Cancer methylome analysis CLL vs. B-Cells

Leona Brandl, Carlotta Brueggen, Tim Kuehn, Violetta Schaaf

21st July 2019

The aim of this project is to find genes which differ in their methylation level for cancer cells derived from chronic lymphocytic leukemia (CLL) patients compared to B-cells derived from healthy patients. Therefore, we analyze methylation data of five CLL patients and five healthy patients, looking for differentially methylated regions which we then examine for statistical and biological significance in carcinogenesis and especially inspect genes which according to literature are connected to CLL pathogenesis. Chronic lymphocytic leukemia is the most common form of leukemia in adults and can be characterized as the uncontrolled proliferation of B-lymphocytes. Since their ability to undergo apoptosis is inhibited, these functionally incompetent B-cells accumulate in the blood vessels and displace healthy blood cells. Consequences of this development are increased vulnerability for infections, a lower oxygen level in the blood and longer bleeding periods in case of injury. As CLL originates from B-cells, we use these for comparison in our analysis. It is proved that aberrant methylation has an effect on oncogenesis and there is evidence that this might also be the case for CLL. The most common form of methylation is the conversion of a cytosine to 5-methylcytosine by the addition of a methyl group in fifth position of the nucleobase by the DNA methyltransferase. Methylation is highly relevant in epigenetics since it has an effect on the transcription of genes and may lead to gene silencing. In CLL, a global hypomethylation compared to healthy cells has been observed, with possible consequences being activation of gene expression as well as genomic instability. Moreover, hypermethylation of gene promotors was frequently detected, which might lead to inhibition of promotor activity and affect the transcription of tumor suppressor genes. The methylation data we work with was obtained by bisulfite sequencing, which is a library strategy providing information on the degree of DNA methylation. In this technique DNA is treated with bisulfite upon amplification with PCR, resulting in the conversion of unmethylated cytosines into thymines while methylated cytosines are protected from modification. Afterwards, it is possible to calculate the beta value, which is defined as the ratio of methylated cytosines to the total number of cytosines in the genome, ranging between 0 and 1.

The analysis will be done in several steps:

1. **Data processing**

- Reorganisation of the data
- Quality control concerning NAs, coverage values and unusefull DNA-regions

2. **Normalisation**

- Preparing beta-values
- Transformation of beta-values into M-values

3. **Dimensionality reduction**

- PCA
- Batch effect analysis

4. **Clustering**

- Via K-means clustering

5. **Identifying DMRs**

- t-test
- p-value correction
- Statistical and biological relevance
- Hypo- and hypermethylation

6. **Logistic regression**

7. **Data evaluation**

- Relevant genes
- Interpretation

1) Data processing

Before we can start analysing our data set it has to be tidied up to make it easy for us to work with it by making it as clear as possible. Datasets have to be split up and reorganized, coloums removed or renamed and a quality control concering coverage values, beta values, NAs and specific DNA segments showing unusual values, needs to be performed.

Reading in the datasets

We start by **reading in our CLL-Bcell-data and the annotations** which is an extra document explaining where our data is coming from.

```
input_data <- readRDS(file ="CLL-Bcells_list.RDS.gz")
annotation <- read.csv("sample_annotation.csv")
```

Reorganisation of the data

Our main dataset, the one containing the data we are working with, is `input_data`. It contains methylation data in form of beta values and coverage values of B-cells of five healthy patients and five patients suffering from CLL. It is divided into four subgroups: tiling, genes, promoters and CpG islands. We have to analyse those subgroups separatly, because methylation in promoters, genes and CpG islands can have different biological reasons and also this way the data the computer deals with is not that much while running the algorithm. Therefor **the dataset gets divided into four subdatasets**.

```
genes <- input_data$genes
promoters <- input_data$promoters
cpgislands <- input_data$cpgislands
tiling <- input_data$tiling
```

Those subgroups datasets contain methylation data for different regions on the genome: `genes`, `promoters` and `cpgislands`contain genome segments with specific functions and methylation features. Genes, especially highly expressed genes, sometimes show methylation in their gene body. Unfortunately the biological function behind this is not fully clear yet. Still it can be interesting investigating in their methylation pattern in terms of cancer. In promoters we already know that methylation plays an important role, because as promoters are the starting point for transcription, their methylation can silence genes laying downstream of these promoters. This way for example tumor suppresor genes don't get expressed any more, which can lead to cancer. Here we expect to find crucial differences in methylation between sick and healthy cells. CpG islands also are important regions on the DNA with a high density of cytosines in neighbourhood to guanines. These CpG islands are often in promoter regions of housekeeping genes and in healthy beeings only rarely methylated, because their methylation usually leads to mistakes in replication and transcription and quickly in death just because functioning promoters are so essential in those genes. Also in those CpG islands a loss of function can lead to cancer, so we expect significant methylation differences. The fourth subgroup is `tiling` which describes a specific section on the genome in steps of 5000 nucleotides without a biological context.

As we found genes influencing the formation of CLL in papers, which we wil mention later on, we will concentrate on the subgroup `genes` In this `genes`dataset we **rename the columns**, which stand for different patients (patient 1 to patient 10). They get short precise names to let the reader get the meaning on the first sight.

```
colnames(genes)[11:30] <- c("P1_healthy_beta","P2_healthy_beta","P3_healthy_beta",
                            "P4_healthy_beta","P5_healthy_beta","P6_CLL_beta","P7_CLL_beta",
                            "P8_CLL_beta","P9_CLL_beta","P10_CLL_beta","P1_healthy_coverage",
```

```
                          "P2_healthy_coverage","P3_healthy_coverage","P4_healthy_coverage",
                          "P5_healthy_coverage","P6_CLL_coverage","P7_CLL_coverage","P8_CLL_coverage",
                          "P9_CLL_coverage","P10_CLL_coverage")
```

Quality control

It is important to remove values that are not needed for or even hinder further analysis. Among other values those are also coverage values, which seem to be unrealistically high due to PCR dublicates or repetetive regions. Also they can be that low, so the belonging beta-value of the DNA section has no meaning, because there was no fitting allignment when mapping back PCR fragments onto the genome. A **data frame only containing coverage values** is created.

```
cov_genes <- genes[,c(1,21:30)]
```

At this point we already delete some genes, which are the genes on the X- and Y-chromosome. Somehow there must have been problems in obtaining the beta and coverage values for those chromosomes, because the beta-values are either not a number (NaN) or mainly 1.000, which could be caused by technical source of error. The belonging coverage is in most cases 0, so we can not use those beta-values values anyway. More likely, the X-chromosome has to be removed due to the fact of its inactivation in women. Since only one X chromosome is allowed to be active in women, as otherwise it could lead to complications, one must be inactivated (since women have two X chromosomes). This is done by methylation of an X chromosome and binding of an inactivating RNA. Since this methylation can not be related to possible cancer, but has a natural cause, one must remove these chromosomes out of the data. If the cause of the cancer were on the Y-chromosome, only men would be affected. Since this is not the case, this chromosome must also be removed from the record.

Source: (Sharp et al., 2011)

Sharp, A.J., Stathaki, E., Migliavacca, E., Brahmachary, M., Montgomery, S.B., Dupre, Y., and Antonarakis, S.E. (2011). DNA methylation profiles of human active and inactive X chromosomes. Genome research 21, 1592-1600.

Removing genes on X- and Y-chsomosome

Following code **removes the X- and Y-chromosome**.

```
cov_genes_new <- cov_genes[-which(cov_genes =="chrX"),]
cov_genes <- cov_genes_new[-which(cov_genes_new =="chrY"),]
rm(cov_genes_new)
cov_genes <- cov_genes[ ,c(2:11)]
```

Finding a coverage value threshold

Concerning the very high or low coverage values: there need to be a threshold defining a senseful coverage value range. To determine a threshold we have a look at the coverage value distribution among all patients using a logarithmic density plot of the coverage means of all patients for each genome segment. Therefore a **matrix containing the means** in generated.

```
cov_genes_means <- rowMeans(data.matrix(cov_genes))
```

A **logarithmic density plot** using this matrix values is generated. To **find an upper threshold** for deleting coverage values we try quantile values used in literature and draw the lines for those quantiles in the logarithmic plot to see whether one of those coincidences with a kink in the curve. For the **lower threshold** we draw lines at the values 10 and 15.

```
plot(density(log10(cov_genes_means)), xlab = "Coverage means", main = "Coverage distribution")
quantile(cov_genes_means, probs = c(.95))
```

```
##     95%
## 53811.9
```

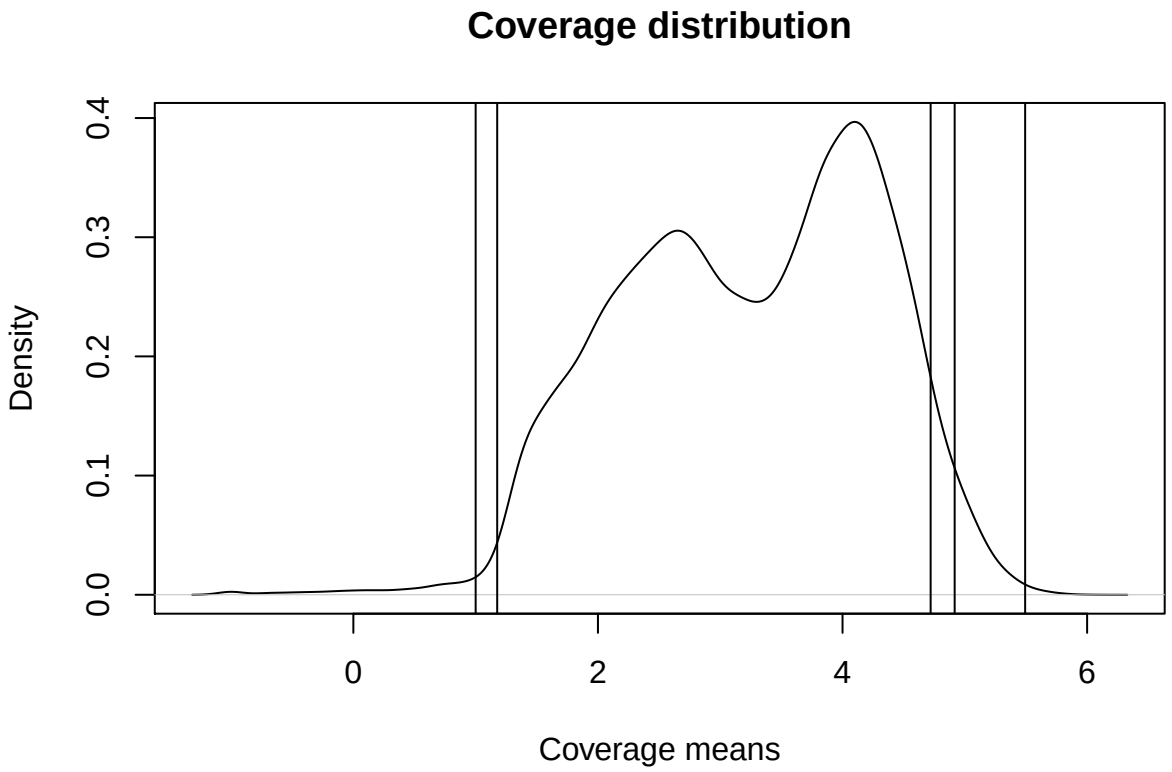
```
abline(v=log10(52529.75))
quantile(cov_genes_means, probs = c(.975))
```

```
##    97.5%
## 84118.43
```

```
abline(v=log10(82556.55))
quantile(cov_genes_means, probs = c(.999))
```

```
##    99.9%
## 313215.5
```

```
abline(v=log10(311230.4))
abline(v=log10(10))
abline(v=log10(15))
```



