Analysing the first set of TMT-tagged data for Trizol-based UV dosage experiments

```
#-------------------------------------------------------------------------
# Author          : Manasa Ramakrishna, mr325@le.ac.uk
# Date started  : 4th August, 2017
# Last modified : 4th August, 2017
# Aim             : To take a look at first SILAC labelled LOPIT data on Trizol
# Depends       : On 'silacFunctions.R'. Make sure they are in the same directory
# Notes         : Works on data from Rayner's first experiments
#-------------------------------------------------------------------------


# Invoking libraries
library(MSnbase)
library(gplots)
library(reshape2)
library(VIM)
library(zoo)
library(spatstat) # "im" function
library(ggbiplot)
library(org.Hs.eg.db)
library(clusterProfiler)
library("biomaRt")
library(goseq)
library(limma)
library(ggplot2)

library(outliers)
library(RColorBrewer)
library(stringr)

#Setting working directories
wd = "/Users/manasa/Documents/Work/TTT/02_Proteomics/02_First-TMT-data/"
setwd(wd)
getwd()
```

```
## [1] "/Users/manasa/Documents/Work/TTT/02_Proteomics/02_First-TMT-data"
```

```
indir = paste(wd,"Input",sep="/")
outdir = paste(wd,paste(Sys.Date(),"Output",sep = "_"),sep = "/")

if (exists(outdir)){
  print("Outdir exists")
}else{
  dir.create(outdir)
}

# Sourcing function file
source("tmtFunctions.R")
```

Now that we have loaded all the packages we need for working with this data, let's move on to the data.

```
# ------------------------------------------------
# Step 00: Read data
# Read in all the data required for analysis
# ------------------------------------------------

# File of contaminants - proteins to exclude from analysis as are things like keratin, alcohol dehydrog
contam = read.delim("Input/Common contaminant_all.csv",sep=",",header=T)

# Read in the sample file that matches columns to sample contents
samp.dat = read.delim("Input/samples.txt",sep="\t",header=T)

# Read in the data files that contain peptide level output from Proteome discoverer...
# Note: The columns that begin with "Found.in.Sample.in" correspond to various samples in the study.
# Columns of interest are "sequence", "modifications","master.protein.accessions","abundance","quan.inf
data = read.delim("Input/Dosages_4step-Trizol_PeptideGroups.txt",sep="\t",comment.char="",as.is=T,heade

# Subset data to only keep columns of interest
prot.data = data[,c(1:6,10:14,17,42:51,62)]

# Rename tmt tagged columns with UV dosage names
for(i in 1:nrow(samp.dat)){
  id = grep(samp.dat$TMT[i],colnames(prot.data))
  colnames(prot.data)[id] = paste(samp.dat$Sample[i])
}
dim(prot.data)
```

```
## [1] 14456    23
```

```
head(prot.data)
```

```
##   Checked Confidence PSM.Ambiguity          Sequence
## 1   False       High   Unambiguous GSGGLGGACGGAGFGSR
## 2   False       High   Unambiguous    QGSGSGQSPGHGQR
## 3   False       High   Unambiguous         QQQQMEQER
## 4   False       High      Selected          LAHCEELR
## 5   False       High   Unambiguous        RGMDDDRGPR
## 6   False       High   Unambiguous        SEDVWLEAAR
##                                 Modifications
## 1 1xTMT6plex [N-Term]; 1xCarbamidomethyl [C9]
## 2                          1xTMT6plex [N-Term]
## 3                          1xTMT6plex [N-Term]
## 4 1xTMT6plex [N-Term]; 1xCarbamidomethyl [C4]
## 5                          1xTMT6plex [N-Term]
## 6                          1xTMT6plex [N-Term]
##              Modifications.all.possible.sites Number.of.Protein.Groups
## 1 1xTMT6plex [N-Term]; 1xCarbamidomethyl [C9]                        2
## 2                          1xTMT6plex [N-Term]                       1
## 3                          1xTMT6plex [N-Term]                       1
## 4 1xTMT6plex [N-Term]; 1xCarbamidomethyl [C4]                        1
## 5                          1xTMT6plex [N-Term]                       1
## 6                          1xTMT6plex [N-Term]                       1
##   Number.of.Proteins Number.of.PSMs Master.Protein.Accessions
## 1                  3              3           P04259; P02538
## 2                  1              9                   Q86YZ3
```

```
## 3                  2              6                    Q15149
## 4                  2              1                 A0A087WUT6
## 5                  1              3                    Q14152
## 6                  1              3                    O94906
##       Protein.Accessions Sequence.Length 150mJ.1 150mJ.2  150mJ.3  275mJ.1
## 1 P04259; P02538; P48668              17    25.8     33.1      7.5    34.1
## 2                 Q86YZ3              14     3.3      2.1       NA      NA
## 3         Q15149; H0YDN1               9    15.7     18.6      6.5    74.1
## 4     O60841; A0A087WUT6               8    36.6     68.6      4.5    24.3
## 5                 Q14152              10    61.7     61.2      6.8    29.7
## 6                 O94906              10    44.5     46.5      3.6    34.0
##   275mJ.2 275mJ.3 400mJ.1 400mJ.2 400mJ.3 Pool.1 Quan.Info
## 1    39.6    24.3    46.3    22.2    43.7   50.4 NotUnique
## 2     6.5      NA     1.8      NA      NA     NA    Unique
## 3    37.8    33.8   123.5    37.1    96.3   29.5    Unique
## 4    56.5    35.5    46.3    20.8    52.9   22.2    Unique
## 5    60.5    27.9    77.2    32.3    69.7   29.9    Unique
## 6    54.0    28.0    60.7    34.6    69.6   38.1    Unique
```

data has 23 columns and 14456 rows - each row belonging to a peptide abundance value across all 10 samples. We now go through a series of filtering steps to obtain a dataset we can use for downstream analyses.

```
# ------------------------------------------------------------------------------
# Step 01 : Filter
# We perform 3 layers of filtering - unique proteins, contaminants,missing values
# ------------------------------------------------------------------------------

# Step 1a : Filter only for those peptides that have a unique master protein. Done using column "quan.i
peptide.stats = table(prot.data$Quan.Info)
peptide.stats
```

```
##
## NoQuanValues    NotUnique       Unique
##         1440          715        12301
```

```
filt.1a = prot.data[which(prot.data$Quan.Info == "Unique"),]
dim(filt.1a) #12301 are unique peptides, 715 are non-unique and 1440 are missing values
```

```
## [1] 12301    23
```

```
# Step 1b : Filter out those proteins that are contaminants from the contaminants list and annotate mis
filt.1b = filt.1a[-which(filt.1a$Master.Protein.Accessions %in% contam$Protein.Group.Accessions),]
num.contams = length(which(filt.1a$Master.Protein.Accessions %in% contam$Protein.Group.Accessions))

dim(filt.1a) # 12301 in total
```

```
## [1] 12301    23
```

```
dim(filt.1b) # 12077 filtered proteins
```

```
## [1] 12077    23
```

```
print(num.contams) # 224 contaminant proteins
```

```
## [1] 224
```

```
# Adding extra information about rows with missing values
filt.1b$count.missing = rowSums(is.na(filt.1b[,c(13:22)]))
```

```
filt.1b$Missing = FALSE
filt.1b$Missing[which(filt.1b$count.missing > 0)] = TRUE

head(filt.1b)
```

```
##   Checked Confidence PSM.Ambiguity       Sequence
## 2   False       High   Unambiguous QGSGSGQSPGHGQR
## 3   False       High   Unambiguous      QQQQMEQER
## 4   False       High      Selected       LAHCEELR
## 5   False       High   Unambiguous     RGMDDDRGPR
## 6   False       High   Unambiguous     SEDVWLEAAR
## 7   False       High   Unambiguous   SSSSQHPEQTGR
##                                 Modifications
## 2                          1xTMT6plex [N-Term]
## 3                          1xTMT6plex [N-Term]
## 4 1xTMT6plex [N-Term]; 1xCarbamidomethyl [C4]
## 5                          1xTMT6plex [N-Term]
## 6                          1xTMT6plex [N-Term]
## 7                          1xTMT6plex [N-Term]
##              Modifications.all.possible.sites Number.of.Protein.Groups
## 2                          1xTMT6plex [N-Term]                        1
## 3                          1xTMT6plex [N-Term]                        1
## 4 1xTMT6plex [N-Term]; 1xCarbamidomethyl [C4]                        1
## 5                          1xTMT6plex [N-Term]                        1
## 6                          1xTMT6plex [N-Term]                        1
## 7                          1xTMT6plex [N-Term]                        1
##   Number.of.Proteins Number.of.PSMs Master.Protein.Accessions
## 2                  1              9                    Q86YZ3
## 3                  2              6                    Q15149
## 4                  2              1                A0A087WUT6
## 5                  1              3                    Q14152
## 6                  1              3                    O94906
## 7                  1              3                    Q5JSZ5
##   Protein.Accessions Sequence.Length 150mJ.1 150mJ.2  150mJ.3  275mJ.1
## 2             Q86YZ3               14     3.3      2.1       NA       NA
## 3     Q15149; H0YDN1                9    15.7     18.6      6.5    74.1
## 4 O60841; A0A087WUT6                8    36.6     68.6      4.5    24.3
## 5             Q14152               10    61.7     61.2      6.8    29.7
## 6             O94906               10    44.5     46.5      3.6    34.0
## 7             Q5JSZ5               12     1.4      1.4       NA     1.6
##   275mJ.2 275mJ.3 400mJ.1 400mJ.2 400mJ.3 Pool.1 Quan.Info count.missing
## 2     6.5      NA     1.8      NA      NA     NA    Unique             6
## 3    37.8    33.8   123.5    37.1    96.3   29.5    Unique             0
## 4    56.5    35.5    46.3    20.8    52.9   22.2    Unique             0
## 5    60.5    27.9    77.2    32.3    69.7   29.9    Unique             0
## 6    54.0    28.0    60.7    34.6    69.6   38.1    Unique             0
## 7     3.4     1.9     2.3      NA     2.5     NA    Unique             3
##   Missing
## 2    TRUE
## 3   FALSE
## 4   FALSE
## 5   FALSE
## 6   FALSE
## 7    TRUE
```

We have a column called "Missing" to identify which peptides have one or more missing values across the 10 samples. "count.missing" tells us how many missing values there are for that peptide.

```
# ------------------------------------------------------------------------------
# Step 02a : Creating an MSnSet which is needed for using the MSnbase backage
# ------------------------------------------------------------------------------

# The rownames of samp.dat have to be the same as column names in the expression data matrix
rownames(samp.dat) = samp.dat$Sample

# Create an MSnSet object
res <- MSnSet(exprs = as.matrix(filt.1b[,c(13:22)]),fData=filt.1b[,c(10,1:9,12,23:25)],pData = samp.dat
res <- res[rowSums(is.na(exprs(res)))!=10,] # exclude  peptides without any quantification
print(res)
```

```
## MSnSet (storageMode: lockedEnvironment)
## assayData: 12077 features, 10 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 150mJ.1 150mJ.2 ... Pool.1 (10 total)
##   varLabels: TMT Sample
##   varMetadata: labelDescription
## featureData
##   featureNames: 2 3 ... 14455 (12077 total)
##   fvarLabels: Master.Protein.Accessions Checked ... Missing (14
##     total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation:
## - - - Processing information - - -
## Subset [12077,10][12077,10] Wed Sep 13 15:36:12 2017
##  MSnbase version: 2.0.2
```

```
# How many missing values per peptide
table(fData(res)$count.missing)
```

```
##
##     0     1     2     3     4     5     6     7     8     9
## 11451   551    38    16     8     4     2     4     1     2
```

```
colSums(is.na(exprs(res)))
```

```
##  150mJ.1 150mJ.2  150mJ.3   275mJ.1  275mJ.2  275mJ.3  400mJ.1  400mJ.2
##       22       11      612       49        6       17        6       33
##  400mJ.3   Pool.1
##        6       31
```

```
table(rowSums(is.na(exprs(res))))
```

```
##
##     0     1     2     3     4     5     6     7     8     9
## 11451   551    38    16     8     4     2     4     1     2
```

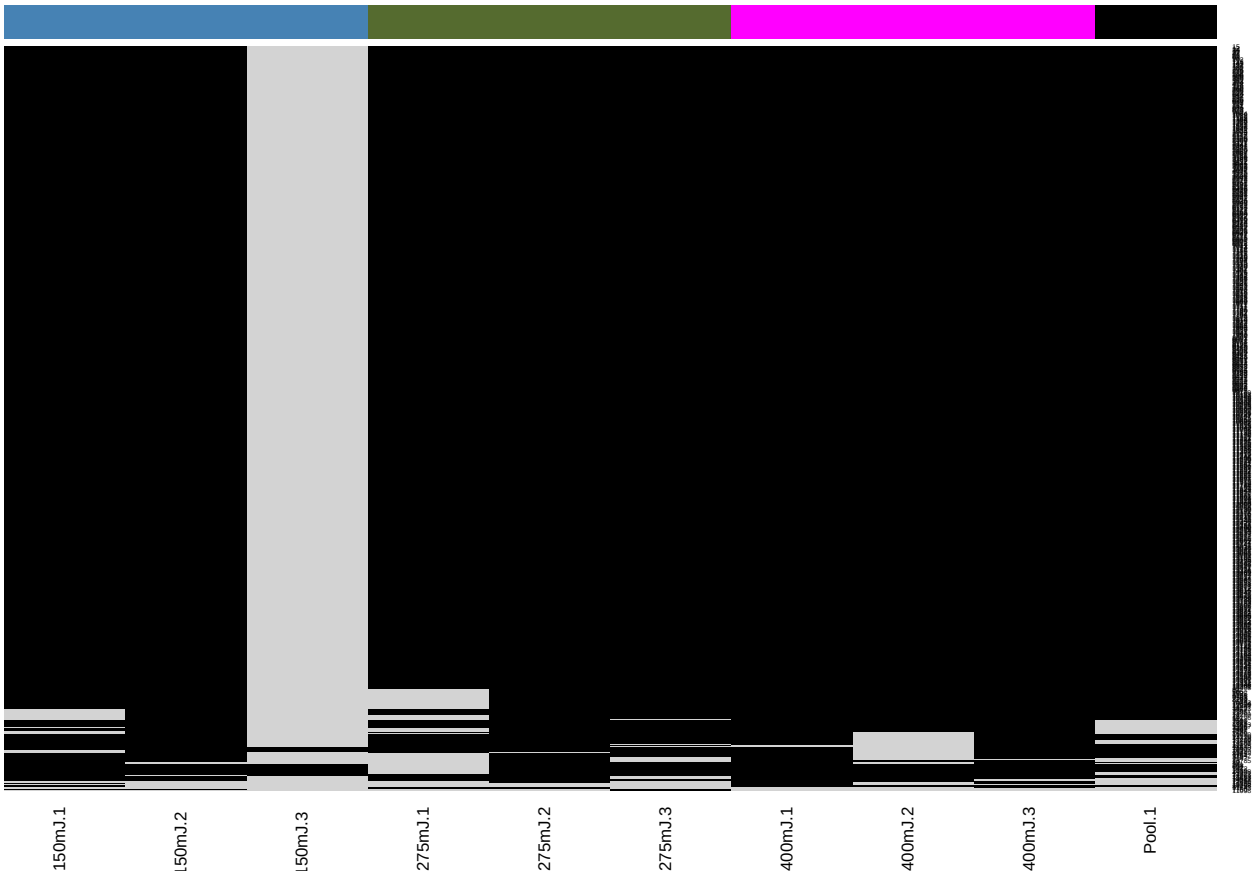
```
# Checking missing values
table(is.na(res))
```

```
##
```

```
##  FALSE   TRUE
## 119977    793
```

The MSnSet object has been created to include protein abundance values, some metadata and sample information (UV dosage in this case)

```
# ------------------------------------------------------------------------------
# Step 02b : Imputing missing values in the protein expression data using 'impute'
# ------------------------------------------------------------------------------

# Subsetting only those peptides with one or more missing values
# Replacing missing values with 0 and non-missing with 1
# Displaying missing values

miss.many = res[rowSums(is.na(exprs(res)))>=1,]
exprs(miss.many)[exprs(miss.many) != 0] = 1
exprs(miss.many)[is.na(exprs(miss.many))] = 0

heatmap.2(exprs(miss.many), col = c("lightgray", "black"),
            scale = "none", dendrogram = "none",
            trace = "none", keysize = 0.5, key = FALSE,Colv=F,
            ColSideColors = rep(c("steelblue", "darkolivegreen","magenta","black"), times = c(3,3,3,1)))
```

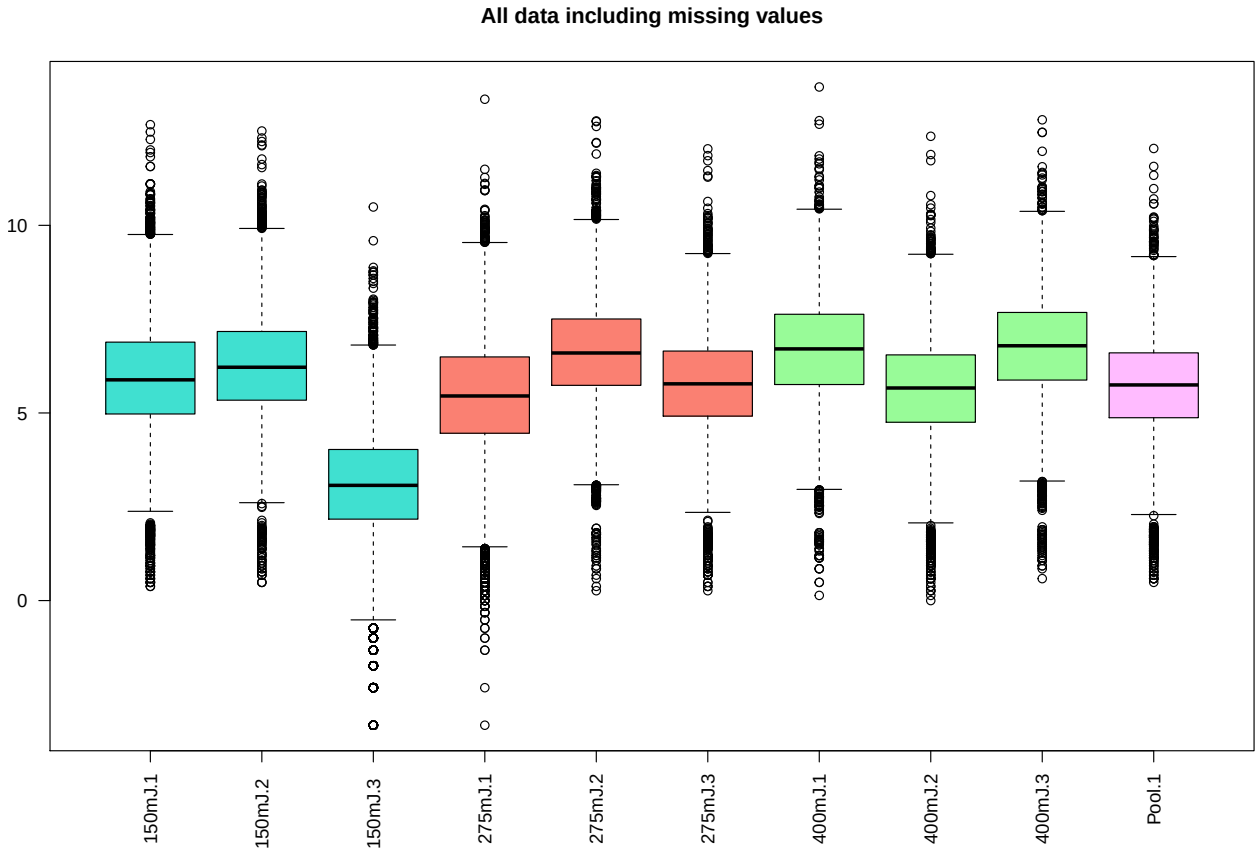



```
# Impute missing values
impute.res <- impute(res,method = "knn")
```

