Statistical Methods: Dimensionality Reduction, Clustering and Regression

Bernd Klaus[1]

European Molecular Biology Laboratory (EMBL),
Heidelberg, Germany
[1]bernd.klaus@embl.de

April 4, 2016

Contents

1 Required packages and other preparations

```
set.seed(777)
library(TeachingDemos)
library(geneplotter)
library(ggplot2)
library(plyr)
library(LSD)
library(boot)
library(ade4)
library(DESeq2)
library(vsn)
library(gplots)
library(RColorBrewer)
library(psych)
library(car)
library(matrixStats)
library(MASS)
library(vegan)
library(locfit)
library(stringr)
library(sva)
library(limma)
library(corpcor)


ggplotRegression <- function (fit) {

ggplot(fit$model, aes_string(x = names(fit$model)[2], y = names(fit$model)[1])) +
  geom_point() +
  stat_smooth(method = "lm", col = "coral2") +
  labs(title = paste("Adj R2 = ",signif(summary(fit)$adj.r.squared, 5),
                     "Intercept =",signif(fit$coef[[1]],5 ),
                     " Slope =",signif(fit$coef[[2]], 5),
                     " P =",signif(summary(fit)$coef[2,4], 5)))
}
```

2 Contents of the lab

In this lab important statistical methods for bioinformatics will be discussed, namely clustering, regression analysis and PCA. We will apply these techniques in the context of the analysis of RNA–Seq data.

3 Dimensionality reduction and clustering of high–dimensional data sets

3.1 Simulated RNA-Seq data

In order to introduce dimensionality reduction methods, we will use simulated RNA–Seq data. We simulate two groups with 4 samples in each group. On top of this, each group contains 2 batches: males and females. Batch effects are an important problem in biology and dimension–reduction and clustering methods are common tools to identify batches. We use the function `makeExampleDESeqDataSet` from the package *DESeq2* to produce the simulated data and then add the sex batches.

```
dds <- makeExampleDESeqDataSet(m=8, betaSD = 2)

## add sex as a batch effect, there are two males and two females in each group
colData(dds)$sex <- factor(rep(c("m", "f"), 4))

## modify counts to add a batch effect, we add normally distributed random noise
## with mean 2 to randomly selected genes of male samples and then round the result
cts <- counts(dds)
ranGenes <- floor(runif(300)*1000)

for(i in ranGenes){
  cts[i, colData(dds)$sex == "m"] <- as.integer(cts[i, colData(dds)$sex == "m"]
                                                + round(rnorm(1,4, sd = 1)))
  }
counts(dds) <- cts

counts(dds)[5:10,]
```

	sample1	sample2	sample3	sample4	sample5	sample6	sample7	sample8
gene5	117	123	243	180	189	147	152	119
gene6	50	33	20	26	43	51	49	91
gene7	21	0	8	30	284	147	296	209
gene8	52	86	62	127	403	539	255	256
gene9	15	16	3	0	1	1	2	0
gene10	8	23	22	3	3	3	3	3

As usual for Bioconductor packages, the simulated counts are saved in a matrix, where the columns correspond to the samples and the rows to the genes.

3.2 Mean–variance relationship for count data

Count data are heteroscesdastic, that is, their variance depends on the mean. For the data set at hand, this means that the higher the gene expression of the gene, the higher is its variance. We can see this relationship by producing a plot of the ranked mean bins against the estimated standard deviation in each bin. Means and standard deviations are estimated per gene in this case. We use the function `meanSdPlot` from the *VSN* package to achieve this.

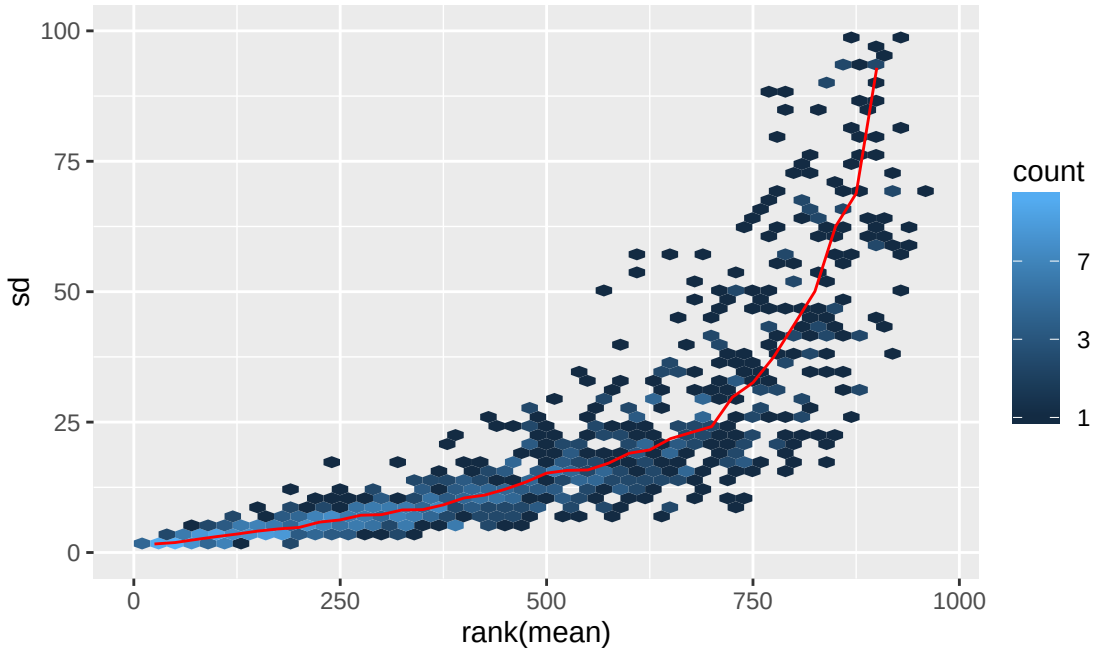
```
pl <- meanSdPlot(counts(dds))
```

```
pl$gg + ylim(0,100)
```

```
  Warning:  Removed 93 rows containing non-finite values (stat_binhex).
```

```
  Warning:  Removed 3 rows containing missing values (geom_hex).
```

```
  Warning:  Removed 3 rows containing missing values (geom_path).
```



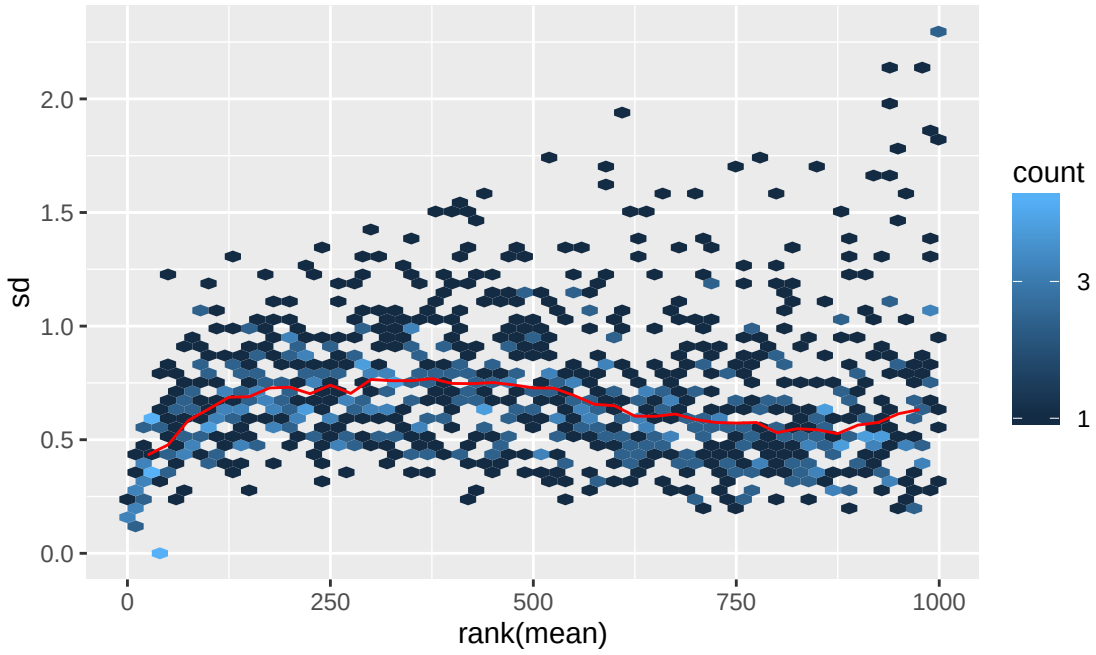
Homoscedastic data would produce a straight red line in this plot. *DESeq2* offers the regularized–logarithm transformation, or rlog for short in order to alleviate this problem.

For genes with high counts, the rlog transformation differs not much from an ordinary log2 transformation.

For genes with lower counts, however, the values are shrunken towards the genes' averages across all samples. Using an empirical Bayesian prior in the form of a ridge penalty, this is done such that the rlog–transformed data are approximately homoscedastic.

We perform an rlog transformation of the simulated data and then produce a mean–sd plot again to check the variance

```
rldSim <- assay(rlogTransformation(dds, blind=TRUE))
meanSdPlot(rldSim)
```



We can see that the data is now approximately homoscedastic and can proceed to the dimension reduction and the associated plots.

3.3 PCA for the simulated RNA–Seq data

A PCA represents the individual samples in our RNA–Seq data data set by two or more "artificial" variables, which are called Principal Components (PCs). The components are linear combinations of the genes and the linear combination is chosen in such a way that the euclidean distance between the individuals using centered and standardized variables is minimized.

If $X = (x_1, \ldots, x_p)$ denotes the original data matrix (i.e. our rlog–transformed data), then the columns of the transformed data matrix, also called the "principal components" or "(PC) scores" is simply given by:

$$\mathrm{PC}_i = Xa_i, \quad i = 1, \ldots, p$$

Where a_i is a vector of length p. The a_i vectors are commonly called "loadings" or "principal axes". These loadings are usually the eigenvectors of the correlation matrix of the data.

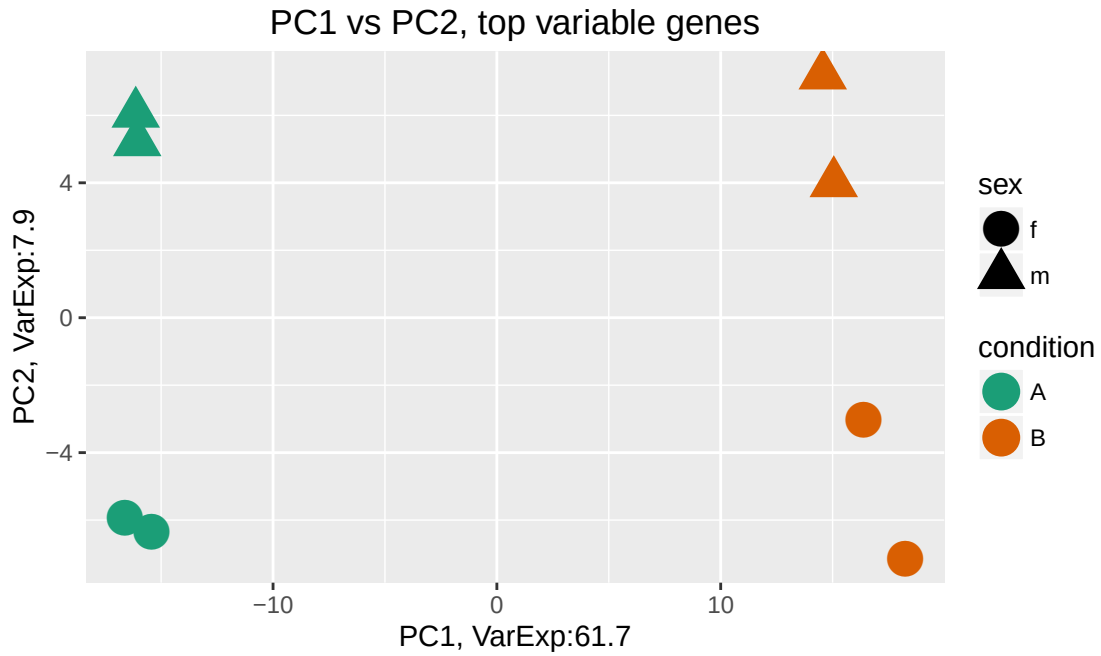
It can be shown that choosing the loadings in this way leads to a transformation that generates principal components that have maximal variance and preserve as good as possible the distances between the original observations.

If we have many variables (genes) as in our case ($p = 1000$), it is advisable to perform a selection of variables since this leads to more stable PCA results.

Here, we simply use the 500 most variable genes.

```
ntop = 500

pvars <- rowVars(rldSim)
select <- order(pvars, decreasing = TRUE)[seq_len(min(ntop,
                                                      length(pvars)))]

PCA <- prcomp(t(rldSim)[, select], scale = F)
```

```
percentVar <- round(100*PCA$sdev^2/sum(PCA$sdev^2),1)


dataGG = data.frame(PC1 = PCA$x[,1], PC2 = PCA$x[,2],
                    PC3 = PCA$x[,3], PC4 = PCA$x[,4],
                    sex = colData(dds)$sex,
                    condition = colData(dds)$condition)

(qplot(PC1, PC2, data = dataGG, color =  condition, shape = sex,
       main = "PC1 vs PC2, top variable genes", size = I(6))
+ labs(x = paste0("PC1, VarExp:", round(percentVar[1],4)),
       y = paste0("PC2, VarExp:", round(percentVar[2],4)))
+ scale_colour_brewer(type="qual", palette=2)
)
```



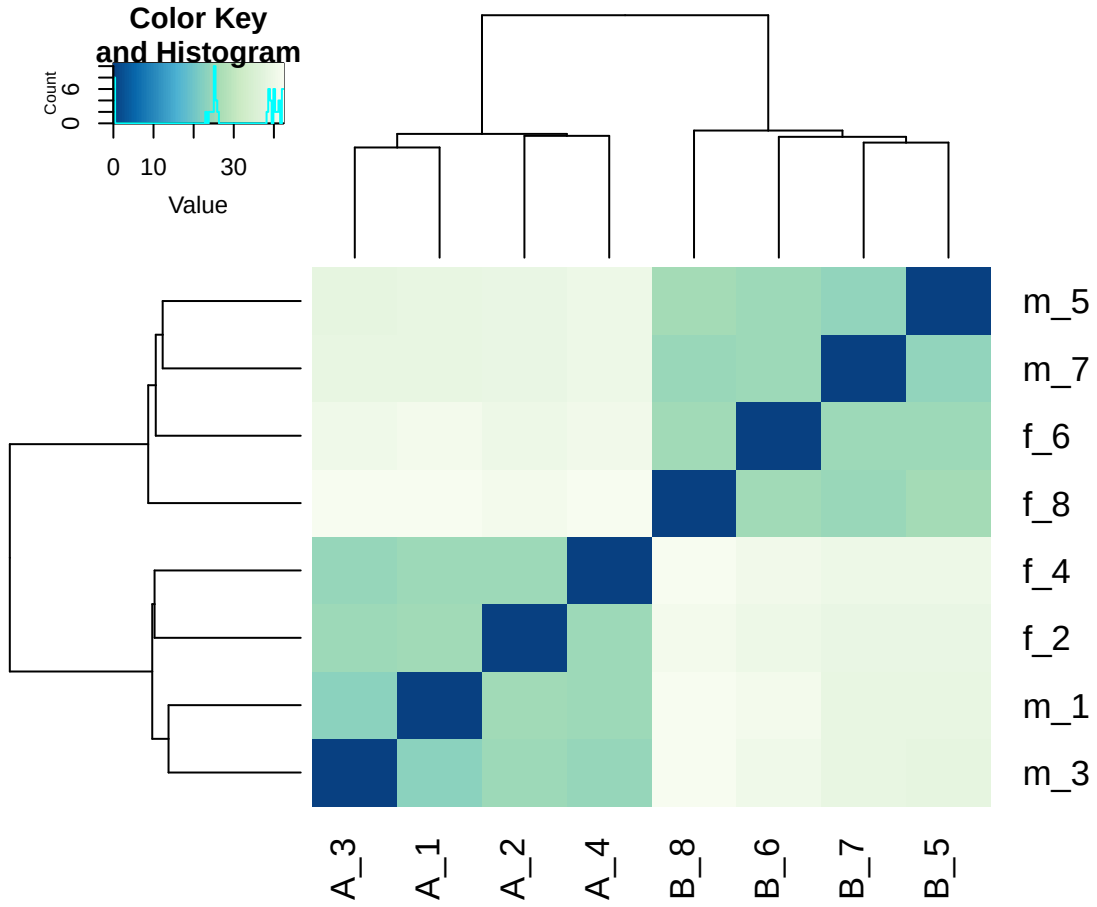
We can see that the first PC separates the two experimental conditions, while the second one splits the samples by sex. This shows that the PCA can be used to visualize and detect batch effects. Note that the principal components are commonly ordered according to "informativeness", i.e. the percentage of the total variance in the data set they explain. So a separation according to the first PC is stronger than a separation according to the second one. In the simulated data, the experimental group is more important the sample batch (= sex).

3.4 Heatmaps and hierarchical clustering

Another very common visualization technique is a heatmap. Analogous to a PCA we can visualize the pairwise distances between the samples using a heatmap, i.e. a false color representation of the distance between two samples. Here we use the euclidean distance of all the genes per sample. This can be conveniently computed using the function `dist`.

This heatmap is then ordered via hierarchical clustering. Hierarchical clustering starts with as many clusters as there are samples and successively merges samples that are close to each other. This merging process is commonly visualized as a tree like graphic called a dendogramm.

```
dists <- as.matrix(dist(t(rldSim)))

rownames(dists) <-  paste0(colData(dds)$sex,
                           "_", str_sub(rownames(colData(dds)), 7))
colnames(dists) <- paste0(colData(dds)$condition,
                           "_", str_sub(rownames(colData(dds)), 7))

hmcol <- colorRampPalette(brewer.pal(9, "GnBu"))(100)
heatmap.2(dists, trace="none", col = rev(hmcol))
```



Again we see that samples cluster mainly by experimental condition, but also by batch.

4 Multidimensional scaling (MDS)

Multidimensional scaling is a technique alternative to PCA. MDS takes a set of dissimilarities and returns a set of points such that the distances between the points are approximately equal to the dissimilarities.