Both lower thresholds and the highest upper threshold seem to be fitting looking at the diagram. But which threshold we will chose eventually also depends on the percentage of deleted rows at the end of quality control. We decided not to delete more than 15 % of the rows, because we don't want to risk to lose reliable information.

The procedure will now be used to set all the coverage values in `cov_genes` that are below or above the chosen threshold to NA. Then all the beta-values associated to those NA coverage values are also turned into NA. Eventually we will set another threshold for the beta-values, defining the maximum number of NA that each row can have in order not to be deleted. In order to do this, we used two nested for-loops.

Preparing the first nested for-loop

Before applying a threshold to the coverage values, the few **NAs among the coverage values are set to 0** in order to allow the following for loops to work. These values will be set to NAs again after the loop, since

0 is below the lower threshold.

```
cov_genes[is.na(cov_genes)] <- 0
```

As the lower threshold we choose the coverage value 15, which is positioned at a kink of the previously showed coverage distribution diagram. As the upper coverage threshold we choose the 90 % quantile. Although it is not positioned at a kink in the diagram we consider it to be a good threshold, because it still only leads to a removal of 7 % of the genes at the end of the quality control, which is far underneath the chosen upper limit of 15 %.

```
threshold1 <- 15
threshold2 <- quantile(cov_genes_means, probs = seq(0.90,0.90,0.05))
```

The first nested for-loop

Now a **nested for-loop** is used to set the coverage values underneath the lower threshold and above the upper threshold to NA. This will be done for each row and each column of the dataframe `cov_genes`.

```
for(i in 1:nrow(cov_genes)){
  for(j in 1:ncol(cov_genes)){
    if(isTRUE(cov_genes[i,j] <= threshold1)){
      cov_genes[i,j] <- NA
    }
    if(isTRUE(cov_genes[i,j] >= threshold2)){
      cov_genes[i,j] <- NA
    }
  }
}
rm(i,j,threshold1,threshold2)
```

Preparing the second nested for-loop

A second nested for-loop is used to set the beta-values to NA that correspond to the NA coverage values. A **dataframe containing only beta-values** is created and the **genes located on the X- or Y-chromosome are removed**.

```
beta_genes <- genes[,c(1,11:20)]
beta_genes_new <- beta_genes[-which(beta_genes =="chrX"),]
beta_genes <- beta_genes_new[-which(beta_genes_new =="chrY"),]
rm(beta_genes_new)
beta_genes <- beta_genes [ ,c(2:11)]
```

The second nested for-loop

This **second nested for-loop** checks every row and every column of the dataframe `cov_genes`. If there is an NA, it sets the associated beta-value in `beta_genes` to NA, too.

```
for(k in 1:ncol(beta_genes)){
  for(l in 1:nrow(beta_genes)){
    if(isTRUE(is.na(cov_genes[k,l] == TRUE))){
      beta_genes[k,l] <- NA
    }
  }
}
rm(k,l)
```

Finding a threshold for NAs among the beta-values

Now the second threshold in quality control needs to be chosen. The threshold establishes how many NA are allowed per gene in order for the gene not be completely deleted. Since some beta-values were set to NA in the last step, we want to get insight in **how many NAs we actually have now**.

```
rmv.rows_beta_genes = apply(beta_genes,1, function(x){sum(is.na(x))})
```

A new dataframe containing only beta-values for genes which are below the NA threshold is created. We chose a threshold of 3 allowed NAs per row (gene), so **every row containing more than 3 NAs gets removed**.

```
beta_genes_cleaned <- beta_genes[-which(rmv.rows_beta_genes >2),]
```

To see **how many rows are deleted** due to quality control the numer of rows before and after quality control is compared. We additionally have a look at the **percantage of rows deleted** and see whether it is beneath 15 %.

```
row_difference = nrow(genes)-nrow(beta_genes_cleaned)
sum(row_difference)
```

```
## [1] 3915
```

```
genes_deleted_percentage = row_difference/nrow(genes)*100
sum(genes_deleted_percentage)
```

```
## [1] 6.961856
```

With 7% we are beneath the recommended upper limit of 15 % of deleted rows.

2) Normalisation

The tidy beta-values we have now are an approximation of the percentage of DNA-methylation. Biologically beta-values are easy to understand. They range from 0 to 1, where 0 means unmethylated and 1 means fully methylated. But their bounded range is also a disadvantage. It leads to the problem that they can not be used for statistical tests like the t-test, because they violate the Gaussian distribution. For highly methylated gene regions and unmethylated gene regions beta-values are very heteroscedastic, which means, that the variability of a variable is unequal across the range of values. Therefore we have to normalise them. Beta-values need to be transformed into M-values via a logit transformation. M-values do not have a bounded range, they can be infinite big and small. While the middle methylation range (beta-value range of 0,2-0,8) is linear to M-values, outside this range the beta-values are compressed and the M-values more accurate and homoscedastic. This is why M-values can be used for statistical tests and are absolutely necessary for further analysis.

Preparing beta-values for logit transformation

Logit transformation turns the extreme beta-values 0 and 1 into -infinite and infinite. Further tests cannot be computed if some values are equal to -infinite and infinite. Because of this, we **set 0 to a very small number higher than 0, and 1 to a very big number smaller than 1**.

```
beta_genes_cleaned[beta_genes_cleaned==0]<-0.00000001
beta_genes_cleaned[beta_genes_cleaned==1]<-0.99999999
```

Creating two seperate data frames for sick and healthy patients

In order to perform our normalisation, we **split our data into beta-values of healthy and diseased patients.**

```
beta_genes_healthy <- beta_genes_cleaned[,c(1:5)]
beta_genes_cancer <- beta_genes_cleaned[,c(6:10)]
```

Now we are able to **replace the remaining NA's by the row means of the different genes.**The rowmeans is different between healthy and sick patients so it is important to work with the two separated dataframes.

```
k <- which(is.na(beta_genes_healthy), arr.ind=TRUE)
beta_genes_healthy[k] <- rowMeans(beta_genes_healthy, na.rm=TRUE)[k[,1]]
l <- which(is.na(beta_genes_cancer), arr.ind=TRUE)
beta_genes_cancer[l] <- rowMeans(beta_genes_cancer, na.rm=TRUE)[l[,1]]
```

Transforming beta-values into M-values

In the next step we **transform our beta-values into M-values** so we are able to do statistical tests with the data. We calculate the M-values with the following equation:

$$M = log_2(\frac{beta}{1 - beta})$$

```
M_genes_healthy <- log2(beta_genes_healthy/(1-(beta_genes_healthy)))
M_genes_cancer <- log2(beta_genes_cancer/(1-(beta_genes_cancer)))
```

For applying the PCA on our data we have to build a **dataset which contains sick and healthy patients** with **short column names**.

```
M_genes <- cbind(M_genes_healthy, M_genes_cancer)
cat <- M_genes[1:10, 1:10]
colnames(M_genes) <- c("1H","2H","3H","4H","5H","6CLL","7CLL","8CLL","9CLL","10CLL")
```

3) Dimensionality reduction

To **reduce our data** we use Principal Component Analysis. PCA reduces a large set of variables into a smaller one which still contains most of the information of the larger dataset. Through the PCA, correlated variables are turned into a much smaller number of uncorrelated variables which are called principal components. The goal is to have a smaller number of variables in order to make the computation easier.