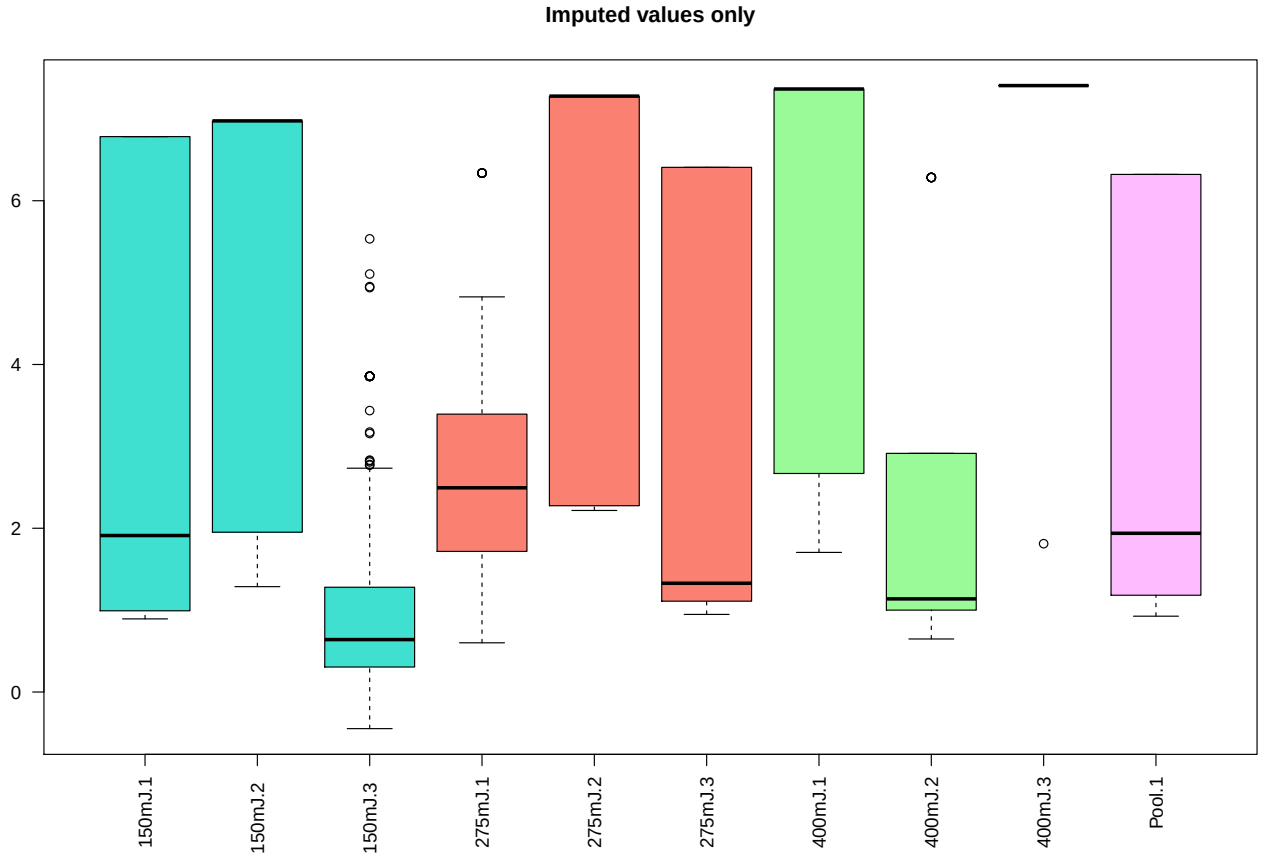
```
## Cluster size 12068 broken into 11832 236
## Cluster size 11832 broken into 8982 2850
```

```
## Cluster size 8982 broken into 3383 5599
## Cluster size 3383 broken into 1164 2219
## Done cluster 1164
## Cluster size 2219 broken into 1271 948
## Done cluster 1271
## Done cluster 948
## Done cluster 2219
## Done cluster 3383
## Cluster size 5599 broken into 3042 2557
## Cluster size 3042 broken into 1360 1682
## Done cluster 1360
## Cluster size 1682 broken into 631 1051
## Done cluster 631
## Done cluster 1051
## Done cluster 1682
## Done cluster 3042
## Cluster size 2557 broken into 1453 1104
## Done cluster 1453
## Done cluster 1104
## Done cluster 2557
## Done cluster 5599
## Done cluster 8982
## Cluster size 2850 broken into 725 2125
## Done cluster 725
## Cluster size 2125 broken into 1238 887
## Done cluster 1238
## Done cluster 887
## Done cluster 2125
## Done cluster 2850
## Done cluster 11832
## Done cluster 236
```

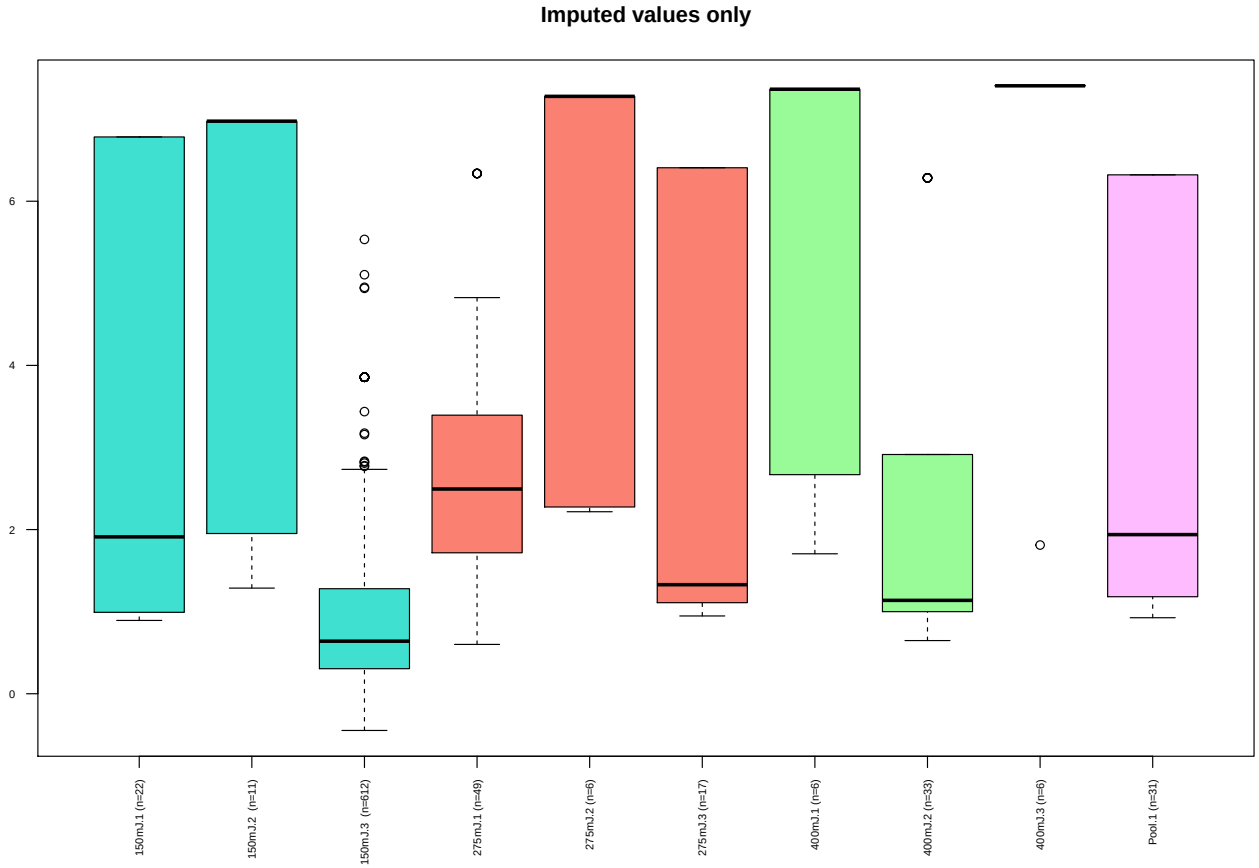
```
pData(impute.res)$Sample = gsub('\\s+','',pData(impute.res)$Sample)

# Plot imputed values
res.miss = melt(exprs(res))
colnames(res.miss) = c("Row","Dosage","Abundance_imp")
res.no.miss = melt(exprs(impute.res))
colnames(res.no.miss) = c("Row","Dosage","Abundance")

imp.vals = res.no.miss[which(is.na(res.miss$Abundance_imp)),]

# Some boxplots of the data
#--------------------------

# All data including missing values
boxplot(log2(res.miss$Abundance)~as.factor(res.miss$Dosage),las=2,col=rep(c("turquoise", "salmon","pale
```



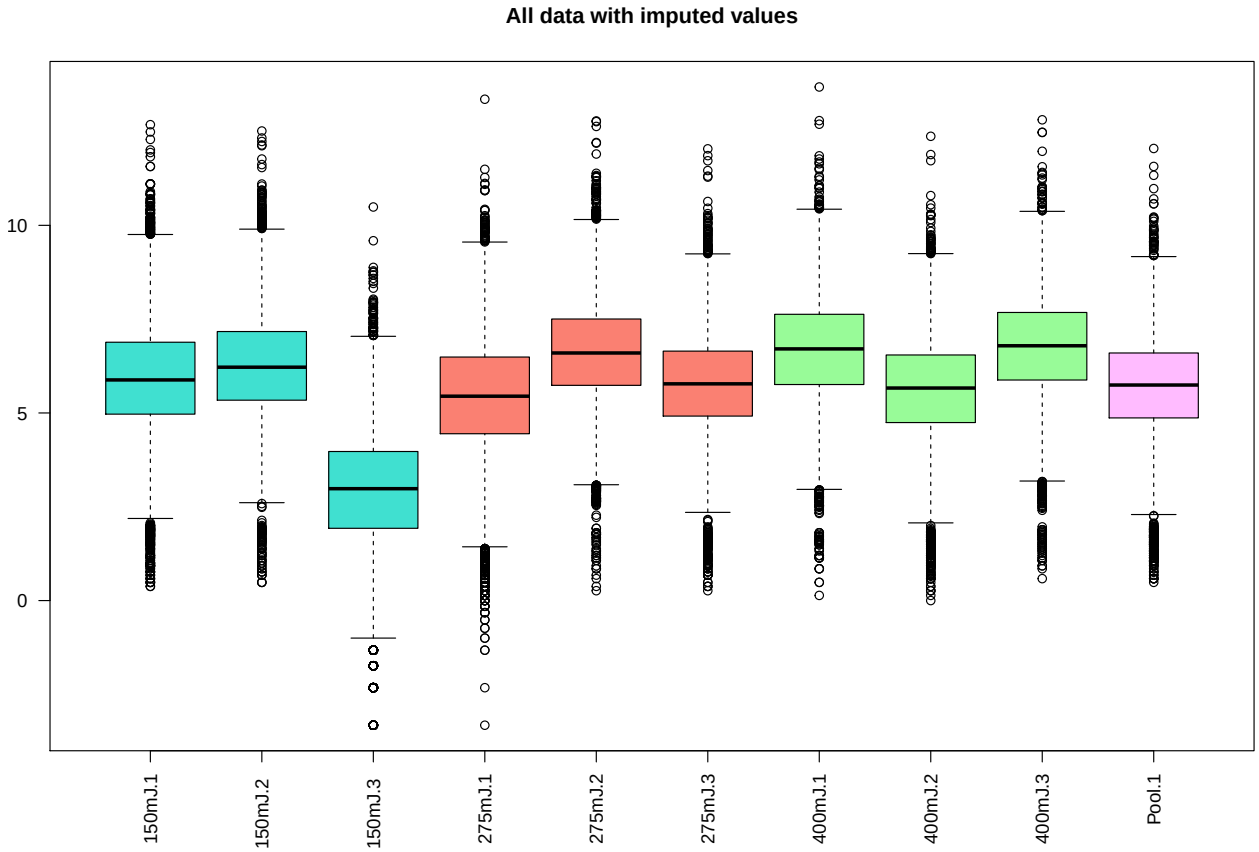

```
# Imputed values/missing values only
b.imp = boxplot(log2(imp.vals$Abundance)~imp.vals$Dosage,las=2,col=rep(c("turquoise", "salmon","palegre
```




```
boxplot(log2(imp.vals$Abundance)~imp.vals$Dosage,las=2,names = paste(b.imp$names," (n=", b.imp$n, ")",s
```




```
# All data icluding newly imputed values
boxplot(log2(res.no.miss$Abundance)~as.factor(res.no.miss$Dosage),las=2,col=rep(c("turquoise", "salmon"
```



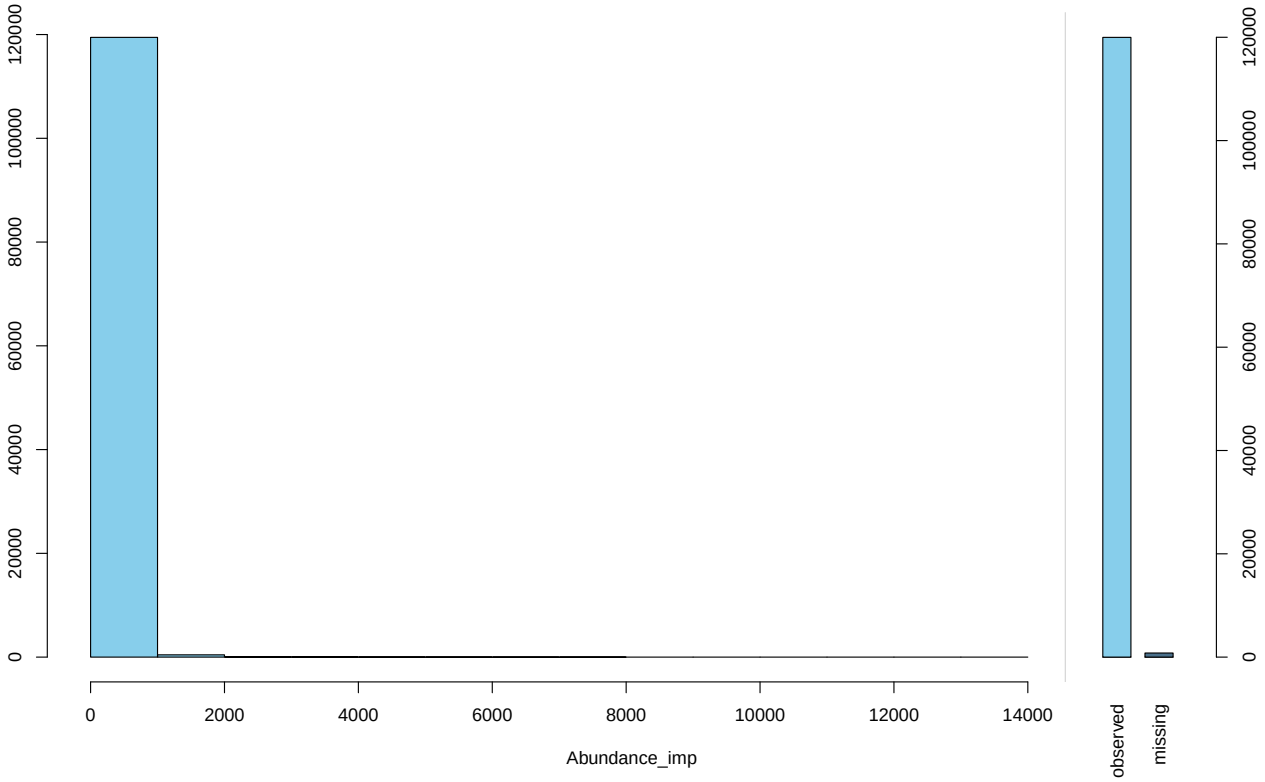

```
# Additional plots showing fraction of missing values
# Not much use as in our case, it is very small.

vim.dat = data.frame(cbind(Abundance_imp=res.miss$Abundance_imp,Abundance=res.no.miss$Abundance),string

head(vim.dat)
```

```
##   Abundance_imp Abundance
## 1           3.3       3.3
## 2          15.7      15.7
## 3          36.6      36.6
## 4          61.7      61.7
## 5          44.5      44.5
## 6           1.4       1.4
```

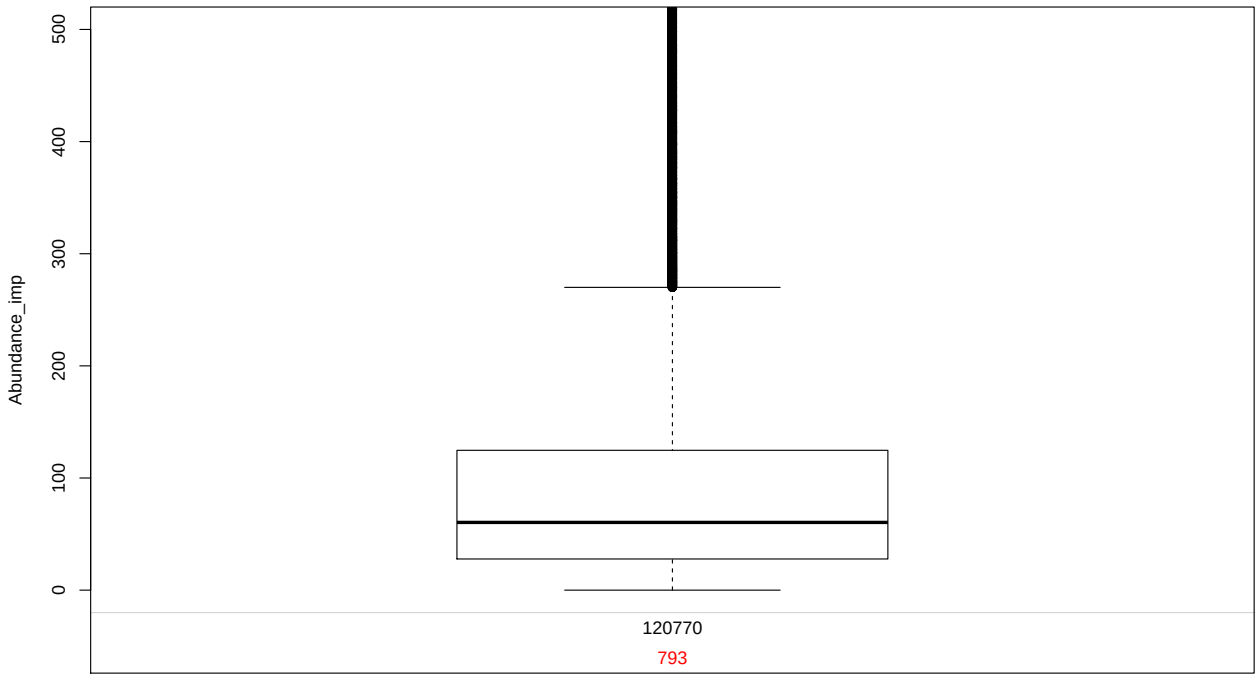
```
vim.dat$which.miss = FALSE
vim.dat$which.miss[which(is.na(vim.dat$Abundance_imp))] = TRUE
table(vim.dat$which.miss)
```

```
##
##  FALSE   TRUE
## 119977    793
```

