testfile

Johannes Schadt, Anastasia Möller, Sylviane Verschaeve, Tine Limberg

2023-07-02

Proteome-wide Screen for RNA-dependent Proteins: *non-synchronized A549 cells*

Zum PDF exportieren

```
#install.packages("tinytex")
#tinytex::reinstall_tinytex()
```

Loading the data:

```
MS_Table <- read.delim('https://www.dropbox.com/s/vm3lxljjm9chau8/RDeeP_A549_NS.csv?dl=1', header=TRUE,
```

1. Preparing data for analysis

1.1. Check for missing values

```
sum(apply(MS_Table, 1, anyNA)) == 0
```

```
## [1] TRUE
```

```
sum(is.na(MS_Table)) == 0
```

```
## [1] TRUE
```

1.2. Check data format

```
sum(apply(MS_Table, 1, is.numeric)) == nrow(MS_Table)
```

```
## [1] TRUE
```

1.3. Deleting rows with only zeros

```
min(MS_Table)
```

```
## [1] 0
```

```
sum(apply(MS_Table,1,sum)==0)
```

```
## [1] 0
```

-> da die Summe der Zeileneinträge keines Proteins 0 entspricht, wurde ein Dataframe aus False erstellt. Einträge ausschließlich False, werden durch die sum Funktion als 0 aufaddiert.

1.4. Rearranging of Data

```
MS_Table_reordered <- MS_Table[, c(
  paste0("Fraction", 1:25, "_Ctrl_Rep1"),
  paste0("Fraction", 1:25, "_Ctrl_Rep2"),
  paste0("Fraction", 1:25, "_Ctrl_Rep3"),
  paste0("Fraction", 1:25, "_RNase_Rep1"),
  paste0("Fraction", 1:25, "_RNase_Rep2"),
  paste0("Fraction", 1:25, "_RNase_Rep3")
)]
# View(MS_Table_reordered)
sum(apply(MS_Table_reordered, 2, is.numeric)) == ncol(MS_Table)
```

1.4.1. Reordering columns

```
## [1] TRUE
```

```
MS_Table_Ctrl <-MS_Table_reordered[,1:75]
#View(MS_Table_Ctrl)
MS_Table_RNase <-MS_Table_reordered[,76:150]
View(MS_Table_RNase)
```

1.4.2. Separate Ctrl and RNase

2. Reproducibility

Here we test whether the replicates are similar to each other. This would mean, that the experiment is reproducible, thus the data is reliable. Proteins that do not satisfy this condition will be removed from the dataset and will not be analysed.

2.1 Pearson Correlation To facilitate the calculation of the correlation between each replicate, we design 6 separate data frames, one for each replicate

```
ctrl.rep1.reprod <- MS_Table_reordered[,1:25]
ctrl.rep2.reprod <- MS_Table_reordered[,26:50]
ctrl.rep3.reprod <- MS_Table_reordered[,51:75]
rnase.rep1.reprod <- MS_Table_reordered[,76:100]
rnase.rep2.reprod <- MS_Table_reordered[,101:125]
rnase.rep3.reprod <- MS_Table_reordered[,126:150]
```

Here we calculate the correlation between the replicates and put them together in one data frame (ctrl.cor and rnase.cor) (?)

Now we eliminate proteins which have NA-correlations (this happens when they contain replicates with only 0s). We then create new separate data frames for each replicate.

```
total.na <- which(rowSums(is.na(ctrl.rnase.cor)) > 0)
length(total.na)
```

```
## [1] 83
```

```
MS.Table.naremoved <- MS_Table_reordered[-total.na,]

ctrl.rep1.naremoved <- MS.Table.naremoved[,1:25]
ctrl.rep2.naremoved <- MS.Table.naremoved[,26:50]
ctrl.rep3.naremoved <- MS.Table.naremoved[,51:75]
rnase.rep1.naremoved <- MS.Table.naremoved[,76:100]
rnase.rep2.naremoved <- MS.Table.naremoved[,101:125]
rnase.rep3.naremoved <- MS.Table.naremoved[,126:150]
```

