```
# 20.11.13/23.03.14, Pauli Tikka, University of Turku, thesis work
# The complete code for handling the gene data
# Contents:
# mRNA processing
# Step 1: Preliminary set-ups,
# Step 2-5: filtering,
# Step 6: Clustering,
# Step 7: Heat Maps,
# Step 8: Wilcox's tests,
# Step 9: Fisher's exact tests,
# Step 10: Piano package, or alternative pathway processing
# miRNA processing
# Step 11: Preliminary set-ups
# Step 12: Preprocessing of miRNA data
# Step 13: T-tests for miRNAs (defining de-miRNAs)
# Step 14: Performing Gene enrichment with Fisher test in a more elaborated fashion
# Step 15: Linear modelling of miRNAs
# Step 15a: Enhancing the data before the linear modelling scheme
# Step 15b: Defining the lists and mirs for linear modelling
# Step 15c: Clustering and Enrichment of one of the lists
# Step 15d: Linear modelling
# Step 15e: Cross-validation
# Step 15f: Obtaining results after linear modelling analysis

### STEP 1 ###
# Setting working directory
setwd("C:/Users/Pauli/Documents/thesis/labour/analysis/TCGA/results")
# Reading the expression text file as a table (matrix), loading takes ~3 minutes:
real <- read.table(file="tcga.breastcancer.rnaseq2.gem.txt", header = TRUE, row.names=1, sep="\t")
# Transforming the genes to rows:
ret <- t(real)
# Reading Vijay's annotation file as a matrix (anno):
anno <- read.table(file="tcga.breastcancer.rnaseq2.gem.sampleannotation2.txt", header = TRUE, row.names=1,
sep="\t")
# Reading a file for original gene symbols so that the TCGA's exprssion matrix symbols can be interpreted:
gsymb <- read.table(file="genesymbols.txt", header = TRUE, row.names=NULL, sep="\t")
# Notice: row.names=NULL (it caused problems at later stages)
# Constructing an expression set (with annotation matrix (anno), and original transposed expression matrix (ret):
metadata =  data.frame(
  labelDescription = c("Exp","Sampletype","TripleNegative","Gender","Age.at.Initial.Pathologic.Diagnosis",
          "ER.Status","PR.Status","HER2.Final.Status","Tumor","Tumor..T1.Coded","Node","Node.Coded","Metastasis"
,"Metastasis.Coded","AJCC.Stage",
          "Converted.Stage","Vital.Status","OS.event","OS.Time","PAM50.mRNA","RPPA.Clusters"),

row.names=c("No/Exp","normal/tumour","ntn/tn","Male/Female","age","er","pr","her2","tumour","tumourT1code
d","Node","NodeCoded","Metastasis",
          "MetastasisCoded","AJCCStage","ConvertedStage","VitalStatus","OSevent","OSTime","PAM50mRNA","RPPA
Clusters"))
pheDa=new("AnnotatedDataFrame",data=anno, varMetadata = metadata)
eset <- new("ExpressionSet", exprs = ret, phenoData = pheDa)
# The log transform is needed for the heat maps!!
ret_log=log2(ret+1)
exprs(eset)=log2(exprs(eset)+1)
# Discarding outlier patient samples that are more than 3.5 stds away from normal:
library(WGCNA)
allowWGCNAThreads(60)
par(mfrow=c(3,1))
```

```
IAC=cor(exprs(eset),use="p")
hist(IAC,sub=paste("Mean=",format(mean(IAC[upper.tri(IAC)]),digits=3)), xlab= ' IAC values')
cluster1= hclust(as.dist(1-IAC), method="average")
n=eset$Sampletype
n=as.matrix(n)
lgc1=as.matrix(n=='tumour')
n[lgc1[,1],1]=1
meanIAC=apply(IAC,2,mean)
sdCorr=sd(meanIAC)
numbersd=(meanIAC-mean(meanIAC))/sdCorr #Where did this formula came from
table(abs(numbersd)>3.6) #FALSE  TRUE 915     6
plot(numbersd,ylab='Number of standard deviation')
abline(h=-3.5,col='red')
#Removing 5 outliers:
lgc2=abs(numbersd)>3.6
anno=anno[lgc2==FALSE,]
M_0=M_0[,lgc2==FALSE]
e_0=e_0[,lgc2==FALSE]

#Removing suspicious annotation:
lgc3=anno[,2]=='normal'
lgc4=anno[,3]==TRUE #TN status
lgc5=lgc3 & lgc4
M_filt_ok=M_filt_ok[,lgc5==FALSE]
e_filt_ok =e_filt_ok[,lgc5==FALSE]

# Clustering and right groups defined by heatmap analysis, here is the function for heatmap (hm should be
expression matrix):
plot(cluster1,cex=0.7,labels=n,plotMyHeatmap <- function(filename, hm){
  library(pheatmap)
  e_filt_ok$TripleNegative[hm$TripleNegative=='TRUE']<-'TripleNegative'
  hm$TripleNegative[hm$TripleNegative=='FALSE']<-'Control'
  hm$Vital.Status=as.vector(hm$Vital.Status)
  hm$Vital.Status[is.na(hm$Vital.Status)]<-'NoInfo'
  hm$ER.Status=as.vector(hm$ER.Status)
  hm$ER.Status[is.na(hm$ER.Status)]<-'NoInfo'
  hm$ER.Status[hm$ER.Status=='Indeterminate']<-'Other'
  hm$ER.Status[hm$ER.Status=='Performed but Not Available']<-'Other'
  hm$ER.Status[hm$ER.Status=='Not Performed']<-'Other'
  hm$PR.Status=as.vector(hm$PR.Status)
  hm$PR.Status[is.na(hm$PR.Status)]<-'NoInfo'
  hm$PR.Status[hm$PR.Status=='Indeterminate']<-'Other'
  hm$PR.Status[hm$PR.Status=='Performed but Not Available']<-'Other'
  hm$PR.Status[hm$PR.Status=='Not Performed']<-'Other'
  hm$HER2.Final.Status=as.vector(hm$HER2.Final.Status)
  hm$HER2.Final.Status[is.na(hm$HER2.Final.Status)]<-'NoInfo'
  hm$HER2.Final.Status[hm$HER2.Final.Status=='Not Available']<-'NoInfo'
  hm$AJCC.Stage=as.vector(hm$AJCC.Stage)
  hm$AJCC.Stage[is.na(hm$AJCC.Stage)]<-'NoInfo'
  hm$AJCC.Stage[hm$AJCC.Stage=='[Not Available]']<-'NoInfo'
  hm$PAM50.mRNA=as.vector(hm$PAM50.mRNA)
  hm$PAM50.mRNA[is.na(hm$PAM50.mRNA)]<-'NoInfo'
  hm$Metastasis=as.vector(hm$Metastasis)
  hm$Metastasis[is.na(hm$Metastasis)]<-'NoInfo'
  hm$Tumor=as.vector(hm$Tumor)
  hm$Tumor[is.na(hm$Tumor)]<-'NoInfo'
  hm$Age.at.Initial.Pathologic.Diagnosis=as.vector(hm$Age.at.Initial.Pathologic.Diagnosis)
```

```
hm$Age.at.Initial.Pathologic.Diagnosis[is.na(hm$Age.at.Initial.Pathologic.Diagnosis)]<- 0
hm$OS.Time=as.vector(hm$OS.Time)
hm$OS.Time[is.na(hm$OS.Time)]<- 0

annotation = data.frame(Var1 = hm$Sampletype, Var2 = hm$TripleNegative, Var3 =
                          hm$Vital.Status,
                        Var4 = hm$ER.Status, Var5 = hm$PR.Status, Var6 = hm$HER2.Final.Status,
                        Var7 = hm$AJCC.Stage, Var8 = hm$PAM50.mRNA, Var9 = hm$Metastasis,
                        Var10=hm$Tumor, Var11 = hm$Age.at.Initial.Pathologic.Diagnosis, Var12 =
                          hm$OS.Time)
rownames(annotation) <- colnames(hm)
#head(annotation)
annotation$Var1 = factor(annotation$Var1, levels = c("normal", "tumour"))
annotation$Var2 = factor(annotation$Var2, levels = c("Control", "TripleNegative"))
annotation$Var3 = factor(annotation$Var3, levels = c("LIVING", "DECEASED", "NoInfo"))
annotation$Var4 = factor(annotation$Var4, levels = c("Positive", "Negative", 'Other',
                                     "NoInfo"))
annotation$Var5 = factor(annotation$Var5, levels = c("Positive", "Negative", 'Other',
                                     "NoInfo"))
annotation$Var6 = factor(annotation$Var6, levels = c("Positive", "Negative", 'Equivocal',
                                     "NoInfo"))
annotation$Var7 = factor(annotation$Var7, levels = c("Stage I", "Stage IA", "Stage IB", "Stage
                                     II", "Stage IIA", "Stage IIB", "Stage III",
                                     "Stage IIIA", "Stage IIIB", "Stage IIIC", "Stage IV", "Stage X",
                                     "NoInfo"))
annotation$Var8 = factor(annotation$Var8, levels = c('Normal-like', 'Basal-like', 'HER2-
                                     enriched', 'Luminal A', 'Luminal B', 'NoInfo'))
annotation$Var9 = factor(annotation$Var9, levels = c("M0", "M1", "NoInfo"))
annotation$Var10 = factor(annotation$Var10, levels = c("T1", "T2", 'T3', "T4", 'TX', "NoInfo"))
# Specify colors
Var1 = c("lightgrey", "black")
names(Var1) = c("normal", "tumour")
Var2 = c("steelblue1", "red")
names(Var2) = c("Control", "TripleNegative")
Var3 = c("lightgreen", "darkgreen","grey")
names(Var3) = c("LIVING", "DECEASED","NoInfo")
Var4 = c("red", "green", "black", "grey")
names(Var4) = c("Positive", "Negative", 'Other', "NoInfo")
Var5 = c("red", "green", "black", "grey")
names(Var5) = c("Positive", "Negative", 'Other', "NoInfo")
Var6 = c("red", "green", "black", "grey")
names(Var6) = c("Positive", "Negative", 'Equivocal', "NoInfo")
Var7 = c("aquamarine", "aquamarine3", "aquamarine4", "deepskyblue", "blue", "navy",
       "pink", "hotpink", "orangered", "red", "magenta", "black", "grey")
names(Var7) = c("Stage I", "Stage IA", "Stage IB", "Stage II", "Stage IIA", "Stage IIB", "Stage
            III",
            "Stage IIIA", "Stage IIIB", "Stage IIIC", "Stage IV", "Stage X", "NoInfo")
Var8 = c("oldlace","pink4", "seagreen", "royalblue4", "purple3", "grey")

names(Var8) = c('Normal-like', 'Basal-like', 'HER2-enriched', 'Luminal A', 'Luminal B',
            'NoInfo')
Var9 = c("white", "black", "grey")
names(Var9) = c("M0", "M1", "NoInfo")
Var10 = c("yellow", "wheat4", "springgreen", "thistle1", "olivedrab", "grey")
names(Var10) = c("T1", "T2", 'T3', "T4", 'TX', "NoInfo")
Var11 = c("red")
Var12 = c("red")
```

```
  ann_colors = list(Var1 = Var1, Var2 = Var2, Var3 = Var3, Var4 = Var4,
                Var5 = Var5, Var6 = Var6, Var7 = Var7, Var8 = Var8,
                Var9 = Var9, Var10 = Var10, Var11 = Var11, Var12 = Var12)
  pdf(file = filename,width=11, height=11, pointsize=12)
  pheatmap(exprs(hm),fontsize=8, scale='column', show_rownames = F, show_colnames =
           F,annotation=annotation, annotation_colors = ann_colors)
  dev.off()
}
plotMyHeatmap(filename='legendarium', e_filt_ok) #time to plot (more than 8 minutes): 13:22-30='Clustering of
samples')
#The results of heatmap evaluation maybe checked with kmeans:
km_val_mfl=kmeans(M_filt_ok, 5)
# get cluster means
aggregate(M_filt_ok,by=list(km_val_mfl$cluster),FUN=mean) #14:56-57
# append cluster assignment
mydata <- data.frame(M_filt_ok, km_val_mfl$cluster)
# Principal component analysis can also be used for checking the correct groups:
prcomp(M_filt_ok[1:100,1:100])

# Subset expression set for modified TN and normal groups:
load('breastCancerTCGAtriplenegativeVSnormal.Rdata')


### STEP 2 ###
#Discarding the zero values:
par(mfrow=c(1,2))
hist(unlist(ret))
hist(unlist(ret_log))
## Two approaches follows:
pdf("mean_exp.pdf")
mean_exp=apply(ret_log,1,mean)          # check this by eye ->  take the tale away using proper cut-off such as 1.5, #
see sd and var also
hist(mean_exp)                          # mem=mean(mean_exp) ~ 6.3 cannot be used here, because then too
much values are discarded
dev.off()
# within one gene (i.e. ABC in row 123, i.e. "1" in apply), how many patient samples, have expressed that gene above
threshold value of 1.5:
sum=apply(ret_log > 5, 1, sum)
# There can be e.g. 175 genes expressed above 1.5 across the whole 1100 patient samples, so the percentage is 175
/ 1100 * 100, and in general:
a = sum/dim(ret_log)[2] * 100
# Finally select only those GENES (in rows (r) in e.g. matrix A(r,c)) that have expression across the all patient samples
above 50%:
M_0 = ret_log[ a > 10,]
dim(M_0)
#[1] 15337  1100
# Apply the same functions also to your gene symbol list (so that the row numbers keep the same) and expression
set matrix
G_0 = gsymb[ a > 10,]
e_0 = eset[ a > 10,]
hist(unlist(M_0))

### STEP 4 ###
# IQR filtration
GeneIQR=apply(M_0,1,IQR)
GeneIQRg=apply(G_0,1,IQR)
GeneIQRm=mean(GeneIQR)
```

```
GeneIQRm
#[1] 1.313247
## or check the plot
pdf("GeneIQR.pdf")
hist(GeneIQR)
dev.off()
# All off the values which gives 'TRUE' (according to the statement) in
#'which' command (that produces logical values (true/false)) are discarded (-)
M_IQR <- M_0 [ - which (GeneIQR <= 2) , ]
dim(M_IQR)
#2597 1100
G_IQR = G_0[ - which (GeneIQR <= 2) ,]
e_IQR = e_0[ - which (GeneIQR <= 2) ,]
#this is kind of much so I do also variance filtering

### STEP 5 ###
# Variance filtering
GeneVar=apply(M_IQR,1,var)
GVM=mean(GeneVar)
GVM
#2.888428
## or check the plot
pdf("GeneVar.pdf")
hist(GeneVar)                    ## ok, in this case the value for cut-off is higher than the mean value, so I
choose 4
dev.off()
M_filt_ok <- M_IQR [ - which (GeneVar <= 4) , ]
dim(M_filt_ok)
#[1] 1157 1100
G_filt_ok = G_IQR[ - which (GeneVar <= 4) ,]
e_filt_ok = e_IQR[ - which (GeneVar <= 4) ,]

## STEP 6 ###
# CLUSTERING for filtered genes:
method="complete"
distance="euclidean"
pdf('clust_all.pdf')
clust=hclust(dist(t(exprs(e_filt_ok)),method=distance),method=method) #ok
plot(clust, labels=F)
dev.off()

## STEP 7 ###
# Heat map generated with a function, if you do not want to see the names of
# the patient samples and filtered genes, insert F for rownames and columnnames in the pheatmap:
plotMyHeatmap <- function(filename, heat){
  library(pheatmap)
  heat$TripleNegative[heat$TripleNegative=='TRUE']<-'TripleNegative'
  heat$TripleNegative[heat$TripleNegative=='FALSE']<-'Control'
  annotation = data.frame(Var1 = heat$TripleNegative, Var2 = heat$Sampletype)
  rownames(annotation) <- colnames(heat)
  #head(annotation)
  annotation$Var1 = factor(annotation$Var1, levels = c("Control", "TripleNegative", "NA"))
  annotation$Var2 = factor(annotation$Var2, levels = c("normal", "tumour"))
  # Specify colours:
  Var1 = c("steelblue1", "red", "yellow")
  names(Var1) = c("Control", "TripleNegative")
  Var2 = c("lightgrey", "black")
```

```
  names(Var2) = c("normal", "tumour")
  ann_colors = list(Var1 = Var1, Var2=Var2)
  pdf(file = filename,width=11, height=11, pointsize=12)
  pheatmap(exprs(heat),fontsize=8, scale='column', show_rownames = T, show_colnames = T,annotation=annotation,
annotation_colors = ann_colors, main="21.11.2013, Pauli Tikka, Heat map")
  dev.off()
  }
# Insert a name of the file, and the filtered expression set matrix (e_filt_ok) for the arguments of this heat map
function:
plotMyHeatmap('Tikka_heat_map_2.pdf',e_filt_ok)                    # with this function one cannot add "x, and y labels",
they need to be added with picture editor.

### STEP 8 ###
# Wilcox's tests, (t-tests similarly, with loop)

# Gene symbol list with only true values:
g_e=gsymb[(eset$Exp == T ),]
# Normal samples:
esn=exprs(esetSub[,esetSub$Sampletype=='normal'])
#Triple negative samples:
est=exprs(esetSub[,esetSub$TripleNegative=='TRUE'])

# with real (r) groups (est, esn (normi)):
wr=1:20531 * 0
for(i in 1:20531){hr=wilcox.test(est[i,],esn[i,]);wr[i]=hr[[3]];}
hist(log10(wr),100)
# Bonferroni method: every p-value is multiplied by the number of performed tests (here: 20,000 (genes))
# Do this analysis again!:
swr=sum(wr[is.nan(wr)==F]<1e-25)
s_r=wr[is.nan(wr)==F]<1e-25
# For gene list:
gr=g_e[s_r,]
grf=gr[,1] # Only Entrez number is required at next tests

### STEP 9 ###
# Fisher's exact tests
# This function works but the matrix is not good and it needs the real pathway names, and not the KEGG pathway
names
pwy_func2=function(pathway.list2,grfx,s_r) {
  for (i in 1:length(pathway.list2)) {

    de_tot=length(grfx)
    g_tot=length(s_r)
    p.val=1:length(pathway.list2) * 0;

    pwy_oxp_l = length(pathway.list2[[i]]);
    pwy_oxp_de=sum(grfx %in% pathway.list2[[i]]);
    pwy_de = de_tot - pwy_oxp_de;
    pwy_oxp_n = pwy_oxp_l - pwy_oxp_de;
    pwy_nn = g_tot - de_tot - pwy_oxp_n;
    fpgs = matrix(c(pwy_oxp_de, pwy_de, pwy_oxp_n, pwy_nn), 2, 2);
    p.val = fisher.test(fpgs);

    at=print(cbind(names(pathway.list2[i]),p.val[1]))
    #if (p.val <= 0.05) {cbind(names((pathway.list2[i])),p.val[1])}
    #if (as.matrix(p.val) <= 0.05) return(p.val)
    #print(as.vector(names(pathway.list2[i])))
```

```
        ##### HERE SHOULD BE RETURN ######
   }
}

### STEP 10 ###
# Piano package, or alternative pathway processing
# One can apply gsea, and page with runGSA function.
# Gene sets are pathways. Instead of g2pwyid, there could be some other list of genes.
myGsc <- loadGSC(g2pwyid)
GSA = runGSA(...
# qusage package could be also applied but the enrich internet page gives you all this more easily, and faster
(http://amp.pharm.mssm.edu/Enrichr/index.html)

#### miRNA processing ####
### STEP 11 ###
# Preliminary steps for microRNA data:
# This is just the directory address where you have saved your TCGA data:
path_to_files = "C:/cygwin64/home/Pauli/miRNA/Level_3"
# Reading all the file names from this directory
files = list.files(path_to_files, pattern="mirna.quantification",all=F)
# Just reading the first file to know how many rows (i.e how many miRNA) are there in the file
tmp <- read.table(paste(path_to_files, files[ 1 ] ,sep = "" ), header=T,as.is=T,sep="\t")
# Creating an empty matrix where to combine all the files
data <- matrix(nrow = nrow (tmp))
# Naming the rows of the data matrix
rownames(data) <- tmp$miRNA_ID
# Removing the tmp variable that we wont use anymore
rm(tmp)
# Loop over all the files to read
for(i in 1:length(files)) {
  tab  <-  read.table(paste(path_to_files, files[ i ] ,sep = "" ), header=T,as.is=T,sep="\t") ## This reads the first file and
so on over the loop of all files
  if(nrow(tab) == nrow(data)) {
  tab  <- tab[order(match(rownames(data),rownames(tab))),]
  data = cbind(data, tab$reads_per_million_miRNA_mapped) ## This will read only the column that you would like to
have i.e read_per_million
  } }
data <- data[, -1]
# Naming the columns with the patient names (IDs)
colnames(data) <- substr(files,1,15)