We can think of "squeezing" a high-dimensional point cloud into a small number of dimensions (2, perhaps 3) while preserving as well as possible the inter–point distances.

Thus, it can be seen as a generalization of PCA in some sense, since it allows to represent any distance, not only the euclidean one.

4.1 Classical MDS for single cell RNA–Seq data

Here we get the cell–cycle corrected single cell RNA–seq data from Buettner et. al.. The authors use a dataset studying the differentiation of T-cells. The authors found that the cell cycle had a profound impact on the gene expression for this data set and developed a factor analysis–type method (a regression–type model where the coefficients are random and follow a distribution) to remove the cell cycle effects. The data are given as log–transformed counts.

We first import the data as found in the supplement of the article and the compute the euclidean distances between the single cells.

Then, a classic MDS (with 2 dimensions in our case) can be produced from this distance matrix. We compute the distance matrix on the centered and scaled gene expression measurements per cell as to make them comparable across cells.

This gives us a set of points that we can then plot.

```
scData <- t(read.csv("http://www-huber.embl.de/users/klaus/nbt.3102-S7.csv",
            row.names  = 1))


scData[1:10,1:10]
```

```
           Cell 1   Cell 2   Cell 3   Cell 4   Cell 5   Cell 6   Cell 7   Cell 8
   Gnai3  3.23220  1.98320  2.24820  3.97790  3.72040  1.70730  1.91810  2.82960
   Cdc45  3.19810  1.17300  3.17050  1.15060  2.57390  1.63370  1.97650 -0.43942
   Narf   0.29411  0.49389  1.62790  0.31071  1.47400  1.02340  0.76475  1.82220
   Klf6   1.73430  3.85050  1.63060  2.46550  4.11580  2.58050  3.81610  3.07760
   Scmh1  0.26642  0.28471 -0.06410  0.10635  1.80160  0.21496  0.26583 -0.13978
   Wnt3   0.53103  0.27724  0.61398  0.60706  3.55570  0.17696  0.07049 -0.00194
   Fert2 -0.00132 -0.00132 -0.00132 -0.00132 -0.00132 -0.00132 -0.00132 -0.00132
   Xpo6   2.02860  2.68080  2.36290  2.34560  0.24584  2.09130  3.34050  2.84180
   Tfe3   0.03441  3.25010 -0.15768  0.58509  0.25423  1.36410  0.68961  1.01050
   Brat1  0.00245  0.57451  0.27495  0.65898  2.12070  1.06870  0.20165  2.78830
           Cell 9  Cell 10
   Gnai3  3.06210  0.45000
   Cdc45  0.27769  2.48950
   Narf   2.08500  2.71180
   Klf6   3.62690  2.14880
   Scmh1  0.04222  1.97900
   Wnt3   0.47497 -0.00194
   Fert2 -0.00132 -0.00132
   Xpo6   1.24430  0.53199
   Tfe3   3.30500 -0.11513
   Brat1  0.49503 -0.02649
```

```
distEucSC <- dist(t(scale(scData)))

scalingEuSC <- as.data.frame(cmdscale(distEucSC, k = 2))
names(scalingEuSC) <- c("MDS_Dimension_1", "MDS_Dimension_2")
head(scalingEuSC)
```

```
          MDS_Dimension_1 MDS_Dimension_2
   Cell 1           -2.54          -14.93
   Cell 2           -3.94           -4.70
   Cell 3           18.71            3.95
   Cell 4          -23.05            2.52
   Cell 5          -13.36           13.91
```

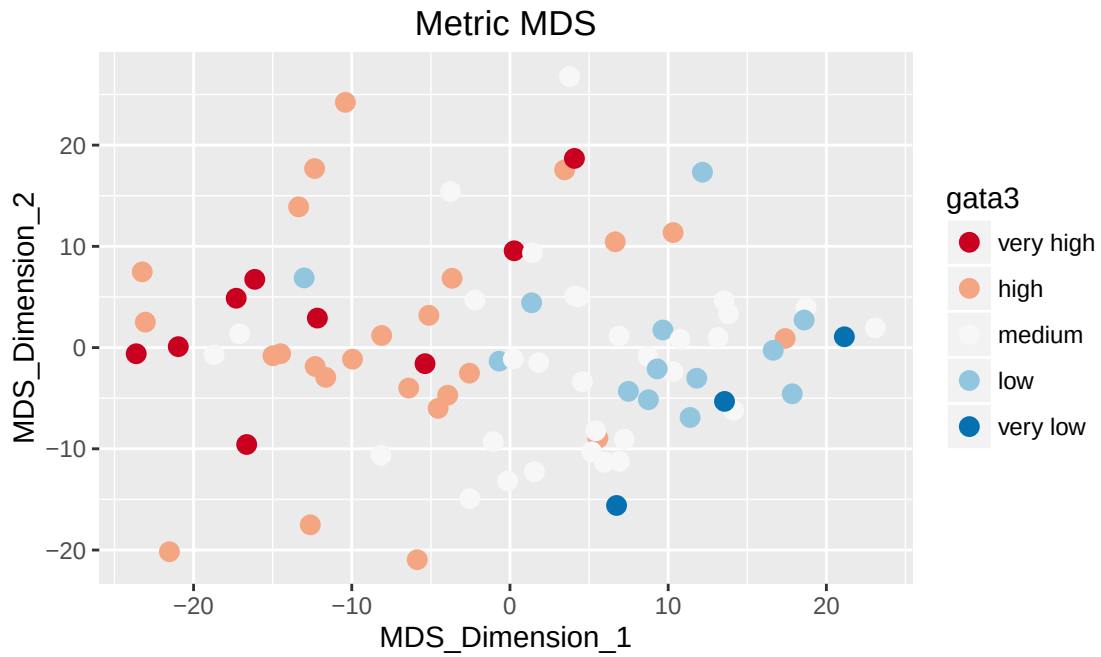
```
   Cell 6           23.09            1.96
```

In the non–linear PCA method used in the original paper a clear two groups separation appears with regard to gata3, a factor that is important in T_H2 differentiation. We color the cells in the MDS plot according to gata3 expression and observe that the first MDS dimension separates two cell populations with high and low gata3 expression. This is consistent with the results reported in the original publication, although a different method , namely classic MDS, to produce a low–dimensional representation of the data was used.

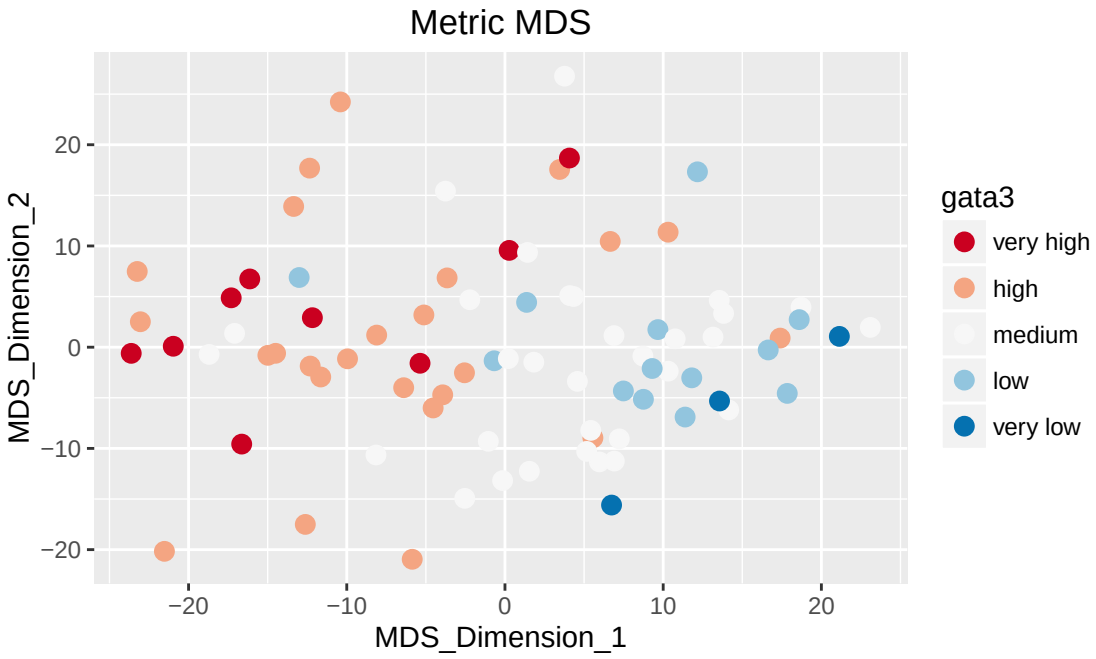
```
rownames(scData)[727]
```

```
   [1] "Gata3"
```

```
gata3 <- cut(scale(scData[727,]), 5,
             labels = c("very low", "low", "medium", "high", 'very high'))

# reverse label ordering
gata3 <- factor(gata3, rev(levels(gata3)))

scPlot <- qplot(MDS_Dimension_1, MDS_Dimension_2, data = scalingEuSC,
      main = "Metric MDS",
      color = gata3, size = I(3))  + scale_color_brewer(palette = "RdBu")


scPlot
```




```
# + scale_color_gradient2(mid = "#ffffb3")
# + scale_colour_manual(values = rev(brewer.pal(5, "RdBu"))))
gata3Colours <- invisible(print(scPlot))$data[[1]]$colour
```



4.2 Assessing the quality of the scaling and non–metric MDS

A measure of how good the scaling is given by a comparison of the original distances to the approximated distances. This comparison is commonly done using a "stress–function" that summarizes the difference between the low dimensional distances and the original ones. Stress functions can also be used to compute a MDS. This is for example done in so called non–metric MDS, where for example the stress function is defined in such a way that small distances (Sammon scaling) or a general monotonic function of the observed distances is used as a target (Kruskal's non–metric scaling).

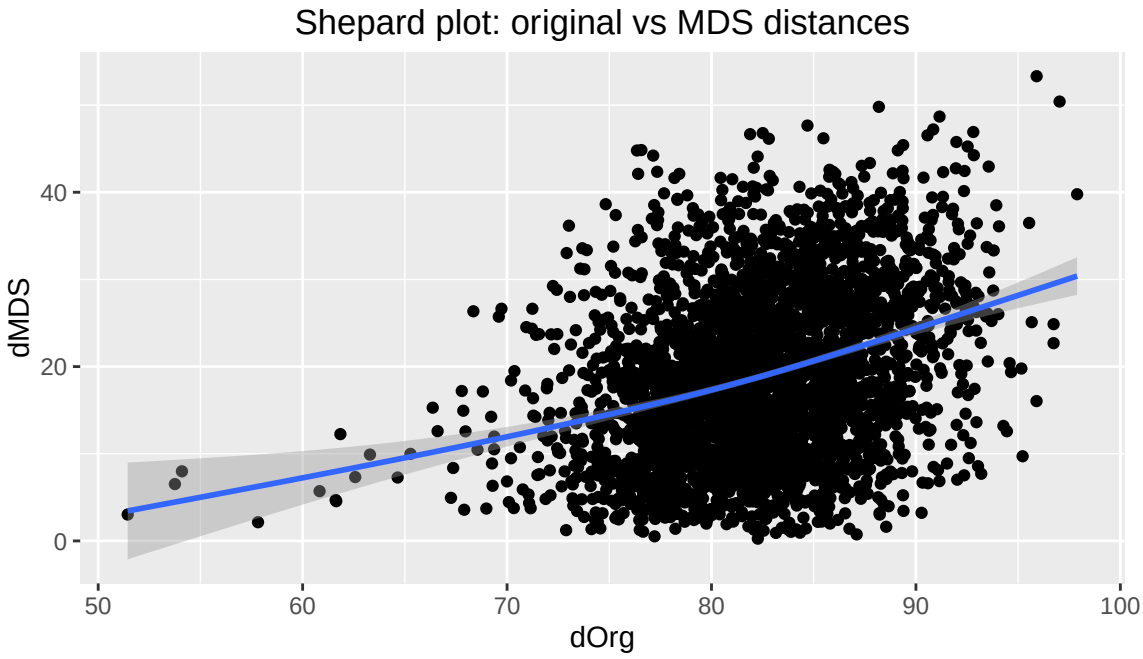
Here we check the classical stress functions, which sums up the squared differences and scales them to the squared sum of the original ones.

Additionally, we produce a "Shepard" plot, which plots the original distances against the ones inferred from using the scaling solution. This allows to assess how accurately the scaling represents the actual distances.

```
distMDS <- dist(scalingEuSC)
## get stress
sum((distEucSC - distMDS)^2)/sum(distEucSC^2)
```

```
   [1] 0.602
```

```
ord <- order(as.vector(distEucSC))
dataGG <- data.frame(dOrg = as.vector(distEucSC)[ord],
                     dMDS = as.vector(distMDS)[ord])

(qplot(dOrg, dMDS, data=dataGG,
       main = "Shepard plot: original vs MDS distances") + geom_smooth())
```



We see that the the scaling is not extremely good.

4.3 Kruskal's Scaling

Kruskal's scaling method is implemented in the function `isoMDS` from the *MASS* package. We used the metric MDS solution as a starting solution.

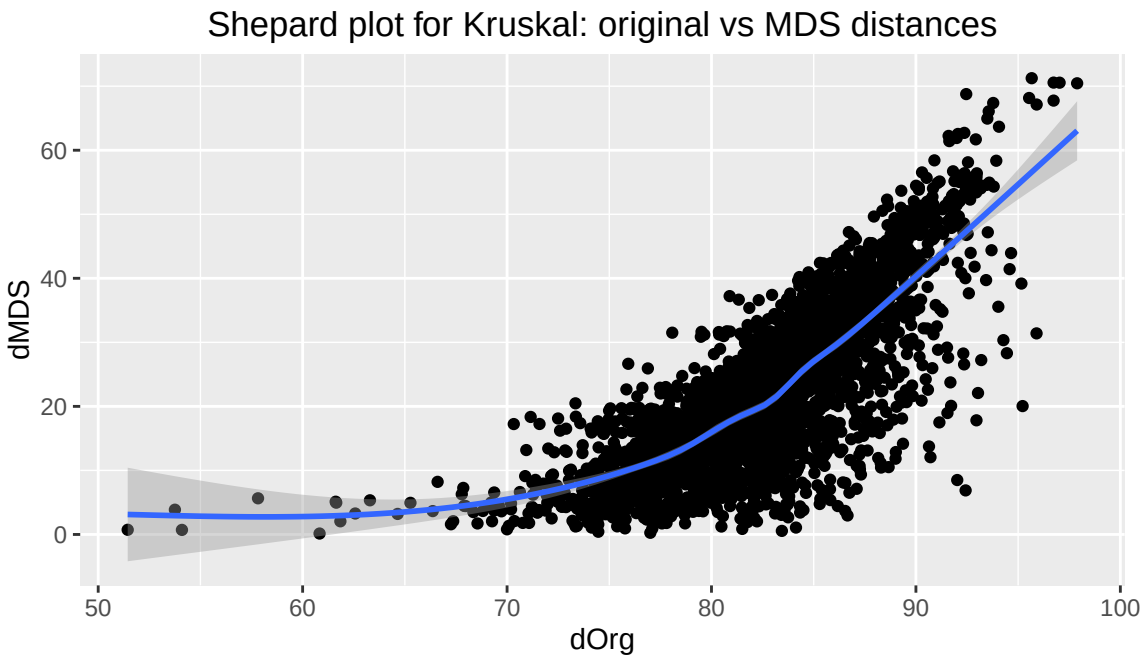
```
kruskalMDS <- isoMDS(distEucSC, y = as.matrix(scalingEuSC))
```

```
   initial  value 42.339647
   iter   5 value 34.391957
   iter  10 value 28.751659
   final  value 28.469270
   converged
```

```
#tressplot(kruskalMDS, distEucSC)

kruskalMDS$stress
```

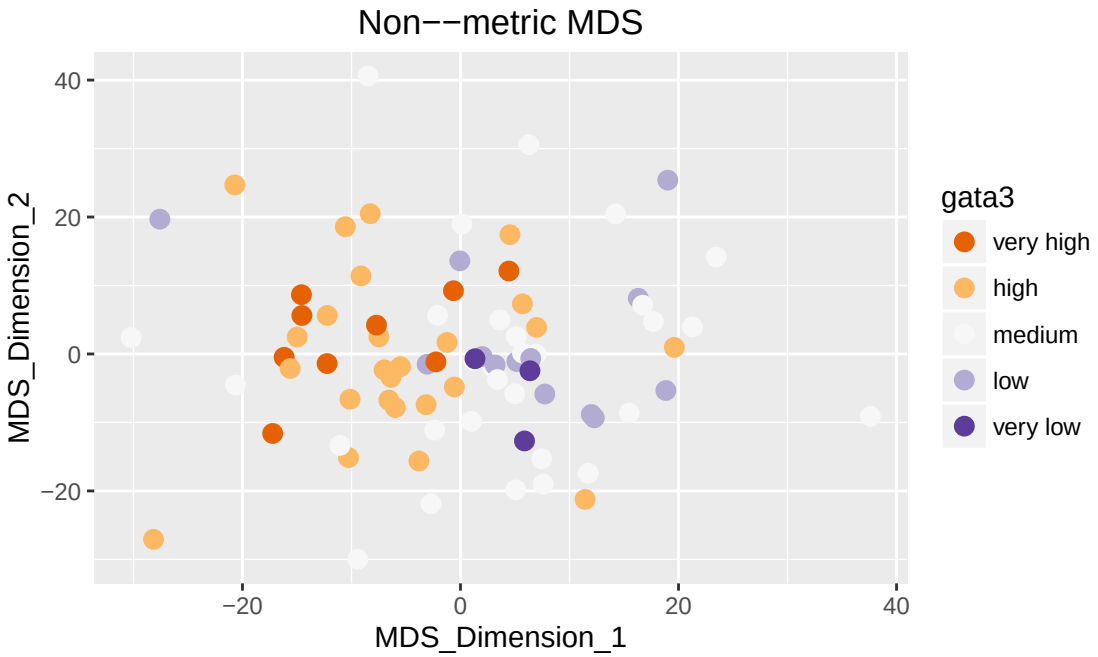
```
   [1] 28.5
```

```
ord <- order(as.vector(distEucSC))
dataGGK <- data.frame(dOrg = as.vector(distEucSC)[ord],
                      dMDS = as.vector(dist(scores(kruskalMDS)))[ord])

(qplot(dOrg, dMDS, data=dataGGK,
       main = "Shepard plot for Kruskal: original vs MDS distances") + geom_smooth())
```



Now the Shepard plot looks much better, the scaling should reflect the original distances in a better way. Let's repeat the plot of the single cells

```
scalingK <- as.data.frame(scores(kruskalMDS))
names(scalingK) <- c("MDS_Dimension_1", "MDS_Dimension_2")

qplot(MDS_Dimension_1, MDS_Dimension_2, data = scalingK,
      main = "Non--metric MDS",
      color = gata3, size = I(3)) + scale_color_brewer(palette = "PuOr")
```



Although the representation is better than the first, metric, MDS the overall visual impression is similar.