PCA

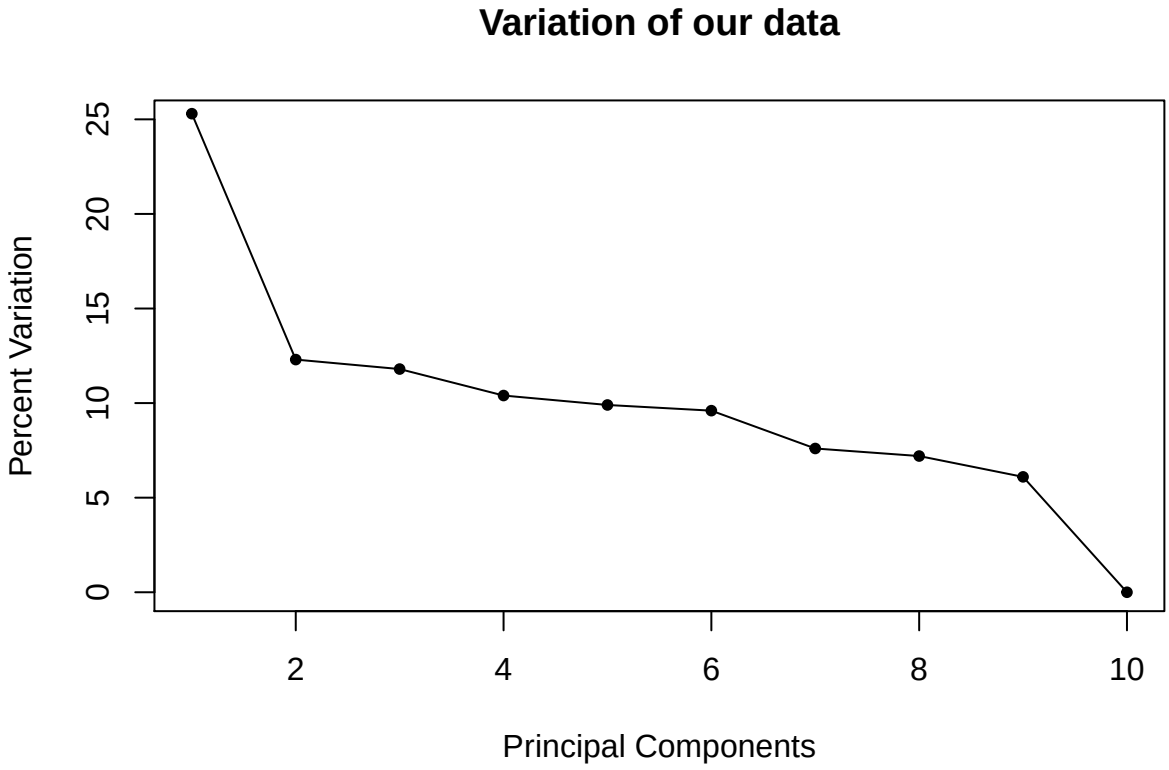
The code for the **PCA** requires the samples (patients) to be in the rows and the genes to be in the columns. In our data the patients are columns and the genes are rows so we have to **transpose the matrix** with the t() function.

```
pca_M <- prcomp(t(M_genes))
```

How much variation does each component account for?

In order to have a better overview on the obtained principal components, we study how much variation in percentage each principal component accounts for. We plot the percentage for each component and decide with the **elbow method** with how many PC we have to work with.

```
var_pca <- pca_M$sdev^2
var_pca_per <- round(var_pca/sum(var_pca)*100, 1)
plot(var_pca_per, main="Variation of our data", xlab="Principal Components",
     ylab="Percent Variation", ylim=c(0,25),type = "o", pch=20)
```

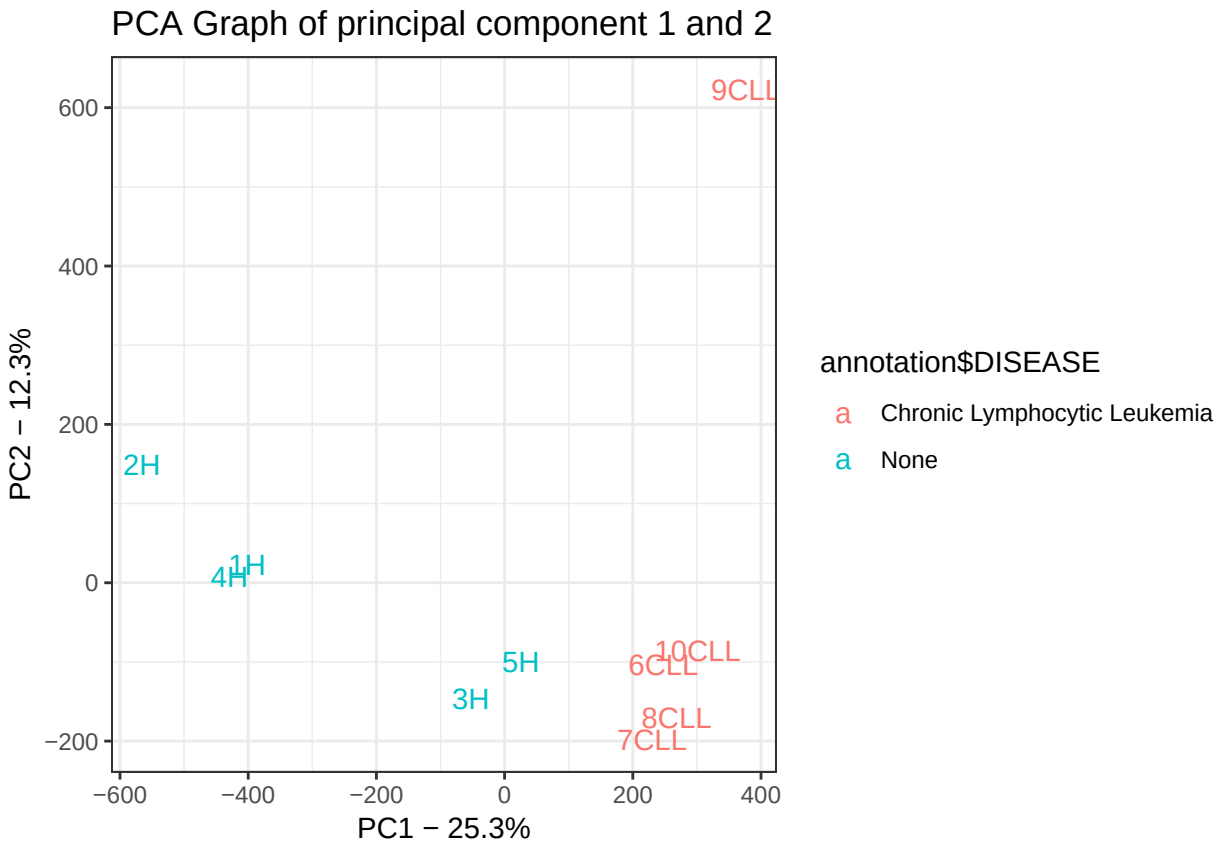


In the plot we can not see a significant elbow. Based on the visualization, we would actually have to continue the downstream analysis with ten principal components. But as the last PCs don't explain much variation anyway and are therefore useless for us, we continue the analysis on the first five PCs.

Drawing a 2D plot of component 1 and 2

First, we check whether the control and disease sample separate into two separate groups based on the first principal components. Therefore, we plot principal component 1 against principle component 2.

```
library(ggplot2)
pca_values <- data.frame(Sample=rownames(pca_M$x),
                         X=pca_M$x[,1],
                         Y=pca_M$x[,2])
ggplot(data=pca_values, aes(x=X, y=Y, label=Sample)) +
  geom_text(aes(colour = annotation$DISEASE)) +
  xlab(paste("PC1 - ", var_pca_per[1], "%", sep="")) +
  ylab(paste("PC2 - ", var_pca_per[2], "%", sep="")) +
  theme_bw() +
  ggtitle("PCA Graph of principal component 1 and 2")
```



Investigation for batch effect

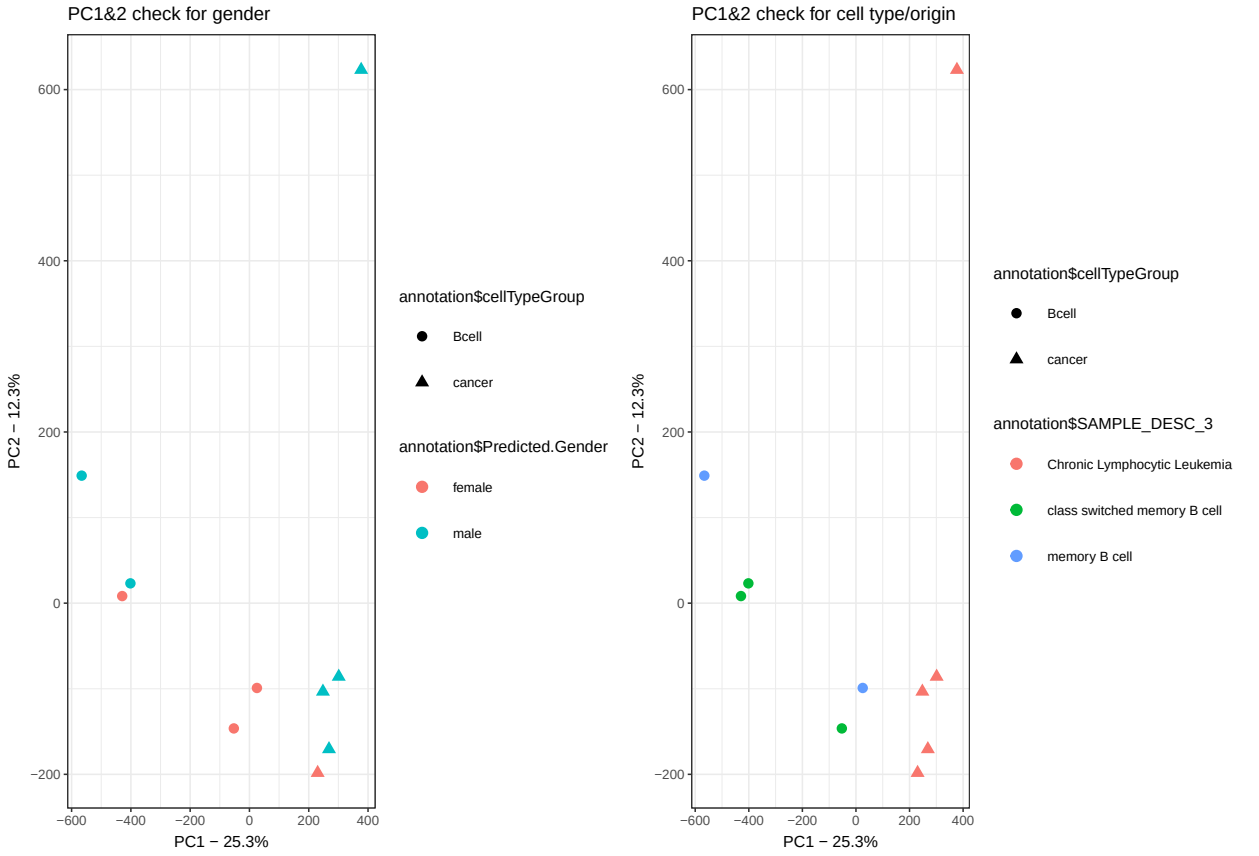
Using different categories of the annotation table makes it possible to check for a potential **batch effect**. Batch effect is a **technical source of error**. It is important that the technical variation does not confound with the biology to not falsify the interpreted data. Therfore it is important to check for a batch effect to see whether the variance explained by the principal components is related to technical source of error or by biological influence.

