Influenza vaccine report

Vin BigData

9/25/2021

1. Introduction

This report is based on article *Transcriptomic profiling facilitates classification of response to influenza challenge*, and dataset *GSE61754*.

This dataset is about an influenza challenge study in which 22 healthy adults (11 vaccinated) were inoculated with H3N2 influenza. Genome-wide gene expression data from peripheral blood is taken immediately before the challenge and at 12, 24 and 48 h post-challenge.

2. Setting

2.1. Installing packages

You have to install all packages below, ignore these code if you have alrealy installed.

```
if (!requireNamespace("BiocManager", quietly=TRUE))
  install.packages("BiocManager")
BiocManager::install("GEOquery", force=TRUE)
BiocManager::install("org.Hs.eg.db", force=TRUE)
install.packages("rlang")
install.packages("ggplot2")
install.packages("gridExtra")
install.packages("reshape2")
install.packages("ggpubr")
install.packages("dplyr")
install.packages("GSVA")
install.packages("pheatmap")
install.packages("limma")
```

2.2. Setting packages

Load all packages.

```
library(GEOquery)
library(org.Hs.eg.db)
library(rlang)
library(ggplot2)
library(gridExtra)
library(reshape2)
library(ggpubr)
library(dplyr)
library(GSVA)
library(pheatmap)
library(limma)
```

3. Data packaging

3.1. Setting directory

folder_directory is the path of your Influenza project.

```
folder_directory <- [...]
setwd(folder_directory)
```

3.2. Downloading data

To get started with this analysis, download the file *GSE61754_series_matrix.txt.gz* available online from https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE61754, and file *GPL10558-50081.txt* available online from https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GPL10558

3.3. Loading data

Table *exprM*, *phenoM* give information about gene expression and phenotype of each sample.

```
geoFile <- "./Data/GSE61754_series_matrix.txt.gz"
GSE <- getGEO(filename = geoFile, GSEMatrix = T)

exprM <- exprs(GSE)
phenoM <- pData(GSE)
```

4. Data pre-processing

Before analyzing, we need to clean the dataset. As with table *platM*, Entrez gene ID or Symbol may not map one-to-one to gene ID. As a result, it is important to check for duplicated gene IDs.

This is the function that transfer the datatype of dataframe to numeric type.

```
getNumericMatrix <- function(data) {
  for(i in 1:(ncol(data))) {
    data[, i] <- as.numeric(as.character(data[, i]))
  }
  return(data)
}
```

We need to load table *platM* to get the mapping from gene ID to Symbol.

```
platMFile <- "./Data/GPL10558-50081.txt"
platM <- read.table(platMFile, header=TRUE, sep="\t", quote="", comment="#", fill=TRUE, stringsAsFactors=FALSE)

exprM <- cbind(rownames(exprM), exprM)
colnames(exprM)[1] <- "ID"
platM <- merge(platM[, c("ID", "Symbol")], exprM, by.x="ID", by.y=1)
geneIDs <- unique(platM$Symbol)
platM[3:90] <- getNumericMatrix(platM[3:90])
```

In this dataset, there are two samples that have NULL value for gene exprssion, so we will drop them. And for duplicate gene IDs, we will calculate the average value of gene expression.

```
processed_exprM <- aggregate(platM, by = list(platM$Symbol), mean)
processed_exprM$Group.1[1:11] <- c("", "DEC1", "MAR1", "MAR2", "MAR3", "MAR5", "MAR6", "MAR7", "7A5", "MAR8", "MAR9")
rownames(processed_exprM) <- processed_exprM$Group.1
processed_exprM <- processed_exprM[, -c(1, 2, 3)]
phenoM <- phenoM[!is.na(colSums(processed_exprM)), ]
processed_exprM <- processed_exprM[, !is.na(colSums(processed_exprM))]

final_exprM <- data.matrix(processed_exprM)

save(final_exprM, file="./Data/final_exprM.rda")
save(phenoM, file="./Data/phenoM.rda")
```

5. Analyzing

```
final_exprM <- get(load("./Data/final_exprM.rda"))
phenoM <- get(load("./Data/phenoM.rda"))
```

Rename index of samples.

```
colnames(final_exprM) <- phenoM[, "description"]
rownames(phenoM) <- phenoM[, "description"]
```

