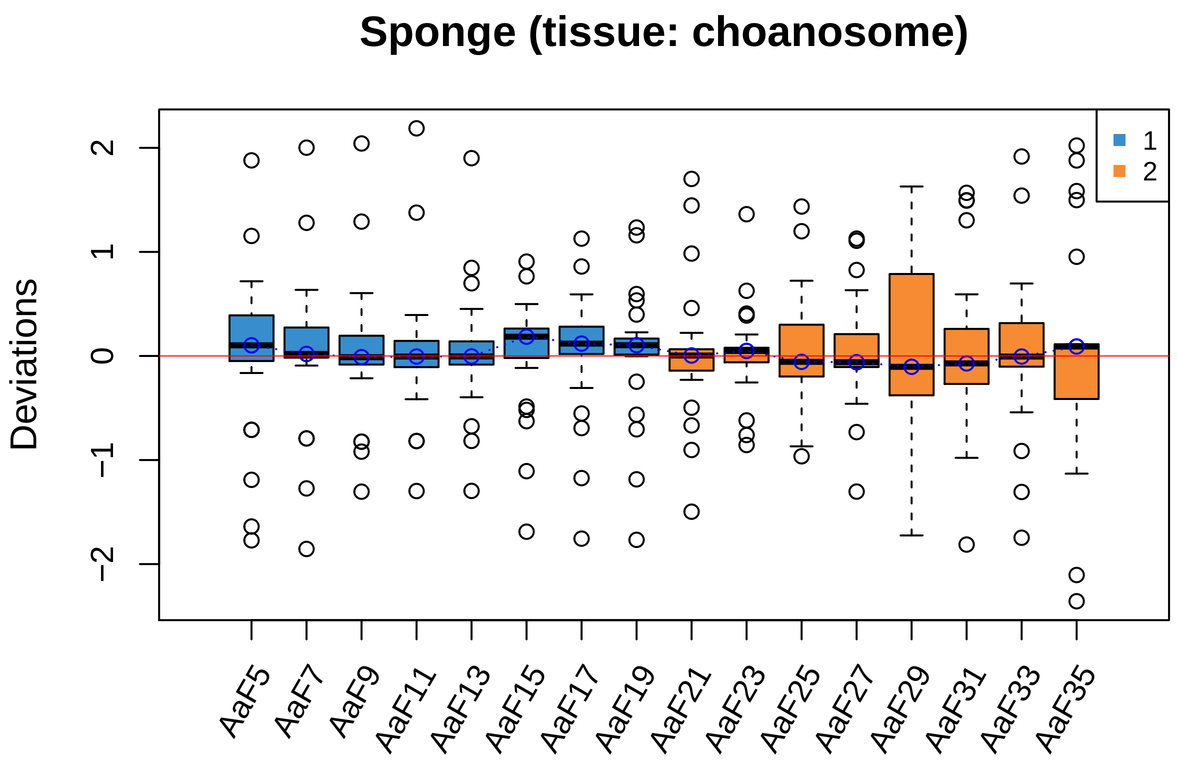
As the batch × treatment design of HD data is nested and unbalanced, the linear model with both treatment (genotype) and batch (cage) is unable to fit. We therefore fit a linear model with batch effect only. The difference between cages is statistically significant ($P < 0.05$). But the difference may also be influenced by treatment and we are unable to exclude this influence.

2.3 RLE plots

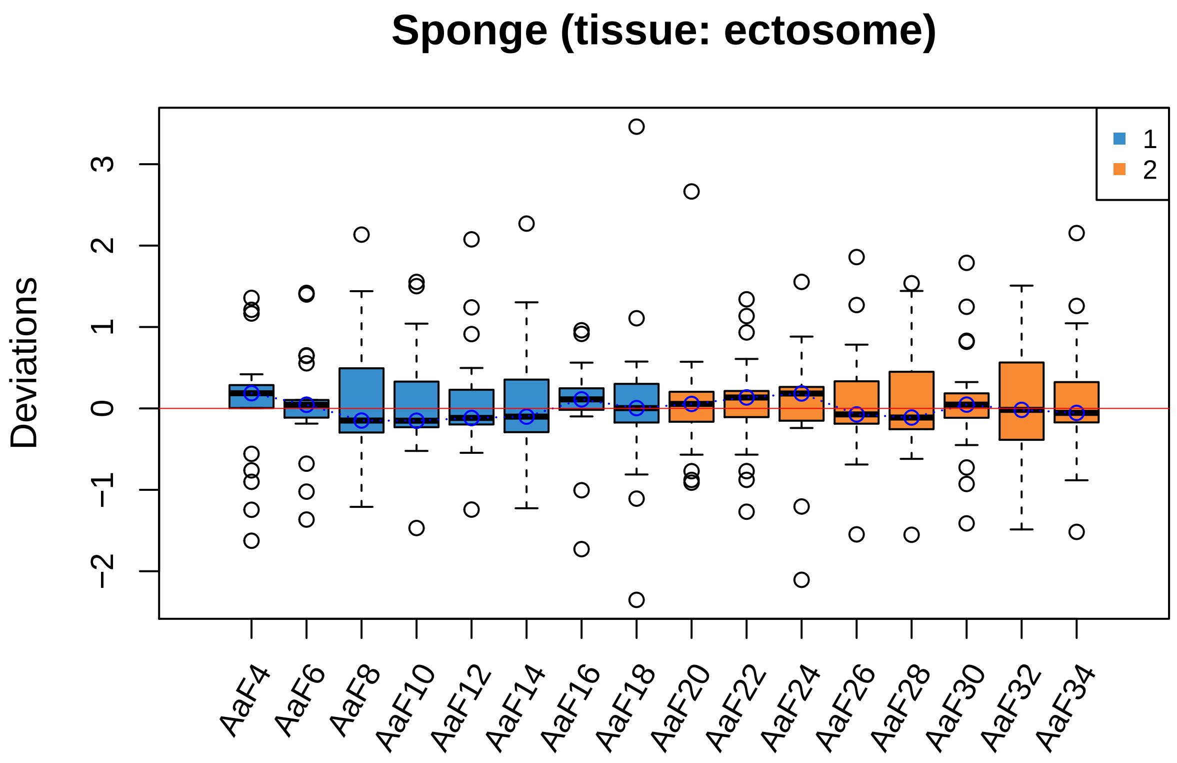
RLE plots can be plotted using ‘RleMicroRna’ in R package ‘AgiMicroRna’. Here, we made some changes on the function ‘RleMicroRna’, thus we call it ‘RleMicroRna2’.

RLE plots are based on the assumption that the majority of microbial variables are unaffected by the effect of interest, and therefore any sample heterogeneity observed - i.e. different distributions and their variances, and medians different from zero, should indicate the presence of batch effects. In our case studies, the treatment information is known, so we generate multiple RLE plots per treatment group, as suggested by (Lin et al., 2018).

```
# sponge data
sponge.batch_c = sponge.batch[sponge.trt == 'C']
sponge.batch_e = sponge.batch[sponge.trt == 'E']

# before
sponge.before_c = sponge.tss.clr[sponge.trt == 'C',]
sponge.before_e = sponge.tss.clr[sponge.trt == 'E',]


RleMicroRna2(object = t(sponge.before_c),batch = sponge.batch_c,maintitle = 'Sponge (tissue: choanosome
```

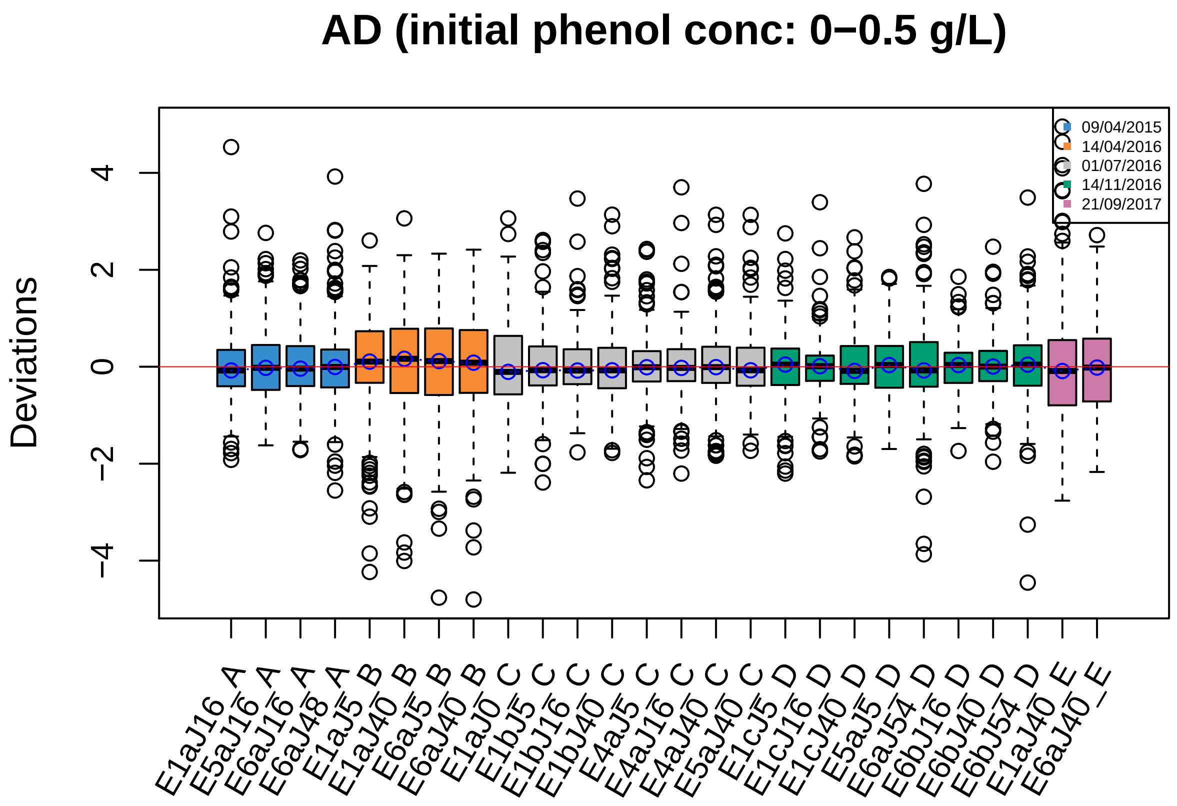


```
RleMicroRna2(object = t(sponge.before_e),batch = sponge.batch_e,maintitle = 'Sponge (tissue: ectosome)'
```

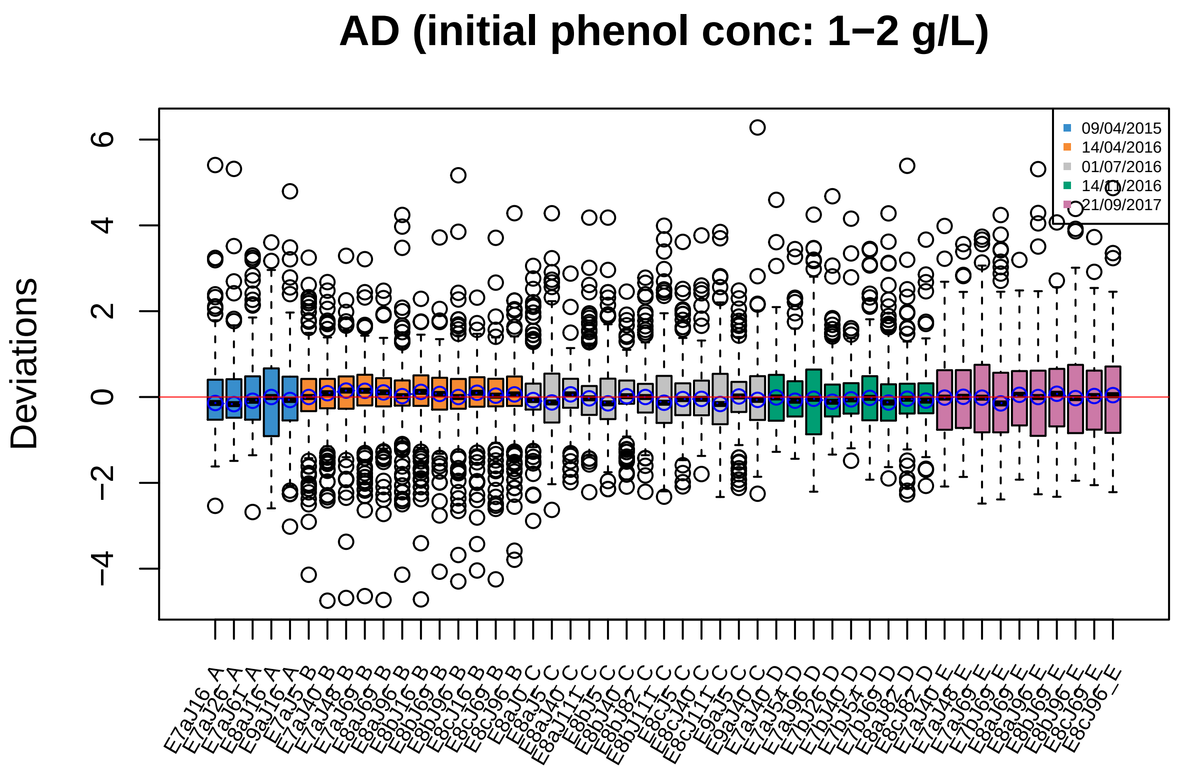


The batch effect before correction is not obvious as all medians of samples are close to zero, but Gel2 has a greater interquartile range (IQR) than the other samples.

```
# ad data
ad.batch_05 = ad.batch[ad.trt == '0-0.5']
ad.batch_2 = ad.batch[ad.trt == '1-2']

# before
ad.before_05 = ad.tss.clr[ad.trt == '0-0.5',]
ad.before_2 = ad.tss.clr[ad.trt == '1-2',]

RleMicroRna2(object = t(ad.before_05),batch = ad.batch_05,maintitle = 'AD (initial phenol conc: 0-0.5 g
```

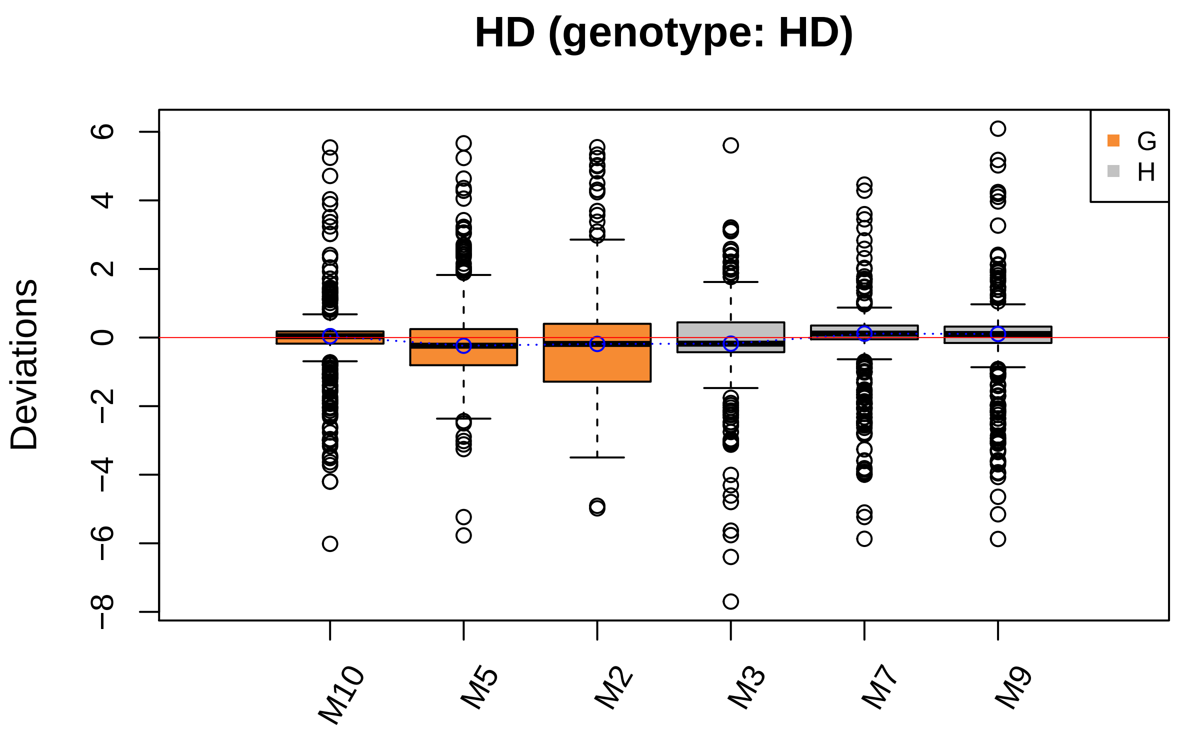


```
RleMicroRna2(object = t(ad.before_2),batch = ad.batch_2,maintitle = 'AD (initial phenol conc: 1-2 g/L)'
```

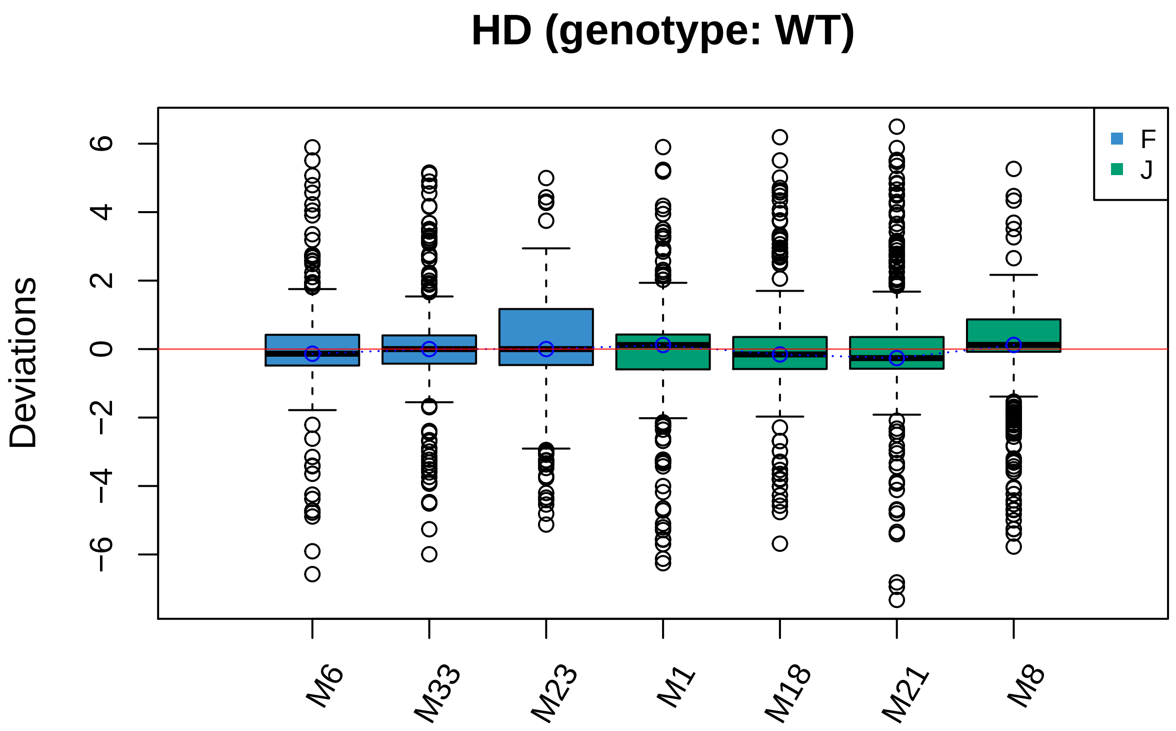


In RLE plots for the AD data, the batch effect before correction is not obvious as all medians of samples are close to zero, but the samples dated 14/04/2016 may be affected by batch.

```r
# hd data ###########
hd.batch_h = hd.batch[hd.trt == 'HD']
hd.batch_w = hd.batch[hd.trt == 'WT']

# before
hd.before_h = hd.tss.clr[hd.trt == 'HD',]
hd.before_w = hd.tss.clr[hd.trt == 'WT',]

RleMicroRna2(object = t(hd.before_h),batch = hd.batch_h,maintitle = 'HD (genotype: HD)')
```



```r
RleMicroRna2(object = t(hd.before_w),batch = hd.batch_w,maintitle = 'HD (genotype: WT)')
```

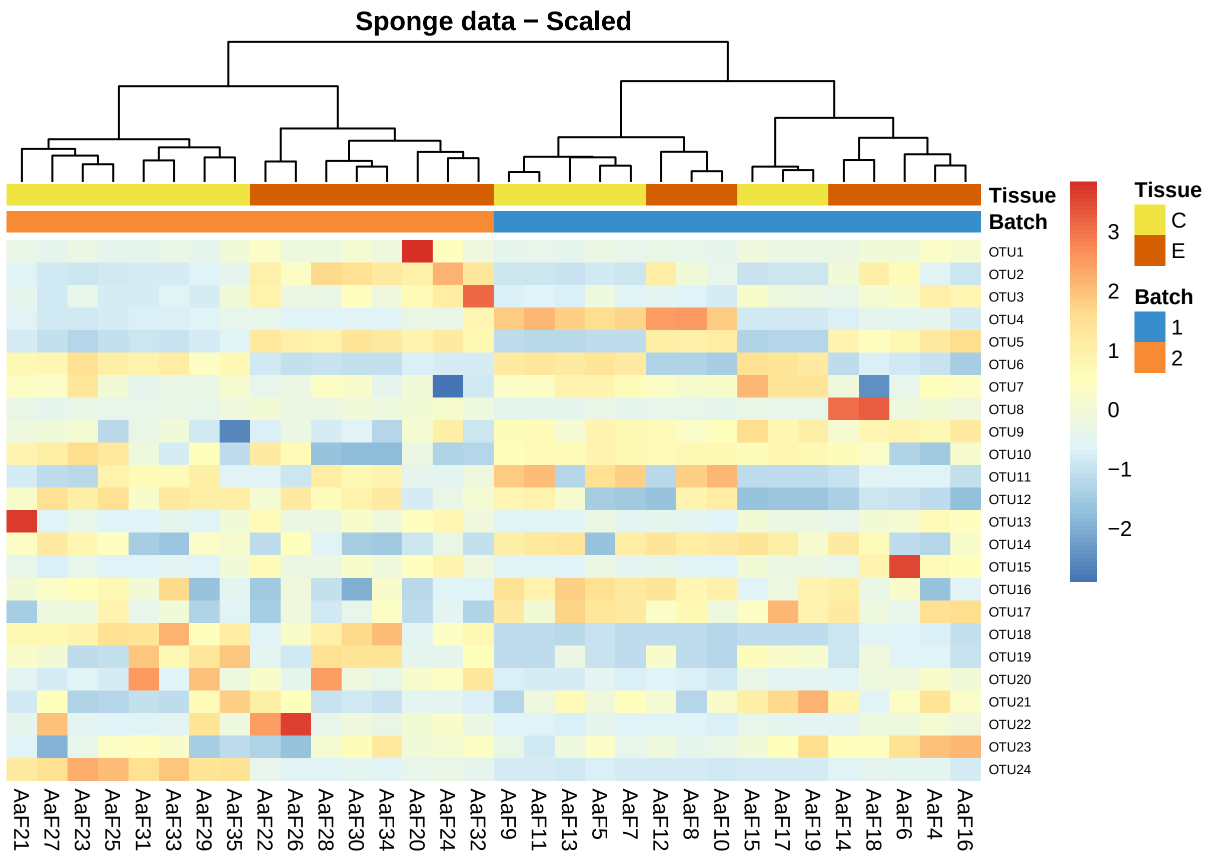


The batch effect in HD data is not easily detected, but Cage G has a greater interquartile range (IQR) than the other samples, which may indicate a batch effect.

2.4 Heatmap

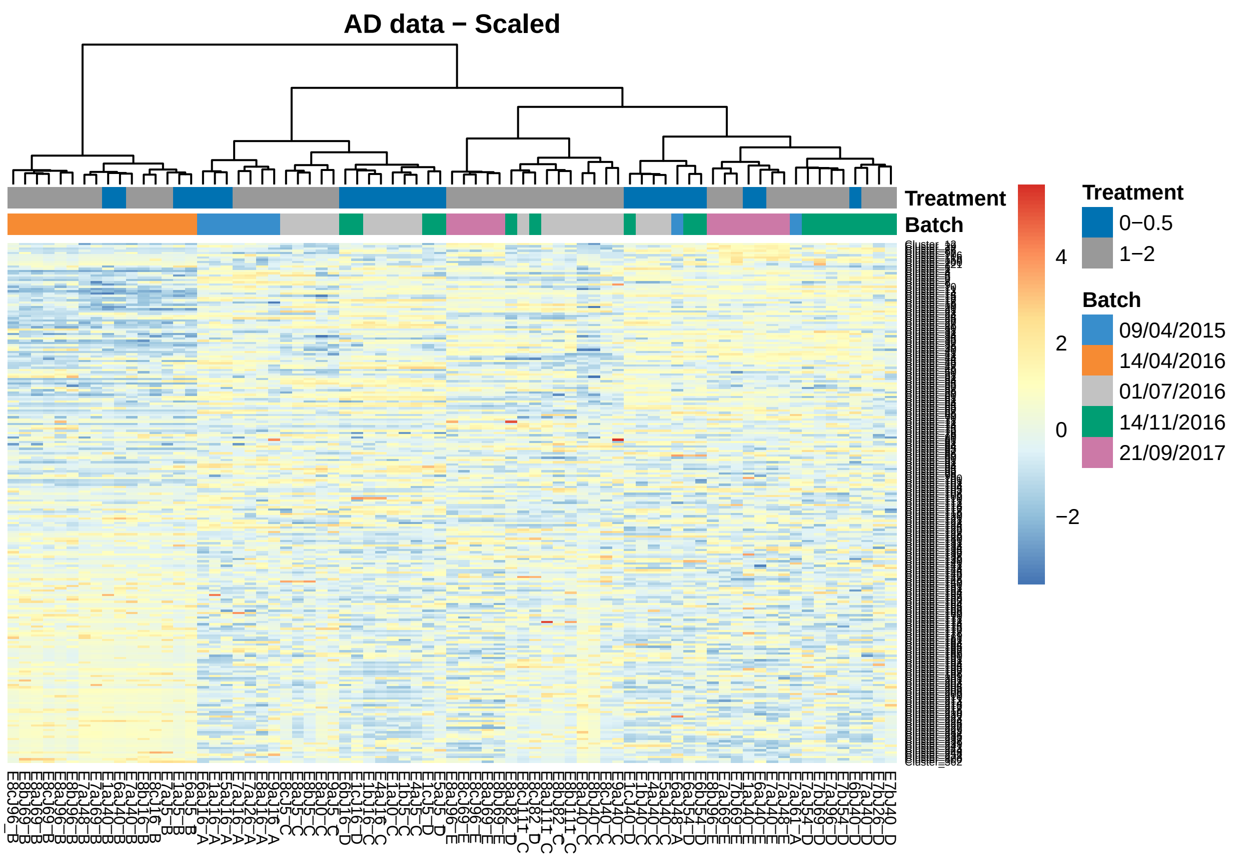
Clustering analysis can be used to detect batch effects. Ideally samples with the same treatment will be clustered together, data clustered by batches instead of treatments indicate a batch effect. Heatmaps and dendrograms are two common approaches to visualise the clusters.

```
# Sponge data
#scale
sponge.tss.clr.scale = scale(sponge.tss.clr,center = T, scale = T) # scale on OTUs
sponge.tss.clr.scale = scale(t(sponge.tss.clr.scale), center = T, scale = T) # scale on samples

sponge.anno_col = data.frame(Batch = sponge.batch, Tissue = sponge.trt)
sponge.anno_metabo_colors = list(Batch = c('1'="#388ECC",'2'="#F68B33"),Tissue = c(C="#F0E442",E="#D55E


pheatmap(sponge.tss.clr.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=5, fontsize_col=8,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = sponge.anno_col,
         annotation_colors = sponge.anno_metabo_colors,
         border_color = 'NA',
         main = 'Sponge data - Scaled')
```



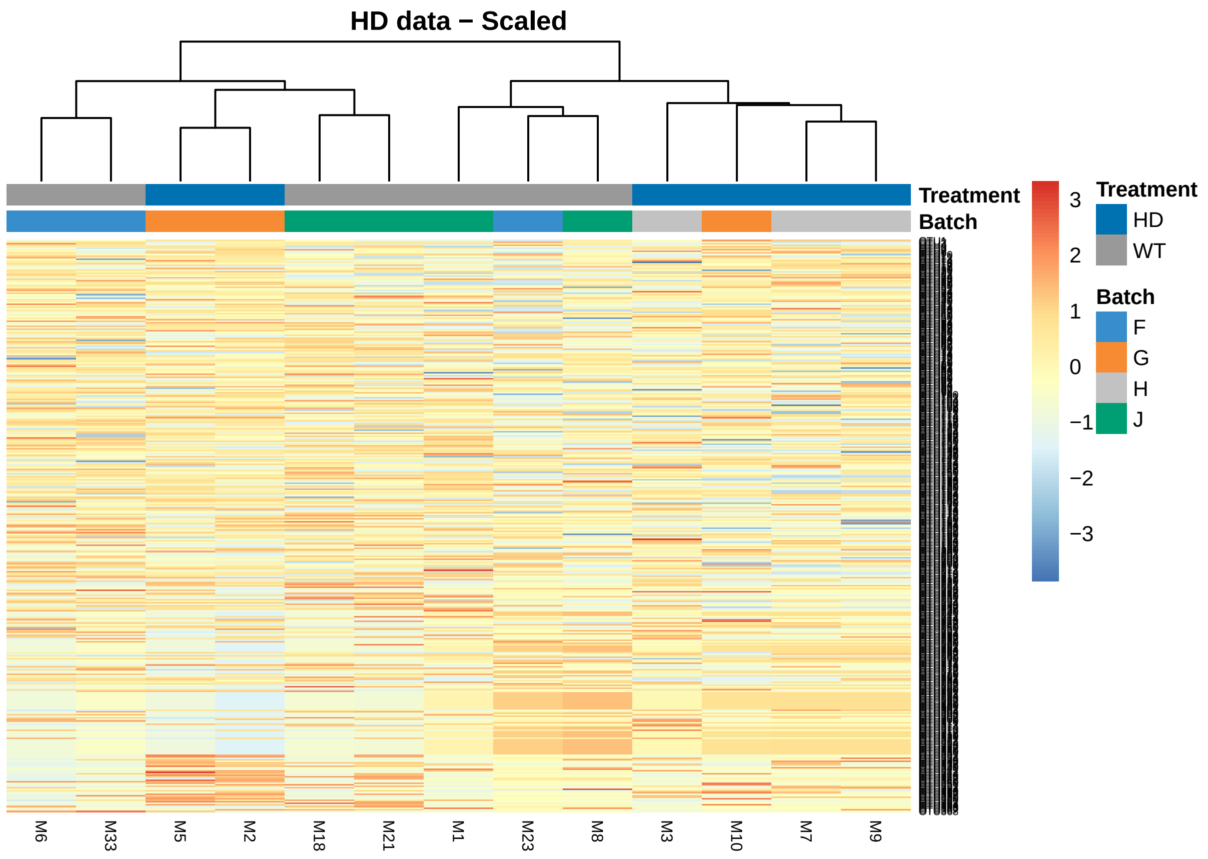
In sponge data, samples are preferentially clustered by batch instead of tissue type, indicating a batch effect.

```
#################
# AD data
#scale
ad.tss.clr.scale = scale(ad.tss.clr,center = T, scale = T) # scale on OTUs
ad.tss.clr.scale = scale(t(ad.tss.clr.scale), center = T, scale = T) # scale on samples

ad.anno_col = data.frame(Batch = ad.batch, Treatment = ad.trt)
ad.anno_metabo_colors = list(Batch = c('09/04/2015'="#388ECC",'14/04/2016'="#F68B33",'01/07/2016'="#C2C


pheatmap(ad.tss.clr.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = ad.anno_col,
         annotation_colors= ad.anno_metabo_colors,
         border_color = 'NA',
         main = 'AD data - Scaled')
```



In AD data, samples within batch 14/04/2016 are clustered and distinct from other samples, also indicating a batch effect.

```r
#################
# HD data
#scale
hd.tss.clr.scale = scale(hd.tss.clr,center = T, scale = T) # scale on OTUs
hd.tss.clr.scale = scale(t(hd.tss.clr.scale), center = T, scale = T) # scale on samples

hd.anno_col = data.frame(Batch = hd.batch, Treatment = hd.trt)
hd.anno_metabo_colors = list(Batch = c('F'="#388ECC",'G'="#F68B33",'H'="#C2C2C2",'J'="#009E73"), Treatm


pheatmap(hd.tss.clr.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = hd.anno_col,
         annotation_colors= hd.anno_metabo_colors,
         border_color = 'NA',
         main = 'HD data - Scaled')
```



The batch effect in HD data is not very obviously visualised through heatmap, because there is no batch with samples clustered together and separated from other samples.

