1 Visualisation de données RNA-seq sous R/Bioconductor

2 Analyse d'enrichissement GO sous Bioconductor

TD: Analyse RNA-Seq sous R/Bioconductor: Visualisation et Enrichissement GO

Ghislain Bidaut, Plateforme Cibi, CRCM, Aix-Marseille Université

24/03/2022

1 Visualisation de données RNA-seq sous R/Bioconductor

Objectif: Dans le cadre de ce TD, nous allons visualiser les données obtenues dans l'analyse différentielle à l'aide de différents outils:

- Heatmaps
- Volcano plots
- Profils d'expression

1.1 Chargement des données

Nous allons recharger la librairie `EdgeR` , qui nous a servi à générer les données dans le TD précédent.

Nous allons également charger les objets `annotated_expression_counts` , `et` , `targets` , et `y` générés lors du TD précédente.

Ces objets sont disponibles dans le fichier `diffAnalysis.RData` .

https://mycore.core-cloud.net/index.php/s/pvmUZqNjTpYybCc?path=%2FTD_RNA_Seq (https://mycore.core-cloud.net/index.php/s/pvmUZqNjTpYybCc?path=%2FTD_RNA_Seq)

```
library(edgeR)
```

```
## Le chargement a nécessité le package : limma
```

```
load(file = "diffAnalysis.RData")
```

Nous disposons ensuite des objets suivants:

```
ls()
```

```
## [1] "annotated_expression_counts" "et"
## [3] "show_sol"                    "targets"
## [5] "y"
```

1.2 Génération d'une heatmap (diagramme de chaleur)

Une heatmap est une représentation matricielle des données. Typiquement, nous représentations les profils d'expression des gènes **en ligne** et les échantillons **en colonnes**. Les valeurs de comptages sont représentées sous la forme d'une couleur adaptée.

Sous Bioconductor, nous disposons du package `pheatmap` .

- Par contre, il est compliqué de représenter l'ensemble du génome. Nous allons donc d'abord représenter **les 50 gènes dont l'expression est la plus variable**.
- Ensuite, nous ferons une seconde heatmap représentant **les profils d'expression des 50 gènes les plus différentiellement exprimés**.

1.2.1 Heatmaps des 50 gènes les plus variables

Chargement des packages: `phetmap` pour la représentation de données et `RcolorBrewer` pour les palettes de couleurs.

```
library(pheatmap)
library(RColorBrewer)
```

Ensuite, nous allons calculer les variances pour chaque profil d'expression, donc chaque ligne dans la matrice d'expression.

```
expres <- cpm(y, log= TRUE)
var_genes <- apply(X = expres, MARGIN = 1, FUN = var)
```

Nous sélectionnons les 50 gènes dont la variance de l'expression à travers les conditions est la plus élevée.

Pour cela, on sort les *indexes* de ces gènes à l'aide de la fonction `order` .

```
select_var <- order(var_genes, decreasing=TRUE)[1:50]
head(select_var)
```

```
## [1]  4685  6338  5388  5784  9704 11486
```

Nous sélectionnons les profils d'expression correspondants à ces gènes, ainsi que leurs identifiants.

```
highly_variable_cpm <- expres[select_var,]

highly_variable_gene_annot <- annotated_expression_counts[select_var, 1:8]

row.names(highly_variable_cpm) <- paste(highly_variable_gene_annot$ensembl_gene_id_vers
ion,  '[', highly_variable_gene_annot$hgnc_symbol , ']')
```

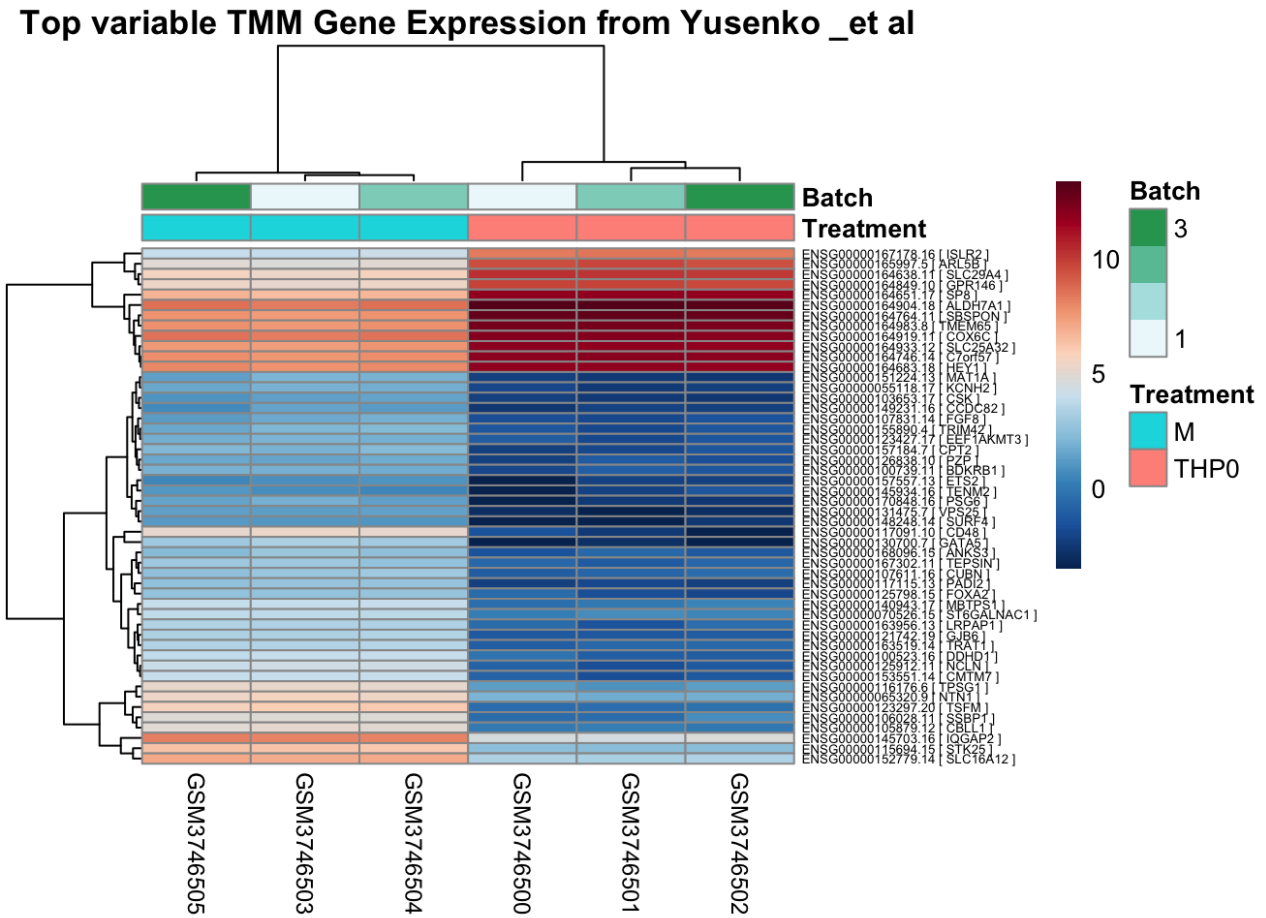
Nous préparons deux variables: L'une est une palette de couleurs `colors` à l'aide de la fonction `colorRampPalette` . L'autre est un tableau `sample_annot` contenant les annotations des échantillons que l'on souhaite afficher sur la heatmap.

```
colors <- rev(colorRampPalette(brewer.pal(10, "RdBu"))(256));

sample_annot <- data.frame(Treatment=targets$Treatment, Batch=targets$Sample, row.names
= targets$SRA_ID)

sample_annot
```

```
##            Treatment Batch
## SRR9005674      THP0     1
## SRR9005675      THP0     2
## SRR9005676      THP0     3
## SRR9005677         M     1
## SRR9005678         M     2
## SRR9005679         M     3
```

