Project
Transplant kidney rejection
High Dimensional Data Analysis

Jan Alexander* Annabel Vaessens† Steven Wallaert‡

8/4/2020

```
load('RejectionStatus.rda')
load('X_GSE21374.rda')
dim(RejectionStatus)
```

```
## [1] 282   2
```

```
dim(X_GSE21374)
```

```
## [1] 54675   282
```

```
GeneExpression <- t(X_GSE21374)
GeneExpression_C <- scale(t(X_GSE21374),scale = F) # centered
GeneExpression_S <- scale(t(X_GSE21374),scale = T) # scaled
# vraagje: zie cursus p 74, men geeft aan voor PCA dat
# indien vars in zelfde eenheden zijn (in ons geval is dit zo), dan beter niet schalen.
# Zouden we dan mss beter niet schalen voor PCA?

GeneExpression <-
  GeneExpression[order(as.numeric(row.names(GeneExpression))), ]
```

```
## Warning in order(as.numeric(row.names(GeneExpression))): NAs introduced by
## coercion
```

```
RejectionStatus <-
  RejectionStatus[order(as.numeric(RejectionStatus$Patient_ID)), ]

all.equal(row.names(GeneExpression), as.character(RejectionStatus$Patient_ID))
```

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

*jan.alexander@ugent.be
†annabel.vaessens@vub.be
‡steven.wallaert@ugent.be

1 Introduction

The data is loaded as presented in the assignment. The data is loaded as the raw dataset, the centered dataset and the standardized dataset.

Three research questions are defined:

- How do the 54675 genes vary in terms of their gene expression levels? Is the variability associated with kidney rejection? (only to be answered in a data explorative manner).
- Which genes are differentially expressed between the two kidney rejection groups? You must control the False Discovery Rate at 10%.
- Can the kidney rejection be predicted from the gene expressions? What genes are most important in predicting the kidney transplant rejection? How well does the prediction model perform in terms of predicting rejection status?

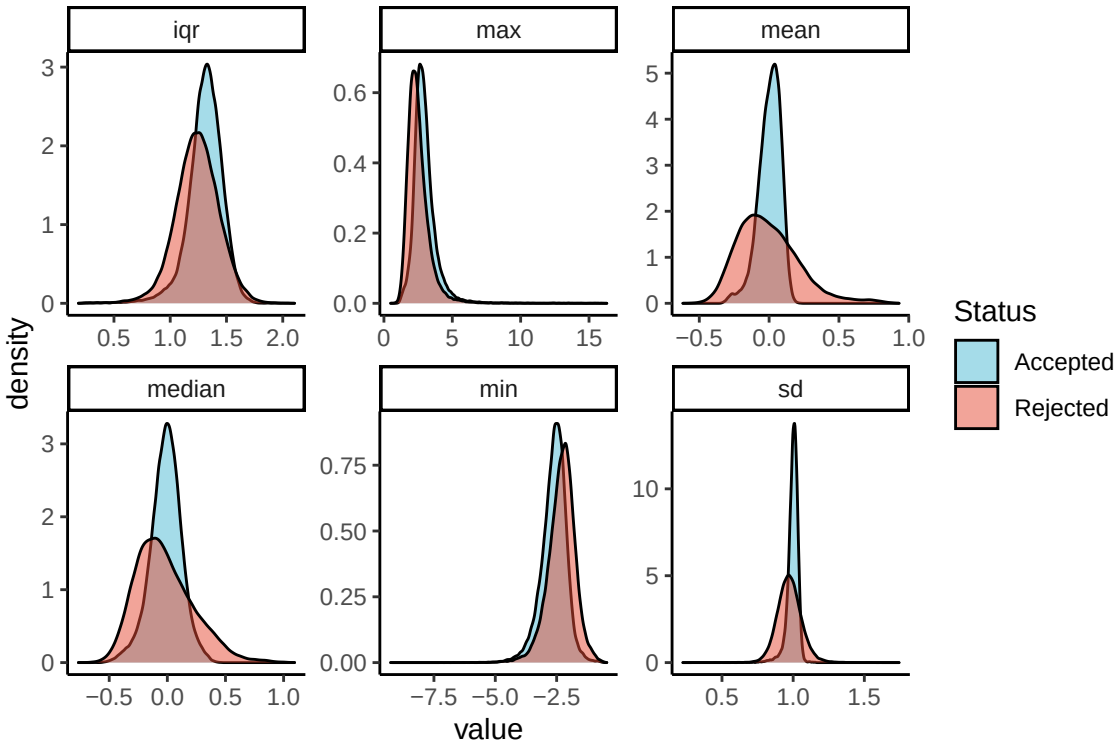
2 Data exploration

In the complete dataset, 27 % of the transplanted kidnesy were rejected.

2.1 Simple plot of the raw data

Before plotting dimension reduced representations of the data, a simple plot of the raw data is made. (insert Stevens code)

```
df <- tibble(patient = RejectionStatus$Patient_ID,
             reject = ifelse(RejectionStatus$Reject_Status == 1, "Rejected", "Accepted"),
             as.data.frame(GeneExpression_C)) %>%
  pivot_longer(cols = c(-reject,-patient) , names_to = "gene", values_to = "expression")

# calculate summaries
df %>%
  group_by(reject, gene) %>%
  summarise(mean = mean(expression),
            sd = sd(expression),
            min = min(expression),
            max = max(expression),
            median = median(expression),
            iqr = IQR(expression)) -> basics
# long format
basics %>%
  pivot_longer(-c(reject, gene), names_to = "statistic", values_to = "value") -> basics_long
# plot
ggplot(basics_long, aes(value, fill = reject)) +
  geom_density(alpha = 0.5) +
  theme_classic() +
  labs(fill = "Status") +
  facet_wrap("statistic", scales = "free" ) +
  scale_fill_manual(values = c("#4DBBD5", "#E64B35"))
```

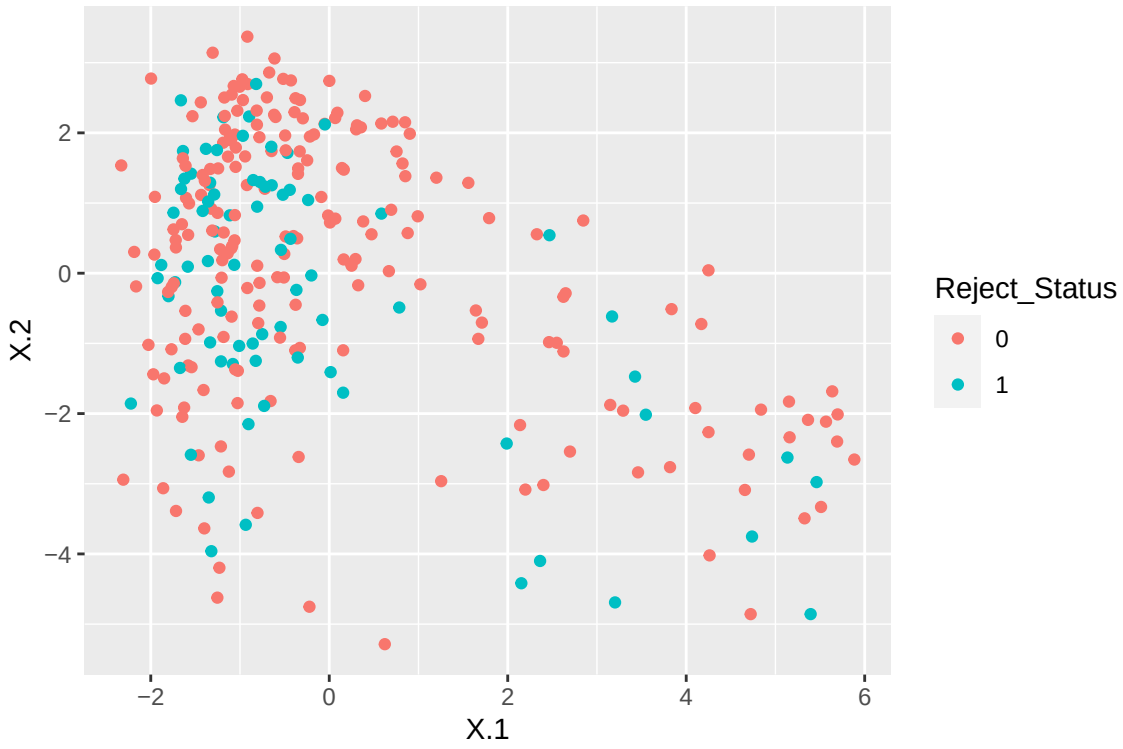


2.2 Sparse principle components analysis

```
Gen_spc <- PMA::SPC(GeneExpression_C, K = 2, sumabsv = 2)
```

```
## 12345678910111213141516171819 20
## 12345678910111213141516171819 20
```

```
Uk <- Gen_spc$u ; Dk <- diag(Gen_spc$d)
Zk <- data.frame(X = Uk %*% Dk, Patient_ID = row.names(GeneExpression_C))
Zk <- merge(Zk, RejectionStatus, by = 'Patient_ID') %>%
  mutate(Reject_Status = as.factor(Reject_Status))
ggplot(data = Zk, aes(x = X.1, y = X.2, col = Reject_Status)) +
  geom_point()
```




```
rm(Zk, Dk, Uk, X_GSE21374, Gen_spc)
```

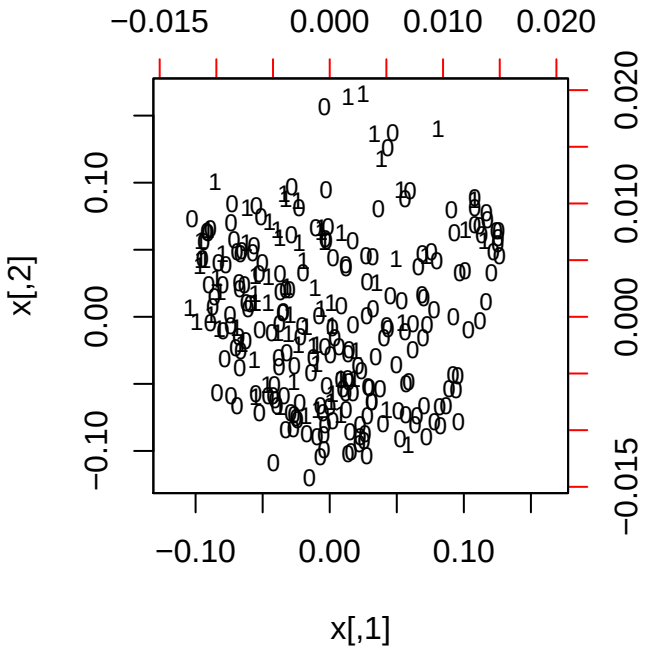
Unfortunately, naive sparce principle component analysis does not seem to work well.

2.3 Partial least squares:

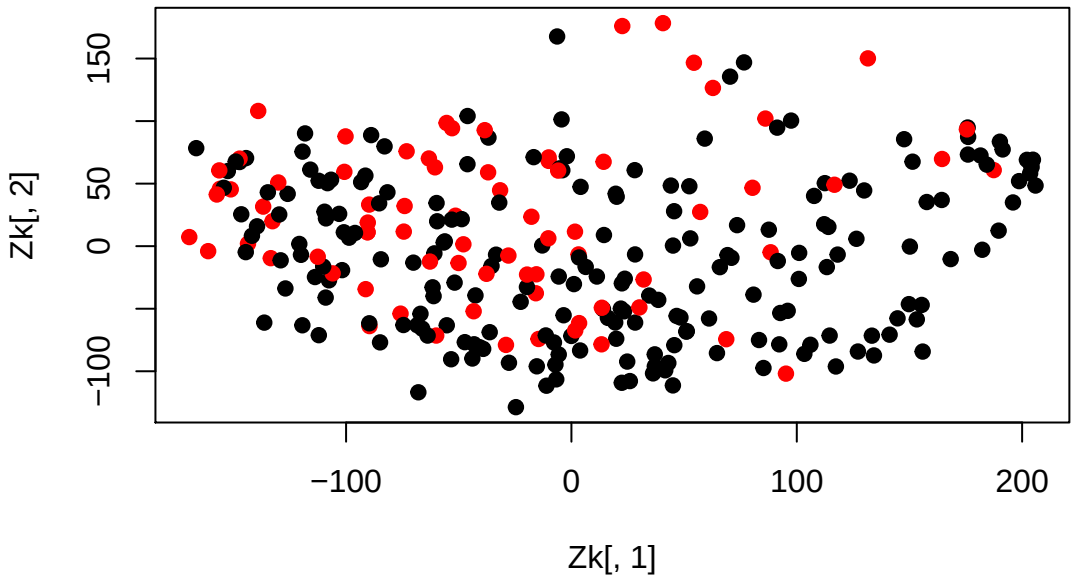
```
GeneExpression_comb <-
  list(genes = as.matrix(GeneExpression_C), Rejection = as.matrix(RejectionStatus$Reject_Status))
pls_model <- pls::plsr(genes ~ Rejection, data = GeneExpression_comb, validation = "CV")
```

2.4 Multi-dimensional scaling:

```
GeneExpression_C.svd <- svd(GeneExpression_C)

k <- 2
Uk <- GeneExpression_C.svd$u[,1:k]; Dk <- diag(GeneExpression_C.svd$d[1:k])
Vk <-GeneExpression_C.svd$v[,1:k]
Xk <- Uk %*% Dk %*% t(Vk)
Zk <- Uk %*% Dk

rownames(Zk) <- RejectionStatus[[2]]
rownames(Vk) <- colnames(GeneExpression_C)
ColnamesNull <- colnames(GeneExpression_C)
ColnamesNull[]<- ""

biplot(Uk, Vk, xlabs=RejectionStatus[[2]], var.axes=FALSE, ylabs=ColnamesNull, cex=0.75)
```