Single cell data is commonly used to infer (developmental) hierarchies of single cells. For a nice bioconductor package wrapping a lot of dimension reduction techniques for single cell data, see *sincell* and the associated article, especially, supplementary table 1.

5 Regression models

In regression we use one variable to explain or predict the other. It is customary to plot the predictor variable on the x–axis and the predicted variable on the y–axis. The predictor is also called the independent variable, the explanatory variable, the covariate, or simply x. The predicted variable is called the dependent variable, or simply y.

In a regression problem the data are pairs (x_i, y_i) for $i = 1, \ldots, n$. For each i, y_i is a random variable whose distribution depends on x_i. We write

$$y_i = g(x_i) + \varepsilon_i.$$

The above expresses y_i as a systematic or explainable part $g(x_i)$ and an unexplained part ε_i. Or more informally: response = signal + noise. g is called the regression function. Often the goal is to estimate g. As usual, the most important tool to infer a suitable function is a simple scatterplot. Once we have an estimate of $\hat{g}$ of g, we can compute $r_i := y_i - \hat{g}(x_i)$. The r_i 's are called residuals. The ε_i's themselves are called errors.

Residuals are used to evaluate and assess the fit of models for g. Usually one makes distributional assumption about them, e.g. that they are i.i.d. normally distributed with identical variance σ^2 and mean zero:

$$r_i \sim N(0, \sigma^2)$$

If we choose the regression function g correctly and the model fits the data well, the residuals should thus scatter around a straight line. Always remember that predicting or explaining y from x is not perfect; knowing x does not tell us y exactly. But knowing x does tell us something about y and allows us to make more accurate predictions than if we didn't know x. You can think of a regression model as a distribution model with covariate–dependent mean.

Regression models are agnostic about causality. In fact, instead of using x to predict y, we could use y to predict x. So for each pair of variables there are two possible regressions: using x to predict y and using y to predict x. Sometimes neither variable causes the other. For example, consider a sample of cities and let x be the number of churches and y be the number of bars. A scatterplot of x and y will show a strong relationship between them. But the relationship is caused by the population of the cities. Large cities have large numbers of bars and churches and appear near the upper right of the scatterplot. Small cities have small numbers of bars and churches and appear near the lower left.

A common choice for the regression function g is a linear function, if we have only single predictor X, the simple linear regression model is:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i; \quad \varepsilon_i \sim N(0, \sigma);$$

We can of course always add more predictors, let their total number be denoted by p. Then we get a multiple linear regression:

$$y_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} + \varepsilon_i; \quad j = 1, ..., p$$

5.1 Example: Bodyfat data set

We will illustrate this using a data set on variables influencing your body fat: A variety of popular health books suggest that the readers assess their health, at least in part, by estimating their percentage of body fat.

The data set "bodyfat", which contains variables that can be used to build models predictive of body fat:

- Density determined from underwater weighing
- Percent body fat from Siri's (1956) equation
- Age (years)
- Weight (lbs)
- Height (inches)
- Neck circumference (cm)
- Chest circumference (cm)
- Abdomen 2 circumference (cm)
- Hip circumference (cm)
- Thigh circumference (cm)
- Knee circumference (cm)
- Ankle circumference (cm)
- Biceps (extended) circumference (cm)
- Forearm circumference (cm)
- Wrist circumference (cm)

First, we load in the data set, inspect it a bit and attach it

```
load(url("http://www-huber.embl.de/users/klaus/BasicR/bodyfat.rda"))
dim   (bodyfat)     # how many rows and columns in the dataset?
```

```
[1] 252  15
```

```
names (bodyfat)     # names of the columns)
```

```
 [1] "density"        "percent.fat"    "age"            "weight"
 [5] "height"         "neck.circum"    "chest.circum"   "abdomen.circum"
 [9] "hip.circum"     "thigh.circum"   "knee.circum"    "ankle.circum"
[13] "bicep.circum"   "forearm.circum" "wrist.circum"
```

To get a first impression of the data, we can compute some summary statistics for e.g. age. We can get all these statistics for all the data at once by using an appropriate apply command.

```
## compute descriptive statistics for "age"
summary(bodyfat$age)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   22.0    35.8    43.0    44.9    54.0    81.0
```

```
sd(bodyfat$age)
```

```
[1] 12.6
```

```
mean(bodyfat$age)
```

```
[1] 44.9
```

```
IQR(bodyfat$age)/1.349
```

```
[1] 13.5
```

```
## mean value of every variable in the bodyfat data set
sapply(bodyfat,  FUN = mean)
```

```
     density    percent.fat            age         weight         height
        1.06          19.15          44.88         178.92          70.15
```

```
      neck.circum   chest.circum abdomen.circum     hip.circum   thigh.circum
            37.99         100.82          92.56          99.90          59.41
      knee.circum   ankle.circum   bicep.circum forearm.circum   wrist.circum
            38.59          23.10          32.27          28.66          18.23
```

5.2 Identifying useful predictors

We have a lot of predictors available to come up with regression model for bodyfat. In order to get an overview of the data, we can first produce a PCA plot and a pairwise heatscatter. Observation 39 is removed from the data, since it is clearly an outlier.

5.2.1 PCA and biplots

A PCA represents the individuals in our data set by two "artificial" variables, which are called Principal Components (PCs). The components are linear combinations of the orirginal 15 variables and the linear combination is chosen in such a way that the euclidean distance between the individuals using centered and standardized variables is minimized.

A Biplot is a PCA plot that shows scaled observations combined with a plot of "variable–arrows". these arrows are a way to represent the original variables in the space of the individuals and are scaled versions of the the loadings.

This scaling is controlled by a tuning parameter 'lambda'. The default setting in R leads to observation points that represent approximations of the Mahalanobis distance, which is a distance that takes the covariance between variables into account. The angles between the arrows approximate their correlations.

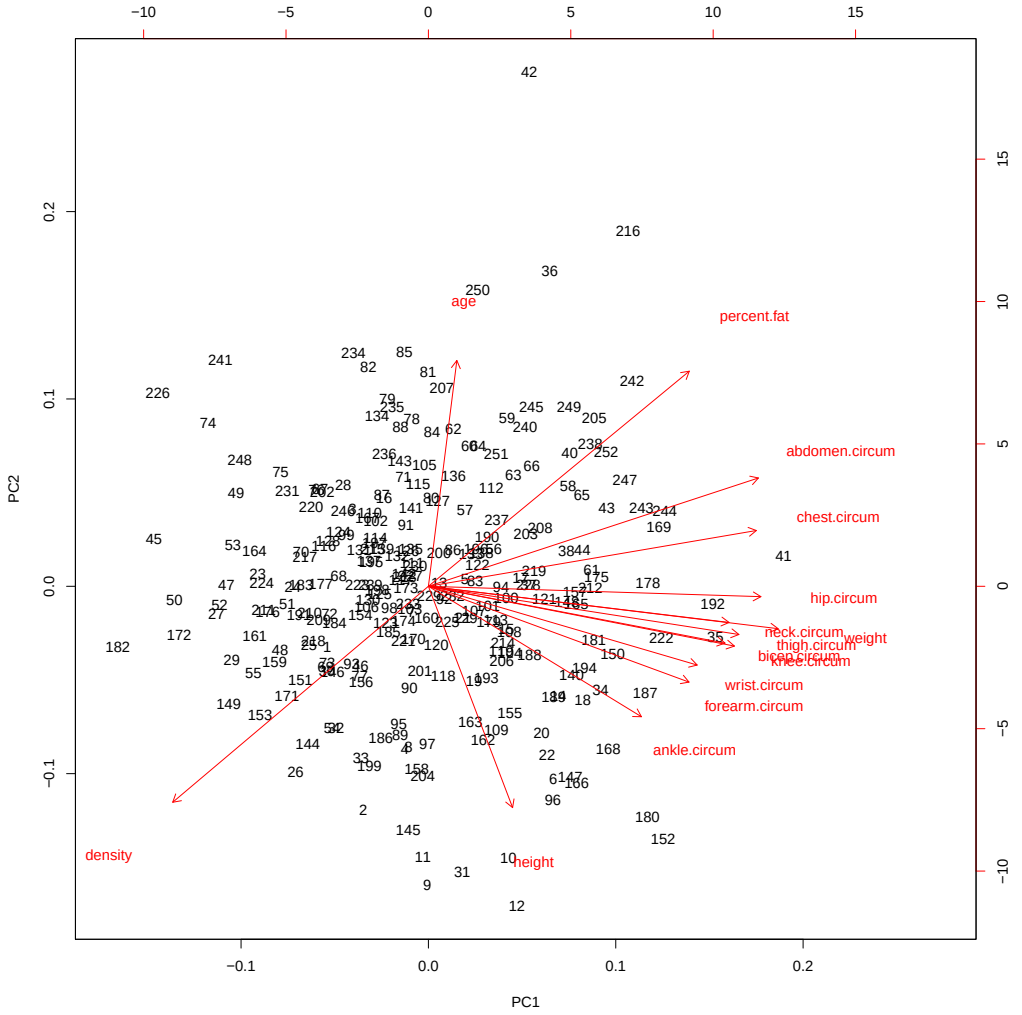
The loadings also correspond to correlation of the variables with the PCs; up to a multiplication by the square–root of the eigenvalue of the corresponding loading vector.

Thus, the x–coordinate of the variable arrows is proportional the correlation with the first PC and the y–coordinate is proportional the correlation with the second PC.

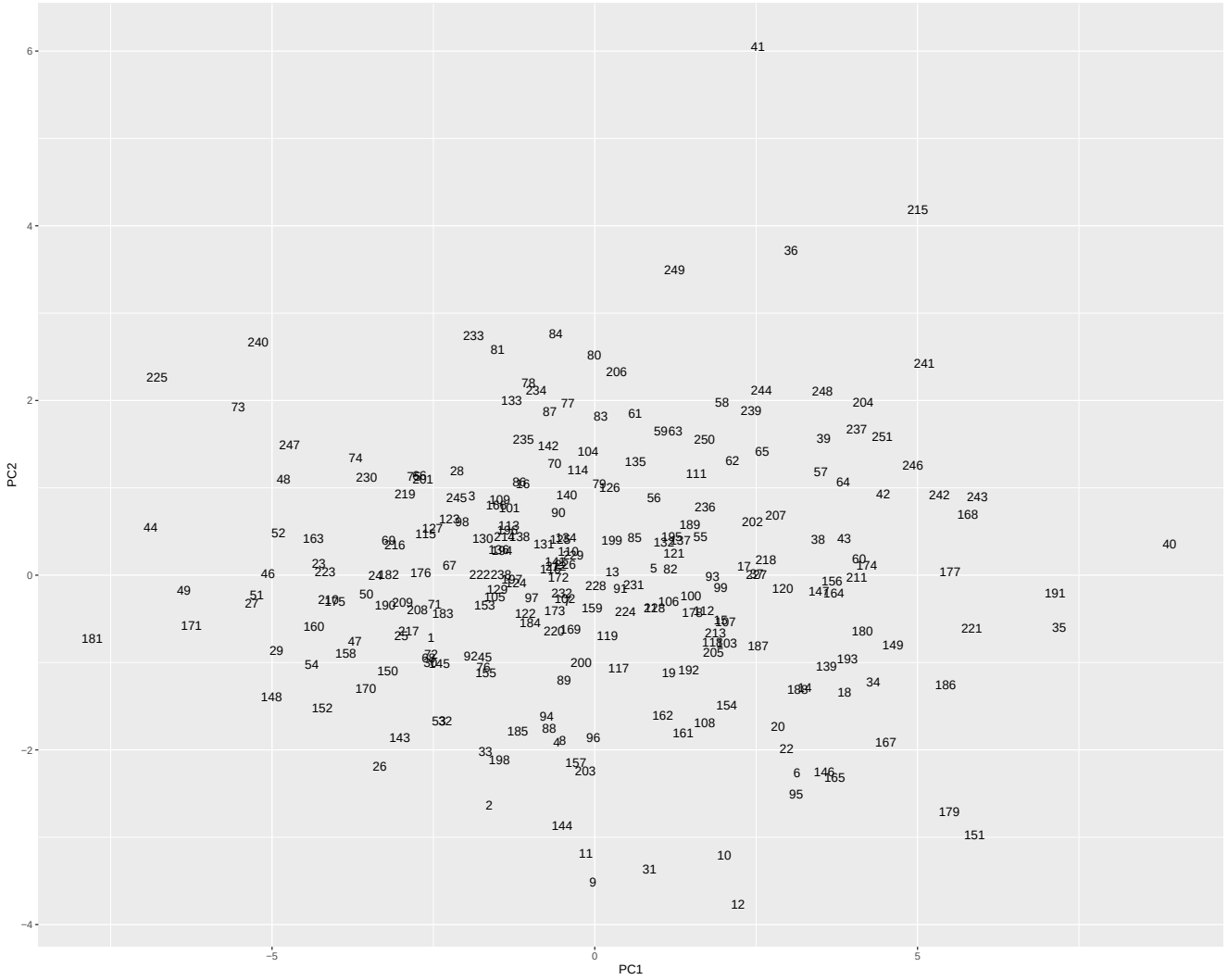
Thus, arrows that point roughly into the same direction indicate correlated variables.

However, since the axis and arrows are rescaled a bit (e.g. by sample size and the tuning parameter) a Biplot can obscure things. Plotting the original PCs we can clearly see that there is essentially one PC that captures all the variation in the data, since the sample points only scatter at random in their PC_2 coordinates. This is hard to see on the Biplot.

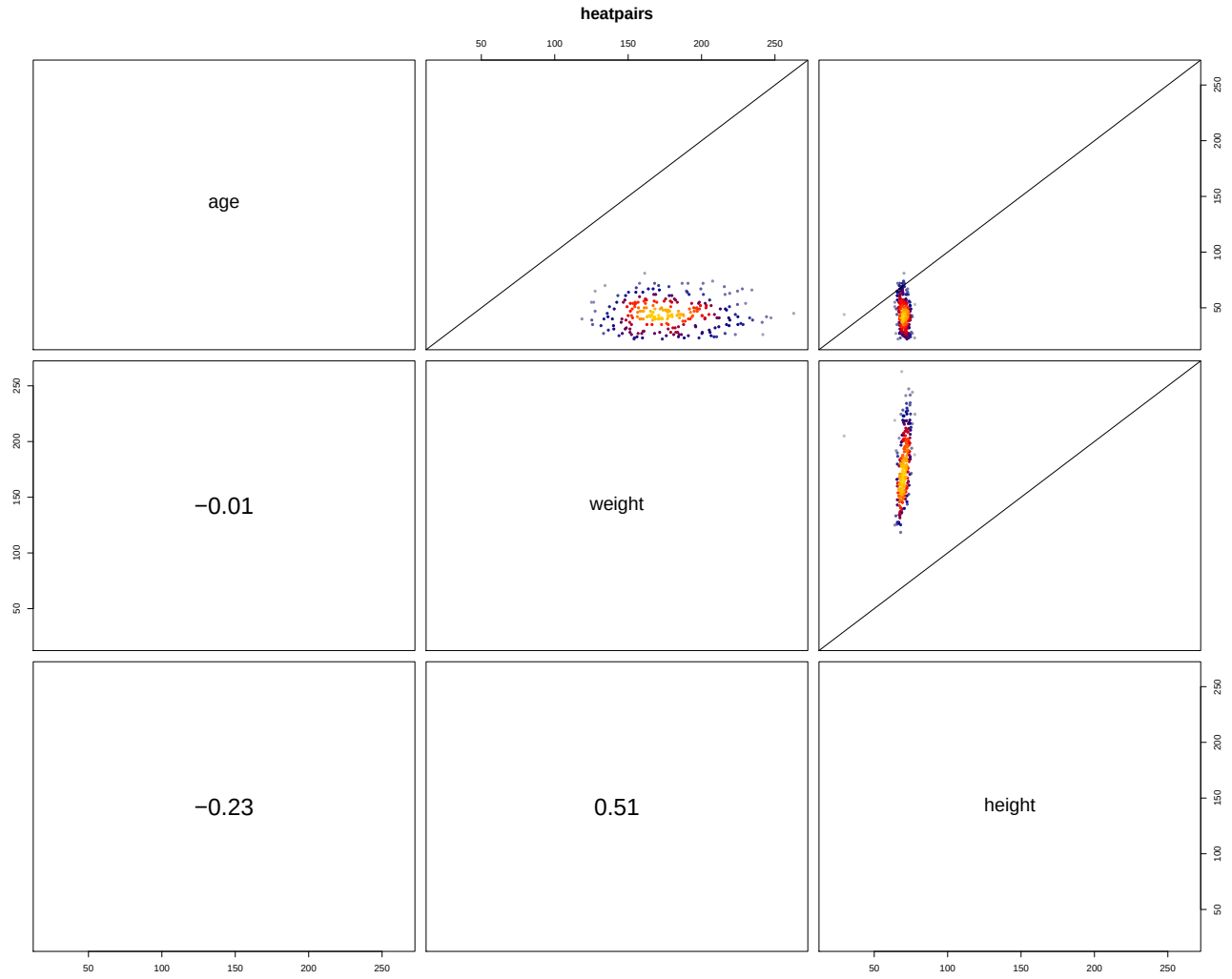
```
if("bodyfat" %in% search()) detach(bodyfat)
bodyfat <- bodyfat[-39,]



PCA <- prcomp(bodyfat, center = TRUE, scale = TRUE)
biplot(PCA)
```




```
## plot original PCs
dataGG <- data.frame(PC1 = PCA$x[,1], PC2 = PCA$x[,2], no = seq_along(PCA$x[,1]))
qplot(PC1, PC2, data = dataGG, label = no, geom = "text")
```




```
heatpairs(as.matrix(bodyfat[,3:5]))
```



5.2.2 The french way of Biplots

The french school of multivariate analysis prefers a different (and in my opinion nicer) way of visualizing a PCA. Instead of producing a biplot, the correlations between the PCs and the original variables are visualized directly in a "correlation circle".