Chapter 3

Batch effect adjustment

3.1 Accounting for batch effects

Methods that account for batch effects estimate unknown batch effects through matrix decomposition and / or assign a known or estimated batch as a covariate with linear models.

3.1.1 Linear model and linear mixed model

LM and LMM are suitable for known batch effects, and can consider batch x treatment interaction and deal with unbalanced batch x treatment design. But they are univariate and rely on a Gaussian likelihood assumption, which may not apply to zero-inflated microbiome data despite CLR transformation.

We fit a linear model for both sponge and AD data.

```
# Sponge data
sponge.trt_p <- apply(sponge.tss.clr, 2, FUN = function(x){
  res.lm <- lm(x ~ sponge.trt + sponge.batch)
  summary.res = summary(res.lm)
  p = summary.res$coefficients[2,4]
})

sponge.trt_q = p.adjust(sponge.trt_p,method = 'fdr')

# AD data
ad.trt_p <- apply(ad.tss.clr, 2, FUN = function(x){
  res.lm <- lm(x ~ ad.trt + ad.batch)
  summary.res = summary(res.lm)
  p = summary.res$coefficients[2,4]
})

ad.trt_q = p.adjust(ad.trt_p,method = 'fdr')
```

As the batch x treatment design of AD data is unbalaced, we fit a linear mixed model considering batch (cage) as random effects.

```
# HD data
hd.trt_p <- apply(hd.tss.clr, 2, FUN = function(x){
  res.lmm <- lmer(x ~ hd.trt + (1|hd.batch))
  summary.res = summary(res.lmm)
  p = summary.res$coefficients[2,5]
})

hd.trt_q = p.adjust(hd.trt_p,method = 'fdr')
```

3.1.2 SVA

SVA can account for unknown batch effects. But it is univariate, relies on a Gaussian likelihood assumption and implicitly introduces a correlation between treatment and batch.

The *sva* function performs two different steps. First it identifies the number of latent factors that need to be estimated. The number of factors can be estimated using the *num.sv*.

```
# sponge data
sponge.mod = model.matrix(~sponge.trt) # full model
sponge.mod0 = model.matrix(~1,data = sponge.trt) # null model
sponge.sva.n <- num.sv(dat = t(sponge.tss.clr), mod = sponge.mod)
```

Next we apply the *sva* function to estimate the surrogate variables:

```
sponge.sva = sva(t(sponge.tss.clr), sponge.mod, sponge.mod0, n.sv = sponge.sva.n)
```

```
## Number of significant surrogate variables is:  1
## Iteration (out of 5 ):1  2  3  4  5
```

We include the estimated surrogate variables in both the null and full models. The reason is that we want to adjust for the surrogate variables, so we treat them as adjustment variables that must be included in both models. The *f.pvalue* function is then used to calculate parametric F-test P-values and Q-values (adjusted P-values) for each OTU of sponge data.

```
sponge.mod.bat = cbind(sponge.mod,sponge.sva$sv)
sponge.mod0.bat = cbind(sponge.mod0,sponge.sva$sv)

sponge.sva.trt_p = f.pvalue(t(sponge.tss.clr),sponge.mod.bat,sponge.mod0.bat)
sponge.sva.trt_q = p.adjust(sponge.sva.trt_p,method="fdr")
```

Now these P-values and Q-values are accounting for surrogate variables (estimated batch effects).

We also apply SVA on both AD and HD data.

```
# ad data
ad.mod = model.matrix(~ad.trt)
ad.mod0 = model.matrix(~1,data = ad.trt)
ad.sva.n <- num.sv(dat = t(ad.tss.clr), mod = ad.mod)
ad.sva = sva(t(ad.tss.clr), ad.mod, ad.mod0, n.sv = ad.sva.n)
```

```
## Number of significant surrogate variables is:  6
## Iteration (out of 5 ):1  2  3  4  5
```

```
ad.mod.bat = cbind(ad.mod,ad.sva$sv)
ad.mod0.bat = cbind(ad.mod0,ad.sva$sv)
ad.sva.trt_p = f.pvalue(t(ad.tss.clr),ad.mod.bat,ad.mod0.bat)
ad.sva.trt_q = p.adjust(ad.sva.trt_p,method="fdr")

# hd data
hd.mod = model.matrix(~hd.trt)
hd.mod0 = model.matrix(~1,data = hd.trt)
hd.sva.n <- num.sv(dat = t(hd.tss.clr), mod = hd.mod)
hd.sva = sva(t(hd.tss.clr), hd.mod, hd.mod0, n.sv = hd.sva.n)
```

```
## Number of significant surrogate variables is:  3
## Iteration (out of 5 ):1  2  3  4  5
```

```
hd.mod.bat = cbind(hd.mod,hd.sva$sv)
hd.mod0.bat = cbind(hd.mod0,hd.sva$sv)
hd.sva.trt_p = f.pvalue(t(hd.tss.clr),hd.mod.bat,hd.mod0.bat)
hd.sva.trt_q = p.adjust(hd.sva.trt_p,method="fdr")
```

3.1.3 RUV2

RUV2 estimates and accounts for unknown batch effects. But it needs negative control variables that are affected by batch effects but not treatment effects.

In the real world, we design negative control variables that are not affected by treatment effects only, we are not sure but assume these controls are affected by batch effects. RUV2 can only account for the difference captured by these controls.

Since our three datasets do not have negative control variables, we use a linear model (or linear mixed model) to identify OTUs less likely to be affected by treatment effects as negative controls.

```
# sponge data
sponge.nc = sponge.trt_q > 0.05
sponge.ruv2 <- RUV2(Y = sponge.tss.clr, X = sponge.trt, ctl = sponge.nc, k = 3) # k is subjective
sponge.ruv2.trt_p <- sponge.ruv2$p
sponge.ruv2.trt_q <- p.adjust(sponge.ruv2.trt_p,method="fdr")

# AD data
ad.nc = ad.trt_q > 0.05
ad.ruv2 <- RUV2(Y = ad.tss.clr, X = ad.trt, ctl = ad.nc, k = 3) # k is subjective
ad.ruv2.trt_p <- ad.ruv2$p
ad.ruv2.trt_q <- p.adjust(ad.ruv2.trt_p,method="fdr")

# HD data
hd.nc = hd.trt_p > 0.05
hd.ruv2 <- RUV2(Y = hd.tss.clr, X = hd.trt, ctl = hd.nc, k = 3) # k is subjective
hd.ruv2.trt_p <- hd.ruv2$p
hd.ruv2.trt_q <- p.adjust(hd.ruv2.trt_p,method="fdr")
```

3.1.4 RUV4

```
# sponge data
sponge.k.obj = getK(Y = sponge.tss.clr, X = sponge.trt, ctl = sponge.nc)
sponge.k = sponge.k.obj$k
sponge.k = ifelse(sponge.k !=0, sponge.k, 1)
sponge.ruv4 <- RUV4(Y = sponge.tss.clr, X = sponge.trt, ctl = sponge.nc, k = sponge.k)
sponge.ruv4.trt_p <- sponge.ruv4$p
sponge.ruv4.trt_q <- p.adjust(sponge.ruv4.trt_p,method="fdr")

# AD data
ad.k.obj = getK(Y = ad.tss.clr, X = ad.trt, ctl = ad.nc)
ad.k =ad.k.obj$k
ad.k = ifelse(ad.k !=0, ad.k, 1)
ad.ruv4 <- RUV4(Y = ad.tss.clr, X = ad.trt, ctl = ad.nc, k = ad.k)
ad.ruv4.trt_p <- ad.ruv4$p
ad.ruv4.trt_q <- p.adjust(ad.ruv4.trt_p,method="fdr")

# HD data
hd.k.obj = getK(Y = hd.tss.clr, X = hd.trt, ctl = hd.nc)
hd.k = hd.k.obj$k
hd.k = ifelse(hd.k !=0, hd.k, 1)
hd.ruv4 <- RUV4(Y = hd.tss.clr, X = hd.trt, ctl = hd.nc, k = hd.k)
hd.ruv4.trt_p <- hd.ruv4$p
hd.ruv4.trt_q <- p.adjust(hd.ruv4.trt_p,method="fdr")
```

3.2 Correcting for batch effects

3.2.1 BMC (batch mean centering)

```
# Sponge data
sponge.b1 = scale(sponge.tss.clr[sponge.batch== 1,],center = TRUE, scale = FALSE)
sponge.b2 = scale(sponge.tss.clr[sponge.batch== 2,],center = TRUE, scale = FALSE)
sponge.bmc = rbind(sponge.b1,sponge.b2)
sponge.bmc = sponge.bmc[rownames(sponge.tss.clr),]

##############
# AD data
ad.b1 = scale(ad.tss.clr[ad.batch=="09/04/2015",],center = TRUE, scale = FALSE)
ad.b2 = scale(ad.tss.clr[ad.batch=="14/04/2016",],center = TRUE, scale = FALSE)
ad.b3 = scale(ad.tss.clr[ad.batch=="14/11/2016",],center = TRUE, scale = FALSE)
ad.b4 = scale(ad.tss.clr[ad.batch=="01/07/2016",],center = TRUE, scale = FALSE)
ad.b5 = scale(ad.tss.clr[ad.batch=="21/09/2017",],center = TRUE, scale = FALSE)
ad.bmc = rbind(ad.b1,ad.b2,ad.b3,ad.b4,ad.b5)
ad.bmc = ad.bmc[rownames(ad.tss.clr),]
```

3.2.2 ComBat

```
# Sponge data
sponge.combat <- t(ComBat(t(sponge.tss.clr),batch=sponge.batch,mod = sponge.mod,par.prior=F,prior.plots
```

```
## Found2batches

## Adjusting for1covariate(s) or covariate level(s)

## Standardizing Data across genes

## Fitting L/S model and finding priors

## Finding nonparametric adjustments

## Adjusting the Data
```

```
##############
# AD data
ad.combat <- t(ComBat(t(ad.tss.clr),batch=ad.batch,mod = ad.mod, par.prior=F,prior.plots = F))
```

```
## Found5batches

## Adjusting for1covariate(s) or covariate level(s)

## Standardizing Data across genes

## Fitting L/S model and finding priors

## Finding nonparametric adjustments

## Adjusting the Data
```

3.2.3 removeBatchEffect

```
# Sponge data
sponge.limma <- t(removeBatchEffect(t(sponge.tss.clr),batch = sponge.batch,design = sponge.mod))

#############
ad.limma <- t(removeBatchEffect(t(ad.tss.clr),batch = ad.batch,design = ad.mod))
```

3.2.4 FAbatch

FAbatch is unable to converge on both sponge data and AD data. This may influence the effect of batch correction.

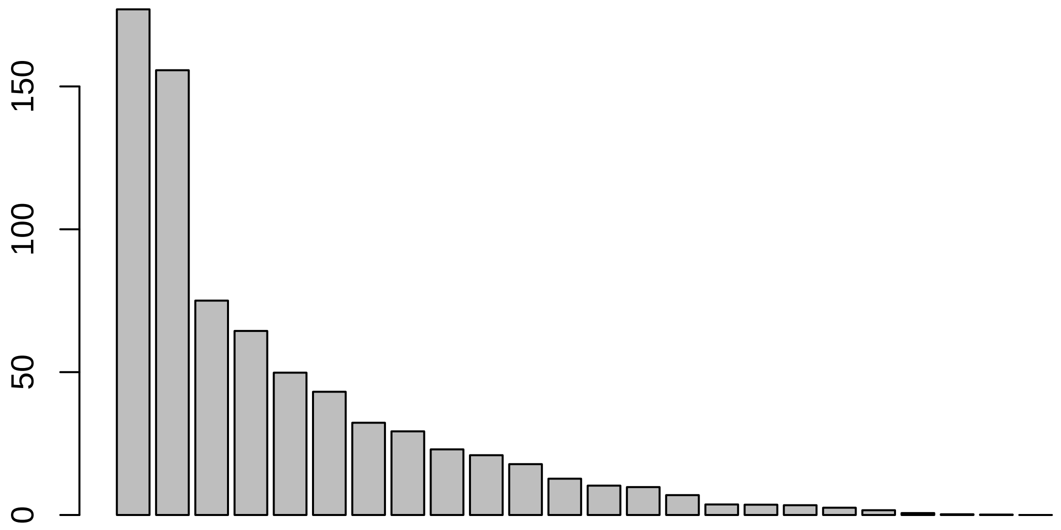
```
# sponge data
sponge.fabatch.obj = fabatch(x = sponge.tss.clr,y = as.factor(as.numeric(sponge.trt)), batch = sponge.ba
sponge.fabatch <- sponge.fabatch.obj$xadj

# ad data
ad.fabatch.obj = fabatch(x = ad.tss.clr,y = as.factor(as.numeric(ad.trt)), batch = as.factor(as.numeric
ad.fabatch <- ad.fabatch.obj$xadj
```

3.2.5 percentile normalisation

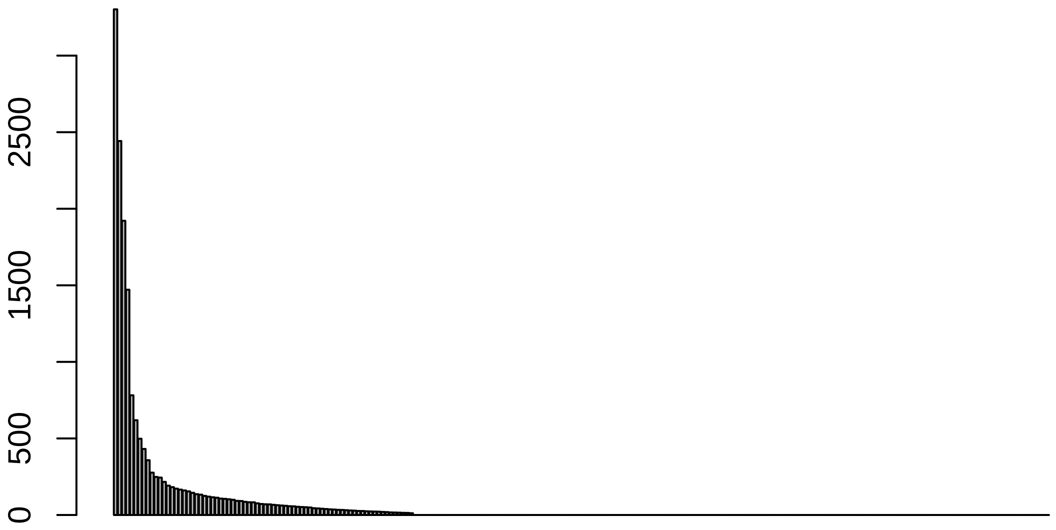
```
sponge.percentile = percentile_norm(data = sponge.tss, batch = sponge.batch, trt = sponge.trt)

# ad data
ad.percentile = percentile_norm(data = ad.tss, batch = ad.batch, trt = ad.trt)
```

3.2.6 SVD-based method

```
#################
# sponge data
sponge.sd = apply(sponge.tss.clr,2,sd)
sponge.mean = apply(sponge.tss.clr,2,mean)
X.sponge = scale(sponge.tss.clr,center = T,scale = T)

m.sponge = crossprod(X.sponge)
m.svd.sponge = svd(m.sponge)
barplot(m.svd.sponge$d)
```



```
a1.sponge = m.svd.sponge$u[,1]
b1.sponge = m.svd.sponge$v[,1]

# component 1
t1.sponge = X.sponge %*% a1.sponge / drop(sqrt(crossprod(a1.sponge)))
c1.sponge = crossprod(X.sponge, t1.sponge) / drop(crossprod(t1.sponge))
defl.matrix_svd1.sponge  = X.sponge - t1.sponge %*% t(c1.sponge)

# #
sponge.svd = defl.matrix_svd1.sponge
sponge.svd[1:nrow(sponge.svd),1:ncol(sponge.svd)] = NA
for(i in 1:ncol(defl.matrix_svd1.sponge)){
  for(j in 1:nrow(defl.matrix_svd1.sponge)){
    sponge.svd[j,i] = defl.matrix_svd1.sponge[j,i]*sponge.sd[i] + sponge.mean[i]
  }
}

#####################
# ad data
ad.sd = apply(ad.tss.clr,2,sd)
ad.mean = apply(ad.tss.clr,2,mean)
X.ad = scale(ad.tss.clr,center = T,scale = T)

m.ad = crossprod(X.ad)
m.svd.ad = svd(m.ad)
barplot(m.svd.ad$d)
```



```
a1.ad = m.svd.ad$u[,1]
b1.ad = m.svd.ad$v[,1]

# component 1
t1.ad = X.ad %*% a1.ad / drop(sqrt(crossprod(a1.ad)))
c1.ad = crossprod(X.ad, t1.ad) / drop(crossprod(t1.ad))
defl.matrix_svd1.ad  = X.ad - t1.ad %*% t(c1.ad)

# #
ad.svd = defl.matrix_svd1.ad
ad.svd[1:nrow(ad.svd),1:ncol(ad.svd)] = NA
for(i in 1:ncol(defl.matrix_svd1.ad)){
  for(j in 1:nrow(defl.matrix_svd1.ad)){
    ad.svd[j,i] = defl.matrix_svd1.ad[j,i]*ad.sd[i] + ad.mean[i]
  }
}
```

3.2.7 RUVIII

RUVIII needs technical sample replicates and negative control variables. As only AD data have sample replicates, RUVIII is only applied on AD data. We use linear model to find variables with less probability of treatment effects, and these variables are treated as negative control variables to fit the assumptions of RUVIII.

```
#####################
# ad data only
replicates.ad = ad.metadata$sample_name.data.extraction
replicates.ad.matrix = replicate.matrix(replicates.ad)

p.ad = matrix(NA,nrow = 2, ncol = ncol(ad.tss.clr))
rownames(p.ad) = c('ad.trt','ad.batch')
colnames(p.ad) = colnames(ad.tss.clr)
for(i in 1:ncol(ad.tss.clr)){
  res = lm(ad.tss.clr[,i] ~ ad.trt + ad.batch)
  summ.res = summary(res)
  anova.res = anova(res)
  p.ad[1,i] = anova.res$`Pr(>F)`[1]
  p.ad[2,i] = anova.res$`Pr(>F)`[2]
```

```r
}

p.adj.ad = apply(p.ad,1,p.adjust,method = 'fdr')
p.adj.ad1 = sort(p.adj.ad[,1],decreasing = T)
nc.otu1 = names(p.adj.ad1[1:75]) #negative control genes need be equal or more than samples
nc1 = rep(FALSE, ncol(ad.tss.clr))
names(nc1) = colnames(ad.tss.clr)
nc1[nc.otu1] = TRUE

ad.ruv <- RUVIII(Y=ad.tss.clr,M = replicates.ad.matrix, ctl = nc1)
rownames(ad.ruv) = rownames(ad.tss.clr)
```

Chapter 4

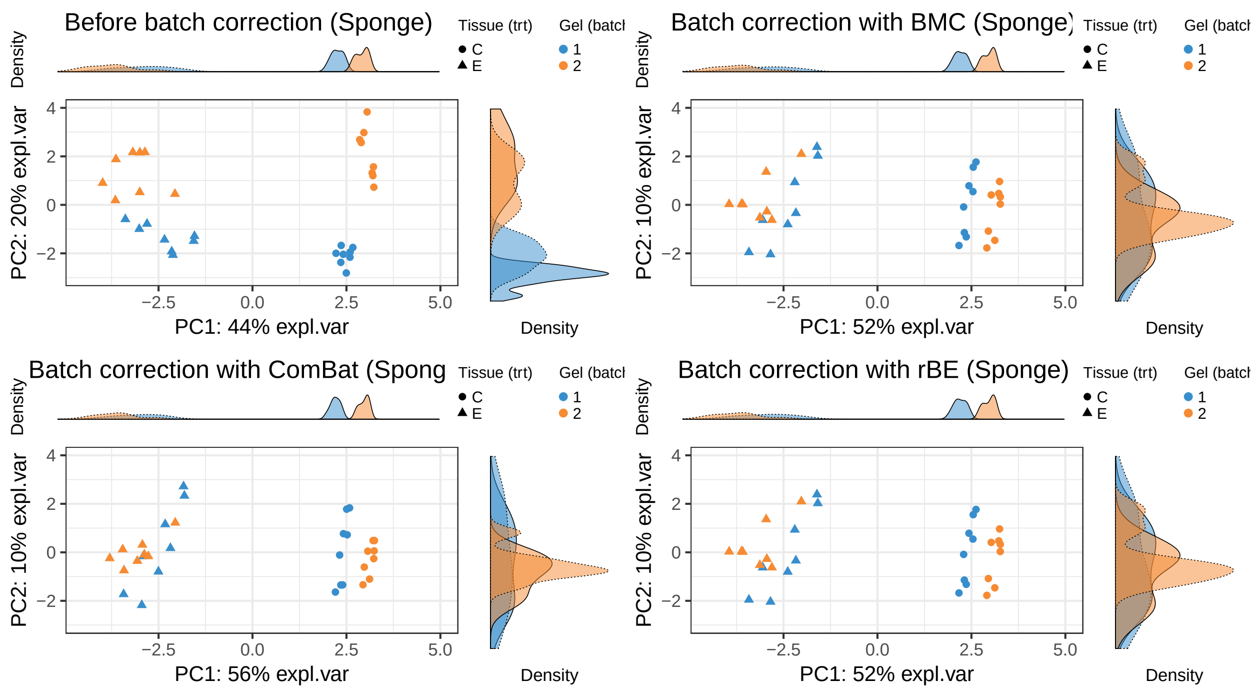
Methods evaluation

4.1 Diagnostic plots

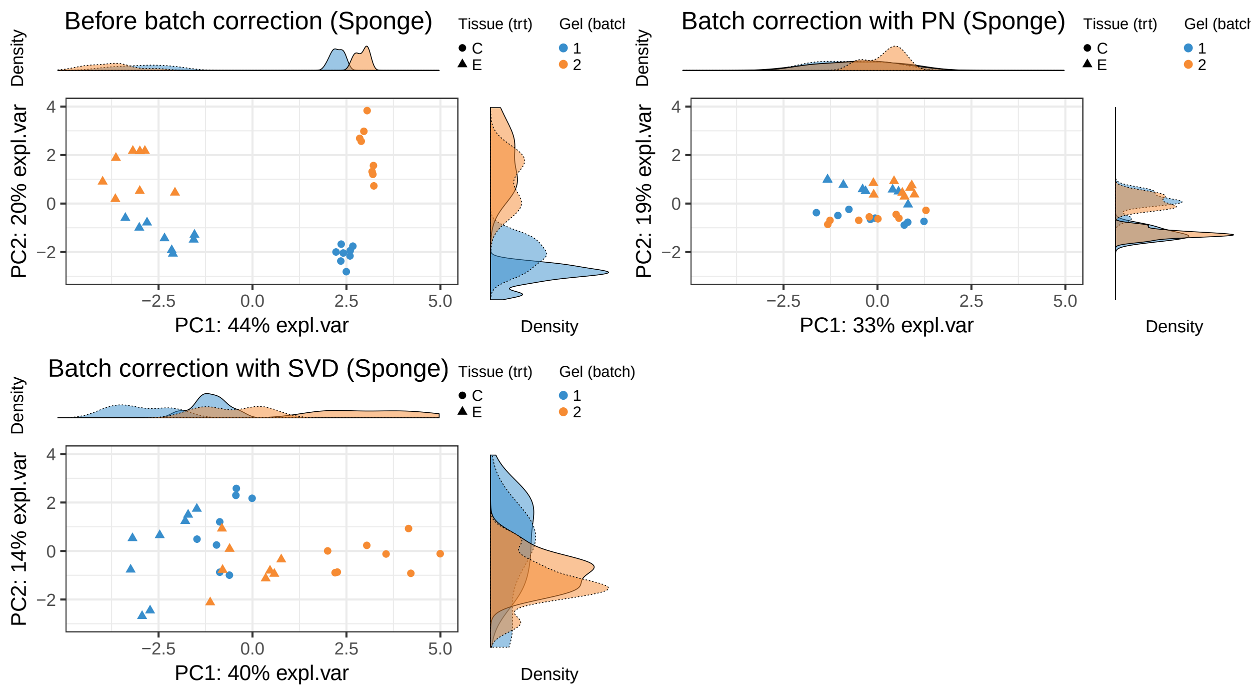
4.1.1 Principal component analysis (PCA) with density plot per component

```
# sponge data
pca.sponge.before = pca(sponge.tss.clr, ncomp = 3)
pca.sponge.bmc = pca(sponge.bmc, ncomp = 3)
pca.sponge.combat = pca(sponge.combat, ncomp = 3)
pca.sponge.limma = pca(sponge.limma, ncomp = 3)
pca.sponge.percentile = pca(sponge.percentile, ncomp = 3)
pca.sponge.svd = pca(sponge.svd, ncomp = 3)

# ad data
pca.ad.before = pca(ad.tss.clr, ncomp = 3)
pca.ad.bmc = pca(ad.bmc, ncomp = 3)
pca.ad.combat = pca(ad.combat, ncomp = 3)
pca.ad.limma = pca(ad.limma, ncomp = 3)
pca.ad.percentile = pca(ad.percentile, ncomp = 3)
pca.ad.svd = pca(ad.svd, ncomp = 3)
pca.ad.ruv = pca(ad.ruv, ncomp = 3)
```

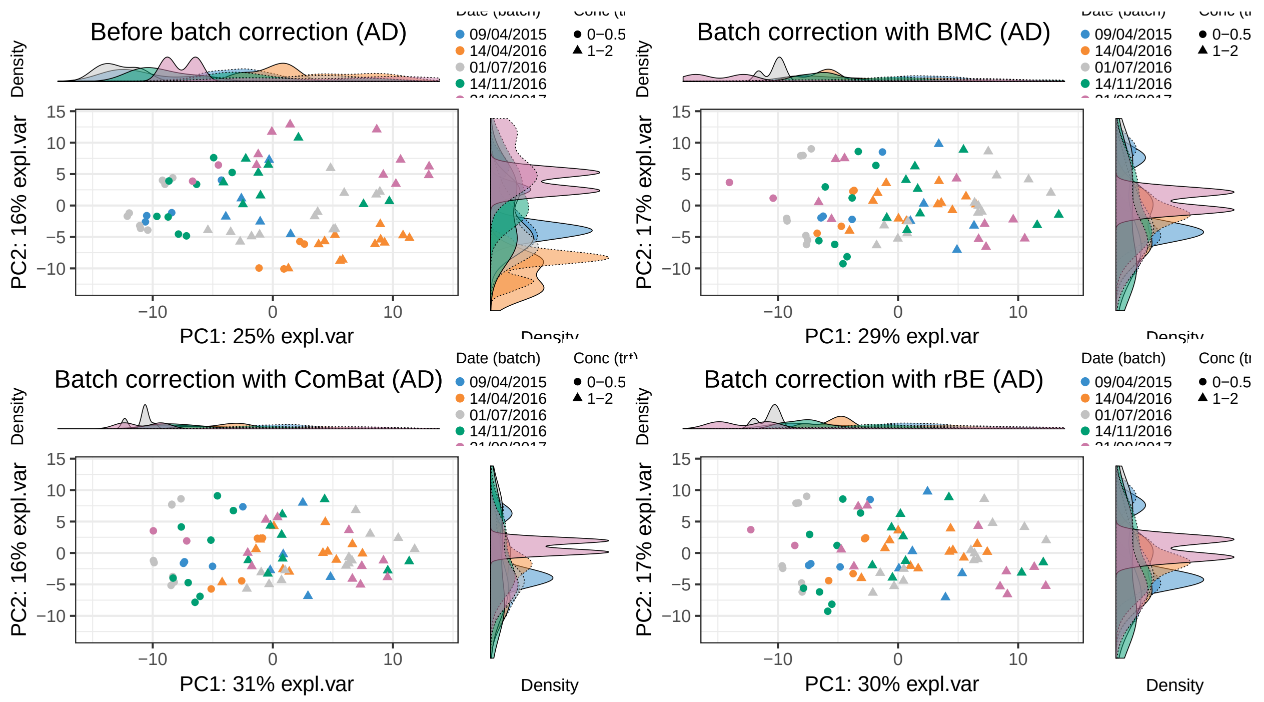
```
grid.arrange(plot.pca.before.sponge, plot.pca.bmc.sponge, plot.pca.combat.sponge,plot.pca.limma.sponge,
```