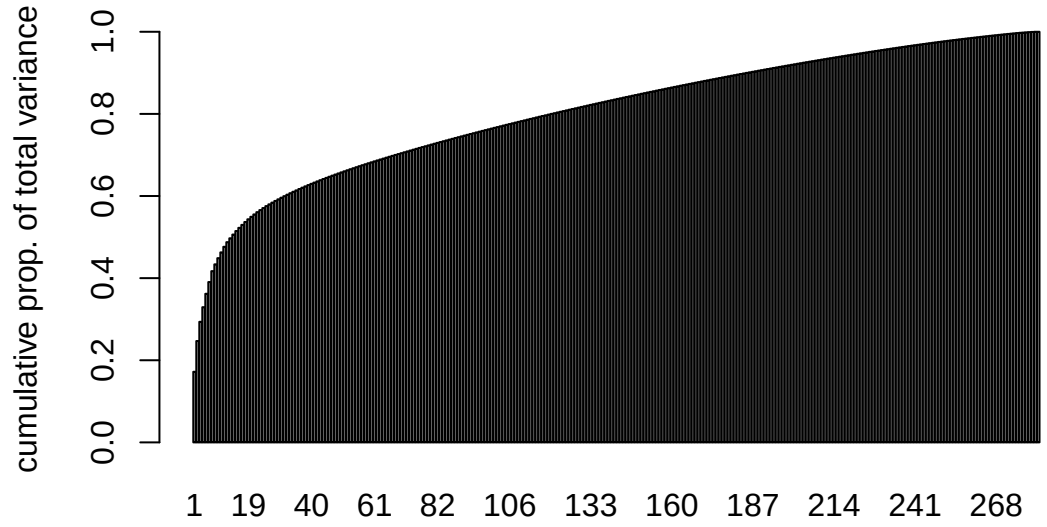



```
# eventueel
plot(Zk[,1], Zk[,2], col=RejectionStatus$Reject_Status+1,pch=19)
```



In the biplot of the two first dimensions of the svd, no distinction can be made between rejected and accepted kidneys.

```
totvar <- sum(GeneExpression_C.svd$d^2)/(nrow(GeneExpression_C)-1)
barplot(cumsum(GeneExpression_C.svd$d^2/(nrow(GeneExpression_C)-1)/totvar), names.arg=1:nrow(GeneExpress
```



In the scree plot above it can be seen that the two first dimensions account for only 26% of the total variance in the dataset.

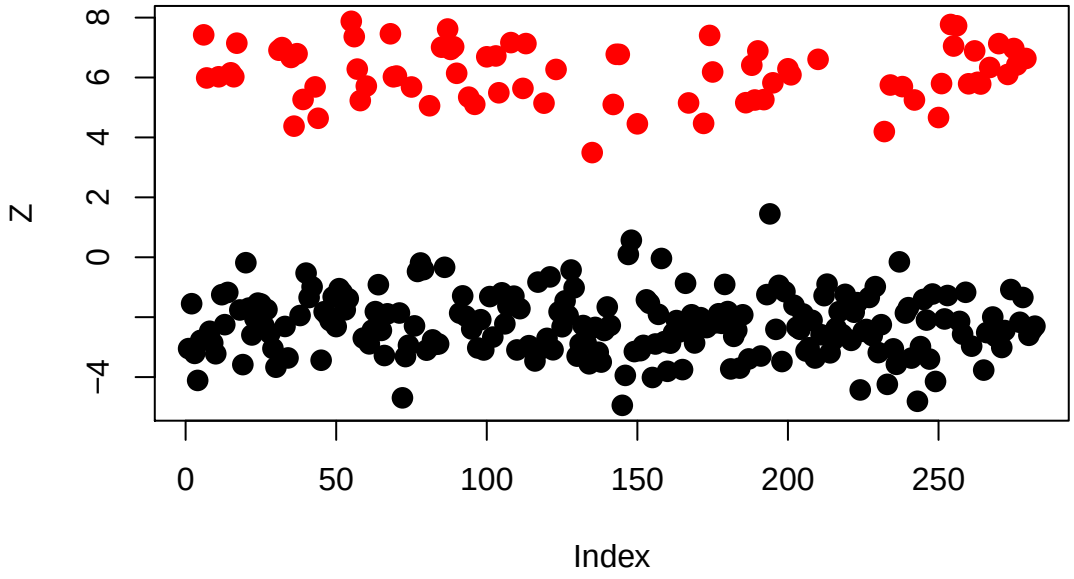
2.5 LDA

In an exploratory fashion, we look at the results of an LDA, performed with the first 300 gene expressions. We chose to do this because LDA performed using all gene expressions did not work.

```
GeneExpression_C.lda <- lda(GeneExpression_C[,1:300], grouping = RejectionStatus$Reject_Status)
```

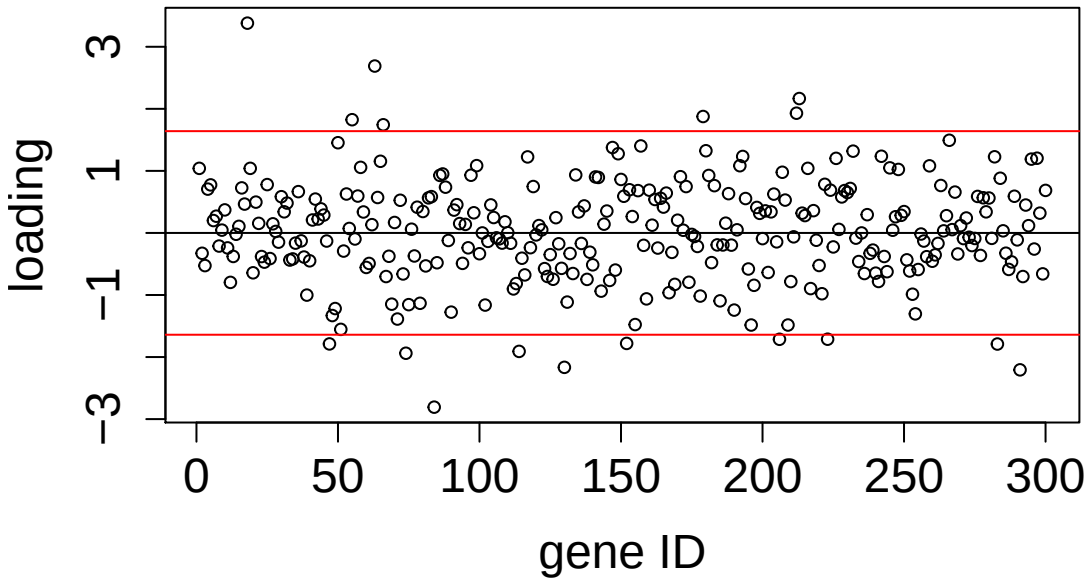
```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
V <- GeneExpression_C.lda$scaling
Z <- GeneExpression_C[,1:300] %*% V

plot(Z, col =  RejectionStatus$Reject_Status+1, pch= 16, cex = 1.5)
```



We can see a clear distinction between the 2 rejection status groups.

```
plot(V, cex = 0.8, cex.axis = 1.5, cex.lab = 1.5, xlab = "gene ID",
     ylab = "loading")

abline(h=0)
abline(h=c(2,-2)*sd(V), col = 2)
```




```
id <- which(abs(V) > 2*sd(V))
```

The most contributing gene expressions are:

```
colnames(GeneExpression_C[,id])
```

```
##  [1] "1552263_at"   "1552307_a_at" "1552318_at"   "1552327_at"   "1552332_at"
##  [6] "1552347_at"   "1552367_a_at" "1552411_at"   "1552436_a_at" "1552473_at"
## [11] "1552508_at"   "1552548_at"   "1552557_a_at" "1552558_a_at" "1552573_s_at"
## [16] "1552658_a_at" "1552667_a_at"
```

We repeated this for 243 folds of 225 gene expressions because

```
243 * 225 == dim(GeneExpression_C)[2]
```

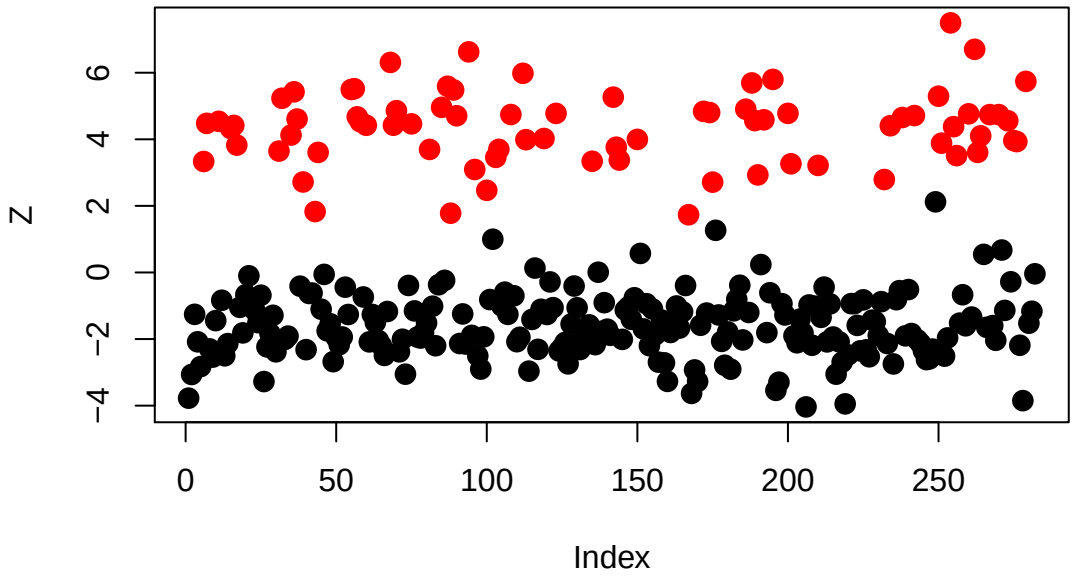
```
## [1] TRUE
```

```
id.all <- vector("numeric")

for (i in 1:(dim(GeneExpression_C)[2]/225)){

  start <- 1 + (i-1)*225
  stop <- start + 224
  gene.lda <- lda(GeneExpression_C[,start:stop], grouping = RejectionStatus$Reject_Status)

  V <- gene.lda$scaling
```

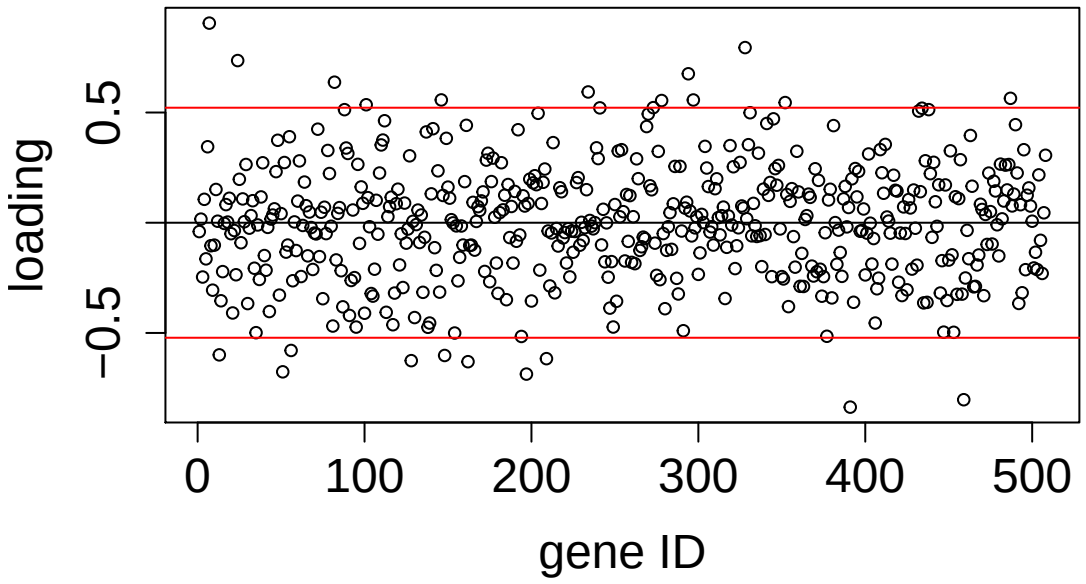
```
  Z <- GeneExpression_C[,start:stop] %*% V


  id.all <- append(id.all, which(abs(V) > 3*sd(V)) - 1 + start) # -1 to start at 0

}
gene.lda <- lda(GeneExpression_C[,id.all], grouping = RejectionStatus$Reject_Status)
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
V <- gene.lda$scaling
Z <- GeneExpression_C[,id.all] %*% V
plot(Z, col =  RejectionStatus$Reject_Status+1, pch= 16, cex = 1.5)
```




```
plot(V, cex = 0.8, cex.axis = 1.5, cex.lab = 1.5, xlab = "gene ID",
     ylab = "loading")

abline(h=0)
abline(h=c(2,-2)*sd(V), col = 2)
```




```
id <- which(abs(V) > 2*sd(V))

colnames(GeneExpression_C[,id.all][,id])
```

```
##  [1] "1553102_a_at" "1553584_at"   "1555495_a_at" "1565703_at"   "1568513_x_at"
##  [6] "201745_at"    "203253_s_at"  "204959_at"    "206313_at"    "206445_s_at"
## [11] "207614_s_at"  "210970_s_at"  "212036_s_at"  "213853_at"    "218757_s_at"
## [16] "219121_s_at"  "220485_s_at"  "220554_at"    "223922_x_at"  "225956_at"
## [21] "229603_at"    "238461_at"    "242669_at"
```

Capping off at +- 3SD, this resulted in 508 gene expressions.