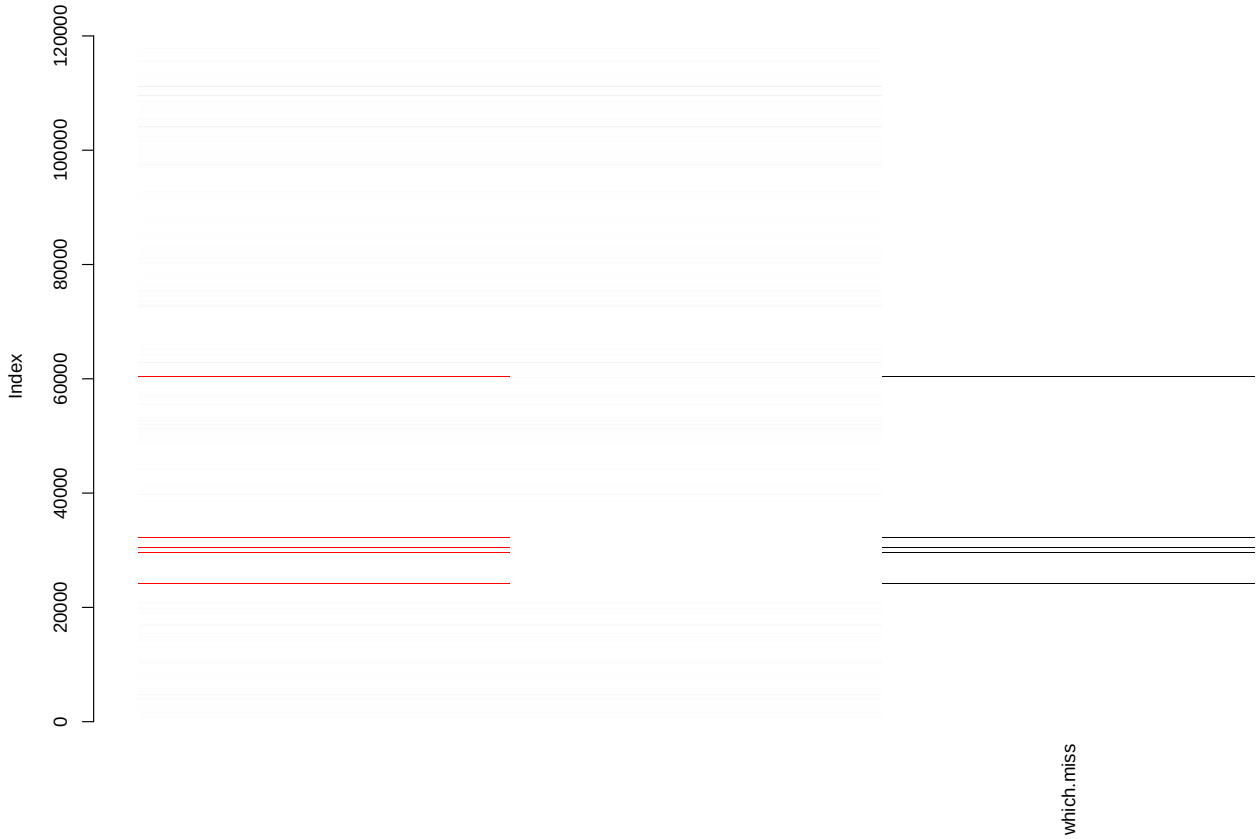
```
histMiss(vim.dat, only.miss = F)
```




```
pbox(vim.dat, ylim=c(0,500),selection="none")
```



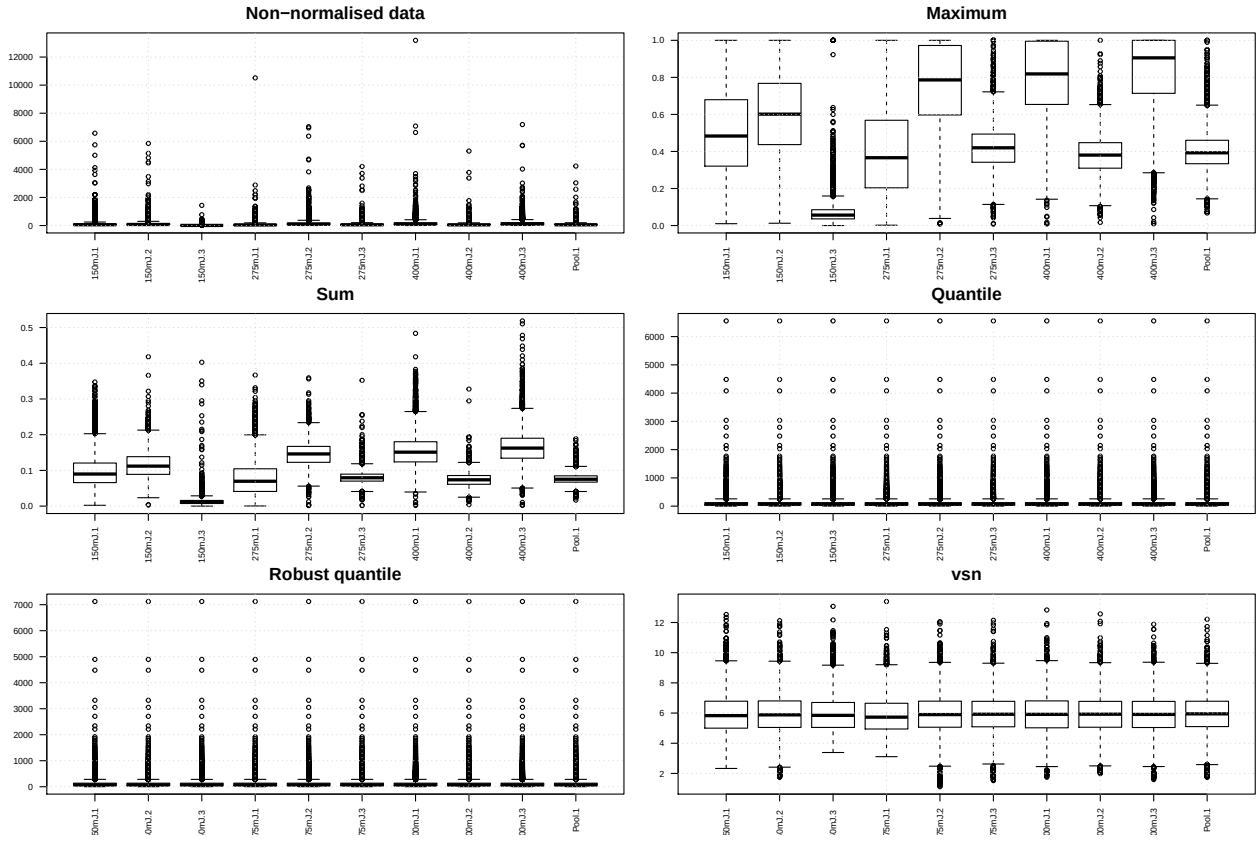

```
matrixplot(vim.dat)
```



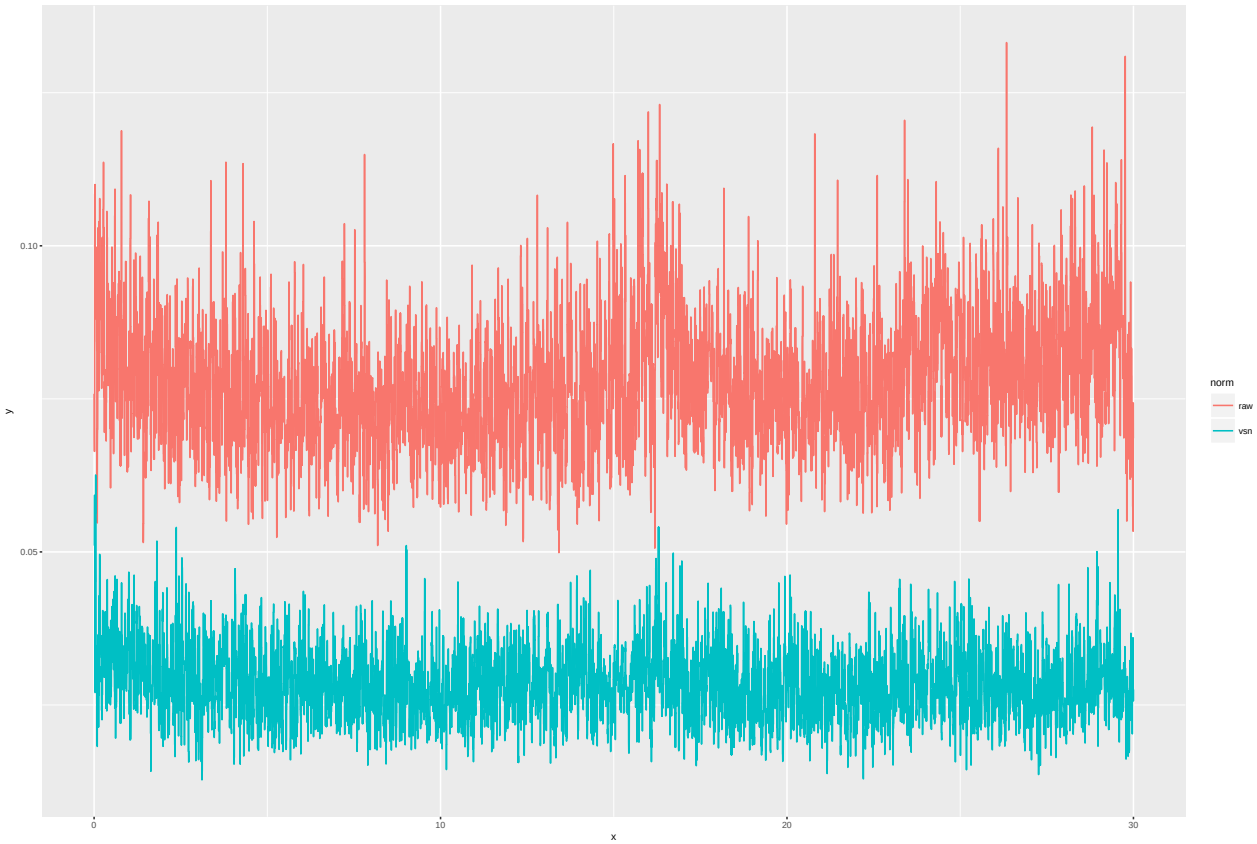
We have a small number of missing values - 793 peptides have one or more missing values out of 12244 peptides in total. 551/793 are missing in 1 sample only and 38/793 in 2 samples. Only 2 peptides are missing in 9/10 samples.

```
# ------------------------------------------------------------------------------
# Step 03 : Normalising imputed data using various methods to determine ideal one
# ------------------------------------------------------------------------------

qnt.max <- normalise(impute.res, "max")
qnt.sum <- normalise(impute.res, "sum")
qnt.quant <- normalise(impute.res, "quantiles")
qnt.qrob <- normalise(impute.res, "quantiles.robust")
qnt.vsn <- normalise(impute.res, "vsn")

## ---- plotting function---------
.plot <- function(x,ttl=NULL) {
  boxplot(exprs(x),
          main=ifelse(is.null(ttl),processingData(x)@processing[2],ttl),
          cex.main=1.5,
          cex.lab=.5,
          cex.axis=0.8,
          cex=.8,
          las=2)
  grid()
}

# Using the plotting function to plot boxplots for all diff types of normalisation methods
oldmar <- par()$mar
```

```
par(mfrow=c(3,2),mar=c(2.9,2.9,2.9,1))
.plot(impute.res, ttl = "Non-normalised data")
.plot(qnt.max, ttl = "Maximum")
.plot(qnt.sum, ttl = "Sum")
.plot(qnt.quant, ttl = "Quantile")
.plot(qnt.qrob, ttl = "Robust quantile")
.plot(qnt.vsn, ttl = "vsn")
```




```
# Checking the effects of normalisation on covariance
sd1 <- apply(log2(exprs(impute.res))+10,1,sd)
mn1 <- apply(log2(exprs(impute.res))+10,1,mean)
cv1 <- sd1/mn1
sd2 <- apply(exprs(qnt.vsn)+10,1,sd)
mn2 <- apply(exprs(qnt.vsn)+10,1,mean)
cv2 <- sd2/mn2
dfr <- rbind(data.frame(rank=order(mn1),cv=cv1,norm="raw"),
             data.frame(rank=order(mn2),cv=cv2,norm="vsn"))

rmed1 <- rollapply(cv1,7,function(x) median(x,na.rm=TRUE))
rmed2 <- rollapply(cv2,7,function(x) median(x,na.rm=TRUE))
dfr2 <- rbind(data.frame(x=seq(0,30,by=30/length(rmed1))[-1],y=rmed1,norm="raw"),data.frame(x=seq(0,30,

p <- ggplot()+geom_line(data=dfr2,aes(x=x,y=y,col=norm)) + theme_gray(7)
plot(p)
```



Will keep the data from the vsn normalisation for downstream analyses as it normalises the data better than other methods used in this comparison such as “sum”, “max”, “quantile”.

For all further steps, we will use the object “qnt.vsn”

```
# ------------------------------------------------------------------------------
# Step 04a : Aggregate peptide data to protein expression values
# There is an in-built function called 'combineFeatures' to do thi within MSnBase
# ------------------------------------------------------------------------------

protnames <- fData(qnt.vsn)$Master.Protein.Accessions
#table(protnames)
length(unique(protnames)) # 1744 proteins present in all 10 samples
```

```
## [1] 1744
```

```
# Aggregating peptide abundance values into protein abundance values
qnt.prot <- combineFeatures(qnt.vsn, groupBy = protnames, fun = "median")
dim(qnt.prot)
```

```
## [1] 1744   10
```

```
head(exprs(qnt.prot))
```

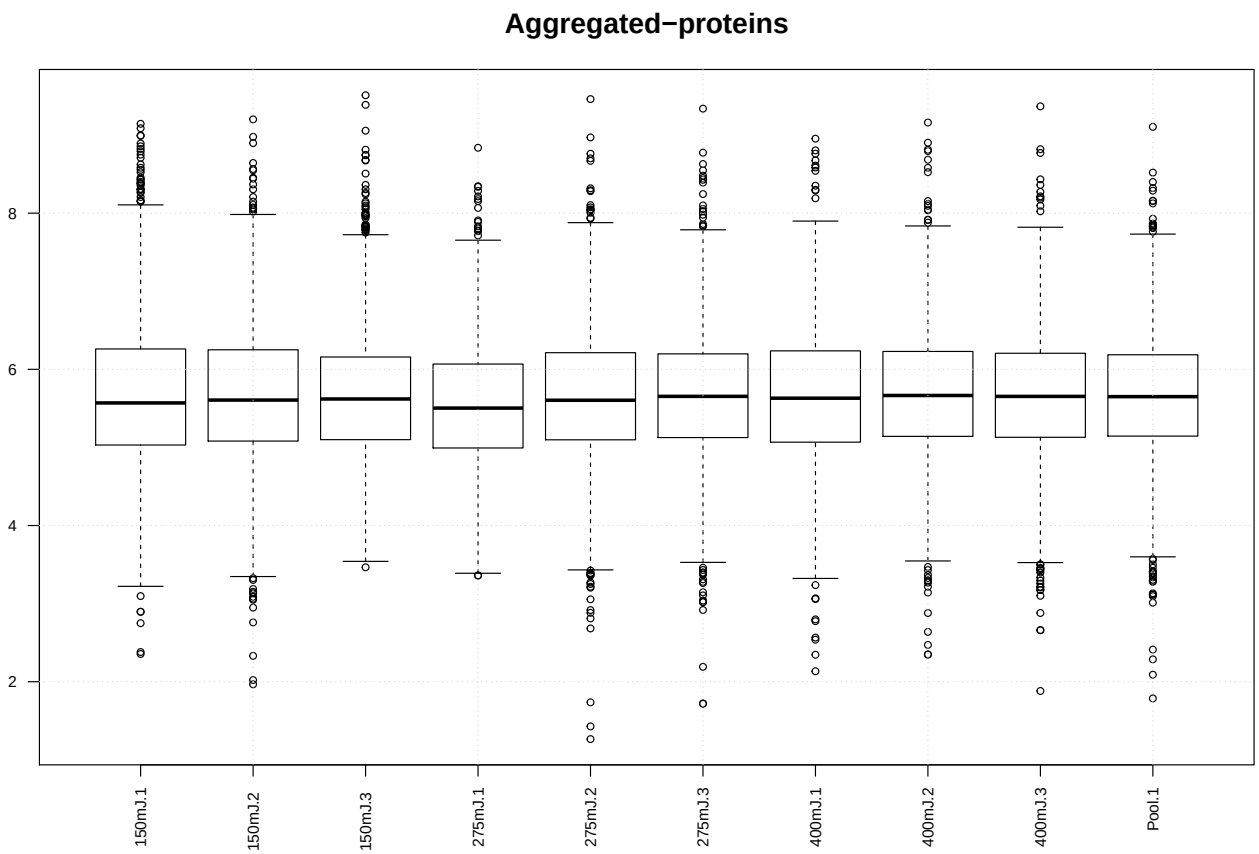
```
##             150mJ.1 150mJ.2  150mJ.3   275mJ.1  275mJ.2  275mJ.3  400mJ.1
## A0A024QZP7 3.816467 3.620175 3.927225 4.052191 3.357771 3.310799 3.357685
## A0A024R216 7.599068 8.060989 6.553413 5.027294 7.777270 7.677502 7.354644
## A0A024R4E5 6.329437 5.803502 6.730733 6.286203 6.320219 6.172760 5.830347
## A0A024R644 5.396222 5.055941 4.600265 4.333401 4.729008 4.757164 4.156337
## A0A075B716 6.043305 5.677170 5.677677 5.247188 5.606109 5.426848 5.970442
```

```
## A0A087WSV8 4.562525 5.022626 4.362861 4.670520 4.962158 5.500905 6.268788
##             400mJ.2  400mJ.3   Pool.1
## A0A024QZP7 3.268323 3.294870 3.651255
## A0A024R216 7.367304 7.238476 7.079061
## A0A024R4E5 5.964010 5.750300 6.287400
## A0A024R644 4.452842 4.393996 4.887391
## A0A075B716 5.801846 5.696659 5.619219
## A0A087WSV8 5.923221 5.913728 4.863713
```