Now we calculate the correlation of the replicates. This time the proteins that containes replicates with only 0 are eliminated, so there should be no NAs anymore.

```
ctrl.cor.naremoved <-
  cbind(ctrl.cor.rep1.rep2.naremoved <-
          sapply(seq.int(dim(ctrl.rep1.naremoved)[1]), function(x) cor(as.numeric(ctrl.rep1.naremoved[x
        ctrl.cor.rep2.rep3.naremoved <-
          sapply(seq.int(dim(ctrl.rep1.naremoved)[1]), function(x) cor(as.numeric(ctrl.rep2.naremoved[x
        ctrl.cor.rep1.rep3.naremoved <-
          sapply(seq.int(dim(ctrl.rep1.naremoved)[1]), function(x) cor(as.numeric(ctrl.rep1.naremoved[x

rnase.cor.naremoved <-
  cbind(rnase.cor.rep1.rep2.naremoved <-
         sapply(seq.int(dim(rnase.rep1.naremoved)[1]), function(x) cor(as.numeric(rnase.rep1.naremoved[
        rnase.cor.rep2.rep3.naremoved <-
          sapply(seq.int(dim(rnase.rep1.naremoved)[1]), function(x) cor(as.numeric(rnase.rep2.naremoved
        rnase.cor.rep1.rep3.naremoved <-
          sapply(seq.int(dim(rnase.rep1.naremoved)[1]), function(x) cor(as.numeric(rnase.rep1.naremoved

#View(ctrl.cor.naremoved)
```

The following plot shows us the general distribution of correlation.

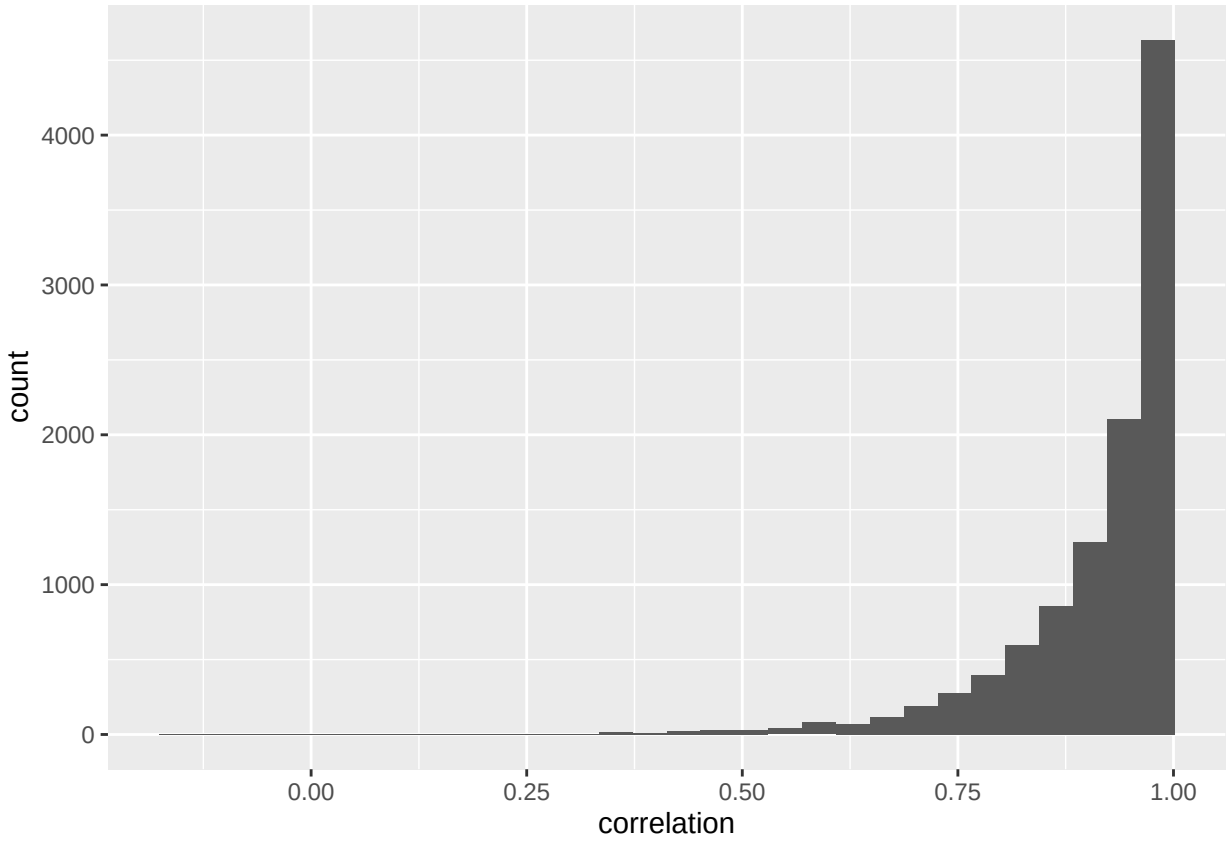
In total we look at 3*(3680-83) correlations This has to be taken into account, when looking at the graphs. It is import to figure out if the 3 cor are for one protein or for 3 different ones.

```
library(ggplot2)
```

```
## Warning: Paket 'ggplot2' wurde unter R Version 4.2.3 erstellt
```

```
ctrl.cor.data.frame.naremoved <- data.frame(c(ctrl.cor.naremoved[,1],ctrl.cor.naremoved[,2],ctrl.cor.na
colnames(ctrl.cor.data.frame.naremoved) <- "correlation"
ggplot(ctrl.cor.data.frame.naremoved, aes(x=correlation)) + geom_histogram()
```