Nous pouvons faire appel à `pheatmap` proprement dit:

```
pheatmap(highly_variable_cpm, cluster_rows = T, cluster_cols = T, main = "Top variable
 TMM Gene Expression from Yusenko _et al", fontsize_row = 5, annotation_col = sample_an
not, fontsize_col = 9, labels_col = as.character(targets$GEO_ID), color = colors, heigh
t = 20)
```



1.3 Heatmap des gènes les plus différentiellement exprimés

Nous extrayons les gènes le plus différentiellement exprimés du test fait avec **EdgeR** dans le TP précédent.

Le résultat du test est stocké dans l'objet `et` . Nous l'extrayons avec `topTags` .

```
tt <- topTags(et, adjust.method = "BH", sort.by = c("PValue"), n=20000)
colnames(tt)
```

```
##  [1] "ensembl_gene_id_version" "entrezgene_id"
##  [3] "hgnc_symbol"             "Chr"
##  [5] "Start"                   "End"
##  [7] "Strand"                  "Length"
##  [9] "logFC"                   "logCPM"
## [11] "PValue"                  "FDR"
```

La table tt (topTags) contient les statistiques associées à l'analyse différentielle des gènes. Nous pouvons trouver les gènes significativement différentiellement exprimés, en appliquant un seuil $alpha$ sur la p-valeur **ajustee** et un seuil $logFC.th$ sur la valeur logFC (Log Fold change). Nous Appliquons un seuil sur la **valeur absolue** de logFC de manière à extraire les gènes **sur-** et **sous-** exprimés.

```
logFC.th <- 2.0
alpha <- 1e-80

tt_selected <- tt[abs(tt$table$logFC) >= logFC.th & tt$table$FDR <= alpha,]

significant.genes <- rownames(tt_selected)
head(significant.genes)
```

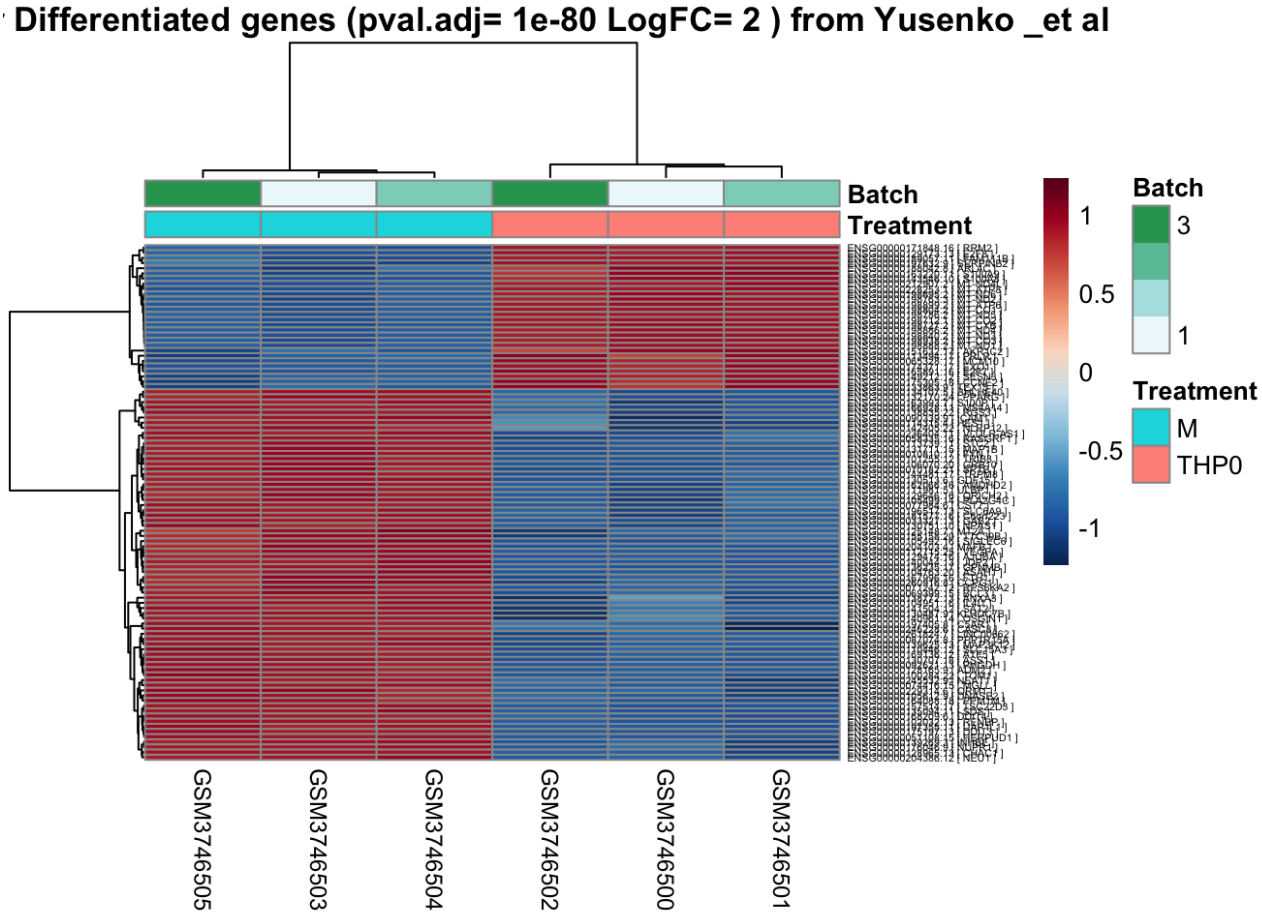
```
## [1] "6299"  "7268"  "17946" "17897" "4241"  "4289"
```