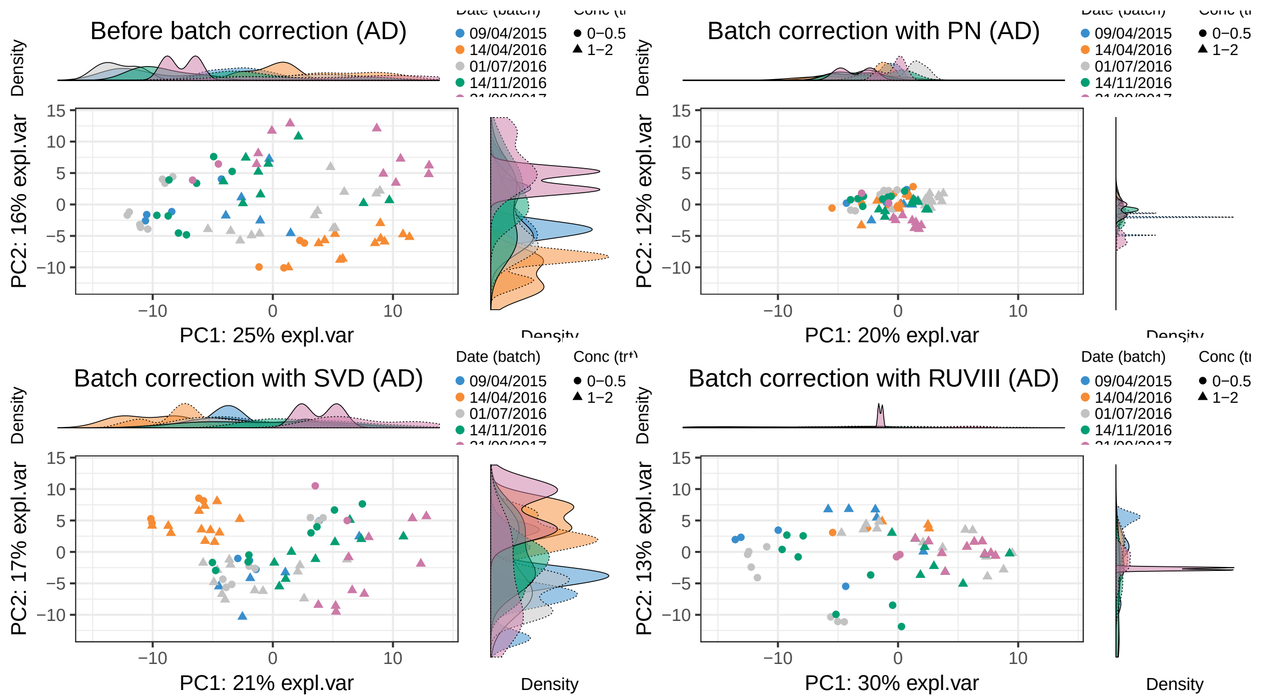
```
grid.arrange(plot.pca.before.sponge, plot.pca.percentile.sponge,plot.pca.svd.sponge,ncol=2)
```



```
grid.arrange(plot.pca.before.ad, plot.pca.bmc.ad, plot.pca.combat.ad,plot.pca.limma.ad,ncol=2)
```



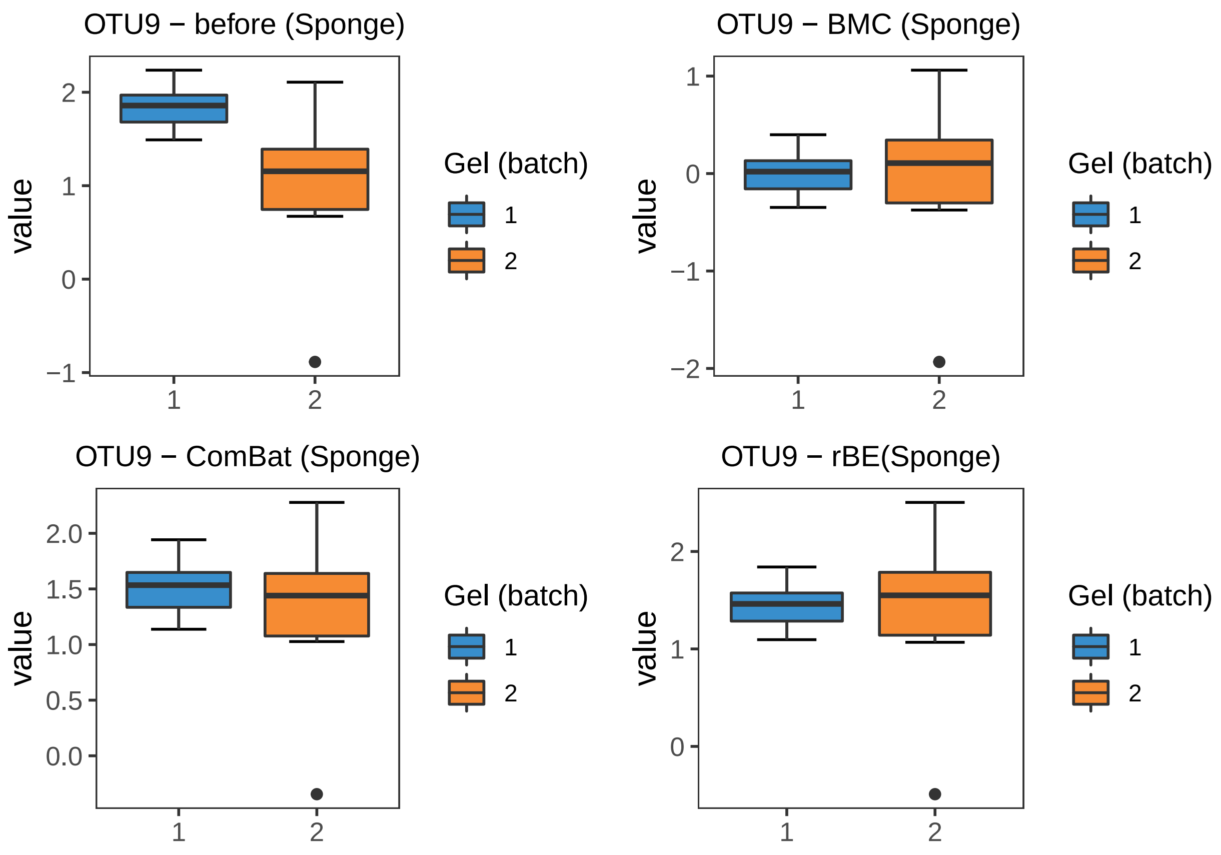
```
grid.arrange(plot.pca.before.ad, plot.pca.percentile.ad,plot.pca.svd.ad,plot.pca.ruv.ad,ncol=2)
```



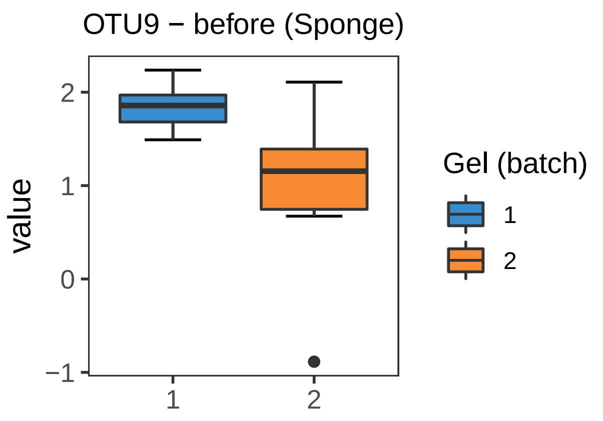
4.1.2 Density plot and box plot

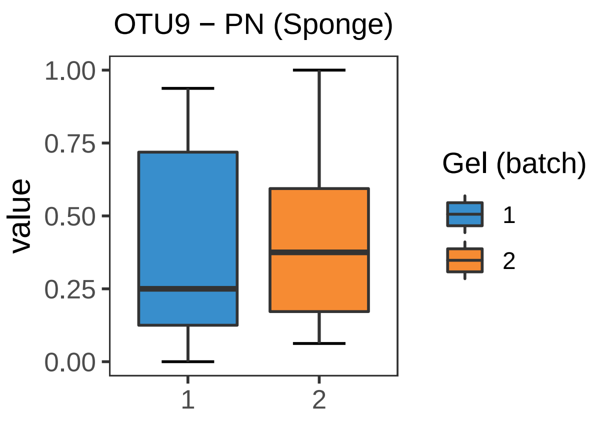
```
###############
## sponge data
sponge.before.df = data.frame(value = sponge.tss.clr[,9], batch = sponge.batch)
sponge.before.boxplot <- box_plot_fun(data = sponge.before.df,x=sponge.before.df$batch,
             y=sponge.before.df$value,title = 'OTU9 - before (Sponge)',
             batch.legend.title = 'Gel (batch)')

sponge.bmc.df = data.frame(value = sponge.bmc[,9], batch = sponge.batch)
sponge.bmc.boxplot <-box_plot_fun(data = sponge.bmc.df,x=sponge.bmc.df$batch,
             y=sponge.bmc.df$value,title = 'OTU9 - BMC (Sponge)',
             batch.legend.title = 'Gel (batch)')


sponge.combat.df = data.frame(value = sponge.combat[,9], batch = sponge.batch)
sponge.combat.boxplot <-box_plot_fun(data = sponge.combat.df,x=sponge.combat.df$batch,
             y=sponge.combat.df$value,title = 'OTU9 - ComBat (Sponge)',
             batch.legend.title = 'Gel (batch)')


sponge.limma.df = data.frame(value = sponge.limma[,9], batch = sponge.batch)
sponge.limma.boxplot <-box_plot_fun(data = sponge.limma.df,x=sponge.limma.df$batch,
             y=sponge.limma.df$value,title = 'OTU9 - rBE(Sponge)',
             batch.legend.title = 'Gel (batch)')

sponge.percentile.df = data.frame(value = sponge.percentile[,9], batch = sponge.batch)
sponge.percentile.boxplot <-box_plot_fun(data = sponge.percentile.df,x=sponge.percentile.df$batch,
             y=sponge.percentile.df$value,title = 'OTU9 - PN (Sponge)',
             batch.legend.title = 'Gel (batch)')

sponge.svd.df = data.frame(value = sponge.svd[,9], batch = sponge.batch)
sponge.svd.boxplot <-box_plot_fun(data = sponge.svd.df,x=sponge.svd.df$batch,
             y=sponge.svd.df$value,title = 'OTU9 - SVD (Sponge)',
             batch.legend.title = 'Gel (batch)')
```

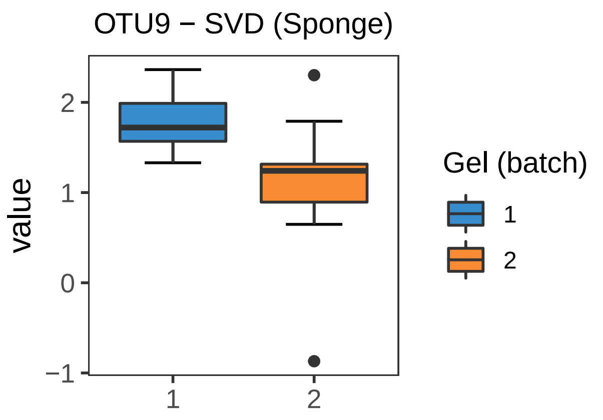
```
grid.arrange(sponge.before.boxplot, sponge.bmc.boxplot, sponge.combat.boxplot, sponge.limma.boxplot,ncol
```



```
grid.arrange(sponge.before.boxplot, sponge.percentile.boxplot,sponge.svd.boxplot,ncol=2)
```

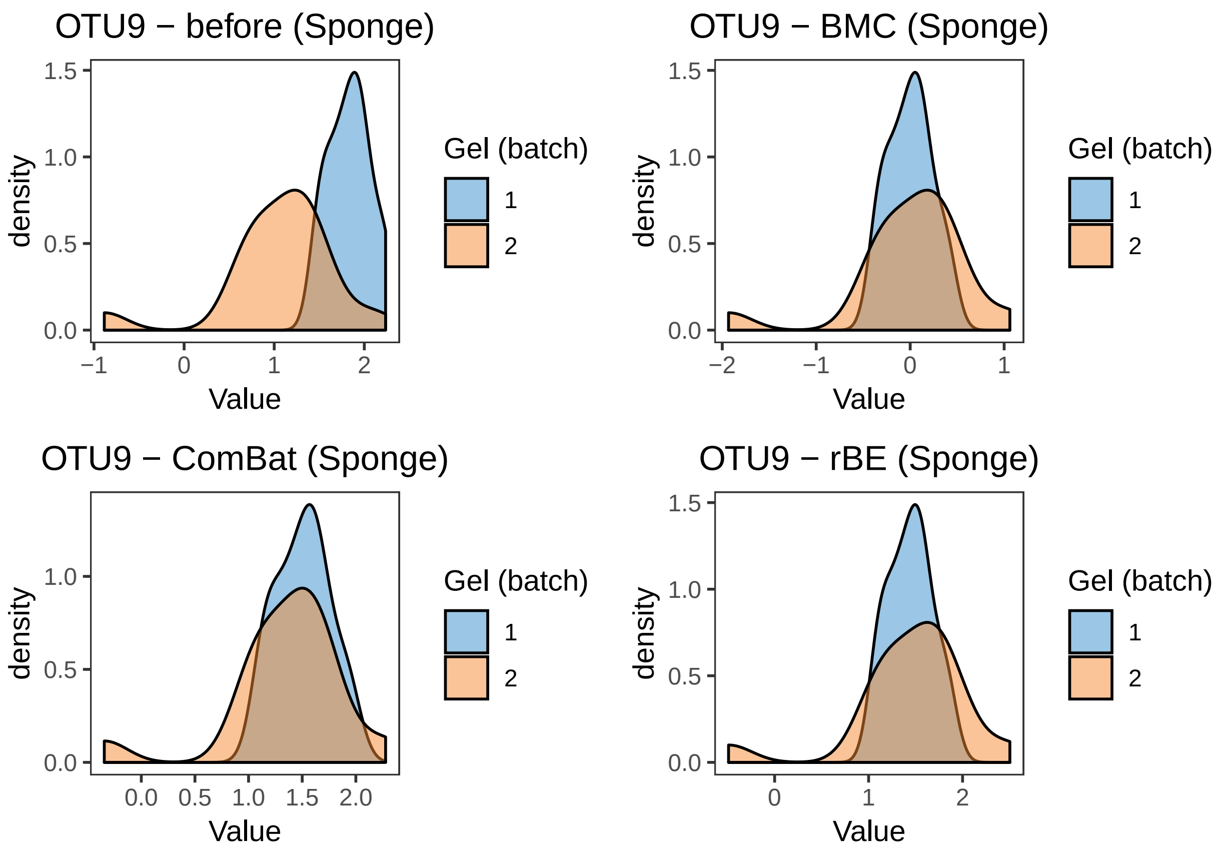




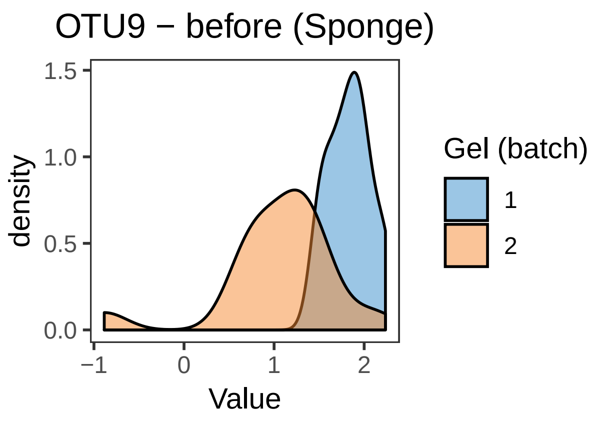


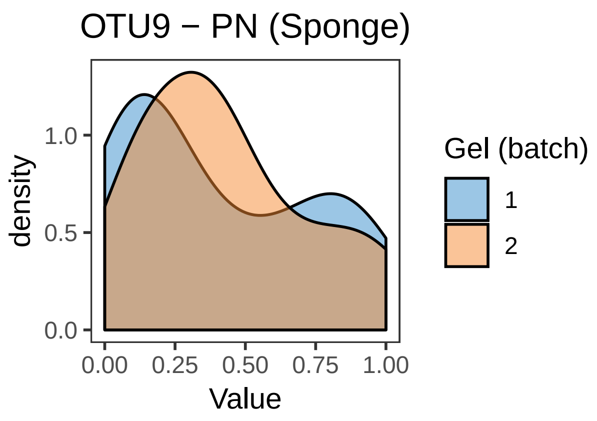
```
## density plot
# before
plot.sponge.before <- ggplot(sponge.before.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5

# BMC
plot.sponge.bmc <- ggplot(sponge.bmc.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + sca


# ComBat
plot.sponge.combat <- ggplot(sponge.combat.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5


# removeBatchEffect
plot.sponge.limma <- ggplot(sponge.limma.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5)


# percentile normal
plot.sponge.percentile <- ggplot(sponge.percentile.df, aes(x = value, fill = batch)) + geom_density(alp


# SVD
plot.sponge.svd <- ggplot(sponge.svd.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + sca
```

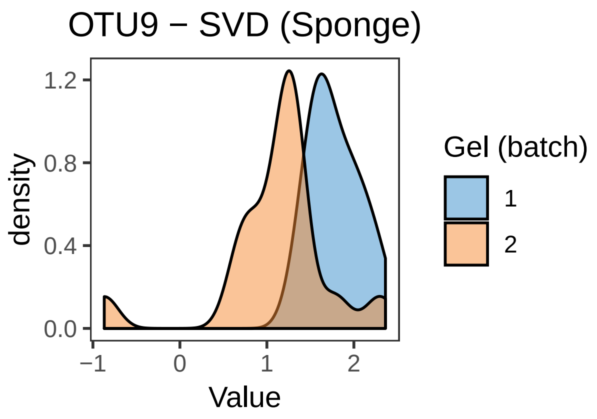
```
grid.arrange(plot.sponge.before, plot.sponge.bmc, plot.sponge.combat,plot.sponge.limma,ncol=2)
```



```r
grid.arrange(plot.sponge.before, plot.sponge.percentile, plot.sponge.svd,ncol=2)
```







```
######### p-values
linearm.sponge.before = lm(sponge.tss.clr[,9]~ sponge.trt + sponge.batch)
summary(linearm.sponge.before)
```

```
##
## Call:
## lm(formula = sponge.tss.clr[, 9] ~ sponge.trt + sponge.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.87967 -0.24705  0.04588  0.24492  1.00757
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)     1.7849     0.1497  11.922 1.06e-12 ***
## sponge.trtE     0.1065     0.1729   0.616    0.543
## sponge.batch2  -0.7910     0.1729  -4.575 8.24e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.489 on 29 degrees of freedom
## Multiple R-squared:  0.4236, Adjusted R-squared:  0.3839
## F-statistic: 10.66 on 2 and 29 DF,  p-value: 0.0003391
```

```
linearm.sponge.bmc = lm(sponge.bmc[,9]~ sponge.trt + sponge.batch)
summary(linearm.sponge.bmc)
```

```
##
```

```
## Call:
## lm(formula = sponge.bmc[, 9] ~ sponge.trt + sponge.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.87967 -0.24705  0.04588  0.24492  1.00757
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -5.323e-02  1.497e-01  -0.356    0.725
## sponge.trtE    1.065e-01  1.729e-01   0.616    0.543
## sponge.batch2  3.925e-17  1.729e-01   0.000    1.000
##
## Residual standard error: 0.489 on 29 degrees of freedom
## Multiple R-squared:  0.01291,    Adjusted R-squared:  -0.05517
## F-statistic: 0.1896 on 2 and 29 DF,  p-value: 0.8283
```

```
linearm.sponge.combat = lm(sponge.combat[,9]~ sponge.trt + sponge.batch)
summary(linearm.sponge.combat)
```

```
##
## Call:
## lm(formula = sponge.combat[, 9] ~ sponge.trt + sponge.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.64583 -0.22414  0.05092  0.24065  0.88585
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    1.46291    0.13491  10.844 1.02e-11 ***
## sponge.trtE    0.09081    0.15578   0.583    0.564
## sponge.batch2 -0.16201    0.15578  -1.040    0.307
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4406 on 29 degrees of freedom
## Multiple R-squared:  0.04672,    Adjusted R-squared:  -0.01902
## F-statistic: 0.7107 on 2 and 29 DF,  p-value: 0.4996
```

```
linearm.sponge.limma = lm(sponge.limma[,9]~ sponge.trt + sponge.batch)
summary(linearm.sponge.limma)
```

```
##
## Call:
## lm(formula = sponge.limma[, 9] ~ sponge.trt + sponge.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.87967 -0.24705  0.04588  0.24492  1.00757
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    1.389e+00  1.497e-01   9.280 3.49e-10 ***
## sponge.trtE    1.065e-01  1.729e-01   0.616    0.543
## sponge.batch2 2.355e-16  1.729e-01   0.000    1.000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.489 on 29 degrees of freedom
## Multiple R-squared:  0.01291,    Adjusted R-squared:  -0.05517
## F-statistic: 0.1896 on 2 and 29 DF,  p-value: 0.8283
```

```
linearm.sponge.percentile = lm(sponge.percentile[,9]~ sponge.trt + sponge.batch)
summary(linearm.sponge.percentile)
```

```
##
## Call:
## lm(formula = sponge.percentile[, 9] ~ sponge.trt + sponge.batch)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.4531 -0.2070 -0.0625  0.1797  0.6562
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.48438    0.09515   5.090 1.97e-05 ***
## sponge.trtE   -0.17187    0.10987  -1.564    0.129
## sponge.batch2  0.03125    0.10987   0.284    0.778
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3108 on 29 degrees of freedom
## Multiple R-squared:  0.08018,    Adjusted R-squared:  0.01674
## F-statistic: 1.264 on 2 and 29 DF,  p-value: 0.2976
```

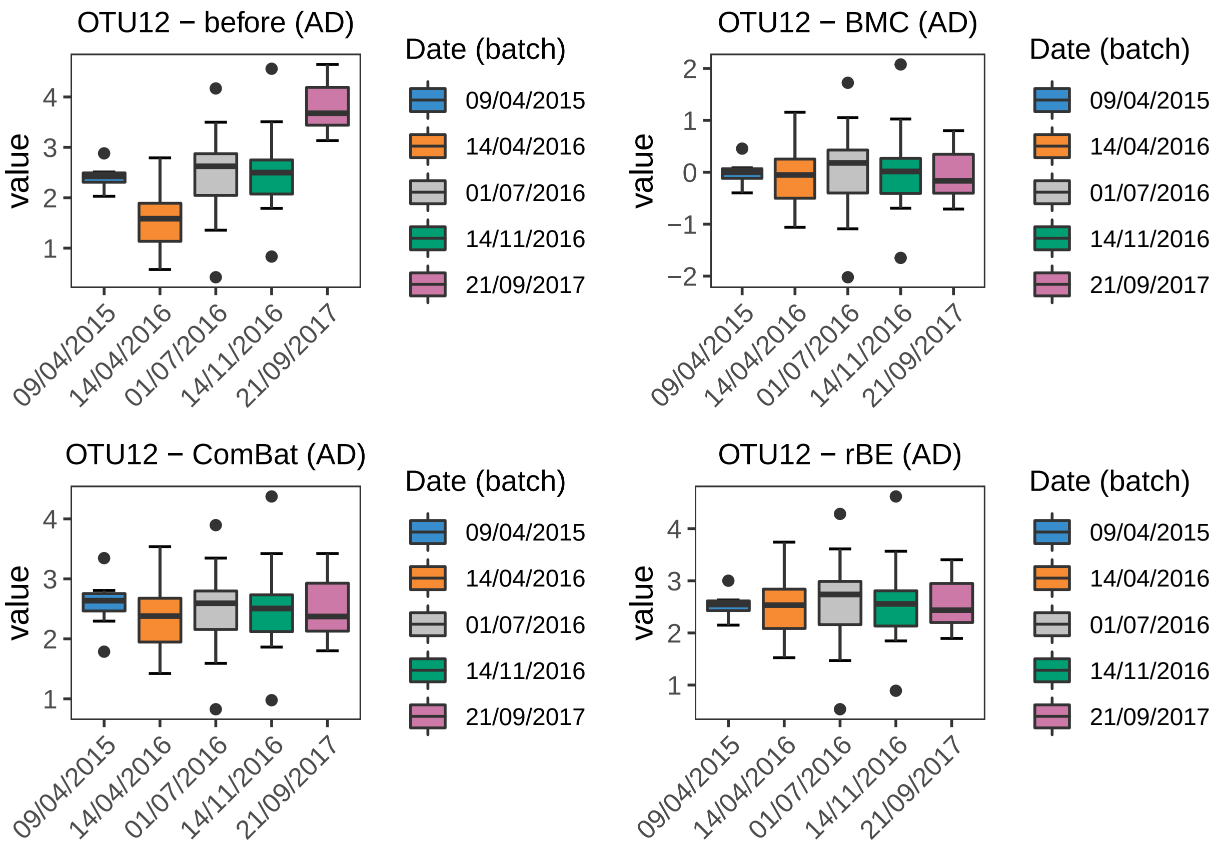
```
linearm.sponge.svd = lm(sponge.svd[,9]~ sponge.trt + sponge.batch)
summary(linearm.sponge.svd)
```

```
##
## Call:
## lm(formula = sponge.svd[, 9] ~ sponge.trt + sponge.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.82982 -0.27539  0.05228  0.28204  1.05932
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)     1.6433     0.1526  10.766 1.21e-11 ***
## sponge.trtE     0.2817     0.1763   1.598  0.12085
## sponge.batch2  -0.6831     0.1763  -3.875  0.00056 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4985 on 29 degrees of freedom
## Multiple R-squared:  0.3773, Adjusted R-squared:  0.3344
## F-statistic: 8.787 on 2 and 29 DF,  p-value: 0.001039
```

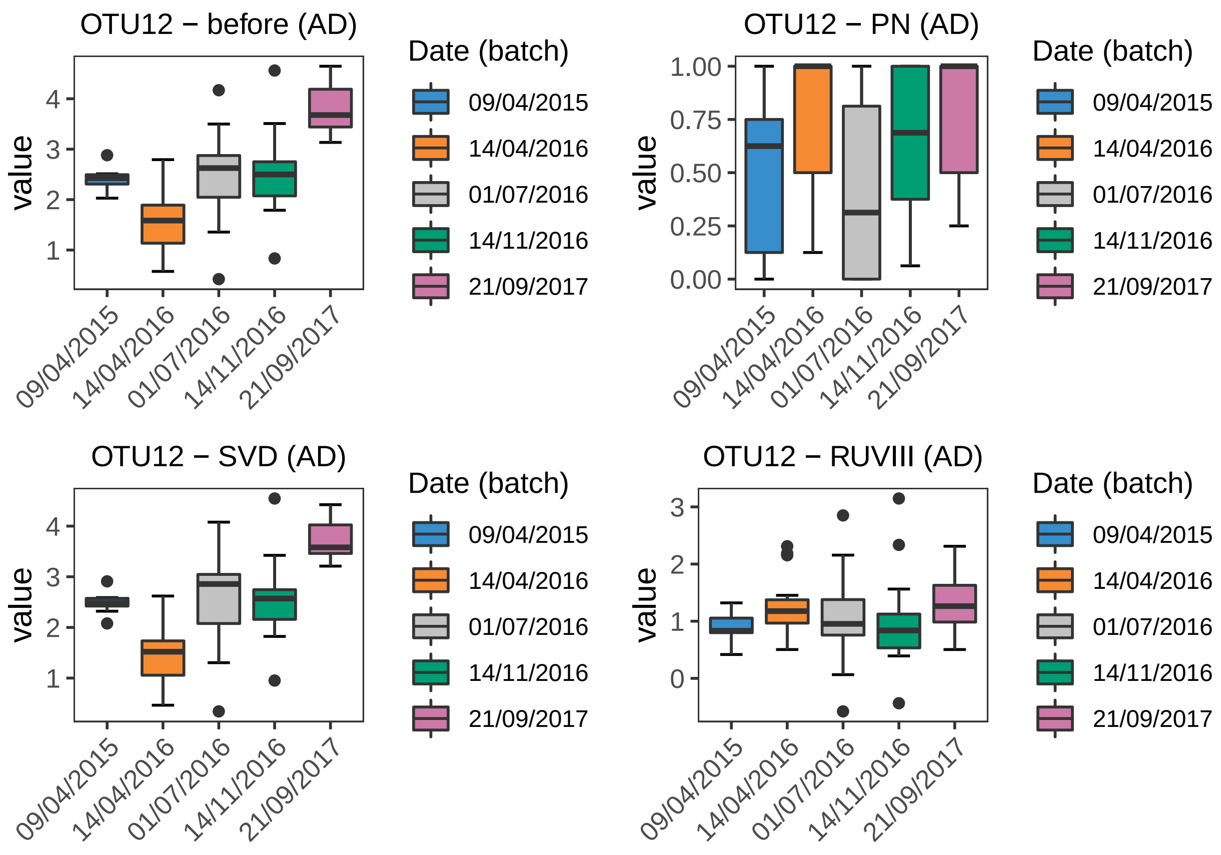
```
##################
# ad data
```

```r
# boxplot
ad.before.df = data.frame(value = ad.tss.clr[,1], batch = ad.batch)

ad.before.boxplot <- box_plot_fun(data = ad.before.df,x=ad.before.df$batch,
             y=ad.before.df$value,title = 'OTU12 - before (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)


ad.bmc.df = data.frame(value = ad.bmc[,1], batch = ad.batch)

ad.bmc.boxplot <- box_plot_fun(data = ad.bmc.df,x=ad.bmc.df$batch,
             y=ad.bmc.df$value,title = 'OTU12 - BMC (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)


ad.combat.df = data.frame(value = ad.combat[,1], batch = ad.batch)

ad.combat.boxplot <- box_plot_fun(data = ad.combat.df,x=ad.combat.df$batch,
             y=ad.combat.df$value,title = 'OTU12 - ComBat (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)

ad.limma.df = data.frame(value = ad.limma[,1], batch = ad.batch)

ad.limma.boxplot <- box_plot_fun(data = ad.limma.df,x=ad.limma.df$batch,
             y=ad.limma.df$value,title = 'OTU12 - rBE (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)


ad.percentile.df = data.frame(value = ad.percentile[,1], batch = ad.batch)

ad.percentile.boxplot <- box_plot_fun(data = ad.percentile.df,x=ad.percentile.df$batch,
             y=ad.percentile.df$value,title = 'OTU12 - PN (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)

ad.svd.df = data.frame(value = ad.svd[,1], batch = ad.batch)

ad.svd.boxplot <- box_plot_fun(data = ad.svd.df,x=ad.svd.df$batch,
             y=ad.svd.df$value,title = 'OTU12 - SVD (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)

ad.ruv.df = data.frame(value = ad.ruv[,1], batch = ad.batch)

ad.ruv.boxplot <- box_plot_fun(data = ad.ruv.df,x=ad.ruv.df$batch,
             y=ad.ruv.df$value,title = 'OTU12 - RUVIII (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)
```

```r
grid.arrange(ad.before.boxplot, ad.bmc.boxplot, ad.combat.boxplot, ad.limma.boxplot,ncol=2)
```