5.1. Check gene expression distribution

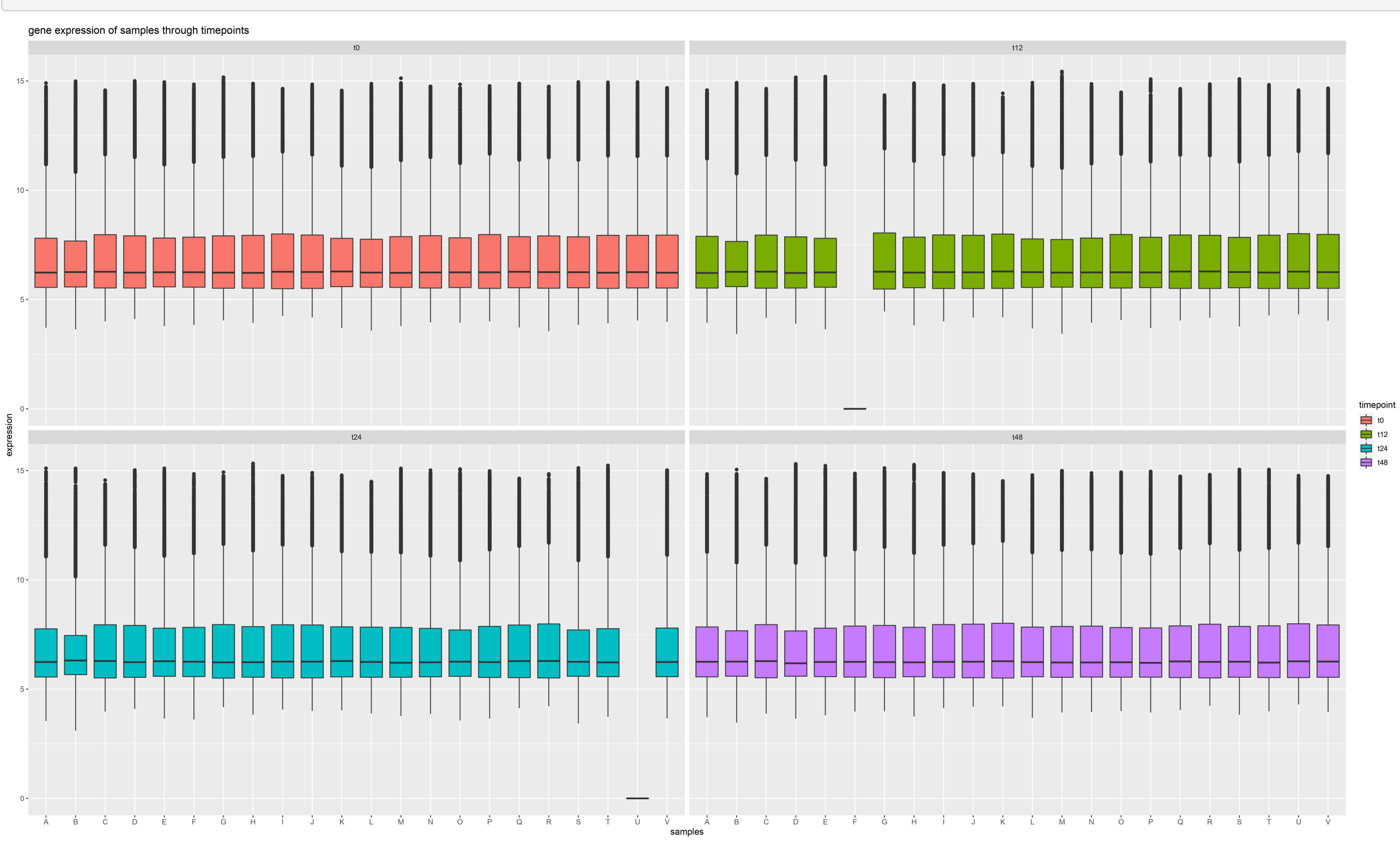
Plot gene expression distribution of samples, range of gene expression often between 5 and 8.

```
boxdf <- melt(final_exprM)
boxdf$sample <- substr(boxdf$Var2, 1, 1)
boxdf$timepoint <- substr(boxdf$Var2, 3, 5)
```

```
pdf("./Results/gene_expression_boxplot.pdf", height=14, width=24)
par(mar=c(10, 5, 1, 1))

ggplot(boxdf, aes(x=sample, y=value, fill=timepoint)) +
      geom_boxplot() +
      facet_wrap(~timepoint) +
      labs(title="gene expression of samples through timepoints",
           x="samples",
           y="expression",
           fill="timepoint")

dev.off()
```



