Mini-projet 2021 - Exploration des données de Pavkovic

Prénom Nom

2021-04-16

Contents


```
The downloaded binary packages are in
    /var/folders/9s/0zkjn8tm8xj7wp0059bl3v9000020t/T//RtmpgQxLMQ/downloaded_packages
```

Table 1: Loaded required libraries

libraries
knitr
FactoMineR
factoextra
pheatmap
biomaRt

```
R version 4.0.2 (2020-06-22)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Mojave 10.14.6
```

```
Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] biomaRt_2.44.4   pheatmap_1.0.12  factoextra_1.0.7 ggplot2_3.3.3    FactoMineR_2.4   knitr_1.31

loaded via a namespace (and not attached):
 [1] ggrepel_0.9.1        Rcpp_1.0.6           lattice_0.20-41      prettyunits_1.1.1    assertthat_0.2
[15] pillar_1.5.1         rlang_0.4.10         progress_1.2.2       curl_4.3             blob_1.2.1
[29] askpass_1.1          pkgconfig_2.0.3      BiocGenerics_0.34.0  htmltools_0.5.1.1    openssl_1.4.3
[43] withr_2.4.1          rappdirs_0.3.3       MASS_7.3-53.1        leaps_3.1            grid_4.0.2
[57] ellipsis_0.3.1       generics_0.1.0       vctrs_0.3.6          RColorBrewer_1.1-2   tools_4.0.2
[71] colorspace_2.0-0     cluster_2.1.1        BiocManager_1.30.12  memoise_2.0.0
```

Synopsis du projet

Travail demandé

Le but de ce travail est de mettre en oeuvre les méthodes vues dans le module 3 “R et statistiques” pour explorer le jeu de données de Pavokovic, et de rendre un rapport d’analyse au format `.Rmd`.

Nous fournissons ci-dessous une trame avec les principales sections attendues. Certaines contiennent déjà du code. Vous devrez en compléter d’autres. Sentez-vous libres d’adapter cette trame ou d’y ajouter des analyses complémentaires si elles vous aident à interpréter vos résultats.

Remise du rapport

Date: **le 10 mai 2021 minuit**. Si vous anticipez un problème pour remettre le rapport à cette date contactez-nous aussi rapidement que possible pour que nous puissions prévoir une remise plus tardive.

- Commencez par renommer le fichier .Rmd en remplaçant Prenom-NOM par vos nom et prénom.
- Le rapport est attendu en formats .Rmd + .HTML (en gardant l’option self_contained de l’en-tête activée).
- Déposez les fichiers dans un sous-dossier de vote compte du cluster. Attention, veillez à respecter précisément cette structure de chemin car nous nous baserons dessus pour récupérer vos résultats.

  `/shared/projects/dubii2021/[login]/m3-stat-R/mini-projet`

Critères d’évaluation

- Reproductibilité des analyses: nous tenterons de regénérer le rapport HTML à partir de votre Rmd, en partant de notre compte sur le serveur IFB.
- Manipulation des objets R
- Mobilisation des méthodes statistiques vues au cours
- Pertinence des interprétations statistiques
- Pertinence des interprétations biologiques
- Clarté de la rédaction
- Clarté des illustrations (figures et tableaux): graphismes, légendes ...

Nous vous encourageons à assurer la lisibilité de votre code (syntaxe, nommage des variables, commentaires de code)

Objectifs scientifiques

Nous partons du même jeu de données *Fil Rouge* de ce module issues de la publication Pavkovic, M., Pantano, L., Gerlach, C.V. et al. Multi omics analysis of fibrotic kidneys in two mouse models. Sci Data 6, 92 (2019). https://doi.org/10.1038/s41597-019-0095-5

Rappel sur les échantillons:

Deux modèles de fibrose rénale chez la souris sont étudiés:

1. Le premier est un modèle de néphropathie réversible induite par l'acide folique (folic acid (FA)). Les souris ont été sacrifiées avant le traitement (normal), puis à jour 1, 2, 7 et 14 (day1,...) après une seule injection d'acide folique.
2. Le second est un modèle irréversible induit chrirurgicalement (unilateral ureteral obstruction (UUO)). les souris ont été sacrifiées avant obstruction (day 0) et à 3, 7 et 14 jours après obstruction par ligation de l'uretère du rein gauche.

A partir de ces extraits de rein, l'ARN messager total et les petits ARNs ont été séquencés et les protéines caratérisées par spectrométrie de masse en tandem (TMT).

But scientifique: Dans le tutoriel sur les dataframes, vous avez travaillé sur les données de ***transcriptome du modèle UUO***. Dans ce mini-projet, vous allez travailler sur les données du ***transcriptome du modèle FA*** afin de regrouper les observations (échantillon) et les gènes selon des profils d'expression similaires.

Votre projet se décompose en 4 parties:

1. statsitiques descriptives des données brutes: commandes fournies
2. normalisation des données : commandes fournies
3. statistiques descriptives des données normalisées: à vous de jouer
4. analyse de regroupement des données: à vous de jouer

1. Les données brutes

Vous n'avez rien à coder ici. Le code est fourni.

Chargement des données brutes

Le bloc suivant contient une fonction qui permet de télécharger un fichier dans l'espace de travail, sauf s'il est déjà présent. Nous l'utiliserons ensuite pour télécharger les données à analyser en évitant de refaire le transfert à chaque exécution de l'analyse.

```
#' @title Download a file only if it is not yet here
#' @author Jacques van Helden email{Jacques.van-Helden@@france-bioinformatique.fr}
#' @param url_base base of the URL, that will be prepended to the file name
#' @param file_name name of the file (should not contain any path)
#' @param local_folder path of a local folder where the file should be stored
#' @return the function returns the path of the local file, built from local_folder and file_name
#' @export©
download_only_once <- function(
  url_base,
  file_name,
  local_folder) {

  ## Define the source URL
  url <- file.path(url_base, file_name)
```

```
  message("Source URL\n\t",  url)

  ## Define the local file
  local_file <- file.path(local_folder, file_name)

  ## Create the local data folder if it does not exist
  dir.create(local_folder, showWarnings = FALSE, recursive = TRUE)

  ## Download the file ONLY if it is not already there
  if (!file.exists(local_file)) {
    message("Downloading file from source URL to local file\n\t",
            local_file)
    download.file(url = url, destfile = local_file)
  } else {
    message("Local file already exists, no need to download\n\t",
            local_file)
  }

  return(local_file)
}
```

Nous téléchargeons deux fichiers dans un dossier local `~/m3-stat-R/pavkovic_analysis` **(vous pouvez changer le nom ou chemin dans le chunk ci-dessous)**, et les chargeons dans les data.frames suivants:

- Données brutes de transcriptome: `fa_expr_raw`
- Métadonnées: `fa_meta`

```
## Define the remote URL and local folder
pavkovic_url <- "https://github.com/DU-Bii/module-3-Stat-R/raw/master/stat-R_2021/data/pavkovic_2019/"

## Define the local folder for this analysis (where the data will be downloaded and the results generat
pavkovic_folder <- "~/m3-stat-R/pavkovic_analysis"

## Define a sub-folder for the data
pavkovic_data_folder <- file.path(pavkovic_folder, "data")

## Download and load the expression data table
## Note: we use check.names=FALSE to avoid replacing hyphens by dots
## in sample names, because we want to keep them as in the
## original data files.
message("Downloading FA transcriptome file\t", "fa_raw_counts.tsv.gz",
  "\n\tfrom\t", pavkovic_url)
fa_expr_file <- download_only_once(
  url_base = pavkovic_url,
  file_name = "fa_raw_counts.tsv.gz",
  local_folder = pavkovic_data_folder)

## Load the expresdsion table
message("Loading FA transcriptome data from\n\t", fa_expr_file)
fa_expr_raw <- read.delim(file = fa_expr_file,
                          header = TRUE,
                          row.names = 1)

## Download the metadata file
message("Downloading FA metadata file\t", "fa_transcriptome_metadata.tsv",
```

```
  "\n\tfrom\t", pavkovic_url)
fa_meta_file <- download_only_once(
  url_base = pavkovic_url,
  file_name = "fa_transcriptome_metadata.tsv",
  local_folder = pavkovic_data_folder)

## Load the metadata
message("Loading FA metadata from\n\t", fa_meta_file)
fa_meta <- read.delim(file = fa_meta_file,
                      header = TRUE,
                      row.names = 1)
```