This is implemented in the package *ade4* which stands for "Analyse de Donnees destine d'abord a la manipulation des donnes Ecologiques et Environnementales avec des procedures Exploratoires d'essence Euclidienne"

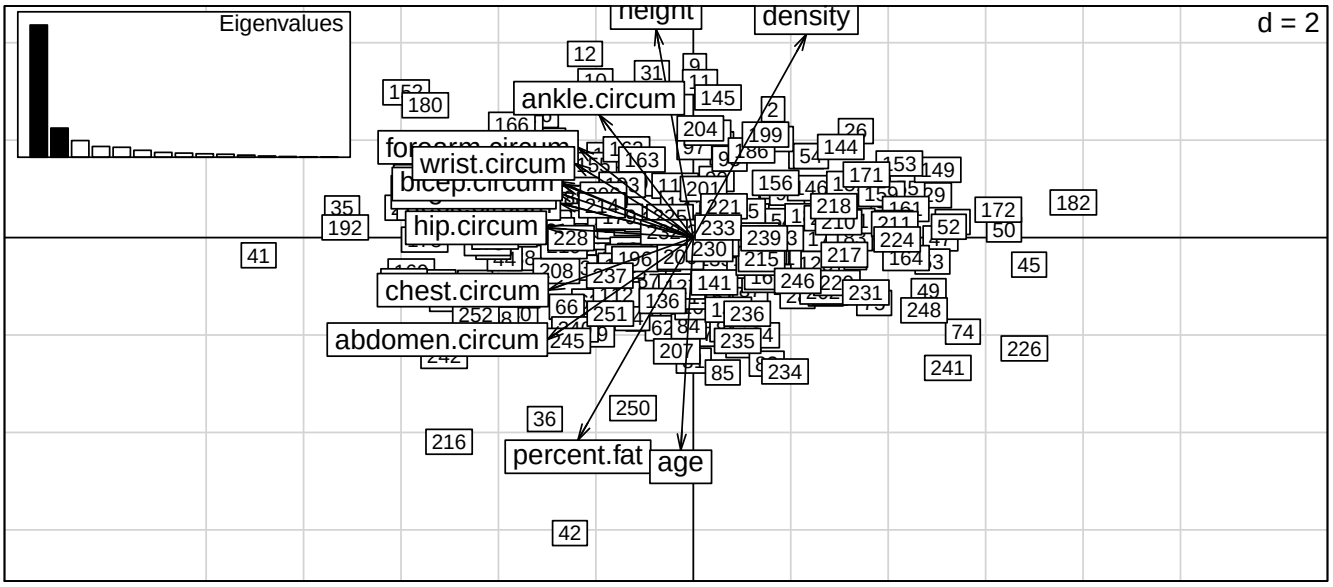
This packages contains a lot of nice multivariate analysis methods, not limited to PCA.

```
frenchPCA <- dudi.pca(bodyfat, scannf = FALSE)
frenchPCA
```

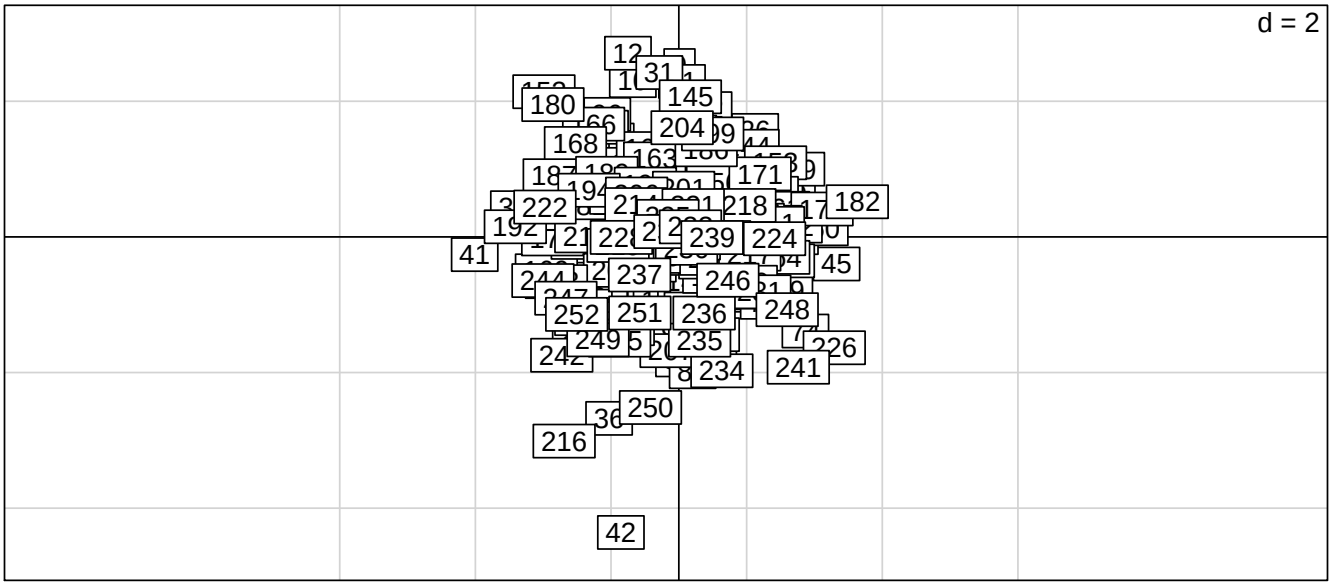
```
   Duality diagramm
   class: pca dudi
   $call: dudi.pca(df = bodyfat, scannf = FALSE)

   $nf: 2 axis-components saved
   $rank: 15
   eigen values: 8.79 1.94 1.11 0.715 0.661 ...
```

```
     vector length mode    content
   1 $cw    15     numeric column weights
   2 $lw    251    numeric row weights
   3 $eig   15     numeric eigen values

     data.frame nrow ncol content
   1 $tab       251  15   modified array
   2 $li        251  2    row coordinates
   3 $l1        251  2    row normed scores
   4 $co        15   2    column coordinates
   5 $c1        15   2    column normed scores
   other elements: cent norm
```

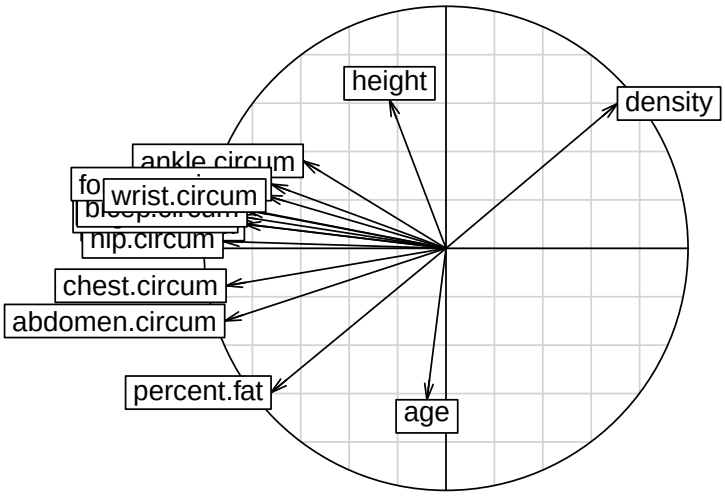
```
# biplot
scatter(frenchPCA, type ="lines")
```




```
# PCA plot
s.label(frenchPCA$l1)
```




```
# correlation circle
s.corcircle(frenchPCA$co)
```



5.2.3 Selecting a predictor

We see that a lot of potential predictor are heavily correlated with one another, in fact all except for age, height and density can be used interchangeably.

Because of this, we can fit a first regression by using the predictor that has the highest individual correlation with bodyfat.

```
### look at correlations of bodyfat with other variables
select.vars <- abs(cor(bodyfat))["percent.fat" ,]
select.vars
```

```
          density    percent.fat            age         weight         height
           0.9876         1.0000         0.2929         0.6193         0.0947
      neck.circum   chest.circum abdomen.circum     hip.circum   thigh.circum
```

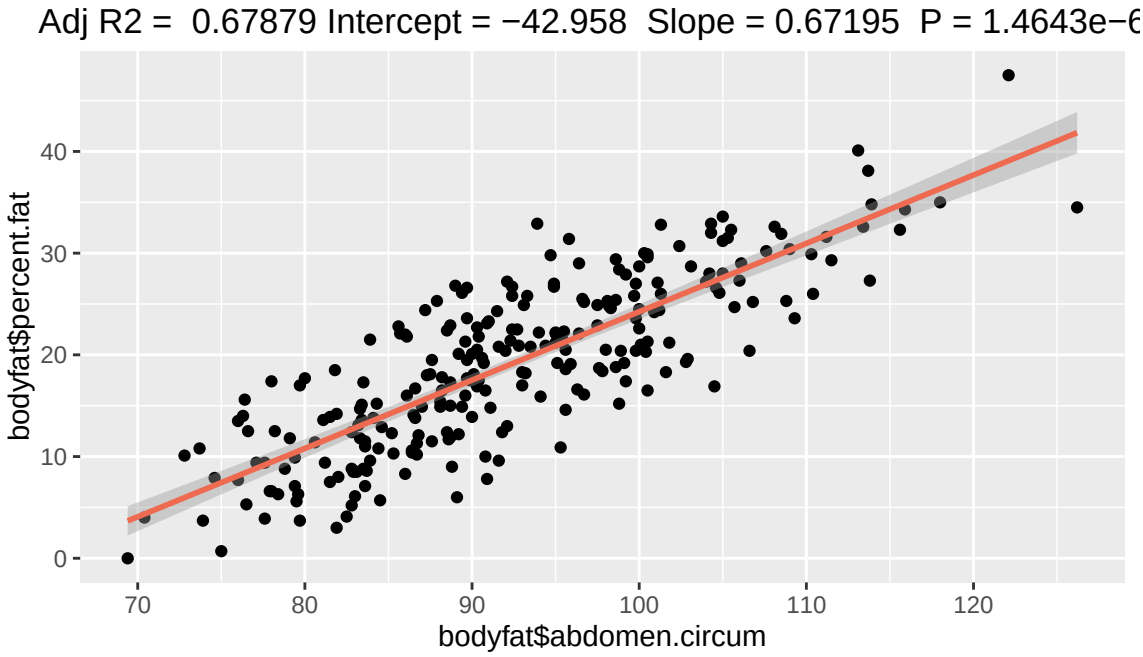
```
          0.4816          0.7005          0.8247          0.6379          0.5550
     knee.circum   ankle.circum   bicep.circum forearm.circum   wrist.circum
          0.4981          0.2457          0.4818          0.3628          0.3306
```

```
select.vars[select.vars > 0.6]
```

```
         density    percent.fat         weight   chest.circum abdomen.circum
           0.988          1.000          0.619          0.701          0.825
      hip.circum
           0.638
```

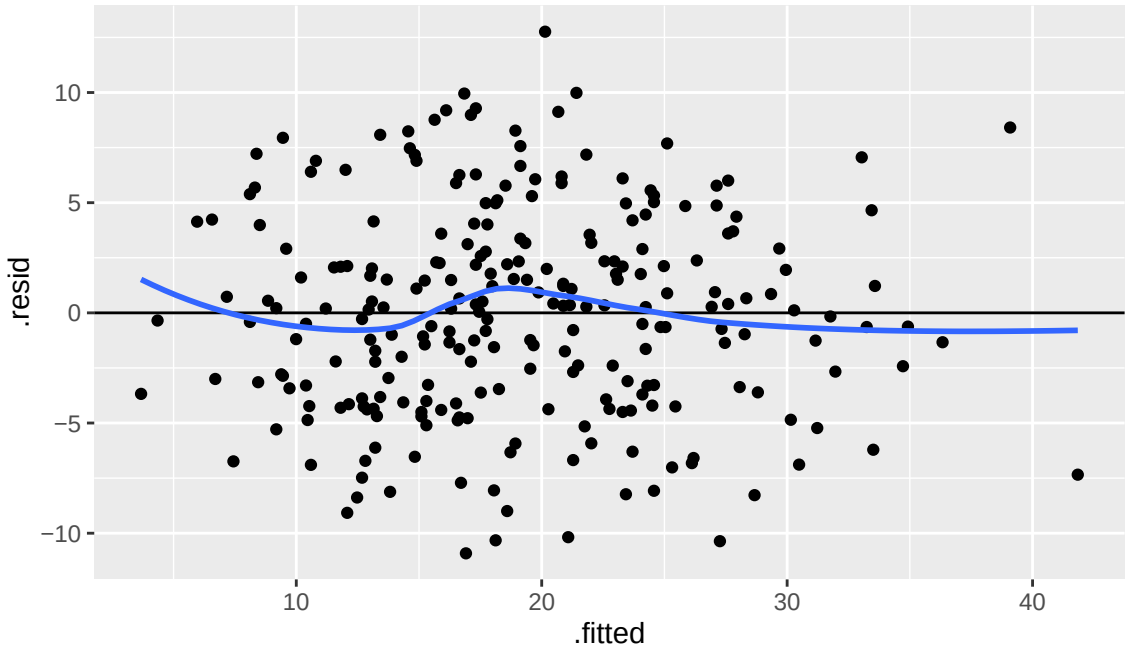
We see that `abdomen.circum` has the highest correlation with bodyfat. Thus we use this variable in a regression model and plot the result.

```
lm.fat <- lm(bodyfat$percent.fat ~ bodyfat$abdomen.circum )


ggplotRegression(lm.fat)
```



The model looks reasonable, we can now visually check whether the residuals follow a Gaussian law with mean zero:

```
qplot(.fitted, .resid, data = fortify(lm.fat)) +
  geom_hline(yintercept = 0) + geom_smooth(se = FALSE)
```



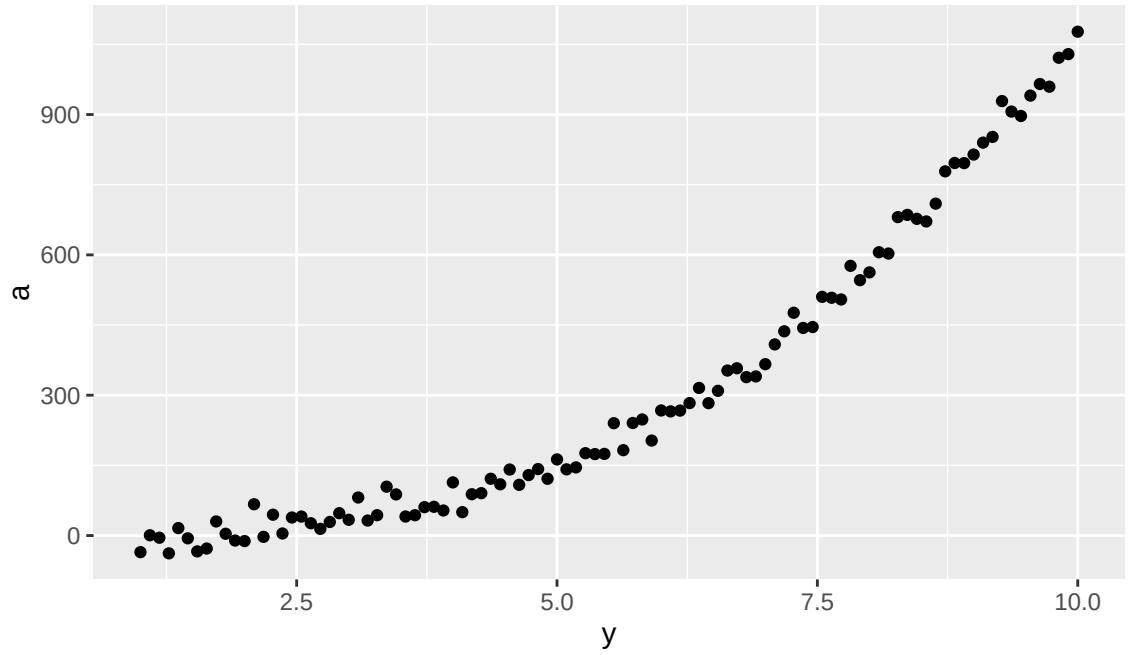
There seems to be no systematic trend in the residuals, so the model fit seem to be fine! Thus abdominal circumference is highly predictive of bodyfat in man. A result which is possibly not really surprising. The data also indicates that almost all body measures can be used to predict bodyfat.

6 Local Regression (LOESS)

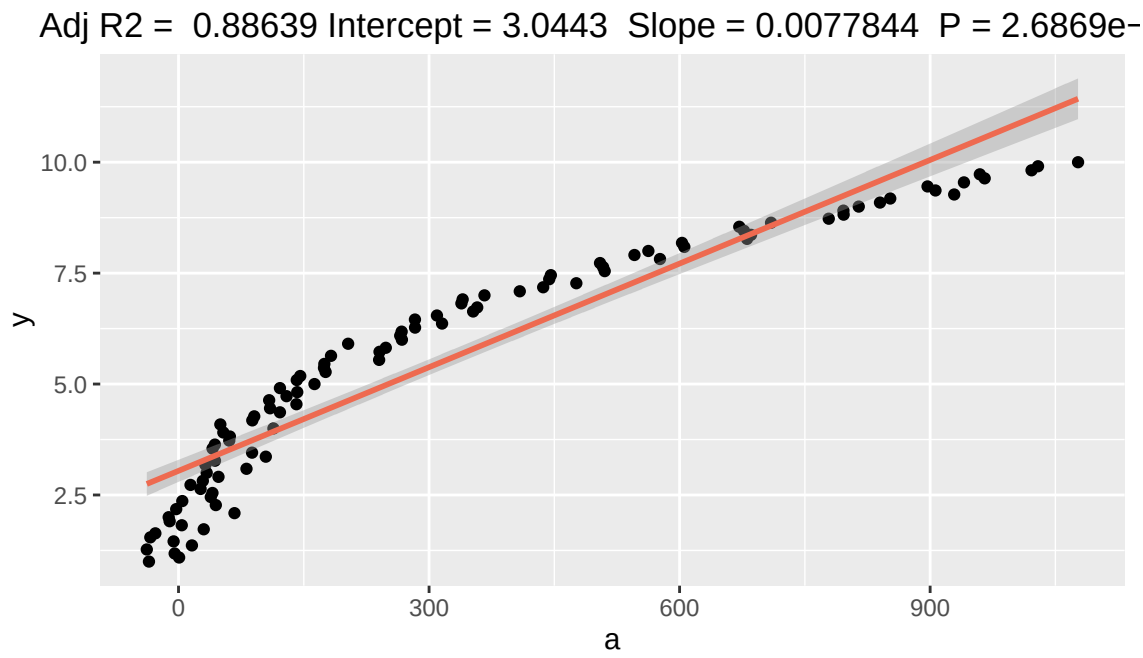
Local regression is a commonly used approach for fitting flexible non–linear functions, which involves computing many local linear regression fits and combining them. Local regression is a very useful technique both for data visualization and trend fitting. Fitting many local models requires quite some computational power, but it usually feasible with today's hardware. We illustrate the local regression using the *locfit* package on simulated data.