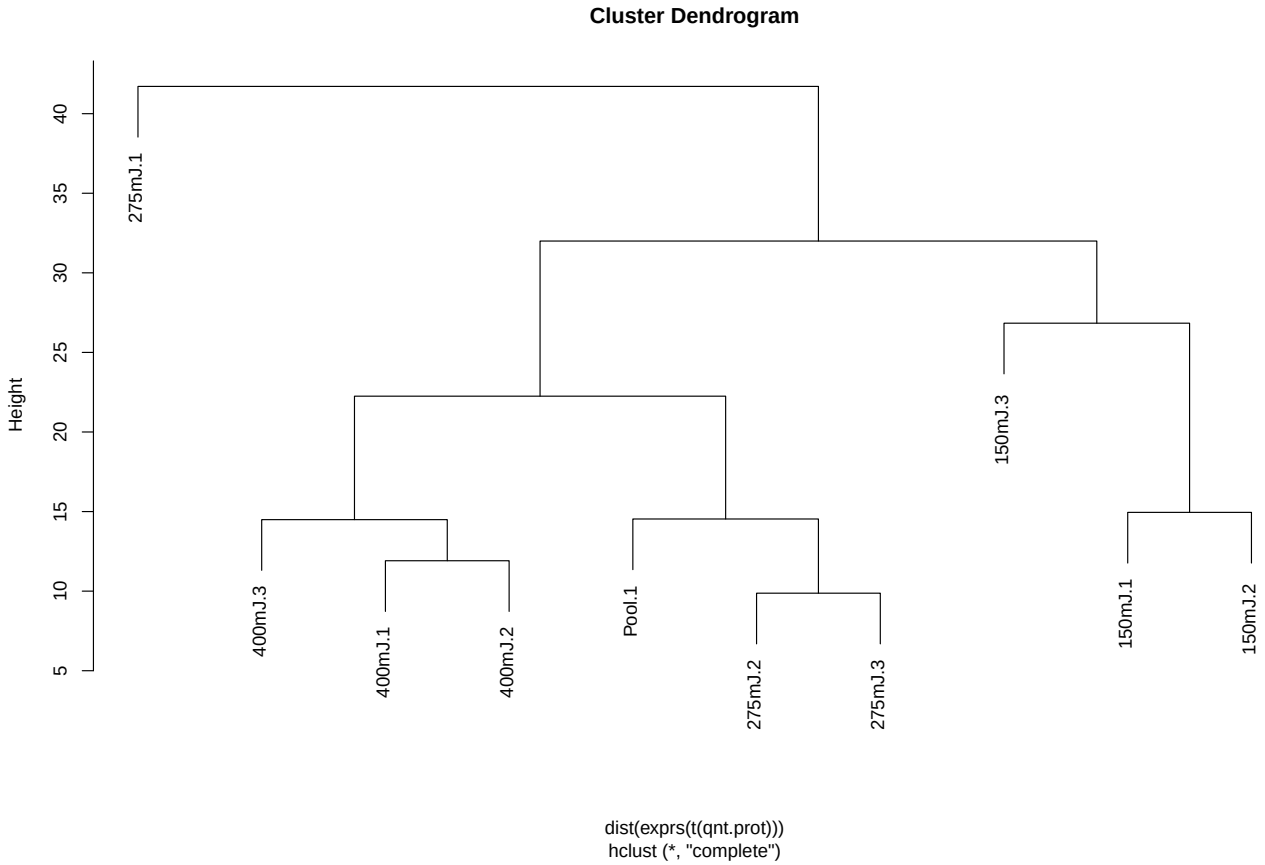
```
head(exprs(qnt.vsn))
```

```
##    150mJ.1 150mJ.2  150mJ.3   275mJ.1  275mJ.2  275mJ.3  400mJ.1  400mJ.2
## 2 2.768493 1.966696 6.596663 6.509413 2.401781 6.540405 1.903026 6.520467
## 3 4.166554 4.035286 5.620443 6.395213 4.589417 5.245986 6.144995 5.496511
## 4 5.200775 5.765512 5.219895 5.058563 5.142504 5.314060 4.817990 4.729333
## 5 5.892083 5.608025 5.671977 5.279402 5.237583 4.981204 5.499543 5.309686
## 6 5.455946 5.232610 4.995591 5.433890 5.079733 4.986121 5.175707 5.402227
## 7 2.355212 1.760496 4.276767 3.345973 1.786217 1.852261 2.003715 2.348329
##    400mJ.3   Pool.1
## 2 6.507783 6.511016
## 3 5.716597 5.117277
## 4 4.899958 4.730216
## 5 5.272549 5.135761
## 6 5.270593 5.470324
## 7 1.819560 2.025044
```

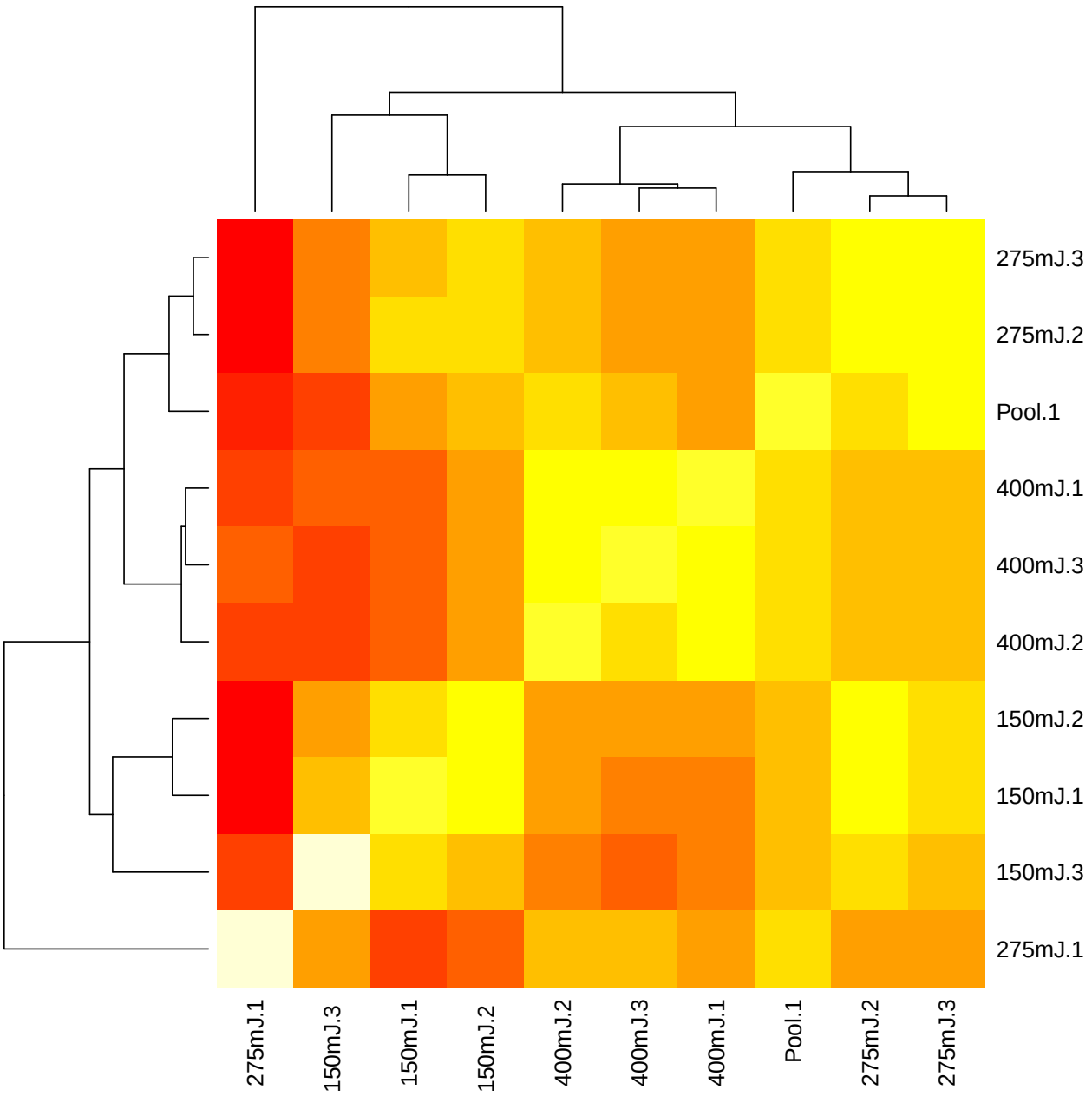
```
# Basic plots of protein data across samples
.plot(qnt.prot,ttl="Aggregated-proteins")
```



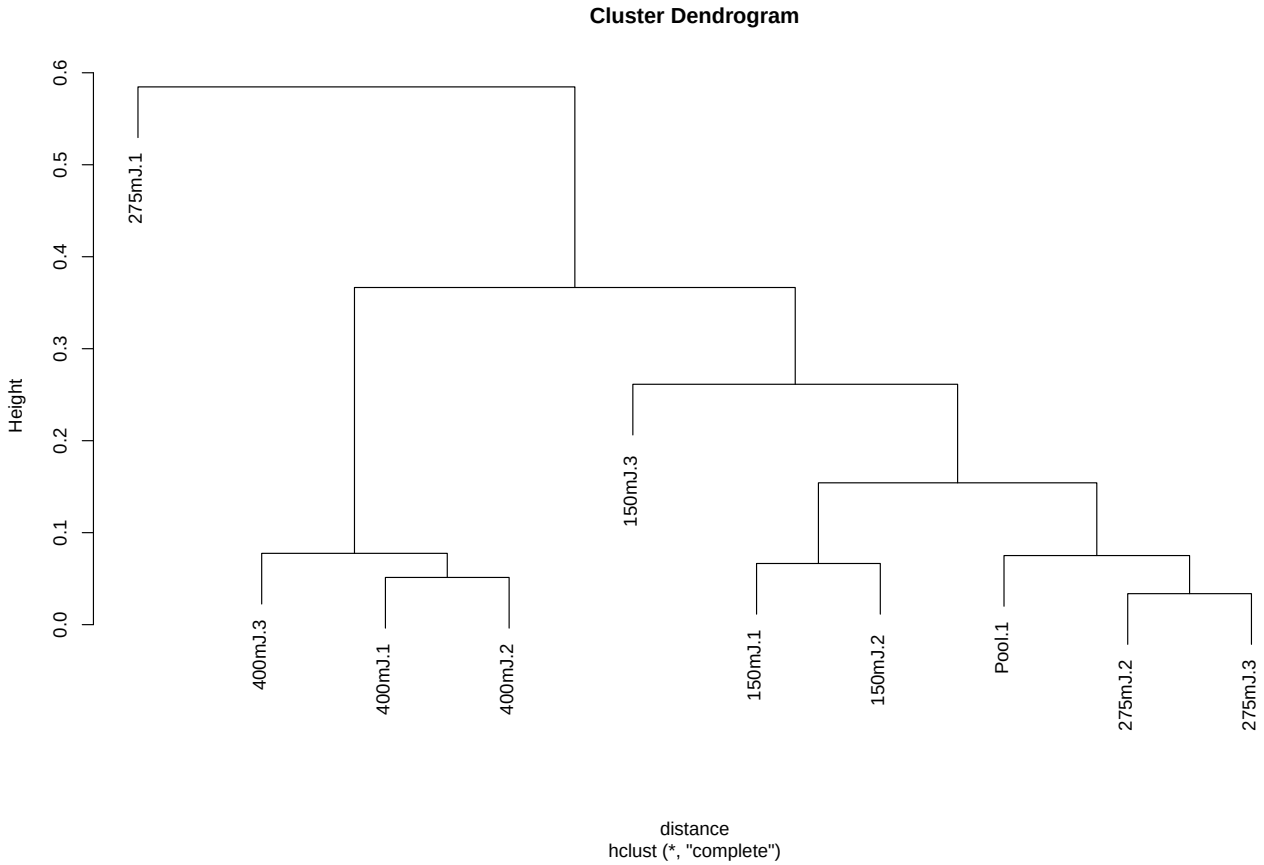
```r
plot(hclust(dist(exprs(t(qnt.prot)))))
```



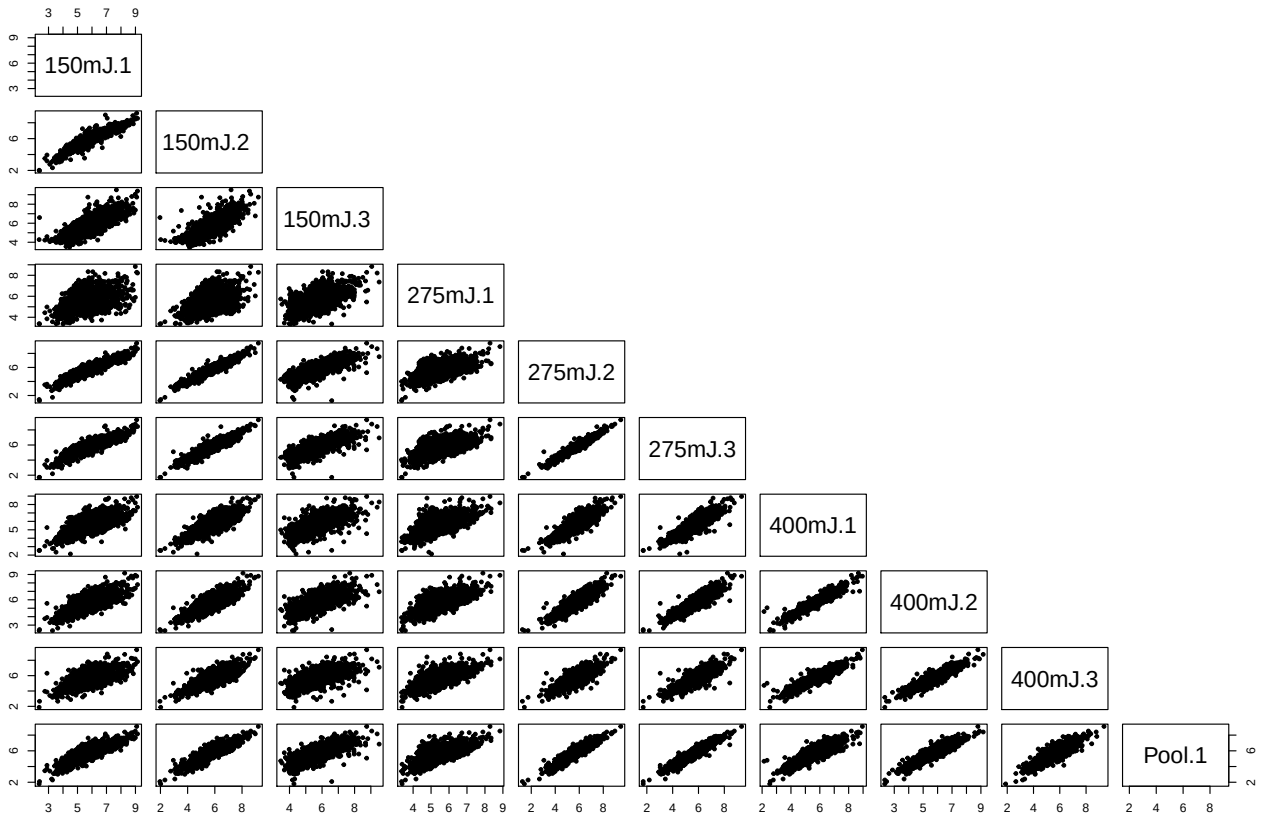

```r
# Looking at sample correlations
cor.prot = cor(exprs(qnt.prot))
heatmap(cor.prot,cex.main = 0.8)
```




```
dissimilarity <- 1 - cor.prot
distance <- as.dist(dissimilarity)
plot(hclust(distance))
```




```
pairs(x = exprs(qnt.prot), upper.panel=NULL, pch=20)
```




```
# Using the "duplicatecorrelation" function in limma to test correlation between technical replicates

# If Rayner's ordering is correct, then the samples are
biolrep.rq <- c(1,1,1,2,2,2,3,3,3,4)

# If there is a swap between Pool.1 and 275mj.rep1, then it should be
biolrep.swap <- c(1,1,1,4,2,2,3,3,3,2)

# If it is indeed a swap, then the correlation should increase when corrected. It does!
rq.cor = duplicateCorrelation(qnt.prot,ndups=1,block=biolrep.rq)$consensus.correlation # 0.203
swap.cor = duplicateCorrelation(qnt.prot,ndups=1,block=biolrep.swap)$consensus.correlation # 0.564

rq.cor
```

```
## [1] 0.2031868
```

```
swap.cor
```

```
## [1] 0.5641893
```

```
cor.prot
```

```
##            150mJ.1  150mJ.2   150mJ.3    275mJ.1   275mJ.2   275mJ.3
## 150mJ.1  1.0000000 0.9335052 0.7911591 0.4153498 0.9280369 0.8853188
## 150mJ.2  0.9335052 1.0000000 0.7551314 0.4370340 0.9500877 0.9236256
## 150mJ.3  0.7911591 0.7551314 1.0000000 0.5989892 0.7830856 0.7672976
## 275mJ.1  0.4153498 0.4370340 0.5989892 1.0000000 0.5515125 0.5783829
## 275mJ.2  0.9280369 0.9500877 0.7830856 0.5515125 1.0000000 0.9663206
## 275mJ.3  0.8853188 0.9236256 0.7672976 0.5783829 0.9663206 1.0000000
## 400mJ.1  0.6821358 0.7780666 0.6600698 0.6087879 0.8269695 0.8494414
```

```
## 400mJ.2  0.7285274 0.8027901 0.6912729 0.6728370 0.8625385 0.8778792
## 400mJ.3  0.6684451 0.7780055 0.6334646 0.6634102 0.8276158 0.8443968
## Pool.1   0.8458091 0.8816818 0.7386629 0.6850855 0.9248869 0.9432591
##            400mJ.1   400mJ.2   400mJ.3    Pool.1
## 150mJ.1  0.6821358 0.7285274 0.6684451 0.8458091
## 150mJ.2  0.7780666 0.8027901 0.7780055 0.8816818
## 150mJ.3  0.6600698 0.6912729 0.6334646 0.7386629
## 275mJ.1  0.6087879 0.6728370 0.6634102 0.6850855
## 275mJ.2  0.8269695 0.8625385 0.8276158 0.9248869
## 275mJ.3  0.8494414 0.8778792 0.8443968 0.9432591
## 400mJ.1  1.0000000 0.9486792 0.9224763 0.8623312
## 400mJ.2  0.9486792 1.0000000 0.9264799 0.9100994
## 400mJ.3  0.9224763 0.9264799 1.0000000 0.8713971
## Pool.1   0.8623312 0.9100994 0.8713971 1.0000000
```

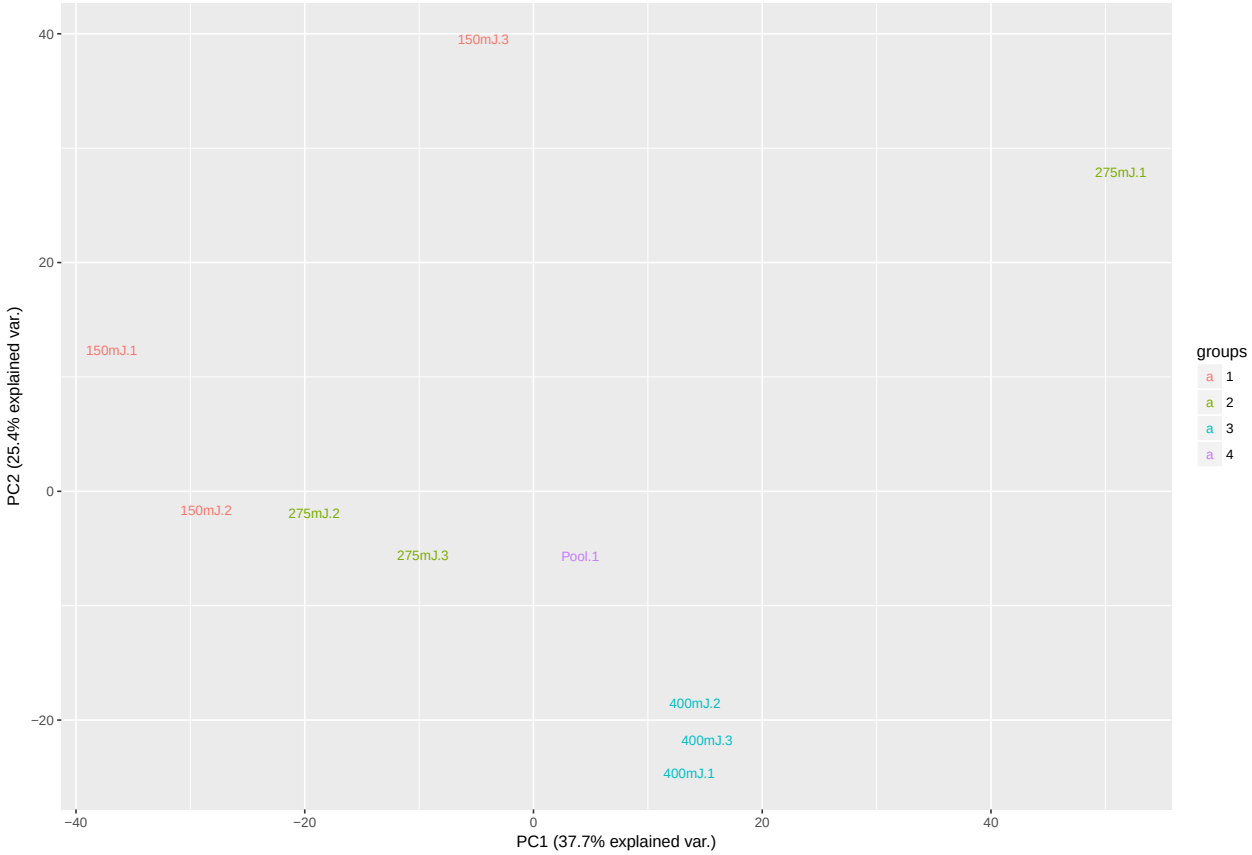
```
#duplicateCorrelation(qnt.prot[,1:9],ndups=1,block=c(1,1,1,2,2,2,3,3,3))$consensus.correlation # 0.23
#duplicateCorrelation(qnt.prot[,c(1:3,5:10)],ndups=1,block=c(1,1,1,2,2,3,3,3,2))$consensus.correlation
```

We have merged the peptides into proteins and are looking here at the correlations within replicates of UV dosage. Replicate 3 of 150mJ dosage is a bit off from the other two while replicate 1 of 275mJ sits by itself allowing the "Pool" sample to cluster with the other 275mJ samples.