Nous regardons la structure de chaque dataframe.

```
str(fa_expr_raw)
```

```
'data.frame':   46679 obs. of  18 variables:
 $ day1_1  : num  2278.8 0 36.3 13.2 0 ...
 $ day1_2  : num  1786.5 0 22.15 7.15 27.9 ...
 $ day1_3  : num  2368.62 0 39.48 1.12 6.9 ...
 $ day14_1 : num  627.758 0 14.471 0.867 5.692 ...
 $ day14_2 : num  559.2 0 10.2 0 1.9 ...
 $ day14_3 : num  611.434 0 31.691 0 0.655 ...
 $ day2_1  : num  2145.22 0 300.56 1.71 57.38 ...
 $ day2_2  : num  262.45 0 4.77 0 0 ...
 $ day2_3  : num  745.84 0 123.9 5.26 38.9 ...
 $ day3_1  : num  987.185 0 51.856 0.802 8.931 ...
 $ day3_2  : num  1077.65 0 8.43 0 6.97 ...
 $ day3_3  : num  1335.1 0 69.9 0 0 ...
 $ day7_1  : num  1096.08 0 6.67 0 7.94 ...
 $ day7_2  : num  1035.846 0 6.955 0.849 101.648 ...
 $ day7_3  : num  1090.04 0 42.58 1.71 0.65 ...
 $ normal_1: num  483.23 0 7.35 0.86 32.06 ...
 $ normal_2: num  1842.1 0 11.2 0 10.4 ...
 $ normal_3: num  475.7 0 1.03 0 0 ...
```

```
str(fa_meta)
```

```
'data.frame':   18 obs. of  5 variables:
 $ dataType    : chr  "transcriptome" "transcriptome" "transcriptome" "transcriptome" ...
 $ sampleName  : chr  "day14_1" "day14_2" "day14_3" "day1_1" ...
 $ condition   : chr  "day14" "day14" "day14" "day1" ...
 $ sampleNumber: int  1 2 3 1 2 3 1 2 3 1 ...
 $ color       : chr  "#FF4400" "#FF4400" "#FF4400" "#BBD7FF" ...
```

Les deux fichiers ne donnent pas les observations de l'échantillon dans le même ordre:

```
fa_meta$sampleName == names(fa_expr_raw)
```

```
 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TR
```

Nous les réorganisons les échantillons dans l'ordre de l'expérience: condition normale, puis day 1 à 14 avec les 3 réplicats.

```
sample_order <- c(paste(rep(c("normal", "day1", "day2", "day3", "day7", "day14"), each = 3),
                        1:3, sep = "_"))
```

```
fa_expr_raw <- fa_expr_raw[,sample_order]
fa_meta <- fa_meta[match(sample_order, fa_meta$sampleName),]

# View(fa_meta)
kable(fa_meta, caption = "Metdata for Pavkovoc FA transcriptome")
```

Table 2: Metdata for Pavkovoc FA transcriptome

	dataType	sampleName	condition	sampleNumber	color
16	transcriptome	normal_1	normal	1	#BBFFBB
17	transcriptome	normal_2	normal	2	#BBFFBB
18	transcriptome	normal_3	normal	3	#BBFFBB
4	transcriptome	day1_1	day1	1	#BBD7FF
5	transcriptome	day1_2	day1	2	#BBD7FF
6	transcriptome	day1_3	day1	3	#BBD7FF
7	transcriptome	day2_1	day2	1	#F0BBFF
8	transcriptome	day2_2	day2	2	#F0BBFF
9	transcriptome	day2_3	day2	3	#F0BBFF
10	transcriptome	day3_1	day3	1	#FFFFDD
11	transcriptome	day3_2	day3	2	#FFFFDD
12	transcriptome	day3_3	day3	3	#FFFFDD
13	transcriptome	day7_1	day7	1	#FFDD88
14	transcriptome	day7_2	day7	2	#FFDD88
15	transcriptome	day7_3	day7	3	#FFDD88
1	transcriptome	day14_1	day14	1	#FF4400
2	transcriptome	day14_2	day14	2	#FF4400
3	transcriptome	day14_3	day14	3	#FF4400

=> Ainsi, nous avons un jeu de données avec un échantillon de 18 observations et des données d'expression de 46679 gènes.

Transformation log2

Appliquez une transformation log2 des données brutes, après avoir ajouté un epsilon $\epsilon = 1$ (les valeurs nulles seront donc représentées par un log2(counts) valant 0. Stockez le résultat dans un data.frame nommé `fa_expr_log2`.

Affchez un fragment des tableaux `fa_expr_raw` et `fa_expr_log2` en sélectionnant les lignes 100 à 109 et les colonnes 5 à 10, afin de vous assurer que la transformation log2 a bien fonctionné.

```
## Log2 transformation of the transcriptome data
epsilon <- 1
fa_expr_log2 <- log2(fa_expr_raw + epsilon)
# dim(fa_expr_log2)
# View(head(fa_expr_log2))

## Display of a fragment of the data before and after log2 transformation
kable(fa_expr_raw[100:109, 5:10], caption = "Fragment des données transcriptomiques brutes")
```

Table 3: Fragment des données transcriptomiques brutes

	day1_2	day1_3	day2_1	day2_2	day2_3	day3_1
ENSMUSG00000000567	599.648075	304.093177	1052.689447	106.995584	347.13842	479.59911

	day1_2	day1_3	day2_1	day2_2	day2_3	day3_1
ENSMUSG00000000568	808.026306	1349.126716	818.257157	116.136417	3406.45030	766.09722
ENSMUSG00000000579	599.832586	500.735597	473.399472	36.258005	410.57787	347.05054
ENSMUSG00000000581	842.511039	744.646735	546.260344	87.788535	319.12679	461.45732
ENSMUSG00000000594	3463.845705	2743.404374	2283.051270	1115.588250	1491.61384	1576.68451
ENSMUSG00000000600	3841.854530	3561.805180	3674.188108	589.840068	1399.10751	3446.01856
ENSMUSG00000000605	791.312561	558.710943	489.579657	77.008213	256.00200	282.80077
ENSMUSG00000000606	4.785252	6.566374	3.970399	0.000000	138.48677	0.00000
ENSMUSG00000000617	15.839486	29.138902	30.228506	6.670500	18.27493	13.41152
ENSMUSG00000000627	36.673777	35.967747	46.297517	2.878275	17.03638	15.45854

```
kable(fa_expr_log2[100:109, 5:10], caption = "Fragment des données transcriptomiques après transformati
```

Table 4: Fragment des données transcriptomiques après transformation log2

	day1_2	day1_3	day2_1	day2_2	day2_3	day3_1
ENSMUSG00000000567	9.230376	8.253106	10.041234	6.754829	8.443517	8.908690
ENSMUSG00000000568	9.978748	10.398879	9.678173	6.872046	11.734477	9.583266
ENSMUSG00000000579	9.230819	8.970783	8.889959	5.219479	8.685022	8.443153
ENSMUSG00000000581	9.881896	9.542348	9.096084	6.472302	8.322500	8.853176
ENSMUSG00000000594	11.581599	11.422277	11.157379	10.124882	10.543625	10.623593
ENSMUSG00000000600	11.706865	11.798798	11.843602	9.206624	10.451322	11.751133
ENSMUSG00000000605	9.629926	9.128538	8.938344	6.285554	8.005636	8.148735
ENSMUSG00000000606	2.532380	2.919602	2.313362	0.000000	7.123984	0.000000
ENSMUSG00000000617	4.073776	4.913555	4.964792	2.939321	4.268654	3.849151
ENSMUSG00000000627	5.235489	5.208195	5.563693	1.955415	4.172838	4.040765

Statistiques descriptives

Dans le tutorial sur les dataframes sur le jeu de données “uuo” (relisez le corrigé), nous vous avons demandé de créer un data.frame qui collectera les statistiques par gène et par échantillon. Nous vous demandons de réaliser une étude similaire sur les données “FA”.