## OR ##
# The way to import all the (miRNA) data (including miRNA isoform expressions) from
# a directory's files to an R list is to use foreign package # http://www.ats.ucla.edu/stat/r/pages/read_multiple.htm
library(foreign)
# This is the directory:
path_to_files = "C:/cygwin64/home/Pauli/miRNA/Level_3/"
# Reading all the file names from this directory
files = list.files(path_to_files, pattern="13.isoform.quantification",all=F)
# Construct a variable that has the path of directory and the file names to read as a table (or list in this case) in R
fa <- file.path(path_to_files, files)
# Lapply command has several functions such as read.dta or read.csv, check the options from above internet site
da <- lapply(fa, read.table)
# Selecting certain values of this list: da_test=c()
for (i in 1:length(dat))( da_test=qpcR:::cbind.na(da_testi, dat[[i]][6]), dat[[i]][4]) #so far so good...
# Then renaming the matrix
sum_mir_dvl=da_test[,-1]
# Preparing the sum matrix from sum_mir_dvl
```

```
# Remove star/mature beginnings from names
# First making the empty matrix: su_mmat=matrix(nrow=7702, ncol=1540) su_mmat = "[<-"(su_mmat,value=0)
# This loop is done three times (so i times and then repeated 3 times), where the pattern is first mature, and then
star
for (i in 1:770)(#su_mmat[,(i*2)]=gsub(pattern="star,", replacement="", su_mmat[,(i*2)], ignore.case = FALSE, perl =
FALSE, fixed = FALSE, useBytes = FALSE)
su_mmat[,((i*2)-1)] = sum_mir_dvl[,((i*2)-1)])
#Changing the colnames of the new matrix:
colnames(su_mmat)=colnames(sum_mir_dvl)
# This is the long wanted sum matrix for mirnas with corresponding mature names and their sum expression values:
# (note: agstest=matrix(nrow=1200, ncol=1540) agstest = "[<-"(agstest,value=0) =
for (i in 1:770)  (agstest=qpcR:::cbind.na(agstest,aggregate(su_mmat[,(i*2-1)], list(su_mmat[,(i*2)]), sum)) )
# Defining the number of rows and their names in agstest matrix according to unique MIMAT-names in agstest in
overall:
# http://stackoverflow.com/questions/12154814/r-get-a-single-column-from-many-columns
result <- data.frame(Wave = unlist(atf,use.names=FALSE))
agstest_rn=unique(result) ## jee
# rownames should not have na-values:
rownames(agstest)=agstest_rn[is.na(agstest_rn)==F,1]
# Then one needs to merge the correct rownames (of agstest matrix) and agstest-mimat names (at every second
column) to obtain the overall matrix dtt:
#defining the new variable before the looping solution:
dtt=matrix(nrow=1048,ncol=1540)
dtt = "[<-"(dtt,value=0) #from art of  r programming
dtt=data.frame(dtt)
# The use of merging function was found at (see below), and then applied for every column:
# http://r.789695.n4.nabble.com/How-to-compare-match-two-columns-from-diferent-dataframe-and-assign-values-
from-one-datafram-to-the-r-td2544573.html
for (i in 1:770) (
dtt[,((i*2-1):(i*2))]=merge(data.frame(x=rownames(agstest)),
                    data.frame(x=agstest[,(i*2-1)],y=agstest[,(i*2)]),by='x',all.x=T) )
# Because dtt was an empty matrix the rownames, and column names must be inserted again, and also removing not
needed extra-mimat-names, that where
# needed for the merging
dtt=dtt[,(rep(c(FALSE,TRUE),770))]
rownames(dtt)=rownames(agstest)
# Some repetition from previous stages is good:
path_to_files = "C:/cygwin64/home/Pauli/miRNA/Level_3/"
files = list.files(path_to_files, pattern="13.isoform.quantification",all=F)
colnames(dtt) <- substr(files,1,15)

#Tuschen, new:
tush_n <- read.table("TS-PT-PI_21TRUE_complete.txt",fill=TRUE)
#Selecting just the miR-infos:
m <- grep("hsa-*", tush_n[,1], value=FALSE)
tush_n=tush_n[m,]
# Preparing the rownames, and deleting an extra column, and NA rows:
x <- tush_n[,1]
y <- gsub(":","",x,fixed=T)
rownames(tush_n)=y
tush_n=tush_n[,-1]
#Replacing empty spots as nas (this is important)
tush_n[tush_n==''] <- 'na'
#  make a list of tush_n
tush_nl=lapply(seq_len(nrow(tush_n)), function(i) tush_n[i,])
for (i in 1:length(tush_nl)) (
tush_nl[[i]]=as.vector(unlist(tush_nl[[i]])) ) # Ok, now it looks the same
```

```
#Na-values (just values, not rows) away:
tush_nl <- lapply(tush_nl, function(x) x[!is.na(x)]) # wei hou
# Naming the list values
names(tush_nl)=rownames(tush_n)

### STEP 12 ###
# Preprocessing of miRNA data #
# Selecting nas away from data:
data_in = data
data_in[is.na(data_in)] <- 0
# Selecting zeros away from data
v=rep(1,dim(data_in)[1])
for (i in 1:dim(data_in)[1]){
  sum=sum(data_in[i,]> 10)/dim(data_in)[2] # 10 is ~1% of mean(data_in)
  v[i]<-sum}
datv=data_in[v>0.1,] # sample values should be above 10 for more than 10% of a particular miRNA

## Annotation and expression redefined
x <- rownames(annoi)
y <- colnames(datv)
matches     <- which (x %in% y)
a1 <- annoi[matches,]
e1 <- expv[,matches]
# Ordering the datas:
e1=e1[,order(colnames(e1))]
a1=a1[order(rownames(a1)),]
datv=datv[,order(colnames(datv))]
# Matching the datav to the annotation (a1)
y <- rownames(a1)
x <- colnames(datv)
matches     <- which (x %in% y)
datv2 <- datv[,matches]

#Discovering the non-equivalent samples and discarding them
colnames(datv2)==rownames(a1)
colnames(datv2)[117:127] # checking the FALSE sites
colnames(e1)[117:127]    # comparing FALSE sites to the ok sites
datv2=datv2[,-119] # deleting the FALSE (duplicate) samples

### STEP 13 ###
## Wilcox tests are needed for skewed data ##
# annotation, expression, and mirna data (after preprocessing of 0s and NAs and matching):
dim(a1)
dim(e1)
dim(datv2)
# Expression set for the ok anno, exp, and data groups:
metadata2 =
  data.frame(labelDescription =
c("Exp","Sampletype","TripleNegative","Gender","Age.at.Initial.Pathologic.Diagnosis","ER.Status","PR.Status","HER2.
Final.Status","Tumor","Tumor..T1.Coded","Node","Node.Coded","Metastasis","Metastasis.Coded","AJCC.Stage","Co
nverted.Stage","Vital.Status","OS.event","OS.Time","PAM50.mRNA","RPPA.Clusters"),

row.names=c("No/Exp","normal/tumour","ntn/tn","Male/Female","age","er","pr","her2","tumour","tumourT1code
d","Node","NodeCoded","Metastasis","MetastasisCoded","AJCCStage","ConvertedStage","VitalStatus","OSevent","O
STime","PAM50mRNA","RPPAClusters"))
pheDa2=new("AnnotatedDataFrame",data=a1, varMetadata = metadata2)
eset_i <- new("ExpressionSet", exprs = datv2, phenoData = pheDa2)
```

```
# own mirna data experssion
e_d=as.matrix(colnames(exprs(eset_i[1,]))) #567, maybe not needed: e_d=e_d[order(e_d)]
# vijay normal column names
e_n=(as.matrix(colnames(exprs(esetSub[,esetSub$Sampletype=='normal']))))  #87
e_n=e_n[order(e_n)]
# Vijay tn
e_t=(as.matrix(colnames(exprs(esetSub[,esetSub$TripleNegative==TRUE]))))
e_t=e_t[order(e_t)]
# for normals and tns (FINALLY ok)
matches = (e_d %in% e_n)
normals=exprs(eset_i[,matches])
tneg=exprs(eset_i[,matches])
# The de-miRNA:)
normals_lg=log10(normals+1)
tneg_lg=log10(tneg+1)

## Eli tää toimii (Wilcox skewille datalle login jälkeen duplikatet poistettuna ja dimmin sijaan
# loopissa on vain dim-arvot sellaisenaan)
# check also that tneg_lg and normals_lg are as numeric (which maybe applied with as.matrix...) (and not data.frame)
# P-values for all of the genes are also needed, i.e. Wilcox for normals (nrm_i) and tns (t_n3) at e11_lg
matches = (e_d %in% e_n)
nrm_i=e11_lg[,matches]
t_n3=e11_lg[,matches]

#  changing data.frames as numeric with as.matrix
nrm_i=as.matrix(nrm_i)
t_n3=as.matrix(t_n3)
# Differentially expressed genes with Wilcox test :
wto=as.vector(1:16458)
wto="[<-"(wto,1:16458,value=0)
for(i in 1:length(wto))
{hr=(wilcox.test(t_n3[i,],nrm_i[i,])); wto[i]=hr[[3]];}
# Bonferroni correction: Take the p-value of each gene and multiply it by the number of genes in the gene list:
wto2=wto*16458
hist(log10(wto2), breaks=25)
# Checking the de_genes:
rownames(t_n3)
de_gen=cbind(rownames(t_n3),as.numeric(wto2))
colnames(de_gen)<-c('gen','pval')
# Selecting the best differentially expressed genes:
de_gen_lgi = de_gen[,2] < 0.05
# dif.exp. mir:s that are ok, according to the pval (in the second column):
de_geno=de_gen[de_gen_lgi,] # dim(de_geno)[1] is needed for Fisher Test

### STEP 14 ###
# Performing Gene enrichment with Fisher test in a more elaborated fashion
# Checking if the dtt or dttv2 data was skewed:
dtt2=as.matrix(dtt)          # Somehow my data was not numeric, so I changed so whit as.matrix command
dttv3=as.matrix(dttv2)
hist(log10(dttv3+1), breaks=50) # ok, the data was skewed so using Wilcox was ok. Log. trans needed:
dttv3_lg=log10(dttv3+1)

# Starting points for miRNA enrichment:
de_miro3        # differentially expressed mature miRNAs
targets_ok      # the miRNA targets (change the rownames, and na. info, and delete the first column)
gsymb2          # gsymbols for entrez to genes
a11             # Annotation, See 6114_pt, and 141213_pt files (not sure if needed, if I have tn/nrm grps)
```

```
e11            # Expression matrix, See 6114_pt, and 141213_pt files (no log trans done yet)
e11_lg=log10(e11+1)  # log -transformation for expression matrix
normals2_lg     # log transformed normal group
normals3                    # normals group with mimat names, and vijay 70 patient nrms
tneg2_lg       # log transformed tn group

# Processing the matrices
targets_ok=read.table("targets_ok.txt", header=T, sep="\t")
# rowname checks for targets_ok and e11_lg
rownames(targets_ok)= targets_ok[,1]
rownames(targets_ok)= sub("(r)", "\\U\\1", rownames(targets_ok), perl=TRUE)
targets_ok=targets_ok[,-1]
rownames(e11_lg) = sub("X", "", rownames(e11_lg), perl=TRUE)

# Up and downregulation for de-Genes, Pre-values
matches = (e_d %in% e_t)
nrm_e=e11[,matches]
tneg_e=e11[,matches]
nrm_e=as.matrix(nrm_e)
tneg_e=as.matrix(tneg_e)
# The regulation of genes:
h=as.vector(1:dim(nrm_e)[1])
h = "[<-"(h,1:dim(nrm_e)[1],value=0)
for (i in 1:dim(nrm_e)[1]) (
 h[i]= (median(tneg_e[i,]) - median(nrm_e[i,])))
gen_rgl=cbind(gsymb2[,1],h)

# de_miRNA list values (demot2) must to be compared to gene names of the good p-values (de_geno)
# Function for this
f_match_down=function(de_geno, demot2) {
  big=as.character(de_geno[,1])
  tmd=NULL
  #smalls
  for (i in 1:length(demot2)) (
  tmd[i]=list(big[(big %in% as.character(unlist(demot2[i])))]))
  return(tmd) } #jee

f_match_down2= function (hero, gen_rgl) {
    h=NULL
    tv=gen_rgl[,2]<0
    big=as.character(gen_rgl[tv,1])

    #smalls
    for (i in 1:length(hero)) (
    h[i]=list(big[(big %in% as.character(unlist(hero[i])))]))
    return(sapply(h, length))
  }

f_match_down3=function(de_geno, demot2) {
  big=as.character(gen_rgl[,2])
  #smalls
  for (i in 1:length(demot2)) (
    tmd[i]=list(big[(big %in% as.character(unlist(demot2[i])))]))
  return(tmd)}

# genes up (when mirs down)
f_match_down4 =function (hero4, gen_rgl) {
```

```
h=NULL
tv=gen_rgl[,2]>0
big=as.character(gen_rgl[tv,1])
#smalls
for (i in 1:length(hero4)) (
  h[i]=list(big[(big %in% as.character(unlist(hero4[i])))]))
return(sapply(h, length))}

# This function might be useful:
f_miR_enrich_fish = function (tush_nl,de_miro3,de_geno,gen_rgl) {
#You need a list of miRNA targets (tush_nl), differentially expressed miRNAs (de_miro3),
# differentially expressed mRNAs (de_geno), and finally log fold change values for genes (only I would say); gen_rgl

# Finding targets (trg) of de_mat_mirnas (dmm):
trg_dmm2=tush_nl[de_miro3[,1]]
f_not_z2=function(trg_dmm2) {
  h=NULL
  for(i in 1:length(trg_dmm2))
    (if (length(trg_dmm2[[i]])!=0) h=c(h,i))
  return(trg_dmm2[h])}
trg_dmm2=f_not_z2(trg_dmm2)
# Changing the names of targets:
names(trg_dmm2)=f_mir_to_mature_vec(names(trg_dmm2))

# matching of low p-value matrix, and gen_rgl<0 matrix
# below zero value genes
ge_rgl_z=gen_rgl[,2]<0
ge_rgl_z=gen_rgl[ge_rgl_z,]
big=as.character(ge_rgl_z[,1])
# de_genes
small=as.character(de_geno[,1])
#matching
matches=big %in% small
gen_ok_f=big[matches]

# matching of low p-value matrix, and gen_rgl>0 matrix
# above zero value genes
ge_rgl_z=gen_rgl[,2]>0
ge_rgl_z=gen_rgl[ge_rgl_z,]
big=as.character(ge_rgl_z[,1])
# de_genes
small=as.character(de_geno[,1])
#matching
matches=big %in% small
gen_ok_fu=big[matches] # this is upregulated,

# Selecting upregulated/downregulated miRs (in general):
u=as.matrix(gen_rgl_mir[,2])>0
u_mir=(gen_rgl_mir[u,1])
d=as.matrix(gen_rgl_mir[,2])<0
d_mir=(gen_rgl_mir[d,1])
# Selecting upregulated mimats from my list
um=intersect(names(trg_dmm2),u_mir)
# Selecting downregulated mimats from my list
dm=intersect(names(trg_dmm2),d_mir)
## de_miRNA list values (trg_dmm2) must to be compared to gene names of the good p-values (de_geno)
hero2=f_match_down(de_geno, trg_dmm2) #using ready made function
```

```
  names(hero2)=names(trg_dmm2)
  # mirs up, genes down
  hero3=hero2[um] #ok
  t_mru_d=f_match_down2(hero3, gen_rgl)
  # mirs down
  hero4=hero2[dm] #ok
  t_mrd_u=f_match_down4(hero4, gen_rgl)

  # Mirs up, genes down Fish:
  fi=as.vector(matrix(0,nrow=length(t_mru_d)))
  for(i in 1:length(t_mru_d))
  {hr=fisher.test(matrix(c(
    t_mru_d[i],
    as.vector(sapply(hero3[i],length))-t_mru_d[i],
    length(gen_ok_f)-t_mru_d[i],
    dim(gen_rgl)[1]-(length(gen_ok_f)-t_mru_d[i])), nrow=2)); fi[i]=hr[[1]];}

  # MIRNA UP (=mu), GENE DOWN (=gd) => mugd
  mugd=cbind(names(hero3),fi)
  colnames(mugd)<-c('Mature miRNA','Fisher test p-value')
  mugd=mugd[order(as.numeric(mugd[,2])),]
  mugd[,2]=as.numeric(mugd[,2])
  mugd[,1]=f_mature_to_mir_vec(mugd[,1])

  # mirs down, genes up
  fit=as.vector(matrix(0,nrow=length(t_mrd_u)))
  for(i in 1:length(t_mrd_u))
  {hr=fisher.test(matrix(c(
    t_mrd_u[i],
    as.vector(sapply(hero4[i],length))-t_mrd_u[i],
    length(gen_ok_fu)-t_mrd_u[i],
    dim(gen_rgl)[1]-(length(gen_ok_fu)-t_mrd_u[i])), nrow=2)); fit[i]=hr[[1]];}

  # MIRNA DOWN (=md), GENE UP (=gd) => mdgu
  mdgu=cbind(names(hero4),fit)
  colnames(mdgu)<-c('Mature miRNA','Fisher test p-value')
  mdgu=mdgu[order(as.numeric(mdgu[,2])),]
  mdgu[,1]=f_mature_to_mir_vec(mdgu[,1])

  # p-values below 0.05, and their combined matrix (str(mdgu2): chr):
  #mir d gen up
  a=as.numeric(mdgu[,2])
  b=a<0.05
  c=mdgu[b,]

  #mir u gen d
  a=as.numeric(mugd[,2])
  b=a<0.05
  c1=mugd[b,]

  c1c=NULL
  c1c=rbind(c1,c)
  c1c=cbind(c(rep('d',dim(c)[1]), rep('u',dim(c1)[1])),c1c) #THIS IS not ok
  colnames(c1c)<-c('regul_miR','miRNA name','Fisher test p-value')
  write.table(c1c, "all_good_ver_alp.txt", sep="\t", row.names=F)
  return(c1c)
}
```

```
mir_reg=f_miR_enrich_fish(tush_nl,de_miro3,de_geno,gen_rgl) #ok
# Selecting the below 0.05 p-value miRs
trg_dmm4=trg_dmm3[mir_reg[,2]]

f_m_enr_fish2 = function (tush_nl,de_miro3,de_geno,gen_rgl,gen_rgl_mir) {
 #You need a list of miRNA targets (tush_nl), differentially expressed miRNAs (de_miro3),
 # differentially expressed mRNAs (de_geno), and finally log fold change values for genes (only I would say); gen_rgl

 # Finding targets (trg) of de_mat_mirnas (dmm):
 trg_dmm2=tush_nl[de_miro3[,1]]
 f_not_z2=function(trg_dmm2) {
  h=NULL
  for(i in 1:length(trg_dmm2))
   (if (length(trg_dmm2[[i]])!=0) h=c(h,i))
  return(trg_dmm2[h])}
 trg_dmm2=f_not_z2(trg_dmm2)
 # Changing the names of targets:
 names(trg_dmm2)=f_mir_to_mature_vec(names(trg_dmm2))

 # matching of low p-value matrix, and gen_rgl<0 matrix
 # below zero value genes
 ge_rgl_z=gen_rgl[,2]<0
 ge_rgl_z=gen_rgl[ge_rgl_z,]
 big=as.character(ge_rgl_z[,1])
 # de_genes
 small=as.character(de_geno[,1])
 #matching
 matches=big %in% small
 gen_ok_f=big[matches]

 # matching of low p-value matrix, and gen_rgl>0 matrix
 # above zero value genes
 ge_rgl_z=gen_rgl[,2]>0
 ge_rgl_z=gen_rgl[ge_rgl_z,]
 big=as.character(ge_rgl_z[,1])
 # de_genes
 small=as.character(de_geno[,1])
 #matching
 matches=big %in% small
 gen_ok_fu=big[matches] # this is upregulated,

 # Selecting upregulated/downregulated miRs (in general):
 u=as.matrix(gen_rgl_mir[,2])>0
 u_mir=(gen_rgl_mir[u,1])
 d=as.matrix(gen_rgl_mir[,2])<0
 d_mir=(gen_rgl_mir[d,1])
 # Selecting upregulated mimats from my list
 um=intersect(names(trg_dmm2),u_mir)
 # Selecting downregulated mimats from my list
 dm=intersect(names(trg_dmm2),d_mir)
 ## de_miRNA list values (trg_dmm2) must to be compared to gene names of the good p-values (de_geno)
 hero2=f_match_down(de_geno, trg_dmm2) #using ready made function
 names(hero2)=names(trg_dmm2)

 # mirs up, genes down
 hero3=hero2[um] #ok
 t_mru_d=f_match_down2(hero3, gen_rgl)
```

```
  # mirs down
  hero4=hero2[dm] #ok
  t_mrd_u=f_match_down4(hero4, gen_rgl)

  # Mirs up, genes down Fish:
  fi=as.vector(matrix(0,nrow=length(t_mru_d)))
  for(i in 1:length(t_mru_d))
  {hr=fisher.test(matrix(c(
    t_mru_d[i],
    as.vector(sapply(hero3[i],length))-t_mru_d[i],
    length(gen_ok_f)-t_mru_d[i],
    dim(gen_rgl)[1]-(length(gen_ok_f)-t_mru_d[i])), nrow=2)); fi[i]=hr[[1]];}

  # MIRNA UP (=mu), GENE DOWN (=gd) => mugd
  mugd=cbind(names(hero3),fi)
  colnames(mugd)<-c('Mature miRNA','Fisher test p-value')
  mugd=mugd[order(as.numeric(mugd[,2])),]
  mugd[,2]=as.numeric(mugd[,2])
  mugd[,1]=f_mature_to_mir_vec(mugd[,1])