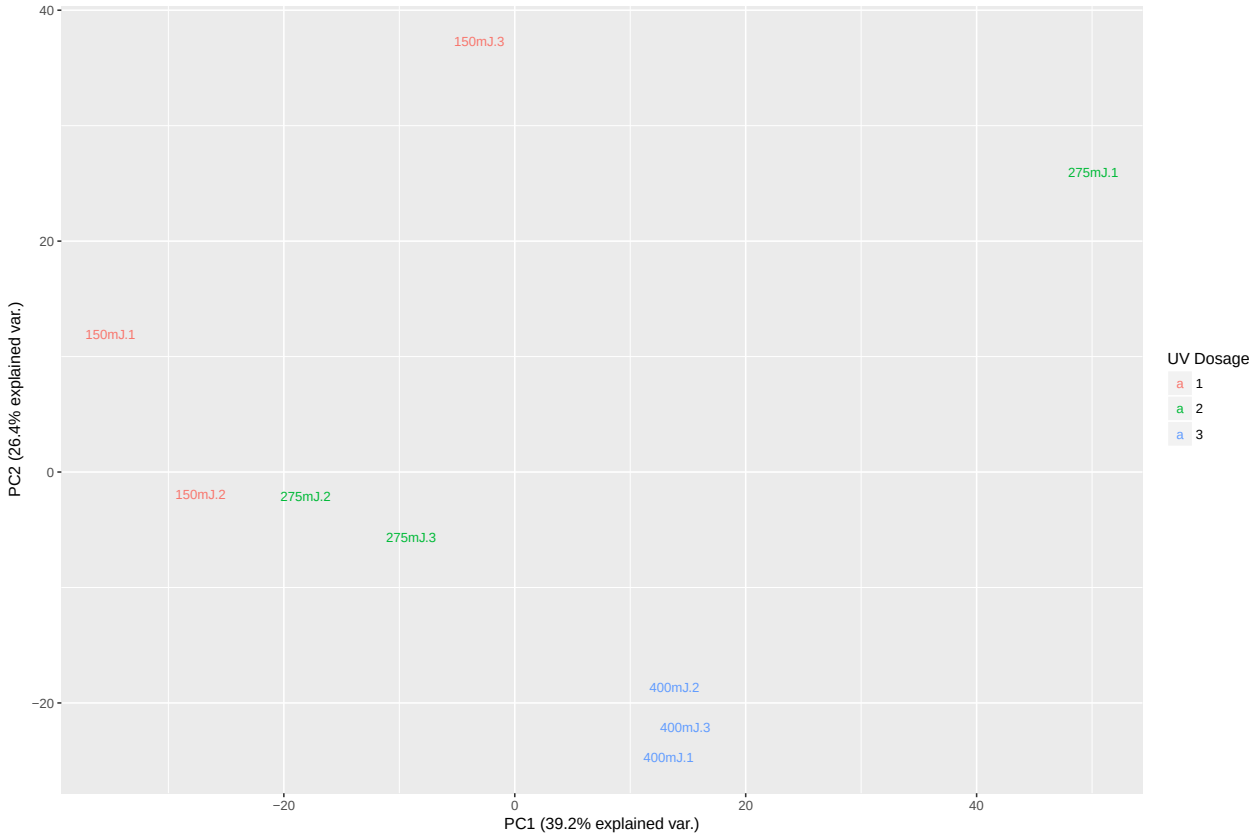
I thought that 275mJ.3 and Pool might have been swapped. So the various plots were done to prove whether or not this was true. The "duplicateCorrelation" function yields a much higher correlation when we assume that 275mJ.3 is swapped with Pool.1 than if we didn't.

Looking at th correlation values, 275.mJ vs Pool is more correlated than 150mJ vs Pool and 400mJ vs Pool. This could be becuase there was more material from sample 275mJ going into the pool. Rayner used the same volume of each of the 9 samples rather than same amount in the pool. Hence the higher correlation (well at least it is my best guess).

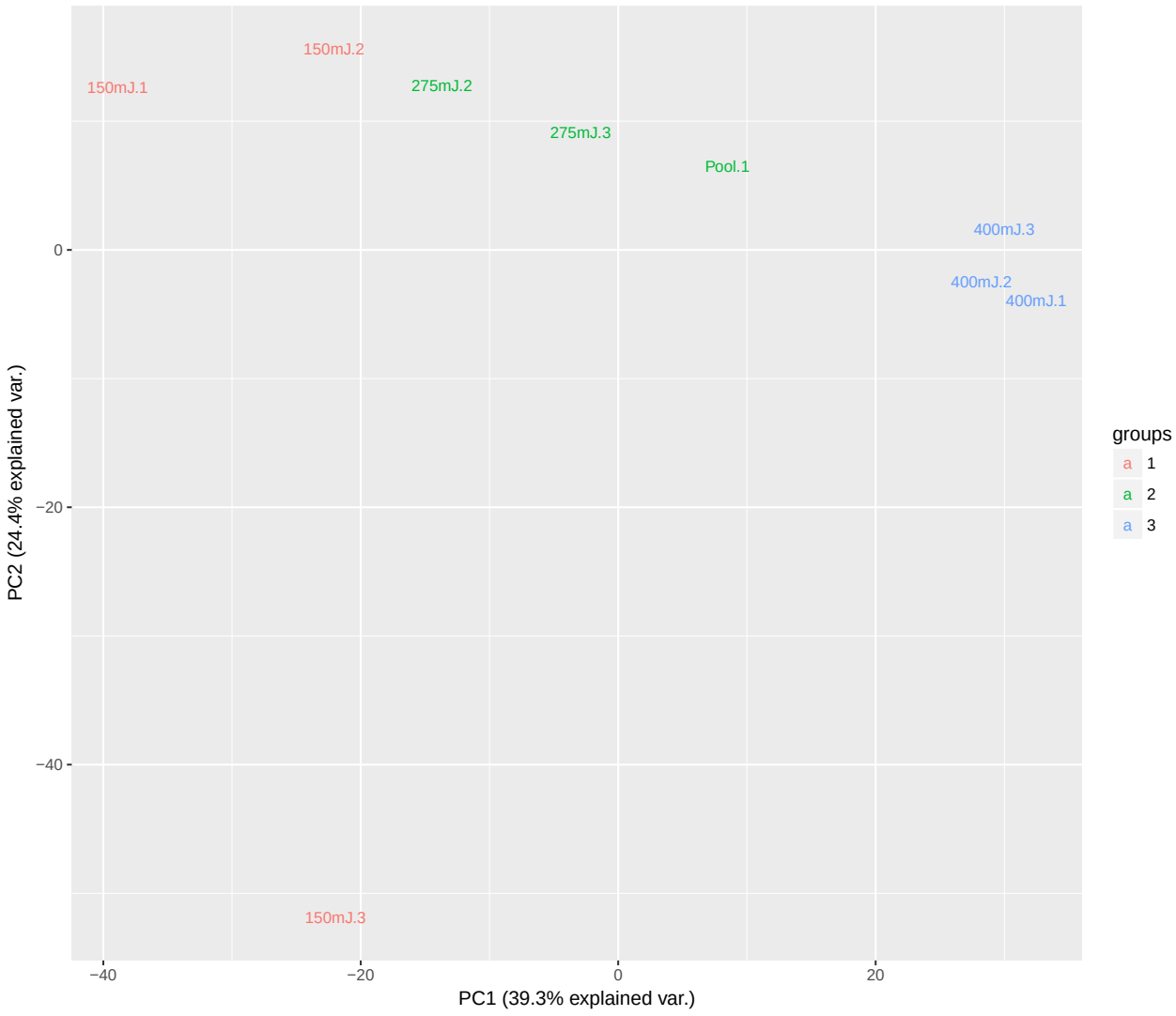
```
# ------------------------------------------------------------------------------
# Step 04a : Plotting PCAs
# This is to look at variability across samples and within replicates
# ------------------------------------------------------------------------------

# Across all 10 samples
prot.pca = prcomp(t(exprs(qnt.prot)),scale=T)
j <- ggbiplot(prot.pca, var.axes=F, groups = factor(c(1,1,1,2,2,2,3,3,3,4)), circle = T,obs.scale=1,labe
print(j)
```



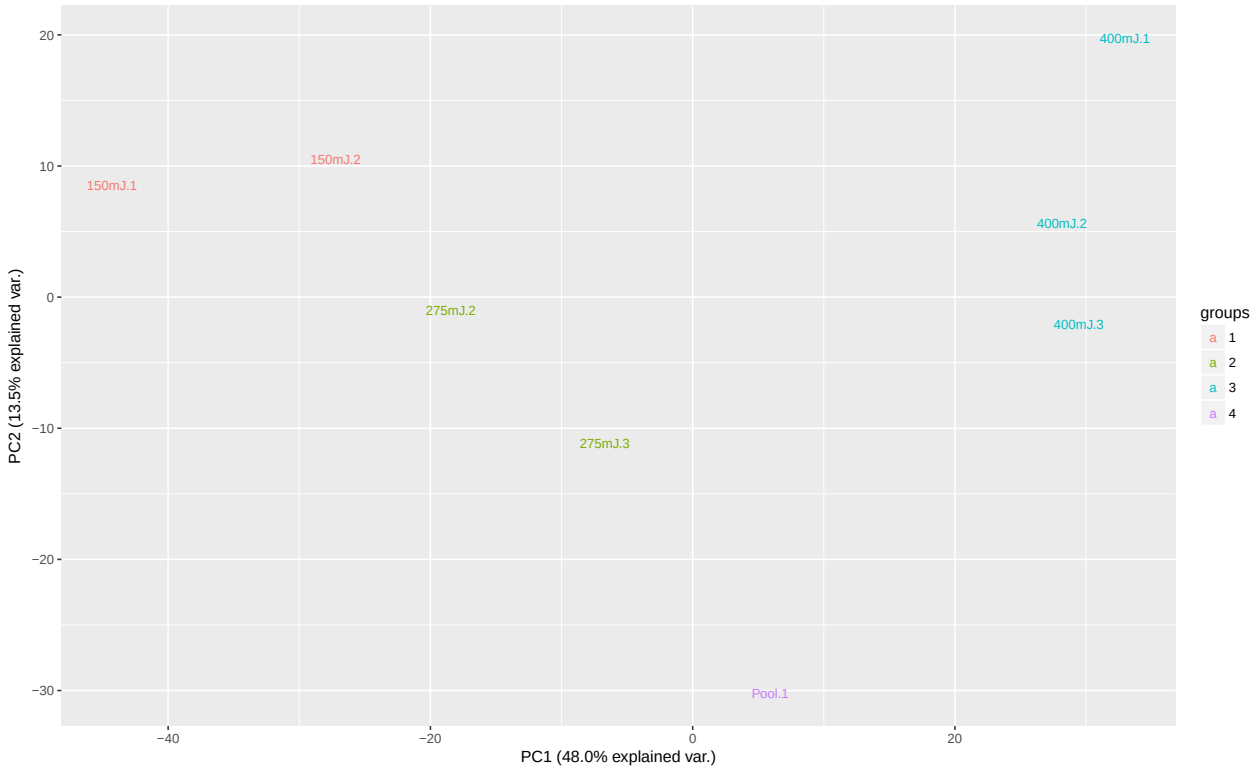

```
# Excluding the Pool.1 sample
prot.pca.rq = prcomp(t(exprs(qnt.prot[,c(1:9)])),scale=T)
g <- ggbiplot(prot.pca.rq, var.axes=F, groups = factor(c(1,1,1,2,2,2,3,3,3)), circle = T,obs.scale=1,lab
g <- g+labs(colour = "UV Dosage")
print(g)
```



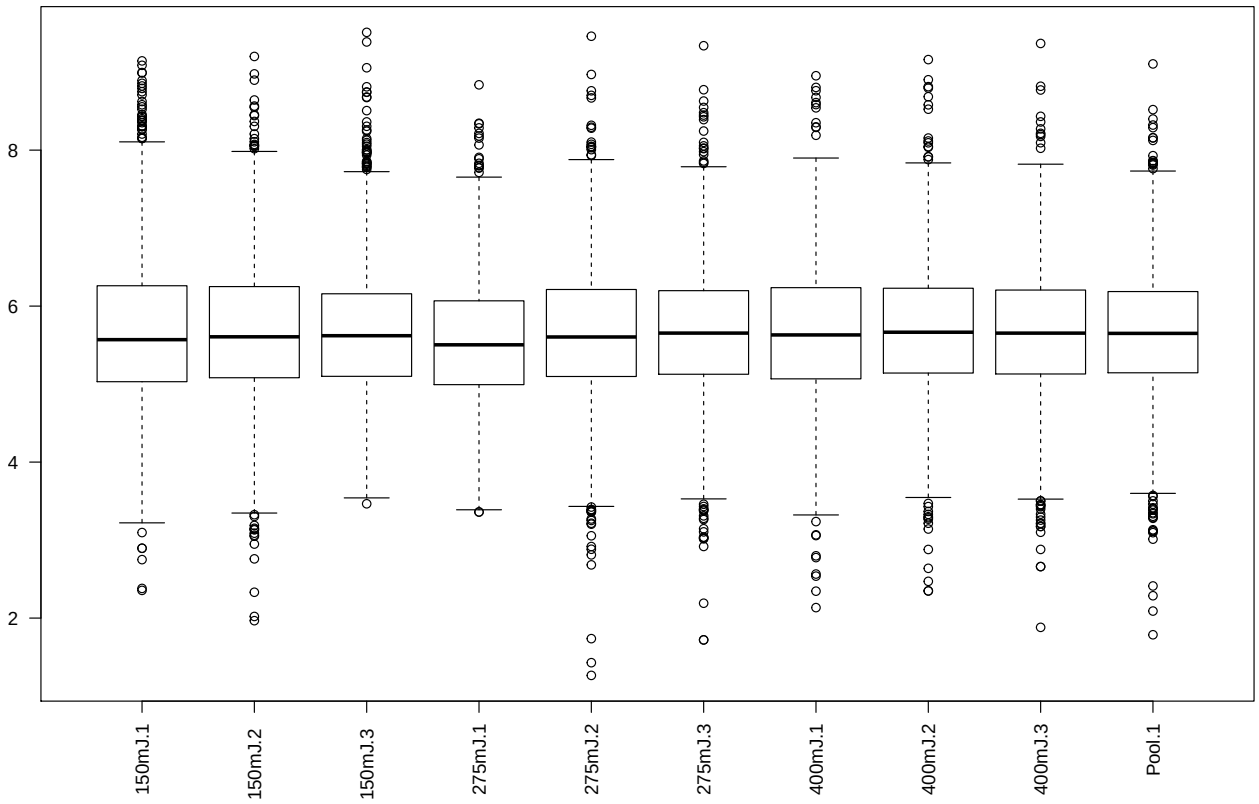

```
# Assuming Pool.1 is actually 275mJ.rep1
prot.pca.swap = prcomp(t(exprs(qnt.prot[,c(1:3,5:10)])),scale=T)
h <- ggbiplot(prot.pca.swap, var.axes=F, groups = factor(c(1,1,1,2,2,3,3,3,2)), circle = T,obs.scale=1,
print(h)
```




```
# Removing both problem samples
prot.pca.no.dud = prcomp(t(exprs(qnt.prot[,c(1:2,5:10)])),scale=T)
k <- ggbiplot(prot.pca.no.dud, var.axes=F, groups = factor(c(1,1,2,2,3,3,3,4)), circle = T,obs.scale=1,
print(k)
```




```
# To finish off, some boxplots....
pData(qnt.prot) = pData(impute.res)

# Are the values vastly different
boxplot(exprs(qnt.prot),las=2)
```



I've been looking to explain what I see. It seems like the most obvious answer would be that Pool.1 and 275mJ.1 are swapped. However, upon speaking with Rayner, this did not happen. He sets up the tubes in a row and adds the labels in order so the last sample was the pool and the 4th was 275mJ.1. What Rayner did say was that the tube with 150mJ.3 was dropped and sample had a little shake/tumble which might have affected its quality. I don't know at which stage of the protocol this happened but it did. This would explain the separation of 150mJ.3 from the other two 150mJs. However, the 150mJ triplicate still cluster together on the hclust plots. With sample 275mJ.1, Rayner remembers it being odd but we don't know why or how.

An alternate explanation is that sample 275mJ.1 did not elute(?) out properly while it was being prepared but it did by the time the pool was made. The pool was made of equal volumes of all 9 samples rather than equal protein amounts so there will be a higher representation of more abundant samples and lower of less abundant samples.

Have gone back to look at the "cor.prot" matrix and as Tom suspected, the Pool.1 correlates slightly better with non-dodgey 275mJ (corr = 0.94) than with 400mJ (0.88) and non-dodgey 150mJ (0.87) but not significantly more, hence the cluster plot looks like it does. The two dogey samples 150mJ.3 and 275mJ.1 have been left out of the average correlation calculation above.

Would like to take a picture of the table of final concentrations of samples that went into the pool and put it in the pData dataframe so we have it for furture reference. Rayner is re-doing experiment.