5.2. PCA - Principal Component Analysis

Principal components plot:

- Using all genes.
- Using 5 genes: CCL2, LAMP3, RTP4, MT1G, OAS3 that be mentioned in original paper.
- Using 5000 genes (the most variable genes)

5.2.1. Using all genes

```
pcas <- prcomp(t(final_exprM))$x

sampleIDs <- rownames(pcas)
timepoints <- phenoM[sampleIDs,"timepoint:ch1"]
vaccine_status <- phenoM[sampleIDs, "vaccination status:ch1"]
severity_status <- phenoM[sampleIDs, "symptom severity:ch1"]

df <- as.data.frame(cbind(pcas[, c(1, 2)], timepoints, vaccine_status, severity_status))
df[, 1] <- as.numeric(df[, 1])
df[, 2] <- as.numeric(df[,2 ])
df[, 3] <- factor(df[, 3], levels=c("Pre-challenge", "12 hours post-challenge", "24 hours post-challenge", "48 hours post-challenge"))
df1 <- df
```

5.2.2. Using 5 genes: CCL2, LAMP3, RTP4, MT1G, OAS3

```
geneset <- c("CCL2", "LAMP3", "RTP4", "MT1G", "OAS3")

pcas <- prcomp(t(final_exprM[geneset, ]))$x

df <- as.data.frame(cbind(pcas[, c(1, 2)], timepoints, vaccine_status, severity_status))
df[, 1] <- as.numeric(df[, 1])
df[, 2] <- as.numeric(df[, 2])
df[, 3] <- factor(df[, 3], levels=c("Pre-challenge", "12 hours post-challenge", "24 hours post-challenge", "48 hours post-challenge"))
df2 <- df
```

5.2.3. Using 5000 genes

```
mostVar <- function(data, n, i_want_most_var=TRUE) {
  data.var <- apply(data, 1, stats::var)
  data[order(data.var, decreasing=i_want_most_var)[1:n], ]
}
mostVarGenes <- mostVar(final_exprM, 5000, i_want_most_var=TRUE)

pcas <- prcomp(t(mostVarGenes))$x

df <- as.data.frame(cbind(pcas[, c(1, 2)], timepoints, vaccine_status, severity_status))
df[, 1] <- as.numeric(df[, 1])
df[, 2] <- as.numeric(df[, 2])
df[, 3] <- factor(df[, 3], levels=c("Pre-challenge", "12 hours post-challenge", "24 hours post-challenge", "48 hours post-challenge"))
df3 <- df
```

5.2.4. Combining plots

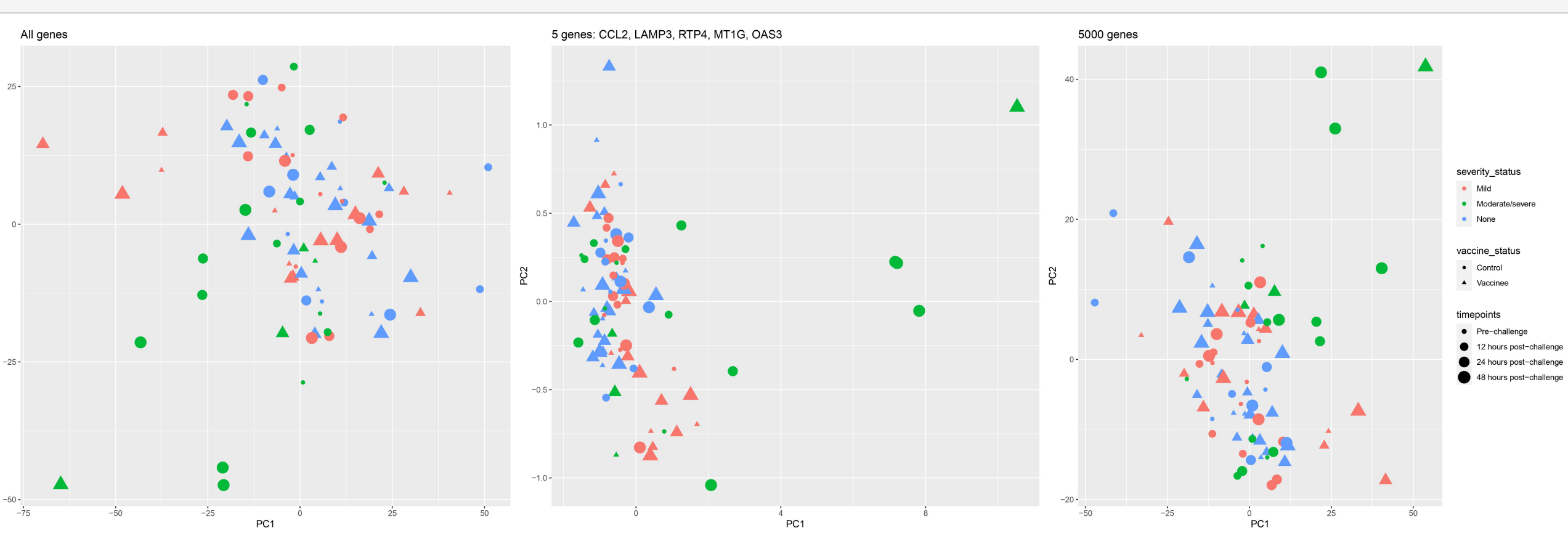
In 3 plots below, there are 4 samples separated from the remaining, which are P_t48, D_t48, T_t48 and N_t48. Phenotype of 4 samples are in below:

Sample	timepoints	severe_status	vaccine_status
D_t48	48	Moderate/severe	Control
N_t48	48	Moderate/severe	Vaccine
P_t48	48	Moderate/severe	Vaccine
T_48	48	Moderate/severe	Vaccine

```
pdf("./Results/pca2d_all.pdf", height=8, width=25)
par(mar=c(10, 5, 1, 1))

ggarrange(ggplot(df1, aes(x=PC1, y=PC2, colour=severity_status, size=timepoints, shape=vaccine_status)) +
          geom_point() + ggtitle("All genes") + theme(legend.position = "none"),
          ggplot(df2, aes(x=PC1, y=PC2, colour=severity_status, size=timepoints, shape=vaccine_status)) +
          geom_point() + ggtitle("5 genes: CCL2, LAMP3, RTP4, MT1G, OAS3") + theme(legend.position = "none"),
          ggplot(df3, aes(x=PC1, y=PC2, colour=severity_status, size=timepoints, shape=vaccine_status)) +
          geom_point() + ggtitle("5000 genes"),
          ncol = 3)

dev.off()
```



5.3. GenderQC:

Genes define gender, create a set of genes to help determine gender, include:

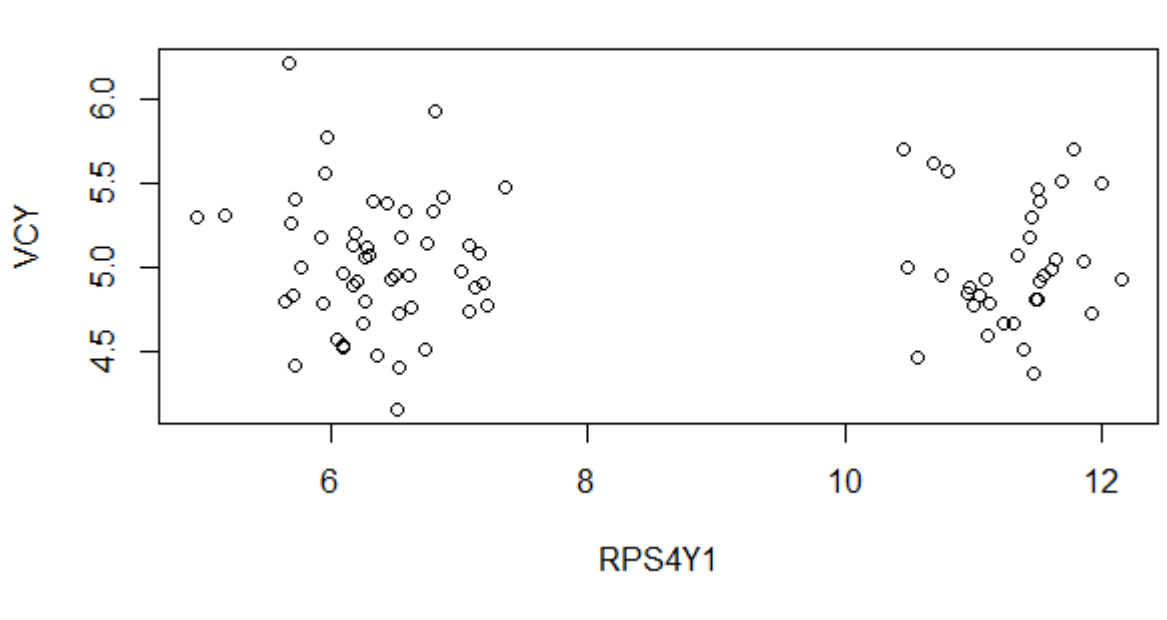
- On X chromosome: XIST, VCX
- On Y chromosome: RPS4Y1, DDX3Y, RPS4Y, UTY, SMCY, DBY, USP9Y, VCY

```
gender_genes <- c(“XIST”, “RPS4Y1”, “DDX3Y”, “RPS4Y”, “UTY”, “SMCY”,
                                  “DBY”, USP9Y”, “VCX”, VCY”)
gender_genes[gender_genes %in% rownames(final_exprM)]
```