Par échantillon avant normalisation

Nous créons un data.frame nommé `sample_stat_prenorm` qui comportera une ligne par échantillon et une colonne par statistique. Nous calculons les statistiques suivantes sur les valeurs log2 d’expression de chaque échantillon:

- moyenne
- écart-type
- intervalle inter-quartiles
- premier quartile
- médiane
- troisième quartile
- maximum
- nombre de valeurs nulles

Il sera affiché avec la fonction `kable()` (n’oubliez pas la légende).

```
sample_stat_prenorm <- data.frame(
  mean = apply(fa_expr_log2, 2, mean, na.rm=TRUE),
```

```
  sd = apply(fa_expr_log2, 2, sd, na.rm=TRUE),
  iqr = apply(fa_expr_log2, 2, IQR, na.rm=TRUE),
  Q1 = apply(fa_expr_log2, 2, quantile, p = 0.25, na.rm=TRUE),
  median = apply(fa_expr_log2, 2, median, na.rm=TRUE),
  Q3 = apply(fa_expr_log2, 2, quantile, p = 0.75, na.rm=TRUE),
  max = apply(fa_expr_log2, 2, max, na.rm=TRUE),
  null = apply(fa_expr_log2 == 0, 2, sum, na.rm=TRUE)
)

kable(sample_stat_prenorm, caption = "Sample-wise statistics before normalisation.")
```

Table 5: Sample-wise statistics before normalisation.

	mean	sd	iqr	Q1	median	Q3	max	null
normal_1	2.876096	3.382687	5.566333	0	1.115495	5.566333	23.48952	21415
normal_2	3.475984	3.851531	6.702346	0	1.991260	6.702346	24.43355	20203
normal_3	2.796962	3.301372	5.419783	0	1.057748	5.419783	23.47218	21660
day1_1	3.003791	3.526143	5.901203	0	1.074190	5.901203	23.65130	21700
day1_2	3.516314	3.877108	6.816493	0	2.019190	6.816493	24.58225	20152
day1_3	3.607632	3.917601	6.946987	0	2.252981	6.946987	24.70005	19621
day2_1	3.608242	3.944769	7.000535	0	2.170400	7.000535	24.32636	19978
day2_2	2.088503	2.832614	3.958139	0	0.000000	3.958139	21.94520	24446
day2_3	3.071767	3.573520	6.008117	0	1.309292	6.008117	23.47538	21455
day3_1	3.233264	3.638371	6.300924	0	1.621087	6.300924	23.41316	20772
day3_2	3.163046	3.595828	6.231985	0	1.556653	6.231985	23.03797	21142
day3_3	3.440775	3.777905	6.727257	0	1.991235	6.727257	23.97460	20287
day7_1	3.296409	3.667439	6.515860	0	1.703173	6.515860	23.34983	20865
day7_2	3.251819	3.587848	6.350051	0	1.822801	6.350051	23.20695	20467
day7_3	3.289691	3.617495	6.400883	0	1.954087	6.400883	23.53623	20045
day14_1	3.102341	3.484017	6.044672	0	1.597272	6.044672	23.41418	20496
day14_2	3.147517	3.557991	6.130466	0	1.600695	6.130466	23.87931	20870
day14_3	3.192445	3.533419	6.250143	0	1.699228	6.250143	23.46460	20396

Par gène avant normalisation

Nous créons ci-dessous un data.frame nommé `gene_stat_prenorm` qui comportera une ligne par gène et une colonne par statistique. Nous calculons les statistiques suivantes sur les valeurs log2 de chaque gène.

- moyenne
- médiane
- écart-type
- premier quartile
- troisième quartile
- maximum
- nombre de valeurs nulles
- intervalle inter-quartiles

Ces résultats seront stockés dans un data.frame avec 1 ligne par échantillon et 1 colonne par statistique. Vous afficherez les lignes 100 à 109 de ce tableau de statistiques avec la fonction `kable()` (n'oubliez pas la légende).

```
## Gene-wise statistics before normalisation
gene_stat_prenorm <- data.frame(
  mean = apply(fa_expr_raw, 1, mean, na.rm = TRUE),
  sd = apply(fa_expr_raw, 1, sd, na.rm = TRUE),
```

```
  iqr = apply(fa_expr_raw, 1, IQR, na.rm = TRUE),
  Q1 = apply(fa_expr_raw, 1, quantile, p = 0.25, na.rm = TRUE),
  median = apply(fa_expr_raw, 1, median, na.rm = TRUE),
  Q3 = apply(fa_expr_raw, 1, quantile, p = 0.75, na.rm = TRUE),
  max = apply(fa_expr_raw, 1, max, na.rm = TRUE),
  null = apply(fa_expr_raw == 0, 1, sum, na.rm = TRUE)
)

kable(gene_stat_prenorm[100:109, ], caption = "Gene-wise statistics before normalisation")
```

Table 6: Gene-wise statistics before normalisation

	mean	sd	iqr	Q1	median	Q3	max	null
ENSMUSG00000032536744	325.636744	286.399084	364.362298	91.046357	261.659973	455.40866	1052.68945	0
ENSMUSG00000057638825	857.638825	694.432189	282.978949	575.571199	711.096105	858.55015	3406.45030	0
ENSMUSG00000022775702	422.775702	162.290415	157.732606	340.417017	447.572742	498.14962	728.51737	0
ENSMUSG00000040578227	405.578227	248.612029	259.324338	320.599085	471.269224	579.92342	942.51104	0
ENSMUSG00000020158334	2015.458334	1237.521694	937.571105	1400.226182	1868.718762	2337.79729	6408.96044	0
ENSMUSG00000022978163	2269.978163	911.222891	1480.192582	1719.874980	2093.572929	3200.06756	3674.18811	0
ENSMUSG00000032552118	325.552118	169.456624	195.879561	221.735904	282.608205	417.61546	791.31256	0
ENSMUSG00000000029	9.000258	32.151350	2.736779	1.178311	1.934744	3.91509	138.48677	4
ENSMUSG00000018074116	18.074116	9.470819	13.654701	12.188276	18.150391	25.84298	36.00235	0
ENSMUSG00000015623306	15.623306	14.341852	12.308127	4.780913	10.064895	17.08904	46.29752	0

2. Filtrage et normalisation des données

Vous n'avez rien à coder ici. Le code est fourni.

Il existe plusieurs façons de normaliser les données de transcriptome vues dans les modules 4 et 5 (cf. total counts, quantiles, TMM, RLE, limma voom,...), mais nous avons choisi ici une solution simple tout en étant robuste pour normaliser les données en standardisant le 3ème quantile.

La méthode choisie ici consiste à :

- écarter les gènes "non-détectés", c'est-à-dire ceux ayant des valeurs nulles dans au moins 90% des échantillons;
- écarter les gènes à peine exprimés, c'est-à-dire ceux ayant une valeur moyenne < 10 (arbitrairement);
- standardiser les échantillons sur le 3ème quartile des valeurs non-nulles: on divise par le 3ème quartile de l'échantillon et on multiplie par le 3ème quartile déterminé sur l'ensemble des échantillons.

Nous fournissons ci-dessous le code.