The boxplots seem to indicate that the two dodgey samples are missing proteins that are lowly expressed. Maybe they were too low to be detected by the mass spec ?

```
# ------------------------------------------------------------------------------------------------------------
# Step 05: Creating a background list of proteins for U2OS to be used for functional enrichment
# Using GO terms and Interpro domains for function of proteins detected by UV crosslinking and Trizol-e
# All data are based on Trizol-4-step and 400mJ of UV exposure for 2 mins (?)
# ------------------------------------------------------------------------------------------------------------
```

```
#=============================================================================================
# Using U2OS protein list from Geiger et al as the background list. This consists of maximal list of pr
# ~ 7500 proteins. We are however reading in the peptides and aggregating them using MSnbase as we did
#=============================================================================================

# Read data from Tom's list of Geiger proteins
# Tom took the Supplementary data from paper and re-annotated it with master proteins using his script.
# The script gets focuses on using Swissprot IDs rather than both Swissprot and Trembl.
# It also marks which proteins are "crap" or "contaminants"
# Finally, it provides a measure of which peptides could be mapped to a unique protein. We use this as
u2os <- read.delim("Input/Geiger_et_al_2012_U2OS_peptides_plus_master.txt",sep="\t",header=T)
head(u2os)
```

```
##                                                Gene.Names
## 1                                               ACTN4;ACTN4
## 2                                             NSEP1;YB1;YBX1
## 3                                                      TKT
## 4            MIG10;OK/SW-cl.110;PGK1;PGKA;hCG_20034;PGK2;PGKB
## 5 NAMPT;PBEF;PBEF1;RP11-92J19.4;RP11-92J19.4-001;DKFZp666B131
## 6                                               FABP5;FABP5
##
## 1
## 2
## 3 cDNA FLJ53217, highly similar to Transketolase (EC 2.2.1.1);cDNA FLJ32530 fis, clone SMINT2000185,
## 4
## 5
## 6
##                                               Uniprot
## 1                           O43707;Q96BG6;A4K467;C9J4H6
## 2                       P67809;A0JLU4;Q05D43;Q6PKI6;Q2VIK8
## 3 B4DVU1;B3KPZ8;P29401;A8K089;B4E022;Q53EM5;B4DE31;B3KSI4
## 4 P00558;A8K4W6;B7Z7A9;B4DHM5;B4E1H9;B4DHB3;B4DWQ3;P07205
## 5                                   P43490;Q5SYT8;Q658Z1
## 6                                          Q01469;A8MTQ5
##                                                                  Proteins
## 1                                    IPI00013808;IPI00845465;IPI00942539
## 2                                    IPI00031812;IPI00643351;IPI00479509
## 3 IPI00940673;IPI00793119;IPI00942979;IPI00643920;IPI00792641;IPI00789310
## 4             IPI00169383;IPI00916818;IPI00909158;IPI00910974;IPI00219568
## 5                                                 IPI00018873;IPI00472879
## 6                                    IPI00007797;IPI00737213;IPI00788781
##                       Sequence Length Missed.Cleavages Charges PEP  Score
## 1  SIVDYKPNIDIIEQQHQIIQEAIIFDNK     28                1   3,4,5   0 438.07
## 2       EDGNEEDKENQGDETQGQQPPQR     23                1 2,3,4,5   0 423.61
## 3   SKDDQVTVIGAGVTIHEAIAAAEIIKK     27                2   3,4,5   0 423.09
## 4      WNTEDKVSHVSTGGGASIEIIEGK     24                1   2,3,4   0 419.11
## 5 TPAGNFVTIEEGKGDIEEYGQDIIHTVFK     29                1     3,4   0 418.08
## 6        KTQTVCNFTDGAIVQHQEWDGK     22                1     3,4   0 414.66
##       Mass Intensity.U2OS_1 Intensity.U2OS_2 Intensity.U2OS_3     Mean
## 1 3323.740         7.778470         8.359893         8.664840 8.267734
## 2 2627.097               NA         8.003762         7.203930 7.603846
## 3 2776.534         7.435526               NA               NA 7.435526
## 4 2513.240         6.081275               NA               NA 6.081275
## 5 3206.577         7.979380         7.092405         8.263778 7.778521
```

```
## 6 2561.197         7.438574         7.483673               NA 7.461123
##        Max master_protein protein_length
## 1 8.664840         O43707            911
## 2 8.003762         P67809            324
## 3 7.435526         P29401            623
## 4 6.081275  P07205;P00558        417;417
## 5 8.263778         P43490            491
## 6 7.483673         Q01469            135
##                         protein_description crap_protein unique
## 1                      sp|O43707|ACTN4_HUMAN            0      1
## 2                      sp|P67809|YBOX1_HUMAN            0      1
## 3                        sp|P29401|TKT_HUMAN            0      1
## 4 sp|P07205|PGK2_HUMAN;sp|P00558|PGK1_HUMAN            0      0
## 5                      sp|P43490|NAMPT_HUMAN            0      1
## 6                      sp|Q01469|FABP5_HUMAN            0      1
```

```
dim(u2os) # 68621 21
```

```
## [1] 68621    21
```

```
# Filter data - keep peptides with unique master proteins, those which are not "crap" and those that ar
u2os.filt1 = u2os[which(u2os$unique == 1 & u2os$master_protein != "" & u2os$crap_protein != 1),]

# Keep only those columnns with data of interest
# Geiger et al produced a triplicate MS dataset for the U2OS cell line
u2os.filt2 = u2os.filt1[,c(5:6,17,11:16,18:19)]
head(u2os.filt2)
```

```
##                       Sequence Length master_protein     Mass
## 1  SIVDYKPNIDIIEQQHQIIQEAIIFDNK     28         O43707 3323.740
## 2       EDGNEEDKENQGDETQGQQPPQR     23         P67809 2627.097
## 3   SKDDQVTVIGAGVTIHEAIAAAEIIKK     27         P29401 2776.534
## 5 TPAGNFVTIEEGKGDIEEYGQDIIHTVFK     29         P43490 3206.577
## 6        KTQTVCNFTDGAIVQHQEWDGK     22         Q01469 2561.197
## 7      SQGISQIYHNQSQGIISQIQGQSK     24         P78344 2628.326
##   Intensity.U2OS_1 Intensity.U2OS_2 Intensity.U2OS_3     Mean      Max
## 1         7.778470         8.359893         8.664840 8.267734 8.664840
## 2               NA         8.003762         7.203930 7.603846 8.003762
## 3         7.435526               NA               NA 7.435526 7.435526
## 5         7.979380         7.092405         8.263778 7.778521 8.263778
## 6         7.438574         7.483673               NA 7.461123 7.483673
## 7               NA         7.487958         7.587711 7.537834 7.587711
##   protein_length   protein_description
## 1            911 sp|O43707|ACTN4_HUMAN
## 2            324 sp|P67809|YBOX1_HUMAN
## 3            623   sp|P29401|TKT_HUMAN
## 5            491 sp|P43490|NAMPT_HUMAN
## 6            135 sp|Q01469|FABP5_HUMAN
## 7            907 sp|P78344|IF4G2_HUMAN
```

```
# Checking for peptide loss. We loose ~ 400 peptides out of nearly 70,000 so not too concerned
dim(u2os) # 68621 21
```

```
## [1] 68621    21
```

```
dim(u2os.filt1) # 64967 21
```

```
## [1] 64967    21
```

```
dim(u2os.filt2) # 64967 11
```

```
## [1] 64967    11
```

```
# Create an MSnSet object of the background list
u2os.samp = data.frame(sample=c("Intensity.U2OS_1","Intensity.U2OS_2","Intensity.U2OS_3"),rep=c(1,2,3))
rownames(u2os.samp) = u2os.samp$sample
res.u2os <- MSnSet(exprs = as.matrix(u2os.filt2[,c(5:7)]),fData=u2os.filt2[,-c(5:7)],pData = u2os.samp)

# Impute missing data in the background list from Geiger et al.
# Haven't drawn any plots for this. Can do it at a later date
impute.u2os <- impute(res.u2os,method = "knn")
```

```
## Cluster size 36434 broken into 28537 7897
## Cluster size 28537 broken into 7812 20725
## Cluster size 7812 broken into 4633 3179
## Cluster size 4633 broken into 3045 1588
## Cluster size 3045 broken into 1231 1814
## Done cluster 1231
## Cluster size 1814 broken into 860 954
## Done cluster 860
## Done cluster 954
## Done cluster 1814
## Done cluster 3045
## Cluster size 1588 broken into 758 830
## Done cluster 758
## Done cluster 830
## Done cluster 1588
## Done cluster 4633
## Cluster size 3179 broken into 1044 2135
## Done cluster 1044
## Cluster size 2135 broken into 1220 915
## Done cluster 1220
## Done cluster 915
## Done cluster 2135
## Done cluster 3179
## Done cluster 7812
## Cluster size 20725 broken into 18245 2480
## Cluster size 18245 broken into 9841 8404
## Cluster size 9841 broken into 3640 6201
## Cluster size 3640 broken into 1877 1763
## Cluster size 1877 broken into 788 1089
## Done cluster 788
## Done cluster 1089
## Done cluster 1877
## Cluster size 1763 broken into 775 988
## Done cluster 775
## Done cluster 988
## Done cluster 1763
## Done cluster 3640
## Cluster size 6201 broken into 3330 2871
## Cluster size 3330 broken into 2482 848
## Cluster size 2482 broken into 1250 1232
```

```
## Done cluster 1250
## Done cluster 1232
## Done cluster 2482
## Done cluster 848
## Done cluster 3330
## Cluster size 2871 broken into 1561 1310
## Cluster size 1561 broken into 350 1211
## Done cluster 350
## Done cluster 1211
## Done cluster 1561
## Done cluster 1310
## Done cluster 2871
## Done cluster 6201
## Done cluster 9841
## Cluster size 8404 broken into 5706 2698
## Cluster size 5706 broken into 2228 3478
## Cluster size 2228 broken into 1013 1215
## Done cluster 1013
## Done cluster 1215
## Done cluster 2228
## Cluster size 3478 broken into 1269 2209
## Done cluster 1269
## Cluster size 2209 broken into 1207 1002
## Done cluster 1207
## Done cluster 1002
## Done cluster 2209
## Done cluster 3478
## Done cluster 5706
## Cluster size 2698 broken into 1450 1248
## Done cluster 1450
## Done cluster 1248
## Done cluster 2698
## Done cluster 8404
## Done cluster 18245
## Cluster size 2480 broken into 1298 1182
## Done cluster 1298
## Done cluster 1182
## Done cluster 2480
## Done cluster 20725
## Done cluster 28537
## Cluster size 7897 broken into 4366 3531
## Cluster size 4366 broken into 1923 2443
## Cluster size 1923 broken into 1388 535
## Done cluster 1388
## Done cluster 535
## Done cluster 1923
## Cluster size 2443 broken into 1369 1074
## Done cluster 1369
## Done cluster 1074
## Done cluster 2443
## Done cluster 4366
## Cluster size 3531 broken into 1164 2367
## Done cluster 1164
## Cluster size 2367 broken into 1331 1036
```

```
## Done cluster 1331
## Done cluster 1036
## Done cluster 2367
## Done cluster 3531
## Done cluster 7897
```

```
# Aggregate Geiger et al data from peptides to proteins
# 64967 peptides are aggregated to 7507 proteins
agg.u2os = combineFeatures(impute.u2os,groupBy = fData(impute.u2os)$master_protein,cv = T,fun = "median
dim(agg.u2os)
```

```
## [1] 7507    3
```

There are 7507 proteins in the U20S cell line, across 3 replicates, based on the Geiger et al., dataset. The next step is to annotate this gene list with the various functional categories that we want to perform enrichment analysis for.

```
#-------------------------------------------------------------------------------------------------------
# Step 06: Mapping 'background' as well as 'expressed' list of proteins to various annotations
# Used a few different packages - wanted to settle on getBM from 'biomaRt' as output in easy to use for
# Except that it was extremely slow. Hence used 'queryMany' from mygene and will reformat data.
#-------------------------------------------------------------------------------------------------------

#---------------------------------------------------
# Mapping Geiger U2OS list of proteins to GO terms
#---------------------------------------------------

# Geiger list being annotated with interpro descriptions - fast as done in chunks
# 7507 uniprot IDs yield 7574 lines of data
library(mygene)
geiger.qm = queryMany(fData(agg.u2os)$master_protein,scopes="uniprot",fields=c("ensembl","name","symbol
```

```
## Finished
## Pass returnall=TRUE to return lists of duplicate or missing query terms.
```

```
geiger.qm$domains = sapply(sapply(geiger.qm$interpro,"[[",3),function(x) paste(x,collapse="; "))
geiger.qm$go.bp = sapply(sapply(geiger.qm$go.BP,"[[",2),function(x) paste(x,collapse="; "))
geiger.qm$go.mf = sapply(sapply(geiger.qm$go.MF,"[[",2),function(x) paste(x,collapse="; "))
geiger.qm$go.cc = sapply(sapply(geiger.qm$go.CC,"[[",2),function(x) paste(x,collapse="; "))
geiger.qm$go.all = paste(geiger.qm$go.bp,geiger.qm$go.cc,geiger.qm$go.mf,sep="; ")
#head(geiger.qm)

#-------------------------------------------------------------------------
# Mapping Trizol-enriched list of proteins to gene domains from Interpro.
#-------------------------------------------------------------------------
# 1744 uniprot IDs yield 1780 lines of data
uv.qm = queryMany(fData(qnt.prot)$Master.Protein.Accessions,scopes="uniprot",fields=c("ensembl","name",
```

```
## Finished
## Pass returnall=TRUE to return lists of duplicate or missing query terms.
```

```
uv.qm$domains = sapply(sapply(uv.qm$interpro,"[[",3),function(x) paste(x,collapse="; "))
uv.qm$go.bp = sapply(sapply(uv.qm$go.BP,"[[",2),function(x) paste(x,collapse="; "))
uv.qm$go.mf = sapply(sapply(uv.qm$go.MF,"[[",2),function(x) paste(x,collapse="; "))
uv.qm$go.cc = sapply(sapply(uv.qm$go.CC,"[[",2),function(x) paste(x,collapse="; "))
uv.qm$go.all = paste(uv.qm$go.bp,uv.qm$go.cc,uv.qm$go.mf,sep="; ")
#head(uv.qm)
```

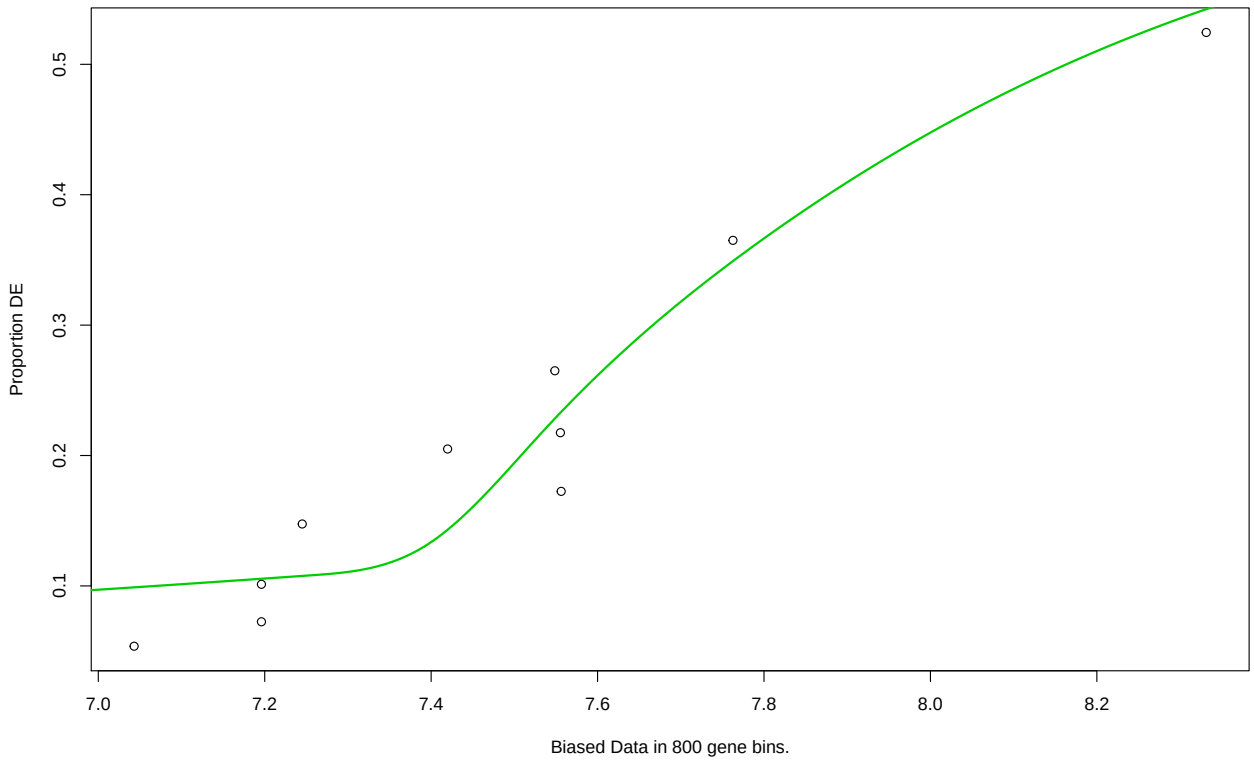
Now that we have mapped genes to annotations, time for some enrichment analysis.

```
#-------------------------------------------------------------------------------------------------------------
# Step 07: Performing enrichment analysis using 'goseq' package
# We use GO terms and Interpro domains for enrichment analysis
#-------------------------------------------------------------------------------------------------------------

# -----------------------------------------------------------------------
# Function  : runGoseq
# Aim       : runs goseq on a list of genes
# Input     : list of genes
# Output    : enriched list of Interpro domains
# -----------------------------------------------------------------------

runGoseq <- function(genelist,bglist,bias.dat=NULL,cat.oligo){

  # setting up goseq object
  all.genes.comp = rep(0,nrow(bglist))
  names(all.genes.comp) = rownames(bglist)
  all.genes.comp[which(names(all.genes.comp) %in% unique(genelist))] = 1
  table(all.genes.comp)

  # Remove missing values
  comp.no.missing = all.genes.comp[which(!is.na(names(all.genes.comp)))]
  table(comp.no.missing)


  # Running the function to calculate weights. We have no bias information as we did in UV experiment
  # This is because mass spec was run in detection mode not quantitation mode.
  pwf.comp = nullp(comp.no.missing,'hg19','geneSymbol', bias=bias.dat,plot.fit=TRUE)

  # goseq enrichment with domains for cross-linked samples
  goseq.comp = goseq(pwf.comp,gene2cat = cat.oligo)
  goseq.comp$BH_over_represented_pvalue = p.adjust(goseq.comp$over_represented_pvalue,method = "BH")
  goseq.comp

  enriched.goseq.comp = goseq.comp[which(goseq.comp$BH_over_represented_pvalue <= 0.05),]
  return(list(goseq.comp,enriched.goseq.comp))

}

#-----------------------------------------------------
# Using Goseq with protein abundance as a bias....
#-----------------------------------------------------

# Want to be able to use both UniProt and Gene symbols as references for bias in downstream analysis
bias.df = data.frame(protbias = rowMax(exprs(agg.u2os)),UNIPROT = fData(agg.u2os)$master_protein,SYMBOL=
head(bias.df)
```

```
##   protbias UNIPROT SYMBOL
## 1 7.196066  A0AVF1  IFT56
## 2 7.555342  A0AVT1   UBA6
## 3 7.196066  A0FGR8  ESYT2
## 4 7.196066  A0JLT2  MED19
## 5 7.196066  A0JNW5  UH1BL
```

```
## 6 6.629695  A0MZ66  SHOT1
```

```
# Converting Geiger et al domains data to a list with each line only having one domain
geiger.doms = data.frame(geiger.qm[,c("query","domains")])
geiger.gos = data.frame(geiger.qm[,c("query","go.all")])

library(data.table)
d.dt <- data.table(geiger.doms, key="query")
geiger.cat <- d.dt[, list(domains = unlist(strsplit(domains, "; "))), by=query]

go.dt = data.table(geiger.gos, key="query")
geiger.cat.go <- go.dt[, list(go.terms = unlist(strsplit(go.all, "; "))), by=query]

# Setting up a 'genes' vector. We've only kept those genes present in u2os as the universe
# Then we mark all those that are enriched in the analysis as genes of interest (DE if you will)
# We have 1435 genes out of 1516 that have bias data and go terms

# Running goseq with GO categories
rownames(bias.df) = bias.df$UNIPROT
uv.enrich.go = runGoseq(uv.qm$query,bias.df,bias.df$protbias,geiger.cat.go)
# uv.enrich.go = uv.enrich.go[order(uv.enrich.go$BH_over_represented_pvalue),]

# Running goseq with Interpro domains
uv.enrich.interpro = runGoseq(uv.qm$query,bias.df,bias.df$protbias,geiger.cat)
```




```
# Writing output to text files
write.table(data.frame(uv.enrich.go[[2]]),paste(outdir,"Trizol-UV-dosage_GO-enrichment.txt",sep="/"),se
write.table(data.frame(uv.enrich.interpro[[2]]),paste(outdir,"Trizol-UV-dosage_Interpro-enrichment.txt"
```

The protein "universe" used here is the list of all proteins discovered in Geiger et al., 2012. The list of peptides was mapped to proteins by Tom and annotated with master protein identifiers, crap proteins and

uniqueness. This was used as it is the most comprehensive list of U2OS proteins mapped using mass spec to date. After filtering, there were ~7500 proteins generated in the study.