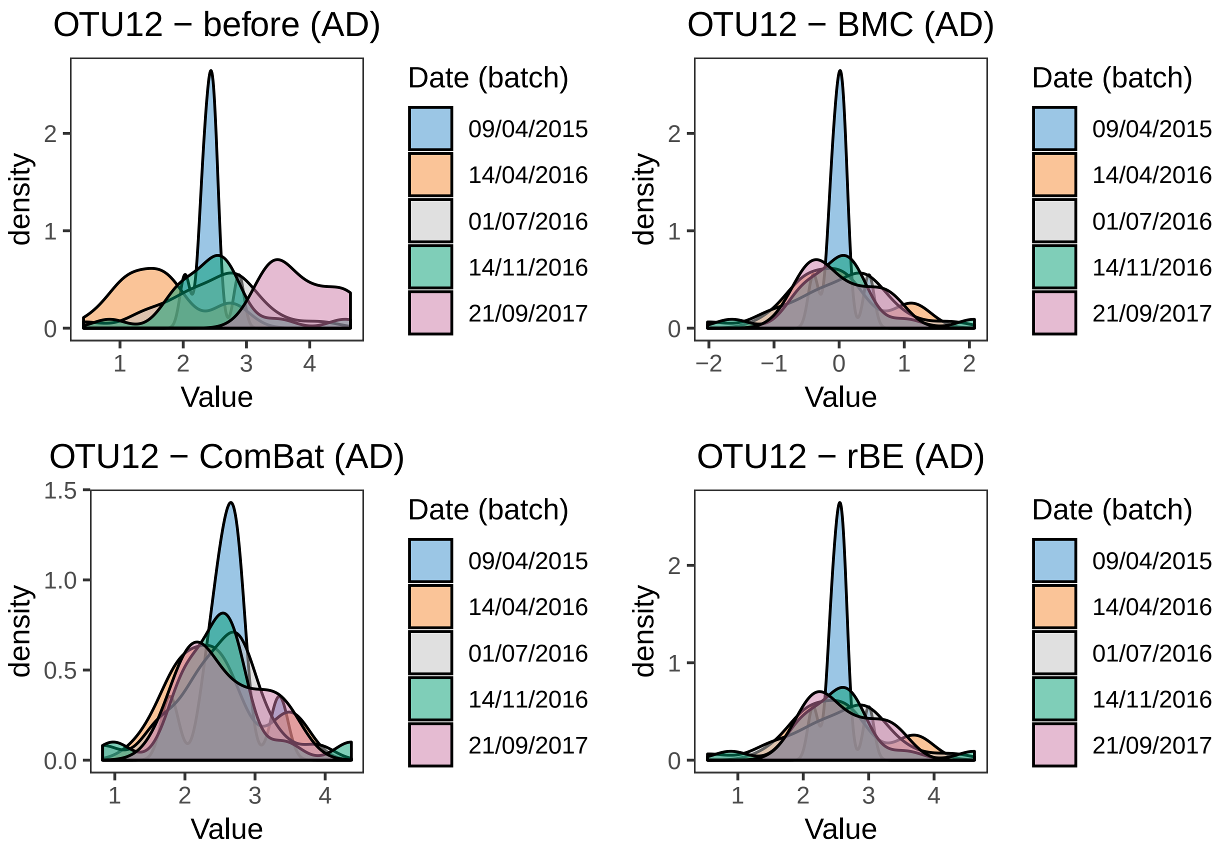


```r
grid.arrange(ad.before.boxplot, ad.percentile.boxplot,ad.svd.boxplot,ad.ruv.boxplot,ncol=2)
```

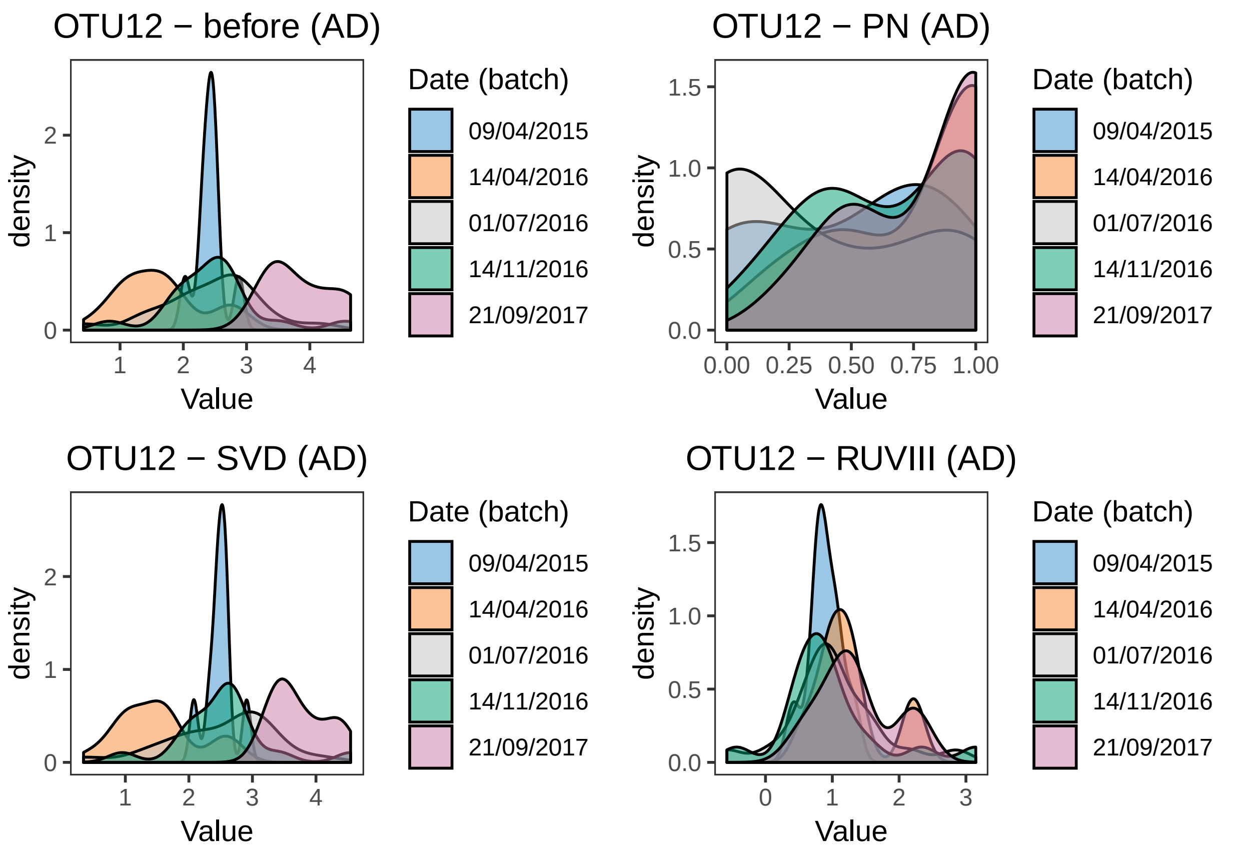


```
# density plot
# before
plot.ad.before = ggplot(ad.before.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_

# BMC
plot.ad.bmc = ggplot(ad.bmc.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_fill_


# ComBat
plot.ad.combat = ggplot(ad.combat.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_


# removeBatchEffect
plot.ad.limma = ggplot(ad.limma.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_f


# percentile norm
plot.ad.percentile = ggplot(ad.percentile.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5)


# SVD
plot.ad.svd = ggplot(ad.svd.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_fill_


# RUVIII
plot.ad.ruv = ggplot(ad.ruv.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_fill_
```

```
grid.arrange(plot.ad.before, plot.ad.bmc, plot.ad.combat,plot.ad.limma,ncol=2)
```



```
grid.arrange(plot.ad.before, plot.ad.percentile, plot.ad.svd,plot.ad.ruv, ncol=2)
```



```
#p-values
linearm.ad.before = lm(ad.tss.clr[,1]~ ad.trt + ad.batch)
anova(linearm.ad.before)
```

```
## Analysis of Variance Table
##
## Response: ad.tss.clr[, 1]
##           Df Sum Sq Mean Sq F value    Pr(>F)
## ad.trt     1  1.460  1.4605  3.1001   0.08272 .
## ad.batch   4 32.889  8.2222 17.4532 6.168e-10 ***
## Residuals 69 32.506  0.4711
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(linearm.ad.before)
```

```
##
## Call:
## lm(formula = ad.tss.clr[, 1] ~ ad.trt + ad.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.09885 -0.39613 -0.00381  0.36645  1.98185
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        2.311213   0.247768   9.328 7.57e-14 ***
## ad.trt1-2          0.203619   0.171183   1.189  0.23833
## ad.batch14/04/2016 -0.828100   0.287918  -2.876  0.00535 **
## ad.batch01/07/2016  0.007239   0.273672   0.026  0.97897
## ad.batch14/11/2016  0.062689   0.282978   0.222  0.82533
## ad.batch21/09/2017  1.361132   0.306373   4.443 3.30e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6864 on 69 degrees of freedom
## Multiple R-squared:  0.5138, Adjusted R-squared:  0.4786
## F-statistic: 14.58 on 5 and 69 DF,  p-value: 9.665e-10
```

```
linearm.ad.bmc = lm(ad.bmc[,1]~ ad.trt + ad.batch)
anova(linearm.ad.bmc)
```

```
## Analysis of Variance Table
##
## Response: ad.bmc[, 1]
##           Df Sum Sq Mean Sq F value Pr(>F)
## ad.trt     1  0.631 0.63084  1.3391 0.2512
## ad.batch   4  0.036 0.00893  0.0190 0.9993
## Residuals 69 32.506 0.47110
```

```
summary(linearm.ad.bmc)
```

```
##
## Call:
## lm(formula = ad.bmc[, 1] ~ ad.trt + ad.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.09885 -0.39613 -0.00381  0.36645  1.98185
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        -0.113122   0.247768  -0.457    0.649
## ad.trt1-2           0.203619   0.171183   1.189    0.238
## ad.batch14/04/2016 -0.039593   0.287918  -0.138    0.891
## ad.batch01/07/2016 -0.012928   0.273672  -0.047    0.962
## ad.batch14/11/2016  0.005323   0.282978   0.019    0.985
## ad.batch21/09/2017 -0.056561   0.306373  -0.185    0.854
##
## Residual standard error: 0.6864 on 69 degrees of freedom
## Multiple R-squared:  0.02009,    Adjusted R-squared:  -0.05091
## F-statistic: 0.283 on 5 and 69 DF,  p-value: 0.9209
```

```
linearm.ad.combat = lm(ad.combat[,1]~ ad.trt + ad.batch)
anova(linearm.ad.combat)
```

```
## Analysis of Variance Table
##
## Response: ad.combat[, 1]
##           Df  Sum Sq Mean Sq F value Pr(>F)
## ad.trt     1  0.6695 0.66954  1.6980 0.1969
## ad.batch   4  0.2373 0.05932  0.1504 0.9622
## Residuals 69 27.2080 0.39432
```

```
summary(linearm.ad.combat)
```

```
##
## Call:
## lm(formula = ad.combat[, 1] ~ ad.trt + ad.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.72209 -0.38700 -0.00346  0.34754  1.79333
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)          2.46834    0.22668  10.889   <2e-16 ***
## ad.trt1-2            0.20995    0.15661   1.341    0.184
## ad.batch14/04/2016 -0.19520    0.26341  -0.741    0.461
## ad.batch01/07/2016 -0.12986    0.25038  -0.519    0.606
## ad.batch14/11/2016 -0.09816    0.25889  -0.379    0.706
## ad.batch21/09/2017 -0.08901    0.28030  -0.318    0.752
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6279 on 69 degrees of freedom
## Multiple R-squared:  0.03225,    Adjusted R-squared:  -0.03787
## F-statistic: 0.4599 on 5 and 69 DF,  p-value: 0.8047
```

```
linearm.ad.limma = lm(ad.limma[,1]~ ad.trt + ad.batch)
anova(linearm.ad.limma)
```

```
## Analysis of Variance Table
##
## Response: ad.limma[, 1]
##           Df Sum Sq Mean Sq F value Pr(>F)
## ad.trt     1  0.704 0.70428   1.495 0.2256
## ad.batch   4  0.000 0.00000   0.000 1.0000
## Residuals 69 32.506 0.47110
```

```
summary(linearm.ad.limma)
```

```
##
## Call:
## lm(formula = ad.limma[, 1] ~ ad.trt + ad.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.09885 -0.39613 -0.00381  0.36645  1.98185
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)        2.432e+00  2.478e-01   9.815    1e-14 ***
## ad.trt1-2          2.036e-01  1.712e-01   1.189    0.238
## ad.batch14/04/2016 2.561e-15  2.879e-01   0.000    1.000
## ad.batch01/07/2016 1.175e-15  2.737e-01   0.000    1.000
## ad.batch14/11/2016 1.116e-15  2.830e-01   0.000    1.000
## ad.batch21/09/2017 2.973e-16  3.064e-01   0.000    1.000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6864 on 69 degrees of freedom
## Multiple R-squared:  0.02121,    Adjusted R-squared:  -0.04972
## F-statistic: 0.299 on 5 and 69 DF,  p-value: 0.9118
```

```
linearm.ad.percentile = lm(ad.percentile[,1]~ ad.trt + ad.batch)
anova(linearm.ad.percentile)
```

```
## Analysis of Variance Table
##
## Response: ad.percentile[, 1]
##           Df Sum Sq Mean Sq F value  Pr(>F)
## ad.trt     1 0.4670 0.46705  3.8934 0.05248 .
## ad.batch   4 1.7037 0.42592  3.5506 0.01081 *
## Residuals 69 8.2772 0.11996
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(linearm.ad.percentile)
```

```
##
## Call:
## lm(formula = ad.percentile[, 1] ~ ad.trt + ad.batch)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.5560 -0.3126  0.1613  0.2029  0.6226
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)          0.43006    0.12503   3.440 0.000992 ***
## ad.trt1-2            0.12590    0.08638   1.457 0.149528
## ad.batch14/04/2016   0.24115    0.14529   1.660 0.101497
## ad.batch01/07/2016 -0.11514    0.13810  -0.834 0.407312
## ad.batch14/11/2016   0.15770    0.14279   1.104 0.273254
## ad.batch21/09/2017   0.25670    0.15460   1.660 0.101375
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3464 on 69 degrees of freedom
## Multiple R-squared:  0.2078, Adjusted R-squared:  0.1504
## F-statistic: 3.619 on 5 and 69 DF,  p-value: 0.005766
```

```
linearm.ad.svd = lm(ad.svd[,1]~ ad.trt + ad.batch)
anova(linearm.ad.svd)
```

```
## Analysis of Variance Table
##
## Response: ad.svd[, 1]
##           Df Sum Sq Mean Sq F value    Pr(>F)
## ad.trt     1  0.222  0.2218  0.4841    0.4889
## ad.batch   4 33.914  8.4784 18.5081 2.256e-10 ***
## Residuals 69 31.608  0.4581
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(linearm.ad.svd)
```

```
##
## Call:
## lm(formula = ad.svd[, 1] ~ ad.trt + ad.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.18899 -0.39114  0.00684  0.40400  1.98037
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)         2.45910    0.24432  10.065 3.57e-15 ***
## ad.trt1-2           0.05243    0.16880   0.311 0.757055
## ad.batch14/04/2016 -0.97834    0.28391  -3.446 0.000973 ***
## ad.batch01/07/2016  0.02012    0.26987   0.075 0.940779
## ad.batch14/11/2016  0.05321    0.27904   0.191 0.849320
## ad.batch21/09/2017  1.24541    0.30211   4.122 0.000103 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6768 on 69 degrees of freedom
## Multiple R-squared:  0.5192, Adjusted R-squared:  0.4844
## F-statistic:  14.9 on 5 and 69 DF,  p-value: 6.658e-10
```

```
linearm.ad.ruv = lm(ad.ruv[,1]~ ad.trt + ad.batch)
anova(linearm.ad.ruv)
```

```
## Analysis of Variance Table
##
## Response: ad.ruv[, 1]
##           Df  Sum Sq Mean Sq F value  Pr(>F)
## ad.trt     1  1.7759 1.77595  4.4258 0.03905 *
## ad.batch   4  1.3555 0.33888  0.8445 0.50179
## Residuals 69 27.6877 0.40127
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

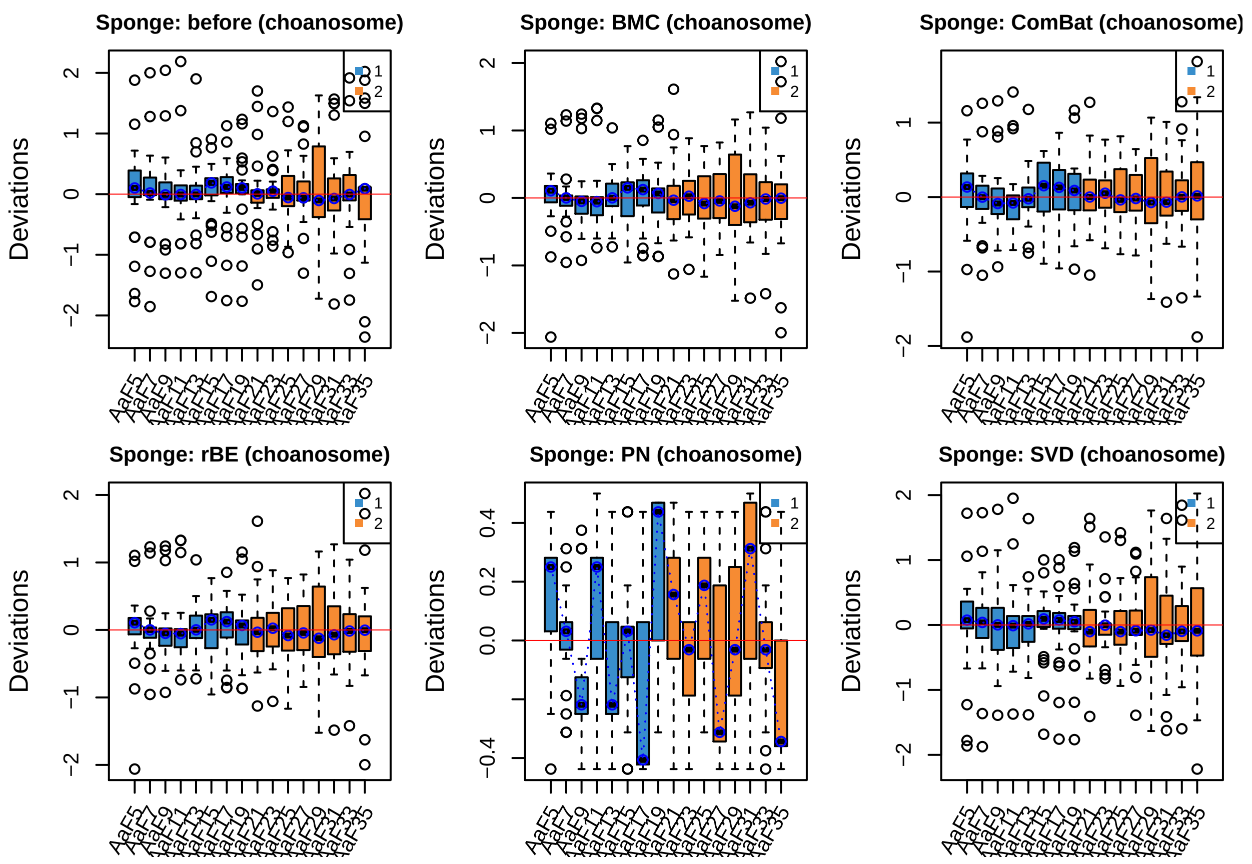
```
summary(linearm.ad.ruv)
```

```
##
## Call:
## lm(formula = ad.ruv[, 1] ~ ad.trt + ad.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.69231 -0.31474 -0.06617  0.22934  2.04171
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)          0.7435     0.2287   3.251  0.00178 **
## ad.trt1-2            0.2599     0.1580   1.645  0.10455
## ad.batch14/04/2016   0.3266     0.2657   1.229  0.22318
## ad.batch01/07/2016   0.1115     0.2526   0.441  0.66027
## ad.batch14/11/2016   0.1022     0.2612   0.391  0.69666
## ad.batch21/09/2017   0.4022     0.2828   1.422  0.15942
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6335 on 69 degrees of freedom
## Multiple R-squared:  0.1016, Adjusted R-squared:  0.03651
## F-statistic: 1.561 on 5 and 69 DF,  p-value: 0.1828
```

4.1.3 RLE plots

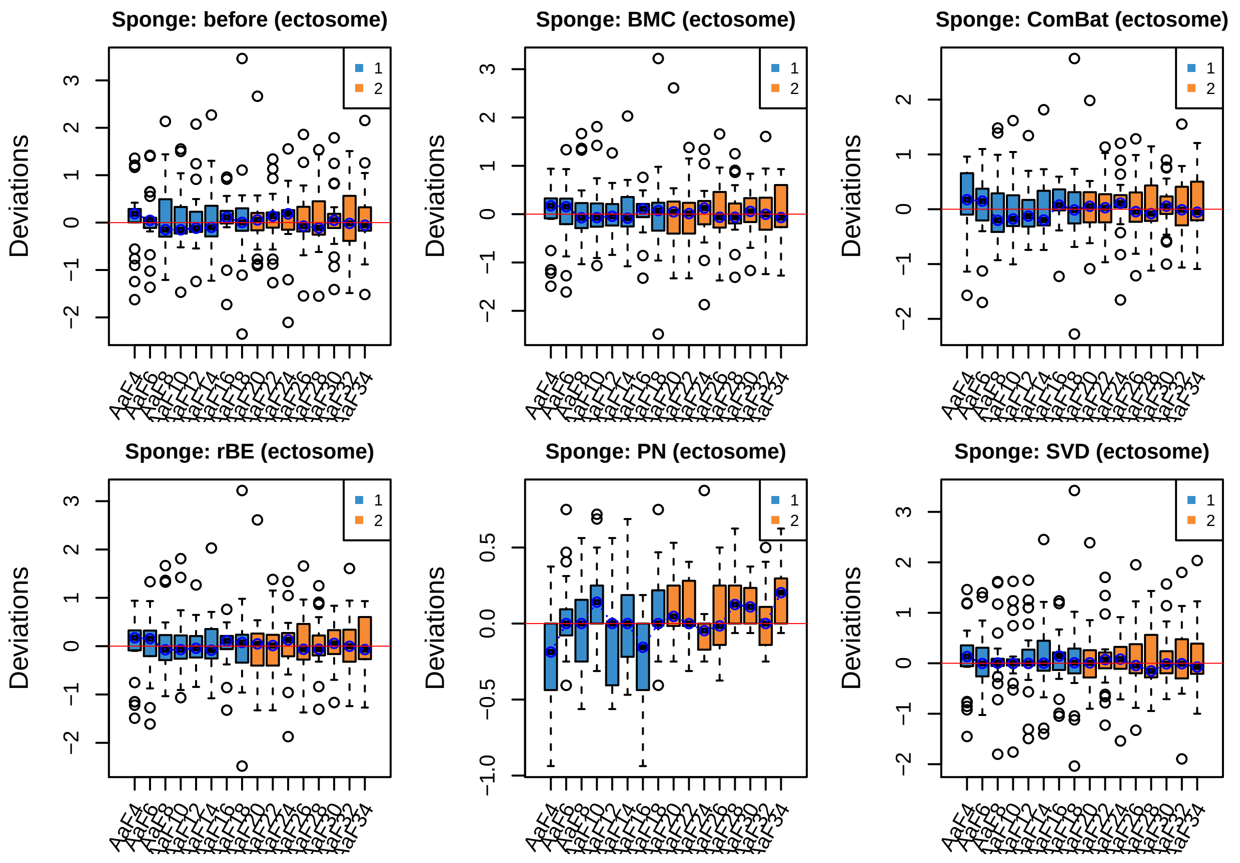
```
# sponge data
# before

###### BMC
sponge.bmc_c = sponge.bmc[sponge.trt == 'C',]
sponge.bmc_e = sponge.bmc[sponge.trt == 'E',]

###### ComBat
sponge.combat_c = sponge.combat[sponge.trt == 'C',]
sponge.combat_e = sponge.combat[sponge.trt == 'E',]

###### rBE
sponge.limma_c = sponge.limma[sponge.trt == 'C',]
sponge.limma_e = sponge.limma[sponge.trt == 'E',]

###### PN
sponge.percentile_c = sponge.percentile[sponge.trt == 'C',]
sponge.percentile_e = sponge.percentile[sponge.trt == 'E',]

###### SVD
sponge.svd_c = sponge.svd[sponge.trt == 'C',]
sponge.svd_e = sponge.svd[sponge.trt == 'E',]
```

```
par(mfrow = c(2,3), mai=c(0.4,0.6,0.3,0.1))

RleMicroRna2(object = t(sponge.before_c),batch = sponge.batch_c,maintitle = 'Sponge: before (choanosome)

RleMicroRna2(object = t(sponge.bmc_c), batch = sponge.batch_c,maintitle = 'Sponge: BMC (choanosome)',ti

RleMicroRna2(object = t(sponge.combat_c),batch = sponge.batch_c,maintitle = 'Sponge: ComBat (choanosome)

  RleMicroRna2(object = t(sponge.limma_c),batch = sponge.batch_c,maintitle = 'Sponge: rBE (choanosome)'

RleMicroRna2(object = t(sponge.percentile_c),batch = sponge.batch_c,maintitle = 'Sponge: PN (choanosome)

RleMicroRna2(object = t(sponge.svd_c),batch = sponge.batch_c,maintitle = 'Sponge: SVD (choanosome)',titl
```



```
par(mfrow = c(1,1))
```

```
par(mfrow = c(2,3), mai=c(0.4,0.6,0.3,0.1))

RleMicroRna2(object = t(sponge.before_e),batch = sponge.batch_e,maintitle = 'Sponge: before (ectosome)'

RleMicroRna2(object = t(sponge.bmc_e), batch = sponge.batch_e,maintitle = 'Sponge: BMC (ectosome)',titl

RleMicroRna2(object = t(sponge.combat_e),batch = sponge.batch_e,maintitle = 'Sponge: ComBat (ectosome)'

RleMicroRna2(object = t(sponge.limma_e),batch = sponge.batch_e,maintitle = 'Sponge: rBE (ectosome)',tit

RleMicroRna2(object = t(sponge.percentile_e),batch = sponge.batch_e,maintitle = 'Sponge: PN (ectosome)'

RleMicroRna2(object = t(sponge.svd_e),batch = sponge.batch_e,maintitle = 'Sponge: SVD (ectosome)',title
```

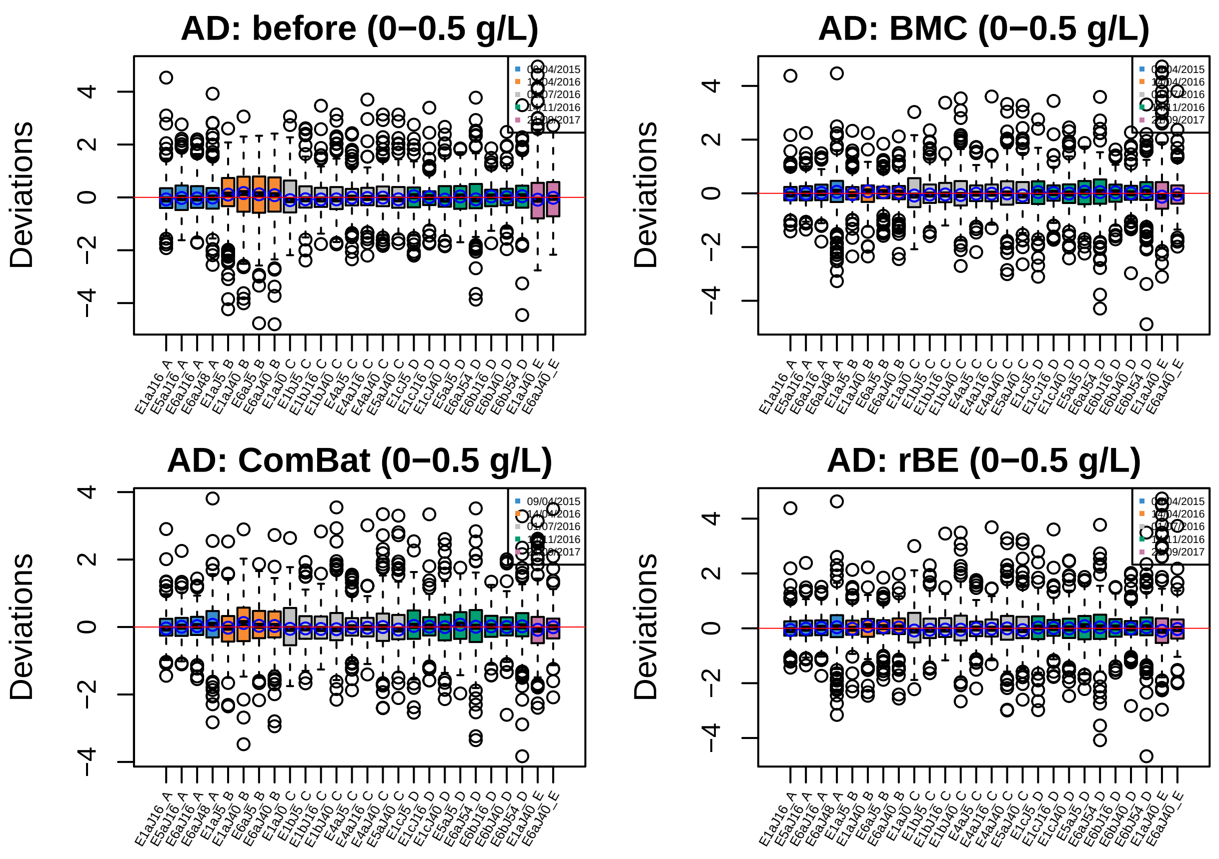


```r
par(mfrow = c(1,1))
```

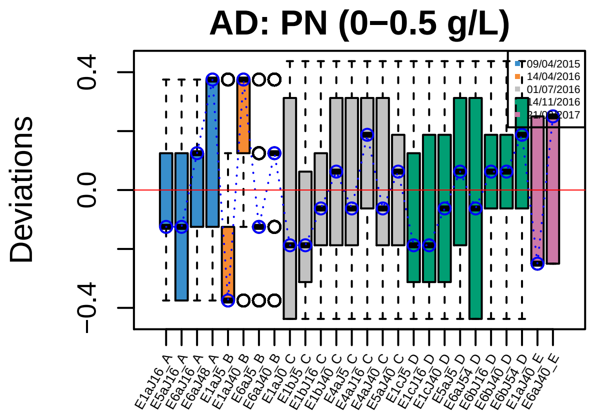
```r
#################
# ad data
# before