on extrait ensuite les valeurs des profils d'expression et les noms des gènes à partir du tableau `annotated_expression_counts` .

```
selected_gene_count <- expres[significant.genes,]

selected_gene_annot <- annotated_expression_counts[significant.genes, 1:8]

row.names(selected_gene_count) <- paste(selected_gene_annot$ensembl_gene_id_version,
'[', selected_gene_annot$hgnc_symbol , ']')

colors <- rev(colorRampPalette(brewer.pal(10, "RdBu"))(256));
```

On peut réutiliser l'objet `sample_annot` défini précédemment puis faire appel à `pheatmap` avec les données présentes dans `selected_gene_count` .

```
pheatmap(selected_gene_count, cluster_rows = T, cluster_cols = T, scale = "row", main =
paste("TMM for Differentiated genes (pval.adj=", alpha, "LogFC=", logFC.th, ") from Yus
enko _et al"), fontsize_row = 4, annotation_col = sample_annot, fontsize_col = 9, label
s_col = as.character(targets$GEO_ID), color = colors, height = 20)
```



1.3.1 Représentation des gènes différentiellement exprimés sous forme de diagramme en volcan (*volcano plot*)

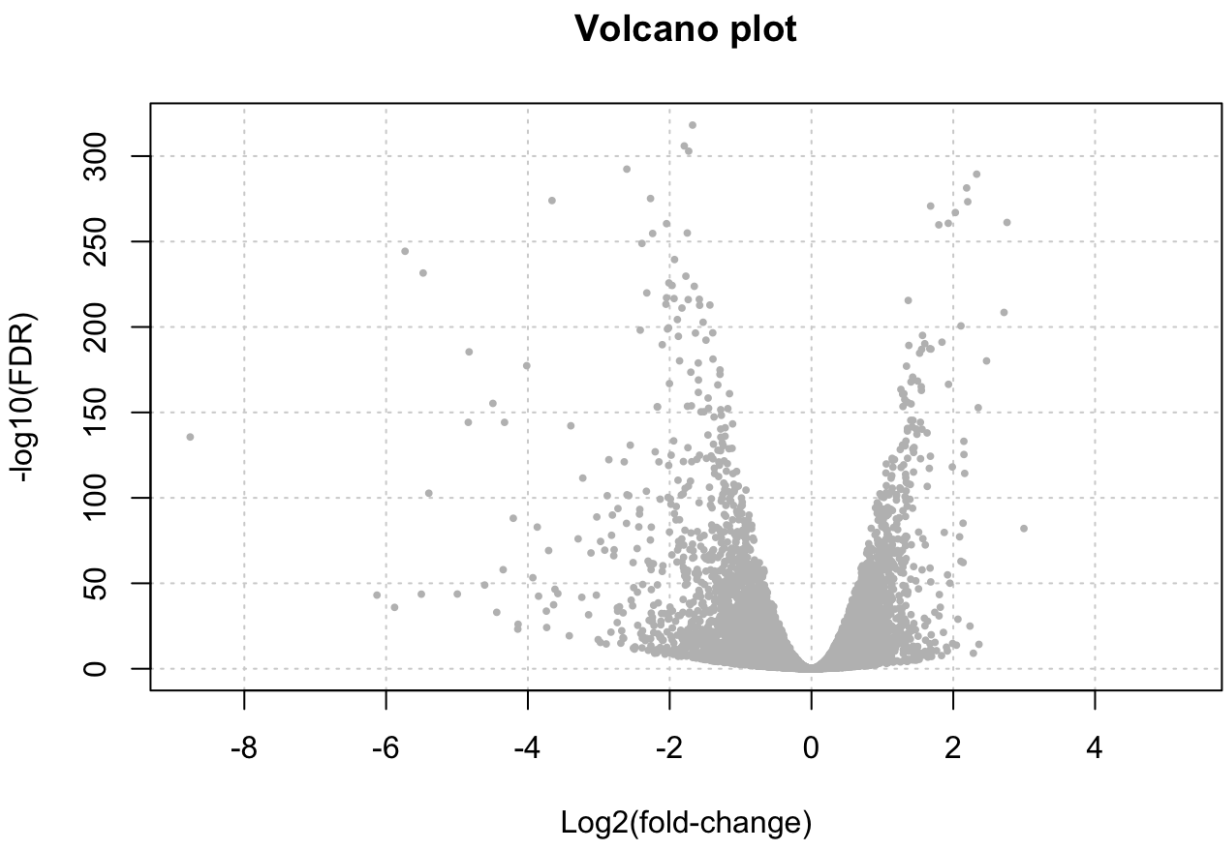
Un *volcano plot* consiste à représenter le nuage de points défini par $(x, y) = (logFC(g), -log_{10}(pVal(g))$, $logFC(g)$ étant la valeur log Fold-Change associé au gène g entre les deux conditions comparées, et $pVal.adj(g)$ étant la p-valeur **ajustée** associée au test statistique utilisé.

