```
# 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...