  # mirs down, genes up
  fit=as.vector(matrix(0,nrow=length(t_mrd_u)))
  for(i in 1:length(t_mrd_u))
  {hr=fisher.test(matrix(c(
    t_mrd_u[i],
    as.vector(sapply(hero4[i],length))-t_mrd_u[i],
    length(gen_ok_fu)-t_mrd_u[i],
    dim(gen_rgl)[1]-(length(gen_ok_fu)-t_mrd_u[i])), nrow=2)); fit[i]=hr[[1]];}

  # MIRNA DOWN (=md), GENE UP (=gd) => mdgu
  mdgu=cbind(names(hero4),fit)
  colnames(mdgu)<-c('Mature miRNA','Fisher test p-value')
  mdgu=mdgu[order(as.numeric(mdgu[,2])),]
  mdgu[,1]=f_mature_to_mir_vec(mdgu[,1])

  # p-values below 0.05, and their combined matrix (str(mdgu2): chr):
  # and BH correction
  ao=as.numeric(mdgu[,2])
  ao=cbind(rep('d',length(ao)), mdgu)

  ap=as.numeric(mugd[,2])
  ap=cbind(rep('u',length(ap)), mugd)

  at=rbind(ao,ap)
  colnames(at)<-c('regul_miR','miRNA name','Fisher test p-value')

  #BH correction:
  c_enmir=p.adjust(at[,3], method="BH")
  at[,3]=c_enmir
  se=as.numeric(at[,3])<0.05
  at=at[se,]

  write.table(at, "enr_mir_BH_cor.txt", sep="\t", row.names=F)
  return(at)
}
```

```
#Values of confusion matrix extracted
f_enr_mir_cmat= function (t_mru_d,hero3,gen_ok_f,gen_rgl,c1) {
  t=vector("list",length(t_mru_d))
  names(t)=f_mature_to_mir_vec(names(hero3))
  a=NULL
  o=NULL
for(i in 1:length(t_mru_d)) (
  t[[i]] =list(t_mru_d[i], as.vector(sapply(hero3[i],length))-t_mru_d[i],
            length(gen_ok_f)-t_mru_d[i],dim(gen_rgl)[1]-(length(gen_ok_f)-t_mru_d[i])) )
a=do.call(rbind,t)
o=match(as.vector(as.vector(c1[,1])),rownames(a))
a=a[o,]
colnames(a)=c("A","B","C","D")
write.table(a,"enr_mir_rgl-uod.txt", row.names=F, sep="\t")
return(a)}
tush_new_miru_cm=f_enr_mir_cmat(t_mru_d,hero3,gen_ok_f,gen_rgl,c1) #ok

#Enhanced version from the previous to get all calculated information at once:
f_enr_mir_cmate= function (t_mru_d,t_mrd_u,hero3,hero4, gen_ok_f,gen_ok_fu,gen_rgl,c1c) {
  t=vector("list",length(t_mru_d))
  names(t)=f_mature_to_mir_vec(names(hero3))
  ti=vector("list",length(t_mrd_u))
  names(ti)=f_mature_to_mir_vec(names(hero4))
  a=NULL
  b=NULL
  c=NULL
  o=NULL
  oi=NULL
for(i in 1:length(t_mru_d)) (
  t[[i]] =list(t_mru_d[i], as.vector(sapply(hero3[i],length))-t_mru_d[i],
            length(gen_ok_f)-t_mru_d[i],dim(gen_rgl)[1]-(length(gen_ok_f)-t_mru_d[i])) )
  for(i in 1:length(t_mrd_u)) (
  ti[[i]] =list(t_mrd_u[i], as.vector(sapply(hero4[i],length))-t_mrd_u[i],
            length(gen_ok_fu)-t_mrd_u[i],dim(gen_rgl)[1]-(length(gen_ok_fu)-t_mrd_u[i]))
  )
a=do.call(rbind,t)
b=do.call(rbind,ti)
o=match(as.vector(as.vector(c1c[,2])),rownames(a))
oi=match(as.vector(as.vector(c1c[,2])),rownames(b))
a=a[o,]
a=a[is.na(rownames(a))!=T,]
b=b[oi,]
b=b[is.na(rownames(b))!=T,]
c=rbind(a,b)
c=as.matrix(cbind(as.data.frame(c1c[,3]),c))
colnames(c)=c("P-value","A","B","C","D")

write.table(c,"enr_mir_rgl-uode.txt", row.names=F, sep="\t")
return(c)}

## Clustering for enriched miRs
d <- dist(tsem, method = "euclidean") # distance matrix
fot <- hclust(d, method="average")
par(ps=15)
plot(fot, hang = -1) # display dendogram
pv2 =   pvclust(t(tsem),method.hclust=method,method.dist=distance,nboot = 500) #notice: nbooot=500
```

```
pvrect(pv2, alpha = 0.95 ,pv="au", type="geq", max.only=FALSE, border=4) #See the borders and edit in an image program

##Important function for getting the targets of miR-list, used below;
f_miR_targs= function (list_mir) {
  aus=vector("list", length=length(tush_nl))
  names(aus)=names(tush_nl)
  cond=c(1:length(tush_nl))[is.na(match(names(tush_nl),list_mir)==T)==F]
  aus=tush_nl[cond]
  return(aus)
}

# A function to get the names of up/down regulated genes (in a group list) for enriched miRNA (down/up=d/u)
f_trg_ud_em_dg = function (trg_dmm4,group2) {
gr1=f_gsymb_ent(names(group2)) #length(group3)
ug1=gen_u[gen_u[,1] %in% gr1,] #dim(ug1)
# corresponding down enr_mirs:
ug1m=f_mult_g_mir(ug1[,1]) #str(ug1m)
ug1m_enr=f_enr_mir_in_lst(trg_dmm4,ug1m)
reg_g1_mir=unique(unlist(ug1m_enr))
mR_enr_d1=mR_enr_d[(mR_enr_d[,1] %in% reg_g1_mir),]
#The function is used to find the targets of these enr_mirs
mR_enr_d1_targ=f_miR_targs(mR_enr_d1[,1])
d1_trg_un=unique(unlist(mR_enr_d1_targ))
#what targets are the same as groups dysregulated genes:
g1_ug_denrm=d1_trg_un[d1_trg_un %in% ug1[,1]]
#g1_ug_denrm_ks=f_ent_gsymb(g1_ug_denrm) #Gene names, length(g1_ug_denrm_ks)
write.table(g1_ug_denrm, "sel_ug_denrm_ks.txt",sep="\t",row.names=F, col.names=F)

# downregulated names(group1),
dg1=gen_d[unique(gen_d[,1]) %in% gr1,] #dim(dg1)
# corresponding up enr_mirs:
ug1md=f_mult_g_mir(dg1[,1]) #str(ug1m)
ug1md_enr=f_enr_mir_in_lst(trg_dmm4,ug1md)
reg_g1_mird=unique(unlist(ug1md_enr))
mR_enr_u1=mR_enr_u[(mR_enr_u[,1] %in% reg_g1_mird),]
#The function is used to find the targets of these enr_mirs
mR_enr_u1_targ=f_miR_targs(mR_enr_u1[,1])
u1_trg_un=unique(unlist(mR_enr_u1_targ))
#what targets are the same as groups dysregulated genes:
g1_dg_uenrm=u1_trg_un[u1_trg_un %in% dg1[,1]]
#g1_dg_uenrm_ks=f_ent_gsymb(g1_dg_uenrm) #Gene names, length(g1_dg_uenrm_ks)
write.table(g1_dg_uenrm, "sel_dg_uenrm_ks.txt",sep="\t",row.names=F, col.names=F)

return(list(g1_ug_denrm_ks,g1_dg_uenrm_ks)) }

tg1=f_trg_ud_em_dg(trg_dmm4,group1)

# for normal up/down (when normal mirs up/down)
f_Trg_ud_M_dg = function (trg_dmm4,de_geno2) {
  gr1=de_geno2[,1] #length(group3)
  ug1=gen_u[gen_u[,1] %in% gr1,] #dim(ug1)
  # corresponding down mirs:
  ug1m=f_mult_g_mir(as.character(ug1[,1])) #str(ug1m)
  f_not_z2(ug1m)
  #ug1m_enr=f_enr_mir_in_lst(trg_dmm4,ug1m)
  reg_g1_mir=unique(unlist(ug1m))
```

```
  mR_d1=miR_d[(miR_d[,1] %in% reg_g1_mir),]
  #The function is used to find the targets of these mirs
  mR_d1_targ=f_miR_targs(mR_d1[,1])
  d1_trg_un=unique(unlist(mR_d1_targ))
  #what targets are the same as groups dysregulated genes:
  g1_ug_denrm=d1_trg_un[d1_trg_un %in% ug1[,1]]
  g1_ug_denrm_ks=f_ent_gsymb(g1_ug_denrm) #Gene names, length(g1_ug_denrm_ks)
  write.table(g1_ug_denrm_ks, "sel_ug_dm_ks.txt",sep="\t",row.names=F, col.names=F)

  # downregulated names(group1),
  dg1=gen_d[unique(gen_d[,1]) %in% gr1,] #dim(dg1)
  # corresponding up mirs:
  ug1md=f_mult_g_mir(dg1[,1]) #str(ug1m)
  #ug1md_enr=f_enr_mir_in_lst(trg_dmm4,ug1md)
  reg_g1_mird=unique(unlist(ug1md))
  mR_u1=miR_u[(miR_u[,1] %in% reg_g1_mird),]
  #The function is used to find the targets of these enr_mirs
  mR_enr_u1_targ=f_miR_targs(mR_u1[,1])
  u1_trg_un=unique(unlist(mR_enr_u1_targ))
  #what targets are the same as groups dysregulated genes:
  g1_dg_uenrm=u1_trg_un[u1_trg_un %in% dg1[,1]]
  g1_dg_uenrm_ks=f_ent_gsymb(g1_dg_uenrm) #Gene names, length(g1_dg_uenrm_ks)
  write.table(g1_dg_uenrm_ks, "sel_dg_um_ks.txt",sep="\t",row.names=F, col.names=F)

  return(list(g1_ug_denrm_ks,g1_dg_uenrm_ks)) }
de_gt_udnm=f_Trg_ud_M_dg(trg_dmm4,de_geno2) #This takes time if de_geno2 is long (de genes)

#Obtain enrichment information from internet with previously defined lists of genes
path_to_files =
"C:/Users/Pauli/Documents/thesis/labour/analysis/TCGA/results/COLLECTED_OKS/enrichment/247_go_mf_bp_mirs
_genecodis/"
# For e.g. MFs
files = list.files(path_to_files, pattern="mf_genecodis",all=F)
enr_gomf_cc <- file.path(path_to_files, files)
enr_gomf_cc_ext <- lapply(enr_gomf_cc, sep="\t", quote = "", row.names = NULL,stringsAsFactors = FALSE,read.csv)
#japadapa duuu!!
write.table(enr_gomf_cc_ext, "247_gomf_cc.txt", sep="\t",row.names=F)

##Kegg pwys for up_genes (d_mirs, normal)
path_to_files = "C:/Users/Pauli/Documents/thesis/labour/analysis/TCGA/results/COLLECTED_OKS/enrichment &
clusterings/"
# For e.g. MFs
files = list.files(path_to_files, pattern="_ks",all=F)
sel_gm <- file.path(path_to_files, files)
sel_gmj <- lapply(sel_gm, sep="\t", quote = "", row.names = NULL,stringsAsFactors = FALSE,read.csv)
sel_ug_dm_ks=do.call(rbind,sel_gmj[2])
sud=sel_ug_dm_ks[,1]
write.table(sud, "sud.txt",sep="\t",row.names=F)
sud=read.table(file="sud.txt")
sel_ug_dm_ks=as.matrix(f_gsymb_ent(sud[,1]))
my_pwys_ud=pwy_func5(pathway.list2, sel_ug_dm_ks, e11_l2) #no result is ok result

#Get certain information from a list quickly to same source (path.files):
# 247_U_gen_D_enrmirs BP
path_to_files = "C:/Users/Pauli/Documents/thesis/labour/analysis/TCGA/results/COLLECTED_OKS/enrichment &
clusterings/Genecodis/List_247_min_2_enrmir/Enr mirs/247_U_gen_D_enrmirs/"
# For e.g. MFs
```

```
files = list.files(path_to_files, pattern="bp",all=F)

#function to get info from genecodis files: bp,mf,kegg
f_prs_1_fil= function(path_to_files,files) {

udem_bp <- file.path(path_to_files, files)
udem_bp_l  <- lapply(udem_bp, sep="\t", quote = "", row.names = NULL,stringsAsFactors = FALSE,read.csv)
#dim(udem_bp_l[[1]])
cond2=udem_bp_l[[1]][,9]<0.01 & !is.na(udem_bp_l[[1]][,9]<0.05)
udem_bp_l2=udem_bp_l[[1]][cond2,]
udem_bp_l3=udem_bp_l2[order(udem_bp_l2[,9]),]
#dim(udem_bp_l3)
t_fap=function(a,b,c,d) ((a/b)/(c/d))
v1=NULL
for (i in 1:dim(udem_bp_l3)[1]) (
v1[[i]]=t_fap(udem_bp_l2[i,c(4)],udem_bp_l2[i,c(5)],udem_bp_l2[i,c(6)],udem_bp_l2[i,c(7)]) )
udem_bp_l4=cbind(udem_bp_l3,v1)
udem_bp_l4=udem_bp_l4[,c(2,3,9,4,11)]
colnames(udem_bp_l4)=c("GO:ID","GO:Term","Q-value","Sig.genes","Odds ratio")
x <- udem_bp_l4[,2]
y <- gsub(" (BP)","",x,fixed=T)
udem_bp_l4[,2] = y
udem_bp_l4[,3]=round(udem_bp_l4[,3],digits=4)
udem_bp_l4[,5]=round(udem_bp_l4[,5],digits=1)
setwd(path_to_files)
write.table(udem_bp_l4, "gc_CHECK.txt", sep="\t",row.names=F)
setwd("C:/Users/Pauli/Documents/thesis/labour/analysis/TCGA/results")
return(udem_bp_l4)
}

#function to get infro from genecodis files: bp,mf,kegg
t_prs_1_fil= f_prs_1_fil(path_to_files,files) #ok

### Step 15 ###
# Linear modelling of miRNAs

### Step 15a ###
# Enhancing the data before the linear modelling scheme:
# miR normalisation
testia=justvsn(as.matrix(dtt2c)) #dtt2c=dtt2b

# miR z-scoring
data_zs=zNorm(testia)
# Check the distribution with the box_plot
boxplot(log2(data_zs[,100:125])) # about ok

# exp log-2 (Vijay also did log-2)
e11_l2=log2(e11+1)

# e11_l2 z-scoring
e11_zs=zNorm(e11_l2)
# Check the distribution with the box_plot
boxplot(log2(e11_zs[,1:35])) #ok

# A function for finding the corresponding gene symbol to Entrez name
f_ent_gsymb = function (list) {
mm=match( as.matrix(names(list)),as.character(gsymb2[,1]))
```

```
return (gsymb2[mm,2])}

### Step 15b ###
# Defining the lists and mirs for linear modelling

# list 1:
list11a=unlist(trg_dmm4) #all genes: length(list11a): [1] 8868
list11=unique(unlist(trg_dmm4)) #unique genes: length(list11): [1] ~4896
# To obtain all the miRNAs that the target gene (x) of interest (from tush_nl list) has:
f_mir_limoo = function(tush_nl, x) {
  x=as.character(x)
  tmd=NULL #note that tush_nl is a list
  #Matching with %in% command
  for (i in 1:length(tush_nl)) {
    if (sum ( (as.character(unlist(tush_nl[i])) %in% x)==T ) > 0 )
      tmd[i]=names(tush_nl)[i] }
  return(tmd[!is.na(tmd)]) }

# Finding miRNAs for these genes
f_mult_g_mir= function(list) {
  #list is a gene list
  gene_nam=vector(mode="list", length=length(list))
  mirs=vector(mode="list", length=length(list))
  for (i in 1:length(list)) {
    gene_nam[[i]] = as.vector((list)[i])
    mirs[[i]] = f_mir_limoo(tush_nl, gene_nam[[i]]) }
  return(mirs) } #or, using matrix
f_mult_g_mir2= function(matrix) {
  #list is a gene list
  gene_nam=vector(mode="list", dim(matrix)[1])
  mirs=vector(mode="list", dim(matrix)[1])
  for (i in 1:dim(matrix)[1]) {
    gene_nam[[i]] = matrix[i,1]
    mirs[[i]] = f_mir_limoo(tush_nl, gene_nam[[i]]) }
  return(mirs) }
mirs_list1=f_mult_g_mir(list11)

#find the number of enriched targets in miRNA list
f_enr_mir_in_lst=function(trg_dmm4,mirs_list3r) {
  x=NULL
  for (i in 1:length(mirs_list3r)) (
    x[[i]]=mirs_list3r[[i]][unlist(mirs_list3r[[i]]) %in% unlist(names(trg_dmm4))]
    )
  return(x)
}
en_mir_deva2t=f_enr_mir_in_lst(trg_dmm4,mirs_list3r)

# list2:
int_gen23=read.table("interesting_genes_zdk_pt_4214.txt",sep="\t")
int_gen23=unique(int_gen23[,1])

# Then you should check:
# All the (miRNA target) gene names that you have in your list of interest /two, "stp3"/:
f_mir_limo2 = function(targets_ok3, stp3) {
  tmd=NULL # stp must be in vector have same names as the genes (so double)
  tmd=stp3[stp3 %in% as.character(unlist(targets_ok3))]
  return(tmd) }
```

```
#matching gene symbols to my Entrez list (the list have slightly less values than in id_converter database) (I should use some other method then)
  f_gsymb_ent= function (list) {
  mm=match( as.matrix((list)),as.character(gsymb2[,2]))
  return (gsymb2[mm,1])} #notice that some gene symbols do not have EntrezNames, check also: http://idconverter.bioinfo.cnio.es/

int_gen23=f_gsymb_ent(int_gen23) # some ent symbols where not available, and they where collected from internet inividually:
ig23l=c("7490","383","5009","4842","2271","189","5563","2645","5163","5313","5106")
### Replace NA values with other values:
y <- which(is.na(int_gen23)==TRUE) # This is how you get the indexes
int_gen23[y]=ig23l
mirs_list3=f_mult_g_mir(int_gen23)

#It seems that I need to check from data_zs_t that I got all the miRs also
f_c_ok_mirs=function (data_zs_t,lists_s_mirs) {
  mirs_ig=NULL
  for (i in 1:length(lists_s_mirs)) (
    mirs_ig[[i]]=lists_s_mirs[[i]][which (lists_s_mirs[[i]] %in% rownames(data_zs_t))]  )
  return(mirs_ig)
}
mirs_list3r=f_c_ok_mirs(data_zs_t,lists_s_mirs)

# Check how many  miRs you have in your miRNa list:
f_enr_mir_nro=function(mirs_list3r) {
  x=NULL
  for (i in 1:length(mirs_list3r)) (
    x[[i]]=length(mirs_list3r[[i]]) )
  return(x)
}
mirs_list3r_nro=f_enr_mir_nro(mirs_list3r)

#Removing nas
f_not_z2=function(list_mirs) {
  h=NULL
  for (i in 1:length(list_mirs)) (
    if ( length(list_mirs[[i]]) != 0) h=c(h,i) )
  return(list_mirs[h])
}
mirs_list3r=f_not_z2(mirs_list3r_nro)

#Adding and parsing some elements to a list... (see tikka_complete_extras.r):
oks4=rownames(e11_zs)[rownames(e11_zs) %in% as.vector(int_gen23y)]
# Then the previous ordering might be done as:
mir_oks4=vector("list", length(oks4))
mir_oks4=f_mult_g_mir(oks4)
mir_oks4=f_c_ok_mirs(data_zs_t,mir_oks4)
names(mir_oks4)=names(oks4)
mir_oks4=f_not_z2(mir_oks4)
#ordering of the list
pito=f_enr_mir_nro(mir_oks4)
mir_oks4_ord = (mir_oks4[rev(order(pito))])
list_2m=cbind(names(mir_oks4_ord), pito)
#write.table(list_2m,"list2m.txt",row.names=F,col.names=F,sep="\t")
```

```
#list 3:
#Finding the gene list from overlapping miRNA's information:
f_overlap2 = function (x) {
  ai=unique(unlist(x)) #4321
  overlap_mat2 = matrix(nrow=length(x),ncol=length(ai))
  overlap_mat2 = "[<-"(overlap_mat2,value=0) # all values are zero
  rownames(overlap_mat2)=names(x)
  colnames(overlap_mat2)=ai
  for (j in 1:length(x)) (
    for (k in 1:length(x[[j]])) (
      for (i in 1:dim(overlap_mat2)[2]) (
        if (colnames(overlap_mat2)[i]==as.character(x[[j]][k]))
          (overlap_mat2[j,i]=1) )))
return(overlap_mat2)}
overlap_mat3=f_overlap2(trg_dmm3)
# Or if you have done this matrix in another computer then you may download it:
overlap_mat3=read.table("overlap_mat.txt", sep="\t", col.names=T)

# One does need to limit the amount of genes so for this a list of most overlapping genes is done:
over_sum=apply(overlap_mat2,2,sum)
over_sum_ok=over_sum > 1
over_lap=over_sum[over_sum_ok]
over_ok=as.matrix(over_lap[rev(order(over_lap))])
## are all the names in the over_ok at e11_zs?
dim(e11_zs[(rownames(e11_zs) %in% rownames(over_ok)),]) # vs. dim(over_ok)
## aha! they do not match! So
over_ok_mod = over_ok[(rownames(over_ok) %in% rownames(e11_zs)),1]
list=over_ok_mod[1:150]

# This matrix only for differentially expressed genes:
f_overlap3 = function (de_geno2, trg_dmm4) {
  ai=unique(unlist(de_geno2[,1])) #1963
  overlap_mati2 = matrix(nrow=length(trg_dmm4),ncol=length(ai))
  overlap_mati2 = "[<-"(overlap_mati2,value=0) # all values are zero dim(overlap_mati2)
  rownames(overlap_mati2)=names(trg_dmm4)
  colnames(overlap_mati2)=ai
  for (j in 1:length(trg_dmm4)) (
    for (k in 1:length(trg_dmm4[[j]])) (
      for (i in 1:dim(overlap_mati2)[2]) (
        if (colnames(overlap_mati2)[i]==as.character(trg_dmm4[[j]][k]))
          (overlap_mati2[j,i]=1) )))
  return(overlap_mati2)}
overlap_mati3=f_overlap3(de_geno2, trg_dmm4)

#Renaming the list (notice that list name should be "3" and not list22...
list22=list
names(list22)=names(ibtaai)
list22=as.matrix(list22)
write.table(list22,"list22.txt",sep="\t", col.names=F) #ok

Combining the lists:
#1: list_11ii
#2: list2ok_nam
#3: list

#For list one (list_11ii) and three, Selecting only the ones that are not in list three
m_l31=match(names(list),rownames(list_11ii))
```

```
l1_red=list_11ii[-m_l31,]
lir1=rownames(l1_red[1:50,])
lir11=rownames(l1_red[1:73,])

#For list 2 (list2ok_nam) and three
m_l21=match(names(list2ok_nam),names(list)) #voitaneen lisätä tällaisenaan
# checking: table(names(list) %in% as.character(list2ok_nam)) #, ok

#All 3 lists (except list nr1 as such in order: 1,2,3)