We first fit a linear regression line to simulated data that follows a polynomial trend and see that it does not really fit well.

```
# create_data
y <- seq(from=1, to=10, length.out=100)
a <- y^3 +y^2  + rnorm(100,mean=0, sd=30)
dataL <- data.frame(a=a, y=y)
qplot(y, a, data = dataL)
```




```
# linear fit
linreg <- lm(y~a, data = dataL)

ggplotRegression(linreg)
```

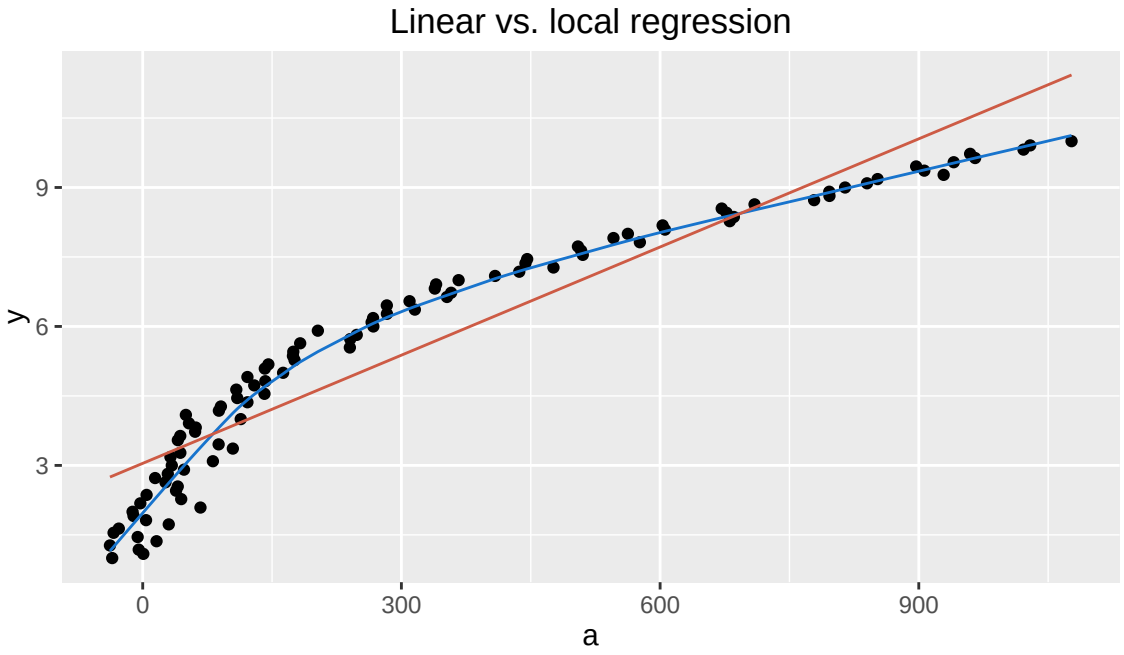



```
dataL$LinReg <- predict(linreg)
```

We now use the function `locfit` to perform a local regression on the data. It takes the predictors wrapped in a call to `lp()`. Within this function we can also set tunning parameters. An important one is the `nn` one, which set the proportion of nearest–neighbors to be used for the local fits.

The lower this percentage, the more closely the line will follow the data points.

```
dataL$locFit  <- predict(locfit(y~lp(a, nn=0.5, deg=1), data=dataL),
                         newdata = dataL$a)


 (qplot(a, y, data = dataL, main = "Linear vs. local regression")
 +  geom_line(aes(x = a, y = locFit), color = "dodgerblue3")
 +  geom_line(aes(x = a, y = LinReg), color = "coral3"))
```



7 Regression and PCA in the analysis of RNA–Seq data

We now illustrate some of the concepts introduced above by looking at the analysis of next generation sequencing data.

Next generation sequencing data such as RNA–Seq are often analyzed in the form of counts assigned to certain bins, e.g. a bin could be all the exons of a gene, or an isoform for RNA–Seq data. This count data then shows overdispersion, i.e. the variance is greater than than the mean.

In fact, it is known that a standard Poisson model can only account for the technical noise in RNA–Seq data. In the Poisson model the variance is equal to the mean, while in RNA–Seq data the variance is greater than the mean.

A popular way to model this is to use a Negative–Binomial-distribution (NB), which includes an additional parameter dispersion parameter α such that $E(NB) = \mu$ and

$$\text{Var}[\text{NB}(\mu, \alpha)] = \mu + \alpha \cdot \mu^2$$

Hence, the variance is greater than the mean. *DESeq2* uses the NB model and fits dispersion values (see below).

We use a dataset produced by Bottomly et al., sequencing two strains of mouse with many biological replicates. This dataset and a number of other sequencing datasets have been compiled from raw data into read counts tables by Frazee, Langmead, and Leek as part of the ReCount project. These datasets are made publicly available at the following website:

http://bowtie-bio.sourceforge.net/recount/

Unlike many sequencing studies, Bottomly et al., realizing the such information is important for downstream analysis, provided the experiment number for all samples. In fact : the strain of mouse. We remove genes with less than 5 counts in any sample from the analyis since we cannot draw much information from genes with a very low number of counts anyway.

The data is stored as an ExpressionSet object.

```
load(url("http://www-huber.embl.de/users/klaus/bottomly_eset.RData"))
bottomly.eset
```

```
   ExpressionSet (storageMode: lockedEnvironment)
   assayData: 36536 features, 21 samples
     element names: exprs
   protocolData: none
   phenoData
     sampleNames: SRX033480 SRX033488 ... SRX033494 (21 total)
     varLabels: sample.id num.tech.reps ... lane.number (5 total)
     varMetadata: labelDescription
   featureData
     featureNames: ENSMUSG00000000001 ENSMUSG00000000003 ...
       ENSMUSG00000090268 (36536 total)
     fvarLabels: gene
     fvarMetadata: labelDescription
   experimentData: use 'experimentData(object)'
   Annotation:
```

```
pData(bottomly.eset)
```

	sample.id	num.tech.reps	strain	experiment.number	lane.number
SRX033480	SRX033480	1	C57BL/6J	6	1
SRX033488	SRX033488	1	C57BL/6J	7	1
SRX033481	SRX033481	1	C57BL/6J	6	2
SRX033489	SRX033489	1	C57BL/6J	7	2
SRX033482	SRX033482	1	C57BL/6J	6	3
SRX033490	SRX033490	1	C57BL/6J	7	3
SRX033483	SRX033483	1	C57BL/6J	6	5
SRX033476	SRX033476	1	C57BL/6J	4	6
SRX033478	SRX033478	1	C57BL/6J	4	7
SRX033479	SRX033479	1	C57BL/6J	4	8
SRX033472	SRX033472	1	DBA/2J	4	1
SRX033473	SRX033473	1	DBA/2J	4	2
SRX033474	SRX033474	1	DBA/2J	4	3
SRX033475	SRX033475	1	DBA/2J	4	5
SRX033491	SRX033491	1	DBA/2J	7	5
SRX033484	SRX033484	1	DBA/2J	6	6
SRX033492	SRX033492	1	DBA/2J	7	6
SRX033485	SRX033485	1	DBA/2J	6	7
SRX033493	SRX033493	1	DBA/2J	7	7
SRX033486	SRX033486	1	DBA/2J	6	8
SRX033494	SRX033494	1	DBA/2J	7	8

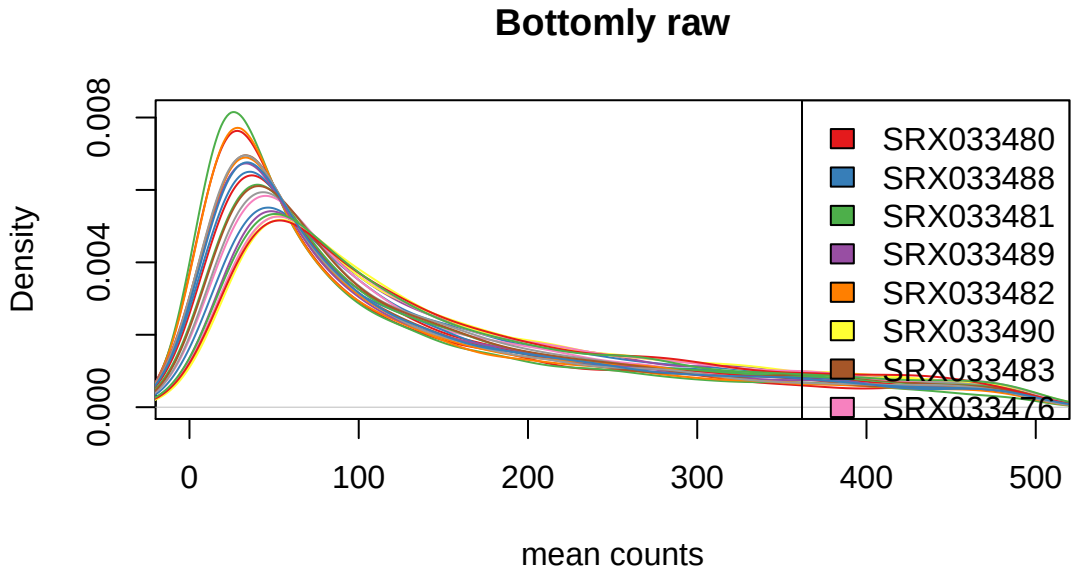
```
idx.nn <- apply(exprs(bottomly.eset), 1, function(x) { all(x > 5)})
bottomly.eset <- subset(bottomly.eset, idx.nn)
```

7.1 Data calibration / normalization

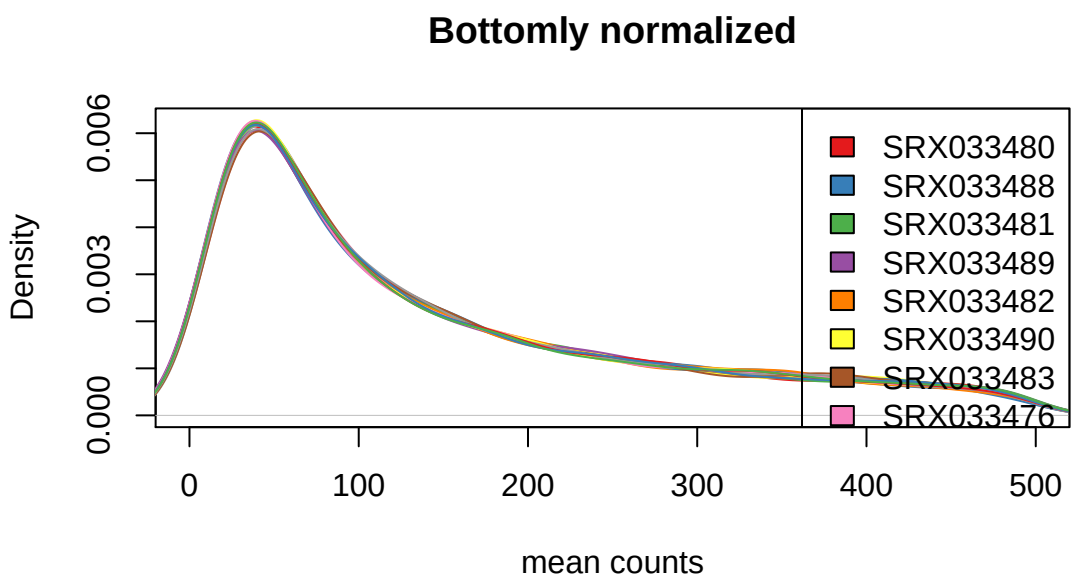
In order to make the samples comparable to each other, including normalization for sequencing depth, we calculate sizefactors. A sizefactor is the median gene expression per sample relative to a reference sample formed by the geometric mean of each single gene across samples.

Thus it, gives a scaling factor which makes the gene expression measurements (counts) comparable across samples. Dividing the counts by the sizefactors will normalize them. Note that the size factor normalization will only work if most of the genes are not differentially expressed between the samples. We use the function `estimateSizeFactorsForMatrix` from the *DESeq2* in order to calculate the size factors for each sample.

```
bottomly.sf <- estimateSizeFactorsForMatrix(exprs(bottomly.eset))


bottomly.norm <- bottomly.eset

 for(i in seq_along(bottomly.sf)){
   exprs(bottomly.norm)[,i] <-  exprs(bottomly.eset)[,i] /  bottomly.sf[i]
 }


multidensity( exprs(bottomly.eset), xlab="mean counts", xlim=c(0, 500),
           main = "Bottomly raw")
```




```
multidensity( exprs(bottomly.norm), xlab="mean counts", xlim=c(0, 500),
           main = "Bottomly normalized")
```



We see that the normalization brings the samples to a common scale. We can check this further by producting pairwise MA plots.

7.2 Pairwise MA plots

MA plots are used to study dependences between the log ratio of two variables and the mean values of the same two variables. The variables are the genes in our examples. The log ratios (= log fold–change) of the two measurements are called M values (from "minus" in the log scale) and are represented in the vertical axis. The mean values of the two measurements are called A values (from "average" in the log scale) and are represented in the horizontal axis. Commonly a log2 scale is used, since this allows a straighfoward interpretation of the M values since a M value of 1 corresponds to a two–fold change, a M value of 2 two a for fold change and so on.

Since we have a total of 21 samples, we will have a total number of 210 pairwise MA plots. We use the function `MDPlot` from the *EDASeq* package to achieve this.

```
bottomly.log2 <- log2(exprs(bottomly.norm))
bottomly.log2.raw <- log2(exprs(bottomly.eset))
```

```
pdf("pairwiseMAsBottomly.pdf")
 MA.idx = t(combn(1:21, 2))

        for( i in  1:dim(MA.idx)[1]){
         MDPlot(bottomly.log2,
                c(MA.idx[i,1],MA.idx[i,2]),
        main = paste( sampleNames(bottomly.norm)[MA.idx[i,1]], " vs ",
         sampleNames(bottomly.norm)[MA.idx[i,2]] ))
        }
dev.off()
```

Exercise: Normalization strategies

(*a*) Create pairwise MA plots for the raw data on the log2 scale. Can you spot samples that a very different from each other?

(*b*) An alternative to size factor normalization is to just devide the counts of every sample by its sums divided by 10^6. This is sometimes called counts per million normalization. Compare this to the size factor normalization by creating appropriate plots.

7.3 Mean–variance relationship

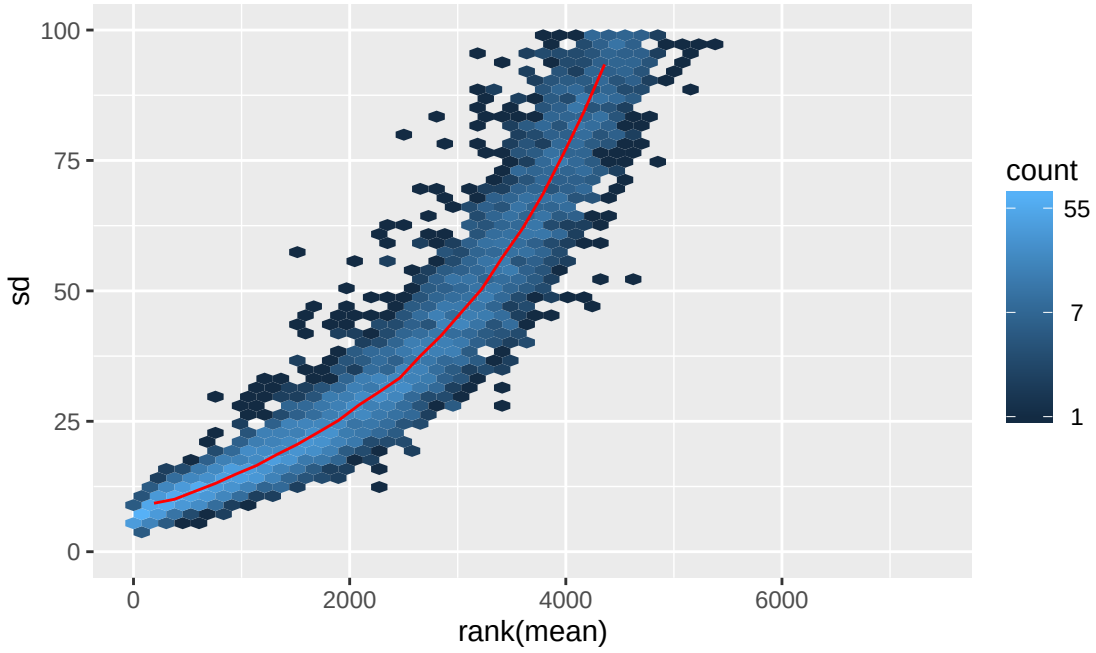
7.3.1 Mean–sd plots

Count data are heteroscesdastic, that is, their variance depends on the mean. For the data set at hand, this means that the higher the gene expression of the gene, the higher is its variance. We can see this relationship by producing a plot of the ranked mean bins against the estimated standard deviation in each bin. Means and standard deviations are estimated per gene in this case. We use the function `meanSdPlot` from the *VSN* package to achieve this.

```
pl <- meanSdPlot(exprs(bottomly.eset))
```