Nous utilisons la fonction `densCols` qui permet d'encoder la densité locale des points à représenter avec une couleur.

```
cols <- densCols(tt$table$logFC,
                 -log10(tt$table$FDR),
                 colramp = colorRampPalette("grey"))
```

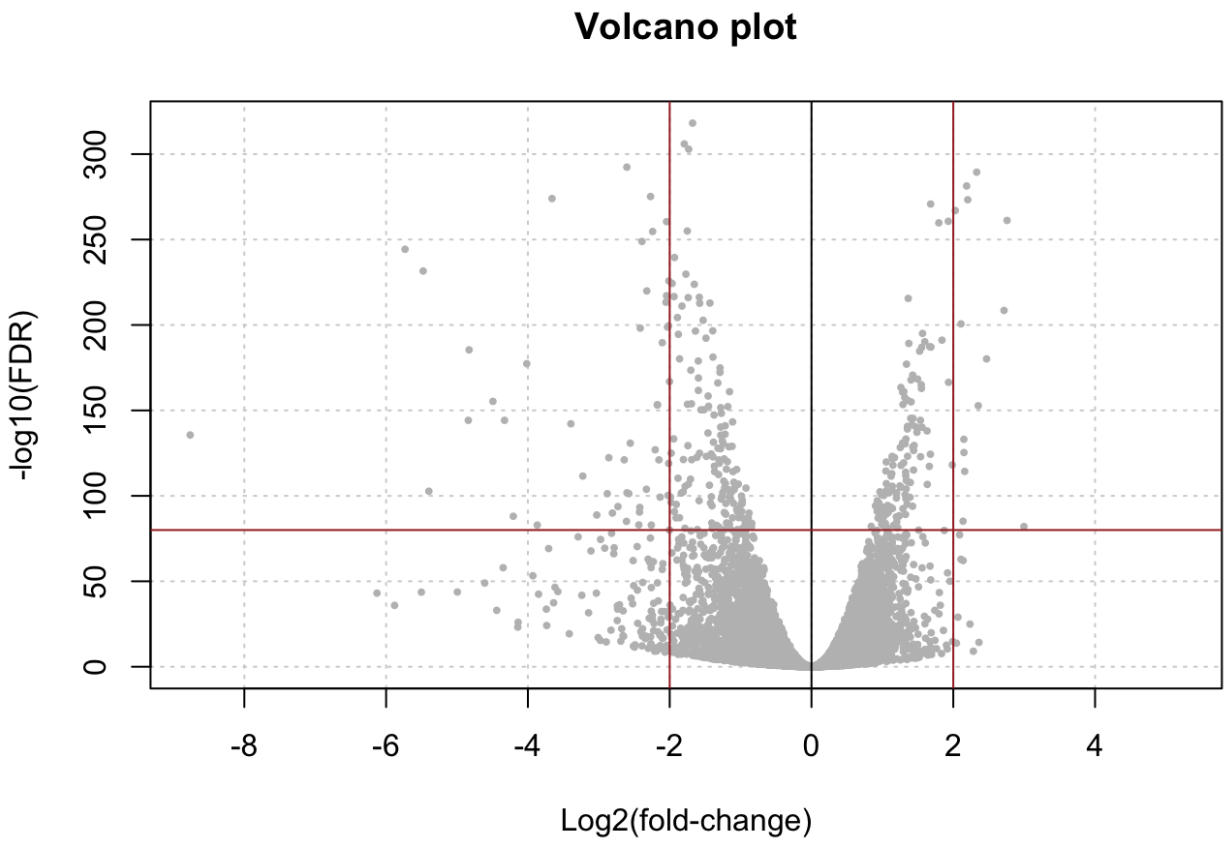
Ensuite, nous faisons appel à la fonction `plot` avec les couleurs calculées précédemment.

```
plot(tt$table$logFC,
     -log10(tt$table$FDR),
     col=cols, panel.first=grid(),
     main="Volcano plot",
     xlab="Log2(fold-change)",
     ylab="-log10(FDR)",
     pch=20, cex=0.6)
```



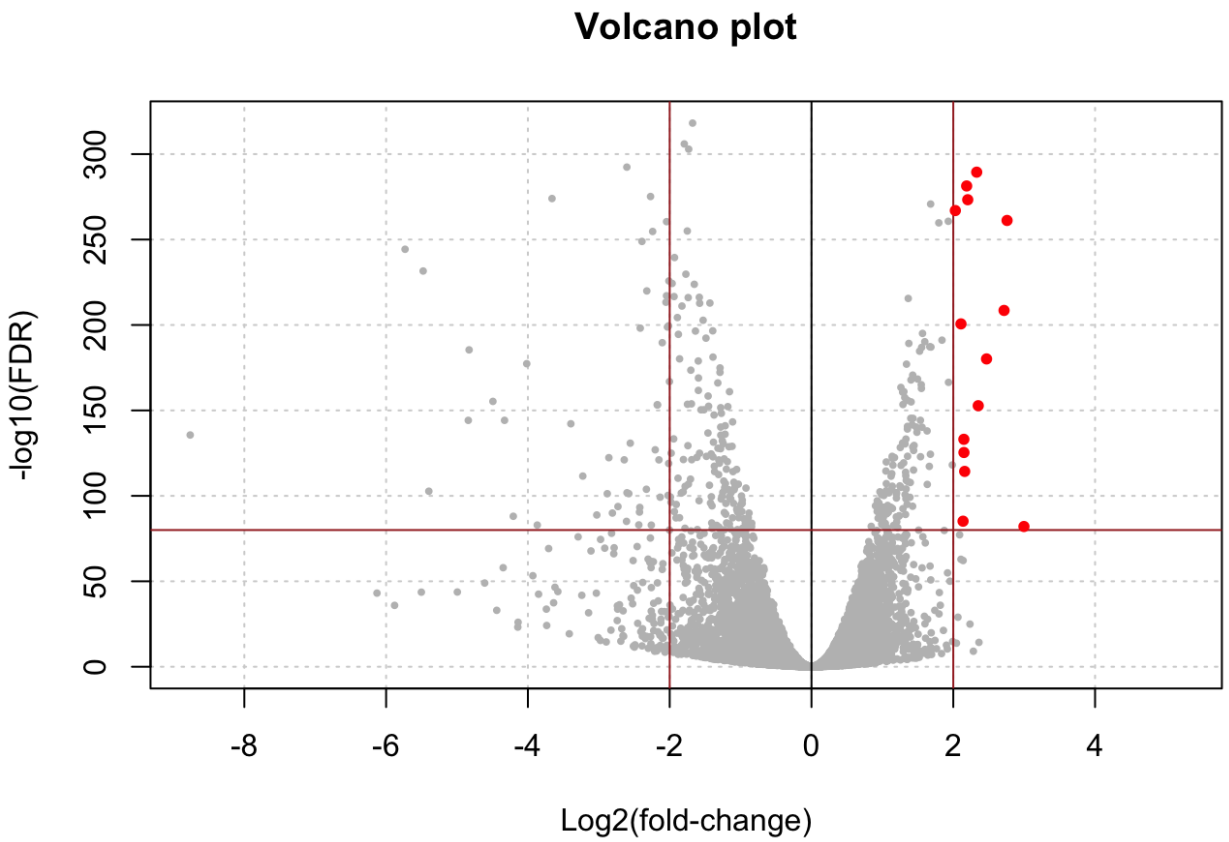
Nous avons maintenant l'ensemble du nuage. Pour la suite, nous allons matérialiser les valeurs seuils $logFC.th$ et $alpha$ appliquées respectivement à $logFC(g)$ et $FDR(g)$.

```
plot(tt$table$logFC,
     -log10(tt$table$FDR),
     col=cols, panel.first=grid(),
     main="Volcano plot",
     xlab="Log2(fold-change)",
     ylab="-log10(FDR)",
     pch=20, cex=0.6)

abline(v=0)
abline(v=c(-logFC.th, logFC.th), col="brown")
abline(h=-log10(alpha), col="brown")
```