###### BMC
ad.bmc_05 = ad.bmc[ad.trt == '0-0.5',]
ad.bmc_2 = ad.bmc[ad.trt == '1-2',]

###### ComBat
ad.combat_05 = ad.combat[ad.trt == '0-0.5',]
ad.combat_2 = ad.combat[ad.trt == '1-2',]

###### rBE
ad.limma_05 = ad.limma[ad.trt == '0-0.5',]
ad.limma_2 = ad.limma[ad.trt == '1-2',]

###### PN
ad.percentile_05 = ad.percentile[ad.trt == '0-0.5',]
ad.percentile_2 = ad.percentile[ad.trt == '1-2',]

###### SVD
ad.svd_05 = ad.svd[ad.trt == '0-0.5',]
ad.svd_2 = ad.svd[ad.trt == '1-2',]

###### RUVIII
```

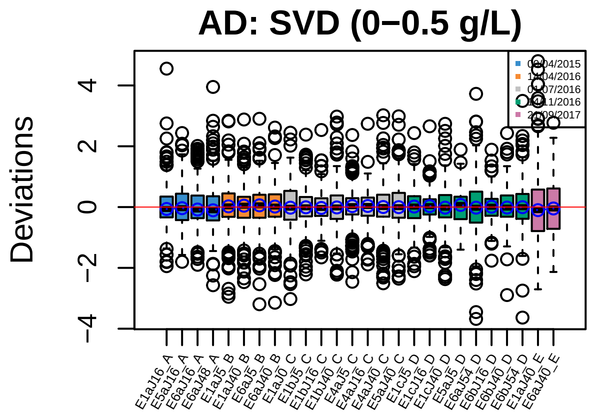
```
ad.ruv_05 = ad.ruv[ad.trt == '0-0.5',]
ad.ruv_2 = ad.ruv[ad.trt == '1-2',]
```

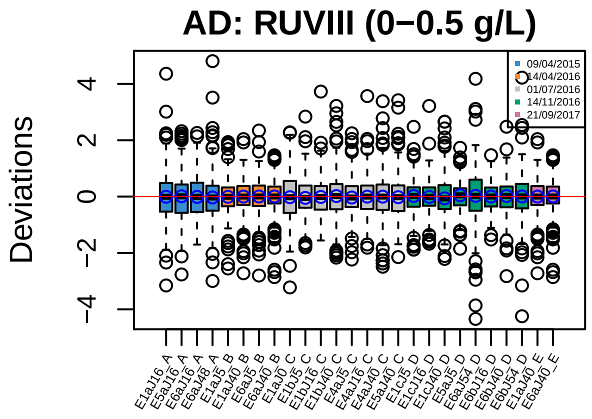
```
par(mfrow=c(2,2), mai=c(0.5,0.8,0.3,0.1))

RleMicroRna2(object = t(ad.before_05),batch = ad.batch_05,maintitle = 'AD: before (0-0.5 g/L)',legend.c

RleMicroRna2(object = t(ad.bmc_05),batch = ad.batch_05,maintitle = 'AD: BMC (0-0.5 g/L)',legend.cex = 0

RleMicroRna2(object = t(ad.combat_05),batch = ad.batch_05,maintitle = 'AD: ComBat (0-0.5 g/L)',legend.c

RleMicroRna2(object = t(ad.limma_05),batch = ad.batch_05,maintitle = 'AD: rBE (0-0.5 g/L)',legend.cex =
```



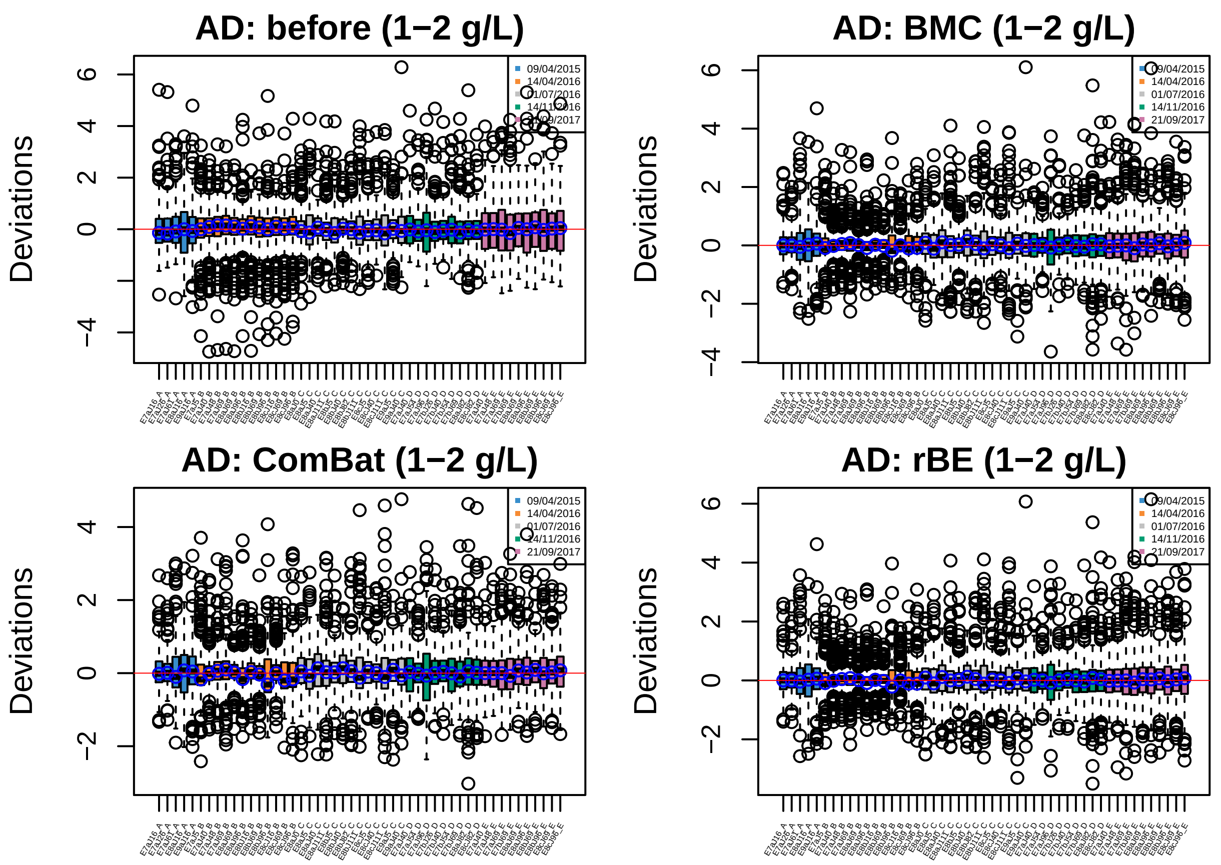
```
RleMicroRna2(object = t(ad.percentile_05),batch = ad.batch_05,maintitle = 'AD: PN (0-0.5 g/L)',legend.c

RleMicroRna2(object = t(ad.svd_05),batch = ad.batch_05,maintitle = 'AD: SVD (0-0.5 g/L)',legend.cex = 0

RleMicroRna2(object = t(ad.ruv_05),batch = ad.batch_05,maintitle = 'AD: RUVIII (0-0.5 g/L)',legend.cex 

par(mfrow = c(1,1))
```



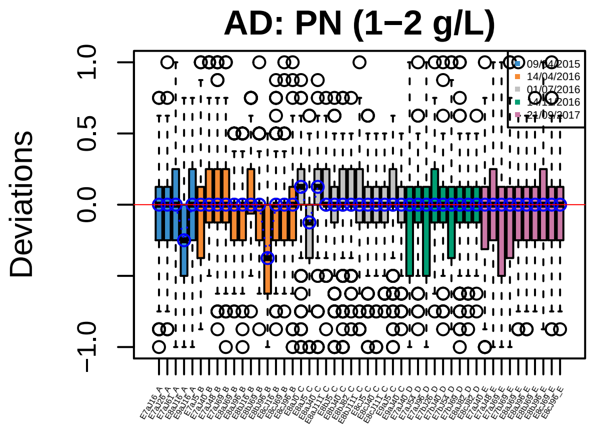


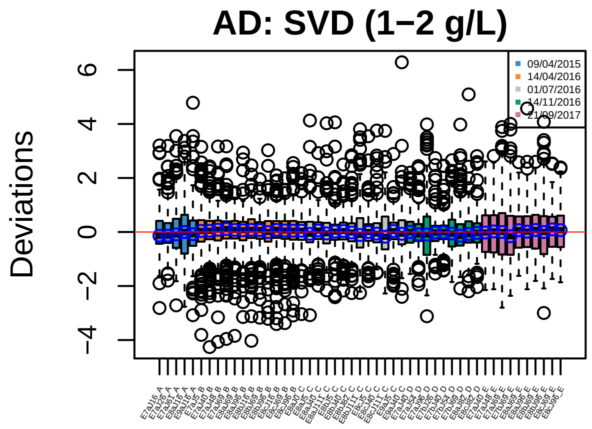


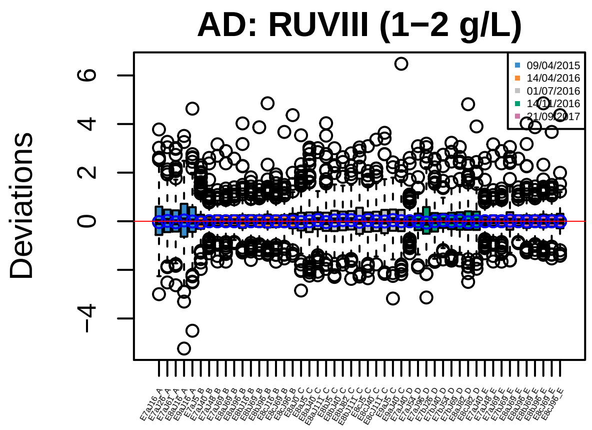
```
par(mfrow=c(2,2), mai=c(0.35,0.8,0.3,0.1))

RleMicroRna2(object = t(ad.before_2),batch = ad.batch_2,maintitle = 'AD: before (1-2 g/L)',legend.cex =

RleMicroRna2(object = t(ad.bmc_2),batch = ad.batch_2,maintitle = 'AD: BMC (1-2 g/L)',legend.cex = 0.4,

RleMicroRna2(object = t(ad.combat_2),batch = ad.batch_2,maintitle = 'AD: ComBat (1-2 g/L)',legend.cex =

RleMicroRna2(object = t(ad.limma_2),batch = ad.batch_2,maintitle = 'AD: rBE (1-2 g/L)',legend.cex = 0.4
```



```
RleMicroRna2(object = t(ad.percentile_2),batch = ad.batch_2,maintitle = 'AD: PN (1-2 g/L)',legend.cex =

RleMicroRna2(object = t(ad.svd_2),batch = ad.batch_2,maintitle = 'AD: SVD (1-2 g/L)',legend.cex = 0.4,

RleMicroRna2(object = t(ad.ruv_2),batch = ad.batch_2,maintitle = 'AD: RUVIII (1-2 g/L)',legend.cex = 0.

par(mfrow = c(1,1))
```

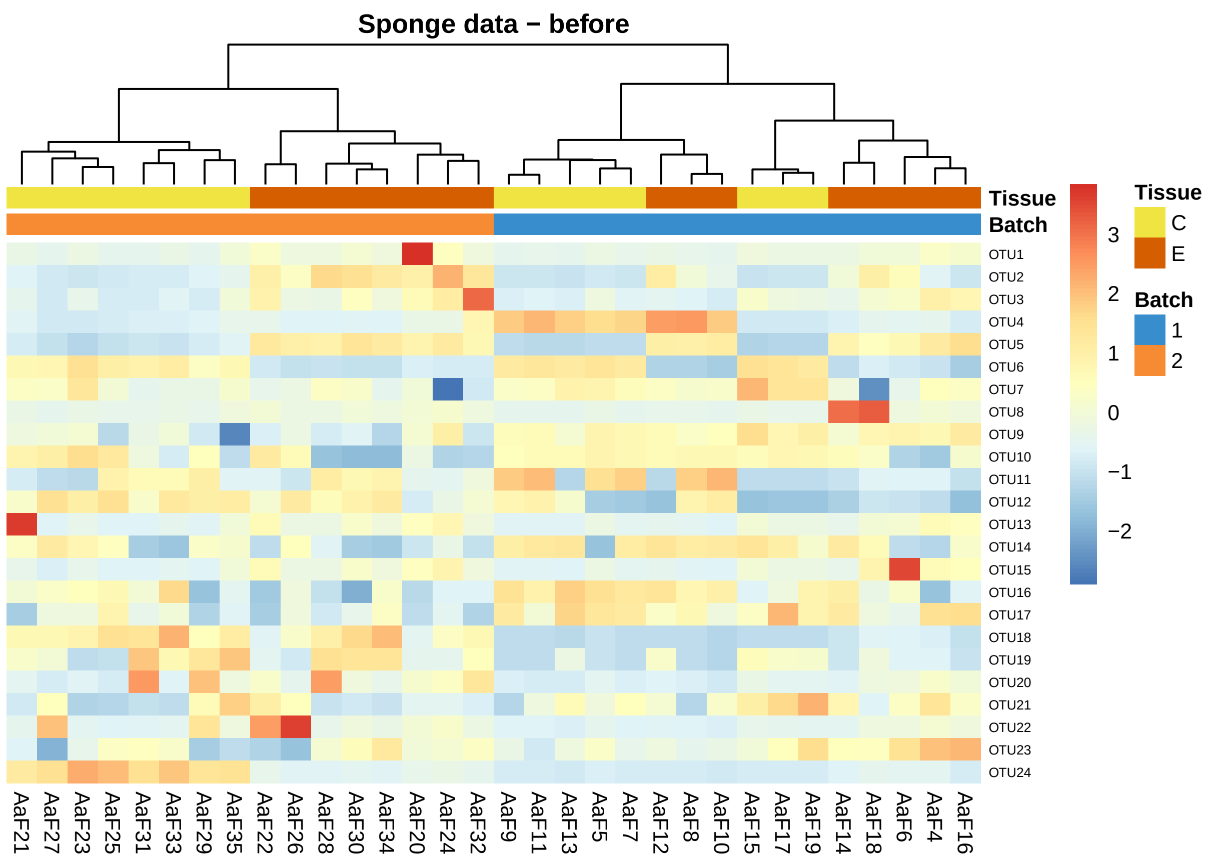




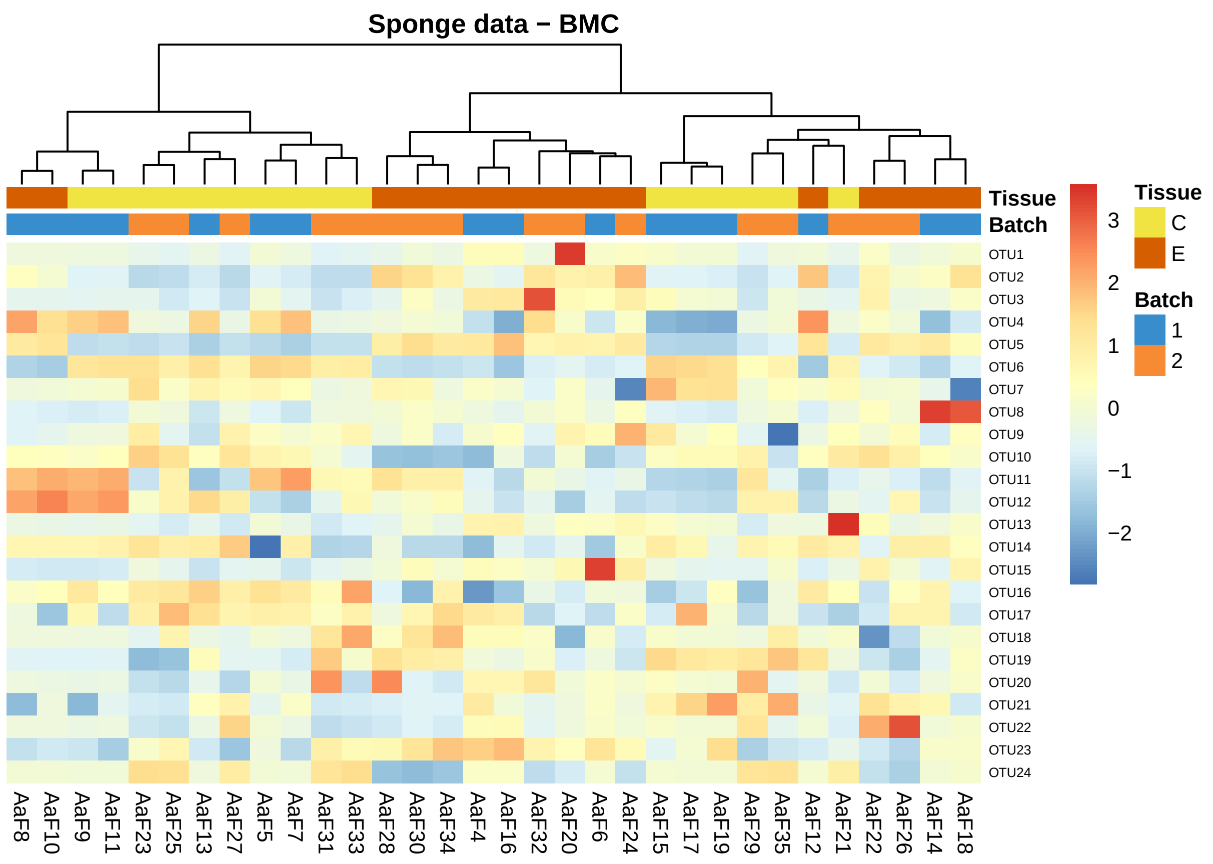


4.1.4 Heatmap

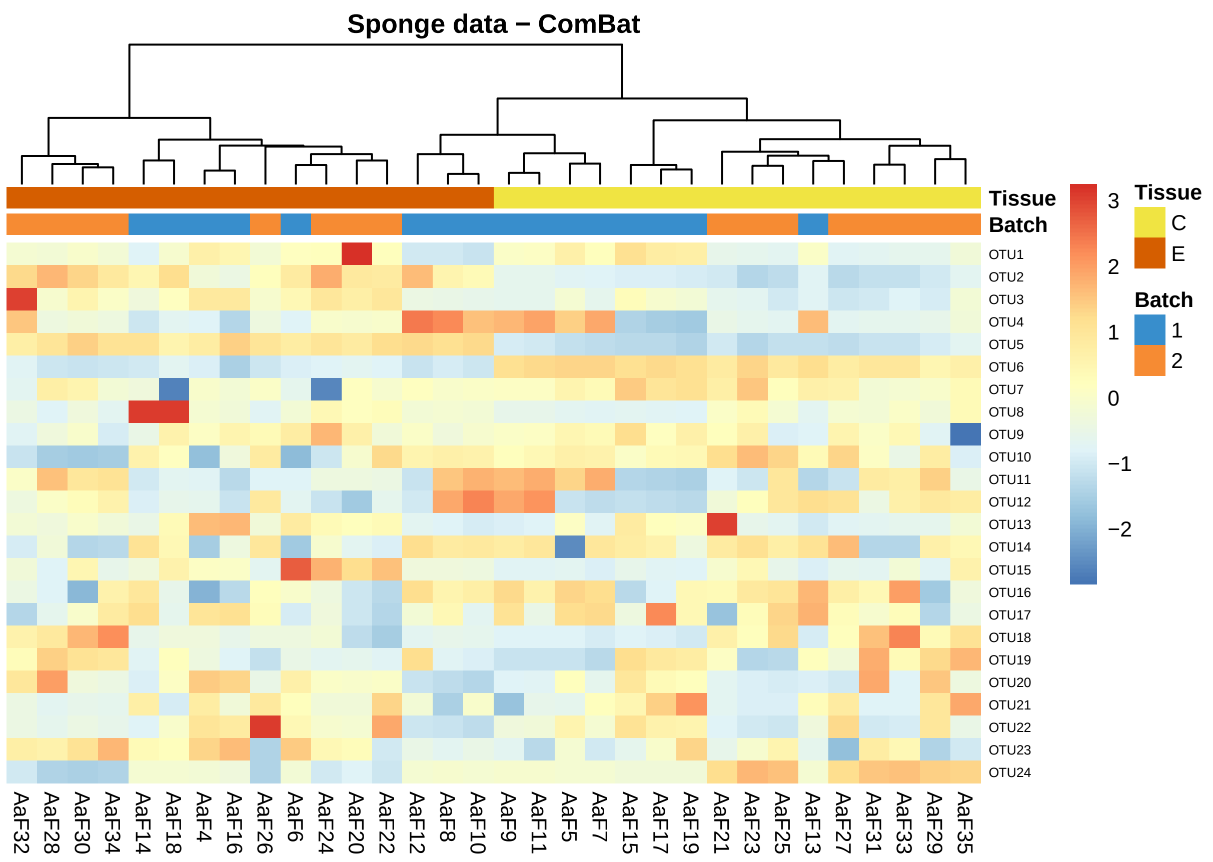
```
# Sponge data
# before
sponge.tss.clr.scale = scale(sponge.tss.clr,center = T, scale = T) # scale on OTUs
sponge.tss.clr.scale = scale(t(sponge.tss.clr.scale), center = T, scale = T) # scale on samples

anno_col.sponge = data.frame(Batch = sponge.batch, Tissue = sponge.trt)
anno_metabo_colors.sponge = list(Batch = c('1'="#388ECC",'2'="#F68B33"),Tissue = c(C="#F0E442",E="#D55E


pheatmap(sponge.tss.clr.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=5, fontsize_col=8,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = anno_col.sponge,
         annotation_colors=anno_metabo_colors.sponge,
         border_color = 'NA',
         main = 'Sponge data - before')
```



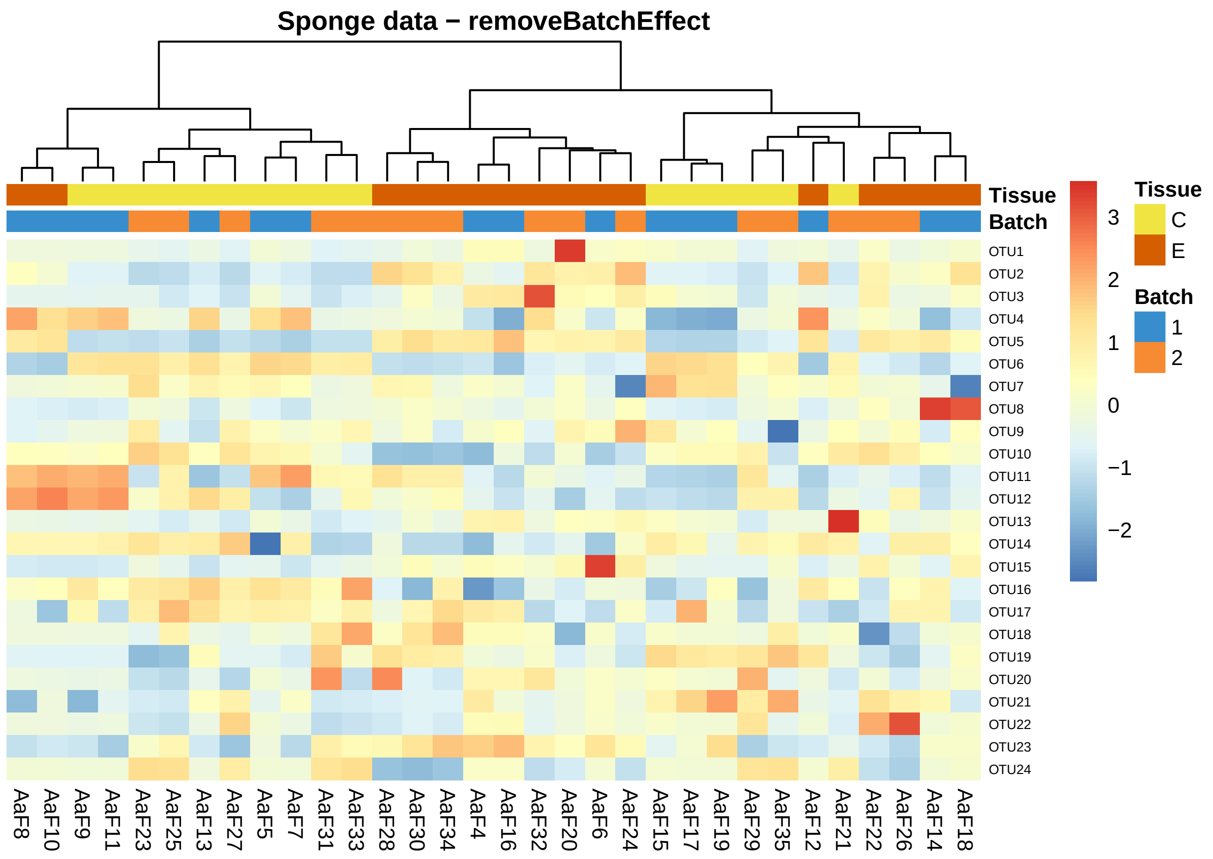
```
# BMC
sponge.bmc.scale = scale(sponge.bmc,center = T, scale = T) # scale on OTUs
sponge.bmc.scale = scale(t(sponge.bmc.scale), center = T, scale = T) # scale on samples

pheatmap(sponge.bmc.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=5, fontsize_col=8,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = anno_col.sponge,
         annotation_colors=anno_metabo_colors.sponge,
         border_color = 'NA',
         main = 'Sponge data - BMC')
```



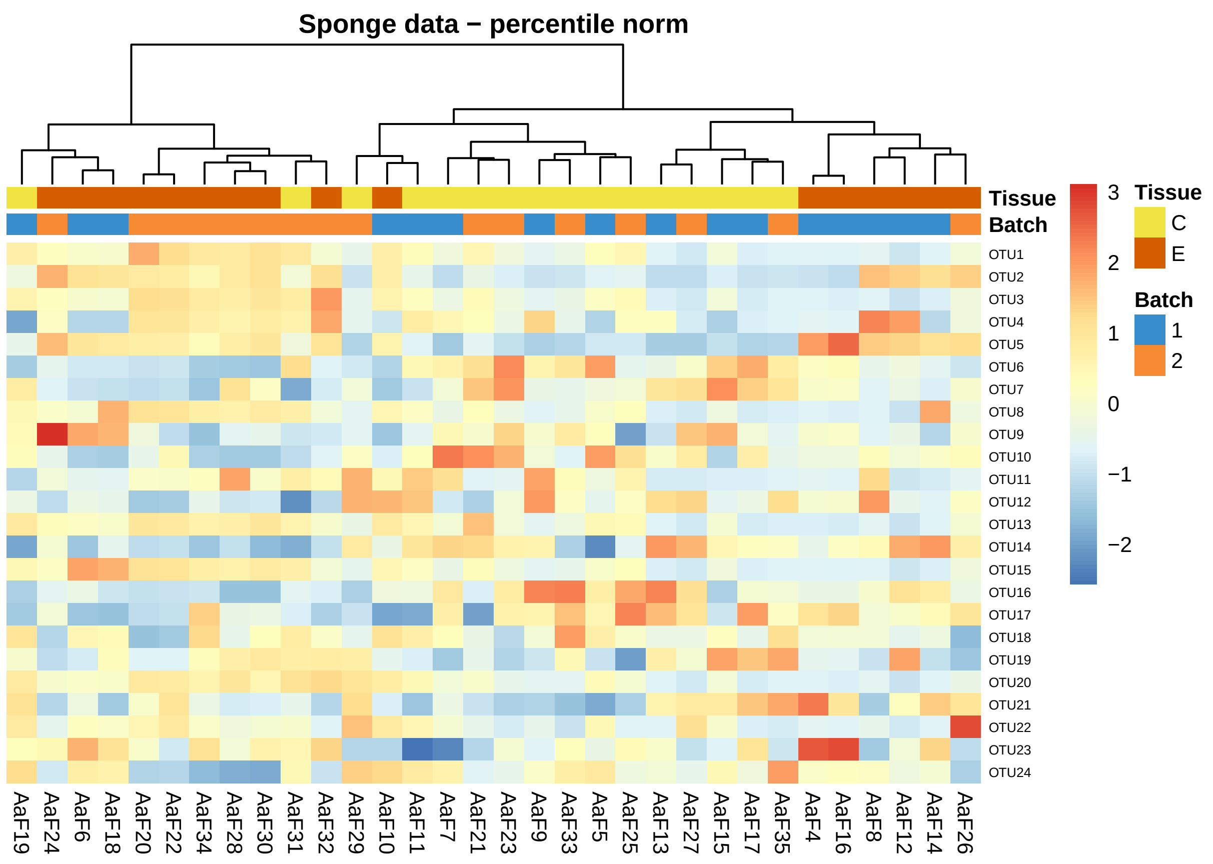
```
# ComBat
sponge.combat.scale = scale(sponge.combat,center = T, scale = T) # scale on OTUs
sponge.combat.scale = scale(t(sponge.combat.scale), center = T, scale = T) # scale on samples

pheatmap(sponge.combat.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=5, fontsize_col=8,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = anno_col.sponge,
         annotation_colors=anno_metabo_colors.sponge,
         border_color = 'NA',
         main = 'Sponge data - ComBat')
```



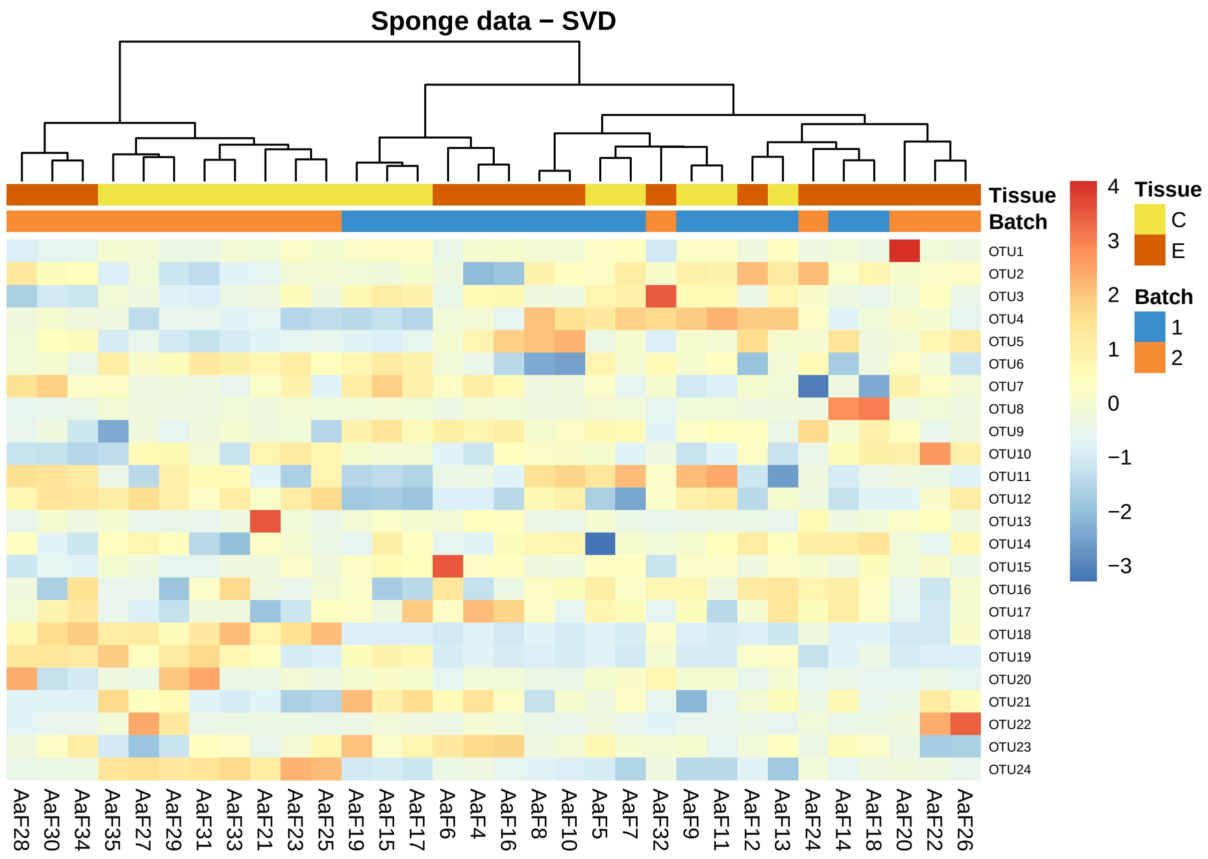
```
# removeBatchEffect
sponge.limma.scale = scale(sponge.limma,center = T, scale = T) # scale on OTUs
sponge.limma.scale = scale(t(sponge.limma.scale), center = T, scale = T) # scale on samples