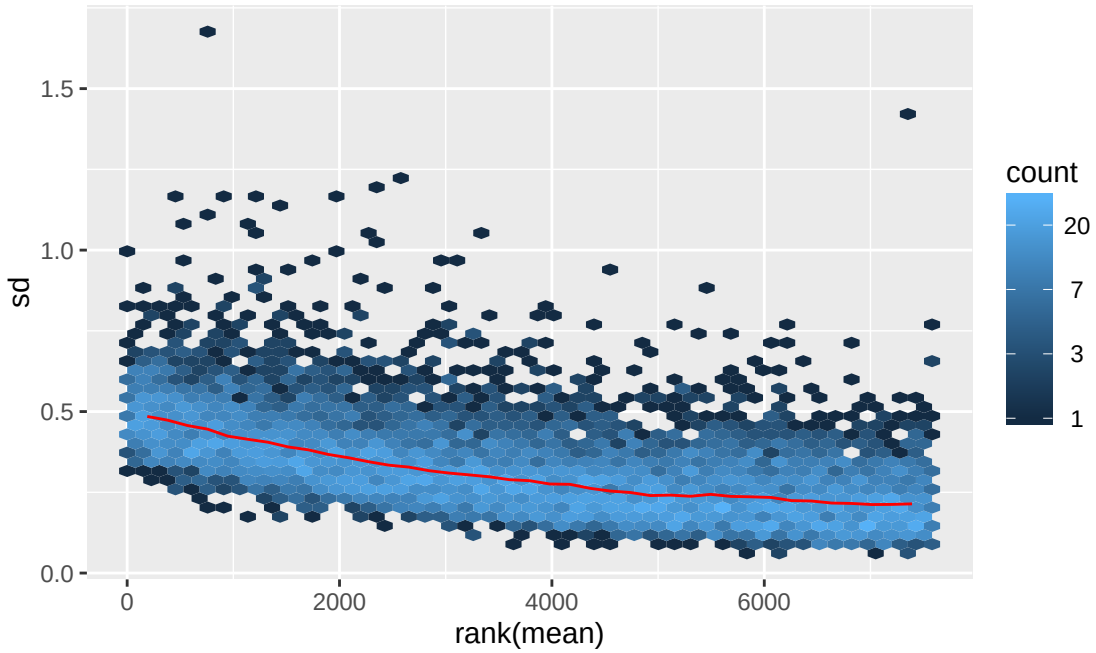
```
pl$gg + ylim(0,100)
```

```
  Warning:  Removed 3109 rows containing non-finite values (stat_binhex).
  Warning:  Removed 1 rows containing missing values (geom_hex).
  Warning:  Removed 16 rows containing missing values (geom_path).
```



Homoscedastic data will produce a straight red line in this plot. The log2–transform aleviates the heteroscedasticity but does not completely remove it as we can see in the following plot.

```
meanSdPlot(bottomly.log2 )
```



Not that the log transform is an example of a special class of transformations called Box–Cox transformations that have been developped in order to make data "more normal", i.e. removing skewness and heteroscedasticity. They are given by:

$$x^{\text{Box–Cox}} = \begin{cases} \frac{(x+1)^{\lambda}-1}{\lambda}, \lambda \neq 0 \\ \log(x+1), \lambda = 0 \end{cases}$$

The parameter λ is a tuning parameter for gene that can be optimized in such a way that the distribution of the transformed data has the largest similarity to a normal distribution.

Another effect of the log–transformation is that log–transformed count values are often more highly correlated than raw ones: the log–transform mitigates the influence of outliers and makes the data "more normal". We check this for sample 6 and 11 of the bottomly data.

```
cor(exprs(bottomly.eset)[,6],exprs(bottomly.eset)[,11])
```

```
   [1] 0.786
```

```
cor(bottomly.log2[,6], bottomly.log2[,11])
```

```
   [1] 0.956
```

The BoxCox transformations are available in the function `bcPower` from the *car*.

7.3.2 Heteroscesdasticity removal by adding pseudocounts

The log2 transform may not work because RNAseq data contains many 0s. One quick way to get around this is by adding a constant (sometimes called "pseudocount") before taking the log. A typical one is 0.5 which gives us a log2 value of -1 for 0s.

It will push all count values upwards, thus mitigating the variance dependency on the mean. This is due to the fact that for log–transformed normalized counts k following a NB distribution and a size factor s we have approximately:

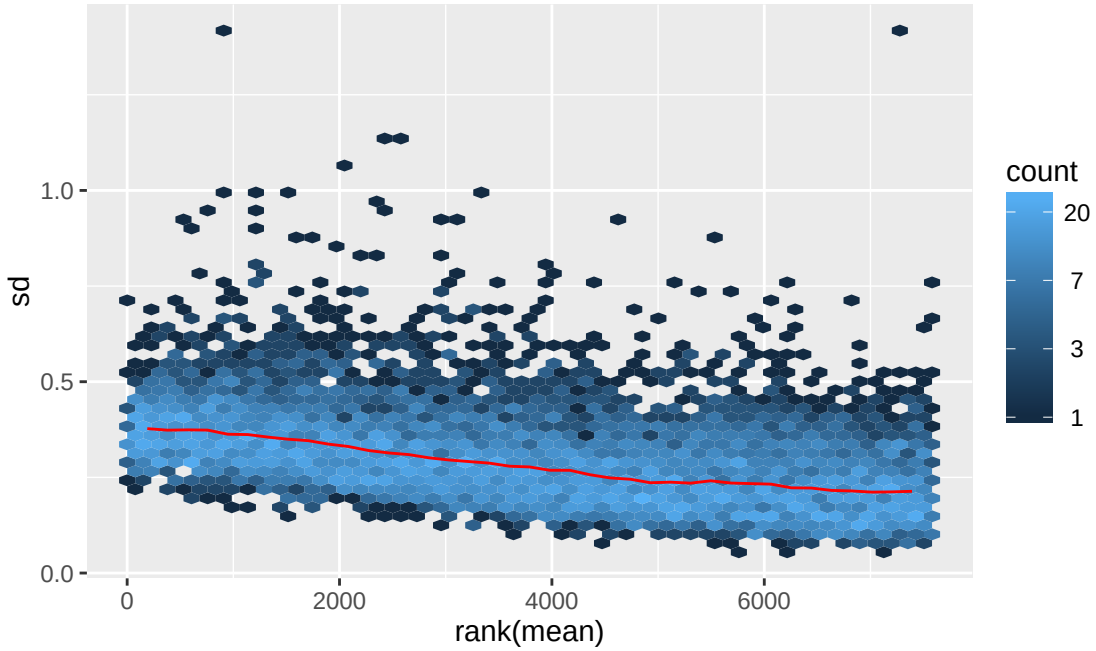
$$\text{Var}(log(sk)) \approx = \frac{1}{sk} + \alpha$$

where α is the dispersion estimate. This shows that for genes with a large mean, the variance will no longer depend on the mean and essentially be equal to alpha since $\frac{1}{sk}$ is very close to zero.

However, adding pseudocounts will artificially lower the the variance for lowly expressed genes, possibly leading to a higher rate of false positive calls in differential expression analysis.

In general, are carefull modelling of the variance mean realtionship is preferable to ad–hoc solutions. Interestingly even adding 5 essentially removes the mean–variance dependency for the Bottomly data:

```
meanSdPlot(log2(exprs(bottomly.norm) + 5) )
```




```
#library(LSD)
#heatscatter(rowMeans(bottomly.log2),rowSds(bottomly.log2),
#xlab= "mean", ylab = "sd")
```

As already indicated, it is however debatable, wether this represents a valid pre–transformation since it artificially lowers the variance for lowly expressed genes. Also note that there is only very mild overdispersion present in the Bottomly data.

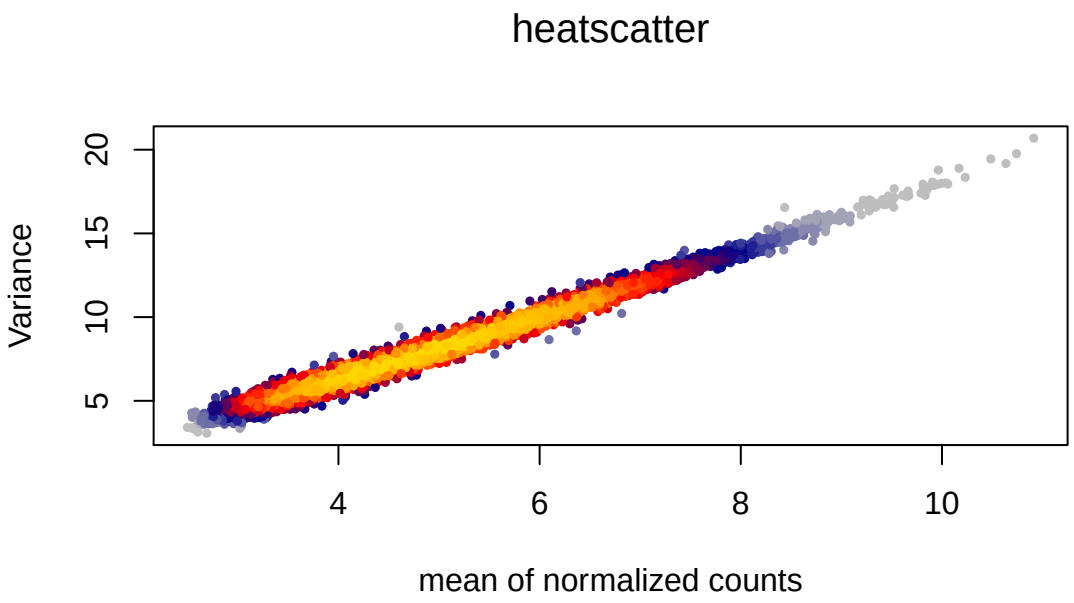
7.4 Modelling the mean–variance relationship

We will now look at the approach of the *DESeq* (and similarly *DESeq2*) package to model the mean–variance relationship. These packages fit a negative binomial model for each gene by trying to infer a dispersion parameter α. We use low–level functions of the *DESeq* to illustrate this. First, means and variances for every gene are obtained. We use the function `heatscatter` from the *LSD* to visualize the mean–variance relationship of the data.

The typical shape of the dispersion fit is an exponentially decaying curve. The asymptotic dispersion for highly expressed genes can be seen as a measurement of biological variability in the sense of a squared coefficient of variation: a dispersion value of 0.01 means that the gene's expression tends to differ by typically $\sqrt{0.01} = 10\%$ between samples of the same treatment group.

For weak genes, the Poisson noise is an additional source of noise, which is added to the dispersion.

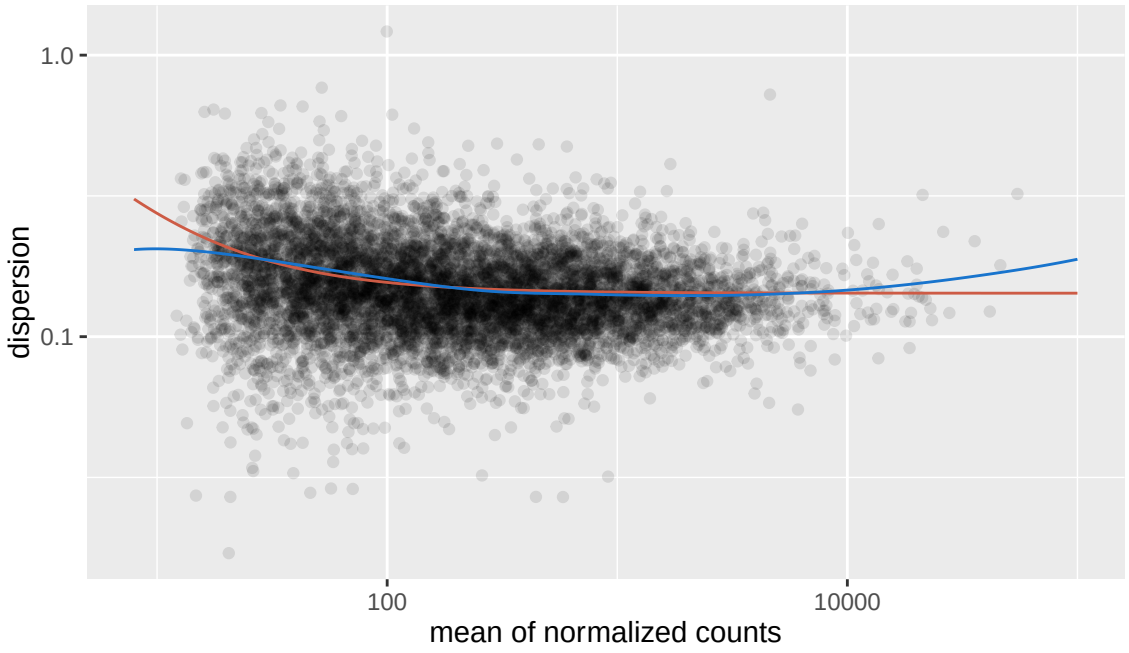
```
meansAndVars <- DESeq:::getBaseMeansAndPooledVariances(
  counts=exprs(bottomly.norm),sizeFactors=bottomly.sf,
   conditions=pData(bottomly.eset)$strain)
heatscatter(log(meansAndVars$baseMean),
            log(meansAndVars$baseVar), xlab= "mean of normalized counts",
            ylab = "Variance")
```



The plot confirms that there is only mild overdispersion present in the data. Most of gene dispersions are only slightly above 0.1 for each gene. In practice, one would then fit a line through the dispersion estimates, and the use this regression line to predict the dispersions. This allows to mitigate the influence of dispersion variability and enables the sharing of information across genes. This is a common strategy with low sample sizes, leading to more stable estimates. We can here plot both a parametric (red) and a local fit (blue).

```
dispersions <-  DESeq:::estimateAndFitDispersionsFromBaseMeansAndVariances(
  means=meansAndVars$baseMean, variances=meansAndVars$baseVar,
  sizeFactors=bottomly.sf, fitType="parametric")

dispersionsLocal <-  DESeq:::estimateAndFitDispersionsFromBaseMeansAndVariances(
  means=meansAndVars$baseMean, variances=meansAndVars$baseVar,
  sizeFactors=bottomly.sf, fitType="local")

    px = meansAndVars$baseMean
    py = dispersions$disps
    xg = 10^seq(0.9, 5, length.out = length(dispersions$disps))
    fitg = dispersions$dispFun(xg)
    fitLocal = dispersionsLocal$dispFun(xg)
    dataGG = data.frame(px, py, xg, fitg)

  (qplot(px, py, data = dataGG, ylab = "dispersion",
         xlab= "mean of normalized counts",
         log = "xy", alpha = I(1/10))
     + geom_line(aes(x = xg, y = fitg), color = "coral3")
      + geom_line(aes(x = xg, y = fitLocal), color = "dodgerblue3")
```

```
)
```



We see that the parametric fit seems to capture the trend better for this data.

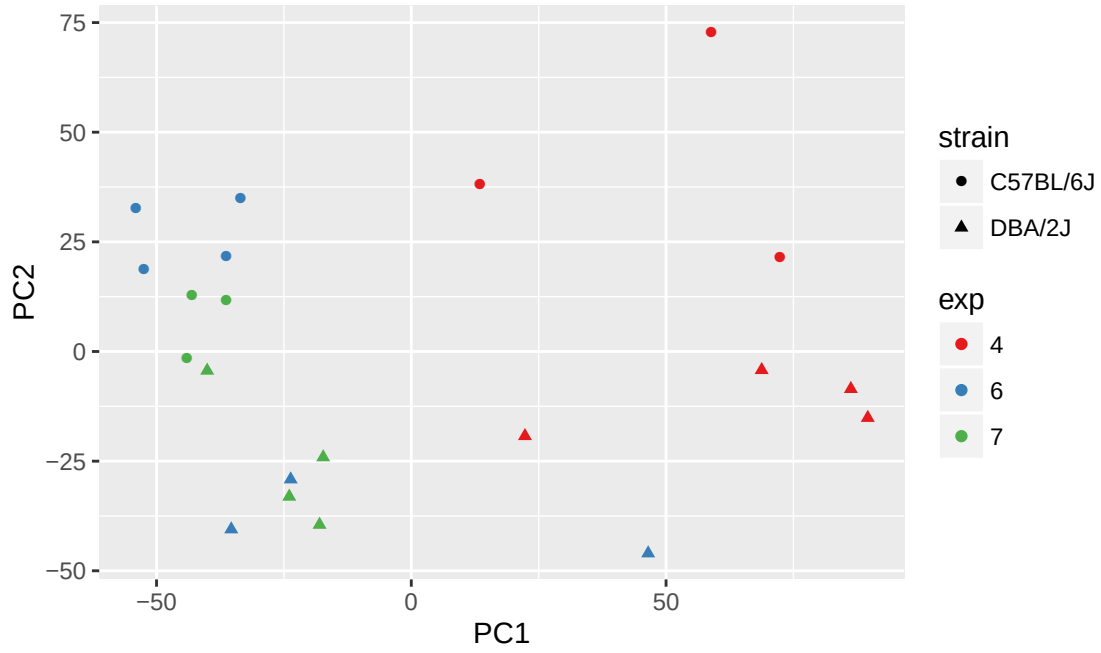
Note that these steps are performed by convenient high level functions when actually using the package *DESeq*. The step by step presentation here is only done to illustrate the concepts.

8 PCA and heatmaps for quality control of high–throughput data

8.1 PCA plot

PCA is also useful for quality control of high–throughput data. In the bottomly data set, we have two mouse strains, a PCA plot should reveal a clustering of the samples according to the strain they belong to since the principal components try to preserve the euclidian distance between the samples.

```
b.PCA = prcomp(t(bottomly.log2), scale = T)
dataGG = data.frame(PC1 = b.PCA$x[,1], PC2 = b.PCA$x[,2], strain
                    = pData(bottomly.eset)$strain,
                    exp = as.factor(pData(bottomly.eset)$experiment.number))
(qplot(PC1, PC2, data = dataGG, color = exp, shape = strain) +
  scale_color_brewer( type = "qual", palette = 6))
```



This is indeed the case. Note that a PCA plot can also be used to spot batch effects, e.g. sample cluster by sequencing center. From the PCA plot we clearlz see that the experimental batch explains more variation than the condition of interest: the two mouse strains.

Samples in batch 4 seem to be very different from the rest of the samples. However, the difference the mouse strains is also clearly visible.