list_s_tot=c(lir1,as.character(list2ok_nam),names(list))
lists_s_tot=c(lir11,as.character(list2ok_nam),names(list)) #Better to have slightly bigger list and round number in total 250
#you need names for your list for the function
names(lists_s_tot)=lists_s_tot
#Check that you have every gene:
table(rownames(e11_zs_t) %in% lists_s_tot)
match(lists_s_tot,rownames(e11_zs_t))

#Checking again
lists_s_mirs=f_mult_g_mir2(lists_s_tot)
names(lists_s_mirs)=lists_s_tot

#This is what you need:
lmt_m_ok=f_c_ok_mirs(data_zs_t,lists_s_mirs)
lmt_m_nro=f_enr_mir_nro(lmt_m_ok)
#There must be at least one miRNA in the model (I corrected the f_limo_restr2 accordingly)
rm_lt=c(1:250)[is.na(match(lmt_m_nro,0)==T)==F]
lists_s_tot=lists_s_tot[-rm_lt] #Removing zero miRNA value genes from list
lir11_add=rownames(l1_red[c(75:76),]) # adding nonzero miRNA genes
lists_s_tot=c(lists_s_tot,lir11_add) #combining the reduced list and the newly added genes
names(lists_s_tot)[249:250]=lir11_add #naming all the entries

#So finally you may use your functions (and there should not be zeros any more):
lmt_m_ok=f_c_ok_mirs(data_zs_t,lists_s_mirs)
lmt_m_nro=f_enr_mir_nro(lmt_m_ok) #Alternative way: lists_mir_lengths=sapply(lmt_m_ok,length), and median(lists_mir_lengths)

# The big driving in HKI computers, first combining list 1-3 (as such):
list_big=c(rownames(l1_red),as.character(list2ok_nam),names(list))

### Step 15c ###
# Clustering and Enrichment of one of the lists (list3, previously: (just top 150) list):
# constructing the function for hammington distance matrix
f_ham_dis= function (ibt2) {
  #ibt is the gene (and miR) information in list of lists
  abcd <- vector("list", length(ibt2))
  abcd=NULL
  for (i in 1:length(ibt2)) {
    abcd[[i]]=c(rownames(ibt2[[(i)]][[2]])) }
  abcd=unlist(abcd)
  abcd=as.matrix(unique(abcd))
  abcd=abcd[(abcd[,1]!="beta_0"),1] # jee

  #the ham matrix constructed
  gen_mir_ham=matrix(0,nrow=length(ibt2),ncol=length(abcd))
  rownames(gen_mir_ham)=names(ibt2)
  colnames(gen_mir_ham)=abcd
```

```
for (i in 1:length(ibt2)) (
for (j in 1:(length(ibt2[[i]][[2]]))) (
if (ibt2[[i]][[2]][j,1]>0) (gen_mir_ham[i,(abcd %in%
as.matrix(rownames(ibt2[[i]][[2]]))[-match("beta_0",rownames(ibt2[[i]][[2]])),1][j])]=1) ) )
return(gen_mir_ham) }

# The distanced maybe thus calculated:
f_calc_dist = function (gen_mir_ham2) {
n <- nrow(gen_mir_ham2)
m <- matrix(nrow=n, ncol=n)
rownames(m)=rownames(gen_mir_ham2)
colnames(m)=rownames(gen_mir_ham2)
for(i in seq_len(n - 1))
  for(j in seq(i, n))
    m[j, i] <- m[i, j] <- sum(gen_mir_ham2[i,] != gen_mir_ham2[j,])
return(m)
}

# Clusterings of Hammington distances of top 150 miRNA overlapping genes correlations
pdf('clust_all.pdf')
method="complete"
distance="euclidean"
test_clust=hclust(dist(mo, method=distance),method=method)
par(ps=3)
plot(test_clust,par) # you may also save as such in the Rstudio as pdf
dev.off()

# Information for cluster groupings:
# Ward Hierarchical Clustering was found to be good one for clustering:
d <- dist(moi2, method = "euclidean") # distance matrix
rownames(moi2)
fit <- hclust(d, method="ward")
par(ps=3)
plot(fit) # display dendogram
groups <- cutree(fit, k=9) # cut tree into 9 clusters
# draw dendogram with red borders around the 9 clusters
rect.hclust(fit, k=9, border="red")
#Group selection from the image is a delicate process:
group1=groups[(groups==1)] # ok
(groups)["BACH2"] ##in group 4
group2=groups[(groups==4)]
(groups)["KLF12"] #in group 2
(groups)["CELF2"] # in group 6
gr3= (groups==2) | (groups==6) # combining two groups (was needed according to image)
group3=groups[gr3]
(groups)["TRPS1"] #in group 5
group4=groups[(groups==5)]
(groups)["IKZF4"] #in group 3
group5=groups[(groups==3)]
(groups)["SLC1A2"] #in group 8
group6=groups[(groups==8)]
(groups)["PHACTR2"] #in group 9
group7=groups[(groups==9)]
(groups)["RC3H1"] #in group 9
group8=groups[(groups==7)]
```

```
#Collecting information of total list (here list 247):
path_to_files =
"C:/Users/Pauli/Documents/thesis/labour/analysis/TCGA/results/COLLECTED_OKS/enrichment/247_kegg_pwys_gen
ecodis/extra/"
# Fore every miR
files = list.files(path_to_files, pattern="_miRs",all=F)
enr_k_cc <- file.path(path_to_files, files)
enr_k_cc_ext <- lapply(enr_k_cc, sep="\t", quote = "", row.names = NULL,stringsAsFactors = FALSE,read.csv)
#japadapa duuu!!
write.table(enr_k_cc_ext, "247_kegg_cc.txt", sep="\t",row.names=F)

mir_info2[[1]][1:4, 1:7]
a=sub(".txt", "", files, perl=TRUE) #jee
names(mir_inf)=a
names(mir_inf2)=a

#Better to have 8 tables where extract information:
grp_tot_nam2=list(names(group1),
names(group2),names(group3),names(group4),names(group5),names(group6),names(group7),names(group8))
names(grp_tot_nam2)=c("group1", "group2","group3", "group4","group5","group6","group7","group8")
lapply(names(grp_tot_nam2),function(x, grp_tot_nam2)
write.table(grp_tot_nam2[[x]], paste(x, ".txt", sep = ""),col.names=FALSE, row.names=FALSE, sep="\t",
quote=FALSE),grp_tot_nam2)

#It is maybe better to do this 8 times for every group the same info:
path_to_files = "C:/Users/Pauli/Documents/thesis/labour/analysis/TCGA/results/cluster_150_enr/"
g_inf=vector("list",8)
files=vector("list",8)
gi=vector("list",8)
pattern=vector("list",8)

# Somehow one loop solution did not work:
for (i in 1:8) (
  pattern[[i]]=paste("_g",i, sep="") )
for (i in 1:8) (
files[[i]] = list.files(path_to_files, pattern[[i]],all=F) )
for (i in 1:8) (
gi[[i]] <- file.path(path_to_files, files[[i]]) )
for (i in 1:8) (
g_inf[[i]] <- lapply(gi[[i]], sep="\t", quote = "", row.names = NULL,stringsAsFactors = FALSE,read.csv) )
names(g_inf)=c("group1", "group2","group3", "group4","group5","group6","group7","group8")

#Enrichment information for these groups from: http://amp.pharm.mssm.edu/Enrichr/index.html
#Selecting certain things for observations:
# Find out which list factor is which:
g_inf[[1]][[1]][1:4, 1:7] ## cancer cell en
g_inf[[1]][[2]][1:4, 1:7] ## go_bp
g_inf[[1]][[3]][1:4, 1:7] ## go_mol_func
g_inf[[1]][[4]][1:4, 1:7] ## kegg
g_inf[[1]][[5]][1:4, 1:7] ## mir

#Now changing the last module you may see all of the values that you want:
f_wanted_gr= function (g_inf) {
abc=NULL
for (i in 1:8)
  (abc[[i]]=g_inf[[i]][[4]][1:5, c(1,2,4)])
   return(abc) }
```

```
#Defining wanted:
sel_grp=f_wanted_gr(g_inf)
names(sel_grp)=names(g_inf)

#Clustering of samples according to the miRNAs:
#First some filtering: (#testia2 is justvsn normalized data, after filtering of zero expressed miRNAs: dim(testia2) 332
567)
# Variance:
mean_mir_expr=apply(testia2,1,mean)
quantile(mean_mir_expr)
m_val=mean_mir_expr[8.270042<=mean_mir_expr] #length(m_val)
m_val_expr=testia2[names(m_val),] # dim(m_val_expr): 83 567

#Next hierachichal clustering of miRNAs/mRNAs
# miRs SAMPLES!!:
install.packages('sparcl')
library(sparcl)
dim(m_val_expr) #83miRNAs
d <- dist(t(m_val_expr), method = "euclidean") # distance matrix
fit <- hclust(d, method="ward")
method = "ward"
distance = "euclidean"
pv2 =   pvclust((m_val_expr),method.hclust=method,method.dist=distance,nboot = 10)

# e_n
yen=c(1:dim(e11_zs)[2])[is.na(match(colnames(e11_zs),e_n)==T)==F]
yen1=c(1:dim(e11_zs)[2])[yen]=1
# e_t
load("e_t.rdata")
yet=c(1:dim(e11_zs)[2])[is.na(match(colnames(e11_zs),e_t)==T)==F]

# all -e_n - e_t
yetn=c(yen,yet)
ym=c(1:dim(e11_zs)[2])[is.na(match(c(1:dim(e11_zs)[2]),yetn)==T)==F]
yea=c(1:dim(e11_zs)[2])[-ym]
y=c(yea,yet,yen) #length(y)
# Perform hierarchical clustering
hc <- hclust(dist(x),method="complete")

# Plot
method = "average" #?
distance = "euclidean"
pv2 =   pvclust((m_val_expr),method.hclust=method,method.dist=distance,nboot = 10)
par(ps=22)
y2 = matrix(ncol=1,nrow=567)
y2 = "[<-"(y2,value=0) #
y2[yen,]=1
y2[yet,]=2
y2[yea,]=3
ColorDendrogram(fit,y=y2,main="",branchlength=150)#,label=F)
pvrect(pv2, alpha = 0.99,pv="au", type="geq", max.only=FALSE, border=8)#, col="WHITE")

#now for genes:
mean_gen_expr=apply(e11_l2,1,mean)
quantile(mean_gen_expr)
ge_val=mean_gen_expr[9.9610451<=mean_gen_expr] #length(ge_val)
ge_val_expr=e11_l2[names(ge_val),] # dim(ge_val_expr), too high: [1] 4115  567
```

```
#Also variance:
GeneVar=apply(ge_val_expr,1,var)
GVM=mean(GeneVar)
ge_val_expr2 <- ge_val_expr [ - which (GeneVar <= 0.8) , ] #dim(ge_val_expr2), 710->ok,

#the rest similarly

### Step 15d ###
# Linear modelling

# Formulating the problem for one (or more genes)
# Samples (i.e. error values):  i, # miRNAs: m
# RHS (right-hand-side, ie. equations, or rows) : n (= i x 2)
# LHS (left-hand-side, i.e. columns): i + m + 1

#First, finding restriction value for miRs
quantile(lmt_m_nro,probs = seq(0, 1, 0.05)) #Ok, let the miRNA in this case be 20% quantile
plot(quantile(lmt_m_nro,probs = seq(0, 1, 0.05)))
# According to mir_amount decide the TimeLimit:
mir_amount=22 # 20% quantile
#Estimate the mean time needed to calculate values between 20-90% quantile, by running the basic function once with miRNA value above miR_amount:
TL=333 #TL= TimeLimit
TL=333 #TL= TimeLimit


# This is the enhanced function for making linear modelling using also a restriction analysis (and training groups if needed):
f_limo_restr2 = function (e11_zs_t, e11_zs_t2, data_zs_t,  data_zs_t2, gene_nam, mirs, mir_amount, TL) {
  mae=max(e11_zs_t)
  mid=min(abs(data_zs_t))
  mult=ifelse(mae/mid>300,300,mae/mid)
  e_lim=ifelse(round(max(e11_zs_t)-mult*(min(data_zs_t)))>25,25,round(max(e11_zs_t)-mult*(min(data_zs_t))))

  mir_b_max = mult
  #mir_amount = 153 #ok, with correction of data_zs finding right miRs cor~-0.1124, and list1,1/1000/227: -0.047, now without bounds: 0.2058, but it uses only one miR?
  #ok, now the correlation is 0.2 (after just bounding miRs with equations, but so many zeros...), with 17 mirs: no correlation. The earlier best celf2: cor:na

  #miRNA list (checking what I can find)
  ee=length(rownames(data_zs_t)[rownames(data_zs_t) %in% as.vector(unlist(mirs))])
  mir_val = if (ee>1) (t(data_zs_t[(rownames(data_zs_t) %in% as.vector(unlist(mirs))),])) else (matrix(t(data_zs_t[(rownames(data_zs_t) %in% as.vector(unlist(mirs))),])))

  colnames(mir_val) = if (dim(mir_val)[2]<=1) rownames(data_zs)[c(1:length(rownames(data_zs)))[is.na(match(rownames(data_zs),unlist(mirs))==T)==F]] else colnames(mir_val)

 #dim(mir_val)
  #colnames(mir_val)
  #mir_val[1:5,]

  #Maximum (or minimum) miRNA expression:
  m=mir_b_max
  #The amount of miRNA you want to use:
  w=mir_amount
```

```
w=ifelse(w>dim(mir_val)[2], dim(mir_val)[2],w)

nro1  = dim(mir_val)[1]
nro2  = dim(mir_val)[2]
nro3  = (nro2+1)

#rows:
#rhs   = (nro1*2+nro2*4+1)
rhs   = (nro1*2+nro2*2+1)
#columns:
lhs   = (1+nro2+nro1+nro2)

#The (row size ) limit of  the "normal matrix" (i.e. after that comes beta restrion constrain equations area):

#oe=rhs-nro2*4-1
oe=rhs-nro2*2-1
# The (column size ) limit of  the "normal matrix"
# (after that the binary variables):
ae=lhs-nro2

# The constraint matrix should then look like:
A_mat_f = matrix(0,nrow=rhs, ncol=lhs, byrow=T)
#dim(A_mat_f)

#Constraints for betas (part 2a) matrix. Building column by column
#zero (column) for constants
A_mat_cont1 = matrix(0,nrow=2*nro2, ncol=1, byrow=T)
# values for miRNAs
# Constraints for betas (part 2b)
be=rbind(-m,m)
Lt=c()
for(i in 1:(nro2))(Lt[i]=list(be)) # The "be" needs to be in a list
b2=data.matrix(bdiag(Lt))
#dim(b2)
#b2[1:,1:4]

A_mat_cont2 = matrix(0,nrow=2*nro2, ncol=nro2, byrow=T)
# I need looping:
for (j in 1:dim(A_mat_cont2)[2]) (
  A_mat_cont2[(b2[,j]!=F),j]=rbind(1,1) )
#dim(A_mat_cont2)
#A_mat_cont2[1:4,1:4]
#zero (column)s for errors
A_mat_cont3 = matrix(0,nrow=2*nro2, ncol=nro1, byrow=T)
#dim(A_mat_cont2)

#putting part 2a and 2b together:
A_mat_cont=cbind(A_mat_cont1,A_mat_cont2,A_mat_cont3,b2)
#dim(A_mat_cont)
#A_mat_cont[1:6,570:574]

#bounds (part3) #not needed => needed (17.2.2014), maybe not needed (20.2.2014)
A_mat_contb = matrix(0,nrow=2*nro2, ncol=lhs, byrow=T)
A_mat_cont2a = matrix(0,nrow=2*nro2, ncol=nro2, byrow=T)
# I need looping:
for (j in 1:dim(A_mat_cont2a)[2]) (
  A_mat_cont2a[(b2[,j]!=F),j]=rbind(1,1) )
```

```
A_mat_contb[,2:(nro3)]=A_mat_cont2a
#dim(A_mat_contb)
#A_mat_contb[1:6,570:574] #ok

#restriction of x_s (part4)
A_mat_res = matrix(0,nrow=1, ncol=dim(A_mat_cont)[2], byrow=T)
A_mat_res[,(ae+1):lhs]=1

#the additional
#old:
#A_mat_more=rbind(A_mat_cont,A_mat_contb,A_mat_res)
#dim(A_mat_more)
#"new old:
A_mat_more=rbind(A_mat_cont,A_mat_res)

# making of vector 1)
v_e=rep(c(1,-1), nro2)
#v_i=matrix(1, nrow=nro1)
#dim(v_i)
#v_e = as.vector(rbind(v_i,-v_i))
#length(v_e)
#miRNA list (check: sortmiro)
#mir_val=t(data_zs_t[(rownames(data_zs_t) %in% mirs),])
#dim(mir_val)
# Make a matrix (mir_neg_pos) that has every other row
# as a negative value of the previous matrix (mir_val):
#mir_neg_pos=matrix(nrow=(oe), ncol=nro2)
#for (i in 1:(oe)) (
#  if (i %% 2 !=0) (mir_neg_pos[i,]=mir_val[(1+(i-1)/2),])
#  else (mir_neg_pos[i,]=-mir_val[(i/2)],) ) # ok
#if (sum(is.na(mir_neg_pos)!=F)>0) (mir_neg_pos[is.na(mir_neg_pos)] <- 0)

mir_neg_pos=matrix(nrow=oe, ncol=nro2)
#for (i in 1:(oe)) (
#  mir_neg_pos[i,] = ifelse (i %% 2 !=0, mir_val[(1+(i-1)/2)], -mir_val[(i/2)]) )

mirval2=mir_val[rep(1:nrow(mir_val),each=2),]
mirval2=rep(c(1,-1),dim(mir_val)[1])*mirval2

mir_neg_pos=mirval2

#mir_neg_pos[1:6,1:6]
#dim(mir_neg_pos)
## The zero list (b):
a=-rbind(1,1) # this is the double. And before the loop an empty list is created:
Lst=c()
for(i in 1:nro1)(Lst[i]=list(a)) # The "a" needs to be in a list
b=data.matrix(bdiag(Lst)) # this is how you use your list to construct this matrix.

#column 1
A_mat_f[1:length(v_e),1]=v_e
# columns 2:(nro2+1)
A_mat_f[1:(oe),2:(nro3)]=mir_neg_pos
# lhs   = (1+nro2+nro1+nro2)
A_mat_f[1:(oe),(nro3+1):(ae)]=b #see comments about the type of matrix above
#dim(b)
#dim(A_mat_f[1:(oe),(nro3+1):(ae)])
```

```
# Final addition row-wise:
A_mat_f[(oe+1):rhs,]=A_mat_more #finally
#dim(A_mat_f[(oe+1):rhs,])

# Making the model list values:
model_f <- list()
# Making of the objective function
obj_f <- vector(mode = "numeric", length = lhs)
obj_f = "[<-"(obj_f,1:lhs,value=0)
obj_f = "[<-"(obj_f,((nro3+1):ae),value=1)

model_f$obj = obj_f
model_f$modelsense <- "min"
#model_f$sense      <- c(rep('<=', oe), rep(rbind('<=', '>='),2*nro2), '=')
model_f$sense      <- c(rep('<=', oe), rep(rbind('<=', '>='),nro2), '=')
#length(model_f$sense)
model_f$lb         <- c(-m,rep(-m,nro2),rep(-e_lim,nro1),rep(0,nro2))
model_f$ub         <- c(m,rep(m,nro2),rep(e_lim,nro1),rep(1,nro2))
#length(model_f$ub )
model_f$vtype      <- c(rep('C',ae), rep('B',nro2))
#length(model_f$vtype)
#lhs
# Making the Right side of equations:
#model_f$rhs       <- as.vector(c(rbind((e11_zs_t[gene_nam,]),(-e11_zs_t[gene_nam,])),(rep(0,2*nro2)),(rep(c(m,-
m),nro2)),w))
model_f$rhs       <- as.vector(c(rbind((e11_zs_t[gene_nam,]),(-e11_zs_t[gene_nam,])),(rep(0,2*nro2)),w))
#length(model_f$rhs )

# When you got A_mat_f ok, then this works (if row and column numbers are ok)
model_f$A          <- A_mat_f
#mir_val[1:4,1:4]
#e11_zs_t[gene_nam,]

#write.table(A_mat_f, "amf.txt", sep="\t")
#params_f <- list(MIPGap=0.01,TIME_LIMIT=12000, ResultFile='model_f.lp') #If doing restriction analysis
#params_f <- list(MIPGap=0.05,TIME_LIMIT=300, ResultFile='model_f.lp') #If doing restriction analysis (small)
params_f <- list(TIME_LIMIT=TL, ResultFile='model_f.lp')
#params_f <- list(Method=-1,ResultFile='model_f.lp') #normally
result_f <- gurobi(model_f,params_f)
#result_f <- gurobi(params_f)

#more info
limo_mir=cbind(result_f$x[1:(nro2+1)]) #ok, I think this is working now :)
#max(result_f$x[nro3:ae])
colnames(limo_mir)=c('lin.mod.coefficient')
rownames(limo_mir)=c("beta_0",colnames(mir_val))

# Ordering the miRs according to coefficient
limo_mir2=limo_mir
limo_mir2=as.matrix(limo_mir[rev(order(limo_mir)),])
colnames(limo_mir2)=c('lin.mod.coefficient')
# Finding overlapping ones to the all miR list
en_mir=as.matrix(limo_mir2[rownames(limo_mir2) %in% names(trg_dmm4),])
colnames(en_mir)=c('lin.mod.coefficient')

#Multiplication:
```

```
  b_m_i_f=data_zs_t2[(rownames(data_zs_t2) %in% rownames(limo_mir)),]
  #table(rownames(data_zs_t2) %in% c("hsa-miR-30d-5p"))
  limo_miri=limo_mir[rownames(limo_mir) %in% rownames(data_zs_t2)]
  #dim(b_m_i_f)
  #length(limo_miri)
  ert=dim(data_zs_t2)[2]
  # matrix (or vector) multiplication is needed :)
  g_pred_f_i=(b_m_i_f*as.numeric(limo_miri))
  auo=if (length(g_pred_f_i) > ert) dim(g_pred_f_i)[2] else length(g_pred_f_i)
  #dim(g_pred_f_i)

  g_pred_fo=c(1:auo)
  #length(g_pred_fo)
  #dim(g_pred_f_i)[2]

  g_pred_fo = "[<-"(g_pred_fo,1:auo,value=result_f$x[1])

  g_pred_f=rbind(g_pred_fo,g_pred_f_i)
  if (sum(is.na(g_pred_f)!=F)>0) (g_pred_f[is.na(g_pred_f)] <- 0)
  #g_pred_f[1:3,1:3]

  # but the equations continue...:
  g_pred_val_f=NULL
  for (j in 1:dim(g_pred_f)[2])
    (g_pred_val_f[j]=sum(g_pred_f[,j]))