The "DE list" or "enriched" list of proteins are those that were expressed in the UV dosage experiment across all 9 samples. This yields 1744 proteins of which ~1500 could be mapped to GO identifiers. I tried doing this mapping with both 'bioMart' and 'MyGene' databases available as packages within R. The latter was phenomenally faster, hence I proceeded with it.

For GO enrichment, I used the 'goseq' package which accounts for any bias (here abundance of protein) and then checks for enrichment. 'goseq' yielded 94 terms of which 40 were BP, 36 were CC and 18 were MF terms. Of the BP (Biological Process) terms, more than half were involved in RNA processing and translation.

Would like to map each of the proteins to PFAM/SMART domains to see if they are indeed RNA binding proteins..... Have mapped RNA BP domains to both Geiger and UV.dosage. Need to look at the genes involved for which we have further evidence that they are indeed RBPs

The domain "Nucleotide-bd_a/b_plait" is present in almost all (16/18) heterogeneous nuclear ribonucleoproteins whose main task is to move mRNA out of the nucleus. This domain has an interpro entry IPR012677 which is no longer valid. This domain is also present in nucleolin and EWSR1. This might be the "RGG-box" domain eluded to in Burd and Dreyfuss, 1994. Excitingly, "Nucleotide-bd_a/b_plait" is a top domain the goseq analysis. I will substitute hnRNP searches with this term.

"Ig-like"/"Ig_sub" from Tom's notes are indicative of glycoproteins which might be RNA-binding. There are 58 Ig term related proteins in the UV-dosage experiments of which only 2 have RNA-binding domains indicating only a small fraction have RNA binding capapbilities but majority of them are involved in other functions.

```
#-----------------------------------------------------------------------
# 08 : Mapping oligodT data to domains and looking for enrichment
#-----------------------------------------------------------------------

oligo.cl.in.nc = as.character(readRDS("Input/CL_proteins.rds")) # 652 unique Swissprot IDs from Tom's p
#oligo.cl.in.nc = read.table("Input/Leicester-oligodT-CL-prot.txt",header=T,sep="\t") # 328 gene symbol

oligo.nc = as.character(readRDS("Input/NCL_proteins.rds")) # 175 unique Swissprot IDs from Tom's peptid
#oligo.nc = read.table("Input/Leicester-oligodT-NC-prot.txt",header=T,sep="\t") # 73 gene symbols from

length(oligo.cl.in.nc)
```

```
## [1] 652
```

```
length(oligo.nc)
```

```
## [1] 175
```

```
# Present in non-crosslinked and hence will be removed

oligo.cl = oligo.cl.in.nc[-which(oligo.cl.in.nc %in% oligo.nc)]
length(oligo.cl) # 489 unique Swissprot IDs from Tom's files not present in non-crosslinked samples
```

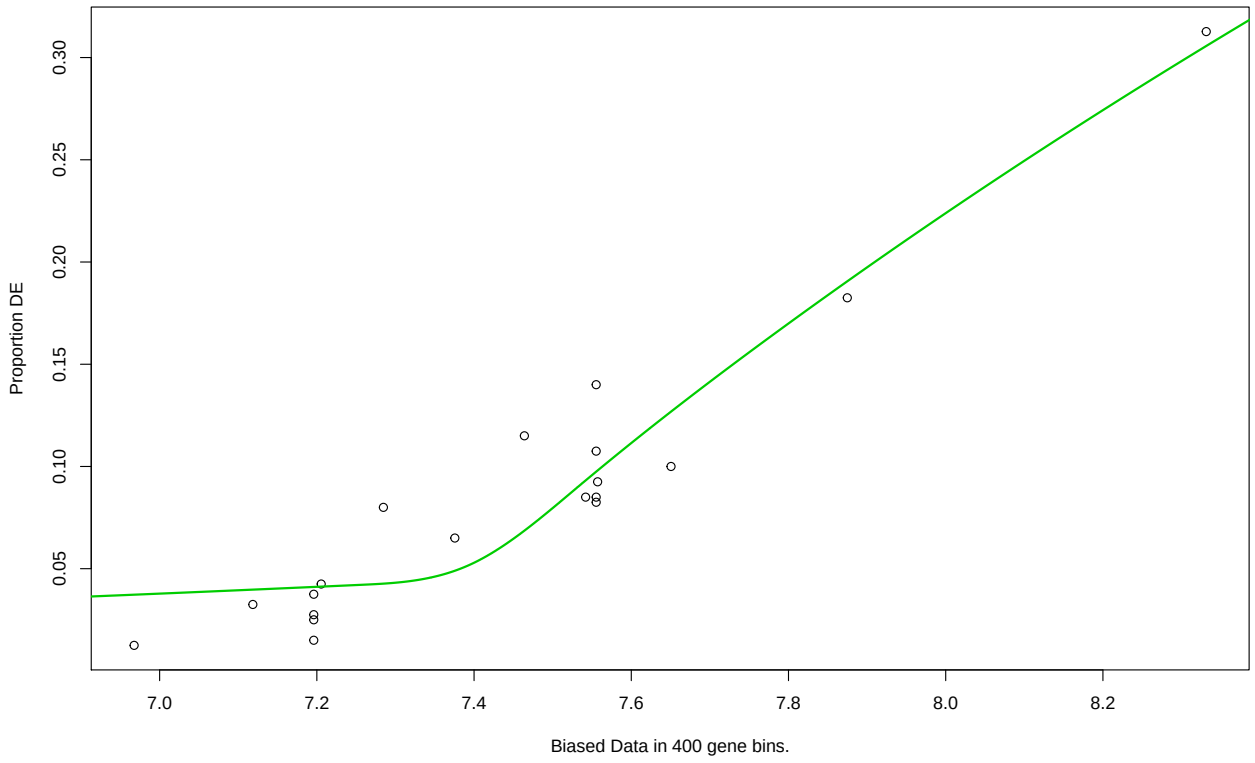
```
## [1] 489
```

```
# Which genes are present in both cross and non-crosslinked samples
cl.and.nc.prot = oligo.cl.in.nc[which(oligo.cl.in.nc %in% oligo.nc)]
length(cl.and.nc.prot) # 163 present in both crosslinked and non-crosslinked samples
```

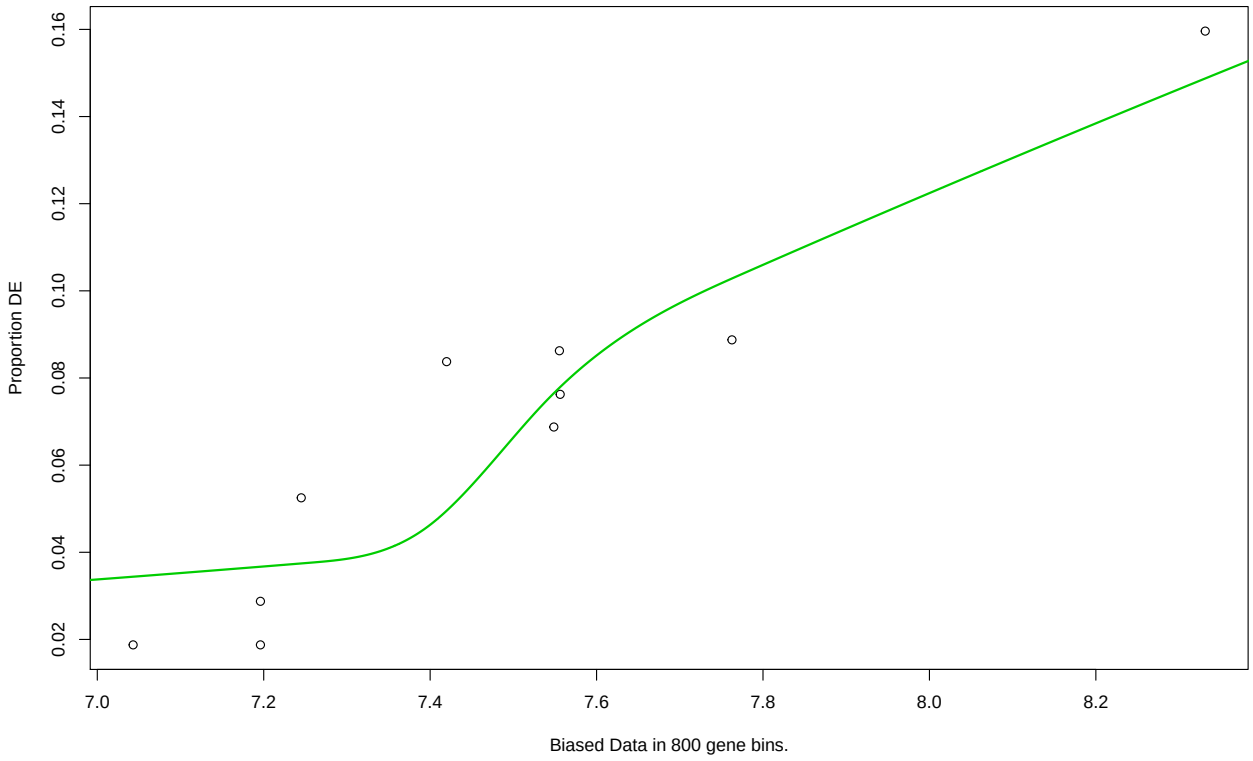
```
## [1] 163
```

```
# Convert data table
# geiger.doms.sym = data.frame(geiger.qm[,c("symbol","domains")])
```

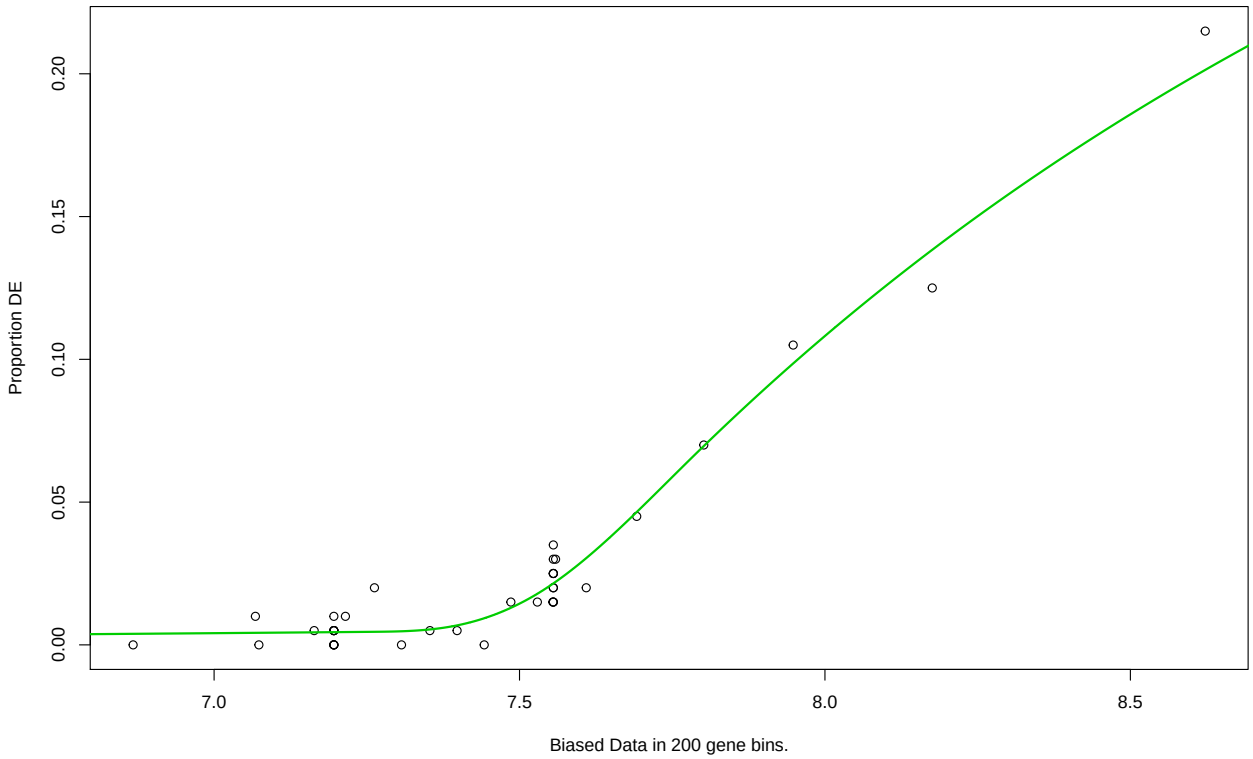
```
# geiger.gos.sym = data.frame(geiger.qm[,c("symbol","go.all")])

# d.dt.sym <- data.table(geiger.doms.sym, key="symbol")
# geiger.cat.sym <- d.dt.sym[, list(domains = unlist(strsplit(domains, "; "))), by=symbol]

# go.dt.sym = data.table(geiger.gos.sym, key="symbol")
# geiger.cat.go.sym <- go.dt.sym[, list(go.terms = unlist(strsplit(go.all, "; "))), by=symbol]

# Setting up a 'genes' vector. We've only kept those genes present in u2os as the universe
# Then we mark all those that are enriched in the analysis as genes of interest (DE if you will)
# We have 1435 genes out of 1516 that have bias data and go terms

# Running goseq for oligodT which have gene symbols as identifiers
# rownames(bias.df) = bias.df$SYMBOL

# Crosslinked oligodT all proteins
cl.all.enrich.go = runGoseq(oligo.cl.in.nc,bias.df,bias.df$protbias,geiger.cat.go)
cl.all.enrich.interpro = runGoseq(oligo.cl.in.nc,bias.df,bias.df$protbias,geiger.cat)
```



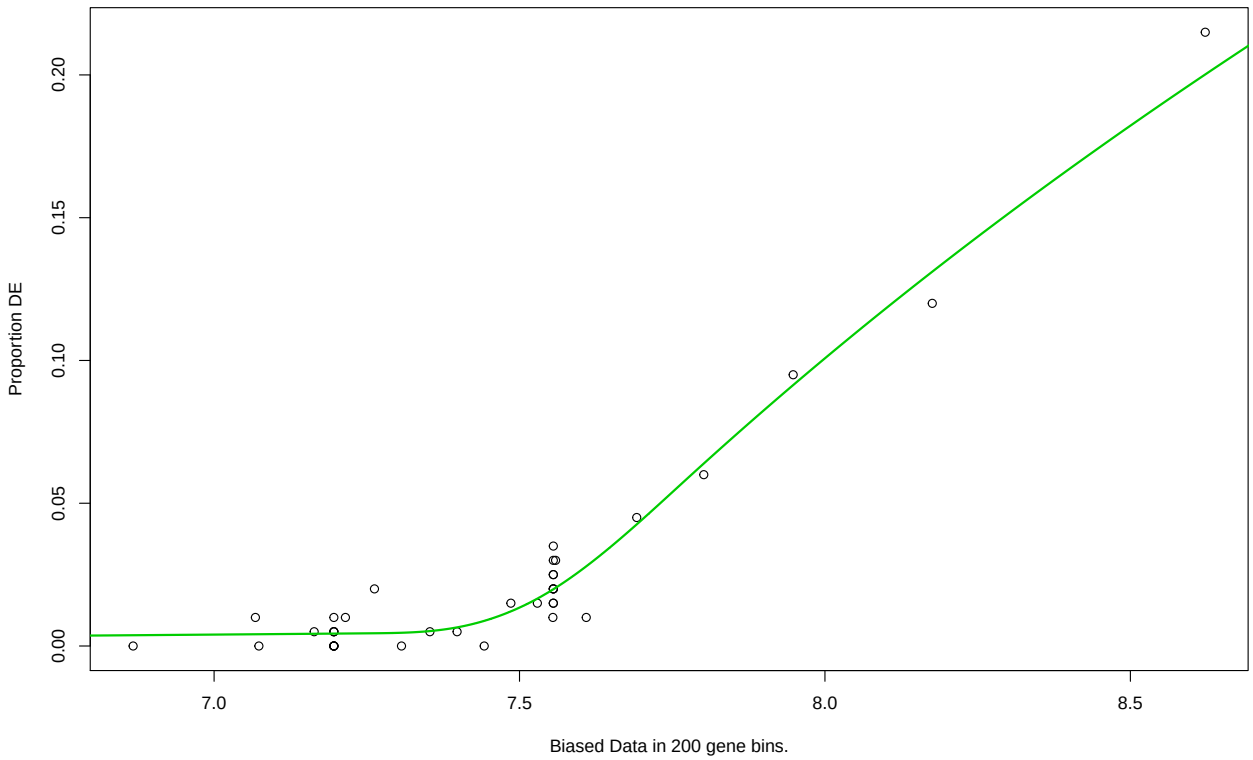

```
write.table(cl.all.enrich.go[[2]][,c(1,6:7,4:5,2,8)],paste(outdir,"oligodT-crosslinked-with-nc-prots_GO-
write.table(cl.all.enrich.interpro[[2]],paste(outdir,"oligodT-crosslinked-with-nc-prots_Interpro-enrichm

# Crosslinked oligodT minus proteins in non-crosslined samples
cl.enrich.go = runGoseq(oligo.cl,bias.df,bias.df$protbias,geiger.cat.go)
cl.enrich.interpro = runGoseq(oligo.cl,bias.df,bias.df$protbias,geiger.cat)
```




```
write.table(cl.enrich.go[[2]][,c(1,6:7,4:5,2,8)],paste(outdir,"oligodT-crosslinked_GO-enrichment.txt",s
write.table(cl.enrich.interpro[[2]],paste(outdir,"oligodT-crosslinked_Interpro-enrichment.txt",sep="/")

# Non-crosslinked samples
nc.enrich.go = runGoseq(oligo.nc,bias.df,bias.df$protbias,geiger.cat.go)
nc.enrich.interpro = runGoseq(oligo.nc,bias.df,bias.df$protbias,geiger.cat)
```




```
write.table(nc.enrich.go[[2]][,c(1,6:7,4:5,2,8)],paste(outdir,"oligodT-non-crosslinked_GO-enrichment.tx
write.table(nc.enrich.interpro[[2]],paste(outdir,"oligodT-non-crosslinked_Interpro-enrichment.txt",sep=

# Common to both cross-linked and non-crosslinked samples
nc.cl.enrich.go = runGoseq(cl.and.nc.prot,bias.df,bias.df$protbias,geiger.cat.go)
```



```
nc.cl.enrich.interpro = runGoseq(cl.and.nc.prot,bias.df,bias.df$protbias,geiger.cat)
write.table(nc.enrich.go[[2]][,c(1,6:7,4:5,2,8)],paste(outdir,"oligodT-cl-and-nc_GO-enrichment.txt",sep=
write.table(nc.enrich.interpro[[2]],paste(outdir,"oligodT-cl-and-nc_Interpro-enrichment.txt",sep="/"),s
```

In this first step, we are reading in oligodT data from one experiment with both crosslinked(cl) and non-crosslinked(nc) samples. We remove proteins from the 'cl' which were also present in 'nc' as we cannot comment on enrichment.