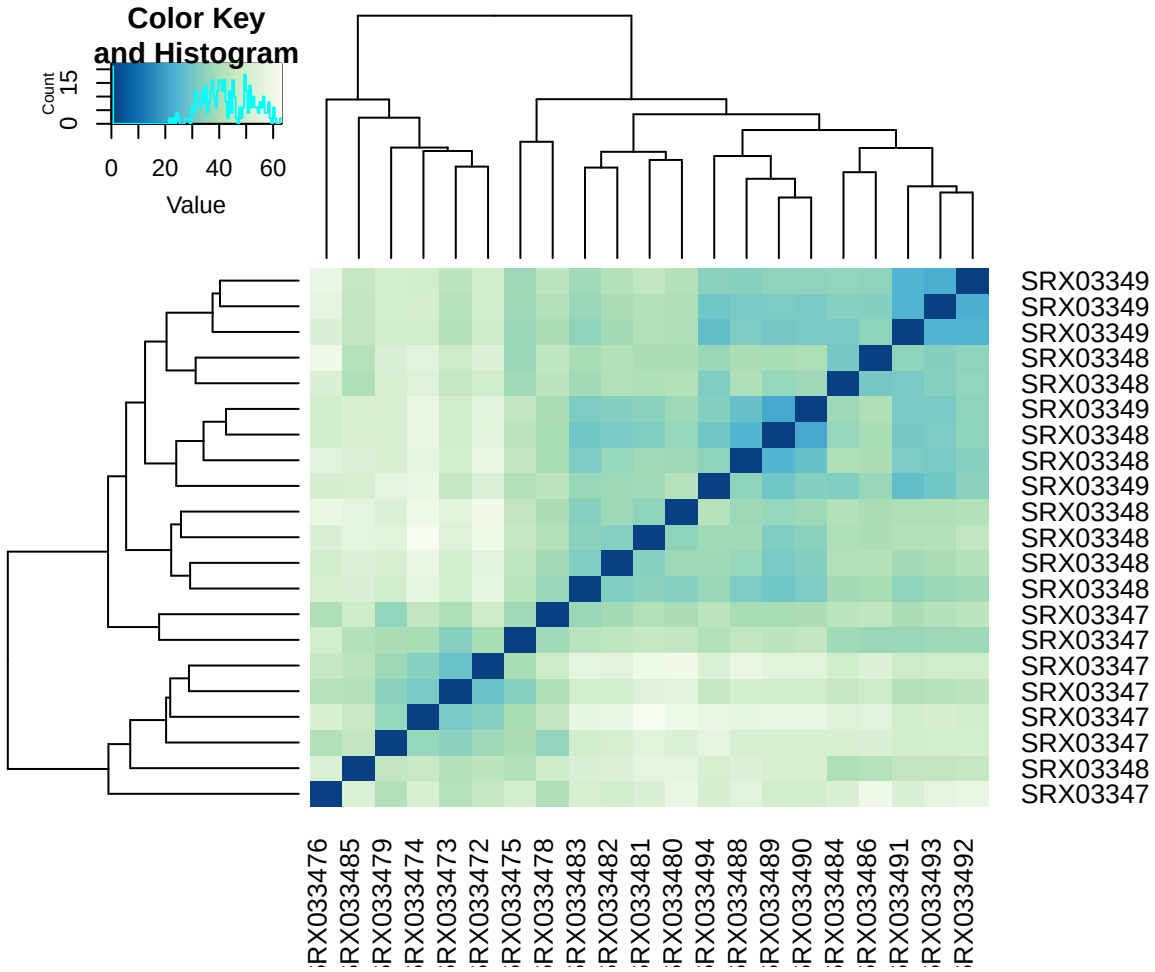
A PCA should always be performed on log–transformed or otherwise variance–stabilized data since otherwise high variance genes will dominate the distance between samples.

8.2 Heatmaps and hierarchical clustering

Another very common visualization technique is a heatmap. Analogous to a PCA we can visualise the pairwise distances between the samples using a heatmap, i.e. a false colour representation of the distance between two samples. Here we use the euclidean distance of all the genes per sample. This can be conveniently computed using the function `dist`.

This heatmap is then ordered via hierarchical clustering. Hierarcical clustering starts with as many clusters as there are samples and successively merges samples that are close to each other. This merging process is commonly visualised as a tree like grapgic called a dendogramm.

```
dists <- as.matrix(dist(t(bottomly.log2)))

hmcol <- colorRampPalette(brewer.pal(9, "GnBu"))(100)
heatmap.2(dists, trace="none", col = rev(hmcol))
```



Again we see that samples (SRX03347x) from experimental batch 7 cluster together. A simple way to correct for this batch effect, is to include the experiment number as a blocking factor in a linear regression model. This will result in comparisons being performed after substracting a batch mean within each experimental batch.

9 Batch effects

One often overlooked complication with high-throughput studies is batch effects, which occur because measurements are affected by laboratory conditions, reagent lots, and personnel differences. This becomes a major problem when batch effects are confounded with an outcome of interest and lead to incorrect conclusions. Here we look at some relevant examples of batch effects and discuss: how to detect, interpret, model, and adjust for batch effects.

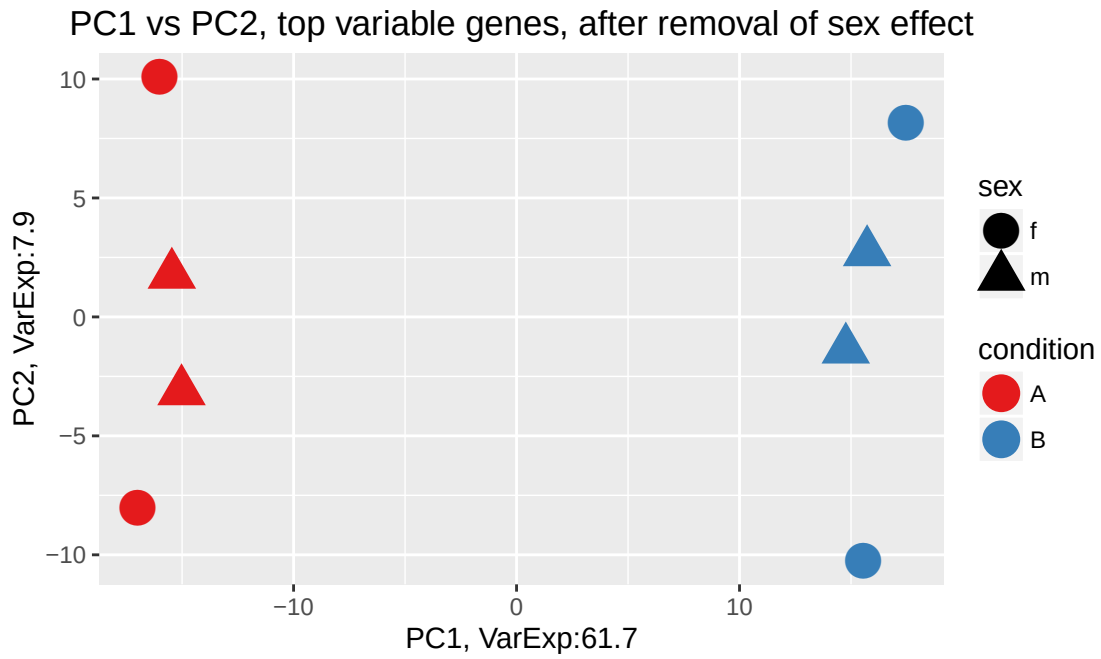
Batch effects are the biggest challenge faced by genomics research, especially in the context of precision medicine. The presence of batch effects in one form or another have been reported among most, if not all, high-throughput technologies for a short review see the paper by Leek et al.

9.1 Removing known batches

As we have seen above, we have a sex–related batch effect in our simulated RNA–Seq data. We saw a clear clustering by sex. A simple way to remove known batches is to fit a regression model that includes the batches to the data, and then substract the coefficients that belong the batch effects. We can use the function `lmFit` from the *limma* to fit a linear model to each gene of our simulated data simultaneously and then simply substract the estimated coefficients that belong

to the batch effects from the data. The function `removeBatchEffect` in *limma* implements this strategy. We specify a batch effect and then a design matrix that incorporates the effects that we do not want to remove (the experimetnal condition in this case).

```
cleaned_data <- removeBatchEffect(rldSim,
                                  batch = as.character(colData(dds)$sex),
                            design =  model.matrix( ~ colData(dds)$condition))

pc_vars<- rowVars(cleaned_data)
selected_vars <- order(pc_vars, decreasing = TRUE)[seq_len(min(ntop,
                                                    length(pc_vars)))]
PCA <- prcomp(t(cleaned_data[selected_vars, ]), scale. = TRUE)
perc_var <- round(100*PCA$sdev^2/sum(PCA$sdev^2),1)


dataGG = data.frame(PC1 = PCA$x[,1], PC2 = PCA$x[,2],
                    PC3 = PCA$x[,3], PC4 = PCA$x[,4],
                    sex = colData(dds)$sex,
                    condition = colData(dds)$condition)

(qplot(PC1, PC2, data = dataGG, color =  condition, shape = sex,
       main = "PC1 vs PC2, top variable genes, after removal of sex effect", size = I(6))
+ labs(x = paste0("PC1, VarExp:", round(percentVar[1],4)),
       y = paste0("PC2, VarExp:", round(percentVar[2],4)))
+ scale_colour_brewer(palette = "Set1")
)
```



As we can see, the sex effect has been removed. The function `ComBat` from the *sva* implements a more sophistaced method for the removal of known batches that adjusts the variances in a addition to removal of batch specific means. For additional details see e.g. this page.

9.2 Tackling "unknown" batches and other unwanted variation

In order to illustrate latent batch effects we will look at the *stockori* data set. It contains genotype information for several plant samples collected in different countries. We also have a variable `Floweringtime` indicating the length of flowering. This data has been provided by Oliver Stegle (EBI).

9.2.1 Overview of the Stockori floweringtime dataset

- 697 individuals (plants)
- 149 genotypes
- Floweringtime
- Country for each plant

9.3 Importing the data

We first load the data and then split the input data into genotypes (coded by 0, 1, 2 corresponding to homozygous reference, heterozygous alternative and homozygous alternative respectively).

We then center and scale the genotype variables for the upcoming analyses. This is a common preprocessing step.

```
stockori <- read.csv("http://www-huber.embl.de/users/klaus/BasicR/stockori.csv")
stockori[sample(dim(stockori)[1],5), 1:10]
```

```
          country flowering flowering_binary geno geno.1 geno.2 geno.3 geno.4
153        Sweden        73                1    2      0      0      0      0
338 Tadjikistan        32                0    0      0      2      2      0
257     Germany        30                0    0      0      2      2      0
250     Germany        24                0    0      0      0      0      0
293       India        43                1    0      0      0      0      0
    geno.5 geno.6
153      2      0
338      0      2
257      0      0
250      0      0
293      0      2
```

```
stockori_gtypes <- scale(as.matrix(stockori[, -seq(1,3)]))
stockori_anno <- stockori[, seq(1,3)]
stockori_anno$flow = factor(stockori_anno$flowering_binary,
                                        labels=c("short", "long"))
```

9.4 PCA Analysis of the stockori data

We calculate the PCA and and a singular value decomposition (more on that later) of the data.

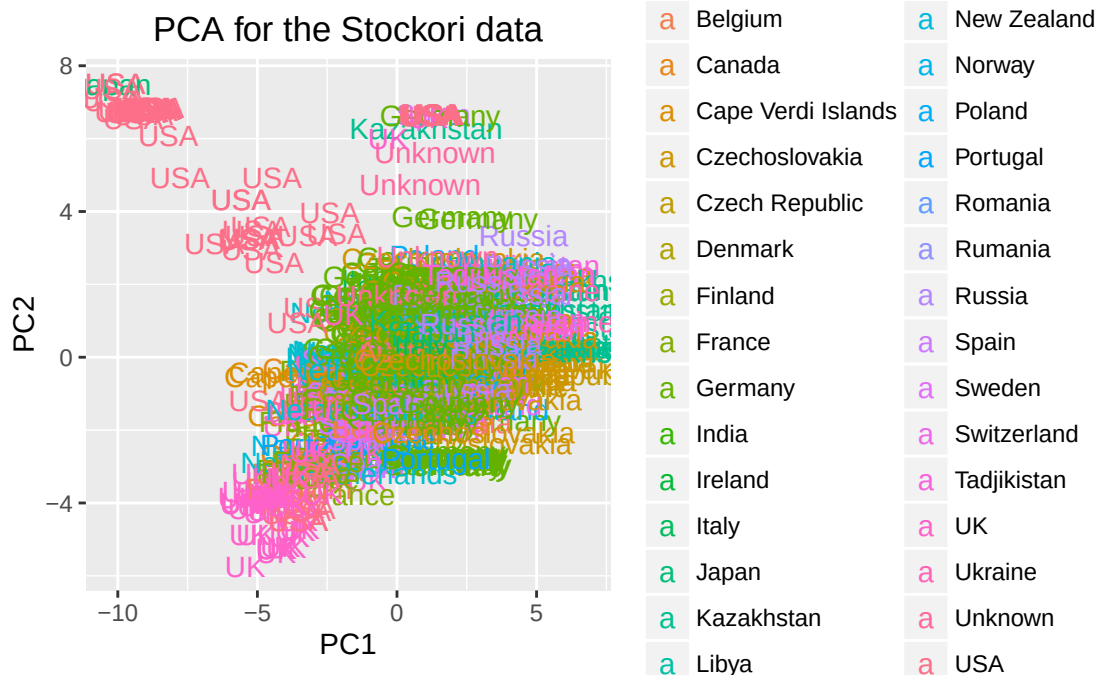
```
stock_PCA = prcomp(stockori_gtypes, center = F, scale = F)
stock_SVD <- fast.svd(stockori_gtypes)
```

We now plot the PCs and color the points by origin. We take the country annotation and simplify it a little bit: Looking at the PCA plot, we can clearly see a strong clustering of the plants by their origin. This is not surprising, since plants cannot move easily, so their genotypes are expected to be strongly associated with their country of origin.

One of the pioneering papers discussing these issues and correcting for this bias is Price et. al. -Principal components analysis corrects for stratification in genome-wide association studies** - Nature Genetics, 2006

They called the PCs "axis of variation". Below, we plot a bar chart of the countries of origin and the principal components. From the inspection of the plot, we can see that the first axis of variation differentiates between central Europe and the rest of the world, while the second one separates the US from the UK.

```
#table(stockori_anno$country)

  pal_func <- function(n){
  rep(brewer.pal(12, name ="Set3")[-9],(n %/% 11 + 1))[seq_len(n)]
  }
  cls = pal_func(length(levels(stockori_anno$country)))


country =  stockori_anno$country
country = ifelse(country %in% c("Cape Verdi Islands","India",
                  "Kazakhstan", "Lybia", "New Zealand",
                  "Tadjikistan", "Unknown", "USA"), "RestOfWorld", "Europe" )

country[stockori_anno$country == "Germany" ] <- "Germany"
country[stockori_anno$country == "UK" ] <- "UK"
country[stockori_anno$country == "USA" ] <- "USA"
country[stockori_anno$country == "Russia" ] <- "Russia"


ggBar <- qplot(stockori_anno$country, fill = stockori_anno$country)


dataGG <- data.frame(PC1 = stock_PCA$x[,1], PC2 = stock_PCA$x[,2],
                    PC3 = stock_PCA$x[,3], PC4 = stock_PCA$x[,4],
                    PC5 = stock_PCA$x[,5], PC6 = stock_PCA$x[,6],
                    country = stockori_anno$country,
                    flowering = stockori_anno$flow)

qplot(PC1, PC2, data = dataGG, label = country, geom = "text", color = country,
      main = "PCA for the Stockori data")
```



9.5 Singular value decomposition (SVD)

To understand the EIGENSTRAT procedure proposed by Price et. al. we first need to have an idea about the SVD. If we denote our genotype matrix by $\mathbf{Y}$, we can compute a decomposition of it as follows:

$$\mathbf{Y_{n\times p}} = \mathbf{U\Sigma V^T}$$

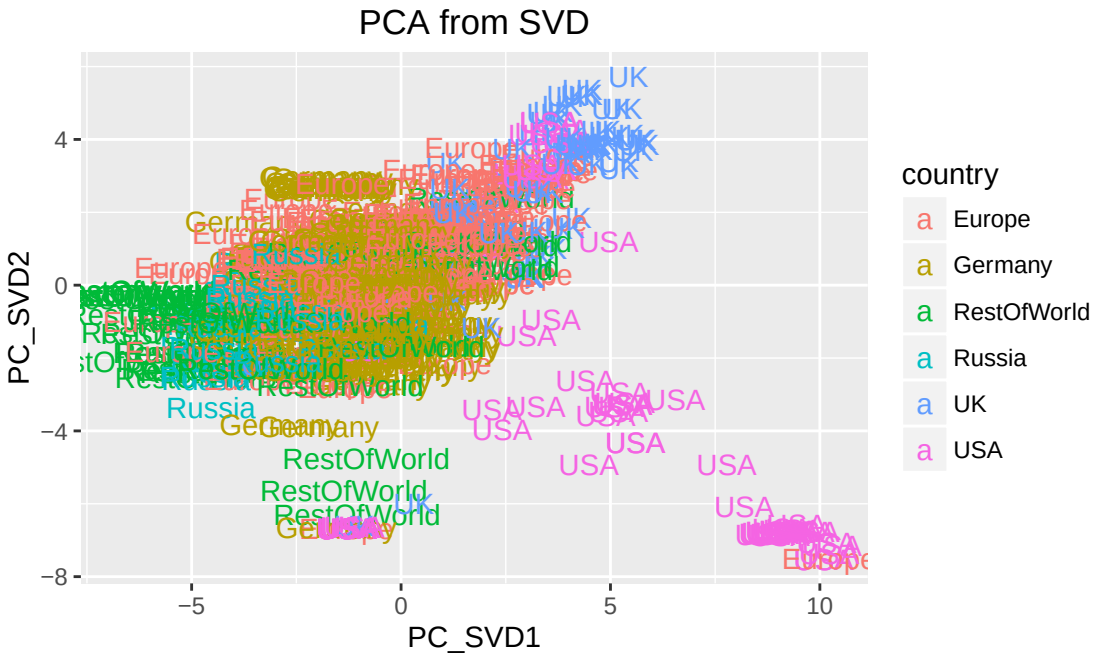
This is called the singular value decomposition of $\mathbf{Y}$. If $\mathbf{Y}$ has n rows (the samples) and p columns (the variables) then U has dimension $n \times min(n,p)$ and V has dimension $p \times min(n,p)$. The columns of $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, i.e. $\mathbf{U^TU} = I$ and $\mathbf{V^TV} = I$. $\mathbf{\Sigma}$ is a diagonal matrix containing the so-called singular values. The SVD is commonly used to compute the PCA. In fact, the matrix $\mathbf{V} = (v_1, \ldots, v_p)$ contains the **loadings** and the matrix $U\Sigma$ contains the **principal components** or **principal component scores** since:

$$\mathbf{Y_{n\times p}V} = \mathbf{U\Sigma}$$

Note that $\mathbf{V}$ contains actually the eigenvectors of the correlation matrix of $\mathbf{Y}$ proportional to $\mathbf{Y^TY}$ if the variables centered and scaled. Hence, the *loadings* are sometimes referred to as **eigenvectors**.