  #g_pred_val_f[1:3]
  #length(g_pred_val_f)

  # And the correlation
  y=as.numeric(e11_zs_t2[gene_nam,])
  x=as.numeric(g_pred_val_f)
  cor=cor(y, x, method = "pearson") #different values if I

  return(list(result_f, limo_mir2, en_mir, cor))
}

#Ok, lets make a function for all:
fi_loop2=function ( e11_zs_t, e11_zs_t2, data_zs_t, data_zs_t2,mir_amount, list) {
  library("gurobi")
  library("Matrix")

  # Making the initial vector:
  mg_list=vector(mode="list", length=length(list))
  names(mg_list) <- f_ent_gsymb(list)

  gene_nam=vector(mode="list", length=length(list))
  mirs=vector(mode="list", length=length(list))

  mi=mir_amount

  for (i in 1:length(list)) {
    gene_nam[[i]] = as.vector((list)[i])
    mirs[[i]] = f_mir_limoo(tush_nl, as.vector(unlist(gene_nam[[i]])))
    mg_list[[i]] = f_limo_restr2(e11_zs_t, e11_zs_t2,data_zs_t, data_zs_t2,as.vector(unlist(gene_nam[[i]])),
as.vector(unlist(mirs[[i]])),mi)  }
  #here are the values:
  return(mg_list)
```

```
}
#Test:
l_tot_s_test1=fi_loop2( e11_zs_t, e11_zs_t2, data_zs_t, data_zs_t2,mir_amount, lists_s_tot)
ltot1_req=f_requested_limooo(l_tot_s_test1,lists_s_tot) #See description later

#Get the values for your validation:
f_train_validate2= function (lim_anno, e11_zs, data_zs, train1, train2, validate, list) {
 # The most important:
 # Selecting appropriate values (e.g. e_n, e_t) for the function to train:
 e_nrtn=c(train1,train2)
 data_zs_t=data_zs[,colnames(data_zs) %in% e_nrtn]
 e11_zs_t=e11_zs[,colnames(e11_zs) %in% e_nrtn]

 #lim_anno is the limited annotation: dim: 567  21
 # group is the validation group name in annotation (column 20)
 # Selecting Luminal A group for the validation:
 a11=lim_anno
 log_a11_na=is.na(a11[,20]) #so nas are evaluated
 log_a11_la=(a11[,20]==as.character(validate)) #la comes from luminal A but it could be something else as well,
group is evaluated
 lA=a11[(log_a11_na!=TRUE & log_a11_la),20] #no nas but all group members
 a11_rn=rownames(a11)
 la_rn=a11_rn[log_a11_na!=TRUE & log_a11_la] #for names in the column

 #values for the validation:
 e11_zs_t2=e11_zs[,la_rn]
 data_zs_t2=data_zs[,la_rn]

 return(list((e11_zs_t), (data_zs_t), (e11_zs_t2), (data_zs_t2)))
}

ftv2o=f_train_validate2(lim_anno=a11, e11_zs, data_zs,  train1=e_n, train2=e_t, validate = 'Luminal A', list)
e11_zs_to=as.matrix(ftv2o[[1]]) #e11_zs_to[1:5,1:5] #ok e11_zs_t[1:5,1:5]
data_zs_to=as.matrix(ftv2o[[2]]) #data_zs_to[1:5,1:5] #ok data_zs_t[1:5,1:5]
e11_zs_t2o=as.matrix(ftv2o[[3]]) #e11_zs_t2o[1:5,1:5] #ok e11_zs_t2[1:5,1:5]
data_zs_t2o=as.matrix(ftv2o[[4]]) #data_zs_t2o[1:5,1:5] #ok data_zs_t2[1:5,1:5] #oks

#Test:
l_tot_s_test2=fi_loop2( e11_zs_t, e11_zs_t2, data_zs_t, data_zs_t2,mir_amount, lists_s_tot)
ltot1_req2=f_requested_limooo(l_tot_s_test2,lists_s_tot) #See description later

# Shuffling the patient names to check if my results are arising from random or not
f_shuffle_rename = function (e11_zs, n_times) {
   #sample vector
         c_nam_e11_zs=vector("list", n_times)

   for (i in 1:n_times) (
    c_nam_e11_zs[[i]]=colnames(e11_zs)[sample(1:dim(e11_zs)[2], dim(e11_zs)[2], replace=F)]
   )
   return(c_nam_e11_zs)
 }

# Testing the model with random shuffling of the patient names:
suffles=f_shuffle_rename(e11_zs, 20)

#This is for the random all model:
f_limos_rnd2 = function (e11_zs_t, e11_zs_t2, data_zs_t,  data_zs_t2,  mir_amount, list, suffles) {
```

```
  tot=vector("list", length(suffles))
  for (i in 1:length(suffles)) (
    tot[[i]]=fi_loop2(e11_zs_t[,suffles[[i]]], e11_zs_t2[,suffles[[i]]], data_zs_t,  data_zs_t2, mir_amount, list)
  )
  return(tot)
}
}
shuffling_comp_new=f_limos_rnd2(e11_zs_t, e11_zs_t2, data_zs_t,  data_zs_t2,  mir_amount, list, suffles)

## 15e: cross-validation:
#Validation groups e11_zst etc:
f_cval_grp= function (sample_nro,div_nro) {
  ab=sample_nro # sample nro vector in 1:i, e.g. 1 2 3 ... i
  max =div_nro #amount of samples in the sample pool
  x <- seq_along(ab)
  d1 <- split(ab, ceiling(x/max))
 #d1 #http://stackoverflow.com/questions/3318333/split-a-vector-into-chunks-in-r
  #str(d1)

  a=NULL
  b=NULL
  for (i in 1:length(d1)) (
    a[[i]]= d1[[i]])
  for (i in 1:length(d1)) (
  b[[i]]=c(1:567)[(c(1:567) %in% d1[[i]])!=T] )
  e_val1=NULL
  e_val2=NULL
  d_val1=NULL
  d_val2=NULL

  for (i in 1:length(d1)) (
    e_val1[[i]] = e11_zs[,b[[i]]] )
  for (i in 1:length(d1)) (
    e_val2[[i]] = e11_zs[,a[[i]]] )
  for (i in 1:length(d1)) (
    d_val1[[i]] = data_zs[,b[[i]]] )
  for (i in 1:length(d1)) (
    d_val2[[i]] = data_zs[,a[[i]]])

  return(list(e_val1,e_val2,d_val1,d_val2))}

  #cv codes
 f_cm_walk_opwj = function (a, mir_amount, list,TL,MG) {
  library("gurobi")
  library("Matrix")


  #Cross-validation values:
  e11_zs_t=NULL
  e11_zs_t2=NULL
  data_zs_t=NULL
  data_zs_t2=NULL

  # Making the initial vector:
  mg_list=vector(mode="list", length=length(a[[1]]))
  #names(mg_list) <- f_ent_gsymb(list)
  names(mg_list) <- c("G1","G2","G3","G4","G5")
```

```
  # for the restriction linear model:
  gene_nam=vector(mode="list", length=length(list))
  mirs=vector(mode="list", length=length(list))

  mi=mir_amount
  #str(a)
  for (j in 1:length(a[[1]])) {
    e11_zs_t[[j]]=a[[1]][[j]]
    e11_zs_t2[[j]]=a[[2]][[j]]
    data_zs_t[[j]]=a[[3]][[j]]
    data_zs_t2[[j]]=a[[4]][[j]]

  for (i in 1:length(list)) {
    gene_nam[[i]] = as.vector((list)[i])
    mirs[[i]] = f_mir_limoo(tush_nl, as.vector(unlist(gene_nam[[i]])))
    mg_list[[j]][[i]] = f_limo_restr2(e11_zs_t[[j]], e11_zs_t2[[j]],data_zs_t[[j]], data_zs_t2[[j]],
              as.vector(unlist(gene_nam[[i]])), as.vector(unlist(mirs[[i]])),mi,TL,MG)
    names(mg_list[[j]][[i]]) <- f_ent_gsymb(list[[i]])}
  }
  #here are the values:
  return(mg_list)
}

 f_cv_cor1 = function (a,tsc5,cv_0.5) {
#data, list and cv result
t_i=vector("list",length=length(tsc5))
for (j in 1:length(t_i)) (

  for (i in 1:length(a[[1]]) ) (
    t_i[[j]][[i]]=cv_0.5[[i]][[j]][[4]] ) )

at=as.matrix(mapply(function(y) mean(y[1:5]), t_i))
rownames(at)=f_ent_gsymb((tsc5))
colnames(at)=c('cor_coefficient') #,'tot_miRs_used','miRs_below_zero','miRs_enr')
at=as.matrix(at[rev(order(at[,1])),])
atm=apply(at,2,median) #median of mean values of correlations

#mean(unlist(sapply(t_i[1:5], '[', 'element')))
return(list(at, atm)) }

  # looping genes in the list, a is the sample groups
  f_cmo_m= function (a,  mir_amounts, list, TL,MG) {
  tot=vector("list", length(mir_amounts))
  mir_amount=NULL
  cor_cofs=NULL
  for (i in 1:length(mir_amounts)) {
    mir_amount[[i]]=mir_amounts[[i]];
    tot[[i]]=f_cm_walk_opwj(a, mir_amount[[i]], list,TL,MG);
    cor_cofs[[i]]=f_cv_cor1(a,list,tot[[i]]) }
  return(cor_cofs)
}

### Step 15f ###
# Obtaining results after linear modelling analysis

# 1) Requested information 1:
```

```
# the correlation coefficients,
# how many miRNAs you have used for the predictions,
# how many of them are above zero
# how many of all of the miRNAs are enriched ones

# A function for this:
f_requested_limooo = function (sl_valid_test2,list) {
  aeu <- vector("list", length(list))

  for (i in 1:length(aeu)) {
    aeu[[i]][1]=as.matrix(sl_valid_test2[[i]][[4]])
    aeu[[i]][2]=(length(sl_valid_test2[[i]][[2]])-1)
          aeu[[i]][3]=sum(sl_valid_test2[[i]][[2]]>0)
    aeu[[i]][4]=(length(sl_valid_test2[[i]][[3]]))

  }
  aeu3=aeu

  #The additional information
  x=f_ent_gsymb((list))
  aeu_et=do.call(rbind, aeu3)
  colnames(aeu_et)=c('cor_coefficient','tot_miRs_used','miRs_mod_nz','miRs_selected')
  rownames(aeu_et)=x
  aeu_et=aeu_et[rev(order(aeu_et[,1])),]
  write.table(aeu_et, "top_150_cor_gene_alpat3_1.txt", sep="\t")
  return(aeu_et)
}

# 2) All correlations for random validation:
f_cor_rnds = function (shuff_comp_new,comp_list_length) {
  #In case of top 150 list, comp_list_length was 150...
cor_suf2a=vector("list",comp_list_length)
for (i in 1:length(shuff_comp_new)) (
  for (j in 1:length(cor_suf2a)) (
    cor_suf2a[[j]][[i]]=shuff_comp_new[[i]][[j]][[4]] ))
names(cor_suf2a)=names(shuff_comp_new[[1]])
hn1=(unlist(cor_suf2a))
return(hn1)
}

hn1=f_cor_rnds(shuff_comp_new,150)

# 3) Calculating means, and Doing Histograms of linear model correlations, and t-testing these correlations (plot all in
same picture with powerpoint):
#rnd
mean(hn1)
#valid
hn2=(l3_r_oklma[,1])
mean(hn2) #[1] 0.2983685
#nrm
hn3=(l3_r_ok[,1])
mean(hn3) #[1] 0.5517583

# Histogram Grey Color
hn1a=hist(hn1, col=rgb(0.1,0.1,0.1,0.5),xlim=c(-0.2,0.8), ylim=c(0,250),breaks=25, main="Overlapping Histogram")
hn1b=hist(hn2, col=rgb(0.8,0.8,0.8,0.5), add=T,breaks=25,xlim=c(-0.2,0.8),ylim=c(0,25))
hn1c=hist(hn3, col=rgb(0.4,0.4,0.4,0.4), add=T,breaks=25,xlim=c(-0.2,0.8),ylim=c(0,25))
```

```
#Scaling (to the smallest curve i.e. here hn1c, according to its counts, without changing it)
hn1a[[2]]=hn1a[[2]]*max(hn1c$counts)/max(hn1a$counts)
hn1b[[2]]=hn1b[[2]]*max(hn1c$counts)/max(hn1b$counts)
#Plotting
plot(hn1a,col=rgb(0.4,0.1,0.1,0.1),xlim=c(-0.2,0.9),ylim=c(0,15),xlab='Correlation coef.',ylab='Genes with this
correlation (scaled)',main="")
plot(hn1b,col=rgb(0.1,0.1,0.1,0.5), add=T,xlim=c(-0.2,0.8),ylim=c(0,15))
plot(hn1c,col=rgb(0.8,0.8,0.8,0.8),add=TRUE)
lines(density(hn1))
lines(density(hn2))
lines(density(hn3))

# Just densities:
plot(density(rndl_cor_f2),xlim=c(-0.3,0.9),ylim=c(0,8),xlab='Correlation coefficient',ylab='Genes with this correlation
(scaled)',main="")
lines(density(hn2))
lines(density(hn3))

#Testing the correlations with each other using t-test:
#rnd against norm:
cor_t2_rnd_nrm=t.test(hn1,hn3)
cor_t2_rnd_nrm[3] #[1] 9.346306e-74

#rnd against validation:
cor_t2_rnd_val=t.test(hn1,hn2)
cor_t2_rnd_val[3] #[1] 4.492909e-10

#validation against norm:
cor_t2_val_nrm=t.test(hn2,hn3)
cor_t2_val_nrm[3] #[1] 2.988722e-22

# Boxplotting interesting gene , enriched miRNA miRNA combinations

gene_name=c("LUZP1")
f_bp_eval=function (lst_fnlm,nrm_tst_f,gene_name) {
  #for the enriched miRNAs
  auk=as.character(gene_name)
  auk=as.character(f_gsymb_ent(auk))
  nrou=c(1:length(lst_fnlm))[lst_fnlm %in% auk]

qkireg2=gen_rgl_mir2[gen_rgl_mir2[,1] %in% names(nrm_tst_f[[nrou]][[3]][nrm_tst_f[[nrou]][[3]][,1]<0,]),] #"hsa-
miR-3609","hsa-miR-29b-3p"
qkireg2=qkireg2[rev(order(qkireg2[,2])),]
#qki=gen_rgl[gen_rgl[,1] %in% auk,]

# for 2 best enr mirs
umt=data_zs[(rownames(data_zs) %in% qkireg2[1,1]),(colnames(data_zs) %in% e_t)]
umn=data_zs[(rownames(data_zs) %in% qkireg2[1,1]),(colnames(data_zs) %in% e_n)]
umt2=data_zs[(rownames(data_zs) %in% qkireg2[2,1]),(colnames(data_zs) %in% e_t)]
umn2=data_zs[(rownames(data_zs) %in% qkireg2[2,1]),(colnames(data_zs) %in% e_n)]
  # for the gene
dt=e11_zs[(rownames(e11_zs) %in% auk),(colnames(e11_zs) %in% e_t)]
dn=e11_zs[(rownames(e11_zs) %in% auk),(colnames(e11_zs) %in% e_n)]

return(boxplot(umn,umt,umn2,umt2,dn,dt,col = rep(c("blue","red"),3), ylab="z-scored expression level")) }

bp_eval=f_bp_eval(lst_fnlm,nrm_tst_f,gene_name)
```

```
#the most regulated enriched miRNAs (according to the limo):
f_bp_eval2=function (lst_fnlm,nrm_tst_f,gene_name,emrs) {
  auk=as.character(gene_name)
  auk=as.character(f_gsymb_ent(auk))
  nrou=c(1:length(lst_fnlm))[lst_fnlm %in% auk]

  qkireg2=gen_rgl_mir2[gen_rgl_mir2[,1] %in% names(nrm_tst_f[[nrou]][[3]][nrm_tst_f[[nrou]][[3]][,1]<0,]),] #"hsa-
miR-3609","hsa-miR-29b-3p"
  qkireg2=qkireg2[rev(order(qkireg2[,2])),]
  qkireg2= if (dim(qkireg2)[2]<emrs) (qkireg2=qkireg2) else (qkireg2=qkireg2[c(1:emrs),])
  #dim(qkireg2) #traditional worked ok...
  #qki=gen_rgl[gen_rgl[,1] %in% auk,]
  umn=NULL
  umt=NULL
  for (i in 1:emrs) {
    umn[[i]]=data_zs[(rownames(data_zs) %in% qkireg2[i,1]),(colnames(data_zs) %in% e_n)]
    umt[[i]]=data_zs[(rownames(data_zs) %in% qkireg2[i,1]),(colnames(data_zs) %in% e_t)]

  }
 #For the genes
  dt=e11_zs[(rownames(e11_zs) %in% auk),(colnames(e11_zs) %in% e_t)]
  dn=e11_zs[(rownames(e11_zs) %in% auk),(colnames(e11_zs) %in% e_n)]
  dt=as.data.frame(dt)
  dt=as.list(dt)
  dn=as.data.frame(dn)
  dn=as.list(dn)

  c_tot=c(umn,umt)
  #Selecting normals and tumours (mirs)
  cu=NULL
  for(i in 1:emrs) {
    cu[[i]]=list(1+(i-1),emrs+(i))
  }

  cu=unlist(cu)
  c_tot=c_tot[cu]
  #combining miRs and genes
  c_toti=c(c_tot,dn,dt)

  #boxplot(x = as.list(umt)) #this works but I need them all
  #http://stackoverflow.com/questions/11346880/r-plot-multiple-box-plots-using-columns-from-data-frame
  ae=boxplot(c_toti,col = rep(c("blue","red"),3), ylab="Z-scored expression level")
  return(list(qkireg2,ae)) }

# bp to luzp1 with best up.reg.mirs to next function f_bp_eval3
c(1:247)[names(nrm_tst_f) %in% c("LUZP1")] #27
rownames(nrm_tst_f[[27]][2][[1]])
cond_m=c(1:332)[gen_rgl_mir2[,1] %in% rownames(nrm_tst_f[[27]][2][[1]])]
luz_mir_up=gen_rgl_mir2[cond_m,]
luz_mir_up=luz_mir_up[rev(order(luz_mir_up[,2])),]
luz_mir_upi=luz_mir_up[1:2,]

#most up-rgulated (selcect by hand in limo) mirs are checked with gene
f_bp_eval3=function (lst_fnlm,nrm_tst_f,gene_name,emrs,luz_mir_upi) {
qkireg2=luz_mir_upi #if you want to select mirs then use: luz_mir_upi
umn=NULL
```

```
umt=NULL
for (i in 1:emrs) {
  umn[[i]]=data_zs[(rownames(data_zs) %in% qkireg2[i,1]),(colnames(data_zs) %in% e_n)]
  umt[[i]]=data_zs[(rownames(data_zs) %in% qkireg2[i,1]),(colnames(data_zs) %in% e_t)]

}

#For the genes
dt=e11_zs[(rownames(e11_zs) %in% auk),(colnames(e11_zs) %in% e_t)]
dn=e11_zs[(rownames(e11_zs) %in% auk),(colnames(e11_zs) %in% e_n)]
dt=as.data.frame(dt)
dt=as.list(dt)
dn=as.data.frame(dn)
dn=as.list(dn)

c_tot=c(umn,umt)
#str(c_tot)
cu=NULL
for(i in 1:emrs) {
  cu[[i]]=list(1+(i-1),emrs+(i))
}

cu=unlist(cu)
c_tot=c_tot[cu]
c_toti=c(c_tot,dn,dt)
ae=boxplot(c_toti,col = rep(c("blue","red"),3), ylab="Z-scored expression level")
return(list(qkireg2,ae))
}
```

```
# 20.11.2013/03.06.2014, Pauli Tikka, University of Turku, thesis work, and beyond.

# Here I insert all the essential matters, codes, and functions that I have discovered during coding with R

# Otherwise SEE THE COMPLETE CODES AND INITIAL STEPS AT Tikka_complete.R or the raw code is at Tikka_raw.R

# Contents:

# General commands, Matrix check and corrections, Plotting, Functions, Tests, Libraries, General learning


### GENERAL COMMANDS ###

#It is possible to see the variables with

ls() #e.g. a, ae, aen, anno, a1a,...

#One can load the variables from the save place by:

save(-all variables as the ones below-, file='pauli.Rdata')

load('pauli.Rdata')

# It is better to save one variable/matrix/list as RData, because write.table does not always give you the right format:

save(list2, file='list2.RData')

# Removing all the variables:

rm(list=ls(all=TRUE))

# If you save the workimage (q() -> yes), and then double click the work image icon => variables/packages are
"installed" again, BUT

# The packages are not loaded within work image.

# Check the INSTALLED packages with:

library()

# Check the LOCATION of installed packages with:

.libPaths()

# Check the LOADED packages with:

search()

# working directory, where all the files are saved, or read, may be checked (or set):

getwd()                                  # (setwd("C:/Users/Pauli.../"))

#Get the text file data (from working directory) with a table (i.e. as a matrix) to R:

read.table(file="sud.txt")

#You can ask for help using the ? command as in:

?read.csv
```

```
# Taking indent away (extra tab spaces or equivalent):
#ctrl + I (in R studio, & check that there are no missing brackets at the end of script)
# We can obtain documentation on a particular package using the help= option of library():
library(help=rattle)


### MATRICES ###
## Matrix accessing, findings and selections ##
# How to call subset of MATRICES:
# REMEMBER matrix A, ROWS r (from 1 to 10 (or n): 1:10), COLUMNS c (similarly) => A[r,c]
# z[1:5,1:2]
#       TCGA-B6-A0RG-01A-11R-A056-07 TCGA-AO-A12F-01A-11R-A115-07
# X2                    6026.5606                57389.091
# X11057               19828.1973                 1274.011
# X2180                  366.3702                 2508.550
# X59                   5006.5525                15131.355
# X60                  93823.7917                124788.090
## Accessing individual element of a matrix
real[55,99]
# Accessing column of a matrix, NOTE THE ''s!!!:
#The following should do it:
new_datat[,'hsa.let.7a.1.']
# similarly to rows:
new_datat['TCGA.A1.A0SB.01',]
#Bonus information
x[,c('name 1','name 2')] #would return two columns just as if you had done: x[,1:2]
x[c('row 1','row 2'),] # And if rows were named..., And finally, the same operations can be used to subset rows: x[1:2,]
# Checking if the conversion of some data is ok (match maybe used instead of using rownames:
result_3mi[match(rownames(dtt_z), result_3mi[,2]),][66:75,] #dtt_z has been converted
# Accessing different places of a matrix (or vector)(e.g. a=matrix(nrow=7,ncol=7))
a[c(1,2, 5:6),c(2:3,7)]
```