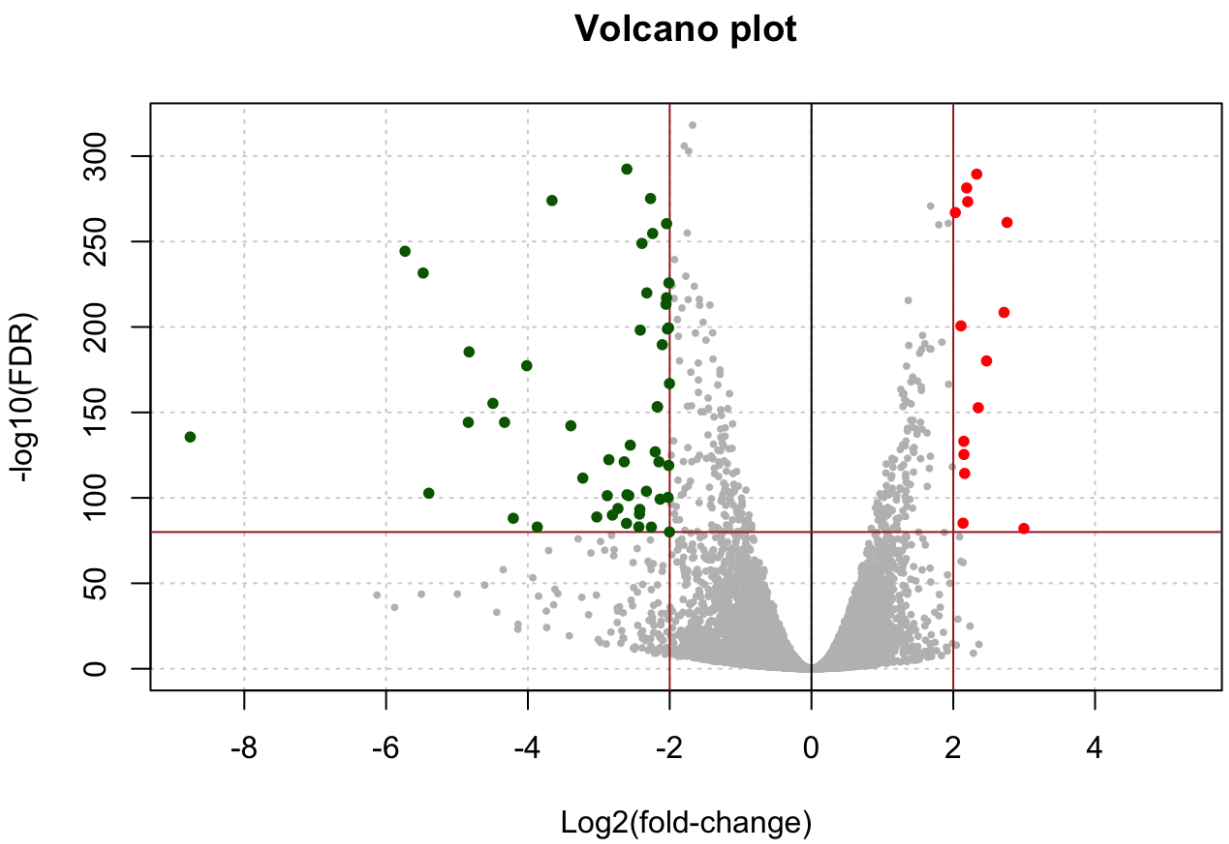
Ensuite, on sélectionne le **sous-ensemble** des données correspondants aux gènes **significativement surexprimés**, c'est à dire avec des $logFC$ et $p-val.adj$ ajustées supérieurs aux seuils $alpha$ et $logFC.th$, et on applique la fonction `points` sur ces données grâce à la fonction `with` . Ces données sont représentées en **rouge**.

```
plot(tt$table$logFC,
     -log10(tt$table$FDR),
     col=cols, panel.first=grid(),
     main="Volcano plot",
     xlab="Log2(fold-change)",
     ylab="-log10(FDR)",
     pch=20, cex=0.6)

abline(v=0)
abline(v=c(-logFC.th, logFC.th), col="brown")
abline(h=-log10(alpha), col="brown")

with(subset(tt$table, logFC >= logFC.th & FDR < alpha ),
     points(logFC, -log10(FDR),
            pch=20,
            col="red"))
```