pheatmap(sponge.limma.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=5, fontsize_col=8,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = anno_col.sponge,
         annotation_colors=anno_metabo_colors.sponge,
         border_color = 'NA',
         main = 'Sponge data - removeBatchEffect')
```



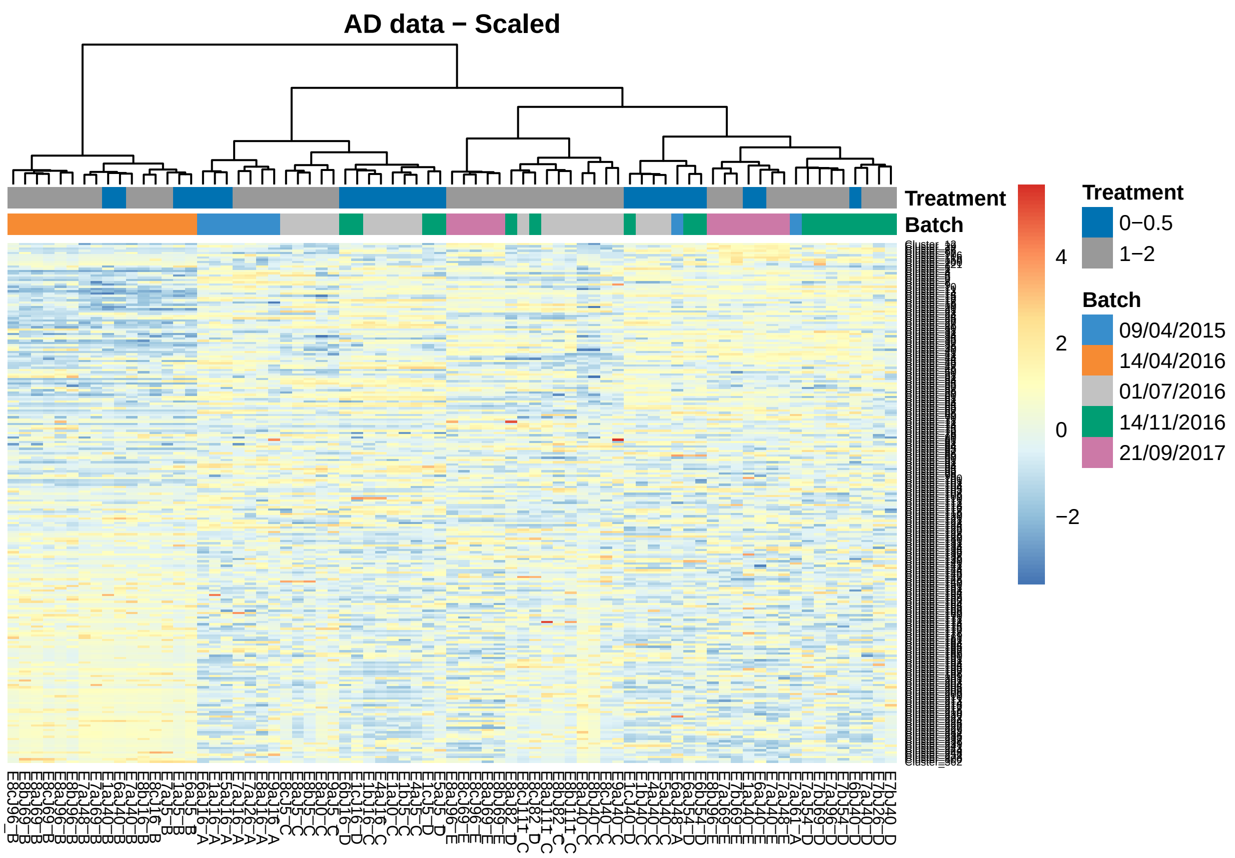
```
# percentile normalisation
sponge.percentile.scale = scale(sponge.percentile,center = T, scale = T) # scale on OTUs
sponge.percentile.scale = scale(t(sponge.percentile.scale), center = T, scale = T) # scale on samples

pheatmap(sponge.percentile.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=5, fontsize_col=8,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = anno_col.sponge,
         annotation_colors=anno_metabo_colors.sponge,
         border_color = 'NA',
         main = 'Sponge data - percentile norm')
```



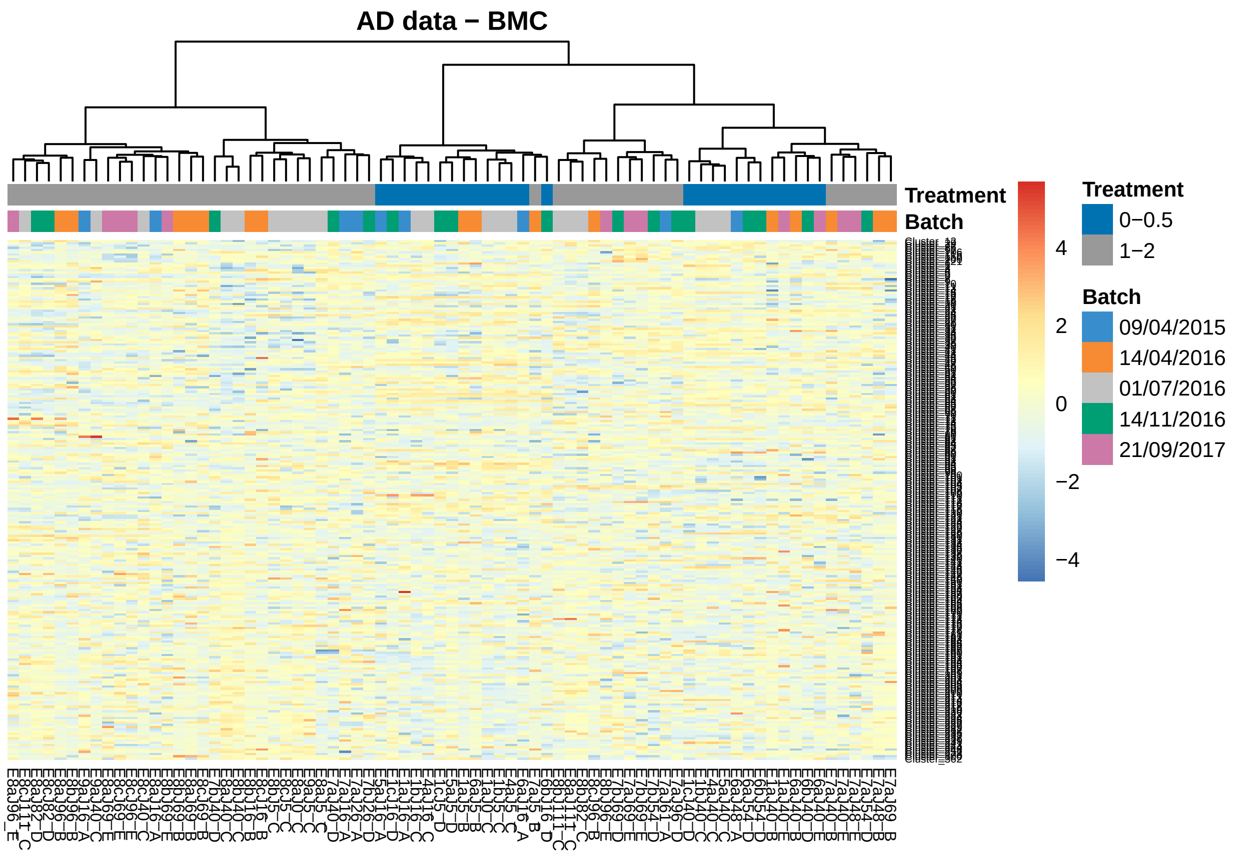
```r
# SVD
sponge.svd.scale = scale(sponge.svd,center = T, scale = T) # scale on OTUs
sponge.svd.scale = scale(t(sponge.svd.scale), center = T, scale = T) # scale on samples

pheatmap(sponge.svd.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=5, fontsize_col=8,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = anno_col.sponge,
         annotation_colors=anno_metabo_colors.sponge,
         border_color = 'NA',
         main = 'Sponge data - SVD')
```



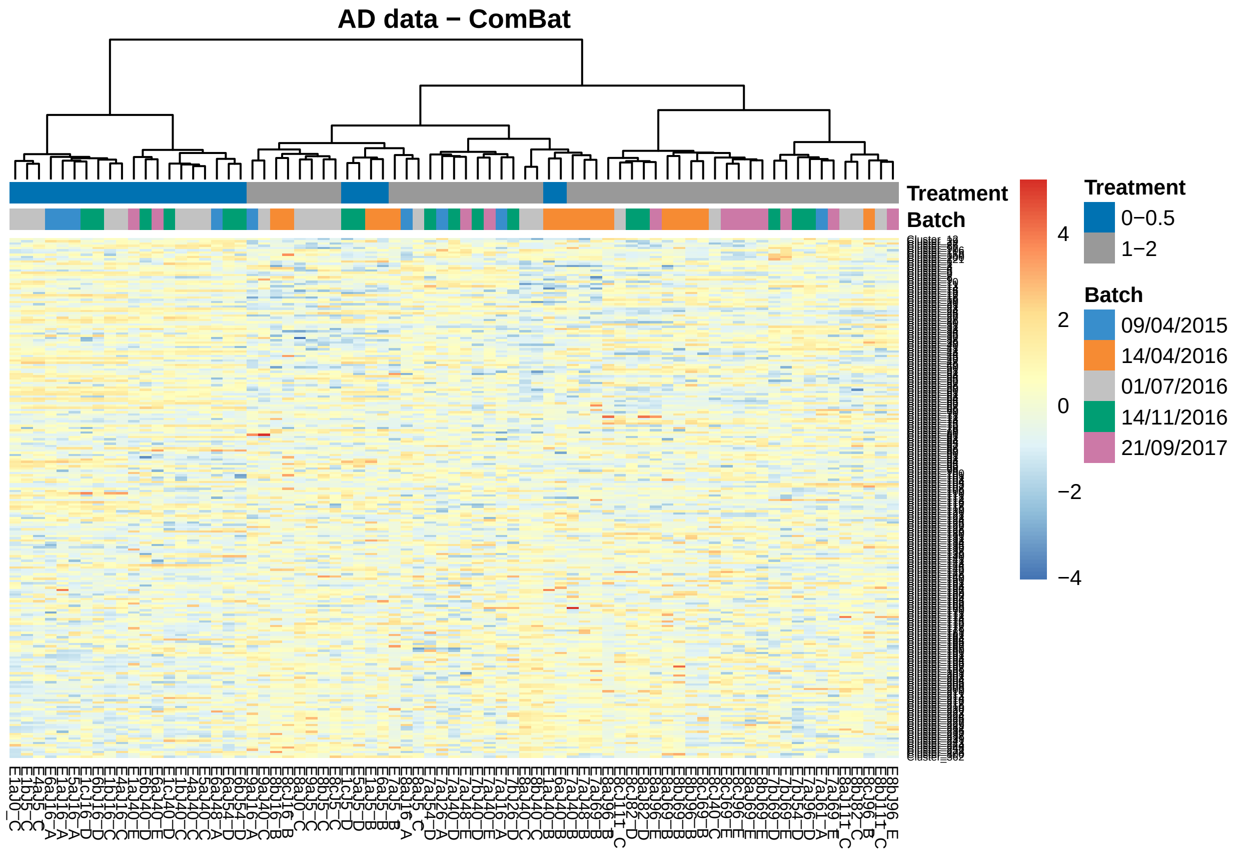
```
#################
# AD data
# before
ad.tss.clr.scale = scale(ad.tss.clr,center = T, scale = T) # scale on OTUs
ad.tss.clr.scale = scale(t(ad.tss.clr.scale), center = T, scale = T) # scale on samples

anno_col.ad = data.frame(Batch = ad.batch, Treatment = ad.trt)
anno_metabo_colors.ad = list(Batch = c('09/04/2015'="#388ECC",'14/04/2016'="#F68B33",'01/07/2016'="#C2C


pheatmap(ad.tss.clr.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = anno_col.ad,
         annotation_colors=anno_metabo_colors.ad,
         border_color = 'NA',
         main = 'AD data - Scaled')
```



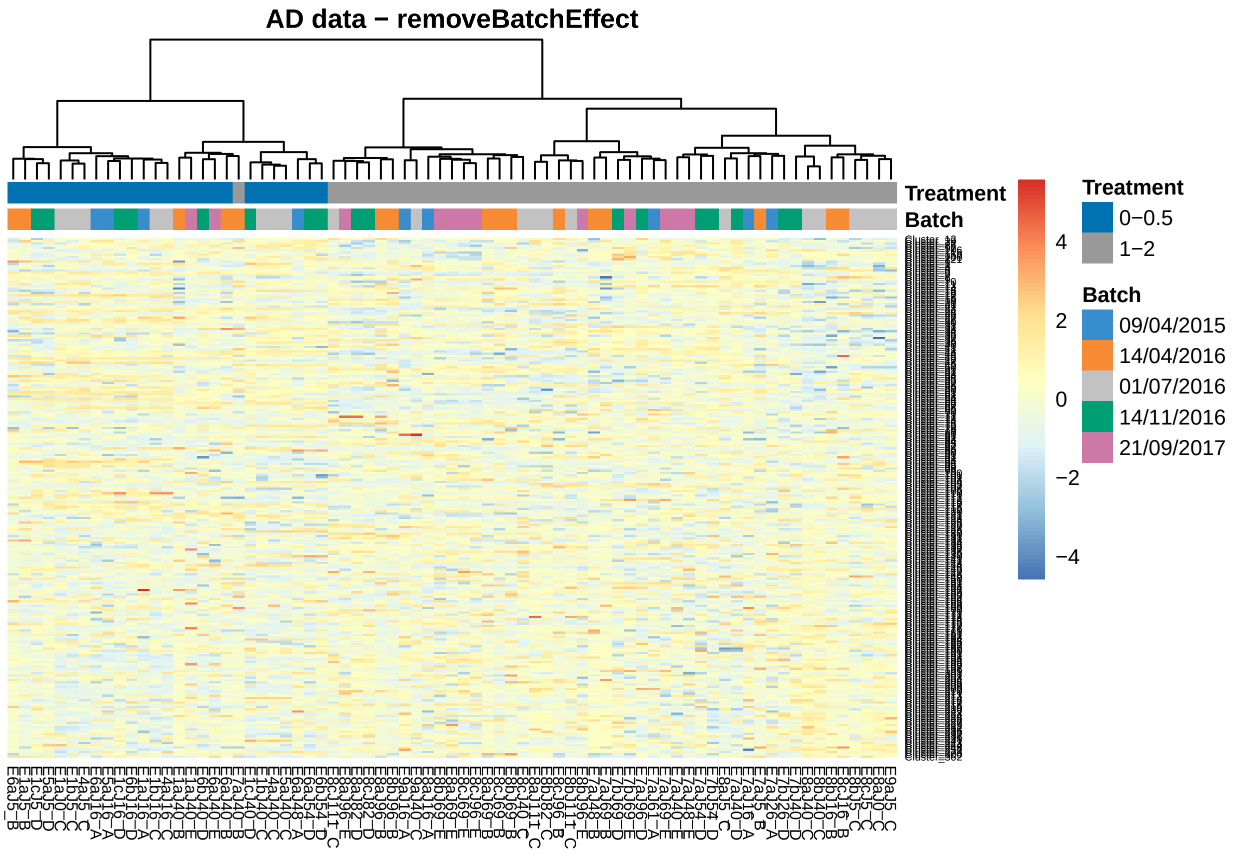
```r
# BMC
ad.bmc.scale = scale(ad.bmc,center = T, scale = T) # scale on OTUs
ad.bmc.scale = scale(t(ad.bmc.scale), center = T, scale = T) # scale on samples

pheatmap(ad.bmc.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = anno_col.ad,
         annotation_colors=anno_metabo_colors.ad,
         border_color = 'NA',
         main = 'AD data - BMC')
```



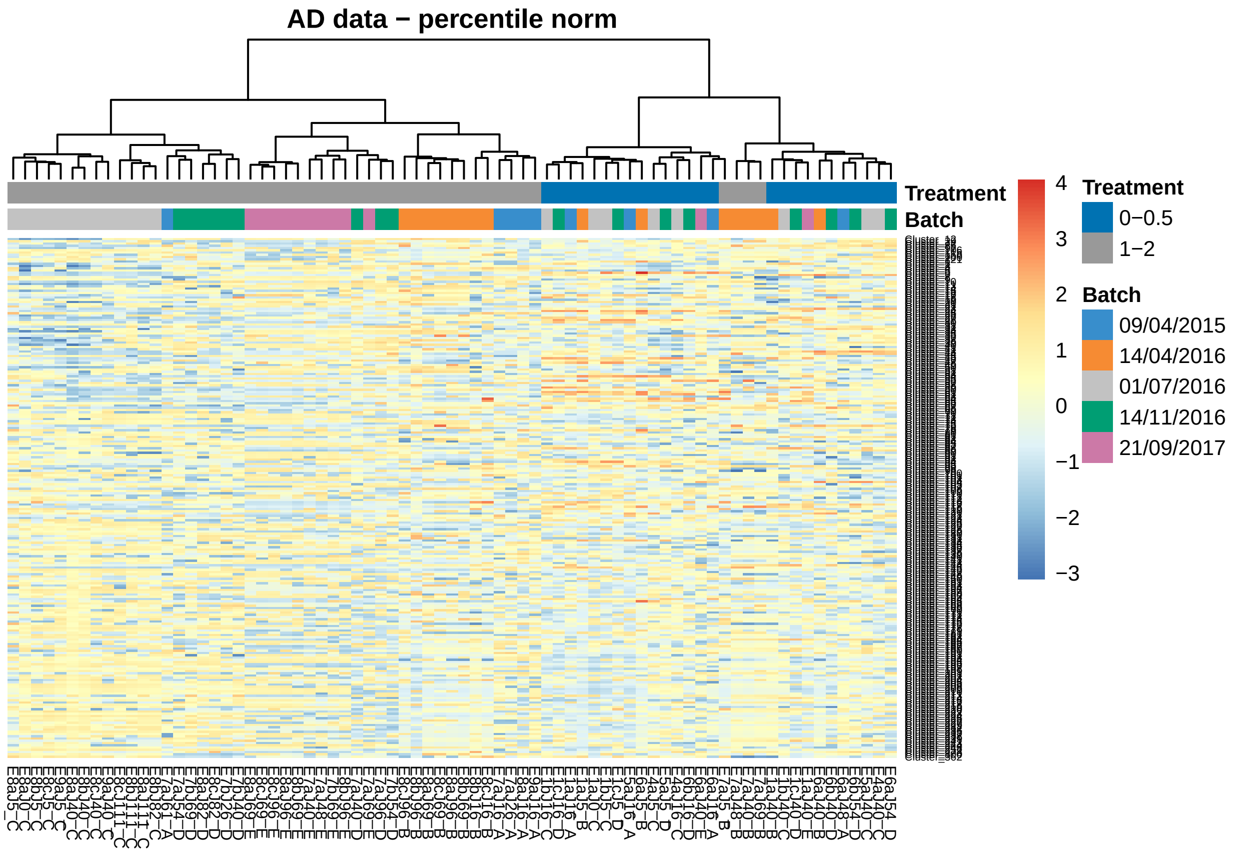
```r
# ComBat
ad.combat.scale = scale(ad.combat,center = T, scale = T) # scale on OTUs
ad.combat.scale = scale(t(ad.combat.scale), center = T, scale = T) # scale on samples

pheatmap(ad.combat.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = anno_col.ad,
         annotation_colors=anno_metabo_colors.ad,
         border_color = 'NA',
         main = 'AD data - ComBat')
```



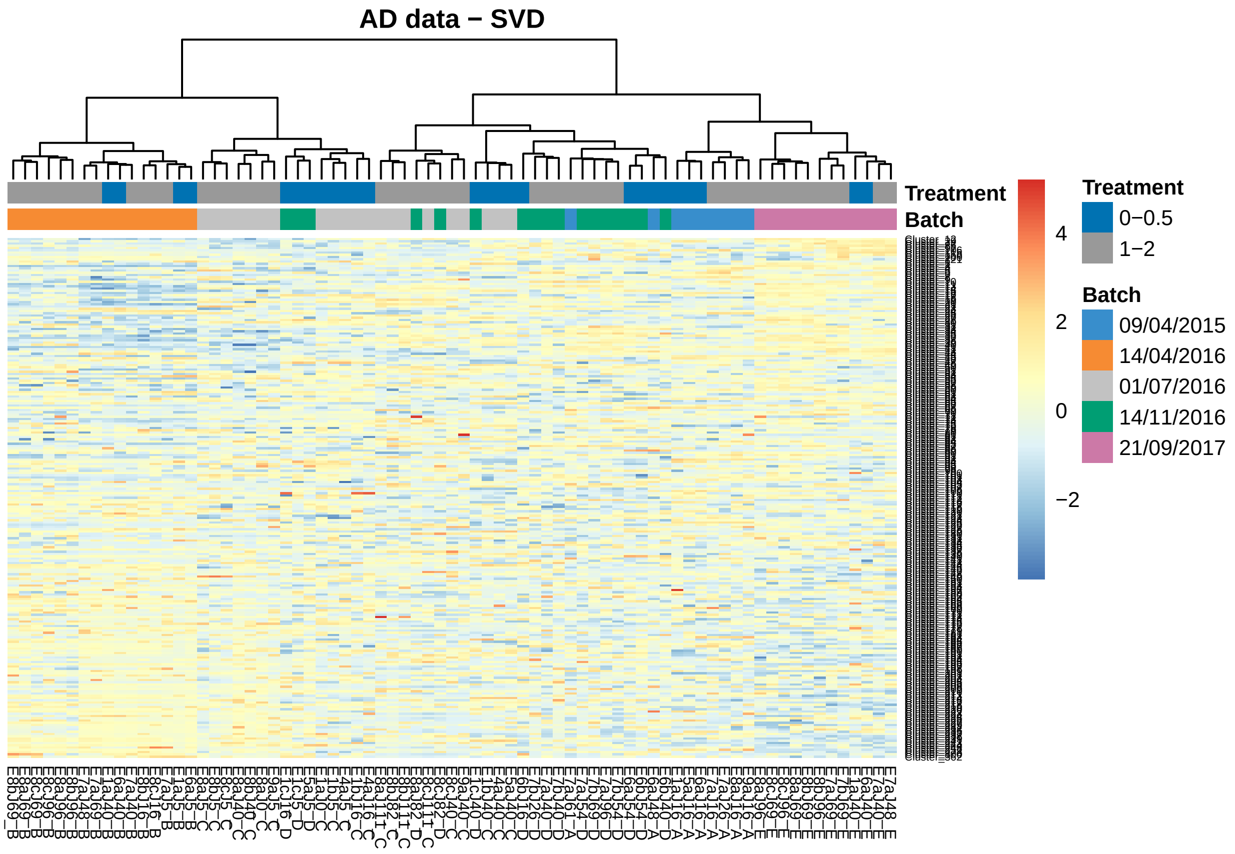
```r
# removeBatchEffect
ad.limma.scale = scale(ad.limma,center = T, scale = T) # scale on OTUs
ad.limma.scale = scale(t(ad.limma.scale), center = T, scale = T) # scale on samples

pheatmap(ad.limma.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = anno_col.ad,
         annotation_colors=anno_metabo_colors.ad,
         border_color = 'NA',
         main = 'AD data - removeBatchEffect')
```



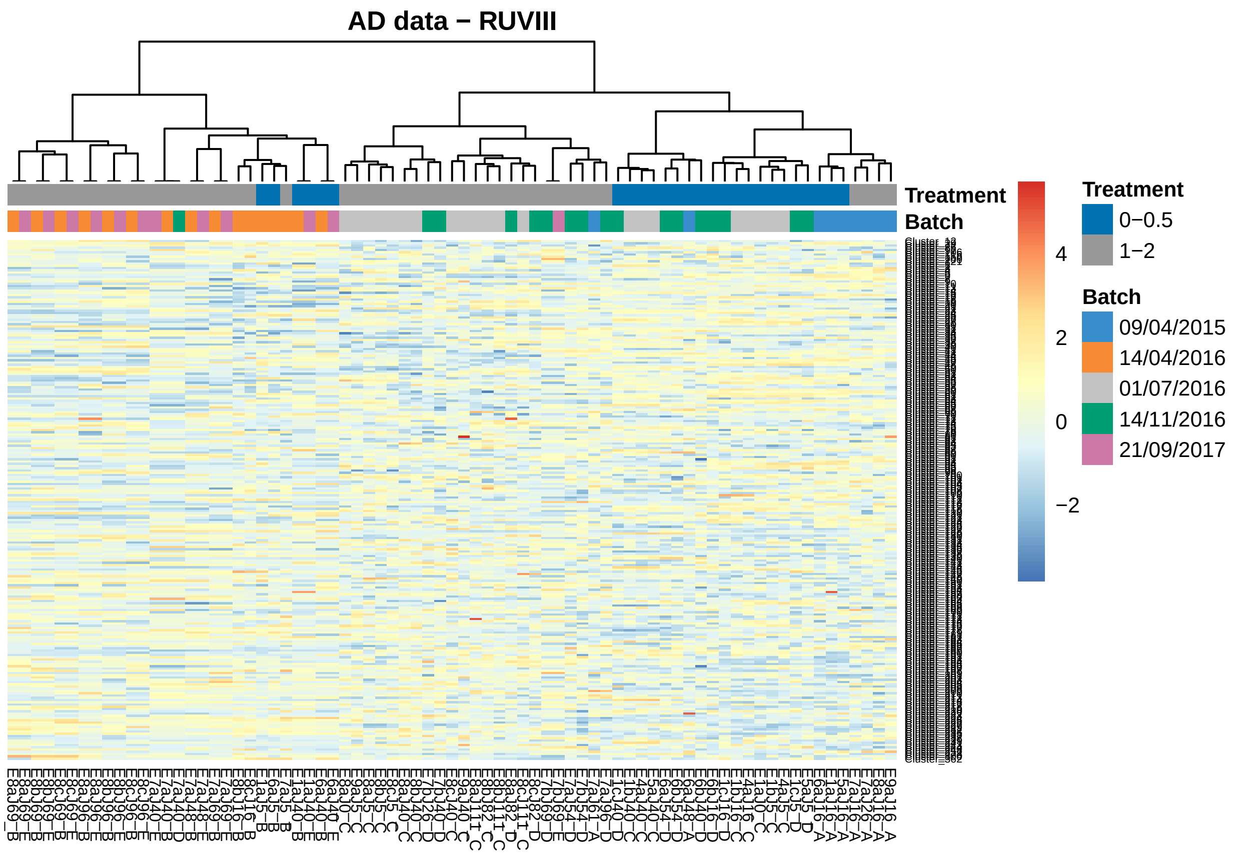
```
# percentile normalisation
ad.percentile.scale = scale(ad.percentile,center = T, scale = T) # scale on OTUs
ad.percentile.scale = scale(t(ad.percentile.scale), center = T, scale = T) # scale on samples

pheatmap(ad.percentile.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = anno_col.ad,
         annotation_colors=anno_metabo_colors.ad,
         border_color = 'NA',
         main = 'AD data - percentile norm')
```



```r
# SVD
ad.svd.scale = scale(ad.svd,center = T, scale = T) # scale on OTUs
ad.svd.scale = scale(t(ad.svd.scale), center = T, scale = T) # scale on samples

pheatmap(ad.svd.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = anno_col.ad,
         annotation_colors=anno_metabo_colors.ad,
         border_color = 'NA',
         main = 'AD data - SVD')
```



```r
# RUVIII
ad.ruv.scale = scale(ad.ruv,center = T, scale = T) # scale on OTUs
ad.ruv.scale = scale(t(ad.ruv.scale), center = T, scale = T) # scale on samples

pheatmap(ad.ruv.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = anno_col.ad,
         annotation_colors=anno_metabo_colors.ad,
         border_color = 'NA',
         main = 'AD data - RUVIII')
```



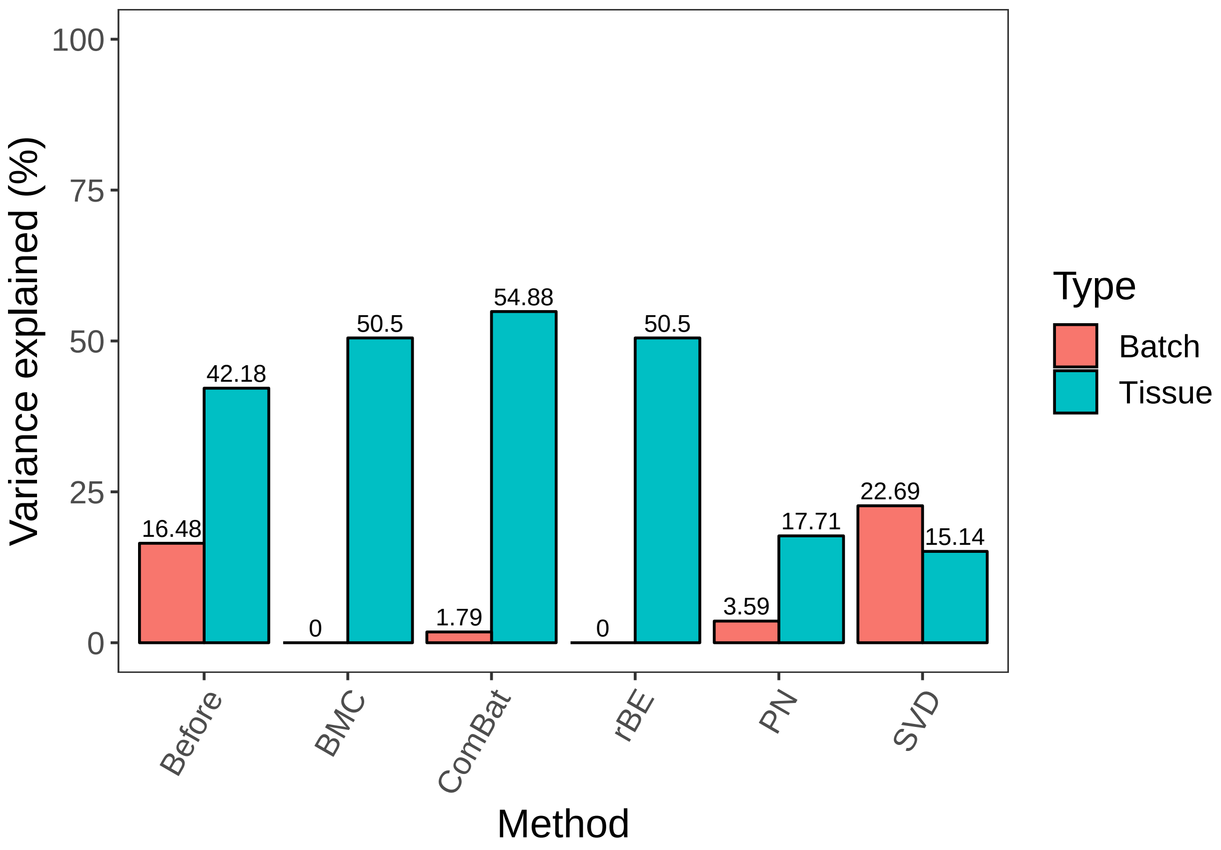
4.2 Variance calculation

4.2.1 RDA