# Accessing only certain rows in a matrix with condition (e.g. rows that have values above 2 in first column)

a[a[,1]>2,]

# Accessing certain rownames of matrix (that occur in another, and if not nas are put in place):

targets_ok2[de_miro2[,1],]

# Apply a condition to a matrix numeric value, where p-values below 0.05, and their combined matrix (str(mdgu2): chr):

a=as.numeric(mdgu2[,2])

b=a<0.05;

mdgu2[b,]

#Accessing (all values of a) column of the matrix (note that column names are variable names):

mean_445=mean(real$X445)

# OR!! (if the previous is not working)

mean_445=mean(realn[,'X435'])

# General matrix information

all(mat=0)  # range(x <- sort(round(rnorm(10) - 1.2,1))), if(any(x < 0)) cat("x contains negative values\n"), if(all(x < 0)) cat("all x values are negative\n")

str(mat)  # see all the description of variables (that are in columns)

dim(mat)  # this will tell you the size of the matrix (if it is numeral)

logical(3)  # At the moment I am not quite sure the necessity of this function, because it produces in this case: [1] FALSE FALSE FALSE

table(mat)  # if it is logical values that you have, then this should be used for checking the size of the variable (matrix)

sum(mat)  # this is also for logicals, the values that are TRUE

length(mat)  #-"-, amount values in total

# General information about the expression set (e.g. eset)

sampleNames(eset)[1:5]

featureNames(eset)[1:5]

# The dimension of columns can be checked:

dim(retn2l)[2] # OR, ncol (retn2l)

# Selecting only unique column names from matrix:

new_data=new_data[,unique(colnames(new_data))]

# Selection of unique row entries (the names or values) of a single column (one in this case), (similarly to rows):

```
unique(mir_targ[,1])

#the amount of 'not-available's (NAs)

sum(is.na(x))

#the amount of infinite values (similarly with finite values)

sum(is.infinite(x))


## Matrix generation and combination ##

# Making basic matrix

b= matrix( c(8,7,13,12,26,8,8,25,12,13,12,10), nrow=12, ncol=1, byrow=TRUE)

## making a matrix and preassigning the values as zero

sum = matrix(ncol=770,nrow=41)

sum = "[<-"(sum,value=0) # all values are zero

# Insert a matrix inside a matrix:

A_mat[,37:603]=b # if b is a special matrix do first: b=data.matrix(b)

# Add row to a matrix:

rbind(...)

# Add column to a matrix:

cbind(...)

# ok, if you want to do MATRIX (and not a list) USE DATA.FRAME

# (and not cbind which does list, and so you get these akward ""s...), notice also: stringsAsFactors=FALSE

im=data.frame(miR_de_name,miR_de_pval, stringsAsFactors=FALSE)

# also

att=data.frame(sort.im[,1])

# One can combine matrices and variables with data.frame command, notice also the transposition with t command (retn3=t(retn2)) :

all_exp_anno=data.frame(annon, retn3) # and rename them shortly: aea=all_exp_anno

# It is also possible to fetch data with read.table command, from internet:

fpe <- read.table("http://data.princeton.edu/wws509/datasets/effort.dat")

# Make a rowwise addition of repetitions (i.e. the row values in each column change accordingly)

a=matrix(rep(c(2,7,4,1),1),nrow=8, ncol=8)

# When making a new variable, it should start with letter, and not with number (hm_29, and not 29_hm..):
```

```
hm_29_3_pt=apina[h,2]

#Make a nrm distributed random matrix:

x=matrix(rnorm(200,mean=0, sd=1), nrow=20,ncol=10)

## Making a double -1 diagonal matrix:

a=-rbind(1,1) # this is the double. And before the loop an empty list is created: Lst=c()

for(i in 1:5)(Lst[i]=list(a)) # The "a" needs to be in a list

bdiag(Lst) # this is how you use your list to construct this matrix.

# Constructing an expression set matrix (eset) with annotation matrix (anno), and expression matrix (ret):

metadata = data.frame(labelDescription = c("Exp", ..., "RPPAClusters"))

pheDa=new("AnnotatedDataFrame",data=anno, varMetadata = metadata)

# note that exp must be matrix: exp=as.matrix(exp) (and not data.frame)

# https://stat.ethz.ch/pipermail/bioconductor/2011-November/042131.html (7.1.2014)

eset <- new("ExpressionSet", exprs = exp, phenoData = pheDa)

# Expression set (e.g. called eset) works as a matrix with exprs function:

exprs(eset)[1:10,1:2]


## Matrix removals and changes ##

# Replacing near zero value with small value

dat[dat < 0.1] <- 0.1

# replacing NAs with zeros:

dat[is.na(dat)] <- 0

# removing NAs

a_vs=a_vs[!is.na(a_vs)]

reta=as.numeric(ret, rm.na=TRUE)

# Removing na rows from matrix (notice that here I assume that if it starts with na, then all of them are nas):

tush_n=tush_n[(!is.na(tush_n[,1])),]

# removing nas (with loop from one column): h=c()

for(i in 1:dim(targets)[1]) (

if(targets[i,1] == 'na') (h=c(h,i)) ) # the application: targets_ok=targets[-h,]

# If you want to remove something else than na:s (although in this example 'na' was used): h=c()
```

for(i in 1:dim(mir_337)[1]) {if(mir_337[i,1] =='na'){h=c(h,i)}} # the application: mir_337=mir_337[-h,]

# Removing rows of nas (similarly to columns)

trg_d[f_not_na(trg_d)!=0,] # where f_not_na is a function that I made explained in functions and scripts sections

# Set infinite values to zero

x[x==Inf] <- 0

# Replace NA values with other values:

y <- which(is.na(int_gen23)==TRUE) # This is how you get the indexes

int_gen23[y]=ig23l # ig23l=c("7490","383","5009","4842","2271","189","5563","2645","5163","5313","5106"), and length(int_gen23): 34

# Changing (whole) columns in a matrix (e.g. im) (this also apply to rows, but then ...[1,...]):

im[,1]=a

# Naming(/changing) the names of the columns (this can be done also for rows)

colnames(pvalmiR)<-c('miRNA','p-value')

# Changing rownames of matrix:

row_new=gsymb[,1] # notice that I am accessing all of the rows of the first column! Renaming: exp_mtn=exp_mt

rownames(exp_mtn, do.NULL = TRUE, prefix = "row")

rownames(exp_mtn) <- row_new

#Deleting a column

miR_t_ok=subset( miR_t_ok, select = -V2 ) # or

df$x = NULL

# delete a row from matrix:

df = df[-1,]

# Change data.frame matrix as numerical by using as.matrix command (needed e.g. in Wilcox test)

# Select every nth value of matrix (here every second):

# pre-info: bi=A_mat_cont2, Lt=c(), for(i in 1:(nro2))(Lt[i]=list(bi)) # The "bi" needs to be in a list, b3=data.matrix(bdiag(Lt))

c <- 1:ncol(b3)

b3=b3[,(c %% 2 != 0)]

### LISTS, VECTORS, STRINGS ##

# Accessing the list values (notice ''s !!)

```
median_445_v2=median_445['X435']

# Accessing the list's (da) certain "list's" values (4th element of 55th list name)

da[[55]][,4]

# Change the variable names of lists:

names(mir_info)<-c("ccle","go_bp","go_cel","go_fnc","hga","kegg","mir","od","onco")

# Combining names of lists to another vector:

grp_tot_nam=unlist(list(names(group1), names(group2)))

# Combine two lists:

mir_tot_sel_g=c(mirs1,mirs2) #ok

#Select elements in lists:

tush_nl[c(1,5,6,7)]

#When you want to enter a singel variable to a function as a "name":

# DO NOT: gene_nam = as.vector(celf2) #not ok like this (celf2:[1] 10659), but

gene_nam = c("10659") # so that gene_nam: [1] "10659"

#Change the names of list factors:

a=as.matrix(list("Term", "Overlap", "P_value", "Adjusted_P_value", "Z_score", "Combined_Score",
        "Genes"))

names(mir_info$ccle)=a #http://stackoverflow.com/questions/6524294/working-with-dataframes-in-a-list-rename-variables

# Doing a list (of rows) that contains certain values:

a=-rbind(1,1) Lst=c() for(i in 1:10) (Lst[i]=list(a))

# Checking the list's third elements first value

Lst[[3]][1]

#Checkt the list's matrice's dimenstions by:

dim(udem_bp_l[[1]])

# Making an empty list

yht <- vector("list", 21) #ok

# Make a certain size of list elements list

mg_list[[2]]=vector(mode="list", length=length(over_ok))

# Selecting 0s away from list of lists [one entry]:

a=he2[[1]]!='0'
```

```
he2[[1]][a]

# Accessing only certain lists in a list

str(a[[1]][c(1,2,3)]) #notice double brackets after a[[]].

# list's lists' listelement's info

shuffling_comp[[1]][[2]][[4]]

# Even one step further (for matrix inside the element):

shuffling_comp[[1]][[2]][[2]][2,1] #for individual component

#Writing list to tables:

names(grp_tot_nam2)=c("group1", "group2","group3", "group4","group5","group6","group7","group8")

lapply(names(grp_tot_nam2), function(x, grp_tot_nam2) write.table(grp_tot_nam2[[x]],

paste(x, ".txt", sep = ""),col.names=FALSE, row.names=FALSE, sep="\t", quote=FALSE),grp_tot_nam2)

# Count the list elements:

length(unlist(a[[1]][1])) ##notice the UNLIST command

# Making a list that contain all specific list values (accessing both the names of a list and also its values):

a=iiid[2]            ## then str(a[[1]][c(1:10)]) ->List of 10 $ : chr [1:133] "hsa-let-7b-5p" ..., $ : chr [1:100] "hsa-let-7b-5p", and

                              ## str(a[[2]][c(1:10)]) Error in a[[2]] : subscript out of bounds

# Making a list that contain only certain values from original list (accessing just the values of the list):

a=iiid[[2]]  ## then str(a[[1]][c(1:10)])   -> chr [1:10] "hsa-let-7b-5p" "hsa-miR-15a-5p" "hsa-miR-16-5p", but notice:: !!!

                               ## str(a[[9]][c(1:10)]) chr [1:10] chr [1:10] "hsa-miR-17-5p"

#Accessing (vector) names inside a list of lists

names(gen_mir_ord[[5]][[1]])

names(gen_mir_ord[[5]][[1]])[4] #this is for a one name

#Finding a one specific entry of a vector:

c(1:247)[lst_fnlm %in% c("7798")]

# Finding certain entries of a vector (lmt_m_nro) (if there are nas in the list):

c(1:length(lmt_m_nro))[is.na(match(lmt_m_nro,0)==T)==F]

# Add more elements to a vector:

r_3mi_rn=c(rownames(result_3m),rest_mimat)

# How to make matrix (demot) as a vector, and then remove na values from list values (that have some nas)

demot2=lapply(seq_len(nrow(demot)), function(i) demot[i,]) # ok,
```

demot2 <- lapply(demot2, function(x) x[!is.na(x)]) # wei hou

# Select values from a matrix column (i.e. vector) that are not nas, and something else (and then apply this to get rownames, here:patients):

log_a11_na=is.na(a11[,20])

log_a11_la=(a11[,20]=='Luminal A')

lA=a11[(log_a11_na!=TRUE & log_a11_la),20]

a11_rn=rownames(a11)

la_rn=a11_rn[log_a11_na!=TRUE & log_a11_la]

# Selecting a certain condition to a list of list, and applying it (note that the original list is mir_inf2, where from mir2=mir_inf2[ab], and ab=rep(c(F,F,F,F,F,T,F,F),17)

for (i in 1:17)

  (print(mir2[[i]][(mir2[[i]][,4]<0.001),c(1,2,4)]))

# Sometimes when extracting information to a list one needs to add fill:

# Lapply command has several functions such as read.dta or read.csv, check the options from above internet site

mir_info <- lapply(mi, fill = TRUE, read.table) #http://stackoverflow.com/questions/19455070/confusing-error-in-r-error-in-scanfile-what-nmax-sep-dec-quote-skip-nli

# Make a text file out of a list: http://stackoverflow.com/questions/3044794/r-print-list-to-a-text-file

lapply(mylist, write, "test3.txt", append=T, ncolumns=1000 )

# Be careful what kind of list you are reading, e.g. is it csv or tab or some other separarted, here is for tab separated file (maybe csv):

mir_inf2 <- lapply(mit, sep="\t", quote = "", row.names = NULL,stringsAsFactors = FALSE,read.csv) #japadapa duuu!!

# command(-where from/to what matrix/lists/files-, arguments, function to apply) # command is here lapply

# where: mit <- file.path(path_to_files, files), and files = list.files(path_to_files, pattern="hsa-miR",all=F)

# Replacing empty spots as nas (in a matrix, this is important so that some list values do not get '1's, although there should not be anything)

tush_n[tush_n==''] <- 'na'

#  make a simple row list from matrix (of tush_n):

tush_nl=lapply(seq_len(nrow(tush_n)), function(i) tush_n[i,])

for (i in 1:length(tush_nl)) (

tush_nl[[i]]=as.vector(unlist(tush_nl[[i]])) ) # This loop, with vectorizing and unlisting, is needed to make a blunt element list (without list of list infos etc.)

# How to make a matrix from equal size of list of lists:

do.call(rbind, a)

```
# I have just nas in the list values, so removing them should go as follows:

do.call('cbind', l[!sapply(l, function(x) all(is.na(x)))])

#This is the amount of elements in a list

sapply(hero, length)

# Naming list values

names(demot2) = rownames(demot)

# Delete after a sentence after a certain word of list:

ae2[[3]]=strsplit(as.character(ae2[[3]])," doi:")[[1]][1]

# Make a list from another list, and then make a vector out of that new list, and find unique entries:

abcd=NULL

for (i in 1:150) {

abcd[[i]]=c(rownames(ibt3iii[[(i)]][[2]])) }

abcd2=unlist(abcd)

unique(abcd2)

# Making a vector (pitkula rivi "elementeillÃ¤", i.e. "character" string) from eset and normal dataframe matrix

e_d=as.matrix(colnames(exprs(eset_i[1,])))

ed=as.vector(colnames(dttv2))

# making of vector of 1s and -1s with certain length (nro1):

v_i=matrix(1, nrow=nro1)

v_e = as.vector(rbind(v_i,-v_i))

# This is how one access only the numerical values of a single vector (which might have "column names")

e_435=as.numeric(e_435)

# Changing values (of a vector from matrices) are made like this:

e_t1 = -e_435[1:20]

e_t2 = e_435[1:20]

e_t = as.vector(rbind(e_t1,e_t2))

# Making a vector with zeros:

obj <- vector(mode = "numeric", length = 6)

# Making a vector that has a certain amount of constants

model$sense      <- rep('<=', 20)
```

## Making a vector and inputting some values in side it:

obj=as.vector(1:20)

obj = "[<-"(obj,1:20,value=1) # Or these two commands more easily by: a=as.vector(matrix(1,nrow=20)), if single values but if reps then better:

obj = "[<-"(obj,1:20,value=rep(c(2,7,4,1),5))

# Comparing list (such as hero4) and vector values are possible if the list is "unlisted" (and possibly a column of a matrix vectorized as here with gen_rgl):

intersect(as.vector(unlist(hero4[3])),as.vector(gen_rgl[gen_rgl[,2]>0,1]))

### PLOTTING ###

# Values maybe plotted to screen of R studio etc. using e.g. hist function:

hist(a)

# OR (this is better, if possible)

function(..., file='test.pdf')

# One possibility is also to create a separate pdf file

pdf('test.pdf')

hv <- function(...) # function such as heatmap...

plot(hv)

dev.off()

# It seems that it is not possible to put file argument for hist and plot functions.

# Plot a matrix with histogram (unlisting it to vector).

hist(unlist(M_0))

# Plot two things to screen:

par(mfrow=c(1,2))