Filtrage 1 : élimination des gènes non détectés ou à peine exprimés

```
## Data filtering: genes having at least 90% null values
undetected_genes <- gene_stat_prenorm$null >= ncol(fa_expr_raw) * 0.9
print(paste0("Undetected genes (null in >= 90% samples): ", sum(undetected_genes)))
```

```
[1] "Undetected genes (null in >= 90% samples): 14380"
```

```
## Data filtering: genes having a mean expression < 10
barely_expressed_genes <- gene_stat_prenorm$mean < 10
print(paste0("Barely expressed genes (mean < 10): ", sum(barely_expressed_genes)))
```

```
[1] "Barely expressed genes (mean < 10): 26286"
```

```
## Apply filtering on both criteria
discarded_genes <- undetected_genes | barely_expressed_genes
print(paste0("Discarded genes: ", sum(discarded_genes)))
```

```
[1] "Discarded genes: 26288"
```

```
kept_genes <- !discarded_genes
print(paste0("Kept genes: ", sum(kept_genes)))
```

```
[1] "Kept genes: 20391"
```

```
## Genes after filtering
fa_expr_log2_filtered <- fa_expr_log2[kept_genes, ]
```

Normalisation entre échantillons

```
## Generate a data frame where null values are replaced by NA
fa_expr_nonull <- fa_expr_log2_filtered
fa_expr_nonull[fa_expr_log2_filtered <= 0] <- NA
sum(is.na(fa_expr_nonull))
```

```
[1] 5598
```

```
## Compute the 3rd quartile of non-null values for each sample and store them in a vector:
sample_q3_nonull <- apply(fa_expr_nonull, 2, quantile, prob = 0.75, na.rm = TRUE)
# print(sample_q3_nonull)

## Compute the Q3 for all the values, which will serve as target value for the standardised sample Q3
all_q3_nonull <- quantile(unlist(fa_expr_nonull), prob = 0.75, na.rm = TRUE)
# print(all_q3_nonull)

## Standardise expression on the third quartile of non-null values
## Beware : for this standardization we keep the null values
## Trick : we transpose the table to apply the ratio sample per sample,
## and then transpose the results to get back the genes in rows and samples in columns
fa_expr_log2_standard <- t(t(fa_expr_log2_filtered) * all_q3_nonull / sample_q3_nonull )
# quantile(unlist(fa_expr_log2_standard), probs = 0.75, na.rm = TRUE)

## We also compute the values for the "nonull" table for
## the sake of comparison and to check that the third quantiles of non-null
## values are well identical across samples.
fa_expr_log2_standard_nonull <- t(t(fa_expr_nonull) * all_q3_nonull / sample_q3_nonull )
# quantile(unlist(fa_expr_log2_standard_nonull), probs = 0.75, na.rm = TRUE)

## Compute Q3 before and after standardisation, including or not the null values
standardisation_impact <- data.frame(
  before_all = apply(fa_expr_log2_filtered, 2, quantile, prob =  0.75, na.rm = TRUE),
  before_nonull = apply(fa_expr_nonull, 2, quantile, prob =  0.75, na.rm = TRUE),
  after_nonul = apply(fa_expr_log2_standard_nonull, 2, quantile, prob =  0.75, na.rm = TRUE),
  after_all = apply(fa_expr_log2_standard, 2, quantile, prob =  0.75, na.rm = TRUE)
)

## Note: after standardization the Q3 of the data show some variations
## because we compute them here with the null values
```

```
kable(standardisation_impact, caption = "Impact of standardization on the third quantile (Q3) per sample
```

Table 7: Impact of standardization on the third quantile (Q3) per sample. Third quantiles are computed before and after standardisation, with either all the values of the filtered table, or only the non-null values.

	before_all	before_nonull	after_nonul	after_all
normal_1	7.892971	7.929471	8.546124	8.506785
normal_2	9.095259	9.120100	8.546124	8.522846
normal_3	7.690496	7.728764	8.546124	8.503808
day1_1	8.275875	8.310843	8.546124	8.510165
day1_2	9.215737	9.236819	8.546124	8.526618
day1_3	9.339495	9.356787	8.546124	8.530330
day2_1	9.372222	9.391459	8.546124	8.528618
day2_2	6.401122	6.563217	8.546124	8.335056
day2_3	8.428494	8.462042	8.546124	8.512242
day3_1	8.600914	8.618327	8.546124	8.528857
day3_2	8.449052	8.476090	8.546124	8.518862
day3_3	8.929730	8.945829	8.546124	8.530744
day7_1	8.637420	8.652095	8.546124	8.531628
day7_2	8.457495	8.478531	8.546124	8.524920
day7_3	8.568555	8.585032	8.546124	8.529721
day14_1	8.181662	8.194184	8.546124	8.533064
day14_2	8.339087	8.364105	8.546124	8.520562
day14_3	8.313689	8.334105	8.546124	8.525188

3. Les données normalisées

A vous de jouer!

Statistiques par gène après normalisation

Générez un data.frame avec une ligne par gène à partir du tableau de données normalisées, avec les statistiques suivantes (une statistique par colonne):

- moyenne
- variance
- écart-type
- coefficient de variation (écart-type divisé par la moyenne)
- intervalle inter-quartiles
- minimum
- médiane
- maximum

```
## Gene-wise statistics after normalisation
gene_stat_norm <- data.frame(
  mean = apply(fa_expr_log2_standard, 1, mean, na.rm = F),
  var = apply(fa_expr_log2_standard, 1, var, na.rm = TRUE),
  sd = apply(fa_expr_log2_standard, 1, sd, na.rm = F),
  CV = NA,
  min = apply(fa_expr_log2_standard, 1, min, na.rm = TRUE),
  Q1 = apply(fa_expr_log2_standard, 1, quantile, p = 0.25, na.rm = TRUE),
  median = apply(fa_expr_log2_standard, 1, median, na.rm = TRUE),
```

```
  Q3 = apply(fa_expr_log2_standard, 1, quantile, p = 0.75, na.rm = TRUE),
  max = apply(fa_expr_log2_standard, 1, max, na.rm = TRUE),
  null = apply(fa_expr_log2_standard == 0, 1, sum, na.rm = TRUE)
  )
gene_stat_norm$CV <- gene_stat_norm$sd / gene_stat_norm$mean
```

Annotation des gènes

Chaque gène étant donné par son identifiant dans la base de données ENSEMBL vous utiliserez le `paquet biomaRt de bioconductor` pour ajouter des annotations : symbole, chromosome, coordonnées génomiques, brin. Suivez pas à pas la méthode proposée:

- sélectionnez la base de données ENSEMBL avec la fonction `useMart()`. Attention à choisir le bon génome avec l'agument `dataset`: `mmusculus_gene_ensembl`
- avec la fonction `getBM()` récupérez de la base de données ENSEMBL les champs demandés (***pour symbole utilisez external__gene__name***) en appliquant "ensembl_geneid" pour l'agument `filter` et en indiquant pour l'argument `values` le vecteur des identifiants des gènes présents dans le dataframe `gene_stat_norm`. Vous obtenez un dataframe.

A présent, ajoutez au dataframe `gene_stat_norm` en 1ères colonnes les annotations retrouvées grâce à biomaRt. Attention, certains gènes ne sont pas retrouvés dans la version d'ENSEMBL sur biomaRt donc laissez des NA comme données manquantes dans ce cas. Nous vous recommandons d'utiliser la function `merge()` ou `left_join()` de dplyr pour fusionner les deux dataframes en un seul.

```
ensembl <- useMart("ENSEMBL_MART_ENSEMBL", host = "www.ensembl.org", dataset = "mmusculus_gene_ensembl")

genes <-getBM(attributes = c("ensembl_gene_id", "external_gene_name", "chromosome_name",
                             "start_position", "end_position", "strand"),
              filter = "ensembl_gene_id",
              values = row.names(gene_stat_norm),
              mart = ensembl
             )


gene_stat_norm <- merge(genes, gene_stat_norm, by.x = "ensembl_gene_id", by.y = 0, sort = FALSE)

kable(gene_stat_norm[100:109, ], caption = "Gene-wise statistics after normalisation")
```

Table 8: Gene-wise statistics after normalisation

	ensembl_gene_id	external_gene_name	chromosome_name	start_position	end_position	strand	mean	var	sd	CV	min	Q1	median	Q3	max	null
100	ENSMUSG00000001724	C1qb	1	77609447	77616109	-1	5.595006	0.197284	0.444166	0.079386	4.831695	5.423054	5.632556	5.829461	6.426584	0
101	ENSMUSG00000001731	Gbp6	1	77865856	77879444	1	4.936591	0.586832	0.766005	0.155137	3.791542	4.333682	4.844505	5.101682	6.247104	0
102	ENSMUSG00000001732	Gbp4	1	77905136	77919747	1	5.543721	0.585901	0.765008	0.138002	4.153459	5.017496	5.473647	5.856682	7.250127	0
103	ENSMUSG00000001738	Gbp7	1	123789681	123824499	1	8.426883	0.274644	0.524006	0.062189	7.477862	8.245083	8.430482	8.699996	9.484120	0
104	ENSMUSG00000001740	Gbp3	1	123829081	123831983	1	10.82083	0.333074	0.577047	0.053371	10.22564	10.56109	10.85025	11.09661	12.16229	0
105	ENSMUSG00000001743	Gbp1	1	123931002	123939502	1	7.619613	0.179407	0.423566	0.055572	6.611012	7.107027	7.398774	7.908298	8.866373	0
106	ENSMUSG00000001751	Gpa1	1	75188997	75239150	1	7.368976	0.522601	0.722982	0.098153	6.384878	6.865172	7.241978	7.676994	9.010795	0
107	ENSMUSG00000001753	Septin1	1	75300597	75313527	-1	2.830629	0.968941	0.984230	0.347906	0.000000	2.259503	2.731434	3.459656	5.177214	4
108	ENSMUSG00000001759	Gbp2b	1	126642771	126722481	1	8.617902	0.302733	0.550212	0.063875	7.458437	8.083891	8.497889	8.879162	10.03960	0
109	ENSMUSG00000001766	Cpne1	1	67085066	67098198	1	5.356086	0.639897	0.799993	0.149335	3.297268	4.832552	5.193353	5.537567	7.401143	0