```
# Sponge data
data.design.sponge = numeric()
data.design.sponge$group = sponge.trt
data.design.sponge$batch = sponge.batch

# before
# conditioning on a batch effect
rda.sponge.before1 = rda(sponge.tss.clr ~ group + Condition(batch), data = data.design.sponge)
rda.sponge.before2 = rda(sponge.tss.clr ~ batch + Condition(group), data = data.design.sponge)

# amount of variance
rda.prop.bat.sponge.before = rda.sponge.before1$pCCA$tot.chi*100/rda.sponge.before1$tot.chi
rda.prop.trt.sponge.before = rda.sponge.before2$pCCA$tot.chi*100/rda.sponge.before1$tot.chi

# BMC
# conditioning on a batch effect
rda.sponge.bmc1 = rda(sponge.bmc ~ group + Condition(batch), data = data.design.sponge)
rda.sponge.bmc2 = rda(sponge.bmc ~ batch + Condition(group), data = data.design.sponge)

# amount of variance
```

```
rda.prop.bat.sponge.bmc = rda.sponge.bmc1$pCCA$tot.chi*100/rda.sponge.bmc1$tot.chi
rda.prop.trt.sponge.bmc = rda.sponge.bmc2$pCCA$tot.chi*100/rda.sponge.bmc2$tot.chi


# combat
# conditioning on a batch effect
rda.sponge.combat1 = rda(sponge.combat ~ group + Condition(batch), data = data.design.sponge)
rda.sponge.combat2 = rda(sponge.combat ~ batch + Condition(group), data = data.design.sponge)

# amount of variance
rda.prop.bat.sponge.combat = rda.sponge.combat1$pCCA$tot.chi*100/rda.sponge.combat1$tot.chi
rda.prop.trt.sponge.combat = rda.sponge.combat2$pCCA$tot.chi*100/rda.sponge.combat2$tot.chi


# limma
# conditioning on a batch effect
rda.sponge.limma1 = rda(sponge.limma ~ group + Condition(batch), data = data.design.sponge)
rda.sponge.limma2 = rda(sponge.limma ~ batch + Condition(group), data = data.design.sponge)

# amount of variance
rda.prop.bat.sponge.limma = rda.sponge.limma1$pCCA$tot.chi*100/rda.sponge.limma1$tot.chi
rda.prop.trt.sponge.limma = rda.sponge.limma2$pCCA$tot.chi*100/rda.sponge.limma2$tot.chi


# percentile
# conditioning on a batch effect
rda.sponge.percentile1 = rda(sponge.percentile ~ group + Condition(batch), data = data.design.sponge)
rda.sponge.percentile2 = rda(sponge.percentile ~ batch + Condition(group), data = data.design.sponge)

# amount of variance
rda.prop.bat.sponge.percentile = rda.sponge.percentile1$pCCA$tot.chi*100/rda.sponge.percentile1$tot.chi
rda.prop.trt.sponge.percentile = rda.sponge.percentile2$pCCA$tot.chi*100/rda.sponge.percentile2$tot.chi


# SVD
# conditioning on a batch effect
rda.sponge.svd1 = rda(sponge.svd ~ group + Condition(batch), data = data.design.sponge)
rda.sponge.svd2 = rda(sponge.svd ~ batch + Condition(group), data = data.design.sponge)

# amount of variance
rda.prop.bat.sponge.svd = rda.sponge.svd1$pCCA$tot.chi*100/rda.sponge.svd1$tot.chi
rda.prop.trt.sponge.svd = rda.sponge.svd2$pCCA$tot.chi*100/rda.sponge.svd2$tot.chi
```

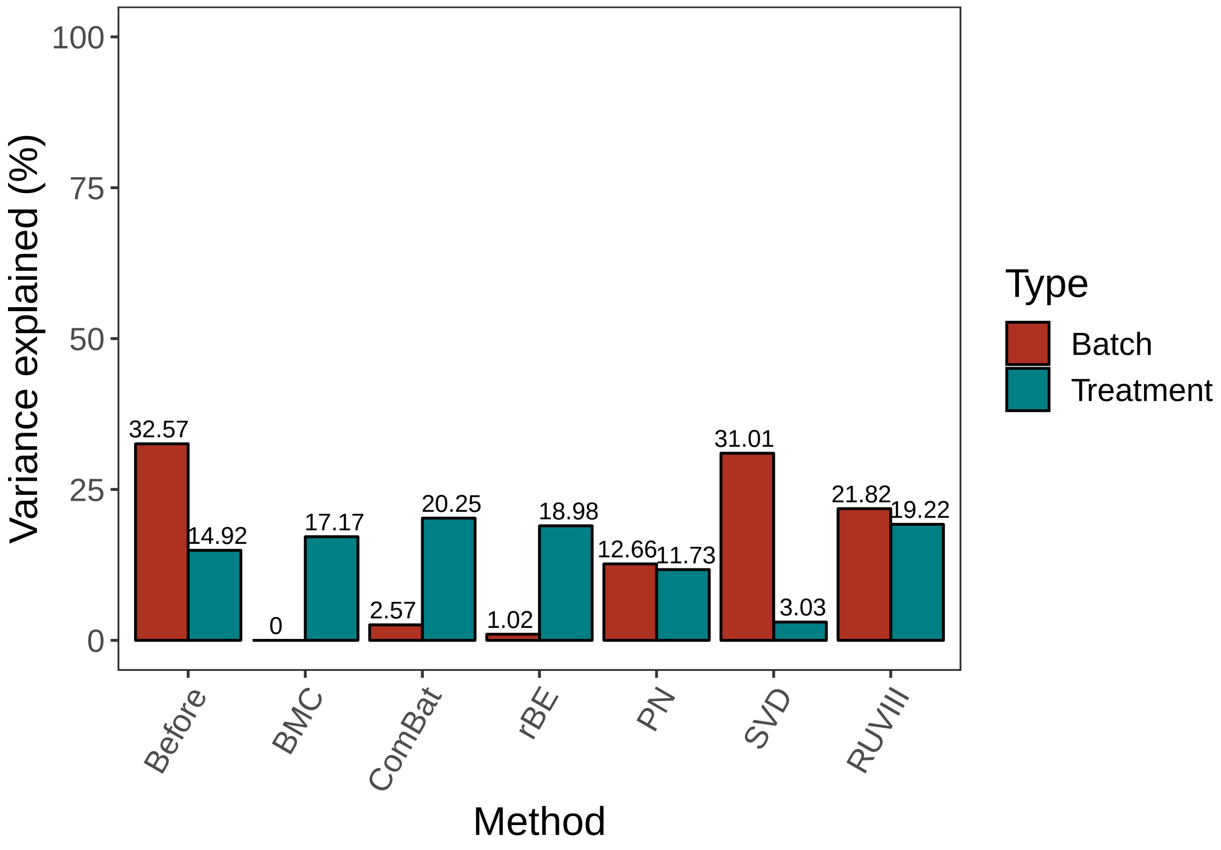
```
# proportion
rda.prop.sponge.before = c(rda.prop.bat.sponge.before,rda.prop.trt.sponge.before)
rda.prop.sponge.bmc = c(rda.prop.bat.sponge.bmc,rda.prop.trt.sponge.bmc)
rda.prop.sponge.combat = c(rda.prop.bat.sponge.combat,rda.prop.trt.sponge.combat)
rda.prop.sponge.limma = c(rda.prop.bat.sponge.limma,rda.prop.trt.sponge.limma)
rda.prop.sponge.percentile = c(rda.prop.bat.sponge.percentile,rda.prop.trt.sponge.percentile)
rda.prop.sponge.svd= c(rda.prop.bat.sponge.svd,rda.prop.trt.sponge.svd)

rda.prop.sponge.val = c(rda.prop.sponge.before,rda.prop.sponge.bmc,rda.prop.sponge.combat,rda.prop.spon
rda.prop.sponge = data.frame(prop = rda.prop.sponge.val, prop.r = round(rda.prop.sponge.val,2), Method
rda.prop.sponge$Method =  factor(rda.prop.sponge$Method, levels = unique(rda.prop.sponge$Method))

ggplot(data = rda.prop.sponge, aes(x=Method,y=prop,fill = Type)) + geom_bar(stat="identity",position =
```



```
# AD data
data.design.ad = numeric()
data.design.ad$group = ad.trt
data.design.ad$batch = ad.batch

# before
# conditioning on a batch effect
rda.ad.before1 = rda(ad.tss.clr ~ group + Condition(batch), data = data.design.ad)
rda.ad.before2 = rda(ad.tss.clr ~ batch + Condition(group), data = data.design.ad)

# amount of variance
rda.prop.bat.ad.before = rda.ad.before1$pCCA$tot.chi*100/rda.ad.before1$tot.chi
rda.prop.trt.ad.before = rda.ad.before2$pCCA$tot.chi*100/rda.ad.before1$tot.chi

# BMC
# conditioning on a batch effect
rda.ad.bmc1 = rda(ad.bmc ~ group + Condition(batch), data = data.design.ad)
rda.ad.bmc2 = rda(ad.bmc ~ batch + Condition(group), data = data.design.ad)

# amount of variance
rda.prop.bat.ad.bmc = rda.ad.bmc1$pCCA$tot.chi*100/rda.ad.bmc1$tot.chi
rda.prop.trt.ad.bmc = rda.ad.bmc2$pCCA$tot.chi*100/rda.ad.bmc2$tot.chi
```

```
# combat
# conditioning on a batch effect
rda.ad.combat1 = rda(ad.combat ~ group + Condition(batch), data = data.design.ad)
rda.ad.combat2 = rda(ad.combat ~ batch + Condition(group), data = data.design.ad)

# amount of variance
rda.prop.bat.ad.combat = rda.ad.combat1$pCCA$tot.chi*100/rda.ad.combat1$tot.chi
rda.prop.trt.ad.combat = rda.ad.combat2$pCCA$tot.chi*100/rda.ad.combat2$tot.chi


# limma
# conditioning on a batch effect
rda.ad.limma1 = rda(ad.limma ~ group + Condition(batch), data = data.design.ad)
rda.ad.limma2 = rda(ad.limma ~ batch + Condition(group), data = data.design.ad)

# amount of variance
rda.prop.bat.ad.limma = rda.ad.limma1$pCCA$tot.chi*100/rda.ad.limma1$tot.chi
rda.prop.trt.ad.limma = rda.ad.limma2$pCCA$tot.chi*100/rda.ad.limma2$tot.chi


# percentile
# conditioning on a batch effect
rda.ad.percentile1 = rda(ad.percentile ~ group + Condition(batch), data = data.design.ad)
rda.ad.percentile2 = rda(ad.percentile ~ batch + Condition(group), data = data.design.ad)

# amount of variance
rda.prop.bat.ad.percentile = rda.ad.percentile1$pCCA$tot.chi*100/rda.ad.percentile1$tot.chi
rda.prop.trt.ad.percentile = rda.ad.percentile2$pCCA$tot.chi*100/rda.ad.percentile2$tot.chi


# SVD
# conditioning on a batch effect
rda.ad.svd1 = rda(ad.svd ~ group + Condition(batch), data = data.design.ad)
rda.ad.svd2 = rda(ad.svd ~ batch + Condition(group), data = data.design.ad)

# amount of variance
rda.prop.bat.ad.svd = rda.ad.svd1$pCCA$tot.chi*100/rda.ad.svd1$tot.chi
rda.prop.trt.ad.svd = rda.ad.svd2$pCCA$tot.chi*100/rda.ad.svd2$tot.chi


# RUVIII
# conditioning on a batch effect
rda.ad.ruv1 = rda(ad.ruv ~ group + Condition(batch), data = data.design.ad)
rda.ad.ruv2 = rda(ad.ruv ~ batch + Condition(group), data = data.design.ad)

# amount of variance
rda.prop.bat.ad.ruv = rda.ad.ruv1$pCCA$tot.chi*100/rda.ad.ruv1$tot.chi
rda.prop.trt.ad.ruv = rda.ad.ruv2$pCCA$tot.chi*100/rda.ad.ruv2$tot.chi
```

```
# proportion
rda.prop.ad.before = c(rda.prop.bat.ad.before,rda.prop.trt.ad.before)
rda.prop.ad.bmc = c(rda.prop.bat.ad.bmc,rda.prop.trt.ad.bmc)
rda.prop.ad.combat = c(rda.prop.bat.ad.combat,rda.prop.trt.ad.combat)
rda.prop.ad.limma = c(rda.prop.bat.ad.limma,rda.prop.trt.ad.limma)
rda.prop.ad.percentile = c(rda.prop.bat.ad.percentile,rda.prop.trt.ad.percentile)
rda.prop.ad.svd= c(rda.prop.bat.ad.svd,rda.prop.trt.ad.svd)
rda.prop.ad.ruv= c(rda.prop.bat.ad.ruv,rda.prop.trt.ad.ruv)

##############
rda.prop.ad.val = c(rda.prop.ad.before,rda.prop.ad.bmc,rda.prop.ad.combat,rda.prop.ad.limma,rda.prop.ad
rda.prop.ad = data.frame(prop = rda.prop.ad.val, prop.r = round(rda.prop.ad.val,2), Method = rep(c('Bef

rda.prop.ad$Method =  factor(rda.prop.ad$Method, levels = unique(rda.prop.ad$Method))

ggplot(data = rda.prop.ad, aes(x=Method,y=prop,fill = Type)) + geom_bar(stat="identity",position = 'dod
```



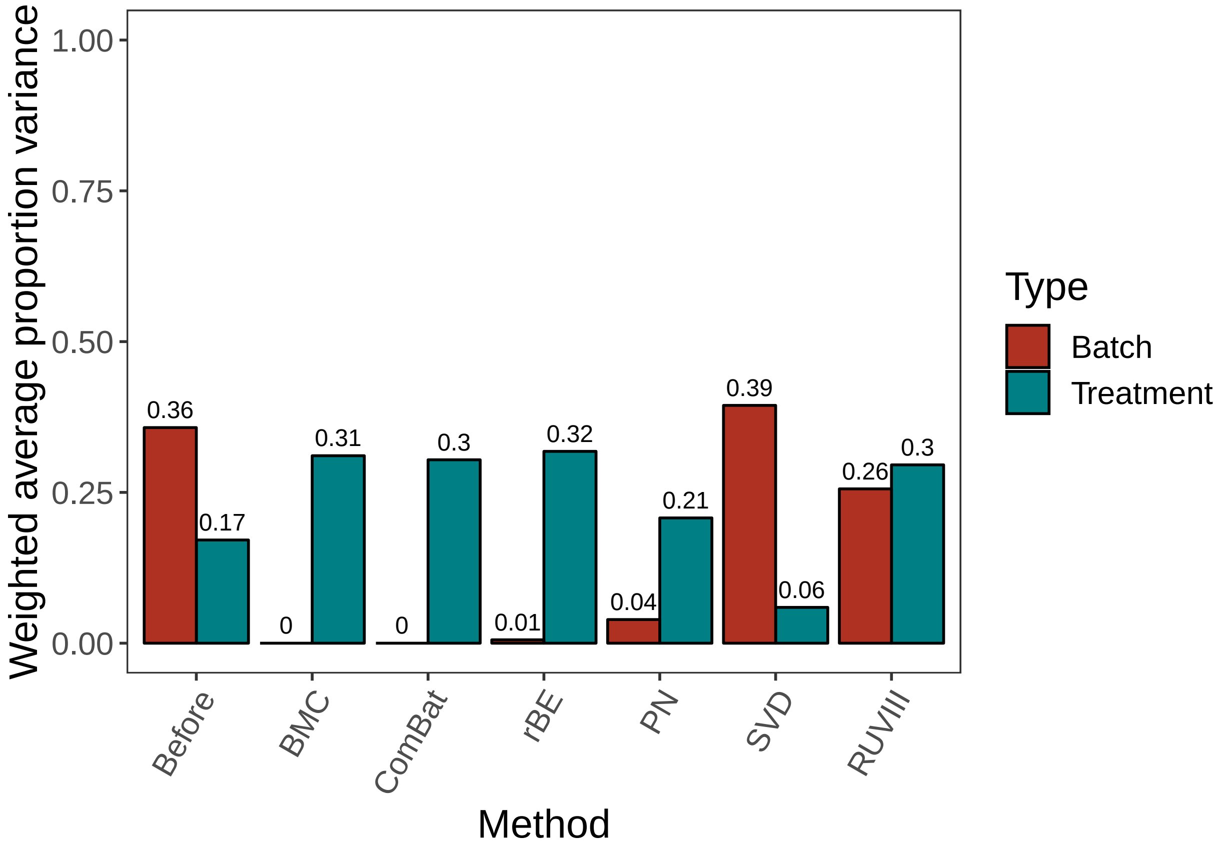
4.2.2 PVCA

It does not work for dataset with only 24 samples, therefore it cannot be applied on Sponge data.

```
# AD data
PVCA.score.ad = data.frame(Interaction = NA, Batch = NA,Treatment = NA,Residuals = NA)

Bat_Int.factors = data.frame(Batch = ad.batch, Treatment = ad.trt)
rownames(Bat_Int.factors) = rownames(ad.tss.clr)
pdata <- AnnotatedDataFrame(Bat_Int.factors)

# before
eset.X.before <- new("ExpressionSet", exprs = t(ad.tss.clr), phenoData = pdata)
```

```
pvcaObj.before <- pvcaBatchAssess(eset.X.before, c('Batch','Treatment'), 0.6)
values.before = pvcaObj.before$dat
PVCA.score.ad[1,] = values.before

# bmc
eset.X.bmc <- new("ExpressionSet", exprs = t(ad.bmc), phenoData = pdata)
pvcaObj.bmc <- pvcaBatchAssess(eset.X.bmc, c('Batch','Treatment'), 0.6)
values.bmc = pvcaObj.bmc$dat
PVCA.score.ad[2,] = values.bmc


# combat

eset.X.combat <- new("ExpressionSet", exprs = t(ad.combat), phenoData = pdata)
pvcaObj.combat <- pvcaBatchAssess(eset.X.combat, c('Batch','Treatment'), 0.6)
values.combat = pvcaObj.combat$dat
PVCA.score.ad[3,] = values.combat

# PN

eset.X.percentile <- new("ExpressionSet", exprs = t(ad.percentile), phenoData = pdata)
pvcaObj.percentile <- pvcaBatchAssess(eset.X.percentile, c('Batch','Treatment'), 0.6)
values.percentile = pvcaObj.percentile$dat
PVCA.score.ad[5,] = values.percentile


# limma

eset.X.limma <- new("ExpressionSet", exprs = t(ad.limma), phenoData = pdata)
pvcaObj.limma <- pvcaBatchAssess(eset.X.limma, c('Batch','Treatment'), 0.6)
values.limma = pvcaObj.limma$dat
PVCA.score.ad[4,] = values.limma



# svd

eset.X.svd <- new("ExpressionSet", exprs = t(ad.svd), phenoData = pdata)
pvcaObj.svd <- pvcaBatchAssess(eset.X.svd, c('Batch','Treatment'), 0.6)
values.svd = pvcaObj.svd$dat
PVCA.score.ad[6,] = values.svd


# RUVIII

eset.X.ruv <- new("ExpressionSet", exprs = t(ad.ruv), phenoData = pdata)
pvcaObj.ruv <- pvcaBatchAssess(eset.X.ruv, c('Batch','Treatment'), 0.6)
values.ruv = pvcaObj.ruv$dat
PVCA.score.ad[7,] = values.ruv

rownames(PVCA.score.ad) =c('Before','BMC','ComBat','rBE','PN','SVD','RUVIII')
```

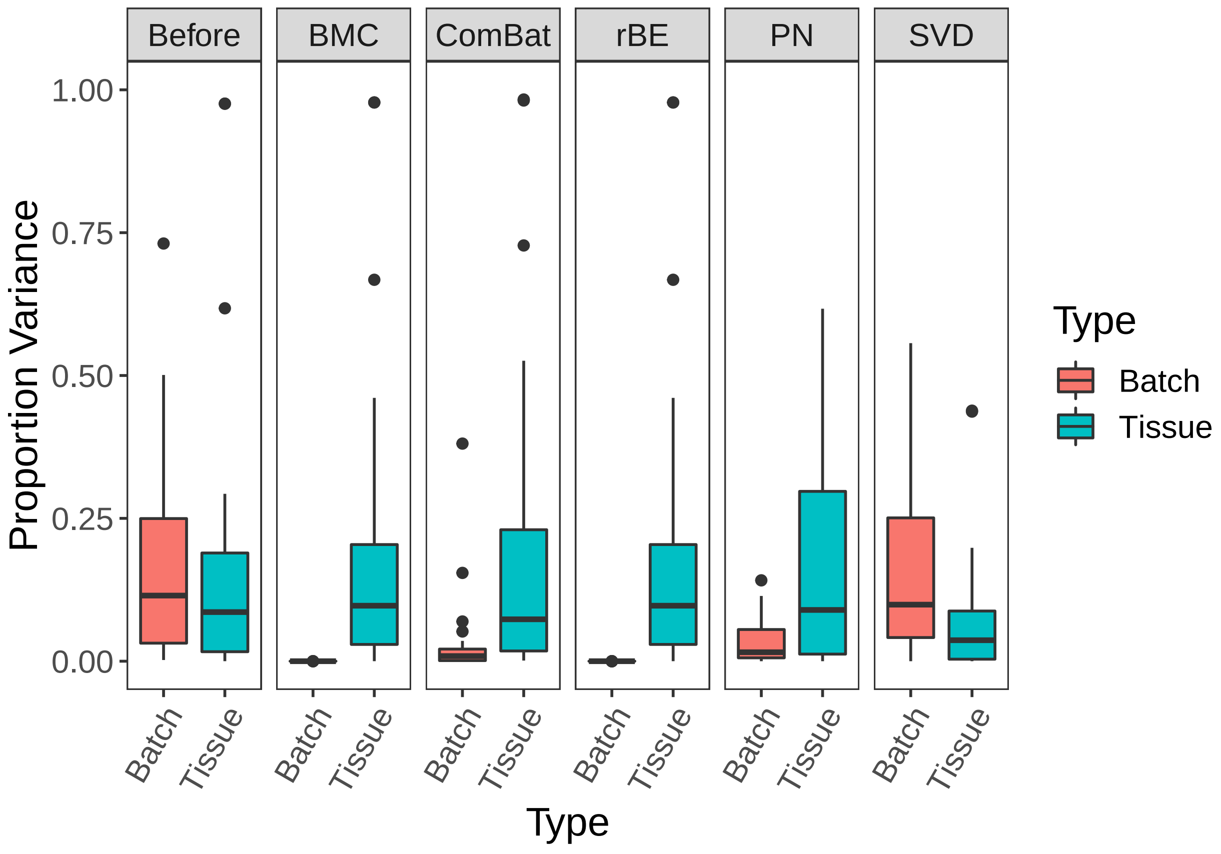
```
#############
pvca.prop.ad.val = c(PVCA.score.ad$Batch,PVCA.score.ad$Treatment)
pvca.prop.ad = data.frame(prop = pvca.prop.ad.val, prop.r = round(pvca.prop.ad.val,2), Method = rep(c('I

pvca.prop.ad$Method =  factor(pvca.prop.ad$Method, levels = unique(pvca.prop.ad$Method))

ggplot(data = pvca.prop.ad, aes(x=Method,y=prop,fill = Type)) + geom_bar(stat="identity",position = 'do
```



4.2.3 Variance partition per variable

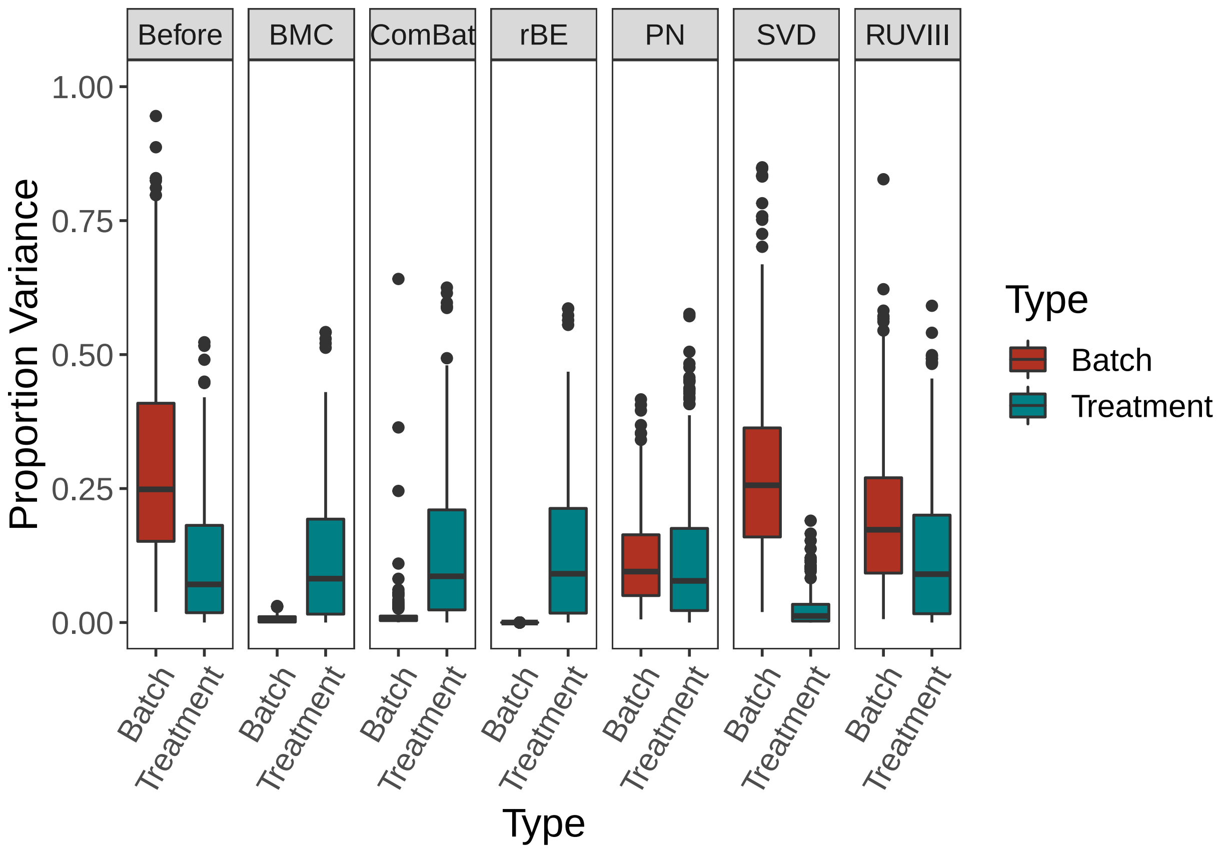
```
## Sponge data
form.sponge <- ~ sponge.trt + sponge.batch
info.sponge = as.data.frame(cbind(rownames(sponge.tss.clr),sponge.trt,sponge.batch))
rownames(info.sponge) = rownames(sponge.tss.clr)

# before
varPart.sponge.before <- fitExtractVarPartModel(t(sponge.tss.clr), form.sponge, info.sponge)

# BMC
varPart.sponge.bmc <- fitExtractVarPartModel(t(sponge.bmc), form.sponge, info.sponge)

# combat
varPart.sponge.combat <- fitExtractVarPartModel(t(sponge.combat), form.sponge, info.sponge)

# removeBatchEffect
varPart.sponge.limma <- fitExtractVarPartModel(t(sponge.limma), form.sponge, info.sponge)
```

```
# percentile normalisation
varPart.sponge.percentile <- fitExtractVarPartModel(t(sponge.percentile), form.sponge, info.sponge)

# svd
varPart.sponge.svd <- fitExtractVarPartModel(t(sponge.svd), form.sponge, info.sponge)


################
#merge them
variance.sponge =  rbind(cbind(variance = varPart.sponge.before$sponge.batch,Type = rep('Batch',24), me
                             cbind(variance = varPart.sponge.before$sponge.trt,Type = rep('Tissue',24),
                             cbind(variance = varPart.sponge.bmc$sponge.batch,Type = rep('Batch',24), m
                             cbind(variance = varPart.sponge.bmc$sponge.trt,Type = rep('Tissue',24), me
                             cbind(variance = varPart.sponge.combat$sponge.batch,Type = rep('Batch',24)
                             cbind(variance = varPart.sponge.combat$sponge.trt,Type = rep('Tissue',24),
                             cbind(variance = varPart.sponge.limma$sponge.batch,Type = rep('Batch',24),
                             cbind(variance = varPart.sponge.limma$sponge.trt,Type = rep('Tissue',24),
                             cbind(variance = varPart.sponge.percentile$sponge.batch,Type = rep('Batch'
                             cbind(variance = varPart.sponge.percentile$sponge.trt,Type = rep('Tissue',
                             cbind(variance = varPart.sponge.svd$sponge.batch,Type = rep('Batch',24), m
                             cbind(variance = varPart.sponge.svd$sponge.trt,Type = rep('Tissue',24), me

variance.sponge = as.data.frame(variance.sponge)
variance.sponge$Type = factor(variance.sponge$Type,levels = unique(variance.sponge$Type))
variance.sponge$method = factor(variance.sponge$method,levels = unique(variance.sponge$method))
variance.sponge$variance = as.numeric(as.character(variance.sponge$variance))

ggplot(variance.sponge, aes(x=Type, y=variance,fill=Type)) + geom_boxplot() + facet_grid(cols = vars(me
```



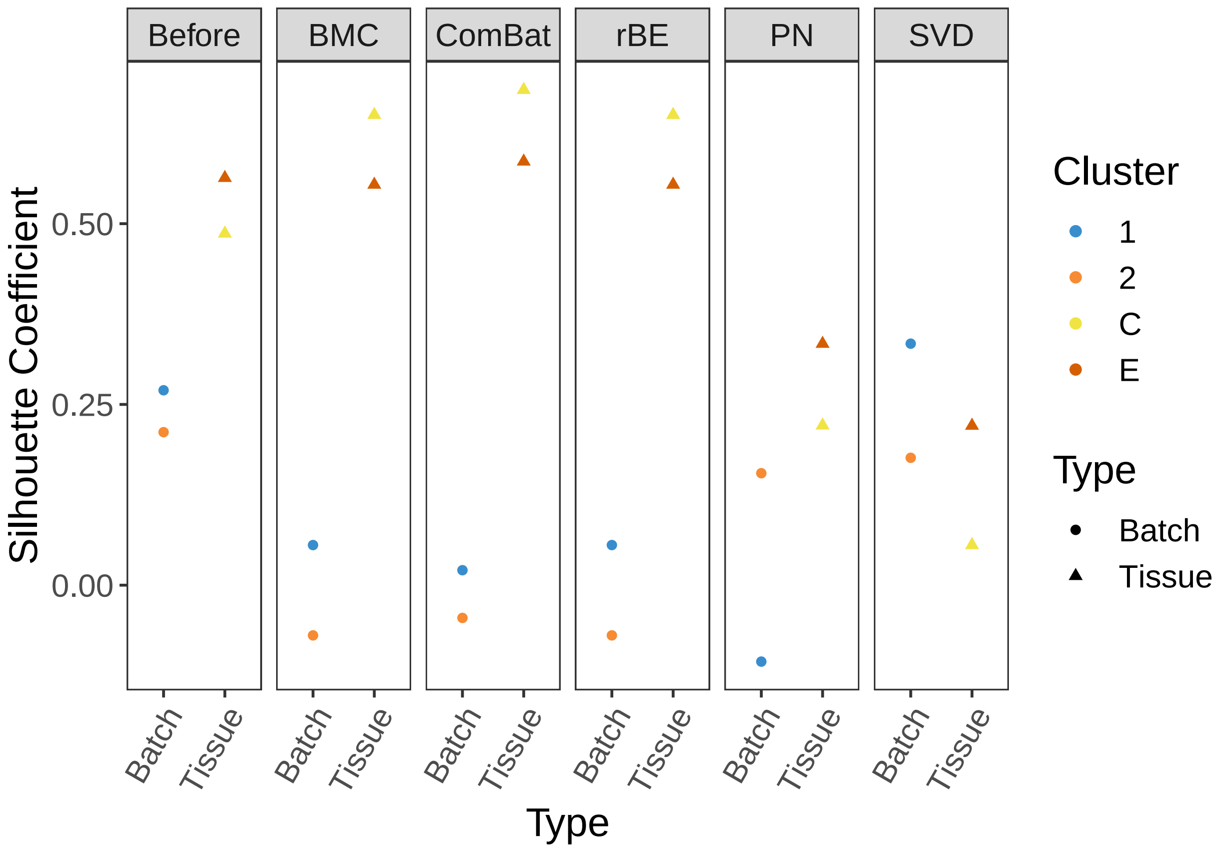
```
##########
# AD data
form.ad <- ~ ad.trt + ad.batch
info.ad = as.data.frame(cbind(rownames(ad.tss.clr),ad.trt,ad.batch))
rownames(info.ad) = rownames(ad.tss.clr)