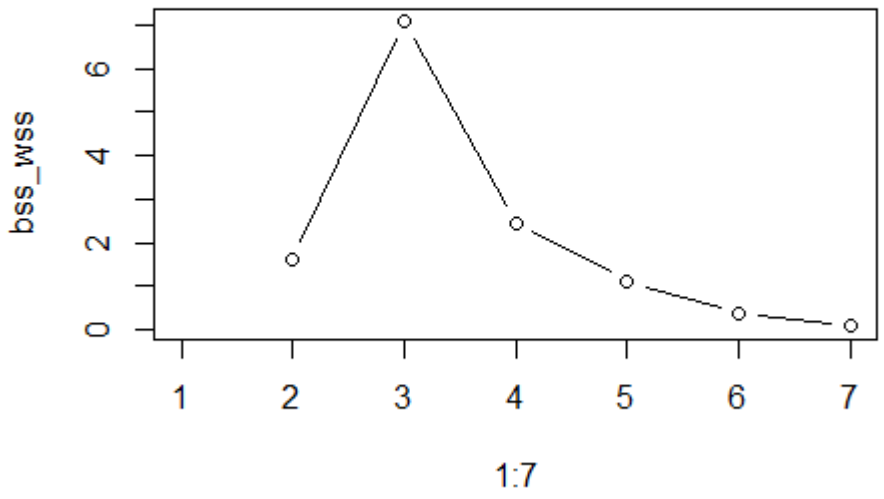
Using all these, and only these gene expressions, an LDA delivers 23 genes.

Why at 3SD? PAS OP UITVOEREN DUURT LANG, in plaats daarvan, zie plot in de map

```
id.all <- vector("numeric")
bss_wss <- vector("numeric")
p <- vector("numeric")

for(j in 2:8){
  id.all <- vector("numeric")
for (i in 1:(dim(gene)[2]/225)){

  start <- 1 + (i-1)*225
  stop <- start + 224
  gene.lda <- lda(gene[,start:stop], grouping = RejectionStatus$Reject_Status)

  V <- gene.lda$scaling
  Z <- gene[,start:stop] %*% V


  id.all <- append(id.all, which(abs(V) > j*sd(V)) - 1 + start) # -1 to start at 0

}
```

```
gene.lda <- lda(gene[,id.all], grouping = RejectionStatus$Reject_Status)

V <- gene.lda$scaling
Z <- gene[,id.all] %*% V

a <- sum((Z[RejectionStatus$Reject_Status==1] - mean(Z[RejectionStatus$Reject_Status==1]))**2)
b <- sum((Z[RejectionStatus$Reject_Status==0] - mean(Z[RejectionStatus$Reject_Status==0]))**2)

within <- a+b


a <- sum((mean(Z[RejectionStatus$Reject_Status==0])-mean(Z))**2)
b <- sum((mean(Z[RejectionStatus$Reject_Status==1])-mean(Z))**2)
between <- a*sum(RejectionStatus$Reject_Status==0)+b*sum(RejectionStatus$Reject_Status==1)

bss_wss[j] <- between/within
p[j] <- length(id.all)
}

plot(1:7, bss_wss, "b")
```

```
knitr::include_graphics("hoeveel_sd.png")
```



2.6 LLE (locally linear embedding)

Locally linear embedding (Roweis and Saul, 2000) is a nonlinear dimension reduction method which finds a low-dimensional representation for each points' local neighbourhood by linearly approximating the data in each neighbourhood and returing these coordinates of lower dimension.

(Value of k (number of neighbours) could be optimised)

```
lle <- RDRToolbox::LLE(data=GeneExpression_C, dim=3, k=50)
labels = c("first component", "second component", 'third component')
plot(lle[,1],lle[,2],col=RejectionStatus$Reject_Status+1,pch=19)
plot(lle[,2],lle[,3],col=RejectionStatus$Reject_Status+1,pch=19)
# 3D interactive plot: (remove in final version)
plot <-RDRToolbox::plotDR(data=lle, labels=RejectionStatus[,2], axesLabels=labels)
```

But also with LLE, we cannot differentiate between the accepted and rejected kidneys. (more on LLE: https://www.stat.cmu.edu/~cshalizi/350/lectures/14/lecture-14.pdf)

2.7 ISOMAP

ISOMAP is a a nonlinear dimension reduction technique presented by Tenenbaum, Silva and Langford in 2000. It preserves rather the global properties of the data. It uses multidimensional scaling but then with incorporating the geodesic distances imposed by a weighted graph of the k neighbours of each point.

```
IM <- RDRToolbox::Isomap(data=GeneExpression_C, dims=3, k=30)
labelsIM <- c("first component", "second component", "third component")
plot(IM$dim3[,1],IM$dim3[,2],col=RejectionStatus$Reject_Status+1,pch=19)
plot(IM$dim3[,2],IM$dim3[,3],col=RejectionStatus$Reject_Status+1,pch=19)
# 3d plot (remove in final version)
RDRToolbox::plotDR(data=IM$dim3, labels=RejectionStatus[,2], axesLabels=labelsIM)

RDRToolbox::Isomap(data=GeneExpression_C, dims=1:10, k=30, plotResiduals=TRUE)
```

The parameter k is varied manually so that the maps are optimal. The value k could be optimised. With ISOMAP, it is also not possible to make a distinction between the group of accepted and rejected kidneys.

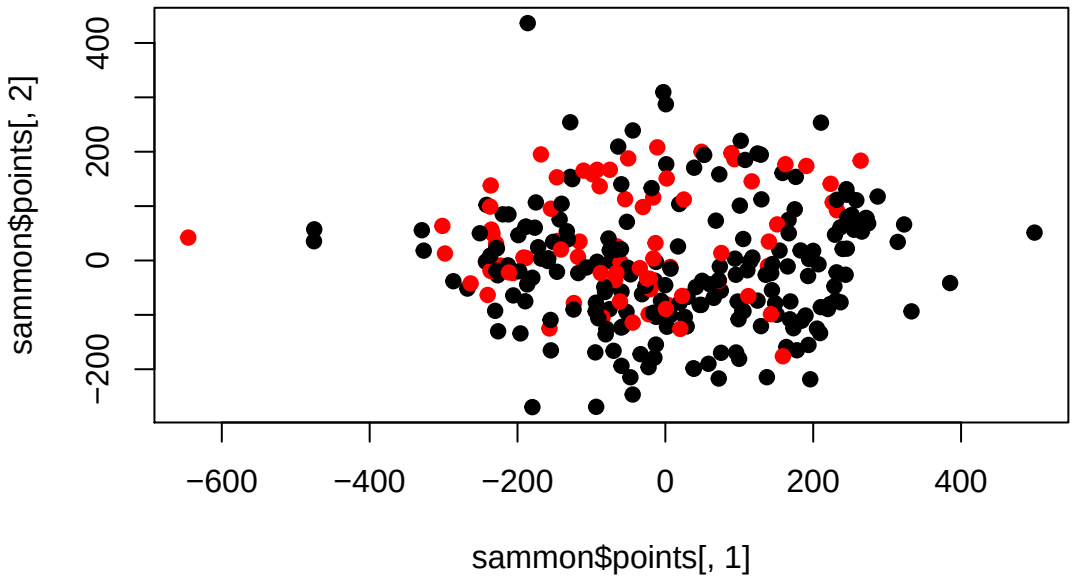
2.8 Sammon mapping

Sammon mapping is also a nonlinear dimension reduction method. The cost function of the Sammon method is similar to that of MDS, except that it is adapted by dividing the squared error in the representation of each pairwise Euclidean distance by the original Euclidean distance in the high-dimensional space. In this way, local structure is better preserved than in MDS.

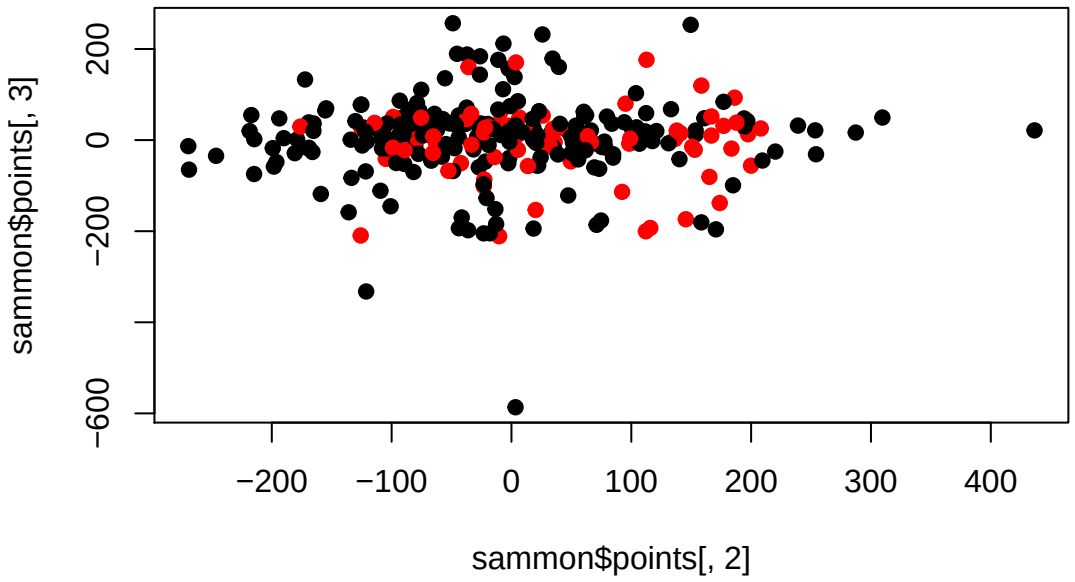
```
sammon <- MASS::sammon(dist(GeneExpression_C), k=3, niter=100)
```

```
## Initial stress        : 0.28532
## stress after  10 iters: 0.21735, magic = 0.004
## stress after  20 iters: 0.16971, magic = 0.009
## stress after  30 iters: 0.13753, magic = 0.004
## stress after  32 iters: 0.13448
```

```
plot(sammon$points[,1], sammon$points[,2], type = "p", col=RejectionStatus$Reject_Status+1, pch=19)
```




```
plot(sammon$points[,2], sammon$points[,3], type = "p", col=RejectionStatus$Reject_Status+1, pch=19)
```



2.9 Diffusion maps

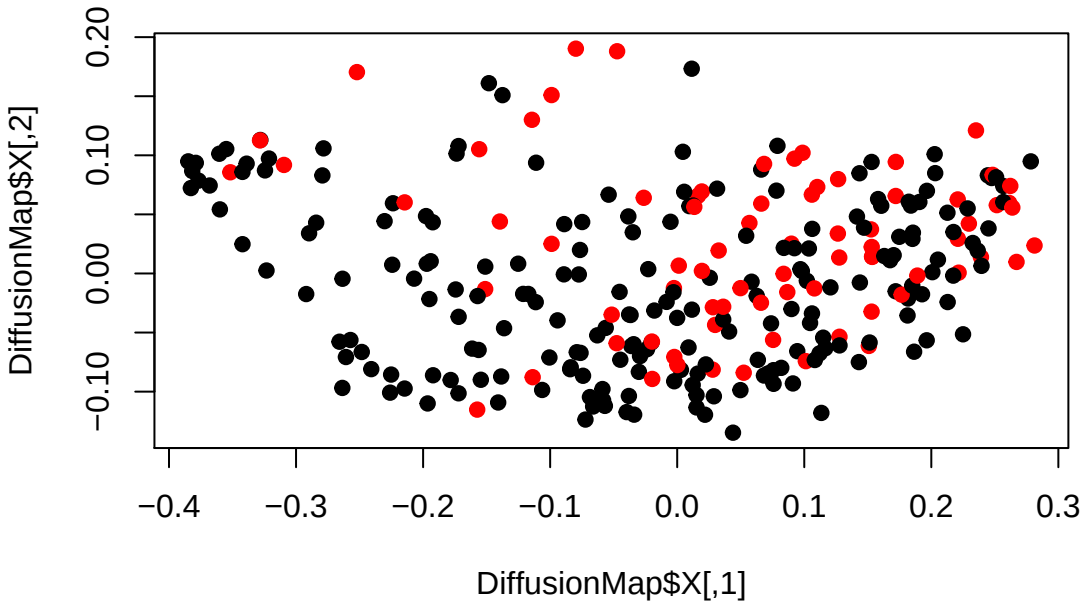
Diffusion mapping is also a nonlinear dimension reduction method based on the eigenvectors and eigenvalues of the data. The global strucure of the data is mapped by integration of local similarities at different scales. It works with probablities of point x randomly walking to y, and is based on the heat diffusion equation.

```
DiffusionMap <- diffusionMap::diffuse(dist(GeneExpression_C), maxdim=3)
```

```
## Performing eigendecomposition
## Computing Diffusion Coordinates

## Warning in min(which(lam < 0.05)): no non-missing arguments to min; returning
## Inf
```

```
## Used default value: 3 dimensions
## Elapsed time: 35.26 seconds
```

```
plot(DiffusionMap$X, type='p', col=RejectionStatus$Reject_Status+1, pch=19)
```



2.10 t-SNE

```
gene_dist <- dist(GeneExpression_C)
tsne_Z <- tsne(gene_dist, k = 1, perplexity = 45)
tibble(as.data.frame(tsne_Z), reject = factor(RejectionStatus$Reject_Status)) %>%
  ggplot(aes(V1, V2, color = reject)) +
  geom_point() +
  theme_classic()
```