Challenge falcultatif:

Enfin, réordonnez les gènes par position génomique et affichez les lignes 5 premières et 5 dernières lignes de ce tableau de statistiques.

```
gene_stat_norm <- gene_stat_norm[order(gene_stat_norm$chromosome_name, gene_stat_norm$start_position),]

kable(gene_stat_norm[c(1:5, (nrow(gene_stat_norm)-4):nrow(gene_stat_norm)), ], caption = "Gene-wise sta
```

Table 9: Gene-wise statistics after normalisation.

	ensembl_gene_id	external_gene_name	chromosome_name	start_position	end_position	strand	mean	var	sd	CV	min	Q1	median	Q3	max	null
11991	[illegible]	[illegible]	[illegible]	3276124	3741721	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
17939	[illegible]	[illegible]	[illegible]	3435954	3438772	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
17870	[illegible]	[illegible]	[illegible]	3663115	3666126	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
17551	[illegible]	[illegible]	[illegible]	3717532	3729127	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
17647	[illegible]	[illegible]	[illegible]	3822233	3824583	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
17896	[illegible]	[illegible]	[illegible]	16691330	16697033	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
18295	[illegible]	[illegible]	[illegible]	40144240	40147214	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
17372	[illegible]	[illegible]	[illegible]	68997874	69008469	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
16671	[illegible]	[illegible]	[illegible]	90796007	90827734	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
17213	[illegible]	[illegible]	[illegible]	90848689	90855309	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Distribution des données

- Dessinez sous forme d'un histogramme la distribution des valeurs après normalisation (tous échantillons confondus)

```
hist(unlist(fa_expr_log2_standard),
     breaks = seq(from = 0, to = max(fa_expr_log2_standard) + 1, by = 0.25),
     xlab = "log2(counts) after standardisation",
     ylab = "number of genes after filtering",
     col = "#BBDDFF",
     las = 1, cex.axis = 0.8,
     main = "distribution after standardisation")
abline(v = mean(fa_expr_log2_standard), col = "darkgreen", lwd = 2)
```

- Dessinez un box plot des échantillons avant et après normalisation, et commentez la façon dont l'effet de la normalisation apparaît sur ces graphiques.

```
#### Box plots to show normalisation impact ####
par(mar = c(4,6,4,1)) ## Set the margins
par(mfrow = c(2,2))
boxplot(fa_expr_log2_filtered,
        horizontal = TRUE,
        xlab = "log2(counts)",
        las = 1,
        col = fa_meta$color,
        main = "Before standardisation\nall values")
boxplot(fa_expr_nonull,
        horizontal = TRUE,
        xlab = "log2(counts)",
        las = 1,
        col = fa_meta$color,
        main = "Before standardisation\nzeros discarded")
boxplot(fa_expr_log2_standard_nonull,
        horizontal = TRUE,
```

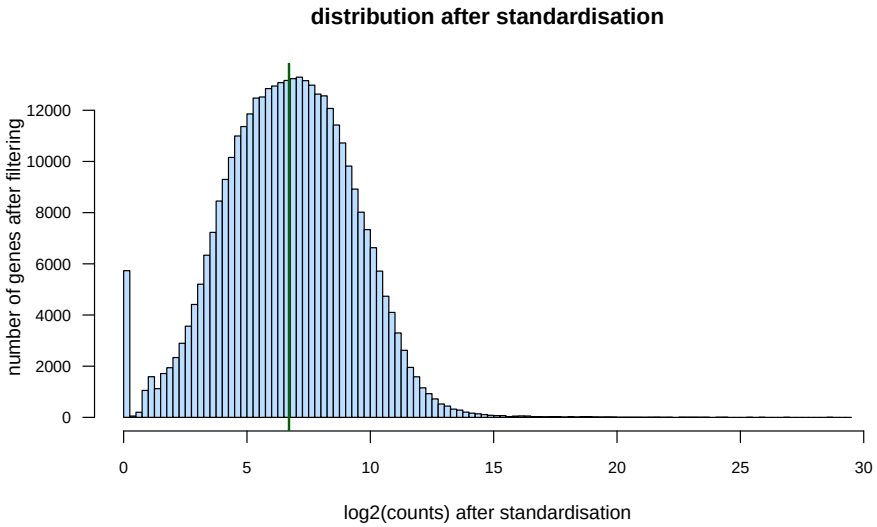


Figure 1: Distribution of expression values (log2 counts) after gene filtering and standardisation on the sample-wise third-quartile of non-null values. The vertical line highlights the mean value.

```
        xlab = "log2(counts)",
        las = 1,
        col = fa_meta$color,
        main = "Standardised\nzeros discarded")
boxplot(fa_expr_log2_standard,
        xlab = "log2(counts)",
        las = 1,
        horizontal = TRUE,
        col = fa_meta$color,
        main = "Standardised\nall values")
```

```
par(mfrow = c(1, 1))
par(mar = c(4,5,5,1))
```

4. Analyse de regroupement des données

Filtrage 2 : sélection de gènes d'expression élevée et variable

Pour réduire le nombre de gènes, nous allons écarter les gènes faiblement exprimés (log2 moyen inférieur à 4), et ne retenir que ceux qui montrent des variations importantes entre échantillons. Pour ce dernier critère, nous nous basons sur le coefficient de variation afin de relativiser la dispersion (écart type) par rapport à la tendance centrale (moyenne).

Sélectionnez les gènes ayant un niveau log2 moyen minimal supérieur à 3 ($s > 3$) et un coefficient de variation supérieur à 0.5 ($CV > 0.5$). Note: ces valeurs sont parfaitement arbitraires, elles ont été choisies pour obtenir un nombre raisonnable de gènes.

```
## Compute a Boolean vector indicating whether each gene passes or not the expression level threshold
high_expression <- gene_stat_norm$mean > 3
# table(high_expression) # count number of genes with high/weak expression

## Compute a Boolean vector indicating whether each gene passes or not the variation coefficient thresho
high_variation <- gene_stat_norm$CV > 0.5
# table(high_variation) # count number of genes with weak high coeffficient of variation
```