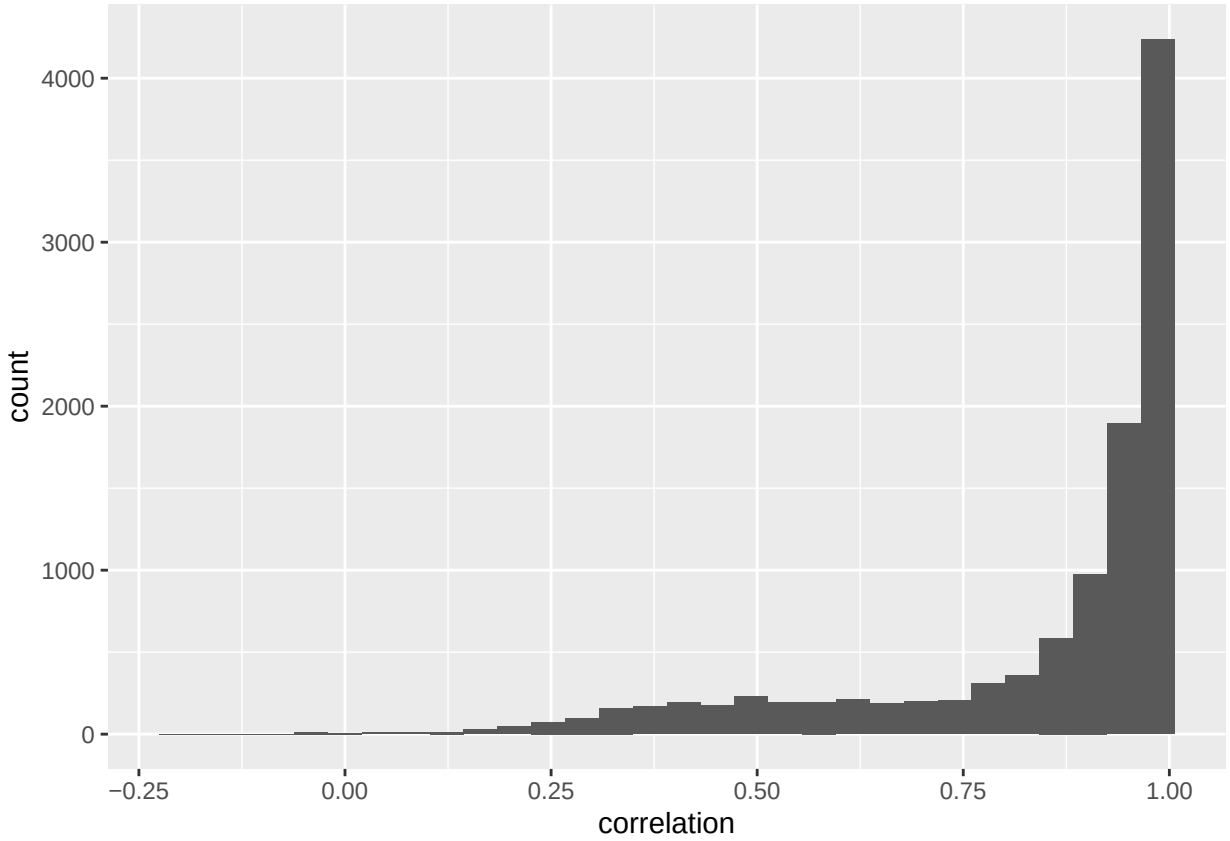
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We do the same for the RNase group:

```
library(ggplot2)
rnase.cor.data.frame.naremoved <- data.frame(c(rnase.cor.naremoved[,1],rnase.cor.naremoved[,2],rnase.co
colnames(rnase.cor.data.frame.naremoved) = "correlation"
ggplot(rnase.cor.data.frame.naremoved, aes(x=correlation)) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Now we select the proteins which have correlations beneath 0.9. Those are not reproducible, thus the data is not safe enough to be used further.

First we determine the proteins that only have correlations under 0.9.