3 Differentiating genes between kidney acceptance and rejection

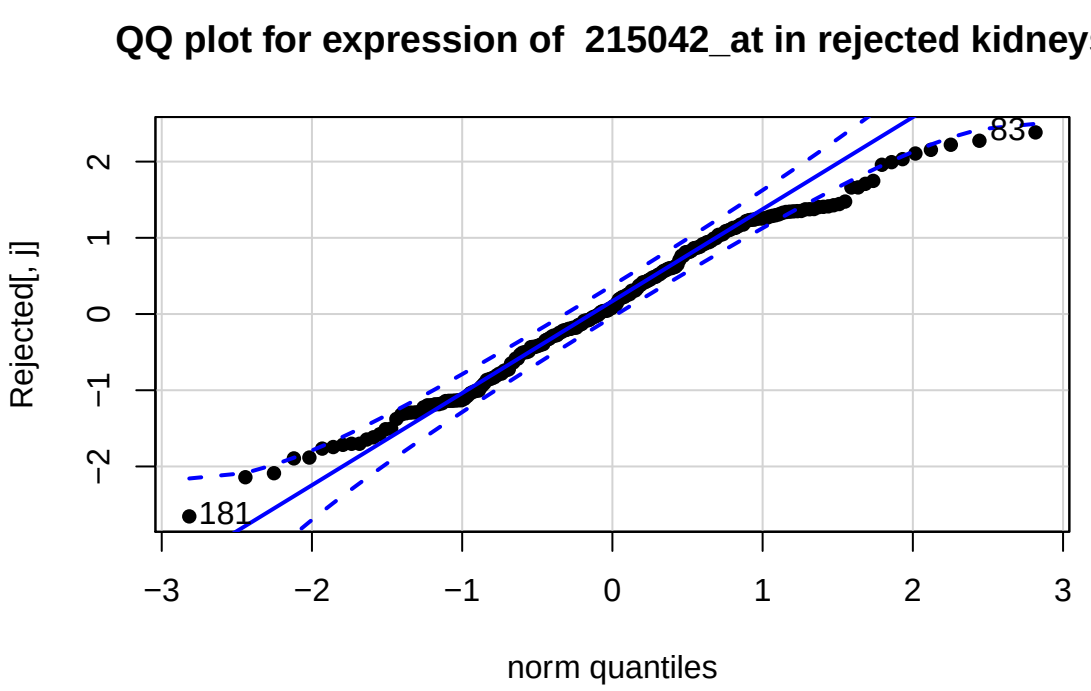
All hypothesis tests are done on the centered, not-standardized data.

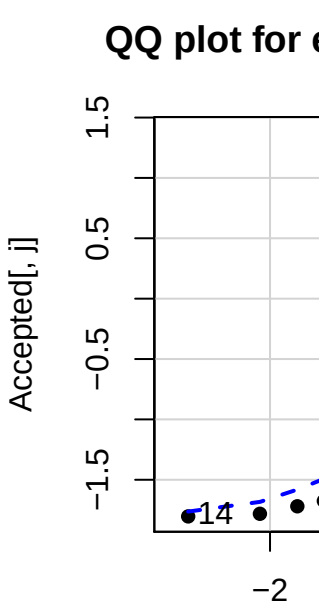
First, a check of the variables follow a normal distribution for both the accepted and rejected kidneys. This is done by checking several QQ plots.

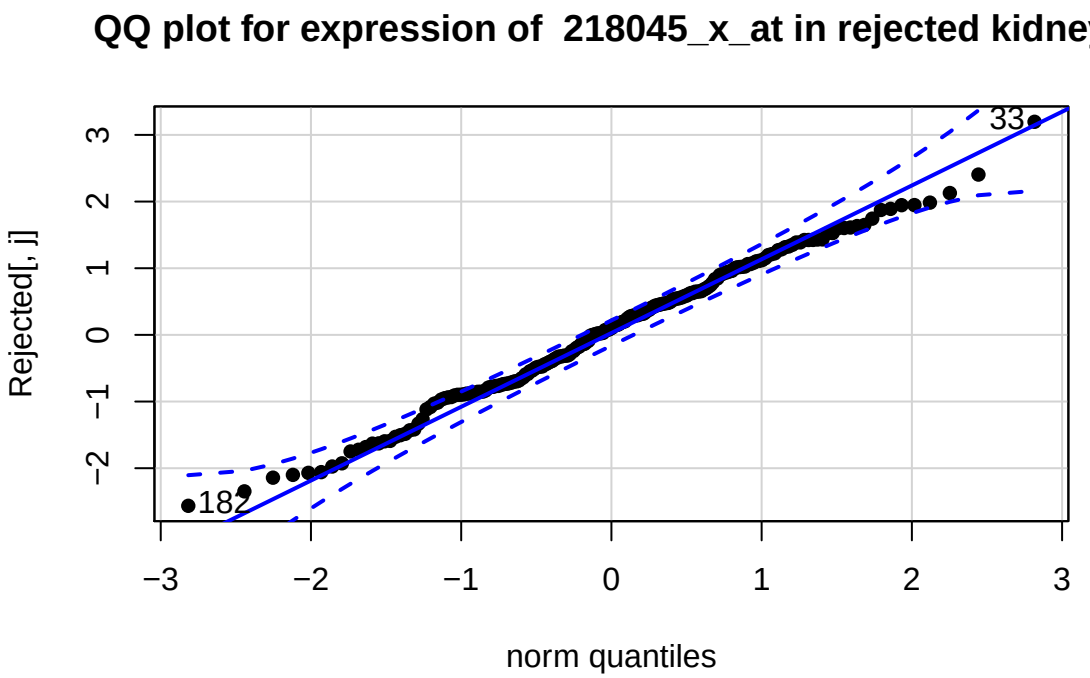
remark 1: noticed the method is not BH after all. Will fix this later an use Stevens suggestion remark 2: also test whether the variances of the two groups is unequal? (there will be some variables for sure where variance is unequal, therefore safer to do the Welch t-test rather than the student t-test

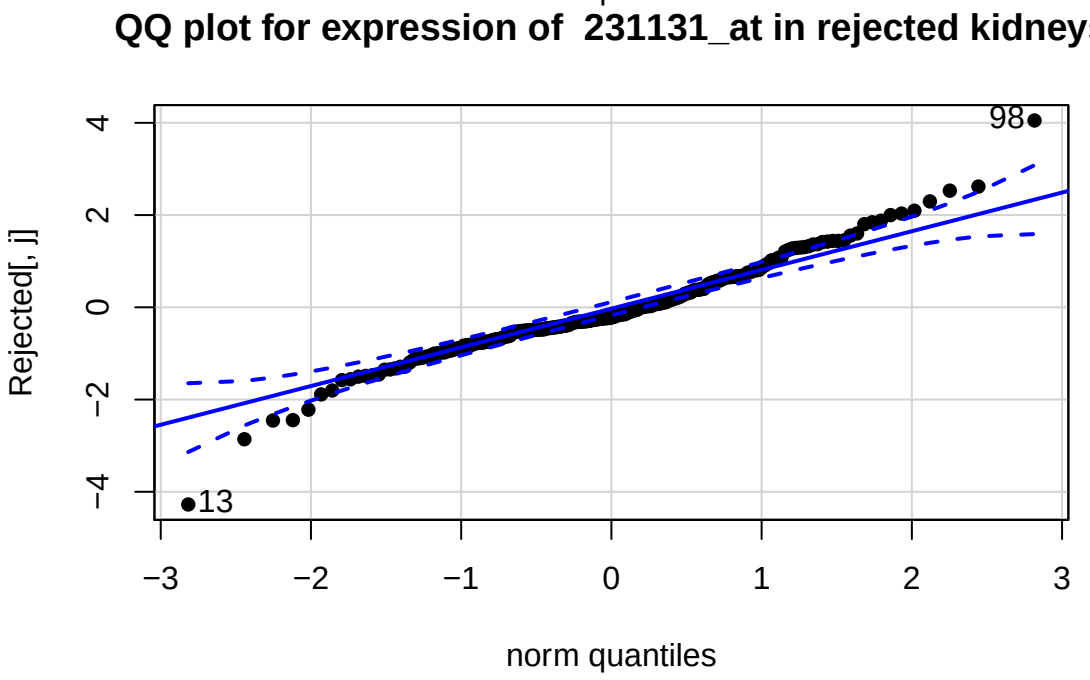
```
Rejected=matrix(ncol=ncol(GeneExpression_C))
for (i in seq(1,nrow(RejectionStatus),1)){
  if (RejectionStatus[[i,2]]==0){
    Rejected <- rbind(Rejected, GeneExpression_C[i,])
  }
}
```

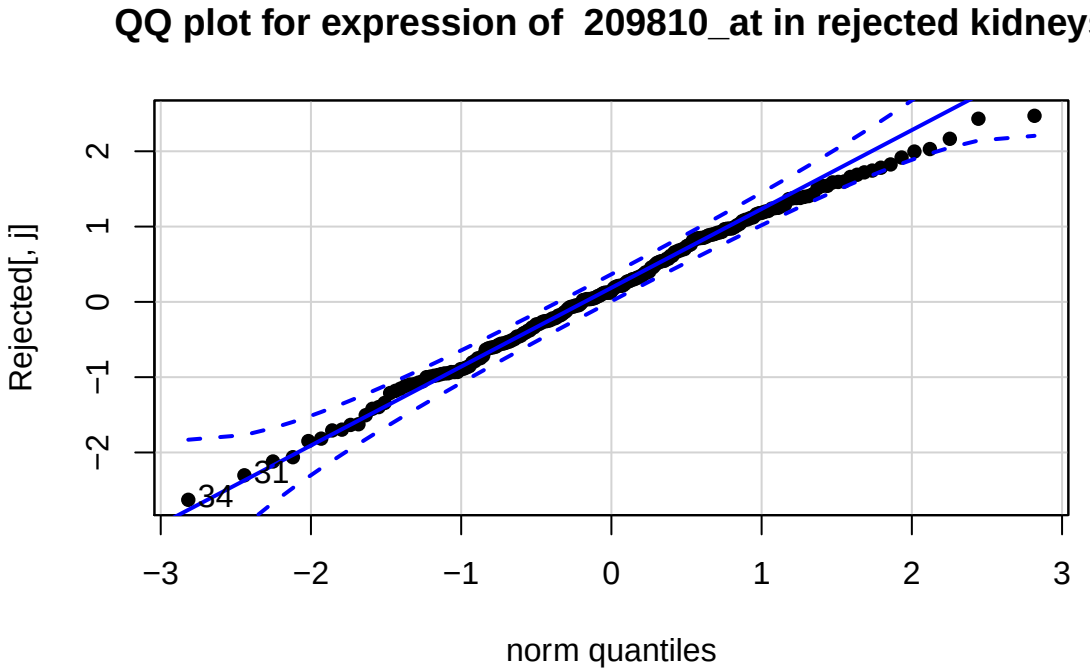
```
Accepted=matrix(ncol=ncol(GeneExpression_C))
for (i in seq(1,nrow(RejectionStatus),1)){
  if (RejectionStatus[[i,2]]==1){
    Accepted <- rbind(Accepted, GeneExpression_C[i,])
  }
}

Selection <- sample(seq(1,ncol(GeneExpression_C)), 15)

for (j in Selection) {
    qqPlot(Rejected[,j], pch = 16,
           main=paste('QQ plot for expression of ', colnames(GeneExpression_C)[j] ,
                      'in rejected kidneys'))
    qqPlot(Accepted[,j], pch = 16,
           main=paste('QQ plot for expression of ', colnames(GeneExpression_C)[j] ,
                      'in accepted kidneys'))
}
```