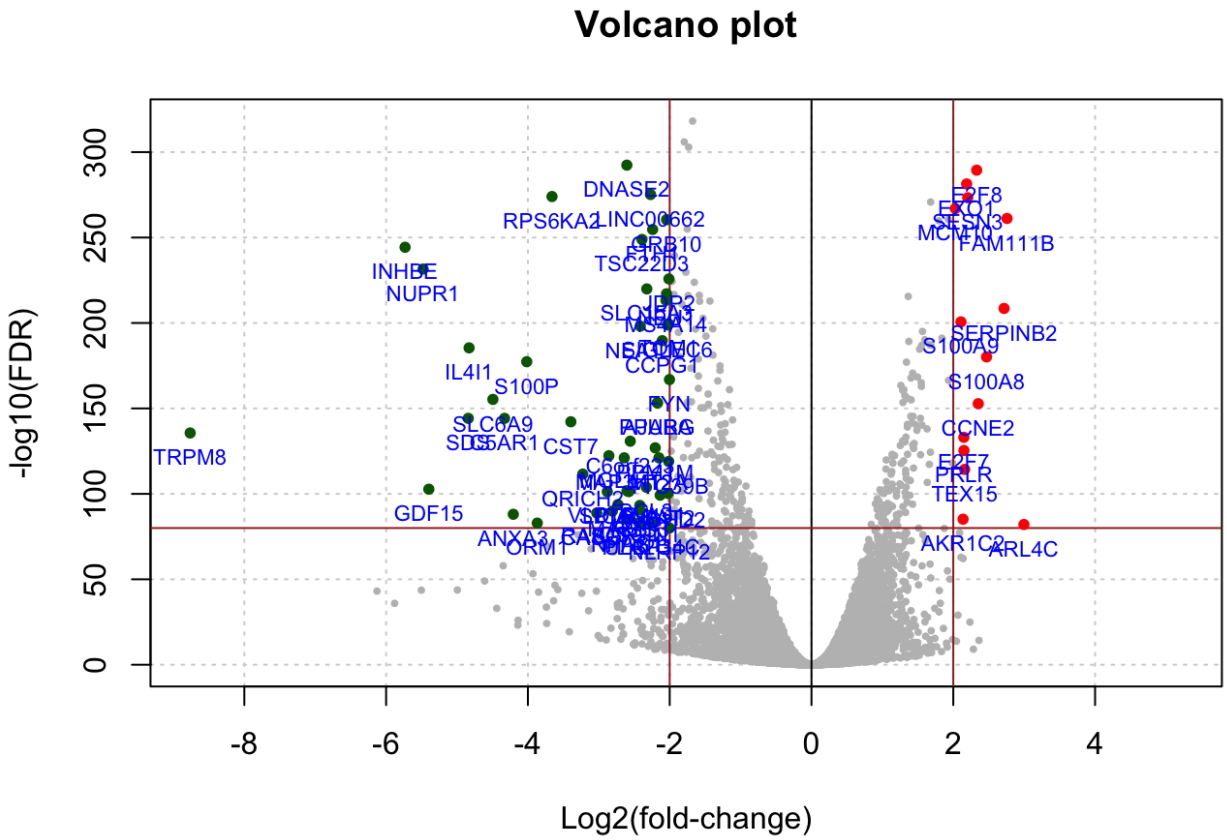
On applique le même type de sélection, pour les gènes **significativement sous-exprimés**. cette fois ci, on appliquera une couleur verte aux données.

```
plot(tt$table$logFC,
     -log10(tt$table$FDR),
     col=cols, panel.first=grid(),
     main="Volcano plot",
     xlab="Log2(fold-change)",
     ylab="-log10(FDR)",
     pch=20, cex=0.6)

abline(v=0)
abline(v=c(-logFC.th, logFC.th), col="brown")
abline(h=-log10(alpha), col="brown")

with(subset(tt$table, logFC >= logFC.th & FDR < alpha ),
     points(logFC, -log10(FDR),
            pch=20,
            col="red"))

with(subset(tt$table, logFC <= -logFC.th & FDR < alpha ),
     points(logFC, -log10(FDR),
            pch=20,
            col="darkgreen"))
```



Enfin, ajoutons les labels des gènes les plus différentiellement exprimés

```
plot(tt$table$logFC,
     -log10(tt$table$FDR),
     col=cols, panel.first=grid(),
     main="Volcano plot",
     xlab="Log2(fold-change)",
     ylab="-log10(FDR)",
     pch=20, cex=0.6)

abline(v=0)
abline(v=c(-logFC.th, logFC.th), col="brown")
abline(h=-log10(alpha), col="brown")

with(subset(tt$table, logFC >= logFC.th & FDR < alpha ),
     points(logFC, -log10(FDR),
            pch=20,
            col="red"))

with(subset(tt$table, logFC <= -logFC.th & FDR < alpha ),
     points(logFC, -log10(FDR),
            pch=20,
            col="darkgreen"))

sub_tt <- subset(tt$table, abs(logFC) >= logFC.th & FDR < alpha )

text(sub_tt$logFC, -log10(sub_tt$FDR),
          labels = sub_tt$hgnc_symbol,
          pos=1, offset = 0.5, cex=0.7,
          col="blue")
```



1.4 Représentation de profils d'expression

Pour cette partie, nous allons utiliser le package Glimma.

http://bioconductor.org/packages/release/bioc/html/Glimma.html (http://bioconductor.org/packages/release/bioc/html/Glimma.html)

Et nous documenter avec sa vignette:

https://bioconductor.riken.jp/packages/3.8/bioc/vignettes/Glimma/inst/doc/Glimma.pdf (https://bioconductor.riken.jp/packages/3.8/bioc/vignettes/Glimma/inst/doc/Glimma.pdf).

C'est un package très intéressant car il permet d'afficher les plots MDS et les profils d'expression de manière interactive et donc d'explorer les données très en profondeur.

On commence par le charger:

```
library(Glimma)
```

1.4.1 Plot MDS avec GLimma:

```
glMDSPlot(y, groups =  targets$Treatment)
```

Click here for interactive version (glimma-plots/MDS-Plot.html)

1.4.2 Exploration des profils d'expression avec Glimma:

```
dt.edger <- decideTestsDGE(et)

glMDPlot(et, status=dt.edger, counts=y, groups = targets$Treatment, transform = TRUE, d
isplay.columns = c("hgnc_symbol", "entrezgene_id"))
```

Click here for interactive version (glimma-plots/MD-Plot.html)

2 Analyse d'enrichissement GO sous Bioconductor

Nous finissions ce TD par la recherche de pathways différentiellement exprimés. Pour cela , nous allons utiliser le package `ClusterProfiler` .

https://bioconductor.org/packages/release/bioc/html/clusterProfiler.html (https://bioconductor.org/packages/release/bioc/html/clusterProfiler.html)

Et un tutoriel intéressant:

https://learn.gencore.bio.nyu.edu/rna-seq-analysis/over-representation-analysis/ (https://learn.gencore.bio.nyu.edu/rna-seq-analysis/over-representation-analysis/)

L'analyse de pathways passe par l'utilisation d'une ontologie dédiée, c'est à dire un vocabulaire contrôlé hiérarchique permettant l'annotation normalisée des gènes. L'ontologie la plaus utilis&é est celle établie par le Gene Ontology Consortium, dont les détails sont disponibles sur ce site: http://geneontology.org (http://geneontology.org).