Gender genes that appear in GSE61754 dataset are: XIST, RPS4Y1, DDX3Y, UTY, USP9Y, VCY

Differences are expressed through gender genes:

```
plot(final_exprM["RPS4Y1",], final_exprM["VCY",], xlab = "RPS4Y1",
       ylab = "VCY")
```



5.4. GSVA Hallmark geneset:

Uses of library: pheatmap, gsva

Access hall mark gene set and split into two columns: gene_symbol and gs_name:

```
hallmarkL <- msigdbr(species = "human", category = "H")
hallmarks_list <- split( hallmarkL$gene_symbol, hallmarkL$gs_name)
head(hallmarks_list)
```

Using GSVA function for GSE6154 dataset:

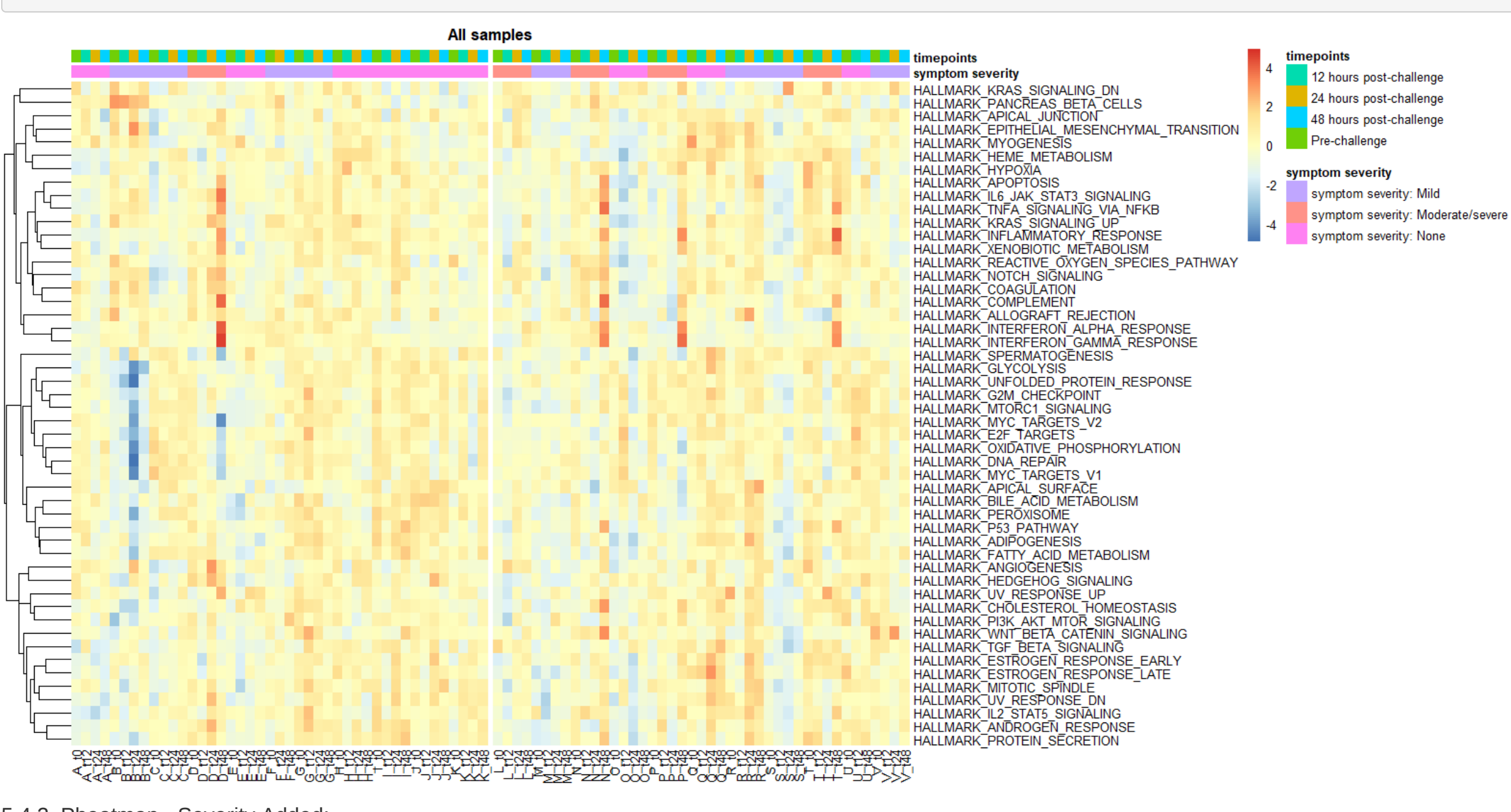
```
gsM <- gsva(data.matrix(final_exprM), gset.idx.list = hallmarks_list,
                       method = "ssgsea")
colnames(gsM) <- unlist(lapply(colnames(gsM), function(x){phenoM[x,"description"]}))
```