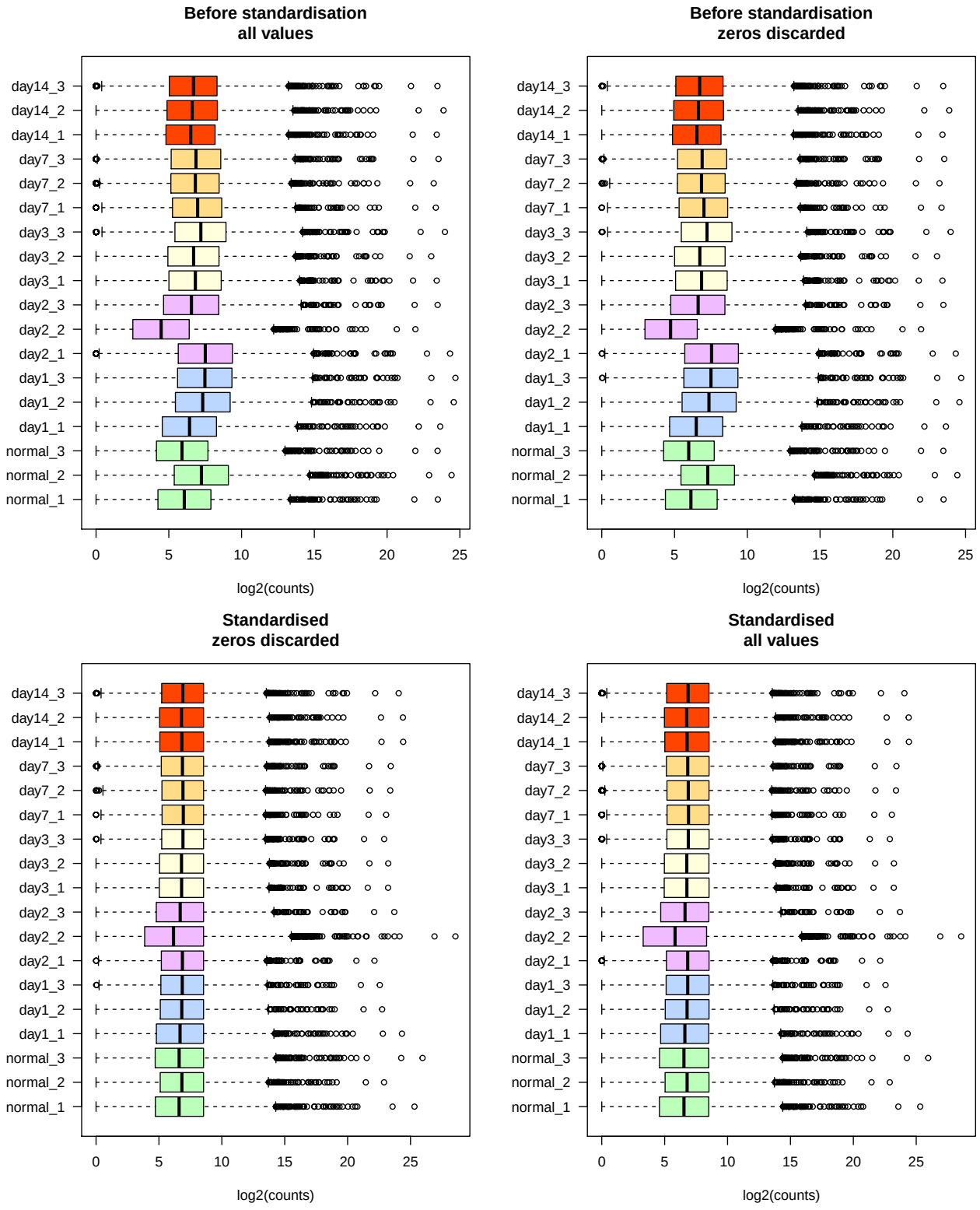


Figure 2: Box plots showing the impact of normalisation

```
## Compute a Boolean vector indicating whether each gene passes or not the variance threshold
# high_variance <- gene_stat_norm$var > 2
# table(high_variance) # count number of genes with weak high variance

## Select genes having both a high mean expression and a high variation coefficien
selected_genes <- high_variation & high_expression
# table(selected_genes) # count number of genes with weak high coeffficient of variation
print(paste0("Selected genes: ", sum(selected_genes)))
```

```
[1] "Selected genes: 473"
```

```
## Create a data frame with the expression of the selected genes
fa_expr_selected <- fa_expr_log2_standard[selected_genes, ]
```

Dessinez des histogrammes des valeurs d'expression avant et après cette sélection de gènes, et commentez les différences.

```
#### Histograms of expression before and after gene selection ####

par(mfrow = c(2,1))
hist(unlist(fa_expr_log2_standard),
     breaks = seq(from = 0, to = max(fa_expr_log2_standard) + 1, by = 0.25),
     las = 1,
     cex.axis = 0.8,
     main = "Standardized values before gene selection",
     col =  "#DDBBFF")

hist(unlist(fa_expr_selected),
     breaks = seq(from = 0, to = max(fa_expr_log2_standard) + 1, by = 0.25),
     las = 1,
     cex.axis = 0.8,
     main = "Standardized values after gene selection",
     col =  "#FFDDBB")
```

```
par(mfrow = c(1,1))



# ## Some quick checks: the selection of highly variable genes select those having many zeros - and high
# hist(unlist(fa_expr_log2_filtered[high_expression, ]), breaks=100)
# hist(unlist(fa_expr_log2_filtered[high_variation, ]), breaks=100)
# hist(unlist(fa_expr_log2_filtered[!high_variation, ]), breaks=100)
# hist(unlist(fa_expr_log2_filtered[selected_genes, ]), breaks=100)
```

Dessinez un box plot par échantillon des valeurs d'expression avant et après sélection des gènes, et commentez le résultat.

```
#### Histogram of expression after gene selection ####

par(mfrow = c(1,2))

boxplot(fa_expr_log2_standard,
        horizontal = TRUE,
        xlab = "log2(counts)",
        las = 1,
        col = fa_meta$color,
        main = "Before gene selection\nstandardised values")
```

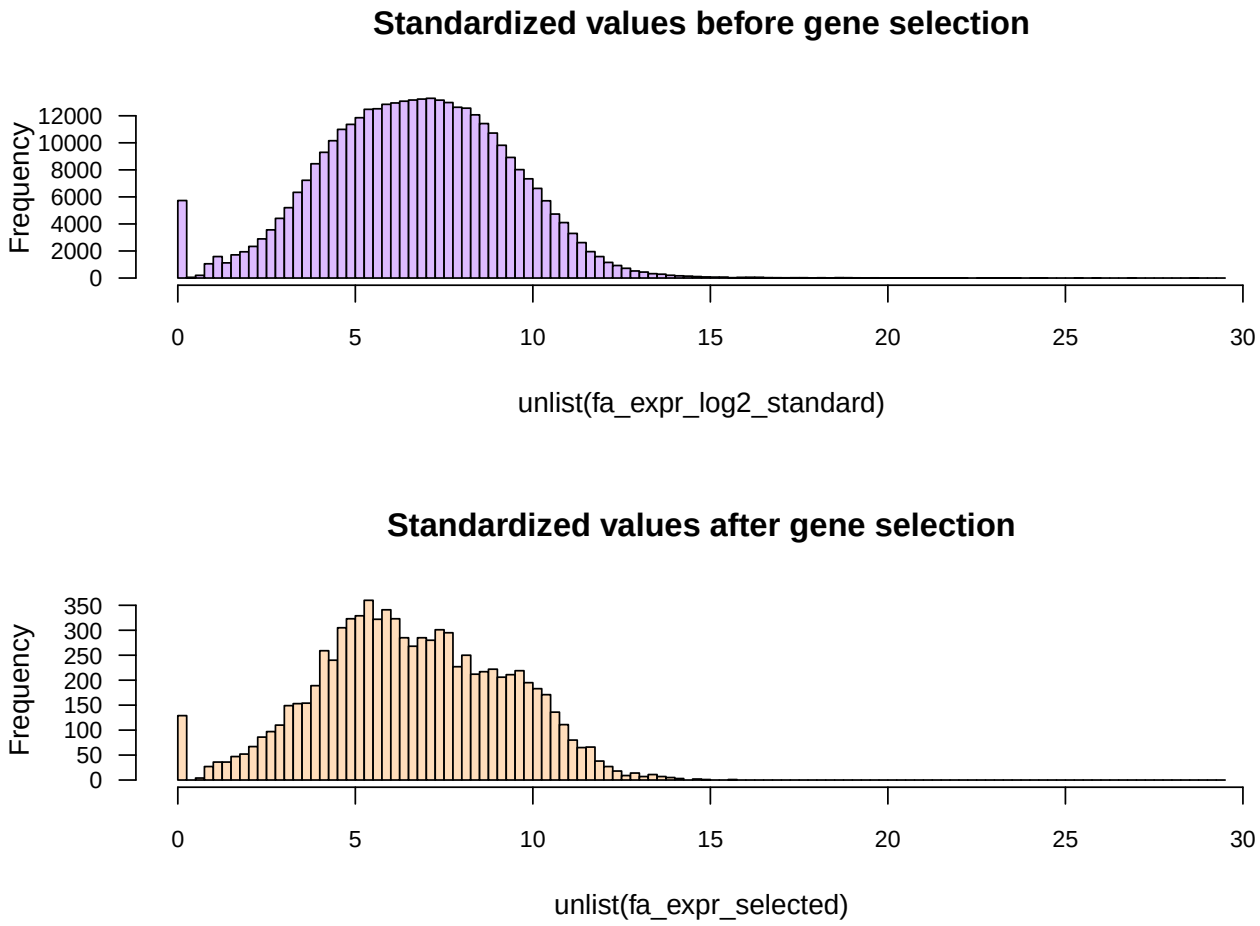


Figure 3: Distribution of expression values before and after gene selection

```
boxplot(fa_expr_selected,
        horizontal = TRUE,
        xlab = "log2(counts)",
        las = 1,
        col = fa_meta$color,
        main = "After gene selection\nstandardised values")
```