# If you want to name your elements in the plot, then first you need to define the variable.

# Labelling the names of the elements is just to add the name of the element to the label command: labels=...

l = (is.na(a_vs)) # define na values

vital = (a_ex == TRUE & a_tn == TRUE & l == F) # expression is true, triple negative is true, and nas are removed

vitl2 = anno[vital,] # your original annotation matrix should have only the elements of defined vector (here: vital)

plot(clust4, labels=vitl2$Vital.Status) # labels = FALSE, means no labels

```
# For elegant flowcharts:

#demo(flowchart)

#demo(plotmat)

#demo(plotweb)

# KnitR/pdf-compiler should work ok.


## MATCHING PROCEDURES ##

## Select a certain proportion of matrix:

x <- rownames(annoi)

y <- rownames(data3)

matches     <- which (x %in% y)

new_anno <- annoi[matches,]

# Order values of matrix

new_anno=new_anno[order(rownames(new_anno)),]

# If you want to order you matrix according to one variable (e.g. miR_de_pval)

sort.im=im[order(miR_de_pval),]

# Check out the "" (hipsut) numeric values by setting these "values" to as.numeric (which do not have ""s (eli hipsuja)):

axx=as.numeric(im[1:12,2])

# Checking the "" (hipsut) away from everywhere (including characters), use as.data.frame:

dm3=as.data.frame(cbind(dm[,1],dm2))

# The "" may also be "removed"/(or "added"), or better still ignored so that the matrices you are comparing have same amount of ""s, by using as.matrix:

as.matrix(miR_t[[1,1]])

as.matrix(im[[1,1]])

# See how many unique there are in a matrix:

a=as.vector(unique(mir_targ[[1]]))

for (i in 1:length(a)){

mir_targ_sel=mir_targ[mir_targ[[1]]==a[i],]

print(dim(mir_targ_sel)) }

# Print out the unique miRNAs and their genes (straight):
```

```
a=as.vector(unique(mir_targ[[1]]))

for (i in 1:length(a)){

mir_targ_sel=mir_targ[mir_targ[[1]]==a[i],]

print(mir_targ_sel[i,1])

print(mir_targ_sel[,2]) }

# Advanced matching procedures:

# Finding a small group (e_t) true values at big group (e_d), and constructing a new matrix with true and matching values:

matches = (e_d %in% e_t)

normals=exprs(eset_i[,matches])

# own mirna data experssion, usually it is better to use as.matrix for matching (rather than as.data.frame)

e_d=as.matrix(colnames(exprs(eset_i[1,]))) #567, maybe not needed: e_d=e_d[order(e_d)]

# e.g. vijay normal column names

e_n=(as.matrix(colnames(exprs(esetSub[,esetSub$Sampletype=='normal'])))  #87, maybe not necessery: e_n=e_n[order(e_n)]

# Vijay tn

e_t=(as.matrix(colnames(exprs(esetSub[,esetSub$TripleNegative==TRUE])))) # not maybe needed: e_t=e_t[order(e_t)]

# names at e_n do not correspond directly to this scenario (so change the names by e.g. excel, or perhaps with regular expressions):

# write.table(e_n, "e_n.txt"), e_n=read.table("e_n.txt", sep="\t"), e_n=as.matrix(e_n)

# write.table(e_t, "e_t.txt"), e_t=read.table("e_t.txt", sep="\t"), e_t=as.matrix(e_t)

# for normals and tns (FINALLY ok)

matches = (e_d %in% e_t) # => normals=exprs(eset_i[,matches]) or tneg=exprs(eset_i[,matches])

# Combining two matched vectors (using function called "c"):

#First reading the variables:

mimat_names=read.table("mimat_names.txt", header=T,sep="\t")

x=mimat_names[,5]

y1=mimat_names[,2]

y2=mimat_names[,3]

#Here are the matching procedures, and important vectorization (which is needed for c-function):

matches = x %in% y1
```

```
x_m=x[matches]

x_m=as.vector(x_m)

matches = x %in% y2

x_m2=x[matches]

x_m2=as.vector(x_m2)

# This is how you do it (with c()):

x_tot=c(x_m,x_m2)

# Changing mature de_mirna names to standard ones:

de_miro3=de_miro2

de_miro3[,1]=as.vector(de_mat_3_5p) # the names are not in the same order:

de_m_or=result_11[matches,]# ok this seems to be the equivalent (mirBase check), so after ordering / match it is ok

# Finding out the way to print de_miro3 names using (maybe better to use as.matrix again)

de_mn_t=cbind(as.matrix(de_m_or), as.matrix(de_mat_3_5p))

dnt=order(de_mn_t[,1])

de_mn_t=de_mn_t[dnt,]

#(names are ok, then changing de_miro similarly)

dnt2=order(de_miro3[,1])

de_miro3=de_miro3[dnt2,]

de_miro3[,1]=de_mn_t[,2]

de_miro3=as.matrix(de_miro3)

dnt3=order(de_miro3[,2])

de_miro3=de_miro3[dnt3,]

## Matching original data file's (mirna_mature) name info to construct a matching rowname vector
(rownames(targets_ok2)) for another data analysis matrix

# Starting from skimming first name info from original matrix (from miRBASE):

e <- data.frame(mirna_mature[,c(F,F,F,F,T,F,F,T)])

# Sometimes missing values can cause problems (not meaning nas but blank spaces)

# http://stackoverflow.com/questions/18144427/how-to-replace-missing-white-space-with-na-in-r

write.table(e, "e.txt", sep="\t")

e=read.table("e.txt" , sep = "\t" , na.strings = "" ) # blanks are renamed as 'nas' which are more easy to handle
```

result_1 <- data.frame(unlist(e,use.names=FALSE)) # I wanted to combine the two mature miRNA names columns in a big column for easing the matching, and this is how it is done.

# Continuing to second name info using the same original file (mirna_mature)

t_t=data.frame(mirna_mature[,c(F,F,F,F,F,T,F,F,T)]) #notice that different conditions are applied than previously

write.table(t_t, "t_t.txt", sep="\t")

t_t=read.table("t_t.txt" , sep = "\t" , na.strings = "" )

result_2 = data.frame(unlist(t_t,use.names=F)) # Combining columns for normal miRNA names

# I am interested in unique mature and normal miRNA names that may be compared in the analysis matrix

result_11=unique(result_1)

result_22=unique(result_2)

result_33=cbind(result_11,result_22) # Here I combine the unique names' columns

# Now some matching to get the data analysis matrix rownames (targets_o_rn)

result_3m=result_33[(order(result_33[,1])),] # The intersect command (see below) actually (maybe) makes the ordering unnecessary, and possbily the result_ii approach

big=as.matrix(result_3m[,2]) # I am matching now the normal miRNA names, which can be compared later to mature names

small=as.matrix(rownames(targets_ok2))

matches=which (big %in% small)

targets_o_rn=result_3m[matches,1] # I wanted to have the mature names, which are at first column.

# Vijay told me about this intersect command, which I apply here, notice that the targets_o_rn for ok order rownames, is first:

fuu=intersect((as.matrix(targets_o_rn)),rownames(targets_ok2))

# I checked targets_o_rn, and fuu, and I needed to order rownames(targets_ok2) acc. to fuu, using match:

moi=match(fuu,rownames(targets_ok2))

targets_ok2=targets_ok2[moi,]

rownames(targets_ok2)=targets_o_rn # Thank you!

# Just the matching first name (de_miro2, important in intersect command, this is what you want) to another (this is where you would want to see them) with intersect

targets_ok2[istd,]; istd=intersect(de_miro2[,1],rownames(targets_ok2)) # If one need sto check with another place then match is needed

# Checking if the conversion of some data is ok (match maybe used instead of using rownames:

result_3mi[match(rownames(dtt_z), result_3mi[,2]),][66:75,] #dtt_z has been converted

result_3mi[match(rownames(dttv3), result_3mi[,1]),][66:75,] #dttv3 is the original data, ok they are the same

```
### FUNCTIONS AND SCRIPTS ###

# A Basic method how do a function, and then apply it:

# This function calculates how many not nas there are in a row (if value is 0 then all are nas)

f_not_na=function(mat) {

nna=NULL                # Assign the variable before the loop

for (i in 1:dim(mat)[1]) {  # notice that there must be {}s

  nna[i]=dim(mat)[2]-sum(is.na(mat[i,]))   }

  return(nna)    }

targ_dm=f_not_na(trg_dmm) # This is how one applies function

# Taking zero (de_mat_mirna value targets, vector_targets) away from target matrix (mirna all targets at de_mat):

f_not_z=function(vector_targets, de_mat) {

  h=NULL

  for (i in 1:length(vector_targets)) (

    if ( vector_targets[i] != 0) h=c(h,i) )

  return(de_mat[h,])} # This can be also done with: trg_d[f_not_na(trg_d)!=0,]

  #matching de_genes in a matrix to de_mirnas in a list

f_match_down=function(de_geno, demot2) {

  big=as.character(de_geno[,1])

  d=NULL

  tmd=NULL

  matches=NULL

  #smalls

  for (i in 1:length(demot2)) (

  tmd[i]=list(big[(big %in% as.character(unlist(demot2[i])))]))

  return(tmd)  } #jee

#Finding fold change value amounts for targets below zero; f_fold_g

f_fold_g=function (tmd1, gen_rgl) {

h=NULL

for (j in 1:dim(gen_rgl)[1]) (
```

```
for (i in 1:dim(tmd1)[1])
if (gen_rgl[j,1]==tmd1[i,1]) h=c(h,i) )
return(sum(gen_rgl[h,2]<0)) }
## Convert Entrez names (in a list) to gene name function: returns a new list with converted names
f_g_nam_conv=function(gsymb2, demot2_m2) {
  big=as.character(gsymb2[,1])
  big2=as.character(gsymb2[,2])
  tmd=NULL
  #loop match
  for (i in 1:length(demot2_m2)) (
    tmd[i]=list(big2[(big %in% as.character(unlist(demot2_m2[i])))]))
  return(tmd) } #jee }
  # mir_trg_cnv=f_g_nam_conv(gsymb2, demot2_m2) # application of that list
  # Getting normal miR names out of mature names (this function can be applied also to get some conversion of some matrix list, to the other side of that matrix list):
  f_mature_to_mir_vec = function(list) {
  big=as.character(result_3mi[,1])
  tmd=NULL
  #loop match
  for (i in 1:length(list)) (
    tmd[i]=list(big[(big %in% as.character(rownames(dtt_z)[i]))]))
  #return(as.vector(unlist(tmd)))
  return(result_3mi[c(as.vector(unlist(tmd))),2])}
# otherway around
f_mir_to_mature_vec = function(list) {
  big=as.character(result_3mi[,2])
  tmd=NULL
  #loop match
  for (i in 1:length(list)) (
    tmd[i]=list(big[(big %in% (list[i]))]))
```

```
  return(as.vector(result_3mi[match(as.vector(unlist(tmd)),as.character(result_3mi[,2])),1])) # match command was needed

}

# Finding miRs that have at least one target wanted gene (i.e. the ones that have been mentioned in the overlapping best gene list):

f_mir_limo=function(targets_ok3, x) {

  x=as.character(x)

  tmd=NULL #note that targets_ok3 is a list

  #Matching with %in% command

for (i in 1:length(targets_ok3)) {

  if (sum ( (as.character(unlist(targets_ok3[i])) %in% x)==T ) > 0 )

    tmd[i]=rownames(targets_ok)[i] }

return(tmd[!is.na(tmd)]) }

# Make a matrix (mv10746) that has every other row as a negative value of some other matrix (mir_val_X10746)

mv10746=matrix(nrow=1134, ncol=35)

for (i in 1:dim(mv10746)[1]) (

if (i %% 2 !=0) (mv10746[i,]=mir_val_X10746[(1+(i-1)/2),])

else (mv10746[i,]=-mir_val_X10746[(i/2),] ) ) # ok

## Delete values with 'which' function, from a matrix A (to obtain new matrix B).

# Some values, that hold 'TRUE' according to the argument of the 'which' function (which(...)), are discarded.

# For example for rows:

B <- A [ - which (x <= 10) , ]

#constructing the function for hammington distance matrix

f_ham_dis= function (ibt2) {

  #ibt is the gene (and miR) information in list of lists

  #the unique miRs are needed:

  abcd <- vector("list", length(list))

  abcd=NULL

  for (i in 1:150) {

    abcd[[i]]=c(rownames(ibt2[[(i)]][[2]])) }

  abcd=unlist(abcd)
```

```
  abcd=as.matrix(unique(abcd))

  abcd=abcd[(abcd[,1]!="beta_0"),1] # jee

  #the ham matrix constructed

  gen_mir_ham=matrix(0,nrow=length(ibt2),ncol=length(abcd))

  rownames(gen_mir_ham)=names(ibt2)

  colnames(gen_mir_ham)=abcd

  # this below deletes the beta from row names, so this must be inserted instead of below "rownames":

  # as.matrix(rownames(ibt2[[66]][[2]]))[-match("beta_0",rownames(ibt2[[66]][[2]])),1]

  for (i in 1:length(ibt2)) (

    for (j in 1:(length(ibt2[[i]][[2]]))) (

      if (ibt2[[i]][[2]][j,1]>0) (gen_mir_ham[i,(abcd %in%

      as.matrix(rownames(ibt2[[i]][[2]]))[-match("beta_0",rownames(ibt2[[i]][[2]])),1][j])]=1) ) )

  return(gen_mir_ham) }

  ## I need to see all the gene names that I have in the miRNA target list:

f_mir_limo2 = function(targets_ok3, stp3) {

  tmd=NULL # stp must be in vector have same names as the genes (so double)

  tmd=stp3[stp3 %in% as.character(unlist(targets_ok3))]

  return(tmd) }

  #matching gene symbols to my Entrez list (the list have slighlty less values than in idconverter database) (I should use some other method then)

  f_gsymb_ent= function (list) {

  mm=match( as.matrix((list)),as.character(gsymb2[,2]))

  return (gsymb2[mm,1])} #notice that some gene symbols do not have EntrezNames, check also: http://idconverter.bioinfo.cnio.es/

## Removing something e.g. from matrices.

# Set the thing to be removed as F, here e.g. NAs are removed (plus other criteria):

# a_vs=anno$Vital.Status                                                                    # anno is a table for annotation information (defined in 181113-pt-combined-filt-group.txt)

l = (is.na(a_vs))                                                                          # Produces boolean values. Here are the NAs in a_vs variable, which are not allowed!

pat_ok_5 = ((a_ex == T) & ... (a_os >= 1825) & l == F)                    # Produces boolean values. Note the setting of l == F  (F=FALSE, T=TRUE)
```

```
# This can be then applied to e.g. expression set matrix (where only true values are included):

tg_5_e  =          exprs(eset[,pat_ok_5])

# Removing empty spaces "" from matrices:

mapGO <- mapGO[mapGO[,2]!="",]

## USAGE OF FOR LOOPS:

## Sophisticated way of removing something from matrices (e.g. ret, to obtain new matrix (M)) by using for-if loop (h is "in-between-function" matrix):

dim(ret)=c(20531,1100)

h=c()

for(i in 1:20531){if(median(ret[i,]) <= 1){

h=c(h,i)} }

M=ret[-h,]

## It is possible to do A single test (for one gene at one row) multiple times (for all genes at all rows) using for loop (for e.g. 20531 genes i.e. rows):

w=1:20531 * 0                                                    # this zero vector will be filled during the for loop

for(i in 1:20531){h=wilcox.test(texp[i,],nexp[i,]);w[i]=h[[3]];}

# Notice that the third value is relevant, w[[3]]; here h[[3]], and it needs to be saved after every time the test is done for each gene (i, meaning rows)

# Notice also that Change data.frame matrix as numerical by using as.matrix command (needed e.g. in Wilcox test)

# Double for loop and print, notice that j is first

g_piano=1535*0;

for (j in 1:1535)  (

for (i in 1:dim(g2pwyid)[1]) (

if  (g2pwyid[i,1] == grf[j])

#g_piano=cbind(g2pwyid[i,2], grf[j])

print(cbind(g2pwyid[i,2], grf[j]))  ))

# Triple for loop for doing a sum of values  (better not to use else command besides if, because it takes time)

for (j in 1:dim(mimat_names)[1]) (

for (i in 1:dim(dat_frdm)[1]) (

for (e in 1:dim(dat_frdm)[2]) (

if  (as.matrix(rownames(dat_frdm)[i]) == as.matrix(mimat_names[j,5])) (
```

```
sum[j,e] = sum[j,e] + dat_frdm[i,e]  )    )   )  )

# Selecting target genes (miR_t) from de-miRs (im), or

# The values (or entries or rows) of a one matrix (im) is compared to the values (or entries or rows) of other matrix (miR_t) to obtain

# all the different values (in rows) from other matrix (fmiR_t) that correspond similar values (in rows) in the other matrix (im).

# The working double looping procedure in R (finally): h=c()

for (j in 1:12) (

for (i in 1:117193) (

if  (as.matrix(miR_t[[i,1]]) == as.matrix(im[[j,1]])) (h=c(h,i)) ))

# see the result by inserting: h, or:

mir_targ=miR_t[h,]

# One can also do this one by one: h=c()

for (i in 1:dim(mirna_gene)[1]) (

if  ('hsa-let-7d' == mirna_gene[i,1]) (h=c(h,i)) ) # and then: t=mirna_gene[h,]

## Apply function (matrix, rows=1, colums=2, and finally the function)

median_445=apply(realn,2,median)

## Making a function and applying the "hidden" outcome to real scenario (this is M.Tuschens zNorm function):

zNorm <- function (matrix) {

   for(i in 1:dim(matrix)[1]) {

      rowMeans <- mean(matrix[i,]) # calculate the row's mean

      rowSd <- sd(matrix[i,]) # calculate the row's standard deviation

      for(j in 1:dim(matrix)[2]) {

         matrix[i,j] <- (matrix[i,j]-rowMeans)/rowSd } }

   return(matrix)}

## The function result can "disappear", but if you insert this value to another matrix name, then you will get the result,

# Notice here that the function (and its argument, which is a matrix) is on the RIGHT HAND SIDE of the equation:

myMatrix_znorm <- zNorm(myMatrix)

# Make a new matrix, rename the columns, and select some values ...

# from this matrix for a yet new matrix that is sorted out according to these values: use also as.matrix here for cbind objects
```

```
de_mir=cbind(rownames(dat_tn),as.numeric(wtg2))

colnames(de_mir)<-c('miRNA','pval')

# Selecting the best differentially expressed miRNAs:

de_mir_lgi = de_mir[,2] < 0.05

# dif.exp. mir:s that are ok, according to the pval (in second column):

de_miro=de_mir[de_mir_lgi,]

Ord1 = order(de_miro[,2])

de_miro=de_miro[Ord1,]

# The way to IMPORT ALL of the (miRNA) DATA from a directory's files to an R list

# is to use foreign package # http://www.ats.ucla.edu/stat/r/pages/read_multiple.htm  #library(foreign)

# This is the directory:

path_to_files = "C:/cygwin64/home/Pauli/miRNA/Level_3/"

# Reading all the file names from this directory

files = list.files(path_to_files, pattern="13.isoform.quantification",all=F)

# Construct a variable that has the path of directory and the file names to read as a table (or list in this case) in R

fa <- file.path(path_to_files, files)

# Lapply command has several functions such as read.dta or read.csv, check the options from above internet site

da <- lapply(fa, read.table)

# Selecting certain values of this list: da_test=c()

for (i in 1:length(dat))( da_test=qpcR:::cbind.na(da_testi, dat[[i]][6]), dat[[i]][4]) #so far so good...

# Then renaming the matrix

sum_mir_dvl=da_test[,-1]

# Preparing the sum matrix from sum_mir_dvl

# Remove star/mature beginnings from names

# First making the empty matrix: su_mmat=matrix(nrow=7702, ncol=1540) su_mmat = "[<-"(su_mmat,value=0)

# This loop is done three times (so i times and then repeated 3 times), where the pattern is first mature, and then star

for (i in 1:770)(#su_mmat[,(i*2)]=gsub(pattern="star,", replacement="", su_mmat[,(i*2)], ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

su_mmat[,((i*2)-1)] = sum_mir_dvl[,((i*2)-1)])

#Changing the colnames of the new matrix:
```

```
colnames(su_mmat)=colnames(sum_mir_dvl)

# This is the long wanted sum matrix for mirnas with corresponding mature names and their sum expression values:

# (note: agstest=matrix(nrow=1200, ncol=1540) agstest = "[<-"(agstest,value=0) =

for (i in 1:770)  (agstest=qpcR:::cbind.na(agstest,aggregate(su_mmat[,(i*2-1)], list(su_mmat[,(i*2)]), sum)) )

# Defining the number of rows and their names in agstest matrix according to unique MIMAT-names in agstest in overall:

# http://stackoverflow.com/questions/12154814/r-get-a-single-column-from-many-columns

#e <- agstest[,c()]

a <- list(atf)

result <- data.frame(Wave = unlist(a,use.names=FALSE))

agstest_rn=unique(result) ## jee

# rownames should not have na-values:

rownames(agstest)=agstest_rn[is.na(agstest_rn)==F,1]

# Then one needs to merge the correct rownames (of agstest matrix) and agstest-mimat names (at every second column) to obtain the overall matrix dtt:

#defining the new variable before the looping solution:

dtt=matrix(nrow=1048,ncol=1540)

dtt = "[<-"(dtt,value=0) #from art of  r programming

dtt=data.frame(dtt)

# The use of merging function was found at (see below), and then applied for every column:

# http://r.789695.n4.nabble.com/How-to-compare-match-two-columns-from-diferent-dataframe-and-assign-values-from-one-datafram-to-the-r-td2544573.html

for (i in 1:770) (

dtt[,((i*2-1):(i*2))]=merge(data.frame(x=rownames(agstest)),

                    data.frame(x=agstest[,(i*2-1)],y=agstest[,(i*2)]),by='x',all.x=T) )

# Because dtt was an empty matrix the rownames, and column names must be inserted again, and also removing not needed extra-mimat-names, that where

# needed for the merging

dtt=dtt[,(rep(c(FALSE,TRUE),770))]

rownames(dtt)=rownames(agstest)

# Some repetition from previous stages is good:

path_to_files = "C:/cygwin64/home/Pauli/miRNA/Level_3/"
```

```
files = list.files(path_to_files, pattern="13.isoform.quantification",all=F)

colnames(dtt) <- substr(files,1,15)

# Select 6 random numbers between 1 and 40, without replacement: http://www.inside-r.org/howto/how-generate-random-number-r

x5 <- sample(1:40, 6, replace=F)

## If you want to return multiple things in the list then add them as list in the end of function (for the return command):

return(list(result_f, limo_mir2,en_mir, cor))

# Shuffling the patient names 20 times and doing a linear modelling for them:

# Shuffling of the patient names:

 f_shuffle_rename = function (e11_zs, n_times) {

  c_nam_e11_zs=vector("list", n_times)

  #sample=vector("list", n_times)

  for (i in 1:n_times) (

   #sample[[i]]=sample(1:567, 567, replace=F)

   c_nam_e11_zs[[i]]=colnames(e11_zs)[sample(1:567, 567, replace=F)]

  )

  return(c_nam_e11_zs)

 }

 suffles=f_shuffle_rename(e11_zs, 20)

 #Let's go for looping this for linear modelling of all the genes (in the list)

 f_limos_rnd = function (data_zs, e11_zs, list, suffles) {

  tot=vector("list", length(suffles))

  for (i in 1:length(suffles)) (

   tot[[i]]=f_limo_all(data_zs,e11_zs[,suffles[[i]]],list)

  )

  return(tot)

 }

 shuffling_comp=f_limos_rnd(data_zs, e11_zs, list, suffles)

 #Some useful information extracted from this suffling:

 f_cor_suf = function (shuffling_comp,list) {
```

```
    cor_suf=vector("list",150)
    for (i in 1:length(shuffling_comp)) (
      for (j in 1:length(list)) (
        cor_suf[[j]][[i]]=shuffling_comp[[i]][[j]][[4]]
      ))
    names(cor_suf)=names(shuffling_comp[[1]])
    cor_mean=vector("list",150)
    cor_mean=lapply(cor_suf,mean)
    cor_mean2=as.matrix(cor_mean)
    write.table(cor_mean2, "cor_mean_alp.txt", sep="\t")
    return(cor_mean2) }
  fcs1=f_cor_suf(shuffling_comp,list)
# Dinding targets of miRNAs (notice how one can access list elements just by numeric indexes introduced in cond):
f_miR_targs= function (list_mir) {
  aus=vector("list", length=length(tush_nl))
  names(aus)=names(tush_nl)
  cond=c(1:length(tush_nl))[is.na(match(names(tush_nl),list_mir)==T)==F]
  aus=tush_nl[cond]
  return(aus)
}

# REGULAR EXPRESSIONS
## Selecting certain names of row or column entries (i.e. values) (finally using regular expressions)
m <- grep("MIMAT*", mir_acc[,2], value=FALSE)
mir_acc_ok=mir_acc[m,]
## Replace a certain value from the rows
a=sub("(r)", "\\U\\1", im[,1], perl=TRUE)
# Selecting certain datas of column
m <- grep("MIMAT*", data_m2, value=FALSE)
dat_frdm=data_tot[m,]
```

#changing row names to contain only mimats: let's try regejÃ¤: notice that two replacements are made:

rn_d_f=gsub(pattern="mature,", replacement="", rownames(dat_frdm), ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

rn_d_f2=gsub(pattern="star,", replacement="", rn_d_f, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

rownames(dat_frdm)=rn_d_f2

# Giving a matrices (or vectors) values as logical according to certain pattern (such as MIMAT)

grepl("*MIMAT*", sum_mir_dvl[,2]) #logical grep

# Replacements:

x <- "hello.world"

y <- gsub(".","-",x,fixed=T)

# Subtracting certain row of elements (substring) from a string/vector etc., here used to truncate column names for matching:

colnames(exprs(esetSub))=substring(colnames(exprs(esetSub)),1,15) # See: (The Art of R programming, p.253)

### TESTS ###

# remember to filter the genesymbol tables at the same time as the gene expression table (also expression set)!!

gsymb <- read.table(file="genesymbols.txt", header = TRUE, row.names=NULL, sep="\t") # notice: row.names=NULL

# T-test:

ttestCutoff = 0.05

ttests = rowttests(M_filt_ok, as.factor(pat_nrm))  #This worked!!!, notice as.factor!!

smPV = ttests$p.value < ttestCutoff

# Check the filtered genes from genesymol list:

G_dif=G_filt_ok[smPV,]

# Apply package conversion: https://stat.ethz.ch/pipermail/bioconductor/2010-March/032455.html

get(ID, org.Sc.sgdGO)

# normalisation by vsn:

testia=justvsn(as.matrix(dtt2c)) #notice AS.MATRIX...