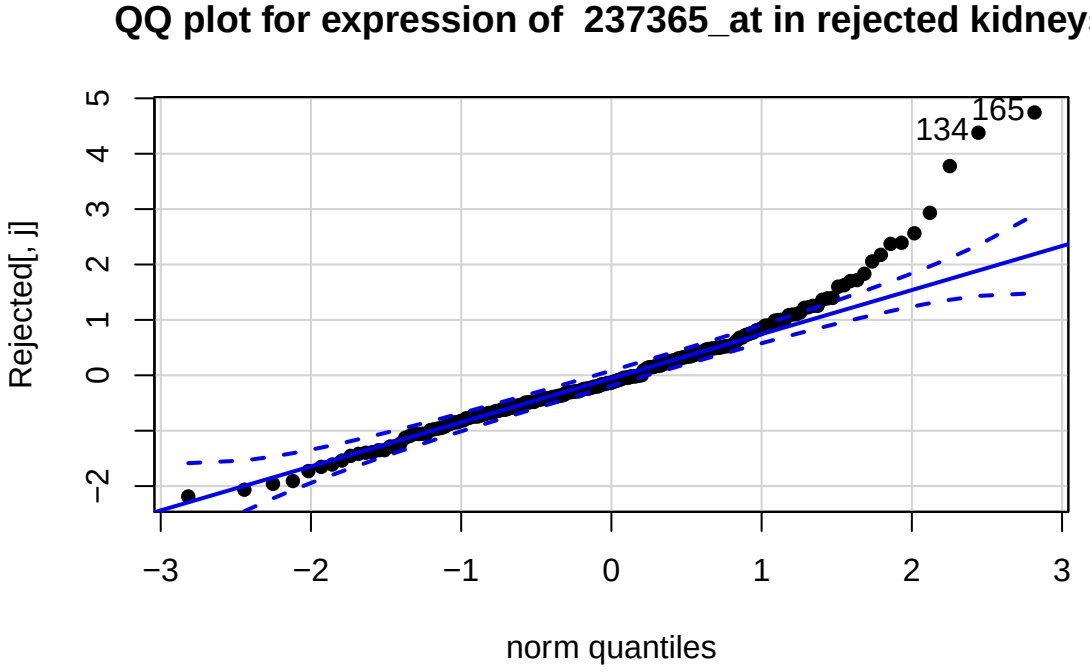


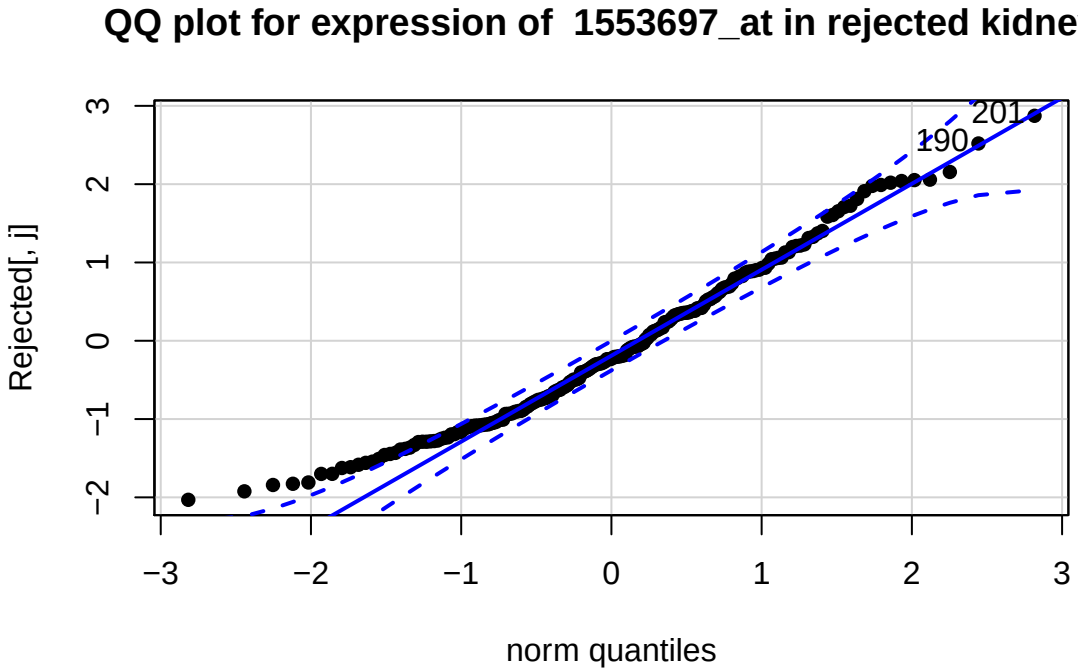


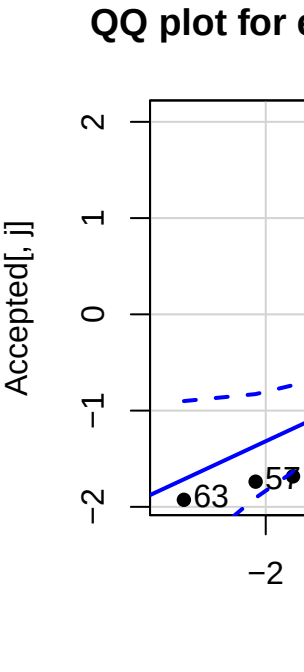

QQ plot for expression of 218045_x_at in rejected kidneys
Rejected[, j]
norm quantiles
33
182

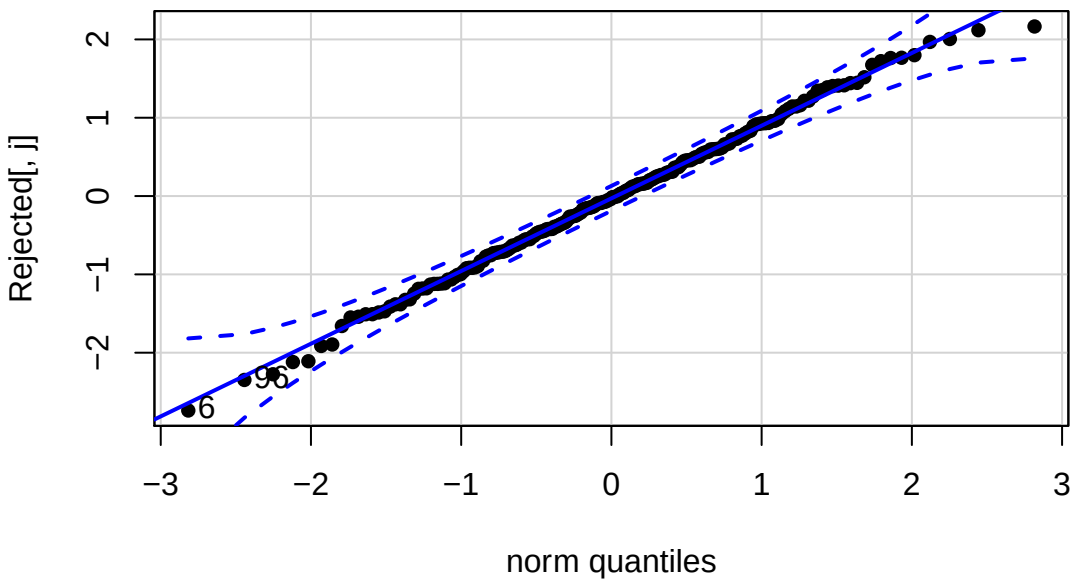

QQ plot for expression of 231131_at in rejected kidneys
Rejected[, j]
norm quantiles
98
13

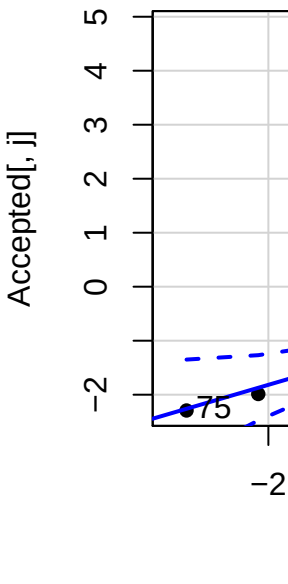

QQ plot for expression of 209810_at in rejected kidneys
Rejected[, j]
norm quantiles
34
31


QQ plot for expression of 237365_at in rejected kidneys
Rejected[, j]
norm quantiles
165
134

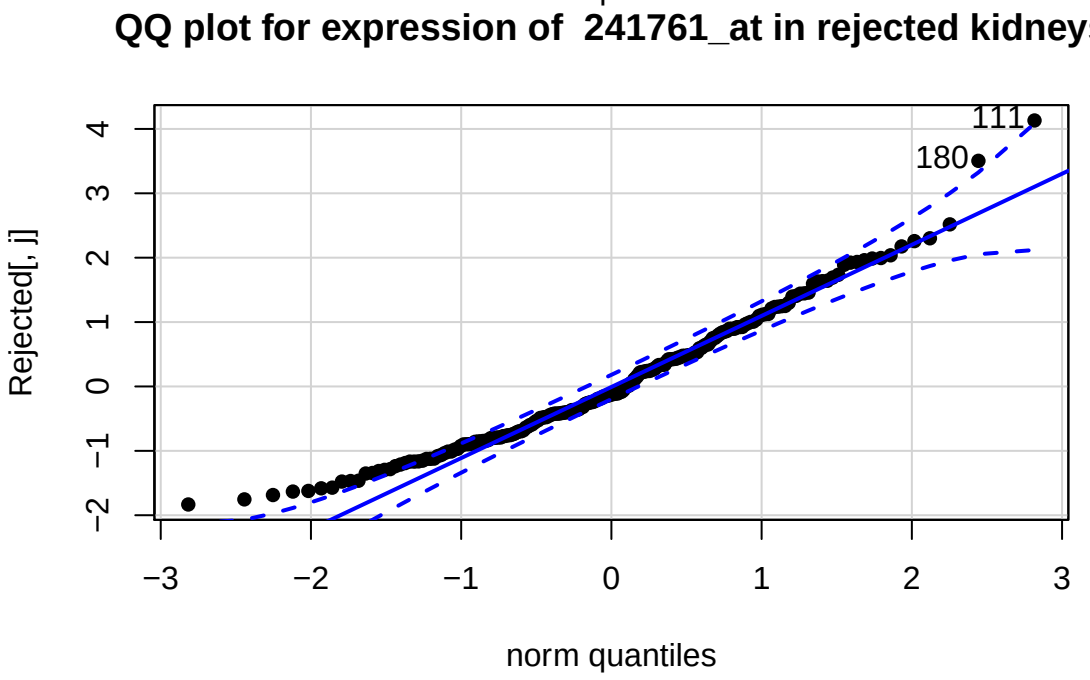

QQ plot for expression of 1553697_at in rejected kidneys
Rejected[, j]
norm quantiles
201
190

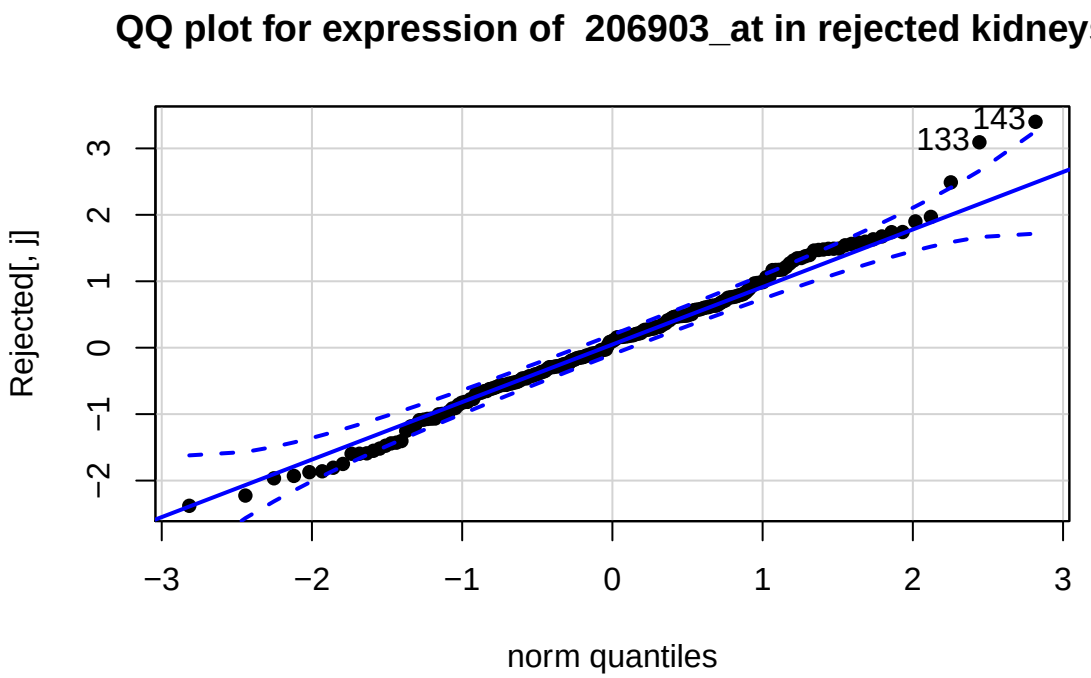

QQ plot for e
Accepted[, j]
63

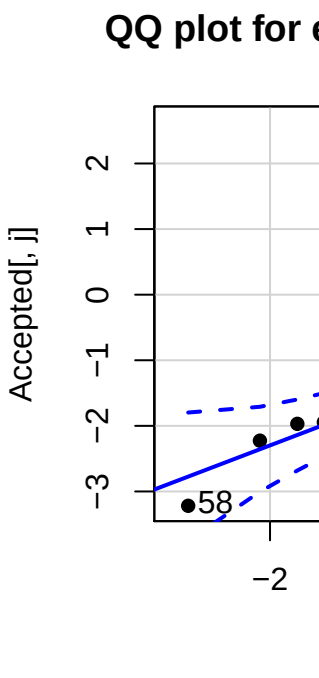

QQ plot for expression of 1553237_x_at in rejected kidneys
Rejected[, j]
norm quantiles
6
96

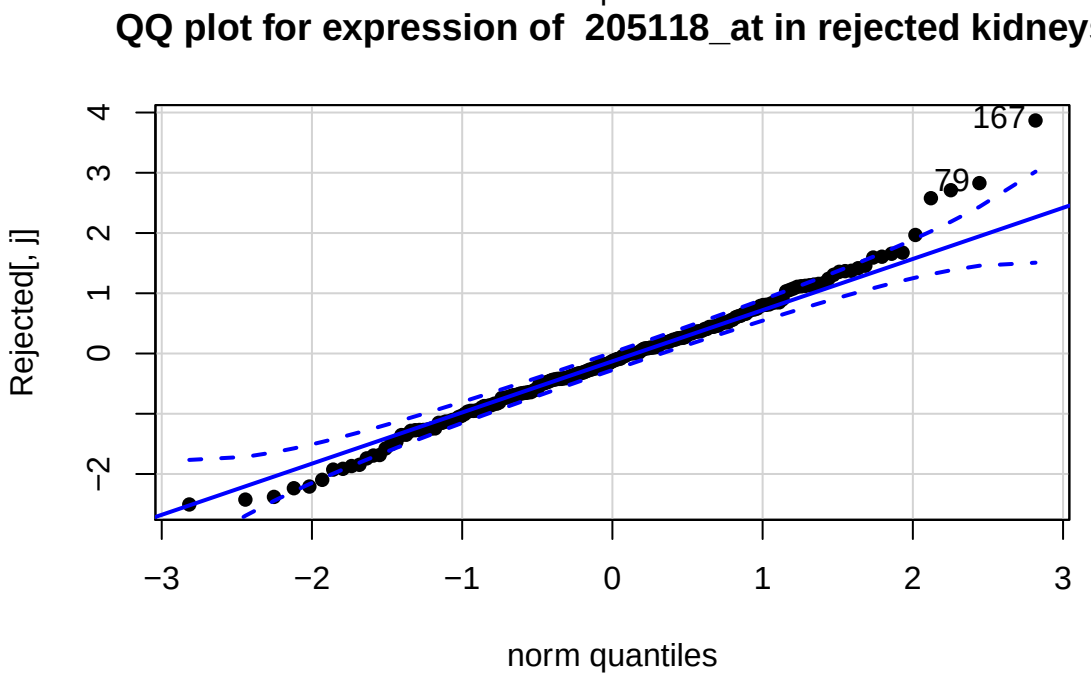

QQ plot for e
Accepted[, j]
75

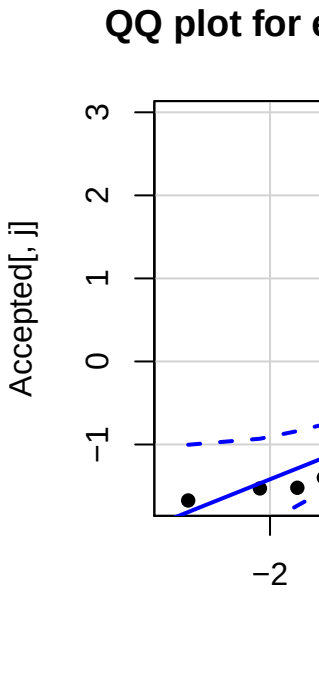

QQ plot for expression of 222113_s_at in rejected kidneys
Rejected[, j]
norm quantiles
173
130
−3
−2
−1
0
1
2
3