First, principal components one and two were examined for the batch effect by distinguishing different categories by different colors and shapes. The categories to be examined were “gender”, “sample desc”, “biomaterial provider” and “age”. However, the age is to be viewed critically, since depending on the age, different levels of methylation may be present and could mistakenly be taken as a batch effect, although it is a biological origin.

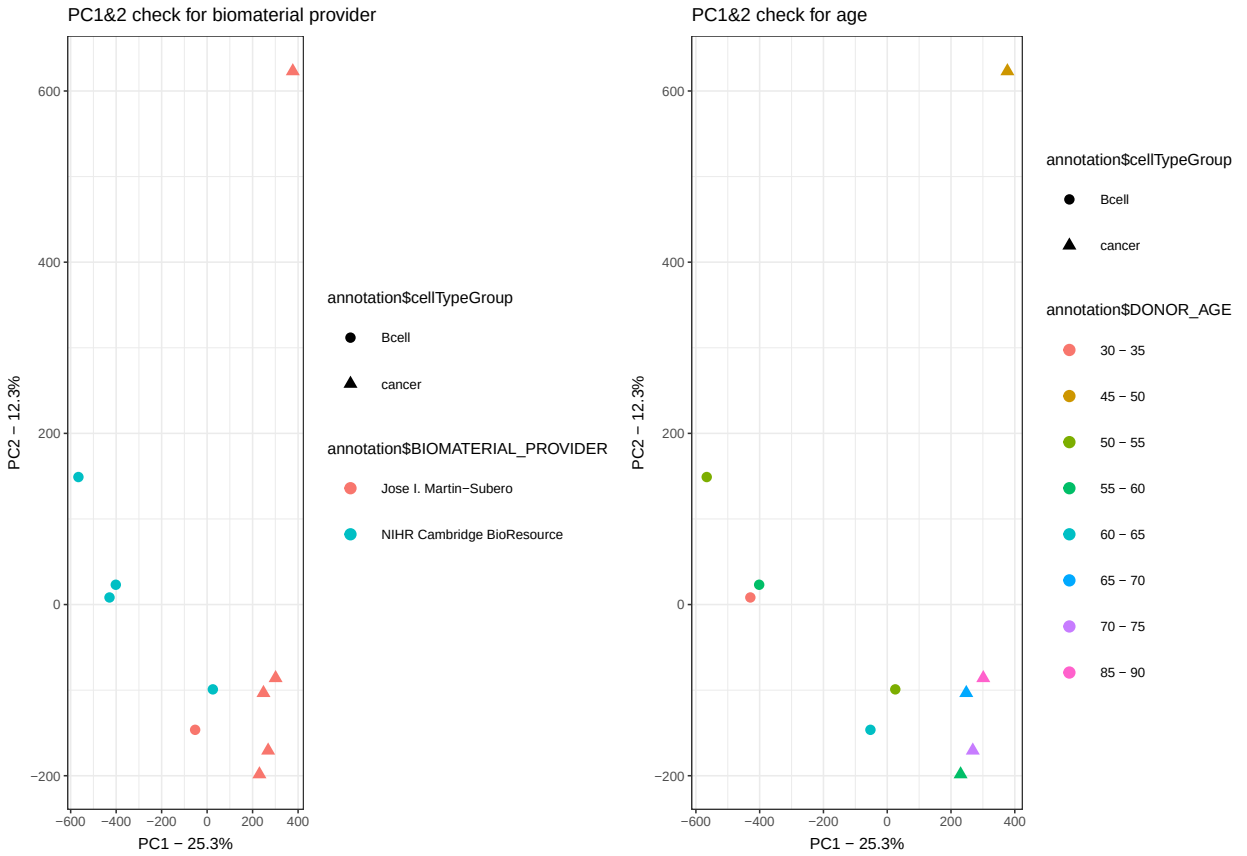
By plotting PC one and two with the mentioned categories, it is possible to visually investigate for possible batch effects. If all data points of one variable are close to each other and they are not well separated by the principal component, this could indicate a batch effect in the respective components.

```
ggplot_1 <- ggplot(data=pca_values, aes(x=X, y=Y, label=Sample)) +
  geom_point(aes(shape=annotation$cellTypeGroup, color=annotation$Predicted.Gender)) +
  xlab(paste("PC1 - ", var_pca_per[1], "%", sep="")) +
  ylab(paste("PC2 - ", var_pca_per[2], "%", sep="")) +
  theme_bw(base_size = 6) +
  ggtitle("PC1&2 check for gender")
ggplot_2 <- ggplot(data=pca_values, aes(x=X, y=Y, label=Sample)) +
  geom_point(aes(shape=annotation$cellTypeGroup, color=annotation$SAMPLE_DESC_3)) +
  xlab(paste("PC1 - ", var_pca_per[1], "%", sep="")) +
  ylab(paste("PC2 - ", var_pca_per[2], "%", sep="")) +
```

```
  theme_bw(base_size = 6) +
  ggtitle("PC1&2 check for cell type/origin")
ggplot_4 <- ggplot(data=pca_values, aes(x=X, y=Y, label=Sample)) +
  geom_point(aes(shape=annotation$cellTypeGroup, color=annotation$DONOR_AGE)) +
  xlab(paste("PC1 - ", var_pca_per[1], "%", sep="")) +
  ylab(paste("PC2 - ", var_pca_per[2], "%", sep="")) +
  theme_bw(base_size = 6) +
  ggtitle("PC1&2 check for age")
ggplot_3 <- ggplot(data=pca_values, aes(x=X, y=Y, label=Sample)) +
  geom_point(aes(shape=annotation$cellTypeGroup, color=annotation$BIOMATERIAL_PROVIDER)) +
  xlab(paste("PC1 - ", var_pca_per[1], "%", sep="")) +
  ylab(paste("PC2 - ", var_pca_per[2], "%", sep="")) +
  theme_bw(base_size = 6) +
  ggtitle("PC1&2 check for biomaterial provider")
library(gridExtra)
library(ggplot2)
grid.arrange(ggplot_1,ggplot_2, ncol=2)
```




```
grid.arrange(ggplot_3,ggplot_4, ncol=2)
```



By having a look at the plotted principal components with regard of the different categories, it seems like **gender** and **age** have no influence on PC one or two, since both graphics show a mixed composition. Looking at **cell type** and **biomaterial provider**, principal component two still does not show abnormalities, whereas principal component one does not separate very well anymore and this might indicate a batch effect.

Checking our PC 1 to 5 for significant batch effect per category

To verify the observations, as visualisation is not an objective way to interprete data, it is important to test if batch effect of different categories is significant for the first five principal components, as they explain most of the data's variance. Therefore, it is necessary to create three dataframes containing PC 1-5 and the categories of the annotation table which want to be investigated. Three dataframes are needed because different tests are performed for different data types.

For numbers the **permutation test** (of Pearson's r correlation coefficient) is used, for two categories the **Wilcoxon test** and for more than two different categories the **Kruskal-Wallis test**.

```
x <- pca_M[["x"]]
pca1_5 <- x[,1:5]
batch_kruskal <- data.frame(pca1_5, annotation$SAMPLE_DESC_3)
batch_wilcoxon <- data.frame(pca1_5, annotation$BIOMATERIAL_PROVIDER, annotation$DISEASE,
                             annotation$Predicted.Gender)
batch_permutation <- data.frame (pca1_5, annotation$SEQ_RUNS_COUNT)
```

Permutation test for numbers

The **permutation test** is a non-parametric test to check, whether two not connected spot checks derive from the same basic total unit. The H0 hypothesis assumes that both spot checks derive from the same basic total unit. The permutation test in the usual form is tailored to the counter hypothesis that the distributions

of the two samples emerge by a shift apart. Often the question is not only if there is a shift, but also how big is this shift. Basically it is possible to check for a batch effect with numbers.

The **permutation test** is used for "SEQ_RUNS_COUNT", because numbers have to be examined.

```
cor.perm <- function (x, y, nperm = 1000)
{
  r.obs <- cor (x = x, y = y)
  P.par <- cor.test (x = x, y = y)$p.value
  #  r.per <- replicate (nperm, expr = cor (x = x, y = sample (y)))
  r.per <- sapply (1:nperm, FUN = function (i) cor (x = x, y = sample (y)))
  r.per <- c(r.per, r.obs)

  P.per <- sum (abs (r.per) >= abs (r.obs))/(nperm + 1)
  return(list(r.obs = r.obs, P.par = P.par, P.per = P.per))
}
x <-batch_permutation$PC1
y <-batch_permutation$annotation.SEQ_RUNS_COUNT
p_PC1_SEQ <- cor.perm(x,y)
x <-batch_permutation$PC2
p_PC2_SEQ <- cor.perm(x,y)
x <-batch_permutation$PC3
p_PC3_SEQ <- cor.perm(x,y)
x <-batch_permutation$PC4
p_PC4_SEQ <- cor.perm(x,y)
x <-batch_permutation$PC5
p_PC5_SEQ <- cor.perm(x,y)
p_SEQ_RUNS_COUNT <- data.frame(p_PC1_SEQ$P.per, p_PC2_SEQ$P.per, p_PC3_SEQ$P.per,
                               p_PC4_SEQ$P.per, p_PC5_SEQ$P.per)
```

Wilcoxon test for 2 categories

Categories "BIOMATERIAL_PROVIDER", "DISEASE" and "Predicted.Gender" were tested for a possible batch effect by using the **Wilcoxon test**, as there are exactly two different results.