```
non.reproducible.ctrl <- which(rowSums(ctrl.cor.naremoved<0.9)>2)
#length(non.reproducible.ctrl)

non.reproducible.rnase <- which(rowSums(rnase.cor.naremoved<0.9)>2)
#length(non.reproducible.rnase)

non.reproducible <- unique(c(non.reproducible.ctrl, non.reproducible.rnase))
#length(non.reproducible)

length(non.reproducible.rnase)
```

```
## [1] 330
```

```
length(non.reproducible.ctrl)
```

```
## [1] 254
```

```
#View(non.reproducible2)
```

Now we eliminate the proteins that only have correlations under 0.9.

```
ctrl.rep <- MS.Table.naremoved[-non.reproducible,1:75]
rnase.rep <- MS.Table.naremoved[-non.reproducible,76:150]

ctrl.cor.removed <- ctrl.cor.naremoved [-non.reproducible,]
rnase.cor.removed <- rnase.cor.naremoved [-non.reproducible,]


length(non.reproducible)
```

```
## [1] 523
```

```
#View(ctrl.rep)
#View(MS.Table.naremoved[,1:75])
#View(non.reproducible)
```

Other proteins are a bit trickier. Some proteins have two replicates similar to each other (correlation < 0.9) and a third one that completely differs. These Proteins have one very high and two smaller correlations. The different replicate is often the third one (mabye batch effect). To avoid loosing too many proteins and still to still have safe data, we try to ignore the bad replicates. For this we first set them to NA: After the normalization-set we can ignore them.

```
for (x in 1:dim(ctrl.rep)[1]){
  if (ctrl.cor.removed[x, 1] < 0.9) {
    if (ctrl.cor.removed[x, 3] < 0.9){
      ctrl.rep[x, 1:25] <- NA
    }
    if (ctrl.cor.removed[x, 2] < 0.9){
      ctrl.rep[x, 26:50] <- NA
    }}

  if (ctrl.cor.removed[x, 3] < 0.9) {
    if (ctrl.cor.removed[x, 2] < 0.9){
      ctrl.rep[x, 51:75] <- NA
    }
  }
     }

for (x in 1:dim(rnase.rep)[1]){
  if (rnase.cor.removed[x, 1] < 0.9) {
    if (rnase.cor.removed[x, 3] < 0.9){
      rnase.rep[x, 1:25] <- NA
    }
    if (rnase.cor.removed[x, 2] < 0.9){
      rnase.rep[x, 26:50] <- NA
    }}

  if (rnase.cor.removed[x, 3] < 0.9) {
    if (rnase.cor.removed[x, 2] < 0.9){
      rnase.rep[x, 51:75] <- NA
    }
  }
}
```