#Linear modelling function for one gene:

f_limo_mir_gene_ii=function (data_zs, e11_zs, gene_nam, mirs) {

```
nro1  = dim(data_zs)[2]

nro2  = length(mirs)

nro3=(nro2+1)

lhs   = (nro1+nro2+1)

rhs   = (nro1*2)


# Left hand side of the equations

A_mat_f = matrix(0,nrow=rhs, ncol=lhs, byrow=T)


# making of vector 1)

v_i=matrix(1, nrow=nro1)

v_e = as.vector(rbind(v_i,-v_i))

#miRNA list (check: sortmiro)

mir_val=t(data_zs[(rownames(data_zs) %in% mirs),])

# Make a matrix (mv10746) that has every other row as a negative value of some other matrix (mir_val_X10746)

mir_neg_pos=matrix(nrow=rhs, ncol=nro2)

for (i in 1:rhs) (

 mir_neg_pos[i,] = ifelse ( (i %% 2 !=0), mir_val[(1+(i-1)/2)], -mir_val[(i/2)]) ) # ok

if (sum(is.na(mir_neg_pos)!=F)>0) (mir_neg_pos[is.na(mir_neg_pos)] <- 0)


## The zero list (b):

o=-rbind(1,1) # this is the double. And before the loop an empty list is created:

Lst=c()

for(i in 1:nro1)(Lst[i]=list(o)) # The "a" needs to be in a list

hmm=data.matrix(bdiag(Lst)) # this is how you use your list to construct this matrix.


#column 1

A_mat_f[,1]=v_e

# columns 2:(nro2+1)

A_mat_f[,2:(nro2+1)]=mir_neg_pos
```

```
A_mat_f[,(nro2+2):lhs]=hmm #see comments about the type of matrix above


# Making the model list values:

model_f <- list()

# Making of the objective function

obj_f <- vector(mode = "numeric", length = lhs)

obj_f = "[<-"(obj_f,1:lhs,value=1)

obj_f = "[<-"(obj_f,1:nro3,value=0)


model_f$obj = obj_f

#obj_f_cons = vector(mode = "numeric", length = (nro2+1))

#model_f$objcon = obj_f_cons

model_f$modelsense <- "min"

model_f$sense      <- rep('<=', rhs)

model_f$vtype      <- 'C'

# Making the Right side of equations:

model_f$rhs        <- as.vector(rbind((e11_zs[gene_nam,]),(-e11_zs[gene_nam,])))


# When you got A_mat_f ok, then this works (if row and column numbers are ok)

model_f$A          <- A_mat_f

params_f <- list(Method=2, ResultFile='model_f.mps')

result_f <- gurobi(model_f,params_f)


#more info

limo_mir=cbind(result_f$x[1:(nro2+1)])

colnames(limo_mir)=c('lin.mod.coefficient')

rownames(limo_mir)=c("beta_0",mirs)


# Ordering the miRs according to coefficient

limo_mir2=limo_mir
```

```
  limo_mir2=as.matrix(limo_mir[rev(order(limo_mir)),])
  colnames(limo_mir2)=c('lin.mod.coefficient')
  # Finding overlapping ones to the all miR list
  en_mir=as.matrix(limo_mir2[rownames(limo_mir2) %in% names(demot2_m2),])
  colnames(en_mir)=c('lin.mod.coefficient')

  #Multiplication:
  b_m_i_f=data_zs[(rownames(data_zs) %in% rownames(limo_mir)),]

  # matrix (or vector) multiplication is needed :)
  g_pred_f_i=(b_m_i_f*as.numeric(limo_mir[2:(nro2+1)])) #[1:13,1:4] # HUT was needed, this worked!!
  g_pred_fo=c(1:nro1)
  g_pred_fo = "[<-"(g_pred_fo,1:nro1,value=result_f$x[1])
  g_pred_f=rbind(g_pred_fo,g_pred_f_i)
  if (sum(is.na(g_pred_f)!=F)>0) (g_pred_f[is.na(g_pred_f)] <- 0)

  # but the equations continue...:
  g_pred_val_f=NULL
  for (j in 1:dim(g_pred_f)[2])
    (g_pred_val_f[j]=sum(g_pred_f[,j]))

  # And the correlation
  y=as.numeric(e11_zs[gene_nam,])
  x=as.numeric(g_pred_val_f)
  cor=cor(y, x, method = "pearson")

  return(list(result_f, limo_mir2,en_mir, cor))
}
# constructing the function for hammington distance matrix
f_ham_dis= function (ibt2) {
```

```
  #ibt is the gene (and miR) information in list of lists
  abcd <- vector("list", length(ibt2))
  abcd=NULL
  for (i in 1:length(ibt2)) {
    abcd[[i]]=c(rownames(ibt2[[(i)]][[2]])) }
  abcd=unlist(abcd)
  abcd=as.matrix(unique(abcd))
  abcd=abcd[(abcd[,1]!="beta_0"),1] # jee

  #the ham matrix constructed
  gen_mir_ham=matrix(0,nrow=length(ibt2),ncol=length(abcd))
  rownames(gen_mir_ham)=names(ibt2)
  colnames(gen_mir_ham)=abcd

for (i in 1:length(ibt2)) (
for (j in 1:(length(ibt2[[i]][[2]]))) (
if (ibt2[[i]][[2]][j,1]>0) (gen_mir_ham[i,(abcd %in%
as.matrix(rownames(ibt2[[i]][[2]]))[-match("beta_0",rownames(ibt2[[i]][[2]])),1][j])]=1) ) )
return(gen_mir_ham) }
# The distanced maybe thus calculated:
f_calc_dist = function (gen_mir_ham2) {
n <- nrow(gen_mir_ham2)
m <- matrix(nrow=n, ncol=n)
rownames(m)=rownames(gen_mir_ham2)
colnames(m)=rownames(gen_mir_ham2)
for(i in seq_len(n - 1))
  for(j in seq(i, n))
    m[j, i] <- m[i, j] <- sum(gen_mir_ham2[i,] != gen_mir_ham2[j,])
return(m)
}
```

```
mo=f_calc_dist(gen_mir_ham2)

## Clusterings of Hammington distances of top 150 miRNA overlapping genes correlations:

pdf('clust_all.pdf')

method="complete"

distance="euclidean"

test_clust=hclust(dist(mo, method=distance),method=method)

par(ps=3)

plot(test_clust,par) # you may also save as such in the Rstudio as pdf

dev.off()

# Example function, the replacement goes ok if you cahnge the name of you values before the loop:

f_mir_limox = function(targets_ok3, x) {

  x=as.character(x)

  targets_ok=targets_ok3

  tmd=NULL #note that targets_ok3 is a list

  #Matching with %in% command

  for (i in 1:length(targets_ok)) {

    if (sum ( (as.character(unlist(targets_ok[i])) %in% x)==T ) > 0 )

      tmd[i]=names(targets_ok)[i] }

  return(tmd[!is.na(tmd)]) }

test_celf2_mir3=f_mir_limox(tush_nl,x) # This gives the right values

# Information for cluster groupings:

# Ward Hierarchical Clustering was found to be good one for clustering:

d <- dist(moi2, method = "euclidean") # distance matrix

rownames(moi2)

fit <- hclust(d, method="ward")

par(ps=3)

plot(fit) # display dendogram

groups <- cutree(fit, k=9) # cut tree into 9 clusters

# draw dendogram with red borders around the 9 clusters

rect.hclust(fit, k=9, border="red")
```

```
#Group selection from the image is a delicate process:

str(groups)

group1=groups[(groups==1)] # ok

(groups)["BACH2"] ##in group 4

...

(groups)["RC3H1"] #in group 9

group8=groups[(groups==7)]

#Better to have 8 tables where extract information:

grp_tot_nam2=list(names(group1),
names(group2),names(group3),names(group4),names(group5),names(group6),names(group7),names(group8))

names(grp_tot_nam2)=c("group1", "group2","group3", "group4","group5","group6","group7","group8")

lapply(names(grp_tot_nam2),function(x, grp_tot_nam2)

write.table(grp_tot_nam2[[x]], paste(x, ".txt", sep = ""),col.names=FALSE, row.names=FALSE, sep="\t",
quote=FALSE),grp_tot_nam2)

#It is maybe better to do this 8 times for every group the same info:

path_to_files = "C:/Users/Pauli/Documents/thesis/labour/analysis/TCGA/results/cluster_150_enr/"

g_inf=vector("list",8)

files=vector("list",8)

gi=vector("list",8)

pattern=vector("list",8)

# Somehow one loop solution did not work:

for (i in 1:8) (

  pattern[[i]]=paste("_g",i, sep="") )

for (i in 1:8) (

files[[i]] = list.files(path_to_files, pattern[[i]],all=F) )

for (i in 1:8) (

gi[[i]] <- file.path(path_to_files, files[[i]]) )

for (i in 1:8) (

g_inf[[i]] <- lapply(gi[[i]], sep="\t", quote = "", row.names = NULL,stringsAsFactors = FALSE,read.csv) )

names(g_inf)=c("group1", "group2","group3", "group4","group5","group6","group7","group8")

#Selecting certain things for observations:
```

```
# Find out which list factor is which:

g_inf[[1]][[1]][1:4, 1:7] ## cancer cell en

g_inf[[1]][[2]][1:4, 1:7] ## go_bp

g_inf[[1]][[3]][1:4, 1:7] ## go_mol_func

g_inf[[1]][[4]][1:4, 1:7] ## kegg

g_inf[[1]][[5]][1:4, 1:7] ## mir

#Now changing the last module you may see all of the values that you want:

f_wanted_gr= function (g_inf) {

abc=NULL

for (i in 1:8)

  (abc[[i]]=g_inf[[i]][[4]][1:5, c(1,2,4)])

   return(abc) }

sel_grp=f_wanted_gr(g_inf)

names(sel_grp)=names(g_inf)

#Values of confusion matrix extracted

f_enr_mir_cmat= function (t_mru_d,hero3,gen_ok_f,gen_rgl,c1) {

  t=vector("list",length(t_mru_d))

  names(t)=f_mature_to_mir_vec(names(hero3))

  a=NULL

  o=NULL

for(i in 1:length(t_mru_d)) (

  t[[i]] =list(t_mru_d[i], as.vector(sapply(hero3[i],length))-t_mru_d[i],

            length(gen_ok_f)-t_mru_d[i],dim(gen_rgl)[1]-(length(gen_ok_f)-t_mru_d[i])) )

a=do.call(rbind,t)

o=match(as.vector(as.vector(c1[,1])),rownames(a))

a=a[o,]

colnames(a)=c("A","B","C","D")

write.table(a,"enr_mir_rgl-uod.txt", row.names=F, sep="\t")

return(a)}

tush_new_miru_cm=f_enr_mir_cmat(t_mru_d,hero3,gen_ok_f,gen_rgl,c1) #ok
```

```
### PACKAGES ###

#I your compiling of pdf does not work in R, try changing the format to knitR (for which to work you need LaTex: https://www.tug.org/protext/)

Sweave2knitr("knitR_test.Rnw")


### LIBRARIES ###

# LOADING

library(parallel)

library(BiocGenerics)

library("Biobase")

library(affy)

library("gplots")

library("ALL")

library("genefilter")

library(heatmap.plus)

library('pheatmap')

library(AnnotationDbi)

library(annotate)

library(graph)

library(GSEABase)

library(pvclust)

data("ALL")

library(cluster)

library(limma)

library(marray)

library(convert)

library(DBI)

library(org.Sc.sgd.db)

library("yeast2.db")

library("gcrma")
```

```
library(preprocessCore)
library("plier")
library("affyPLM")
library("gtools")
library("plotrix")
library("biomaRt")
library('foreign')
library('lattice')
library('Matrix')
library('nlme')
library('mgcv')
library(piano)
library('exactci')
library('exact2x2')
library("AnnotationDbi")
library( IRanges)
library( XVector)
library( GenomicRanges)
library( GenomicFeatures)
library(TxDb.Mmusculus.UCSC.mm10.ensGene)
library( Biostrings)
library( Rsamtools)
library( VariantAnnotation)
library( ensemblVEP)
library( org.Hs.eg.db)
library( hgu95av2.db)
library( GO.db)
library( TxDb.Hsapiens.UCSC.hg19.knownGene)
library( OrganismDbi)
library( Homo.sapiens)
```

```
library( AnnotationHub)
library( TxDb.Athaliana.BioMart.plantsmart16)
library("annotation")
library("globaltest")
library("KEGG.db")
library("SparseM")
library("topGO")
data(geneList)
library("slam")
library("gurobi")
library(plyr)
library(gdata)
library(MASS)
library(minpack.lm)
library(rgl)
library(robustbase)
library('qpcR')
library('stats')
library(data.table)
library(qusage)
library(hash)
library("vsn")
library(rattle) # Weather dataset
library(ggplot2) # Beautiful plots
library(xtable) # LaTeX tables
library(grid)
library(lattice)
library(splines)
library(survival)
library(Formula)
```

```
library(Hmisc) # LaTeX string preparation

library(shape)

library(diagram) # Flowchart

library('knitr')

library('impute')

library('WGCNA')

library(rattle)

data('weather')

library('dynamicTreeCut')

library('flashClust')

biocLite('amsthm')

library('reshape')

library("KEGGgraph")

library("XML")

library("pathview")

library(sparcl)



library(

# SOURCE FOR SOME INSTALLATIONS

source("http://bioconductor.org/biocLite.R")

source("http://bioconductor.org/workflows.R")

biocLite()

# INSTALLATION

#(should be installed in workspace) (sometimes install from directory, e.g. gurobi.zip was done so)

install.packages('gplots')

install.packages('ALL')

biocLite(c("ALL"))

biocLite(c("genefilter"))

biocLite(c("affy"))
```

```
install.packages('heatmap.plus')

install.packages('pheatmap')

biocLite("GSEABase")

install.packages("pvclust")

install.packages('cluster')

biocLite("convert")

biocLite("igraph")

biocLite("piano")

pkgs <- c("yeast2.db", "limma", "plier", "affyPLM", "gtools", "plotrix")

biocLite(pkgs)

biocLite("biomaRt")

install.packages('exact2x2')

workflowInstall("annotation")

biocLite("AnnotationDbi")

biocLite("globaltest")

biocLite("KEGG.db")

biocLite("SparseM")

biocLite("topGO")

install.packages("C:/gurobi560/win64/R/gurobi_5.6-0.zip", repos = NULL)

install.packages("plyr")

biocLite("ensemblVEP")

install.packages('foreign')

install.packages('lattice')

install.packages('Matrix')

install.packages('nlme')

install.packages('mgcv')

install.packages('qpcR')

install.packages('stats')

install.packages('data.table')

biocLite("qusage")
```

```
install.packages('hash')

biocLite("vsn")

install.packages('rattle')

install.packages('ggplot2')

install.packages('xtable')

install.packages('Hmisc')

install.packages('diagram')

install.packages('knitr')

install.packages('WGCNA')

biocLite('impute')

install.packages('dynamicTreeCut')

install.packages('flashClust')

biocLite('GenomicFeatures')

biocLite('VariantAnnotation')

biocLite('OrganismDbi')

isntall.packages('reshape')

biocLite("pathview")

install.packages('sparcl')

biocLite(

install.packages(

# UPDATING (simetimes needed)

update.packages(repos=biocinstallRepos(), ask=FALSE)


### GENERAL LEARNING ###

## IF A COLLAGUE OR A FRIEND SUGGEST YOU SOME CODE OR PROCEDURE (IN PLANNING) THEN DO PRECISELY SO.

## IF YOU MADE MISTAKES WITHIN THAT SUGGESTED CODE, AND AFTER CHECKING BY YOURSELF USING COMMON SENSE, YOU MAY ASK WHAT IS WRONG WITH YOUR VERSION (OF YOUR FRIEND'S CODE), DIRECTLY.

## FINALLY YOU MAY TRY TO IMPROVISE AFTER THESE TWO STEPS.

## IN FILTERING: NO PATIENTS ARE DISCARDED, JUST GENES!!

## Using R studio is very helpful in R programming.

## The loading of about more than 1 gb workspace image to R takes time.
```

## If possible hold only those variables that you need for one R session.

## Remember to change the new variables inside the function, when applying after previous change!!

## Remember to save the workimage twice a day!

## Keep the same matrix rownames and column names all the time constant when you are doing your analysis (rather than changing them), because it is easier to convert

#  them in the form you want to present them later, than doing multiple matching procedures between the analysis.

## If you seem to be blogged, check NA/0/missing values from your list/matrix etc.

## Try to put rownames (or names) for matrices (or lists) every time (it is easier to access them with these).

## Keep everything in one programming testing file within project (many files will confuse).

## Make comments and INSERT ALL THE CODE THAT YOU EXECUTE (because afterwards if you need to reproduce it you may save some time).

## To save space save the workspace ONLY once (without name) so the new will overwrite the old. Sometimes do backups of the overall workspaces.

## First test your function or script with small values/data before applying to it to bigger case. If there are any error messages after/during run, check:

## 1) Syntax, 2) All the individual elements (do they work as such without loops or functions), 3) Check the error message from google,

## 4) If 1-3 does not help, ask a friend or try to do your function in a different way, or think the problem in a new way: check also google-cases/books etc.

## 5) Also evaluate if the thing(s) that you are doing at the moment is(are) really important/something that you really need to do. Ask yourself &/ supervisor(s).

## R can recognize the "" values (e.g. "5687") as numbers, when doing matching (limo_mir_X10746_i[(limo_mir_X10746_i[,2]>0),]), but in numerical

## cases change the values as.numeric.

## See that all the variables are defined correctly from the beginning, and not uncommented.

## If a smallish scripting line/task is taking more than 1,5h to perform then you need to change the way your are handling the matter,

## such as in the case of shuffling the patient names and doing a big list from the result you should just apply shuffling in linear model as such.

## Sometimes the R studio has performed its big operations, but is still giving "in progress stop sign", ignore/stop it, if it last more than anticipated, and see the result.

## When you are inserting values for the function make sure that the group selections are actually true, i.e. do not write small letters if the groups are written in small.

## Be aware of line separation symbols (\n) when copy-pasting from win-to-unix (or something like this).

## If you do modifications to functions remember to save those functions before running them (you possibly need to this saving also for functions that uses that function).

## The function value replacement goes ok if you change the name of you values before the loop

## Load the secondary workspace with some name (even if it is completely same as the original), and when you save the primary workspace DO NOT use any name. This will

## avoid the system crash ( I think) especially if you have some work pending on the secondary workspace when you are saving the first, and then opening this saved workspace.

## Do not destroy any processing code while the job is still on the way...