For illustrative purposes, we create a PCA from the SVD.

```
PCA_from_SVD <- stockori_gtypes %*% stock_SVD$v

dataGG = data.frame(PC_SVD1 = PCA_from_SVD[,1], PC_SVD2 = PCA_from_SVD[,2],
                    PC_SVD3 = PCA_from_SVD[,3], PC_SVD4 = PCA_from_SVD[,4],
                    PC_SVD5 = PCA_from_SVD[,5], PC_SVD6 = PCA_from_SVD[,6],
                    country = country, flowering = stockori_anno$flow)

qplot(PC_SVD1, PC_SVD2, data = dataGG, label = country, geom = "text",
      color = country, main ="PCA from SVD")
```



Note that the PCs are uniquely determined up to their sign, which can be chosen in an arbitrary way, so that the PCA plot created from the SVD is a mirrored version of the original one in this case.

9.6 How can we use SVD to correct for batch effects?

We have measurements for p genotypes and n samples in a matrix $Y_{n\times p}$. Suppose we suspect that a batch effect is responsible for most the variability. We know that some samples fall in one batch and the rest in an other, but we don't know which sample is in which batch. Can we discover the batch? The columns of $\mathbf{V}$ are very helpful for this ...

If we assume that many genotypes will have a different average in one batch compared to the other this will introduce a high genotype variation, thus we can rephrase this problem as searching for a **latent genotpye variable** having **maximal variance**. This variable should then capture most of the batch effect.

In fact there is a mathematical result that $Y_{n\times p}v_1$ will have maximum variance among all vectors v satisfying $v^Tv = 1$. Thus subtracting the first principal component from the data $Y_{n\times p}$ like this:

$$r_{n\times p} = Y_{n\times p} - Y_{n\times p}v_1v_1^T$$

should remove a great deal of the unwanted variation. In the formula above we substract a rank 1 approximation to the data matrix from the original data.

Then using the the vector v_2 results in the most variable vector $r_{n\times p}v_2$ and so on. Thus, an iterative subtraction of principal components from the data will correct for batch effects through the removal of **latent variables**.

However, note that we did not take the flowering indicator into account here, e.g. we might remove a lot of the differences between the two groups of plants as well by simply subtracting principal components. The algorithms EIGENSTRAT and SVA try to deal with this.

9.7 The EIGENSTRAT procedure

The idea of the EIGENSTRAT procedure is to use a certain number of k standardized principal components in a regression model as **latent variables**. Let the p th column (the p th genotype) denoted by Y_p the matrix of inferred orthonormal

principal components or "**axes of variation**" by $\mathbf{U}$ and their l th column by u_l. Then the following regression model is fit to each genotype p separately:

$$Y_p = \beta(p)_0 + g\beta(p)_1 + \sum_{l=1}^{k} u_l r(p)_l$$

Here g is the flowering indicator (e.g. 0 for short and 1 for long) and $\beta_0(p)$, $\beta_1(p)$ and r_l are regression coefficients to be estimated. $\beta_1(p)$ corresponds to the difference in genotype between short and long flowering plants.

Note that by estimating the the coefficients $r(p)$, we take the **signal of interest** $\beta(p)_0 + g\beta(p)_1$ into account. We call them 'rotation' in the following code and set the number of components to 10.

```
no_comp <- 10

mod <- model.matrix(~flow, data=stockori_anno)
modEig <- cbind(mod, stock_SVD$u[,seq_len(no_comp)])
stock_EIG_rotation <- lmFit(t(stockori_gtypes),
                                modEig)$coefficients[,-c(1,2)]

stock_EIG_adj  <- stockori_gtypes - tcrossprod(stock_SVD$u[,seq_len(no_comp)],
                                               stock_EIG_rotation)
## alternative way of computation
stock_EIG_adj <- stockori_gtypes -  Reduce("+", lapply(seq_len(no_comp),
         function(k){tcrossprod(stock_SVD$u[,k], stock_EIG_rotation[,k])}))
```

9.8 The Surrage Variable Analysis (SVA) Algorithm

The EIGENSTRAT procedure has an apparent weakness: if the signal is strong, using the orthonormal principal components as obtained from the SVD as latent variable will remove much of the signal as well. This usually does not seem to be a problem for genetic association studies, were the signal is weak but a method that does not rely on a weak signal is desirable.

The SVA algorithm tries to overcome this weakness: the estimation of the SVD is taking the signal of the model into account, in our case the grouping variable. Full details of the algorithm can be found in the supplement of Leek JT and Storey JD - A general framework for multiple testing dependence- PNAS, 2008

The SVA algorithm also has built in algorithm to estimate the number of latent (surrogate) variables . Interestingly, it doesn't find any in our data set. Thus, we set the number manually.

```
mod0 <- model.matrix(~1, data = stockori_anno)
n_sv <- num.sv(t(stockori_gtypes), mod, method="leek")
n_sv # does not find anything
```

```
   [1] 0
```

```
svaobj <- sva(t(stockori_gtypes), n.sv = no_comp, mod, mod0, method="irw")
```

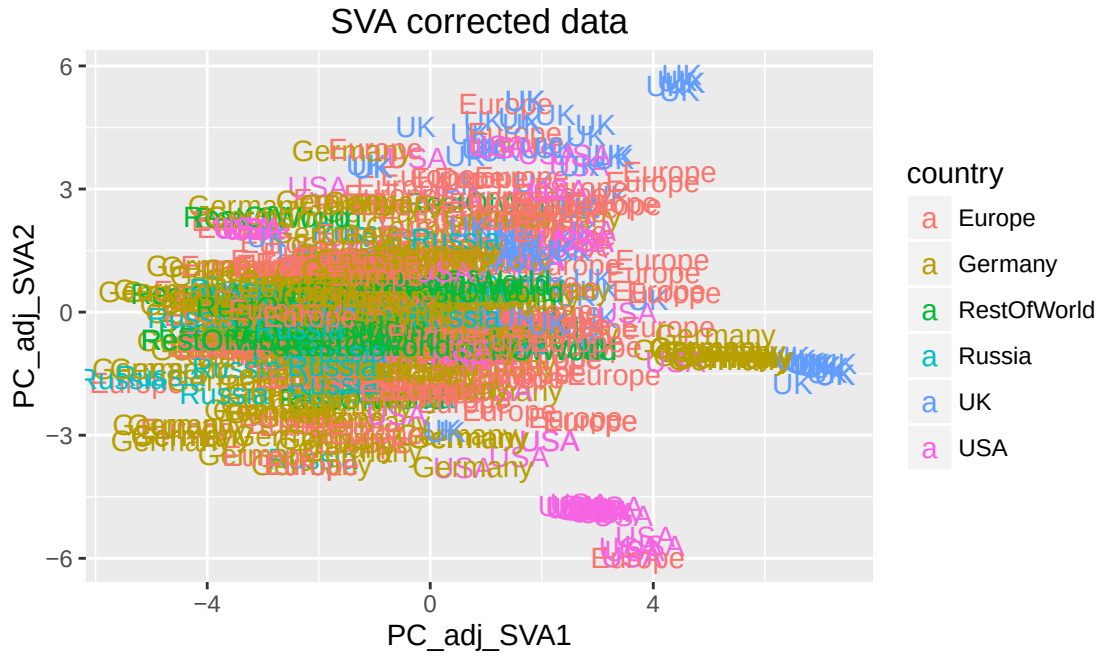
```
   Number of significant surrogate variables is:  10
   Iteration (out of 5 ):1  2  3  4  5
```

```
modSVA <- cbind(mod,svaobj$sv)
stock_sva_rotation <- lmFit(t(stockori_gtypes),
                                modSVA)$coefficients[,-c(1,2)]
stock_sva_adj <- stockori_gtypes - tcrossprod(svaobj$sv, stock_sva_rotation)
```

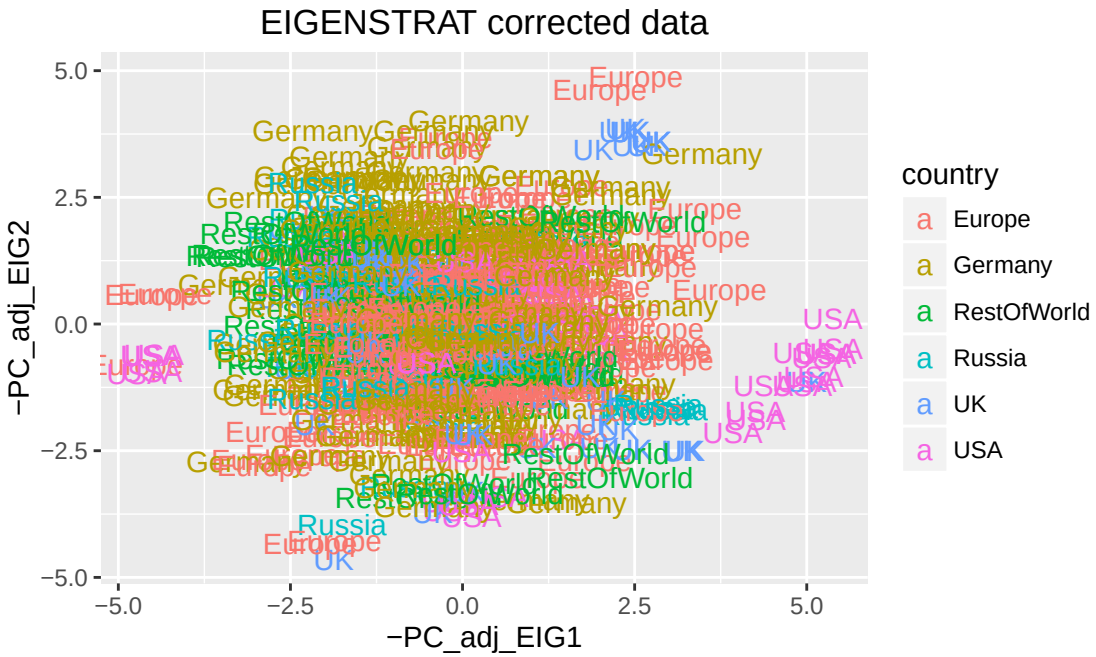
9.9 Compare corrections from EIGENSTRAT and SVA

We can now produce a plot of the principal components of the data again to check whether the strong clustering by country of origin has be alleviated.

```
stock_sva_adj_PCA <-prcomp(stock_sva_adj)
stock_EIG_adj_PCA <-prcomp(stock_EIG_adj)


dataGG = data.frame(PC_adj_SVA1 = stock_sva_adj_PCA$x[,1],
                    PC_adj_SVA2 = stock_sva_adj_PCA$x[,2],
                    PC_adj_EIG1 = stock_EIG_adj_PCA$x[,1],
                    PC_adj_EIG2 = stock_EIG_adj_PCA$x[,2],
        country = country, flowering = stockori_anno$flow)
qplot(PC_adj_SVA1 , PC_adj_SVA2, data = dataGG, label = country, geom = "text",
      color = country, main ="SVA corrected data")
```




```
qplot(-PC_adj_EIG1 , -PC_adj_EIG2, data = dataGG, label = country, geom = "text",
      color = country, main ="EIGENSTRAT corrected data")
```

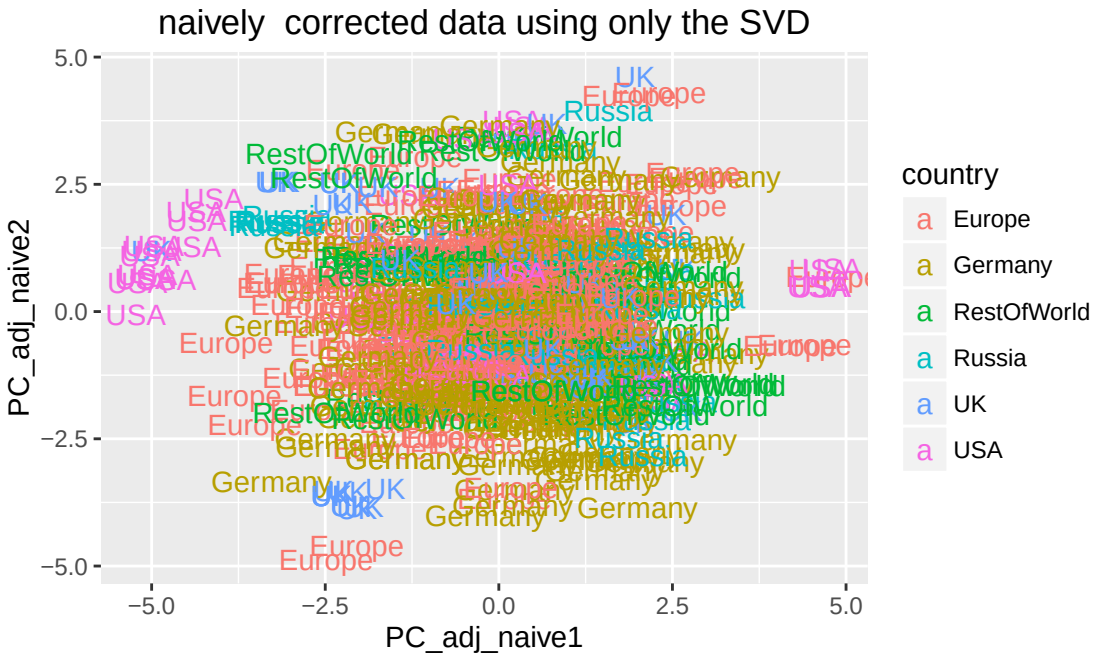


As we can see, both EIGENSTRAT and SVA are successful in removing the country dependency.

9.10 The "naive" correction

The naive correction for batch effects consists in simply removing principal components from the data without taking the signal into account. This is also quite often performed in real data analysis. We also check how this works out for our data set.

```
stock_naive_adj <- stockori_gtypes -
  Reduce("+", lapply(seq_len(no_comp),
         function(k){stock_SVD$d[k]*tcrossprod(stock_SVD$u[,k],
                                               stock_SVD$v[,k])}))
stock_naive_adj_PCA <-prcomp(stock_naive_adj)


dataGG = data.frame(PC_adj_naive1 = stock_naive_adj_PCA$x[,1],
                    PC_adj_naive2 = stock_naive_adj_PCA$x[,2],
        country = country, flowering = stockori_anno$flow)
qplot(PC_adj_naive1 ,PC_adj_naive2, data = dataGG, label = country, geom = "text",
      color = country, main ="naively  corrected data using only the SVD")
```



While e.g. SVA seems to preseverve some outliers (3 samples from Germany) the naive method removes a lot more variation and leads to a complete loss of any clustering.

9.11 Best practices and caveats in batch effect removal

While the batch effect removal methods often seem to be magic procedure, they have to be applied carefully. In particular, using the cleaned data directly can lead to false positive results and exagarrated effects.

The best thing is avoid the use of the cleaned data directly (except for visualisation purposes) and rather include the estimated latent variables or batch effects into a linear model. Two recent papers disucss these caveats for known and unknown batches respectively.

- Nygaard et. al. - Methods that remove batch effects while retaining group differences may lead to exaggerated confidence in downstream analyses - 2015
- Jaffe et. al. -Practical impacts of genomic data cleaning on biological discovery using surrogate variable analysis - 2015

Additional remarks

- Most of the literature on batch effect removal looks a t $p \times n$ matrix, while we used a $n \times p$ notation throughout since this is the natural input for a PCA.
- Actually, SVA fits a multivariate regression model to obtain the rotation matrix, while in the above code we fit a model to each genotype separately. However, the authors of the SVA package suggest using the latent variables in a limma analysis, so this should be okay.
- If we consider the latent variables as random, it can be seen that they introduce a correlation structure, this roughly corresponds to the factor analysis model and explains why Leek and Story call the latent component of the model a dependence kernel. Mike Love and Rafael Irizarry provide an elaboration on this point here.

10 Answers to Exercises

Exercise: Normalization strategies

(*a*) Create pairwise MA plots for the raw data on the log2 scale. Can you spot samples that a very different from each other?

(*b*) An alternative to size factor normalization is to just devide the counts of every sample by its sums divided by 10^6. This is sometimes called counts per million normalization. Compare this to the size factor normalization by creating appropriate plots.

Solution: Normalization strategies

```
#a
################################################################################
# Pairwise MA plots for raw data

pdf("pairwiseMAsBottomlyRaw.pdf")
 MA.idx = t(combn(1:21, 2))

  for( i in  1:dim(MA.idx)[1]){
         MDPlot(log2(exprs(bottomly.eset)),
                c(MA.idx[i,1],MA.idx[i,2]),
        main = paste( sampleNames(bottomly.norm)[MA.idx[i,1]], " vs ",
         sampleNames(bottomly.norm)[MA.idx[i,2]] ))
        }
dev.off()

#b
################################################################################
bottomly.cS <- colSums(exprs(bottomly.eset)) * 1e-6
multidensity( t(t(exprs(bottomly.eset)) /bottomly.cS) , xlab="mean counts", xlim=c(0, 500),
           main = "Bottomly total sum normalized")
```

Exercise: PCA based quality control

Produce a PCA plot, where you also mark the lane instead of experimental batch only (e.g. using size). What do you observe?

Solution: PCA based quality control

```
b.PCA = prcomp(t(bottomly.log2), scale = T)
dataGG = data.frame(PC1 = b.PCA$x[,1], PC2 = b.PCA$x[,2], strain
                    = pData(bottomly.eset)$strain,
                    exp = as.factor(pData(bottomly.eset)$experiment.number)
                    ,lane = as.factor(pData(bottomly.eset)$lane.number))
qplot(PC1, PC2, data = dataGG, color = exp, shape = strain)
```