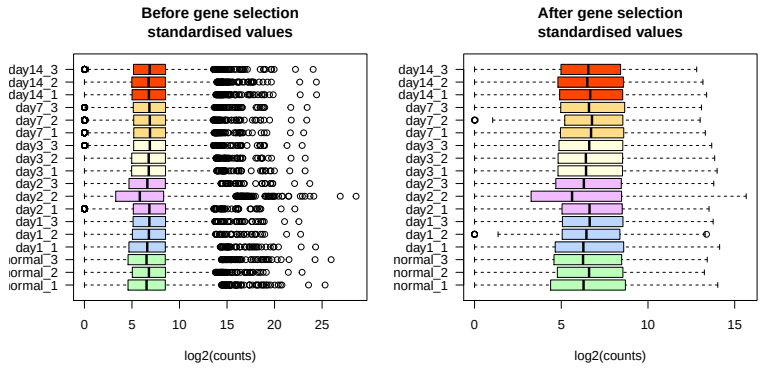


Figure 4: Box plots of standardised expression values before and after gene selection.

```
par(mfrow = c(1,1))
```

ACP

Dessinez un plot ACP des échantillons en les colorant par condition avant et après normalisation.

- avec les comptages bruts de la matrice d'expression initiale (fa_expr)

```
## Raw expression values, all genes
ma_pca_raw_tt <- PCA(t(fa_expr_raw),
                     scale.unit = FALSE,
                     graph = FALSE)
# plot(ma_pca_raw_tt, choix = "ind")
fviz_pca_ind(ma_pca_raw_tt, col.ind = fa_meta[, "color"])
```

- avec la matrice de valeurs filtrées et après transformation log2

```
ma_pca_filtered <- PCA(t(fa_expr_log2_filtered), scale.unit = FALSE,
                   graph = FALSE)
# plot(ma_pca_filtered, choix = "var")
# plot(ma_pca_sel, choix = "ind")
fviz_pca_ind(ma_pca_filtered, col.ind = fa_meta[, "color"])
```

- avec la matrice finale (transformation log2, filtre des gènes non-détectés, standardisation et sélection des gènes fortement exprimés et à haut coefficient de variation)

```
ma_pca_sel <- PCA(t(fa_expr_selected), scale.unit = FALSE,
                  graph = FALSE)
# plot(ma_pca_sel, choix = "var")
# plot(ma_pca_sel, choix = "ind")
fviz_pca_ind(ma_pca_sel, col.ind = fa_meta[, "color"])
```

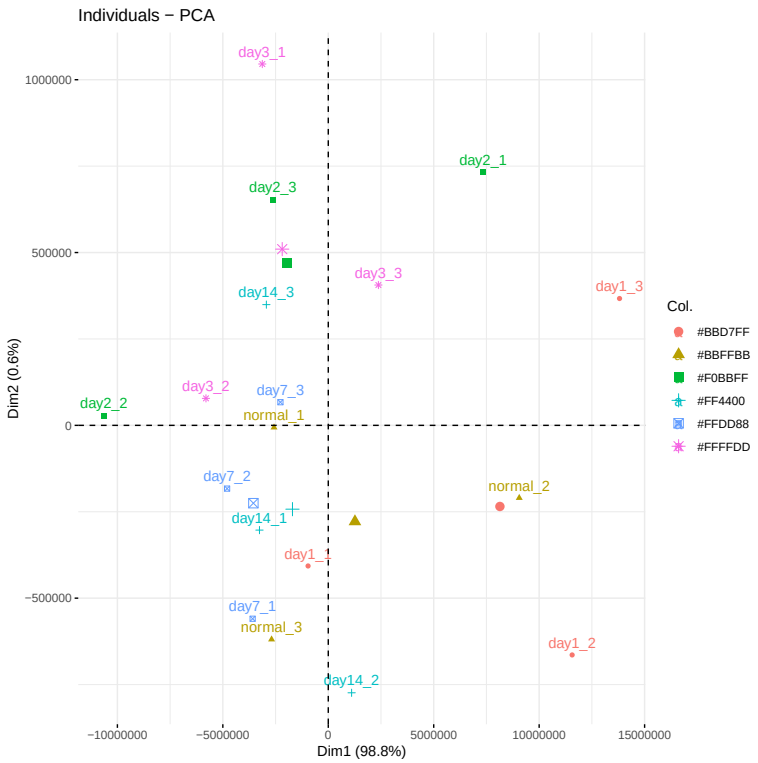


Figure 5: PC plot of the samples from the raw expression values of all genes.

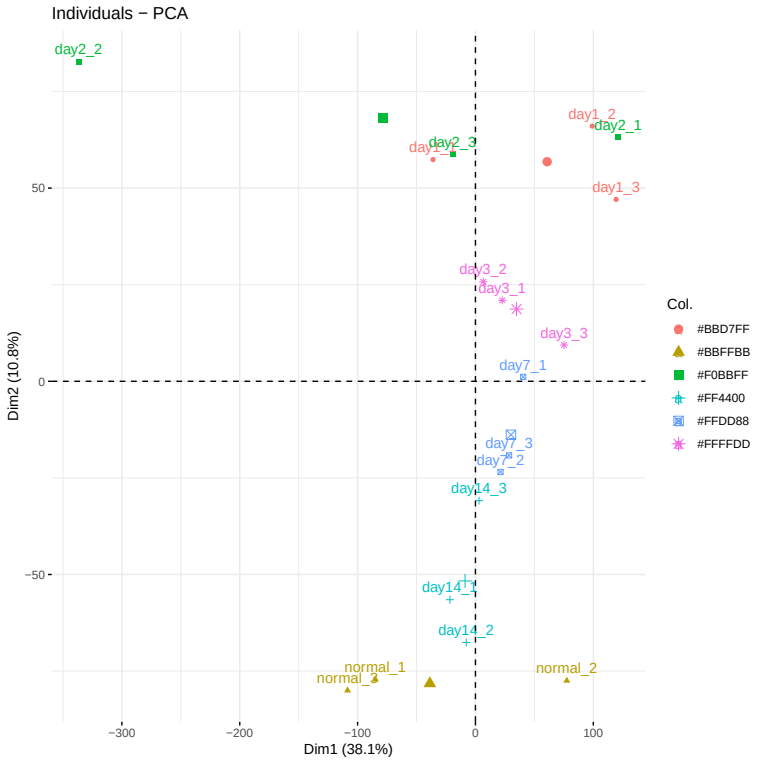


Figure 6: PC plot of the samples from standardised values after gene selection.