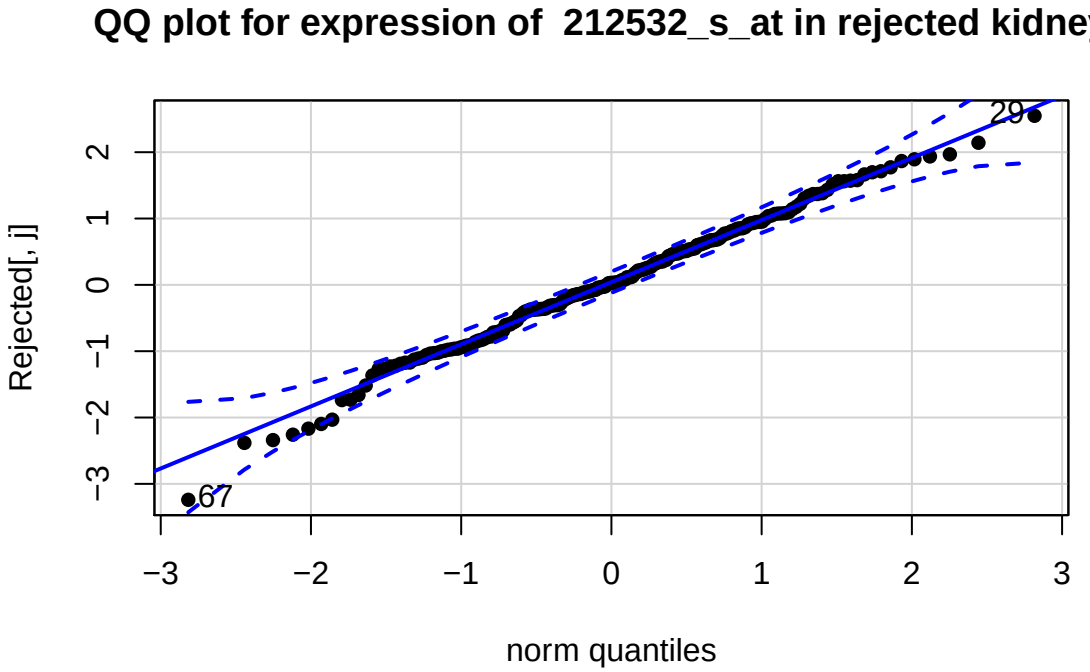

QQ plot for expression of 241761_at in rejected kidneys
Rejected[, j]
norm quantiles
111
180
−3
−2
−1
0
1
2
3
4

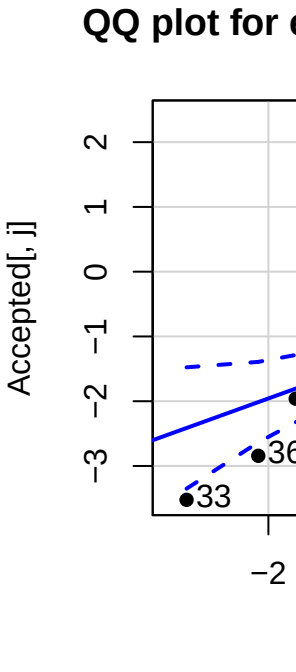

QQ plot for expression of 206903_at in rejected kidneys
Rejected[, j]
norm quantiles
143
133

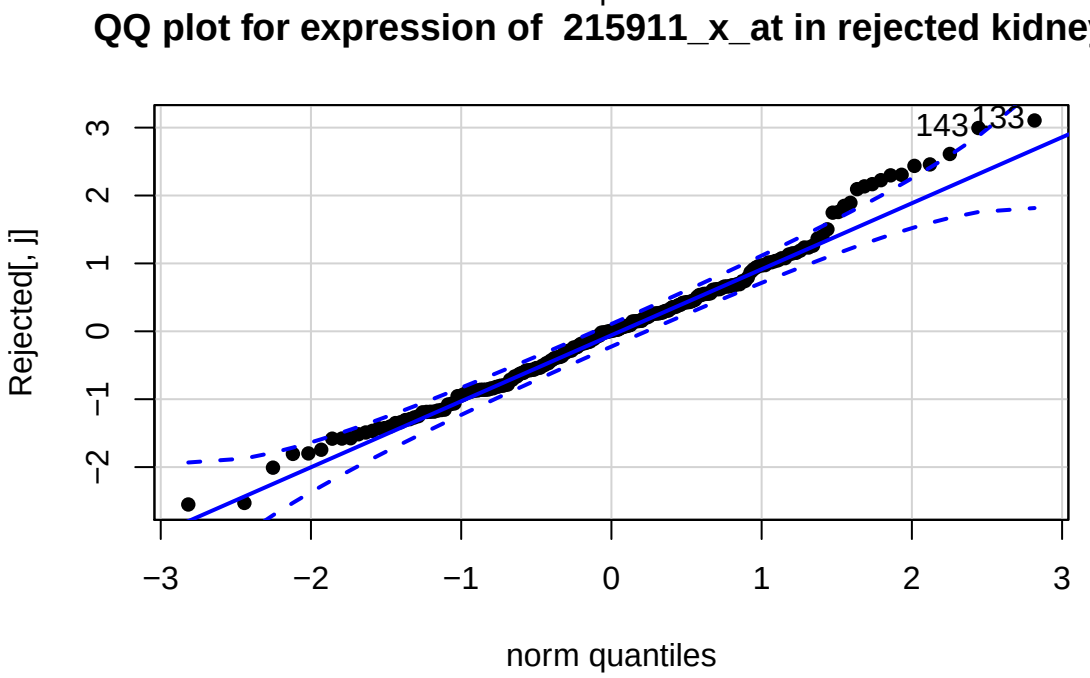

Accepted[, j]
58

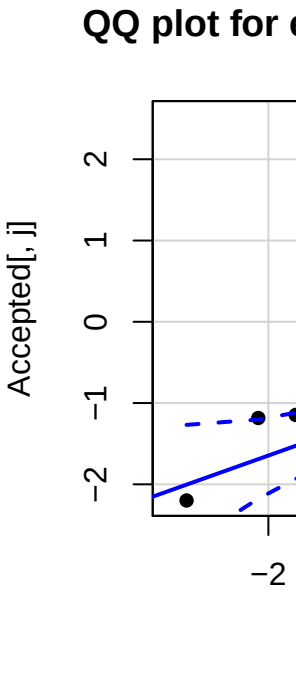

QQ plot for expression of 205118_at in rejected kidneys
Rejected[, j]
norm quantiles
167
79

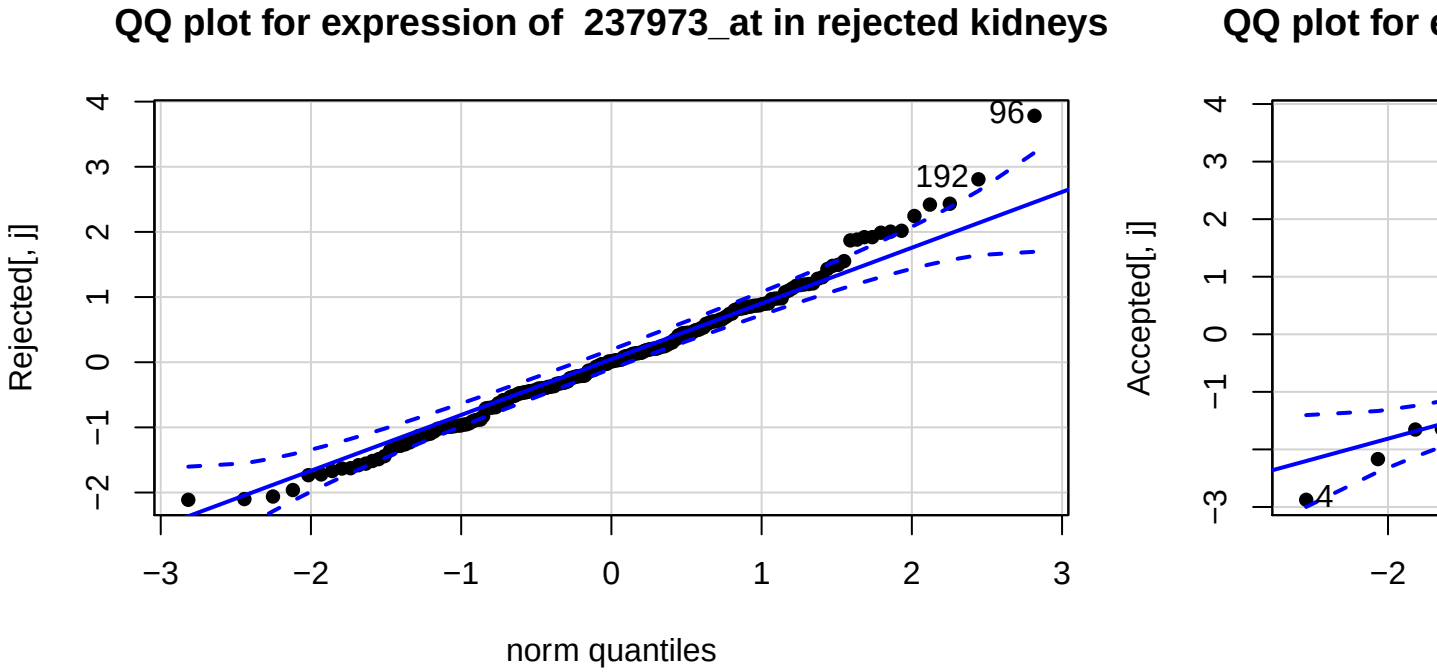

Accepted[, j]

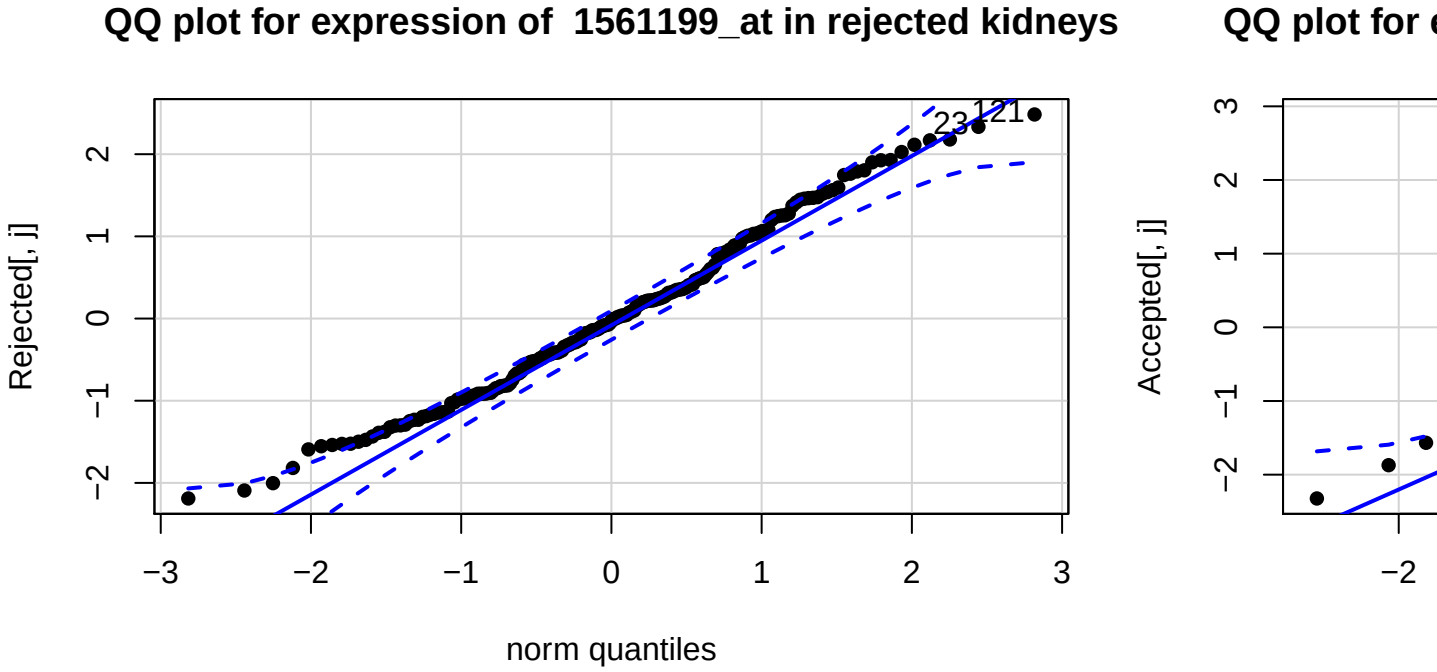

QQ plot for expression of 212532_s_at in rejected kidneys
Rejected[, j]
norm quantiles
29
67


Accepted[, j]
33


QQ plot for expression of 215911_x_at in rejected kidneys
Rejected[, j]
norm quantiles
143
133


Accepted[, j]





The QQ plots show that most genes follow the normal distribution, but there are also genes that do not.