By applying a for loop to the created dataset, in this case the first five principal components of patients one to ten together with the categories of investigation, it is possible to create a dataframe, which holds the 15 p values for the three categories.

```
p_values_wilcoxon <- c()
for (j in 1:5){
  for (i in 6:8)  {
    p_value <- wilcox.test(batch_wilcoxon[,j] ~ batch_wilcoxon[,i], data = batch_wilcoxon,
                           exact = FALSE)$p.value
    p_values_wilcoxon <- append(p_values_wilcoxon , p_value)

  }
}
```

Kruskal-Wallis for several categories

Category "sample_desc_3" is invesigated by **Kruskal-wallis test** as it has three different non-numeric resultst.

```
batch_kruskal <- within(batch_kruskal, {
  PC1 <- as.numeric(as.character(PC1))
})
```

```
batch_kruskal <- within(batch_kruskal, {
  PC2 <- as.numeric(as.character(PC2))
})
batch_kruskal <- within(batch_kruskal, {
  PC3 <- as.numeric(as.character(PC3))
})
batch_kruskal <- within(batch_kruskal, {
  PC4 <- as.numeric(as.character(PC4))
})
batch_kruskal <- within(batch_kruskal, {
  PC5 <- as.numeric(as.character(PC5))
})
sample_desc_3_pc1 <- kruskal.test(batch_kruskal$PC1 ~ batch_kruskal$annotation.SAMPLE_DESC_3,
                                  data = batch_kruskal)
sample_desc_3_pc2 <- kruskal.test(batch_kruskal$PC2 ~ batch_kruskal$annotation.SAMPLE_DESC_3,
                                  data = batch_kruskal)
sample_desc_3_pc3 <- kruskal.test(batch_kruskal$PC3 ~ batch_kruskal$annotation.SAMPLE_DESC_3,
                                  data = batch_kruskal)
sample_desc_3_pc4 <- kruskal.test(batch_kruskal$PC4 ~ batch_kruskal$annotation.SAMPLE_DESC_3,
                                  data = batch_kruskal)
sample_desc_3_pc5 <- kruskal.test(batch_kruskal$PC5 ~ batch_kruskal$annotation.SAMPLE_DESC_3,
                                  data = batch_kruskal)
p_SAMPLE_DESC_3 <- data.frame(sample_desc_3_pc1$p.value, sample_desc_3_pc2$p.value,
                              sample_desc_3_pc3$p.value, sample_desc_3_pc4$p.value,
                              sample_desc_3_pc5$p.value)
```

We create a data frame containing **all p values of the categories** obtained.

```
p_DISEASE_t <- as.data.frame(p_values_wilcoxon[c(2,5,8,11,14)])
p_BIOMATERIAL_PROVIDER_t <- as.data.frame(p_values_wilcoxon[c(1,4,7,10,13)])
p_Predicted.Gender_t <- as.data.frame(p_values_wilcoxon[c(3,6,9,12,15)])
p_SEQ_RUNS_COUNT_t <- as.data.frame(t(p_SEQ_RUNS_COUNT))
p_SAMPLE_DESC_3_t <- as.data.frame(t( p_SAMPLE_DESC_3))
total_pvalue <- cbind(p_DISEASE_t,p_BIOMATERIAL_PROVIDER_t,p_Predicted.Gender_t,
                      p_SEQ_RUNS_COUNT_t,p_SAMPLE_DESC_3_t)
```

Giving columns and rows informative names.

```
names(total_pvalue)[1] <- "p_DISEASE"
names(total_pvalue)[2] <- "p_BIOMATERIAL"
names(total_pvalue)[3] <- "p_GENDER"
names(total_pvalue)[4] <- "p_SEQ_RUNS_COUNT"
names(total_pvalue)[5] <- "p_SAMPLE_DESC"
rownames(total_pvalue)[rownames(total_pvalue) == "p_PC1_SEQ.P.per"] <- "PC1"
rownames(total_pvalue)[rownames(total_pvalue) == "p_PC2_SEQ.P.per"] <- "PC2"
rownames(total_pvalue)[rownames(total_pvalue) == "p_PC3_SEQ.P.per"] <- "PC3"
rownames(total_pvalue)[rownames(total_pvalue) == "p_PC4_SEQ.P.per"] <- "PC4"
rownames(total_pvalue)[rownames(total_pvalue) == "p_PC5_SEQ.P.per"] <- "PC5"
```

Preparation for heatmap

To **decide which PC is useful for further analysis**, it is helpful to create a heatmap showing which principle component is affected significantly by a batch effect. All significant p-values are shown in red and non-significant ones in blue. To analyse and compare the p values, the limit of significance is set to be 0.1, as this is a common value for statistical tests. To make visualization easier, values above the significance

threshold are set to 10 and values equal or below the threshold to 1, because it is easiert to deal with two colours.

```
total_pvalue <- data.matrix(total_pvalue)
for (i in 1:ncol(total_pvalue)){
  for (j in 1:nrow(total_pvalue)){
    if(isTRUE(total_pvalue[i,j] > 0.1)){
      total_pvalue[i,j] <- 10
    }
    if (isTRUE(total_pvalue[i,j] <= 0.1)){
      total_pvalue [i,j] <- 1
    }
  }
}
```

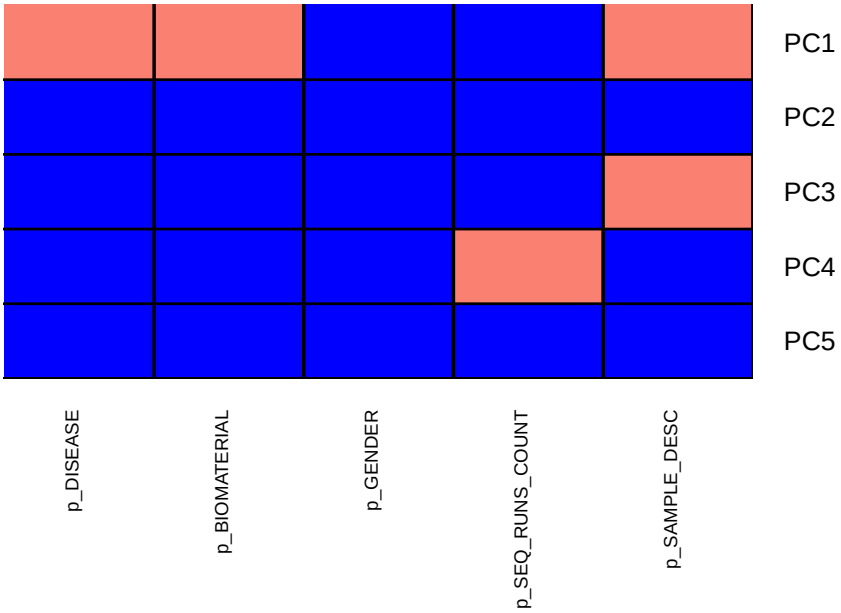
Heatmap

A **heatmap** with the the new values out of the for loop is created.

```
library(gplots)
par(cex.main = 1)

    heatmap.2(
      total_pvalue, main = "Batch effect in our principal components",
      title(main, cex.main = 1 * op[["cex.main"]]),
      margins = c(8,5),
      Colv = NA,
      Rowv = NA,
      dendrogram = "none",
      sepwidth = c(0.01, 0.01),
      sepcolor = "black",
      trace= "none",
      col = colorRampPalette(c("salmon","blue")),
      breaks = c(seq(0, 1, length = 2),
                 seq(1.1, 10, length = 2)),
      colsep = 1:ncol(total_pvalue),
      rowsep = 1:nrow(total_pvalue),
      cexCol = 0.7,
      cexRow = 1,
      key = FALSE)
```

Batch effect in our principal components



By looking at the heatmap or at the p-values respectively, one can see that **principal component 1** has **batch effect** for the categories **sample desc** and **biomaterial provider** (and for **disease**).

To sum it up: Principal component one in general explains more variance than principal component two (Chapter 2: Normalisation). However, the heatmap shows that part of the variance must come from the batch effect and not from dividing the two patient groups in sick and healthy. Looking at the heatmap highlights that principal component one shows batch effect for three categories, or two categories of investigation and a category of controll respectively. The two important categories are **sample desc** and **biomaterial provider**, and the controll category is **disease**. A signigicant difference in the first two categories is not good for the principal component, as it is not clear how much of the variance really is explainable by batch effect or a biological difference. On the other hand, the significance for "disease", as indicated in the heatmap, is a good sign, as this is a controll category and should show significance, since it separates the data according to the criterion which is searched for. Principal component two explains most variance except of component one. By having a look at the heatmap, it shows no significance at all, which is a good sign for the investigated categories, but not for the category of controll respecitvely. Since it is not known how much of the explained variance was caused by the batch effect and not due to biological abuse, **we go on with our analysis using principal component 2** which doesn't show a batch effect at all.

Feature selection

create a dataframe with loading scores of the 10000 most importat genes, which have the most influence for principal component 2.

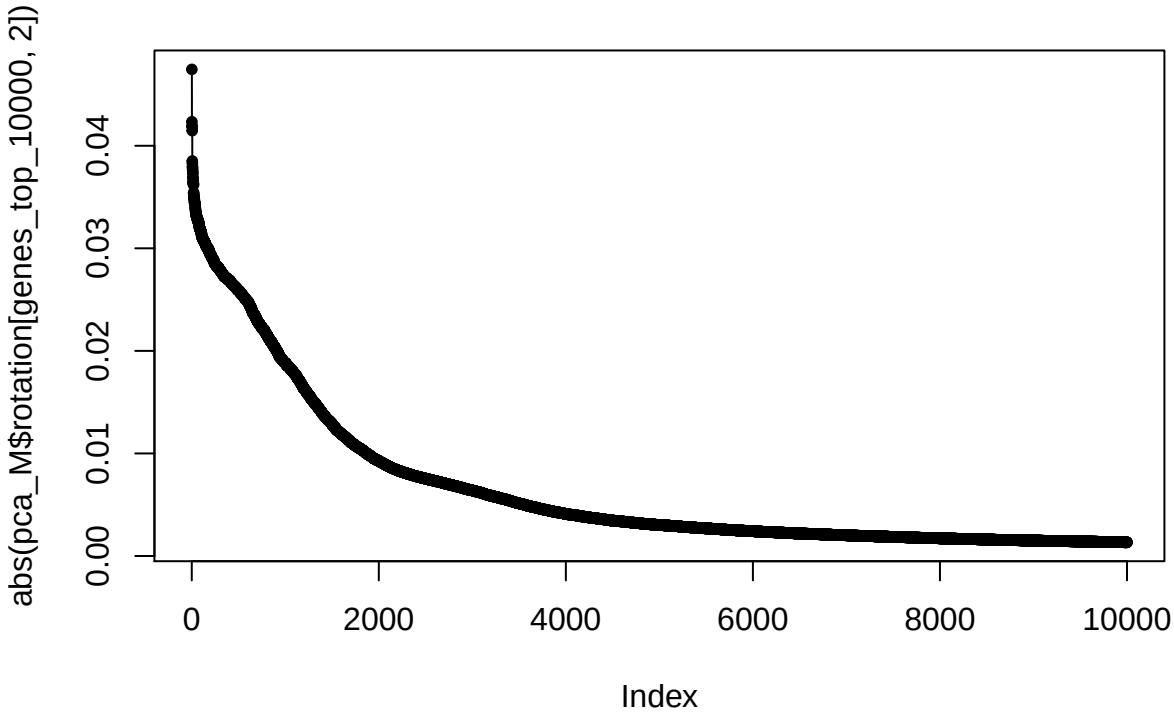
Now we want to compute feature selection, which means that we want to **find the most important genes** for our analysis. We decide to select the genes that have the most influence on the principal component two. For that we use the loading scores which we can find in rotation of the PCA. The loadings describe the weight of each gene in the principal component. If a gene has a high loading it has a high weight in the principal component. A loading can be positive or negative but we will first have a look on the absolut values. We create a dataframe containing the 2000 and 10000 most important genes in principal component 2 in ranked order.

```
loading_scores <- pca_M$rotation[,2]
gene_scores <- abs(loading_scores)
ranked_gene_score <- sort(gene_scores, decreasing=TRUE)
genes_top_10000 <- names(ranked_gene_score[1:10000])
genes_top_2000 <- names(ranked_gene_score[1:2000])
```