# before
varPart.ad.before <- fitExtractVarPartModel(t(ad.tss.clr), form.ad, info.ad)

# BMC
varPart.ad.bmc <- fitExtractVarPartModel(t(ad.bmc), form.ad, info.ad)

# combat
varPart.ad.combat <- fitExtractVarPartModel(t(ad.combat), form.ad, info.ad)

# removeBatchEffect
varPart.ad.limma <- fitExtractVarPartModel(t(ad.limma), form.ad, info.ad)

# percentile normalisation
varPart.ad.percentile <- fitExtractVarPartModel(t(ad.percentile), form.ad, info.ad)

# svd
varPart.ad.svd <- fitExtractVarPartModel(t(ad.svd), form.ad, info.ad)

# ruv
varPart.ad.ruv <- fitExtractVarPartModel(t(ad.ruv), form.ad, info.ad)
```

```
#################
#merge them
variance.ad =  rbind(cbind(variance = varPart.ad.before$ad.batch,Type = rep('Batch',231), method = rep(
                             cbind(variance = varPart.ad.before$ad.trt,Type = rep('Treatment',231), met
                             cbind(variance = varPart.ad.bmc$ad.batch,Type = rep('Batch',231), method =
                             cbind(variance = varPart.ad.bmc$ad.trt,Type = rep('Treatment',231), method
                             cbind(variance = varPart.ad.combat$ad.batch,Type = rep('Batch',231), metho
                             cbind(variance = varPart.ad.combat$ad.trt,Type = rep('Treatment',231), met
                             cbind(variance = varPart.ad.limma$ad.batch,Type = rep('Batch',231), method
                             cbind(variance = varPart.ad.limma$ad.trt,Type = rep('Treatment',231), meth
                             cbind(variance = varPart.ad.percentile$ad.batch,Type = rep('Batch',231), m
                             cbind(variance = varPart.ad.percentile$ad.trt,Type = rep('Treatment',231),
                             cbind(variance = varPart.ad.svd$ad.batch,Type = rep('Batch',231), method =
                             cbind(variance = varPart.ad.svd$ad.trt,Type = rep('Treatment',231), method
                             cbind(variance = varPart.ad.ruv$ad.trt,Type = rep('Treatment',231), method

variance.ad = as.data.frame(variance.ad)
variance.ad$Type = factor(variance.ad$Type,levels = unique(variance.ad$Type))
variance.ad$method = factor(variance.ad$method,levels = unique(variance.ad$method))
variance.ad$variance = as.numeric(as.character(variance.ad$variance))

ggplot(variance.ad, aes(x=Type, y=variance,fill=Type)) + geom_boxplot() + facet_grid(cols = vars(method
```

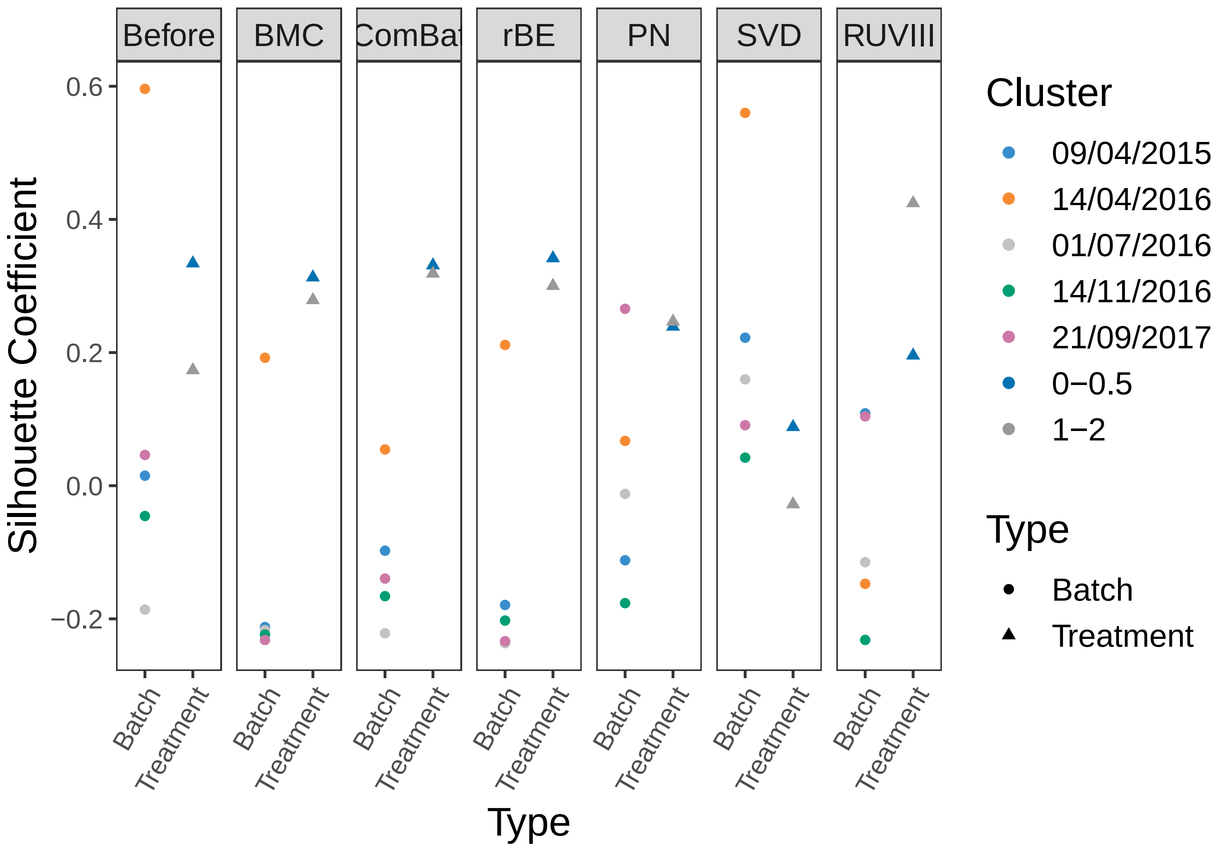


4.2.4 Silhouette coefficient

```
###################
# Sponge data

silh.sponge.before = calc.sil(pca.sponge.before$variates$X,y1 = sponge.batch, y2= sponge.trt, name.y1 =
silh.sponge.bmc = calc.sil(pca.sponge.bmc$variates$X,y1 = sponge.batch, y2= sponge.trt, name.y1 = 'Batcl
silh.sponge.combat = calc.sil(pca.sponge.combat$variates$X,y1 = sponge.batch, y2= sponge.trt, name.y1 =
silh.sponge.limma = calc.sil(pca.sponge.limma$variates$X,y1 = sponge.batch, y2= sponge.trt, name.y1 = 'I
silh.sponge.percentile = calc.sil(pca.sponge.percentile$variates$X,y1 = sponge.batch, y2= sponge.trt, na
silh.sponge.svd = calc.sil(pca.sponge.svd$variates$X,y1 = sponge.batch, y2= sponge.trt, name.y1 = 'Batcl


data.plot.sponge = rbind(silh.sponge.before, silh.sponge.bmc, silh.sponge.combat, silh.sponge.limma, sil
data.plot.sponge$method = c(rep('Before', nrow(silh.sponge.before)),
                            rep('BMC', nrow(silh.sponge.bmc)),
                            rep('ComBat', nrow(silh.sponge.combat)),
                            rep('rBE', nrow(silh.sponge.limma)),
                            rep('PN', nrow(silh.sponge.percentile)),
                            rep('SVD', nrow(silh.sponge.svd))
)
data.plot.sponge$method = factor(data.plot.sponge$method,levels = unique(data.plot.sponge$method))
data.plot.sponge$Cluster =  factor(data.plot.sponge$Cluster, levels = unique(data.plot.sponge$Cluster))
data.plot.sponge$Type =  factor(data.plot.sponge$Type, levels = unique(data.plot.sponge$Type))


ggplot(data.plot.sponge, aes(x=Type, y=silh.coeff, color = Cluster, shape = Type)) + geom_point() + fac
```



```
############
# AD data
silh.ad.before = calc.sil(pca.ad.before$variates$X,y1 = ad.batch, y2= ad.trt, name.y1 = 'Batch',name.y2
silh.ad.bmc = calc.sil(pca.ad.bmc$variates$X,y1 = ad.batch, y2= ad.trt, name.y1 = 'Batch',name.y2 = 'Tr
silh.ad.combat = calc.sil(pca.ad.combat$variates$X,y1 = ad.batch, y2= ad.trt, name.y1 = 'Batch',name.y2
silh.ad.limma = calc.sil(pca.ad.limma$variates$X,y1 = ad.batch, y2= ad.trt, name.y1 = 'Batch',name.y2 =
silh.ad.percentile = calc.sil(pca.ad.percentile$variates$X,y1 = ad.batch, y2= ad.trt, name.y1 = 'Batch'
silh.ad.svd = calc.sil(pca.ad.svd$variates$X,y1 = ad.batch, y2= ad.trt, name.y1 = 'Batch',name.y2 = 'Tr
silh.ad.ruv = calc.sil(pca.ad.ruv$variates$X,y1 = ad.batch, y2= ad.trt, name.y1 = 'Batch',name.y2 = 'Tr


data.plot.ad = rbind(silh.ad.before, silh.ad.bmc, silh.ad.combat, silh.ad.limma, silh.ad.percentile, si
data.plot.ad$method = c(rep('Before', nrow(silh.ad.before)),
                            rep('BMC', nrow(silh.ad.bmc)),
                            rep('ComBat', nrow(silh.ad.combat)),
                            rep('rBE', nrow(silh.ad.limma)),
                            rep('PN', nrow(silh.ad.percentile)),
                            rep('SVD', nrow(silh.ad.svd)),
                        rep('RUVIII', nrow(silh.ad.ruv))
)
data.plot.ad$method = factor(data.plot.ad$method, levels = unique(data.plot.ad$method))
data.plot.ad$Cluster =  factor(data.plot.ad$Cluster, levels = unique(data.plot.ad$Cluster))
data.plot.ad$Type =  factor(data.plot.ad$Type, levels = unique(data.plot.ad$Type))

ggplot(data.plot.ad, aes(x=Type, y=silh.coeff, color = Cluster, shape = Type)) + geom_point() + facet_g
```


Before
BMC
ComBat
rBE
PN
SVD
RUVIII
Silhouette Coefficient
0.6
0.4
0.2
0.0
−0.2
Batch
Treatment
Type
Cluster
09/04/2015
14/04/2016
01/07/2016
14/11/2016
21/09/2017
0−0.5
1−2
Type
Batch
Treatment

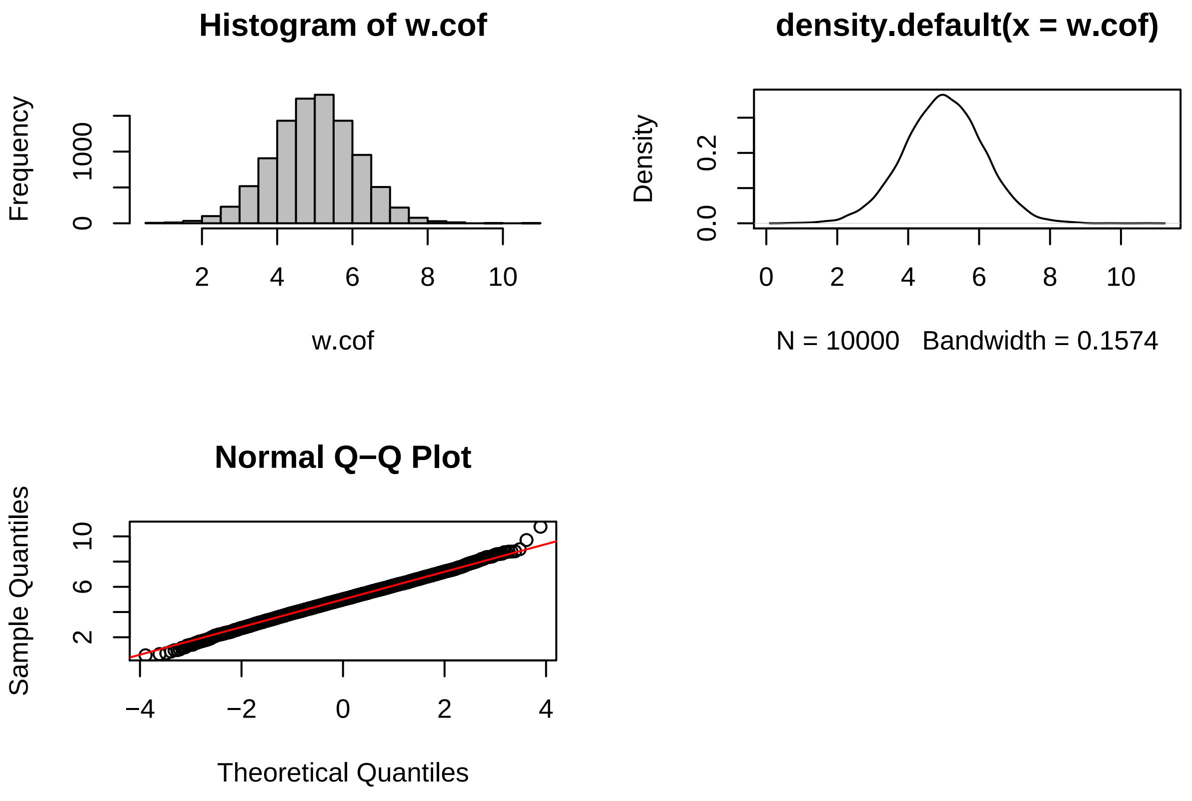
Chapter 5

Simulations of systematic and non-systematic batch effects

5.1 Mean=5,unequal variance

- $\beta_j \sim N(5, 1^2)$ for $j = 1, ..., p$ OTUs;
- $\sigma_j \sim N(0, 2^2)$ for $j = 1, ..., p$ OTUs;
- $\beta_{ij} \sim N(\beta_j, \sigma_j^2)$ for $i = 1, ..., n$ samples.

```
## Create the simulated data
m = 50
n = 10000
nc = 1000 #negative controls without treatment effects
p = 1
k = 1
ctl = rep(FALSE, n)
ctl[1:nc] = TRUE
# treatment effect
X = matrix(c(rep(0,floor(m/2)), rep(1,ceiling(m/2))), m, p)
beta = matrix(rnorm(p*n,5,1), p, n) #treatment coefficients
beta[,ctl] = 0
# batch effect
W = as.matrix(rep(0,m),m,k)
W[c(1:12,38:50),1] =  1
alpha = matrix(rnorm(k*n,5,1),k,n)
Y_alpha = sapply(alpha, function(alpha){rnorm(m, mean =  alpha, abs(rnorm(1, mean = 0, sd = 2)))})
YY_alpha = apply(Y_alpha,2,function(x){x*W})

epsilon = matrix(rnorm(m*n,0,1),m,n)
Y = X%*%beta + YY_alpha + epsilon


# estimate batch coefficient for each OTU
w.cof = c()
for(i in 1:ncol(Y)){
  res = lm(Y[,i] ~ X + W)
  sum.res = summary(res)
  w.cof[i] = sum.res$coefficients[3,1]
}

par(mfrow=c(2,2))
hist(w.cof,col = 'gray')
plot(density(w.cof))
qqnorm(w.cof)
qqline(w.cof, col='red')
par(mfrow=c(1,1))
```

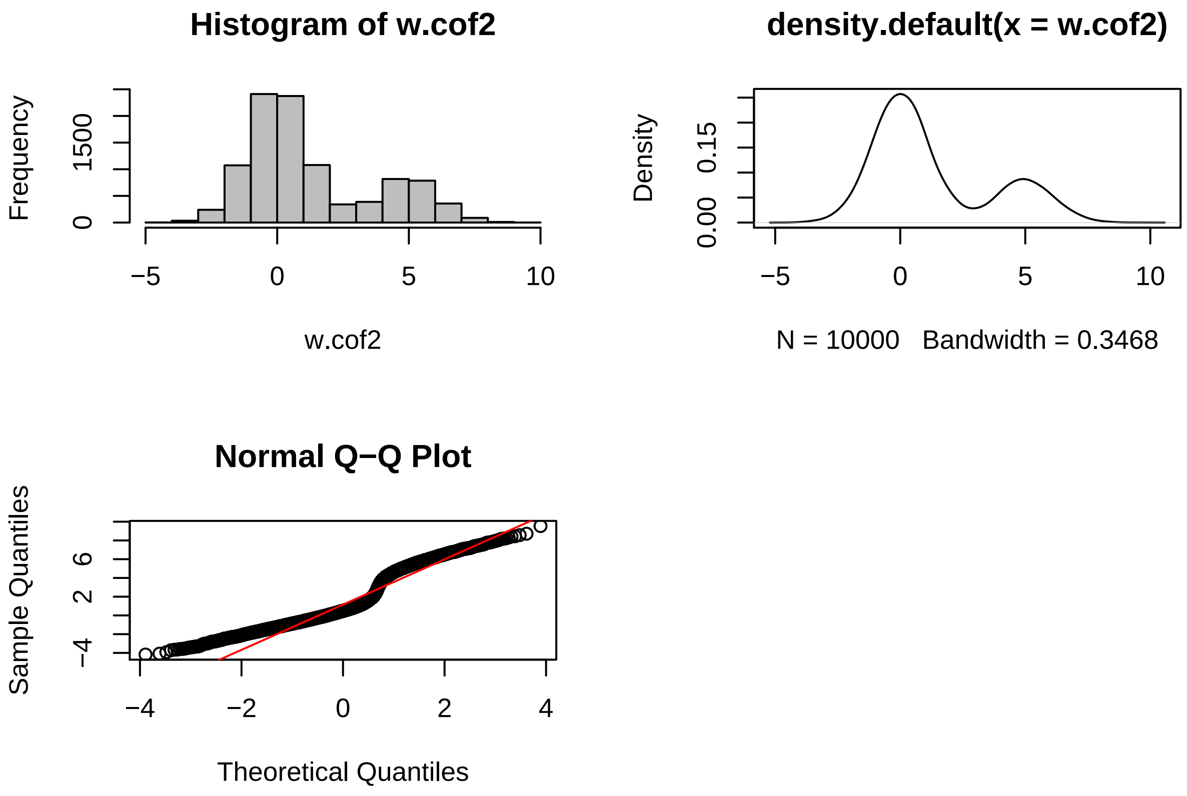


5.2 Mean=0&5,unequal variance

- $\beta_t \sim N(0, 1^2)$ and $\beta_k \sim N(5, 1^2)$ for $t = 1, ..., T$ OTUs, $k = 1, ..., K$ OTUs and $T = \frac{3}{4}p$, $K = \frac{1}{4}p$;
- $\sigma_j \sim N(0, 2^2)$ for $j = 1, ..., p$ OTUs;
- $\beta_{ij} \sim N(\beta_j, \sigma_j^2)$ for $i = 1, ..., n$ samples.

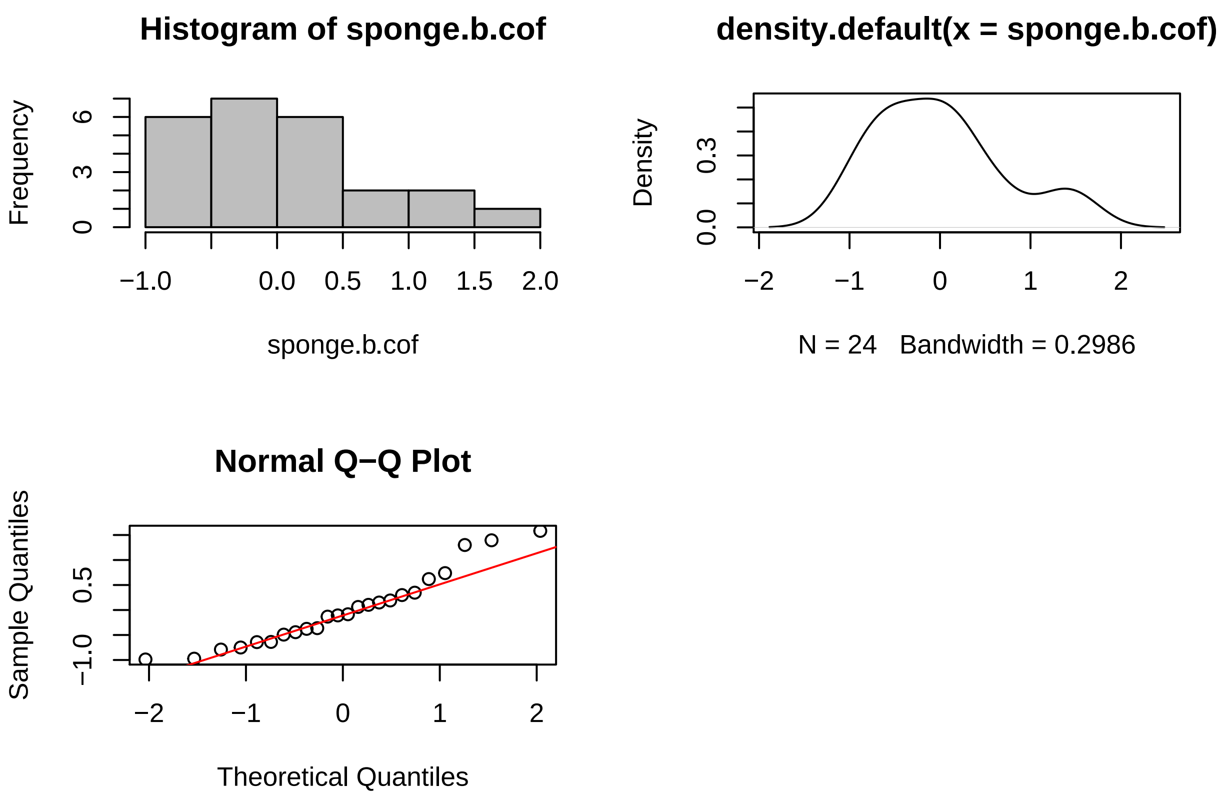
```
## Create the simulated data
m = 50
n = 10000
nc = 1000 #negative controls without treatment effects
p = 1
k = 1
ctl = rep(FALSE, n)
ctl[1:nc] = TRUE
```

```
# treatment effect
X = matrix(c(rep(0,floor(m/2)), rep(1,ceiling(m/2))), m, p)
beta = matrix(rnorm(p*n,5,1), p, n) #treatment coefficients
beta[,ctl] = 0
# batch effect
W = as.matrix(rep(0,m),m,k)
W[c(1:12,38:50),1] =  1
alpha2 = matrix(sample(c(rnorm(k*(3*n/4),0,1),rnorm(k*(n/4),5,1)),n),k,n)
Y_alpha2 = sapply(alpha2, function(alpha){rnorm(m, mean =  alpha, sd = abs(rnorm(1, mean = 0, sd = 2))
YY_alpha2 = apply(Y_alpha2,2,function(x){x*W})

epsilon = matrix(rnorm(m*n,0,1),m,n)
Y2 = X%*%beta + YY_alpha2 + epsilon





w.cof2 = c()
for(i in 1:ncol(Y2)){
  res = lm(Y2[,i] ~ X + W)
  sum.res = summary(res)
  w.cof2[i] = sum.res$coefficients[3,1]
}

par(mfrow=c(2,2))
hist(w.cof2,col = 'gray')
plot(density(w.cof2))
qqnorm(w.cof2)
qqline(w.cof2, col='red')
par(mfrow=c(1,1))
```



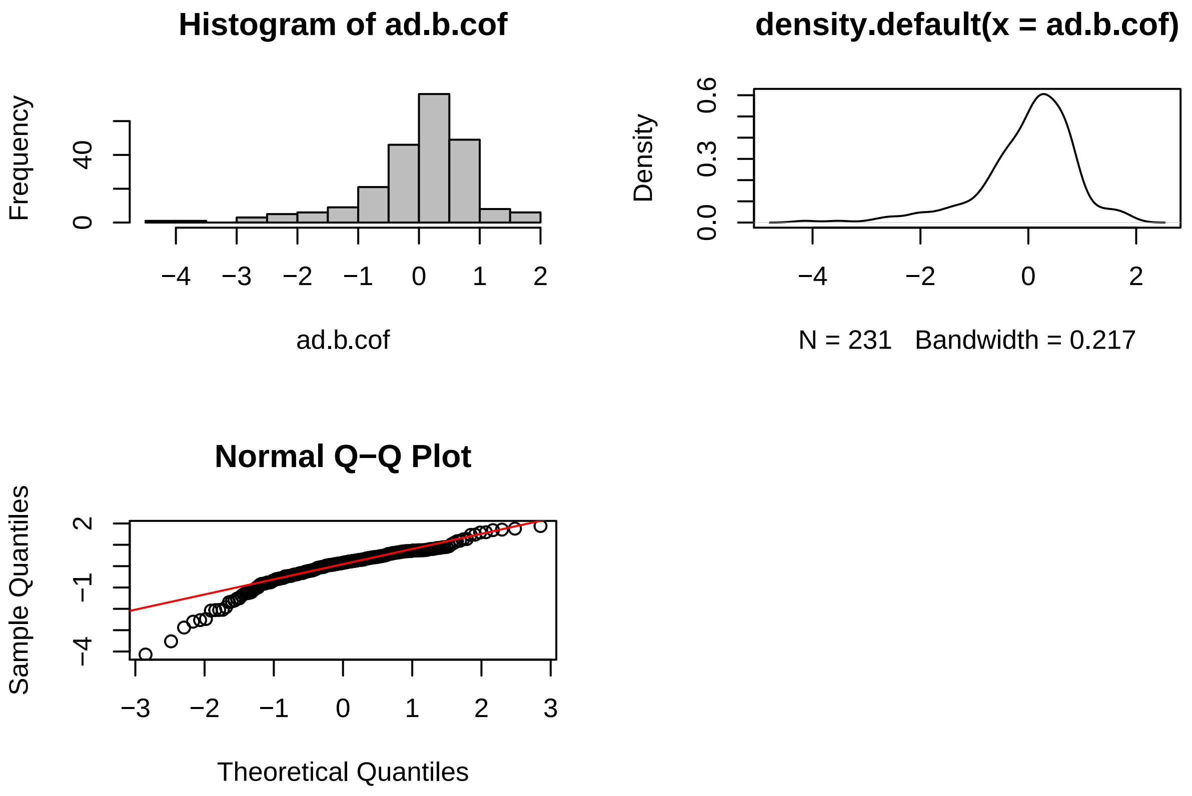
5.3 Sponge data

```
sponge.b.cof = c()
for(i in 1:ncol(sponge.tss.clr)){
  res = lm(sponge.tss.clr[,i] ~ sponge.trt + sponge.batch)
  sum.res = summary(res)
  sponge.b.cof[i] = sum.res$coefficients[3,1]
}

par(mfrow=c(2,2))
hist(sponge.b.cof,col = 'gray')
plot(density(sponge.b.cof))
qqnorm(sponge.b.cof)
qqline(sponge.b.cof, col='red')
par(mfrow=c(1,1))
```



5.4 AD data

```
ad.b.cof = c()
ad.batch.relevel = relevel(ad.batch, '01/07/2016')
for(i in 1:ncol(ad.tss.clr)){
  res = lm(ad.tss.clr[,i] ~ ad.trt + ad.batch.relevel)
  sum.res = summary(res)
  ad.b.cof[i] = sum.res$coefficients[4,1]
}

par(mfrow=c(2,2))
hist(ad.b.cof,col = 'gray')
plot(density(ad.b.cof))
qqnorm(ad.b.cof)
qqline(ad.b.cof, col='red')
par(mfrow=c(1,1))
```


Histogram of ad.b.cof
Frequency
ad.b.cof
density.default(x = ad.b.cof)
Density
N = 231   Bandwidth = 0.217
Normal Q−Q Plot
Sample Quantiles
Theoretical Quantiles

Bibliography

Arumugam, M., Raes, J., Pelletier, E., Le Paslier, D., Yamada, T., Mende, D. R., Fernandes, G. R., Tap, J., Bruls, T., Batto, J.-M., et al. (2011). Enterotypes of the human gut microbiome. *nature*, 473(7346):174.

Chapleur, O., Madigou, C., Civade, R., Rodolphe, Y., Mazéas, L., and Bouchez, T. (2016). Increasing concentrations of phenol progressively affect anaerobic digestion of cellulose and associated microbial communities. *Biodegradation*, 27(1):15–27.

Kong, G., Lê Cao, K.-A., Judd, L. M., Li, S., Renoir, T., and Hannan, A. J. (2018). Microbiome profiling reveals gut dysbiosis in a transgenic mouse model of Huntington's disease. *Neurobiology of Disease*.

Lin, Y., Ghazanfar, S., Wang, K., Gagnon-Bartsch, J. A., Lo, K. K., Su, X., Han, Z.-G., Ormerod, J. T., Speed, T. P., Yang, P., et al. (2018). scMerge: Integration of multiple single-cell transcriptomics datasets leveraging stable expression and pseudo-replication. *BioRxiv*, page 393280.

Sacristán-Soriano, O., Banaigs, B., Casamayor, E. O., and Becerro, M. A. (2011). Exploring the links between natural products and bacterial assemblages in the sponge Aplysina aerophoba. *Applied and Environmental Microbiology*, 77(3):862–870.