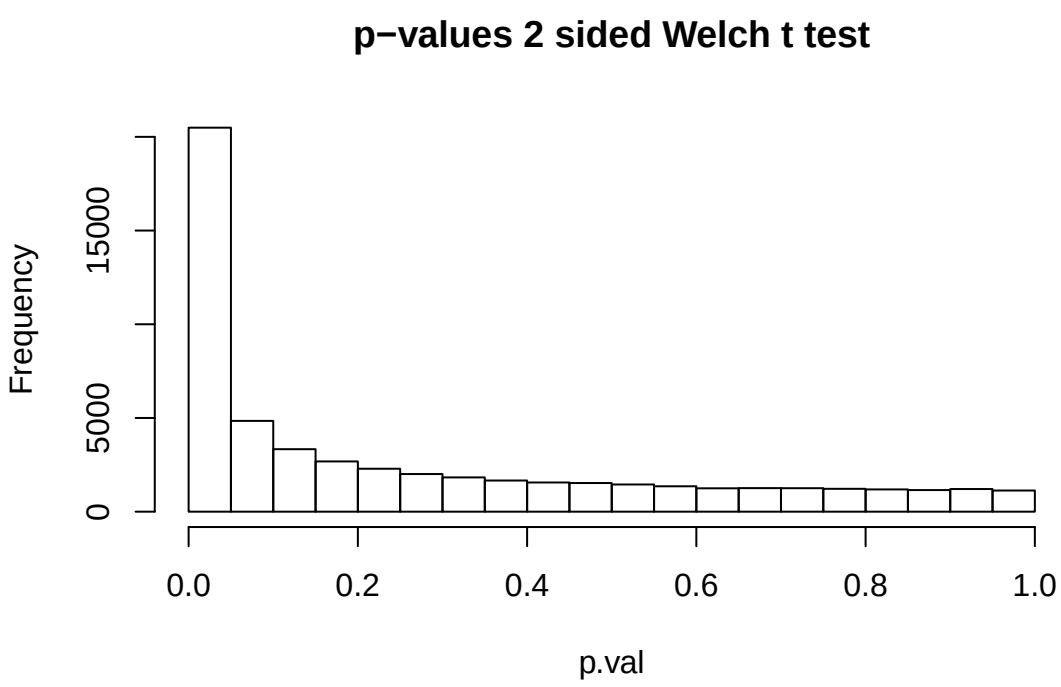
remark: maybe replace by a large-scale hypothesis test for normality later on and then decide for the Welch t-test or Wilcoxon Mann-Whitney (now both are in the code)

(remark on remark: I would rather not do that: On this scale you are guaranteed to have rejections. So in a way you already know the outcome (some are non normal). But if you would allow for a fraction to be non normal, then you have to decide how large that margin would be and then it is again 'guess work'.)

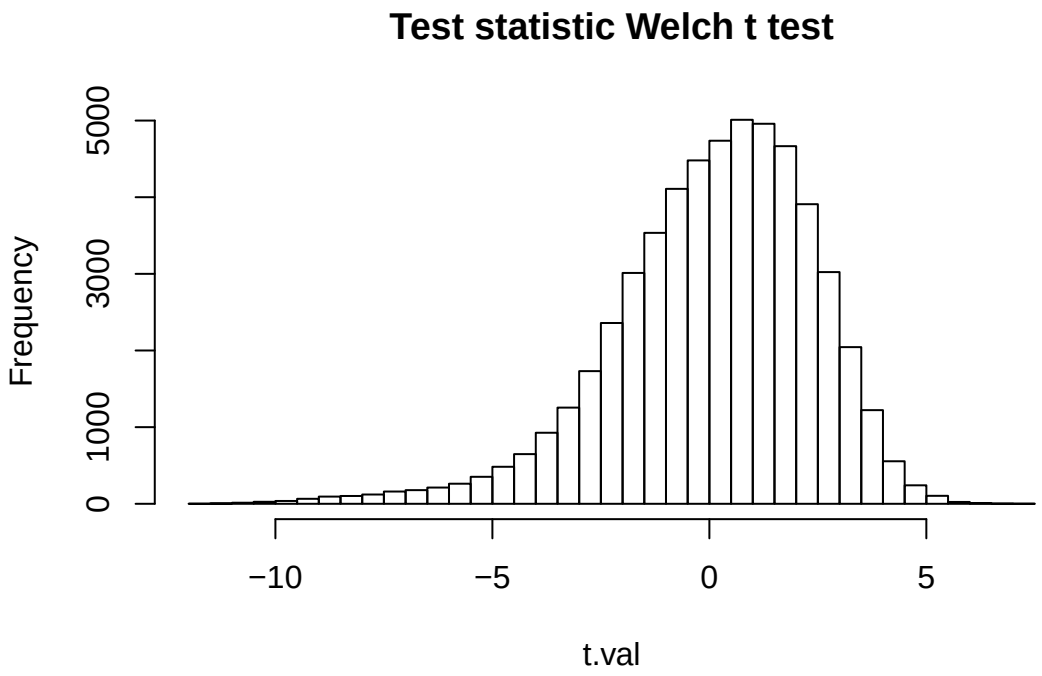
A two-sided two-sample Welch t-test is done on the centered data (not standardized) to determine whether the two groups (rejected and accepted kidneys) can be differentiated based on the gene data, whether the mean of the two groups is different.The Welch t-test is done because of unequal variances/difference in amount of samples between the two groups.

The test takes one hour to run. Therefore, the resulting object is loaded instead.

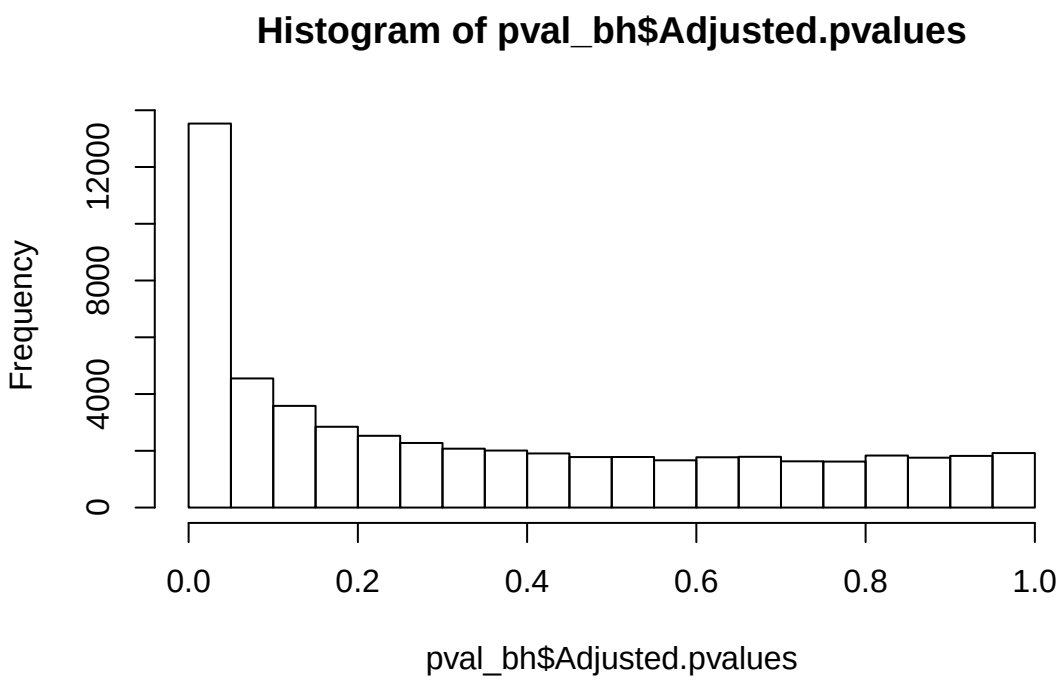
```
# testMTP_Welch-t <- multtest::MTP(X=t(GeneExpression_C), Y = RejectionStatus[,2],
#    na.rm = TRUE, test = "t.twosamp.unequalvar", robust = FALSE,
#    standardize = TRUE , alternative = "two.sided", typeone='fdr',
#     fdr.method =       "conservative", alpha = 0.10)

# duurt niet lang
p.val <- numeric(ncol(GeneExpression_C))
t.val <- numeric(ncol(GeneExpression_C))

for(i in 1:length(p.val)){
   t_object <- t.test(GeneExpression_C[RejectionStatus$Reject_Status == 0,i],
         GeneExpression_C[RejectionStatus$Reject_Status == 1,i],
         var.equal = F)

  p.val[i] <- t_object$p.value
  t.val[i] <- t_object$statistic
}
hist(p.val, main = "p-values 2 sided Welch t test")
```




```
hist(t.val, main = "Test statistic Welch t test", breaks = 50)
```




```
# duurt wel lang (+- 5 minuten):
pval_bh <- BH(p.val, alpha = .1)
hist(pval_bh$Adjusted.pvalues)
```




```
table(p.val < 0.1)
```

```
##
## FALSE  TRUE
## 29343 25332
```

```r
table(pval_bh$Adjusted.pvalues < 0.1)
```

```
##
## FALSE  TRUE
## 36594 18081
```

```r
pval_bh$FDR
```

```
## [1] 0.0486
```

Interpretation: after exercise session 5

A Wilcoxon Mann-Whitney test is done on the centered data, to determine whether the two groups follow the same distribution.

```r
# testMTP_WMW <- multtest::MTP(X=t(GeneExpression_C), Y = RejectionStatus[,2],
#    na.rm = TRUE, test = "t.twosamp.equalvar", robust = FALSE,
#    standardize = TRUE , alternative = "two.sided", typeone='fdr',
#    fdr.method = "conservative",
#    alpha = 0.10)
```

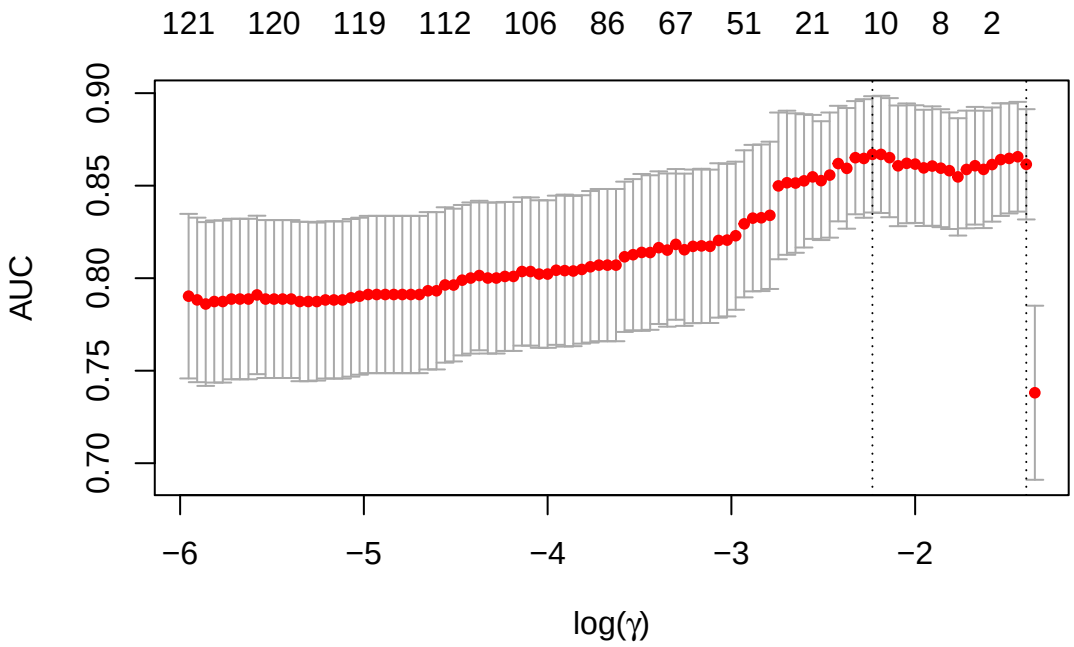
Interpretation: after exercise session 5

4 Prediction of kidney transplant rejection

```r
ind_train <-
  sample(seq_len(nrow(RejectionStatus)), size = floor(nrow(RejectionStatus) * 0.80))

Y_train <- as.matrix(RejectionStatus[ind_train, 'Reject_Status'])
X_train <- as.matrix(GeneExpression_C[ind_train,])
Y_test <- as.matrix(RejectionStatus[-ind_train, 'Reject_Status'])
X_test <- as.matrix(GeneExpression_C[-ind_train,])
```