Prepare samples information: symptom severity and timepoints:

```
#remove two columns with NULL values
phenoM_gene_set_new <- phenoM[-c(22, 83),]
condition <- cbind(phenoM_gene_set_new$characteristics_ch1.2,
                   phenoM_gene_set_new$`timepoint:ch1`)
condition <- data.frame(condition)
row.names(condition) <- colnames(gsM)
names(condition)[1] <- paste("symptom severity")
names(condition)[2] <- paste("timepoints")
condition$timepoints <- replace(condition$timepoints, condition$timepoints == "0 hour",
                                                                 "Pre-challenge")
```

5.4.1. Pheatmap - All samples:

```
pheatmap(gsM, scale = "row", main = "All samples",
       cluster_cols = F, angle_col = 90, gaps_col = 43,
       annotation_col = condition)
```



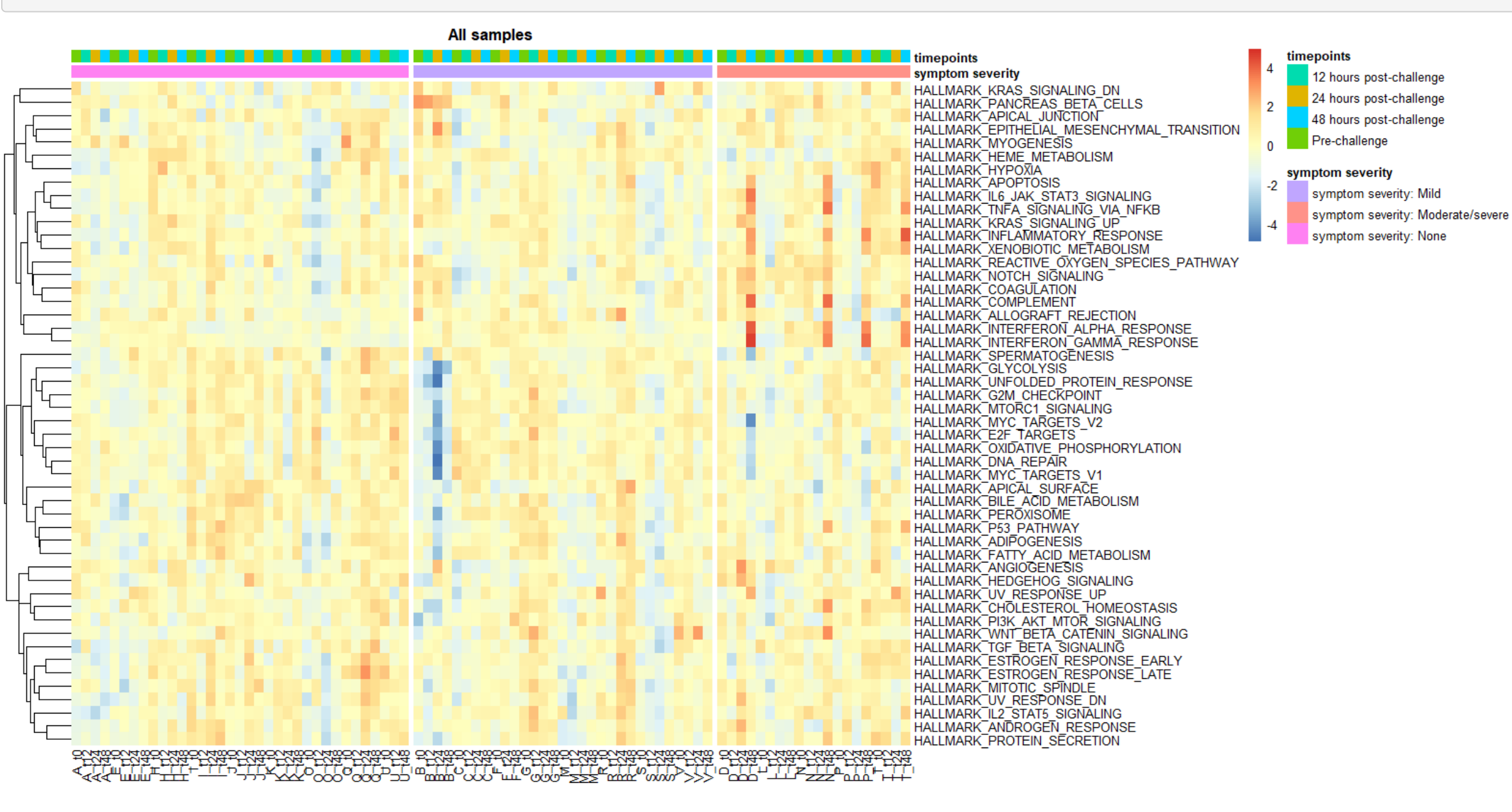
5.4.2. Pheatmap - Severity Added:

Prepare severity order: nonePatient, mildPatient and severePatient

```
nonePatient <- intersect(colnames(gsM),
                         as.character(phenoM_gene_set_new[phenoM_gene_set_new$`symptom severity:ch1`=="None",
                                                          "description"]))
mildPatient <- intersect(colnames(gsM),
                         as.character(phenoM_gene_set_new[phenoM_gene_set_new$`symptom severity:ch1`=="Mild",
                                                          "description"]))
severePatient <- intersect(colnames(gsM),
                         as.character(phenoM_gene_set_new[phenoM_gene_set_new$`symptom severity:ch1`=="Moderate/severe",
                                                          "description"]))
```

Pheatmap with severity added:

```
pheatmap(gsM[,c(nonePatientIDs, mildPatientIDs, severePatientIDs)],
          scale = "row", main = "All samples", cluster_cols = F,
          angle_col = 90,
          gaps_col = c(length(nonePatientIDs), length(nonePatientIDs)+length(mildPatientIDs)),
          annotation_col = condition)
```



5.5. Up and Down genes:

Uses of library: limma

Fitting linear models funtion (rank genes in order):

```
deanalysis <- function(data, group1, group2)
              {
               group <- c(rep(1, length(group1)), rep(0, length(group2)))
               df <- data[,c(group1, group2)]
               design <- model.matrix(~ group)
               #Linear modelling in limma
               fit <- lmFit(df, design)
               #Empirical Bayes moderation
               fit <- eBayes(fit)
               #Get top gene
               res <- topTable(fit, n = nrow(data))

               return(res)
              }
```

Attributes used when plotting:

```
upGeneL <- NULL
downGeneL <- NULL
allGeneL <- NULL
p_thr <- 0.05 #p-value
lfc_thr <- 0 #logFC
#Timepoints information in plot
timepoints <- c(“0”, “12”, “24”, “48”)
phenoM[phenoM$`timepoint:ch1`=="Pre-challenge","timepoint:ch1"] <- "0 hour"
#Data use for plot
mydata <- final_exprM
```

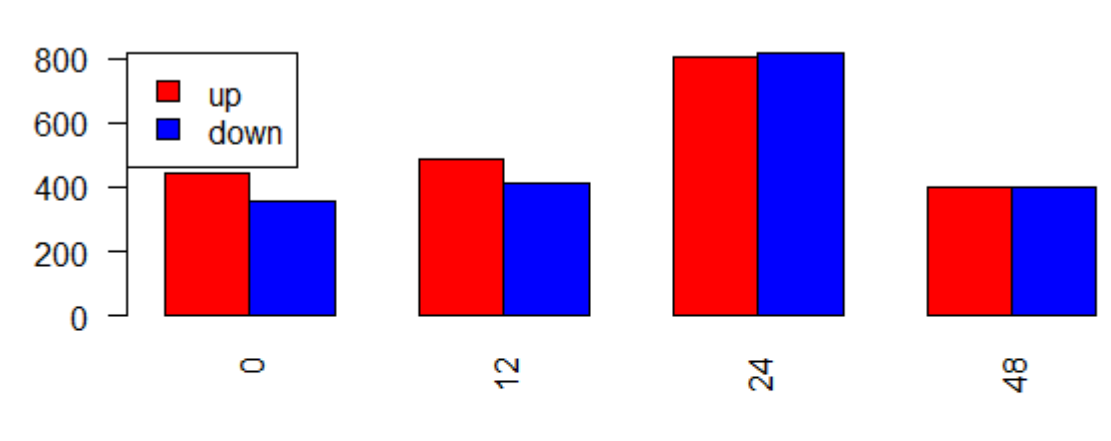
5.5.1. Up and Down genes all samples:

Calculate up and down genes each timepoint:

```
for(i in timepoints)
{
 #Create data for two group: Vaccine and Control in samples
 group1 <- rownames(phenoM)[phenoM$`vaccination status:ch1`=="Vaccinee"&grepl(paste(i, "hour"),phenoM$`timepoint:ch1`)]
 group2 <- rownames(phenoM)[phenoM$`vaccination status:ch1`=="Control"&grepl(paste(i, "hour"),phenoM$`timepoint:ch1`)]
 group1 <- intersect(group1, colnames(mydata))
 group2 <- intersect(group2, colnames(mydata))
 #Calculate Up and Down gene by using limma
 if(length(group1)>=1 & length(group2)>=1) {
 res <- deanalysis(mydata, group1, group2)
 rownames(res) <- rownames(mydata)
 #Up gene (use the bounds of p-value and logFC to find the desired result)
 upRes <- res[res$P.Value < p_thr & res$logFC > lfc_thr,]
 upGeneL[[i]] <- rownames(upRes[complete.cases(upRes),])
 #Down gene (use the bounds of p-value and logFC to find the desired result)
 downRes <- res[res$P.Value < p_thr & res$logFC < lfc_thr,]
 downGeneL[[i]] <- rownames(downRes[complete.cases(downRes),])
 allGeneL[[i]] <- c(upGeneL[[i]], downGeneL[[i]])
}
```

Plot up and down genes with barplot():

```
#Up and Down gene data over time
deGeneLen <- rbind(lengths(upGeneL), lengths(downGeneL))
par(mar=c(10,4,2,2))
barplot(deGeneLen, las=2, beside = T, col = c("red","blue"))
legend("topleft", legend = c("up","down"), fill =c("red","blue"))
```



5.5.2. Up and Down genes severity added:

Calculate up and down genes each timepoint (severity added):

```
for(i in timepoints)
{
 #Create data for two groups: Vaccine and Control in samples Moderate/Severe
 group1 <- rownames(phenoM)[phenoM$`symptom severity:ch1`=="Moderate/severe"&
                            phenoM$`vaccination status:ch1`=="Vaccinee"&
                            grepl(paste(i, "hour"),phenoM$`timepoint:ch1`)]
 group2 <- rownames(phenoM)[phenoM$`symptom severity:ch1`=="Moderate/severe"&
                            phenoM$`vaccination status:ch1`=="Control"&
                            grepl(paste(i, "hour"),phenoM$`timepoint:ch1`)]
 group1 <- intersect(group1, colnames(mydata))
 group2 <- intersect(group2, colnames(mydata))
 #Calculate Up and Down gene by using limma
 if(length(group1)>=1 & length(group2)>=1) {
 res <- deanalysis(mydata, group1, group2)
 rownames(res) <- rownames(mydata)
 #Up gene (use the bounds of p-value and logFC to find the desired result)
 upRes <- res[res$P.Value < p_thr & res$logFC > lfc_thr,]
 upGeneL[[i]] <- rownames(upRes[complete.cases(upRes),])
 #Down gene (use the bounds of p-value and logFC to find the desired result)
 downRes <- res[res$P.Value < p_thr & res$logFC < lfc_thr,]
 downGeneL[[i]] <- rownames(downRes[complete.cases(downRes),])
 allGeneL[[i]] <- c(upGeneL[[i]], downGeneL[[i]])
}
```

Plot up and down genes (severity added) with barplot():

```
#Up and Down gene data over time
deGeneLen <- rbind(lengths(upGeneL), lengths(downGeneL))
par(mar=c(10,4,2,2))
barplot(deGeneLen, las=2, beside = T, col = c("red","blue"))
legend("topleft", legend = c("up","down"), fill =c("red","blue"))
```