Look for elbow in top gene variance

Now we want to **decide how many genes we want to keep for further analysis** by looking for an elbow in the plot of loading scores.

```
plot(abs(pca_M$rotation[genes_top_10000,2]),type = "o", pch=20)
```



Actually there is a kink in the elbow plot of loading scores of the genes at round about 2000 genes. K-means clustering already worked with 2000 genes but by using only 2000 genes we get only 3 significant different methylated genes, which is not as many as expected. By using 10000 genes we get 24 differentially methylated genes, which is an amount easier to work with.

4) Clustering with k-means

K-means is a method to find groups in the data. It uses an iterative algorithm which means that there is a repetition of steps with the aim of finding the right cluster for each datapoint. In the first step the datapoints are split into k clusters (k chosen by the user). Each datapoint is assigned to one of the k clusters. In this dataframe, two centers are selected, as the outcome of sick and healthy is expected. If it was not clear how many centers make sense, an elbow plot could give you an idea. Next step is an updating of the clusters by calculating center of each cluster and define them as new clusterpoints. These two steps will be repeated as long as no datapoint change the belonging cluster. By performing k-means clustering it is possible to **check whether sick and healthy patients are separated in two different clusters**. For this two data sets with the M values of the 2000 and 10000 most important genes from PC2 have to be generated.

```
k_means_data2 <- M_genes[genes_top_2000,]
k_means_data <- M_genes[genes_top_10000,]
```

```r
set.seed(2)

k <-
  kmeans(
    x = t(k_means_data2),
    centers = 2,
    iter.max = 1000
  )

cluster <- data.frame(k[["cluster"]])
cluster
```

```
##       k...cluster...
## 1H                 1
## 2H                 1
## 3H                 1
## 4H                 1
## 5H                 1
## 6CLL               2
## 7CLL               2
## 8CLL               2
## 9CLL               2
## 10CLL              2
```

One can see that the sick and healthy patients belong to two different clusters so it is possible to use principal component two for further testing.

5) Identifying differetially methylated regions (DMRs)

Student's t-test

To see whether the methylation differences, so differences in M-values, between the two Sample groups are significant, **a student t-test is performed**. It can be used because due to the normalisation computed the variances of the values are equal. The test compares the mean M-value of sick patients to the mean M-value of healthy patients and gives a p-value, which is significant or not depending on how strict is the threshold you set.

```r
cluster <- k[["cluster"]]
p_value = NULL
for(i in 1:nrow(k_means_data)){
  x = k_means_data[i, 1:5]
  y = k_means_data[i, 6:10]
  t_test = t.test(x, y)
  p_value[i] = t_test$p.value
}
```

We **create a dataframe wich contains the p-values** from t-test.

```r
pvalues <- data.frame(p_value)
```

To get the gene ID belonging to those p-values we **change the rownames**.

```r
row_names <- row.names(k_means_data)
p_combined <- pvalues
rownames(p_combined) <- row_names
```

Correction of p values by using the holm method

P values are corrected for multiple comparisons. The *holm correction* is applied. Additionally, a *dataframe* containing the uncorrected p-values and the corrected p-values in rising order and the belonging gene ID and symbol is created.

```
p_combined$p_adjusted = p.adjust(p_combined$p_value, method = "holm")
p_holm <- data.frame(p_combined)
symbols <- genes[rownames(p_holm),]
holm <- cbind(p_holm, "Symbols" = symbols$symbol)
holm_new <- holm[order(holm$p_value),]
```

Finding DMRs

If you see the p-values in rising order you can see, that only the first 24 genes are significantly differentially methylated using a threshold of a corrected p-value of 0,1.

```
relevant_genes1 <- holm_new[c(1:24),]
print(relevant_genes1)
```

```
##                      p_value   p_adjusted                    Symbols
## ENSG00000215946 1.175067e-09 1.175067e-05                   MIR941-1
## ENSG00000204802 1.015632e-08 1.015530e-04                       <NA>
## ENSG00000264890 1.376601e-08 1.376326e-04                       <NA>
## ENSG00000199643 1.475274e-08 1.474831e-04                   RNU4-90P
## ENSG00000206921 1.722710e-08 1.722021e-04                  RNU6-481P
## ENSG00000279920 2.285155e-08 2.284012e-04                       <NA>
## ENSG00000252391 3.152707e-08 3.150815e-04                  RNU6-638P
## ENSG00000277437 4.763914e-08 4.760580e-04                  MIR3687-1
## ENSG00000222071 6.322396e-08 6.317338e-04                    MIR1915
## ENSG00000273587 1.331822e-07 1.330624e-03                    SNORA78
## ENSG00000275908 1.661780e-07 1.660119e-03                       <NA>
## ENSG00000277942 2.189029e-07 2.186621e-03                    MIR8075
## ENSG00000207554 3.582260e-07 3.577961e-03                     MIR647
## ENSG00000211837 6.980247e-07 6.971173e-03                     TRAJ53
## ENSG00000278903 7.360642e-07 7.350337e-03                       <NA>
## ENSG00000273981 8.751361e-07 8.738234e-03                       <NA>
## ENSG00000274450 9.378005e-07 9.363000e-03                       <NA>
## ENSG00000216141 1.213785e-06 1.211721e-02 MIR941-2;MIR941-3;MIR941-4
## ENSG00000248979 3.518207e-06 3.511874e-02                  LAMTOR3P2
## ENSG00000278639 4.424737e-06 4.416330e-02                       <NA>
## ENSG00000258577 4.986456e-06 4.976483e-02                    SNRPGP1
## ENSG00000277532 5.631739e-06 5.619912e-02                       <NA>
## ENSG00000207508 9.249911e-06 9.229562e-02                 RNU6-1237P
## ENSG00000252749 9.339817e-06 9.318335e-02                  RNU7-116P
```

Statistical and biological relevant methylation differences in genes

A vulcano plot can help visualising whether a gene is claimed to be differently methylated in healthy and sick patients because of statistical reasons, which means a significant p-value or because of biological reasons, which would be a big difference in the mean of the M-values or beta-values. Only the genes which show both characteristics, really have a significant methylation difference.

Fold change calculation

On the x-axis of the volcano plot there is the **difference of the beta-value means of each sample group**.

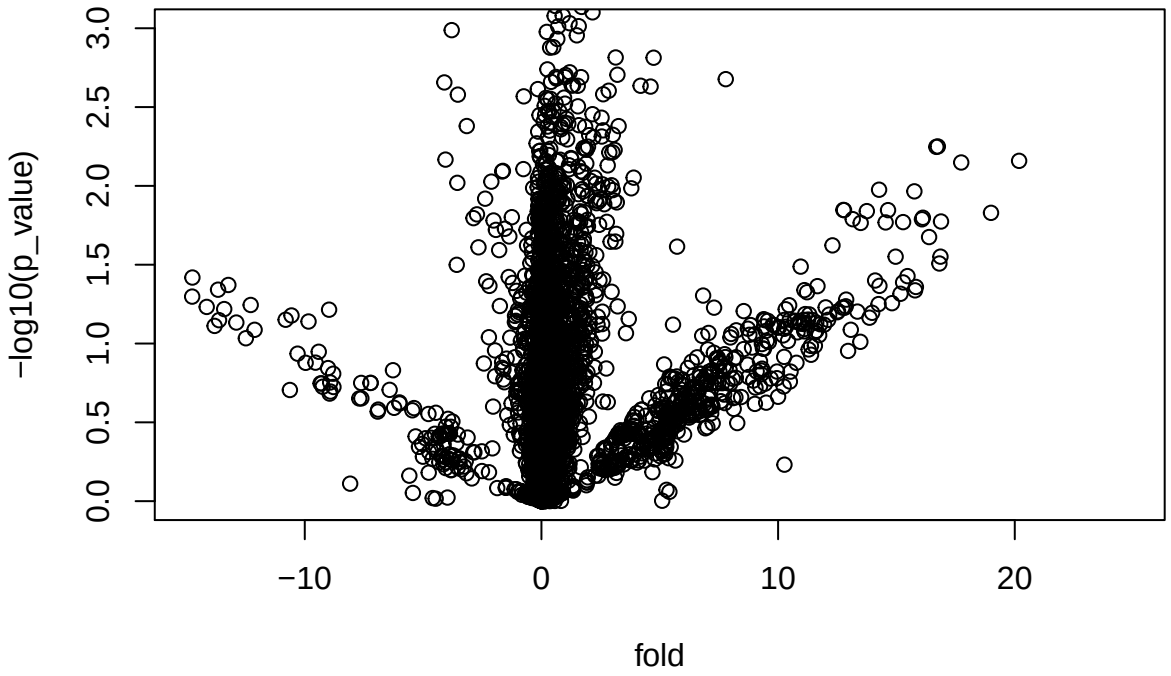
This is the axis showing the biological relevance. Usually for better visualization you use the log2 fold change of the M-values. Due to the negative M-values that exist, this is mathematically not possible. Therefore we do not use the log2 fold change of the M-values, but the one of the beta-values.

```
beta_10000 <- beta_genes_cleaned[genes_top_10000,]
beta_10000_log <- log2(beta_10000)
control <- beta_10000_log[, 1:5]
tumor <- beta_10000_log[, 6:10]
control_mean <- apply(control, 1, mean)
tumor_mean <- apply(tumor, 1, mean)
fold = control_mean - tumor_mean
```

Visualisation including information about hyper- and hypomethylation

The y-axis shows the p-values and therefore the statistical relevance. Actually the corrected p-values should be used, but somehow a lot of p-values were corrected to the value 1, which distorts the plot. Therefor we use the uncorrected p-values for the y-axis and **create a volcano plot**

```
plot(fold, -log10(p_value), ylim = c(0,3))
```



To visualise genes that are statistically and biologically relevant and subdivide them into upmethylated and hypomethylated genes in case of cancer, we first **create thresholds for significance** in p-values and beta-value-difference (fold).