4.1 Lasso regression

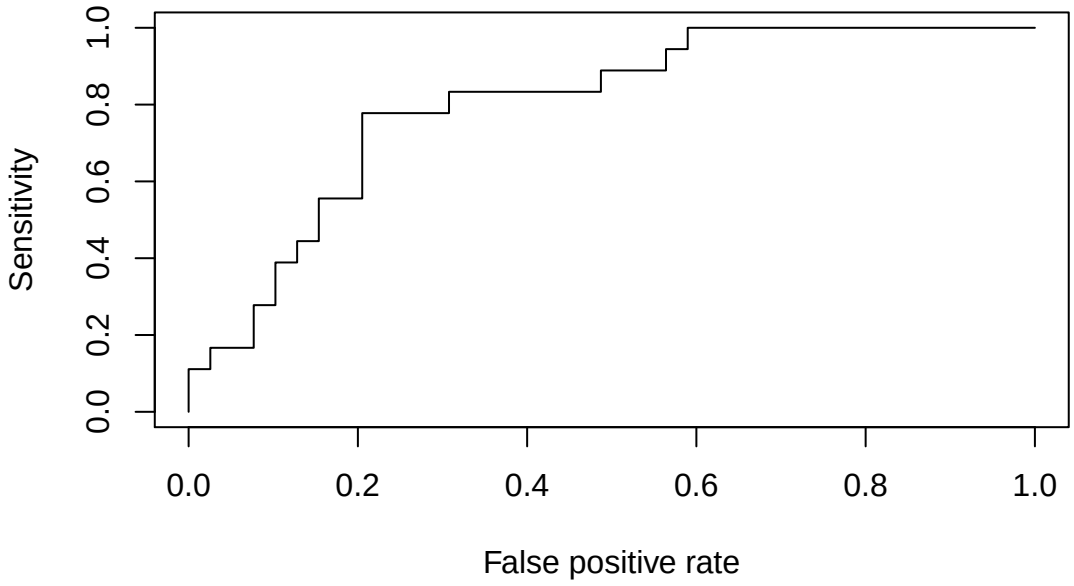
```r
m.cv <-
  cv.glmnet(
    x = X_train,
    y = Y_train,
    alpha = 1,
    family = 'binomial',
    type.measure = "auc"
  )
plot(m.cv, xlab = TeX(" $ log(\\gamma ) $ "))
```



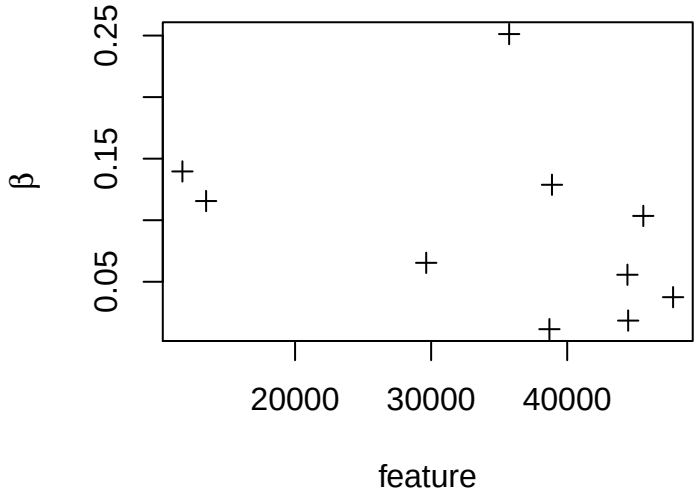
In the figure above, one can see that for γ equal to 0.1072724 , the area under the curve (AUC) is maximal for the train dataset based on a 10-fold cross-validation over the train dataset.

The ROC curve, estimated with the cross-validation dataset, is shown below:

```
m <- glmnet(
  x = X_train,
  y = Y_train,
  alpha = 1,
  family = 'binomial',
  lambda = m.cv$lambda.min
)
pred_m <-
  prediction(predict(
    m,
    newx = X_test,
    type = 'response'
  ),
  Y_test)
perf <- performance(pred_m, 'sens', 'fpr')
plot(perf)
```

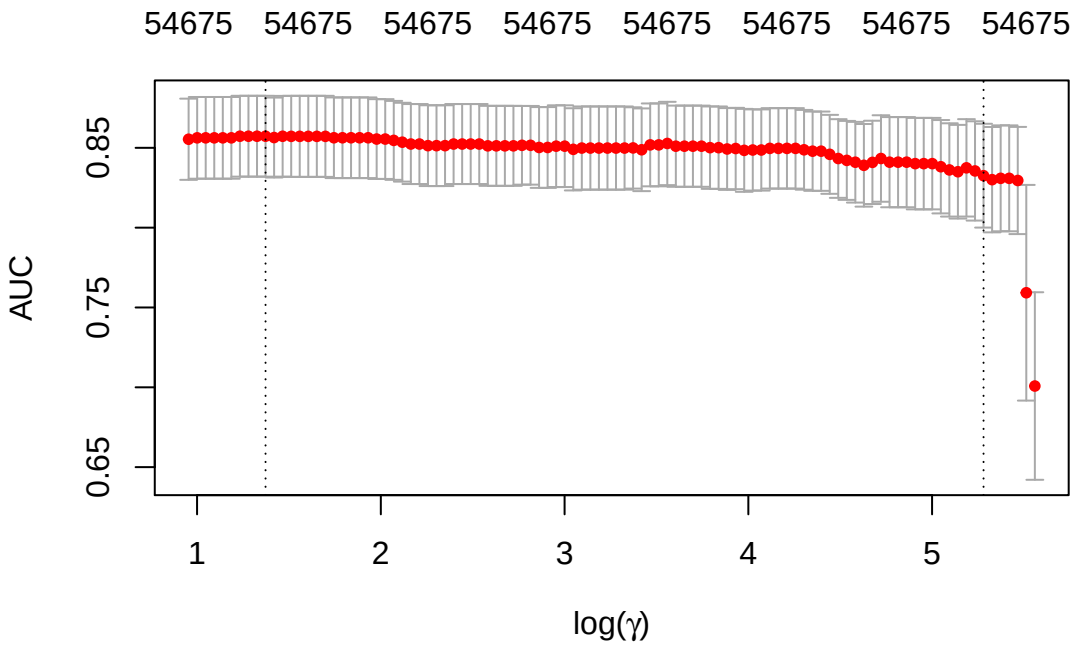


This model uses 10 of the features. This is a considerable dimensional reduction. This is illustrated below. This figure shows the loadings of the 10 selected values.



4.2 Ridge regression

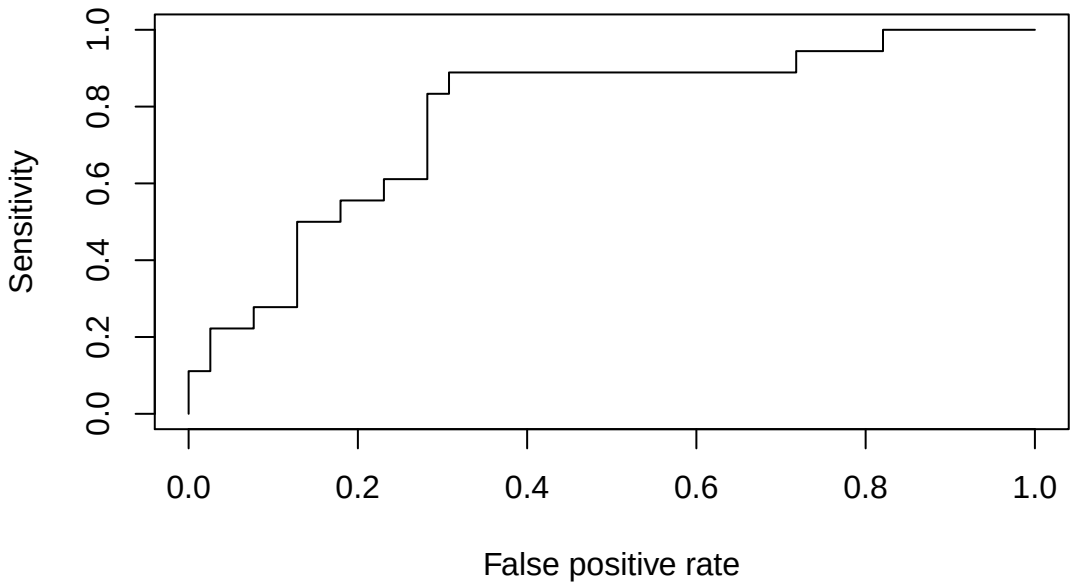
```
m.cv <-
  cv.glmnet(
    x = X_train,
    y = Y_train,
    alpha = 0,
    family = 'binomial',
    type.measure = "auc"
  )
plot(m.cv, xlab = TeX(" $ log(\\gamma ) $ "))
```



In the figure above, one can see that for γ equal to 3.9458848 , the area under the curve (AUC) is maximal for the train dataset based on a 10-fold cross-validation over the train dataset.

The ROC curve, estimated with the cross-validation dataset, is shown below:

```
m <- glmnet(
  x = X_train,
  y = Y_train,
  alpha = 0,
  family = 'binomial',
  lambda = m.cv$lambda.min
)
pred_m <-
  prediction(predict(
    m,
    newx = X_test,
    type = 'response'
  ),
  Y_test)
perf <- performance(pred_m, 'sens', 'fpr')
plot(perf)
```



4.3 Principal component regression

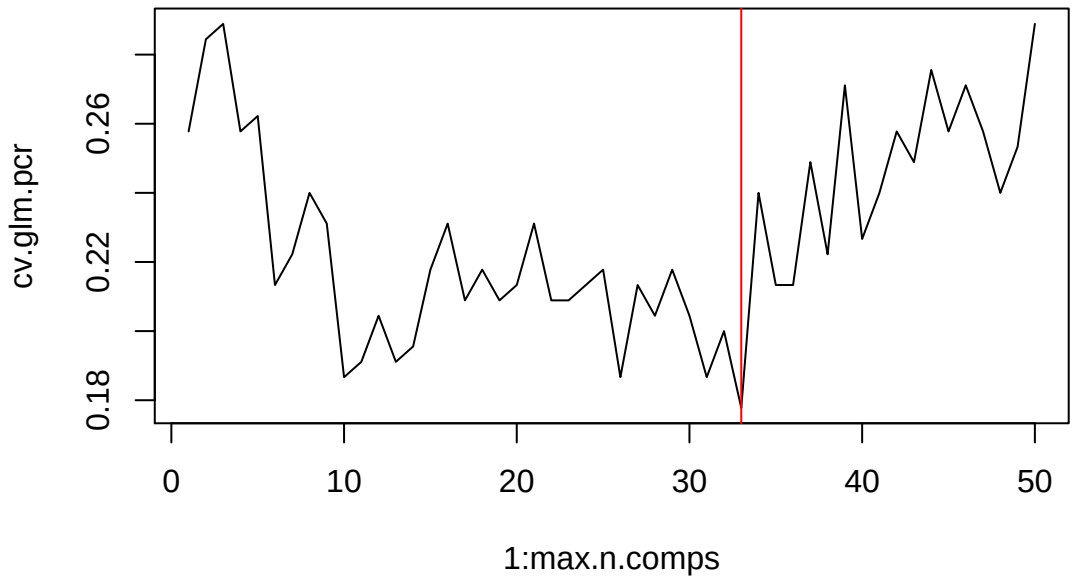
```
# cost function for CV
MISERR <- function(obs, pred, cutoff = 0.5){
  ypred <- as.numeric(pred > cutoff)
  tab <- table(obs, ypred)
  miserr <- 1 - sum(diag(tab))/sum(tab)
  return(miserr)
}

max.n.comps <- 50 #random nr

cv.glm.pcr <- rep(NA, max.n.comps)

X_train.svd <- svd(X_train)

U <- X_train.svd$u
D <- diag(X_train.svd$d)
Z_train <- U%*%D

for(i in 1:max.n.comps){
  fitdata <- data.frame(Y_train, Z_train[,1:i])

  mod <- glm(Y_train ~ ., data = fitdata, family = "binomial")

  cv.glm.pcr[i] <- cv.glm(fitdata, mod, cost = MISERR, K = 10)$delta[1]
}
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

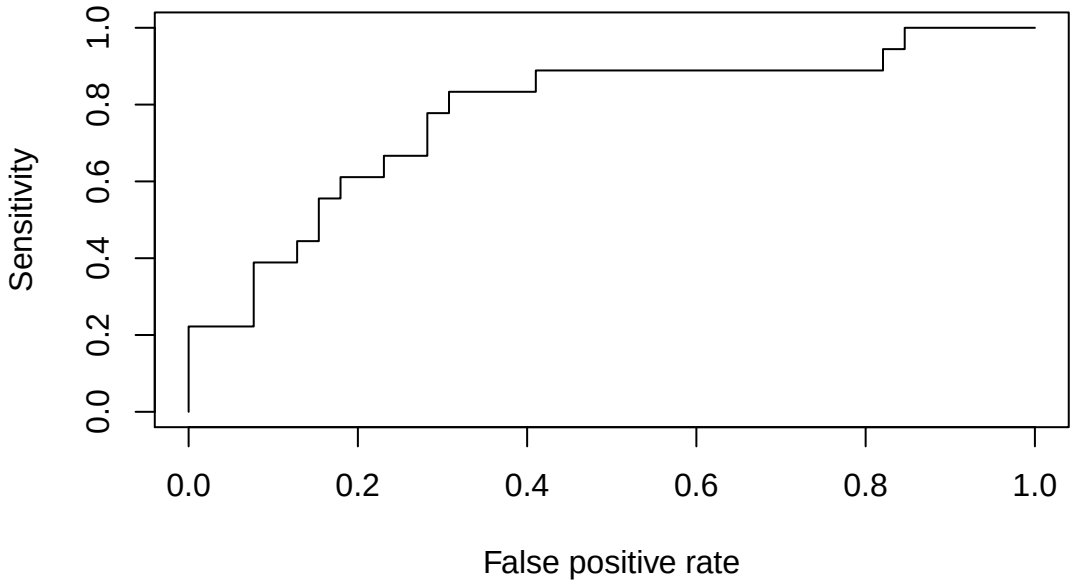
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
npc.min <- which.min(cv.glm.pcr)

plot(1:max.n.comps, cv.glm.pcr, type = "l")
abline(v=npc.min, col =2)
```




```r
V <- X_train.svd$v
Z_test <- X_test %*% V

fitdata <- data.frame(Y_train, Z_train[,1:npc.min])

mod <- glm(Y_train ~ ., data = fitdata)

preddata <- data.frame(Z_test[,1:npc.min])

pred_mod <- prediction(predict(mod, newdata = preddata), Y_test)

perf_mod <- performance(pred_mod, "sens", "fpr")

plot(perf_mod)
```



AUC is 0.776353276353276.