Interestingly, the oligodT data seems a lot "cleaner" than trizol dataset. By this I mean, the top domains are most definitely all known RNA-binding domains based on literature looking at RBPs (Lunde 2007, Burd 1994). In the Trizol data we get "Ig-like" domains as one of the top hits which we don't see at all in the oligodT data.

Between crosslinked and non-crosslinked samples, we still see some overlap in that we are getting RNA-binding domains and proteins in non-crosslinked samples but the number of such proteins in a LOT lower in non-crosslinked than in crosslinked samples.

```
#-------------------------------------------------------------------------------------------------------
# Step 09 : Annotating all protein lists with counts of RBD domains
#-------------------------------------------------------------------------------------------------------

#-----------------------------------------------------------------------------
# Making a list of domains that define RNA binding proteins
# Couldn't find PIWI/PAZ domains in the UV gene list but present in Geiger list
# Looking at Burd paper which includes 'RGG' box which I think is HnRNP as well as cold_shock domain
# Can't find TRAP protein (trp RNA-Binding attenuation protein)
# Have added new RBDs from Castello's 2016 paper - Thioredoxin, PDZ, DZF,
#-----------------------------------------------------------------------------

rbd.doms = c("RRM_dom","KH_dom","dsRBD_dom","Znf_CCCH","Znf_C2H2","Znf_CCHC","S1_dom","PAZ_dom","Piwi",
rbd.doms
```

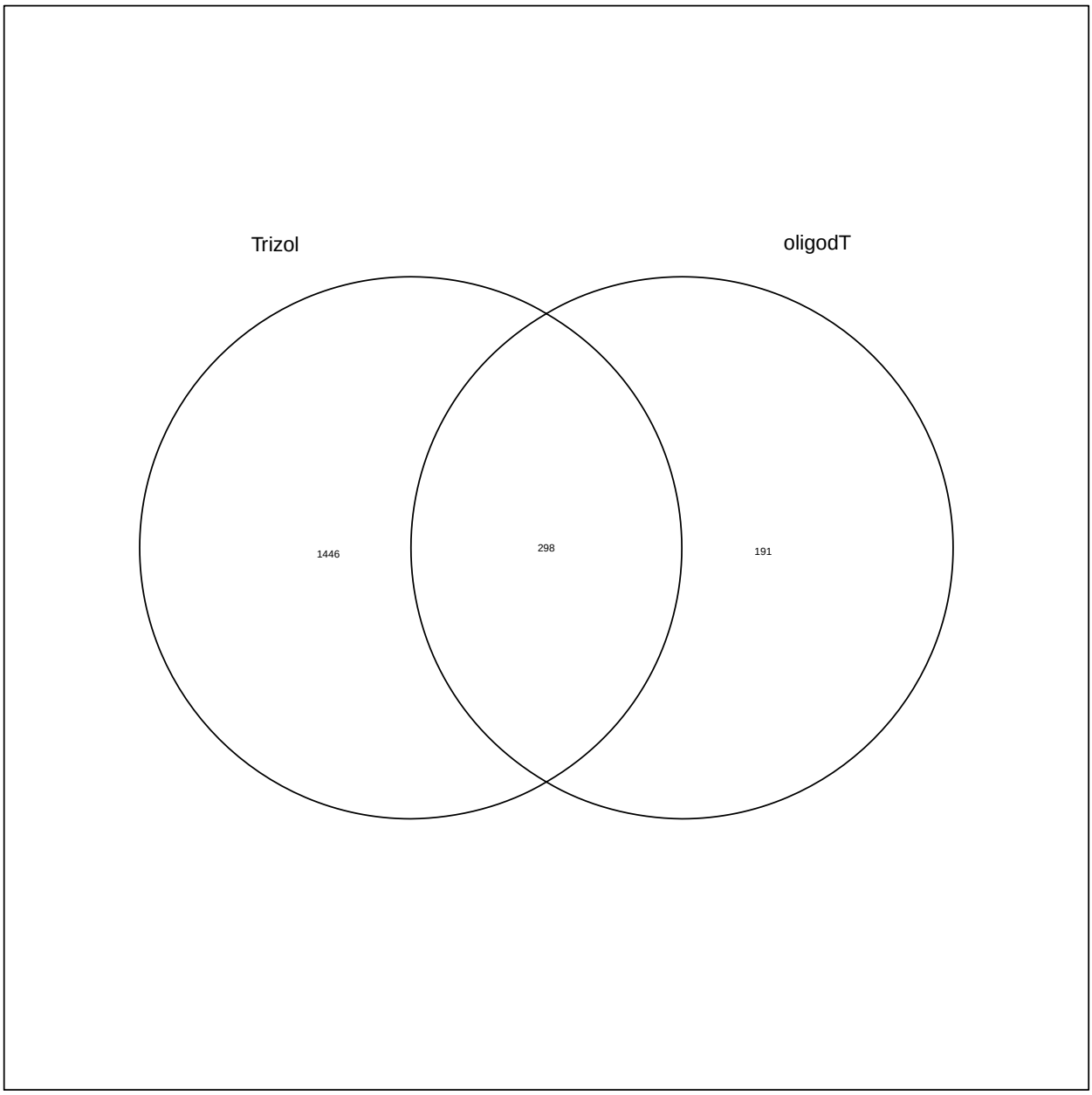
```
##  [1] "RRM_dom"                "KH_dom"
##  [3] "dsRBD_dom"              "Znf_CCCH"
##  [5] "Znf_C2H2"               "Znf_CCHC"
##  [7] "S1_dom"                 "PAZ_dom"
##  [9] "Piwi"                   "Nucleotide-bd_a/b_plait"
## [11] "CSD"                    "Cold-shock_CS"
## [13] "Pumilio_RNA-bd_rpt"     "SAM;"
## [15] "SAM/pointed"            "DEAD"
## [17] "Thioredoxin"            "PDZ"
## [19] "DZF_dom"
```

```
# Annotating both geiger and uv-dosage datasets with known "rbd" domains and how many of these RBD doma
uv.qm$num.rbds = rowSums(sapply(rbd.doms, function(x) grepl(x, uv.qm$domains)))
uv.qm$which.rbd = apply(sapply(rbd.doms, function(x) grepl(x, uv.qm$domains)),1, function(y) paste(name
#head(uv.qm[which(uv.qm$num.rbds != 0),],n=10)

geiger.qm$num.rbds = rowSums(sapply(rbd.doms, function(x) grepl(x, geiger.qm$domains)))
geiger.qm$which.rbd = apply(sapply(rbd.doms, function(x) grepl(x, geiger.qm$domains)),1, function(y) pa
#head(geiger.qm[which(geiger.qm$num.rbds != 0),],n=10)


# Printing what percentage  of each list is annotated with RNA binding domains
obj.names = c("oligo.cl.in.nc","oligo.cl","oligo.nc","cl.and.nc.prot")
count = 0
```

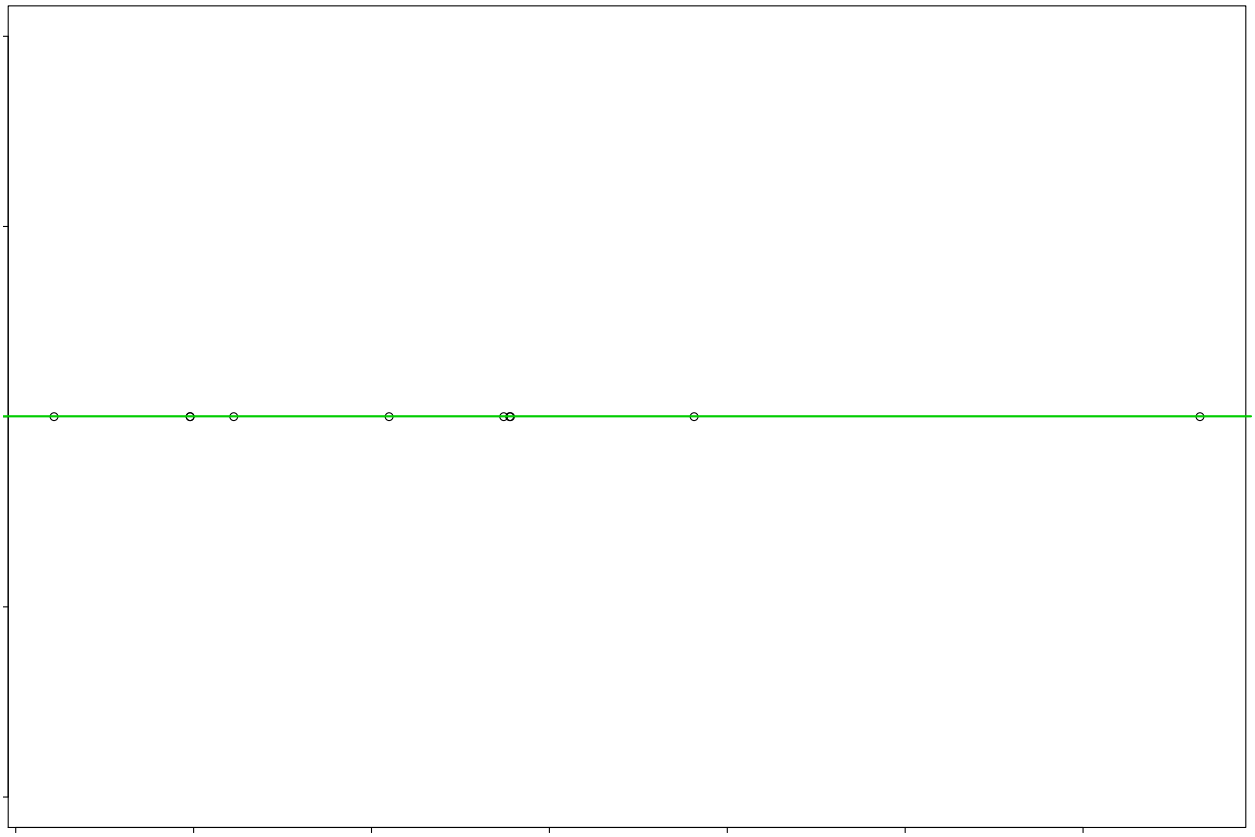
```
for(y in list(oligo.cl.in.nc,oligo.cl,oligo.nc,cl.and.nc.prot)){
  count = count+1
  y = queryMany(y,scopes="uniprot",fields=c("ensembl","name","symbol","interpro","go"))
  y$domains = sapply(sapply(y$interpro,"[[",3),function(x) paste(x,collapse="; "))
  y$num.rbds = rowSums(sapply(rbd.doms, function(x) grepl(x, y$domains)))
  y$which.rbd = apply(sapply(rbd.doms, function(x) grepl(x, y$domains)),1, function(z) paste(names(whic
  print(table(y$num.rbds))
  print(paste("Percentage of proteins with RNA-binding domains in ", obj.names[count]," = ",round(100*su
}
```

```
## Finished
## Pass returnall=TRUE to return lists of duplicate or missing query terms.
##
##   0   1   2   3
## 449 107  99  13
## [1] "Percentage of proteins with RNA-binding domains in oligo.cl.in.nc = 32.78"
## Finished
## Pass returnall=TRUE to return lists of duplicate or missing query terms.
##
##   0   1   2   3
## 338  83  63  10
## [1] "Percentage of proteins with RNA-binding domains in oligo.cl = 31.58"
## Finished
## Pass returnall=TRUE to return lists of duplicate or missing query terms.
##
##   0   1   2   3
## 119  27  37   3
## [1] "Percentage of proteins with RNA-binding domains in oligo.nc = 36.02"
## Finished
## Pass returnall=TRUE to return lists of duplicate or missing query terms.
##
##   0   1   2   3
## 111  24  36   3
## [1] "Percentage of proteins with RNA-binding domains in cl.and.nc.prot = 36.21"
```

```
#-------------------------------------------------------------------------
# 10 : Looking at the intersect of proteins between oligodT and Trizol
#-------------------------------------------------------------------------

# Using Trizol mapped to hgnc_symbol as oligodT is only in symbols.
library(venn)
Trizol = unique(uv.qm$query)
oligodT = unique(oligo.cl)
v = venn(list(Trizol=Trizol,oligodT=oligodT),intersections=T)
```




```
both = attr(v,"intersection")$`Trizol:oligodT` # n = 298
oligo.only = attr(v,"intersection")$`oligodT` # n = 191
trizol.only = attr(v,"intersection")$`Trizol` # n = 1446

# Enrichment for overlaps and setdiffs
m = c("both","oligo-only","trizol-only")
c = 0
for(t in list(both=both,oligo.only=oligo.only,trizol.only=trizol.only)){
  c=c+1
  t.enrich.go = runGoseq(t,bias.df, bias.df$protbias,geiger.cat.go)
  t.enrich.interpro = runGoseq(t,bias.df, bias.df$protbias,geiger.cat)
  write.table(t.enrich.go[[2]][,c(1,6:7,4:5,2,8)],paste(outdir,paste(m[c],"-genes-GO-enrichment.txt",sep
  write.table(t.enrich.interpro[[2]],paste(outdir, paste(m[c],"-genes-Interpro-enrichment.txt",sep=""),s
}
```

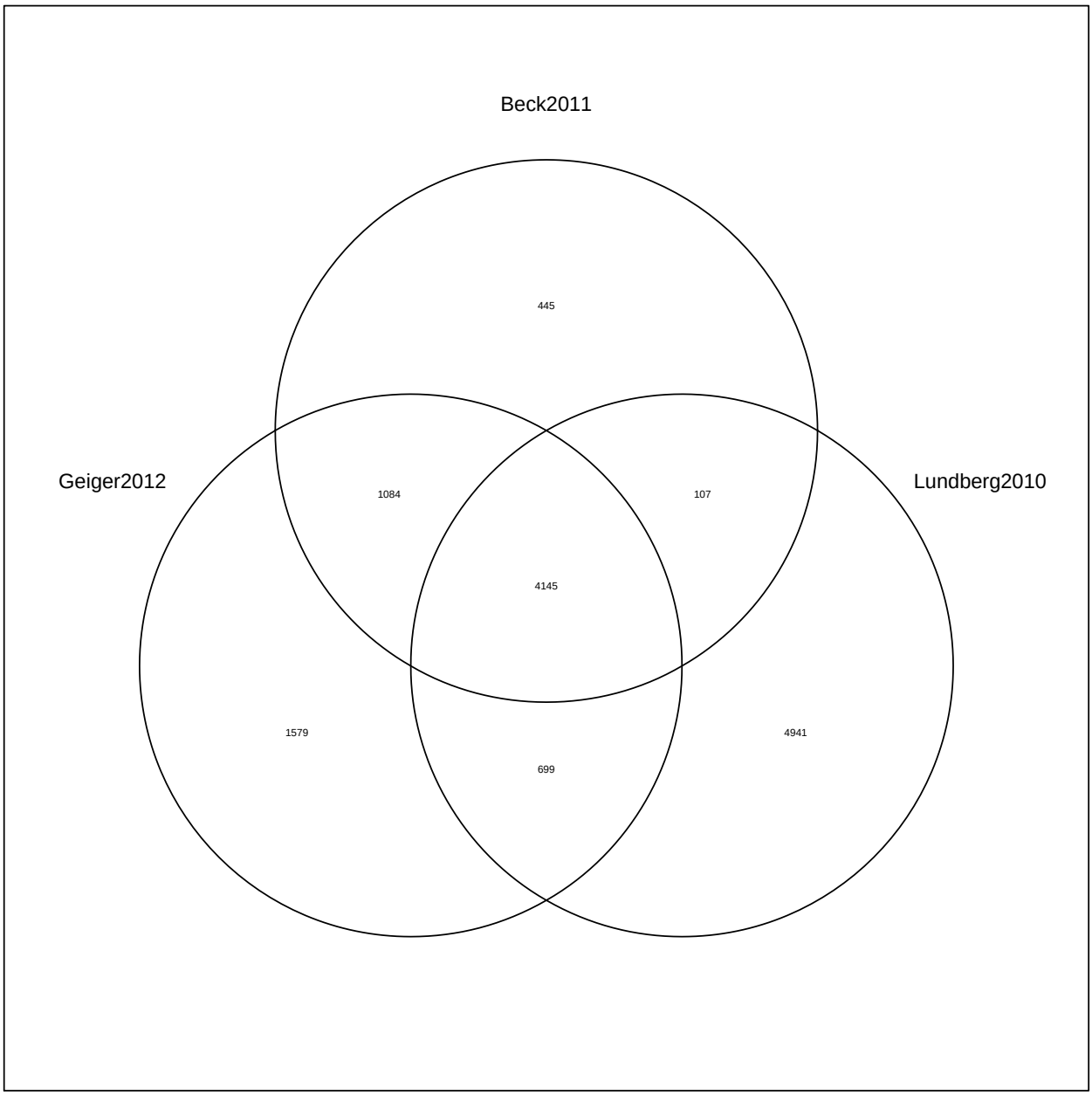


```
oo = intersect(oligo.only,geiger.qm$query) # 170/191
tt = intersect(trizol.only,geiger.qm$query) # 1144/1446
bl = intersect(both,geiger.qm$query) # 297/298 mising EIF3C/Q99613
```

```
#-----------------------------------------------------------------------------------------------------
# Step 11 : Comparing 3 different U2OS proteomics datasets from the current decade
#-----------------------------------------------------------------------------------------------------

geiger = as.character(unique(fData(agg.u2os)$master_protein))

beck = read.table("Input/Beck2011_n5781.txt")
beck = as.character(beck$V1)

lundberg.ens = read.table("Input/Lundberg2010_n5480.txt")
lundberg = unique(bitr(as.character(lundberg.ens$V1),fromType="ENSEMBL", toType="UNIPROT", OrgDb="org.Hs
length(lundberg)
```

```
## [1] 9892
```

```
library(venn)
library(gplots)
library(limma)

venn.u2os = venn(list(Geiger2012=geiger,Beck2011=beck,Lundberg2010=lundberg))
```



Small exercise on how much Geiger et al, Beck et al, and Lundberg et al., protein sets from Mass Spectrometery experiments overlap. With Lundberg et al., I had to map Ensembl IDs to UNIPROT and then do the comparison which caused a one-to-many mapping. The excess 4941 only found in Lundberg et al is almost exclusively due to the mapping of one Ensembl ID to many UNIPROT IDs. Geiger et al, being the most recent study does claim to have maximal protein mapping for U2OS. Beck et al.,

Should we use just the 4145 that overlaps as our high confident set or focus on Geiger as it the most recent ?