```
Nr <- c(1:dim(rnase.rep)[1])
rnase.with.proteinnumbers <- cbind(Nr, rnase.rep[,1:25], Nr, rnase.rep[,26:50], Nr, rnase.rep[,51:75])
#View(rnase.with.proteinnumbers)
#View(rnase.cor.removed)

ctrl.with.proteinnumbers <- cbind(Nr, ctrl.rep[,1:25], Nr, ctrl.rep[,26:50], Nr, ctrl.rep[,51:75])
#View(ctrl.with.proteinnumbers)
```

We now have 3074 Proteins left. They are stored in new variables:

We now have clean data, with proteins that have reproducible data we can use for further analysis.

3. Scaled and Reduced Dataset

For the normalization each replicate has to be separated, therefore we design 6 separate dataframes.

```
ctrl.rep1 <- ctrl.rep[,1:25]
ctrl.rep2 <- ctrl.rep[,26:50]
ctrl.rep3 <- ctrl.rep[,51:75]
rnase.rep1 <- rnase.rep[,1:25]
rnase.rep2 <- rnase.rep[,26:50]
rnase.rep3 <- rnase.rep[,51:75]
```

3.1. Mean Value Method

3.1.1. Normalization We perform the mean-value-method (mvm) on each replicate, both control and RNase:

```
# Control Replicate 1 MVM
ctrl.rep1.mvm.norm <- t(apply(ctrl.rep1, 1, function(x) {
  normalized <- x - mean(x)
  normalized[normalized < 0] <- 0
  scaled <- normalized * (100 / sum(normalized))
  return(scaled)
}))

# View(ctrl.rep1.mvm.norm)

# Control Replicate 2 MVM
ctrl.rep2.mvm.norm <- t(apply(ctrl.rep2, 1, function(x) {
  normalized <- x - mean(x)
  normalized[normalized < 0] <- 0
  scaled <- normalized * (100 / sum(normalized))
  return(scaled)
}))

# Control Replicate 3 MVM
ctrl.rep3.mvm.norm <- t(apply(ctrl.rep3, 1, function(x) {
  normalized <- x - mean(x)
  normalized[normalized < 0] <- 0
  scaled <- normalized * (100 / sum(normalized))
```

```
  return(scaled)
}))

# RNase Replicate 1 MVM
rnase.rep1.mvm.norm <- t(apply(rnase.rep1, 1, function(x) {
  normalized <- x - mean(x)
  normalized[normalized < 0] <- 0
  scaled <- normalized * (100 / sum(normalized))
  return(scaled)
}))

# RNase Replicate 2 MVM
rnase.rep2.mvm.norm <- t(apply(rnase.rep2, 1, function(x) {
  normalized <- x - mean(x)
  normalized[normalized < 0] <- 0
  scaled <- normalized * (100 / sum(normalized))
  return(scaled)
}))

# RNase Replicate 3 MVM
rnase.rep3.mvm.norm <- t(apply(rnase.rep3, 1, function(x) {
  normalized <- x - mean(x)
  normalized[normalized < 0] <- 0
  scaled <- normalized * (100 / sum(normalized))
  return(scaled)
}))
```

3.1.2. Reduction To reduce we take the mean value between each replicate. Here we must consider the NA-values of non-reproducible replicates.