Commençons par charger le package:

```
library(clusterProfiler)
```

Nous travaillons dans l'humain, et devons donc charger la base de données correspondante. Les annotations disponibles sont listées ici: http://bioconductor.org/packages/release/BiocViews.html#___OrgDb (http://bioconductor.org/packages/release/BiocViews.html#___OrgDb).

```
organism = "org.Hs.eg.db"

# Eventuellement l'installer:
#BiocManager::install(organism, character.only = TRUE)

library(organism, character.only = TRUE)
```

Ensuite, nous allons analyser les pathways pour les gènes les plus différentiellement exprimés et su-exprimés dans la condition *M* vs *TH0* (donc vérifiant $logFC > 0$).

Pour cela, nous devons extraire cette liste sous forme d'identifiants de gènes, à partir de la table `tt` . La fonction `subset` fera l'affaire.

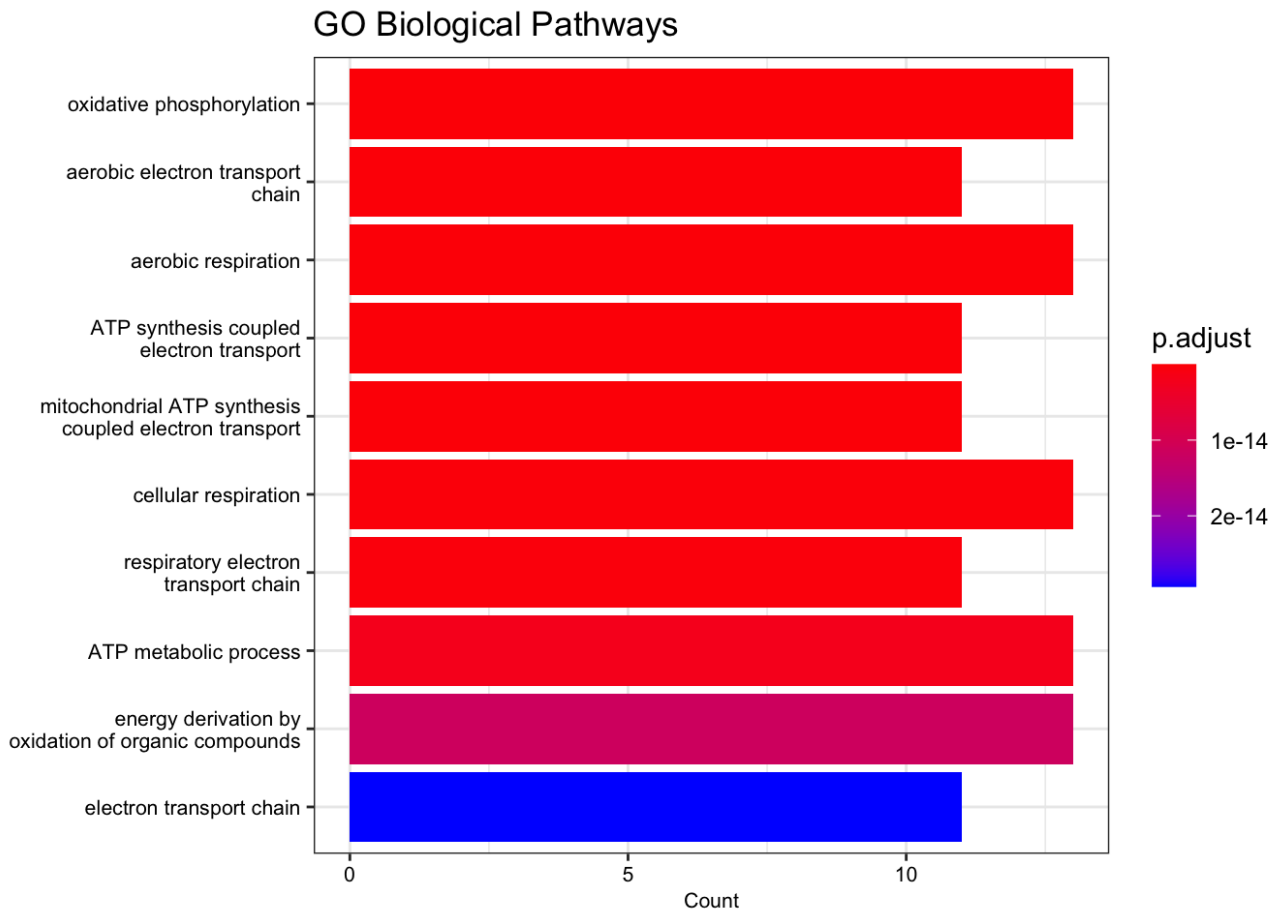
```
sub_tt <- subset(tt$table, logFC >= logFC.th & FDR < alpha )

gene_names <- sub_tt$entrezgene_id
```

Créer l'objet EnrichGO. Nous spécifions que nous disposons d'une liste sous la forme d'identifiant de type **Entrez Gene ID** et que nous souhaitons spécifiquement extraire les *Biological Pathways*.

```
go_enrich <- enrichGO(gene = gene_names,
                      OrgDb = organism,
                      keyType = 'ENTREZID',
                      readable = T,
                      ont = "BP",  # Biological process
                      pvalueCutoff = 0.05,
                      qvalueCutoff = 0.10)
```