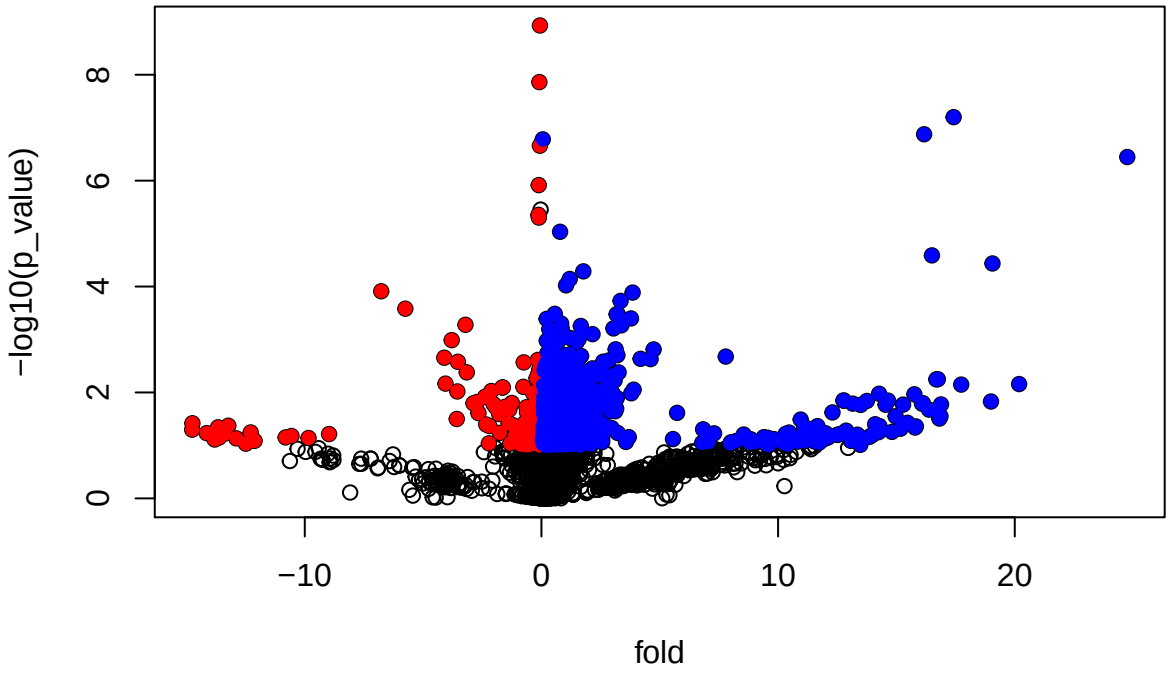
```
threshold_significant = 0.1
threshold_fold = 0.05
```

We create a **filter for biological relevant genes** using the threshold for beta-value-difference, one for **statistically relevant genes** and one that **combines both in one filter**.

```
filter_by_fold = abs(fold) >= threshold_fold
filter_by_pvalue = abs(p_value) <= threshold_significant
filter_combined = filter_by_fold & filter_by_pvalue
```

The genes which fit this filter and are **upmethylated** in cancer cells are coloured in red, while the **hypomethylated** genes are coloured in blue.

```r
plot(fold, -log10(p_value))
points (fold[filter_combined & fold < 0],
        -log10(p_value[filter_combined & fold < 0]),
        pch = 16, col = "red")
points (fold[filter_combined & fold > 0],
        -log10(p_value[filter_combined & fold > 0]),
        pch = 16, col = "blue")
```



6) Logistic regression

Logistic regression is used to model the relationship between a binary outcome variable and one or more predictor variables. In this case, the predictor variables are methylation data of the genes and the outcome is either healthy or sick. Hence we want to **predict a patient's health status by the methylation status of their genes**.

We create a dataframe, which contains M values and health status of 7 random patients. Therefore, the M value dataframe needs to be transformed first. We then add the patients health status. This data frame is used for generating the logistic regression model. We then do cross validation by applying the regression model to a test set data frame and by checking whether the predicted and actual health status match, we can see if the regression model makes works.

```r
log_regression <- k_means_data[,c(1,2,4,5,8,9,10)]
log_regression <- t(log_regression)
log_regression <- data.frame(log_regression)
test_set <- k_means_data[,c(3,6,7)]
test_set <- data.frame(t(test_set))
healthstatus <- annotation$DISEASE
healthstatus <- data.frame(healthstatus)
healthstatus_regression <- healthstatus[c(1, 2, 4, 5, 8, 9, 10),]
train_set <- cbind(healthstatus_regression, log_regression)
regression_model <- glm(healthstatus_regression ~ ., family = "binomial", data = train_set)
summary_regression_model <- summary(regression_model)
summary_regression_model$coefficients
```

```
##                    Estimate Std. Error       z value  Pr(>|z|)
## (Intercept)     -68.3042527  272581.35 -0.0002505830 0.9998001
## ENSG00000275217   4.6430226   17166.33  0.0002704727 0.9997842
## ENSG00000280145  25.3715551   98344.01  0.0002579878 0.9997942
## ENSG00000227979 -31.7522839  157443.93 -0.0002016736 0.9998391
## ENSG00000276242   2.3408763   14049.26  0.0001666192 0.9998671
## ENSG00000271650  -0.8263456   43984.03 -0.0000187874 0.9999850
## ENSG00000272312   1.4226377    4825.18  0.0002948362 0.9997648
```

We only get estimate values for the genes shown above whereas the values for the remainig genes are NA and therefore not shown in the table. This is because of multicollinearity, which means that one or more predictor variables are highly correlated. The estimated value explains the influence of the predictor variable on the outcome variable. Positive values indicate that a gene contributes to health whereas genes with negative values promote disease. It is notable that we get high standard errors, which implies that the regression model is not very reliable. These high standard error values probably result from mulitcollinearity.

```
prediction <- predict(regression_model, newdata = test_set, type = "response")
prediction
```

```
##           3H         6CLL         7CLL
## 3.012131e-10 2.220446e-16 2.220446e-16
```

```
levels(train_set$healthstatus_regression)
```

```
## [1] "Chronic Lymphocytic Leukemia" "None"
```

Since we only have very few sample available, the prediction of the logistic regression model based on the train set containing 70% of our data does not correspond with the actual health status. Even though the predicted probability is 10^6 times higher for the healthy patient than for the ones with CLL, it is still very close to 0, which implies that the patient is probably sick. Therefore we use all data available to model logistic regression without performing cross validation. In this way, more samples are available to train the model.

```
k_means_data_no_cv <- t(k_means_data)
k_means_data_no_cv <- data.frame(k_means_data_no_cv)
full_data <- cbind(healthstatus, k_means_data_no_cv)
model_no_cv <- glm(healthstatus ~ ., family = "binomial", data = full_data)
prediction_no_cv <- predict(model_no_cv, newdata = full_data, type = "response")
summary_model_no_cv <- summary(model_no_cv)
summary_model_no_cv$coefficients
```

```
##                    Estimate Std. Error       z value  Pr(>|z|)
## (Intercept)     -62.3090250 123717.301 -5.036404e-04 0.9995982
## ENSG00000275217   5.0612636  10153.185  4.984902e-04 0.9996023
## ENSG00000280145  54.1513644 143983.605  3.760940e-04 0.9996999
## ENSG00000227979 -50.0417969 141092.708 -3.546732e-04 0.9997170
## ENSG00000276242   2.1658625   8097.999  2.674565e-04 0.9997866
## ENSG00000271650  -9.8198190  42406.585 -2.315635e-04 0.9998152
## ENSG00000272312  -0.3410775   7096.218 -4.806469e-05 0.9999616
## ENSG00000265127  -2.1800901   9128.043 -2.388343e-04 0.9998094
## ENSG00000271876   0.1846260   5218.514  3.537904e-05 0.9999718
## ENSG00000278927  -0.9134562   5588.323 -1.634580e-04 0.9998696
```

We now apply the second model to our data set.

```
prediction_no_cv
```

```
##           1H           2H           3H           4H           5H
## 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
```

```
##          6CLL          7CLL          8CLL          9CLL         10CLL
## 2.143345e-11 2.143345e-11 2.143345e-11 2.143345e-11 2.143345e-11
```

The predicted probability to be healthy is 1 for the healthy patients and very close to 0 for the sick patients, so it corresponds with the actual health status. This is the result we expected, since we made the prediction for the data we used to create the model, which is why the model is not reliable. Therefore, logistic regression with this dataset is not possible.