```
r1c <- ctrl.rep1.mvm.norm
r2c <- ctrl.rep2.mvm.norm
r3c <- ctrl.rep3.mvm.norm

r1c0 <- ctrl.rep1.mvm.norm
r2c0 <- ctrl.rep2.mvm.norm
r3c0 <- ctrl.rep3.mvm.norm

r1c0[is.na(r1c)] <- 0
r2c0[is.na(r2c)] <- 0
r3c0[is.na(r3c)] <- 0


ctrl.mvm.reduced <-  (r1c0 + r2c0 + r3c0)/(3-((sum(is.na(r1c[x,]) + is.na(r2c[x,]) + is.na(r3c[x,])))/25
```

```
r1r <- rnase.rep1.mvm.norm
r2r <- rnase.rep2.mvm.norm
r3r <- rnase.rep3.mvm.norm

r1r0 <- rnase.rep1.mvm.norm
r2r0 <- rnase.rep2.mvm.norm
r3r0 <- rnase.rep3.mvm.norm
```

```
r1r0[is.na(r1r)] <- 0
r2r0[is.na(r2r)] <- 0
r3r0[is.na(r3r)] <- 0

rnase.mvm.reduced <-  (r1r0 + r2r0 + r3r0)/(3 - ((sum(is.na(r1r[x,]) + is.na(r2r[x,]) + is.na(r3r[x,])))

#View(rnase.mvm.reduced)
```

3.1.3. Scaling To test whether we have "lost" our scaling during the merge, and find out whether scaling back to 100 is necessary, we scale the control to 100 and compare it with the original control.

```
ctrl.mvm.scaled =
  sweep(ctrl.mvm.reduced,1,100/rowSums(ctrl.mvm.reduced),'*')

# Check if the two data frames are identical
is_identical <- identical(ctrl.mvm.reduced, ctrl.mvm.scaled)

# Print the result
if (is_identical) {
  print("The data frames are identical.")
 } else {
  print("The data frames are not identical.")
 }
```

```
## [1] "The data frames are not identical."
```

-> scaling back to 100 is necessary

Because scaling back to 100 is necessary, we do it for the RNase too:

```
rnase.mvm.scaled =
  sweep(rnase.mvm.reduced,1,100/rowSums(rnase.mvm.reduced),'*')
```

Now we have normalized our data using the mean-value-method, and scaled it to 100. The two variables that will be used later on either contain the normalized (mvm) and scaled data of the control: **ctrl.mvm** or the normalized (mvm) and scaled data of the rnase: **rnase.mvm**

```
new.colnames.ctrl <- c("Fraction_1_Ctrl","Fraction_2_Ctrl","Fraction_3_Ctrl","Fraction_4_Ctrl","Fractio

new.colnames.rnase <- c("Fraction_1_RNase","Fraction_2_RNase","Fraction_3_RNase","Fraction_4_RNase","Fra

ctrl.mvm <- ctrl.mvm.scaled
colnames(ctrl.mvm) <- new.colnames.ctrl

rnase.mvm <- rnase.mvm.scaled
colnames(rnase.mvm) <- new.colnames.rnase
```

3.2. z - Transformation

3.2.1. Normalization The z-Transformation does not work with df that have NA-values. This means we cannot use the ctrl.clean and rnase.clean df. We have to use the old ctrl.rep or the MS.Table.naremoved[-non.reproducible,1:75] dfs. On these we first perform z-Transformation and then we use the same algorithms as before to reduce the dataset in regards to reproducibility. (boah mein Elglisch)

```
ctrl.rep.zt <- MS.Table.naremoved[-non.reproducible,1:75]
rnase.rep.zt <- MS.Table.naremoved[-non.reproducible,76:150]

ctrl.rep1z <- ctrl.rep.zt[,1:25]
ctrl.rep2z <- ctrl.rep.zt[,26:50]
ctrl.rep3z <- ctrl.rep.zt[,51:75]
ctrl.repz <- cbind(ctrl.rep1z,ctrl.rep2z,ctrl.rep3z)


rnase.rep1z <- rnase.rep.zt[,1:25]
rnase.rep2z <- rnase.rep.zt[,26:50]
rnase.rep3z <- rnase.rep.zt[,51:75]
rnase.repz <- cbind(rnase.rep1z,rnase.rep2z,rnase.rep3z)
```

First the normalization for the Ctrl:

Since the protein amount in each replicate is different, it is better to calculate the mean for each replicate separately. However the sd-value does not have to be adapted. Because the replicates have the same variance (same procedure for every replicate in the wet lab), we don't have to calculate the standard devidation for the replicates in each fraction extra. We can calculate the sd-value for one row /sd.

sd-values and mean values of Ctrl:

```
sd.ctrl <- apply(ctrl.repz, 1, sd)

mean.ctrl.rep1 <- apply(ctrl.rep1z, 1, mean)
mean.ctrl.rep2 <- apply(ctrl.rep2z, 1, mean)
mean.ctrl.rep3 <- apply(ctrl.rep3z, 1, mean)
```

Normalization of Ctrl:

```
ctrl.rep1.meanvalue <- sweep(ctrl.rep1z,1,mean.ctrl.rep1,'-')
ctrl.rep1.zt.norm <- sweep(ctrl.rep1.meanvalue,1,sd.ctrl,'/')

ctrl.rep2.meanvalue <- sweep(ctrl.rep2z,1,mean.ctrl.rep2,'-')
ctrl.rep2.zt.norm <- sweep(ctrl.rep2.meanvalue,1,sd.ctrl,'/')

ctrl.rep3.meanvalue <- sweep(ctrl.rep3z,1,mean.ctrl.rep3,'-')
ctrl.rep3.zt.norm <- sweep(ctrl.rep3.meanvalue,1,sd.ctrl,'/')

#calculate again the min value for each protein
min.ctrl.rep1.zt.norm <- apply(ctrl.rep1.zt.norm, 1, min)
min.ctrl.rep2.zt.norm <- apply(ctrl.rep2.zt.norm, 1, min)
min.ctrl.rep3.zt.norm <- apply(ctrl.rep3.zt.norm, 1, min)

#substract the min value from each position to discard the 0
ctrl.rep1.zt.norm.pos <- sweep(ctrl.rep1.zt.norm,1,min.ctrl.rep1.zt.norm,FUN = '-')
ctrl.rep2.zt.norm.pos <- sweep(ctrl.rep2.zt.norm,1,min.ctrl.rep2.zt.norm,FUN = '-')
ctrl.rep3.zt.norm.pos <- sweep(ctrl.rep3.zt.norm,1,min.ctrl.rep3.zt.norm,FUN = '-')
```

sd-values and mean values of RNase:

```
sd.rnase = apply(rnase.repz, 1, sd)

mean.rnase.rep1 <- apply(rnase.rep1z, 1, mean)
mean.rnase.rep2 <- apply(rnase.rep2z, 1, mean)
mean.rnase.rep3 <- apply(rnase.rep3z, 1, mean)
```

Normalization of RNase:

```
rnase.rep1.meanvalue <- sweep(rnase.rep1z,1,mean.rnase.rep1,'-')
rnase.rep1.zt.norm <- sweep(rnase.rep1.meanvalue,1,sd.rnase,'/')

rnase.rep2.meanvalue <- sweep(rnase.rep2z,1,mean.rnase.rep2,'-')
rnase.rep2.zt.norm <- sweep(rnase.rep2.meanvalue,1,sd.rnase,'/')

rnase.rep3.meanvalue <- sweep(rnase.rep3z,1,mean.rnase.rep3,'-')
rnase.rep3.zt.norm <- sweep(rnase.rep3.meanvalue,1,sd.rnase,'/')

#calculate again the min value for each protein
min.rnase.rep1.zt.norm <- apply(rnase.rep1.zt.norm, 1, min)
min.rnase.rep2.zt.norm <- apply(rnase.rep2.zt.norm, 1, min)
min.rnase.rep3.zt.norm <- apply(rnase.rep3.zt.norm, 1, min)

#substract the min value from each position to discard the 0
rnase.rep1.zt.norm.pos <- sweep(rnase.rep1.zt.norm,1,min.rnase.rep1.zt.norm,FUN = '-')
rnase.rep2.zt.norm.pos <- sweep(rnase.rep2.zt.norm,1,min.rnase.rep2.zt.norm,FUN = '-')
rnase.rep3.zt.norm.pos <- sweep(rnase.rep3.zt.norm,1,min.rnase.rep3.zt.norm,FUN = '-')
```