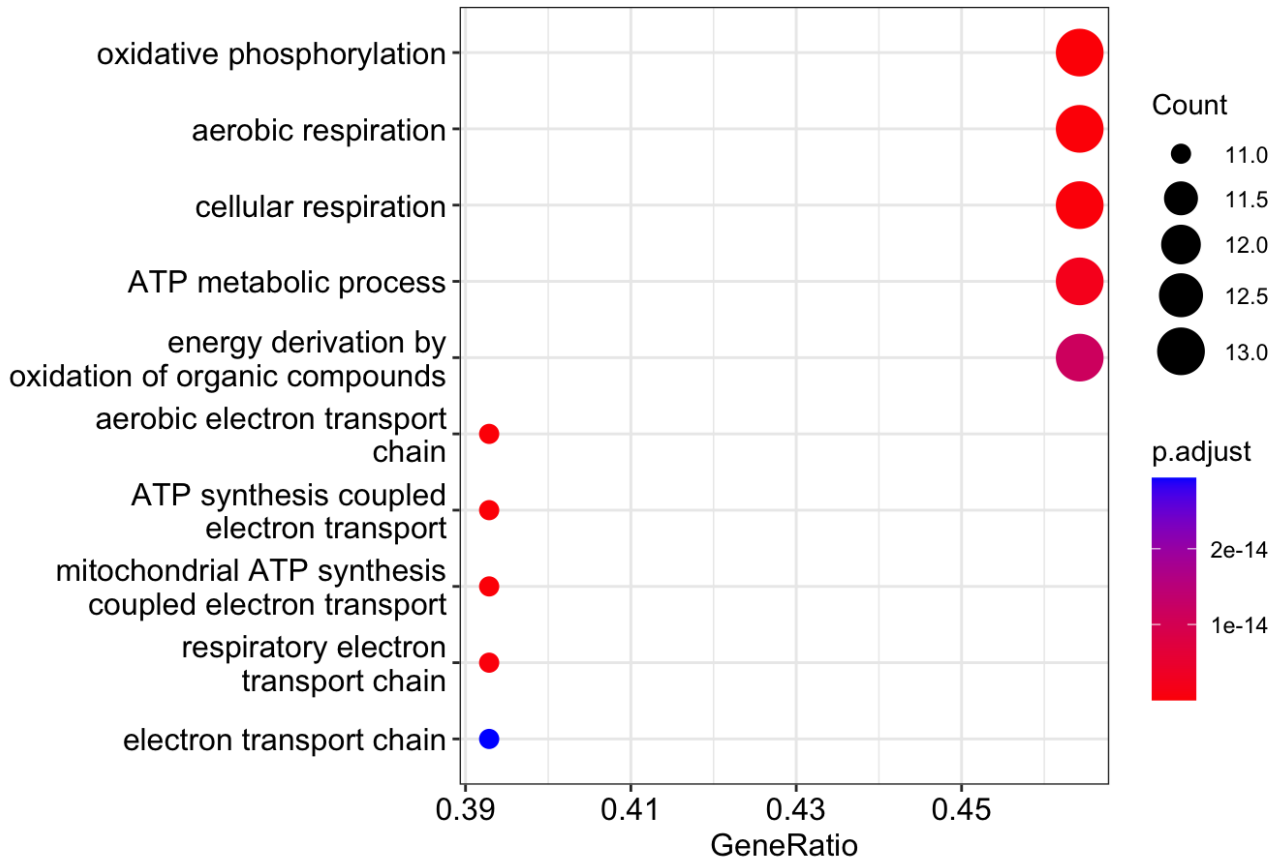
On peut ensuite afficher les *Biological Pathways* les plus différentiellement exprimés à l'aide de `barplot` .

```
barplot(go_enrich,
        drop = TRUE,
        showCategory = 10,
        title = "GO Biological Pathways",
        font.size = 8)
```



Ou de DotPlot:

```
dotplot(go_enrich)
```



Il est également possible de voir la sortie complète d' `enrichGO` sous forme de table.

```
head(data.frame(go_enrich))
```

```
##                    ID                                          Description
## GO:0006119 GO:0006119                              oxidative phosphorylation
## GO:0019646 GO:0019646                         aerobic electron transport chain
## GO:0009060 GO:0009060                                    aerobic respiration
## GO:0042773 GO:0042773                  ATP synthesis coupled electron transport
## GO:0042775 GO:0042775 mitochondrial ATP synthesis coupled electron transport
## GO:0045333 GO:0045333                                   cellular respiration
##            GeneRatio   BgRatio       pvalue     p.adjust       qvalue
## GO:0006119     13/28 141/18723 4.841038e-21 2.192990e-18 1.330011e-18
## GO:0019646     11/28  87/18723 2.279891e-19 3.688047e-17 2.236738e-17
## GO:0009060     13/28 189/18723 2.442415e-19 3.688047e-17 2.236738e-17
## GO:0042773     11/28  95/18723 6.314362e-19 5.720812e-17 3.469576e-17
## GO:0042775     11/28  95/18723 6.314362e-19 5.720812e-17 3.469576e-17
## GO:0045333     13/28 230/18723 3.282396e-18 2.478209e-16 1.502992e-16
##                                                                  geneID Count
## GO:0006119 COX1/COX2/ND4/ND4L/COX3/ND6/ND3/ATP8/ATP6/ND5/CYTB/ND1/ND2    13
## GO:0019646           COX1/COX2/ND4/ND4L/COX3/ND6/ND3/ND5/CYTB/ND1/ND2    11
## GO:0009060 COX1/COX2/ND4/ND4L/COX3/ND6/ND3/ATP8/ATP6/ND5/CYTB/ND1/ND2    13
## GO:0042773           COX1/COX2/ND4/ND4L/COX3/ND6/ND3/ND5/CYTB/ND1/ND2    11
## GO:0042775           COX1/COX2/ND4/ND4L/COX3/ND6/ND3/ND5/CYTB/ND1/ND2    11
## GO:0045333 COX1/COX2/ND4/ND4L/COX3/ND6/ND3/ATP8/ATP6/ND5/CYTB/ND1/ND2    13
```

2.1 TP:

- Chercher les termes GO enrichis dans la comparaison M vs THP0.
- Explorer les autres ontologies:
    - *Molecular Function*

    - *Cellular Component*
- Faire la même analyse avec les pathways KEGG.


(http://creativecommons.org/licenses/by-nc-nd/4.0/)

Cette œuvre est mise à disposition selon les termes de la Licence Creative Commons:
(http://creativecommons.org/licenses/by-nc-nd/4.0/)
Attribution - Pas d'Utilisation Commerciale - Pas de Modification 4.0 International (CC BY-NC-ND 4.0).