7) Data evaluation

Relevant genes

Only 24 genes are significantly differently methylated using a threshold of a corrected p-value of 0,1. Besides there have been genes, often correlated to CLL, which we found in literature before. **Literature genes**, which we also found in our initial dataset `genes` are those ones.

```
#grep("symbol", genes$symbol, value = TRUE)
```

- TWIST2
- VHL
- ABI3
- ADORA3
- PRF1
- CLLU1
- LPL
- NOTCH1
- SCARF1
- TNF
- CTBP2
- EGR1

The following genes from the literature can be found in our dataset heaped, as they probably belong to a family:

- NFAT
- EBF
- ODC

The **literature genes still existing in the dataframe after the t-test** are those four ones. They don't show a significant p-value, but still we will do some research on them in the interpretation part.

```
#grep("gene_name", holm_new$Symbols, value = TRUE)
```

- ENSG00000048462 -> TNFRSF17
- ENSG00000159958 -> TNFRSF13C
- ENSG00000232810 -> TNF
- ENSG00000233125 -> ACTBP12 (Maybe not that important, because original gene out of literature was named "CTBP2")

Those ones still existing in the dataframe after the t-test and the 24 significant genes are the relevant genes which may have an impact on CLL forming. We **create a dataframe of relevant genes** and their p-valus to have an overview of genes we should do some research about.

```
relevant_genes1 <- holm_new[c(1:24),]
relevant_genes2 <- holm_new[c("ENSG00000048462","ENSG00000159958","ENSG00000232810",
                              "ENSG00000233125"),]
relevant_genes <- rbind(relevant_genes1,relevant_genes2)
```

Hyper- and hypomethylation in relevant genes

By now we have created the dataframe `relevant_genes` containing the genes which show significant differences in methylation and four genes, which do not show significant methylation differences between sick and healthy, but seem to be worth a look, because we found them in literature before. But to know whether those genes in cancer patients are hyper or hypomethylated we need to have a look at the fold value. Therefore we **add the fold values to the relevant genes**. Genes in cancer patients are **upmethylated** if the **fold is negative** and the absolute fold higher than 0.05. And Genes are **downmethylated** if the **fold is positiv** and the absolute fold higher than 0.05.

```
fold <- data.frame(fold)
fold2 <- as.data.frame(fold[rownames(relevant_genes),])
relevant_genes <- cbind(relevant_genes, "fold" = fold2$fold)
print(relevant_genes)
```

```
##                       p_value   p_adjusted                    Symbols
## ENSG00000215946 1.175067e-09 1.175067e-05                    MIR941-1
## ENSG00000204802 1.015632e-08 1.015530e-04                        <NA>
## ENSG00000264890 1.376601e-08 1.376326e-04                        <NA>
## ENSG00000199643 1.475274e-08 1.474831e-04                    RNU4-90P
## ENSG00000206921 1.722710e-08 1.722021e-04                   RNU6-481P
## ENSG00000279920 2.285155e-08 2.284012e-04                        <NA>
## ENSG00000252391 3.152707e-08 3.150815e-04                   RNU6-638P
## ENSG00000277437 4.763914e-08 4.760580e-04                   MIR3687-1
## ENSG00000222071 6.322396e-08 6.317338e-04                     MIR1915
## ENSG00000273587 1.331822e-07 1.330624e-03                     SNORA78
## ENSG00000275908 1.661780e-07 1.660119e-03                        <NA>
## ENSG00000277942 2.189029e-07 2.186621e-03                     MIR8075
## ENSG00000207554 3.582260e-07 3.577961e-03                      MIR647
## ENSG00000211837 6.980247e-07 6.971173e-03                      TRAJ53
## ENSG00000278903 7.360642e-07 7.350337e-03                        <NA>
## ENSG00000273981 8.751361e-07 8.738234e-03                        <NA>
## ENSG00000274450 9.378005e-07 9.363000e-03                        <NA>
## ENSG00000216141 1.213785e-06 1.211721e-02 MIR941-2;MIR941-3;MIR941-4
## ENSG00000248979 3.518207e-06 3.511874e-02                   LAMTOR3P2
## ENSG00000278639 4.424737e-06 4.416330e-02                        <NA>
## ENSG00000258577 4.986456e-06 4.976483e-02                     SNRPGP1
## ENSG00000277532 5.631739e-06 5.619912e-02                        <NA>
## ENSG00000207508 9.249911e-06 9.229562e-02                  RNU6-1237P
## ENSG00000252749 9.339817e-06 9.318335e-02                    RNU7-116P
## ENSG00000048462 3.398150e-02 1.000000e+00                     TNFRSF17
## ENSG00000159958 9.172109e-02 1.000000e+00                    TNFRSF13C
## ENSG00000232810 2.391774e-01 1.000000e+00                          TNF
## ENSG00000233125 5.847920e-01 1.000000e+00                     ACTBP12
##                          fold
## ENSG00000215946 -0.062098540
## ENSG00000204802          NaN
## ENSG00000264890 -0.087534616
## ENSG00000199643          NaN
## ENSG00000206921          NaN
## ENSG00000279920          NaN
## ENSG00000252391          NaN
## ENSG00000277437          NaN
## ENSG00000222071 17.415912725
## ENSG00000273587 16.170359452
```

```
## ENSG00000275908  0.056731496
## ENSG00000277942 -0.065590372
## ENSG00000207554 24.754040645
## ENSG00000211837          NaN
## ENSG00000278903          NaN
## ENSG00000273981          NaN
## ENSG00000274450          NaN
## ENSG00000216141 -0.113837067
## ENSG00000248979 -0.035460336
## ENSG00000278639 -0.122955439
## ENSG00000258577 -0.107881498
## ENSG00000277532          NaN
## ENSG00000207508  0.787502101
## ENSG00000252749          NaN
## ENSG00000048462  1.307070457
## ENSG00000159958 -1.316244737
## ENSG00000232810 -0.978602969
## ENSG00000233125 -0.006033483
```

Interpretation

The most differently methylated gene is MIR-941-1. It is a gene for a microRNA (non-coding and short RNA) that plays a role in the transcriptional regulation of gene expression. MicroRNAs have an influence on stability and translation of mRNAs by being a part of the RNA induced silencing complex (RISC). RISC can bind mRNAs complementary to its miRNA or siRNA. This can either inhibit or destabilize the mRNA, so that less or no proteins coded by the mRNA get produced. MIR-941-1 is known to be transcriptionally silenced in gastric cancer cells which fits to our results in which a hypermethylation of this gene can be oberserved too. Other differently methylated genes in our dataset which are coding for miRNAs were MIR3687-1, MIR1915, MIR8075, MIR941-2, MIR941-3, MIR941-4 and MIR647. We can't determine a trend in hypomethylation or hypermethylation. Another group of differently methylated genes play a role in splicing. Three of the 24 tested genes coded for the U6 subunit of the spliceosome, namely RNU6-481P, RNU6-638P and RNU6-1237P. Splicing is part of the processing of the mRNA in eukaryotes which makes it very important for the protein activity. Also, we found out that RNU7-116P is differently methylated. RNU7-116P codes for a U7 small nuclear ribonucleoprotein which is used for histone RNA processing. Both the U6 and U7 subunit both play a part in transcriptional regulation. Due to this regulation, proteins can be differently expressed which might lead to cancer development.

Sources:

(Kim et al., 2014) (Mannoor et al., 2012) (Stefanovic et al., 1995)

Some additional information to our genes out of the literature:

As already mentioned earlier in **Data evaluation**, the following genes have been identified in the literature as potential genes associated with cancer. Therefore, these genes are discussed in more detail.

ACTBP12:

ACTBP12 is just a pseudogene with no specific function. As the gene out of our literature analysis was named CTBP2 and not ACTBP12, this is not surprising. Still we wanted to see, whether there is a relation between these genes.

TNF:

This gene belongs to the tumor necrosis factor (TNF) superfamily and encodes a multifunctional proinflammatory cytokine. This cytokine is mainly secreted by macrophages, which are also important for the immune system. By binding to specific receptors, this cytokine is involved in the regulation of a wide spectrum of biological processes including cell proliferation, differentiation, apoptosis, lipid metabolism, and coagulation.

It is linked with a variety of diseases for example autoimmune diseases but also cancer, which is important in our case. These specific TNF receptors are expressed in a wide variety of tissues in mammals, especially in leucocytes which include B-cells. This signalling pathways, as already mentioned, is able to promote B-cell survival and proliferation.

Sources to TNF: (Rickert et al., 2011) (Gravestein and Borst, 1998)

TNFRSF13C:

Tumor necrosis factor receptor superfamily member 13C (TNFRSF13C) is, as the name suggests, a gene which encodes for a receptor of the TNF. It is also known as B-cell-activating factor receptor (BAFF-R). In general, BAFF promotes the proliferation and differentiation of B-cells.

Sources to TNFRSF13C: (Mackay et al., 2010)

TNFRSF17:

The protein encoded by the gene tumor necrosis factor receptor superfamily member 17 (TNFRSF17) gene is a member of the TNF-receptor superfamily. It is a receptor, especially expressed in mature B lymphocytes and might play a role in B cell development. It leads to NF-kappaB and MAPK8/JNK activation and therefore it binds to different TRAF family members. It might transduce signals for cell survival and proliferation.

Sources in general for the DMRs:

Kim, J.G., Kim, T.O., Bae, J.H., Shim, J.W., Kang, M.J., Yang, K., Ting, A.H., and Yi, J.M. (2014). Epigenetically regulated MIR941 and MIR1247 target gastric cancer cell growth and migration. Epigenetics 9, 1018-1030.

Mannoor K, Liao J, Jiang F. Small nucleolar RNAs in cancer. Biochim Biophys Acta. 2012;1826(1):121–128. doi:10.1016/j.bbcan.2012.03.005

Stefanovic B, Wittop Koning TH, Schümperli D. A synthetic histone pre-mRNA-U7 small nuclear RNA chimera undergoing cis cleavage in the cytoplasm of Xenopus oocytes. Nucleic Acids Res. 1995;23(16):3152–3160. doi:10.1093/nar/23.16.3152

Gravestein, L.A., and Borst, J. (1998). Tumor necrosis factor receptor family members in the immune system. Seminars in Immunology 10, 423-434.

Rickert, R.C., Jellusova, J., and Miletic, A.V. (2011). Signaling by the tumor necrosis factor receptor superfamily in B-cell biology and disease. Immunological reviews 244, 115-133.

Mackay, F., Figgett, W.A., Saulep, D., Lepage, M., and Hibbs, M.L. (2010). B-cell stage and context-dependent requirements for survival signals from BAFF and the B-cell receptor. Immunological reviews 237, 205-225.