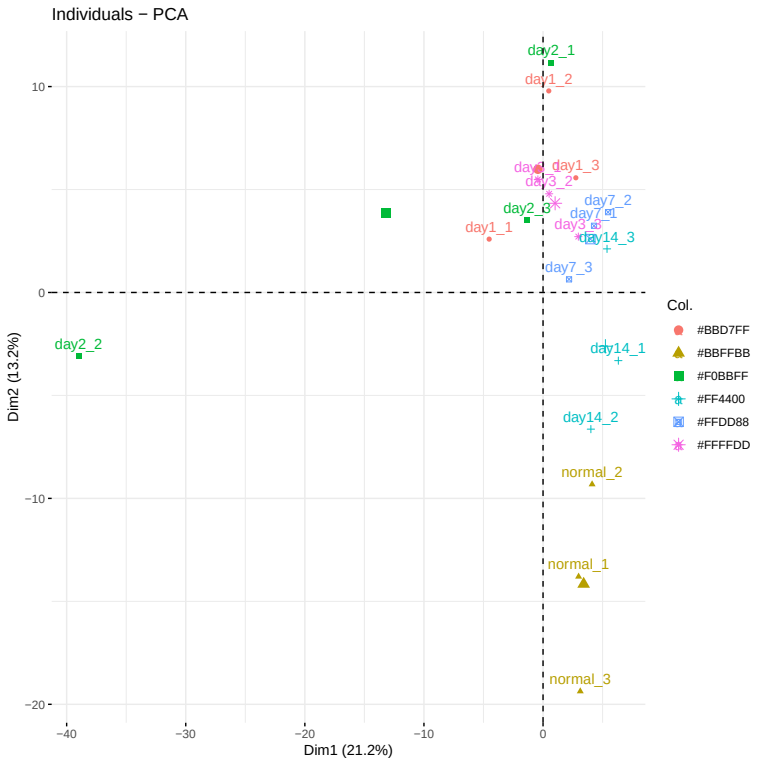


Figure 7: PC plot of the samples from standardised values after gene selection.

Clustering

- Calculez les matrices de distance entre échantillons, en utilisant respectivement les distances euclidienne (`dist()`), coefficient de Pearson (`cor(, method = "pearson")`) et de Spearman (`cor(, method = "spearman")`).

```
#### Sample distances ####
dist_euc_sel <- dist(t(fa_expr_selected))
cor_pearson_sel <- as.dist(1 - cor(fa_expr_selected))
cor_spearman_sel <- 1 - as.dist(cor(fa_expr_selected, method = "spearman"))
```

- Effectuez un clustering hiérarchique des échantillons, en utilisant le critère de Ward (`ward.d2`) pour l'agglomération. Comparez les arbres d'échantillons obtenus avec ces trois métriques et choisissez celle qui vous paraît la plus pertinente.

```
#### Sample clustering ####
par(mfrow = c(1,3))
plot(hclust(dist_euc_sel), hang = -1,
     main = "euclidean distance")
plot(hclust(cor_pearson_sel), hang = -1,
     main = "pearson")
plot(hclust(cor_spearman_sel), hang = -1,
     main = "spearman")
```

```
par(mfrow = c(1,1))
```

- Effectuez un clustering hiérarchique des gènes en utilisant la distance basée sur le coefficient de Pearson et le critère de Ward

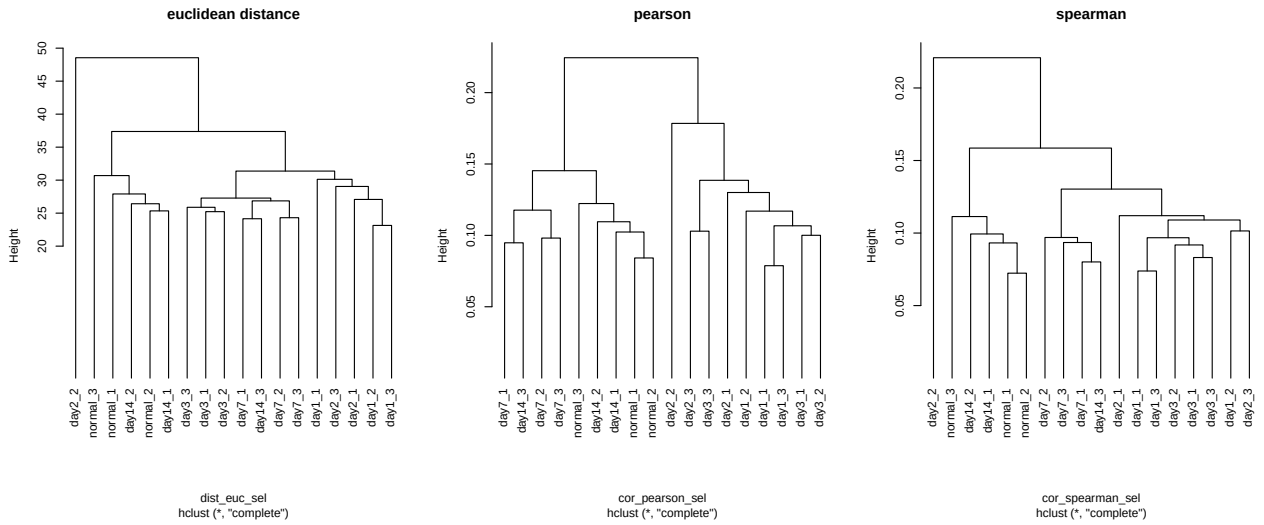
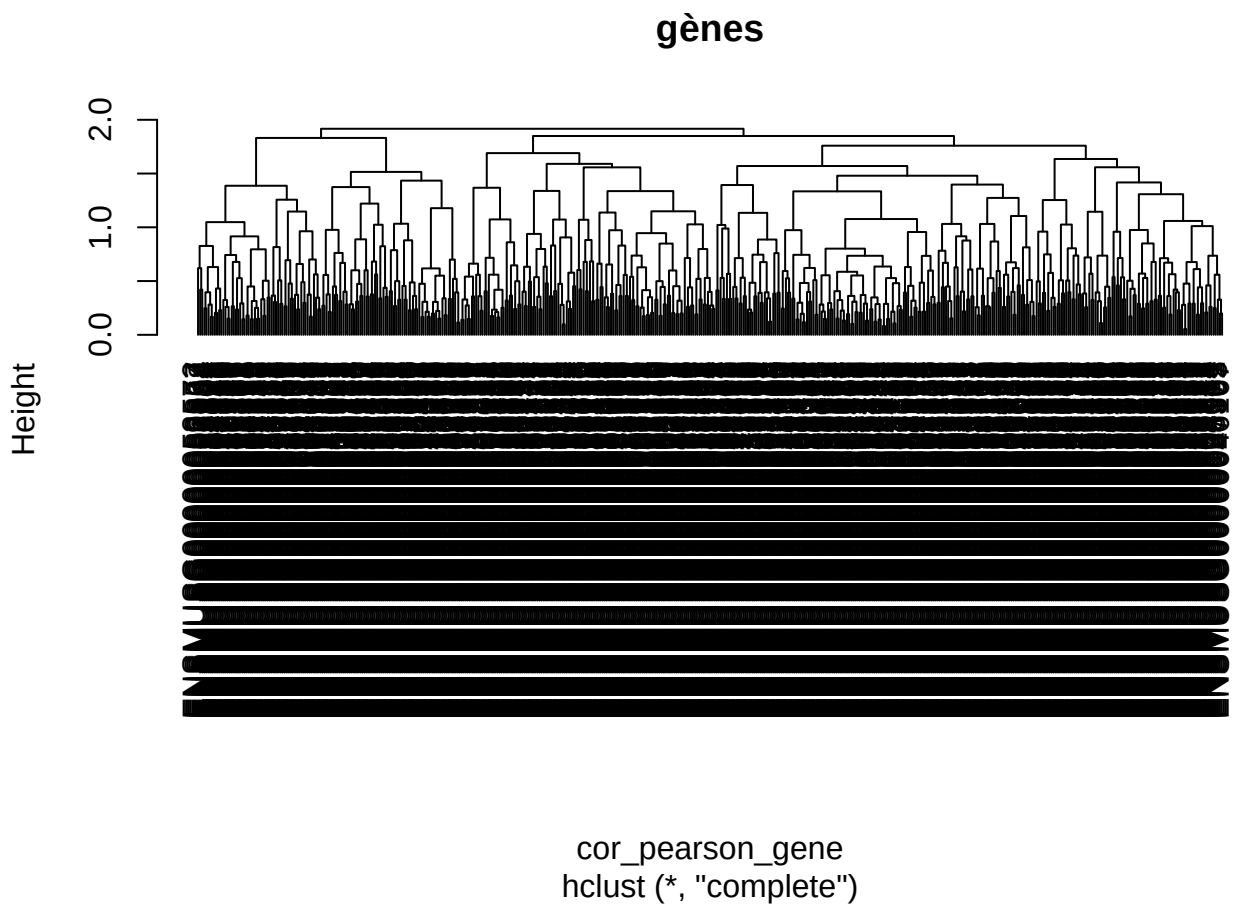


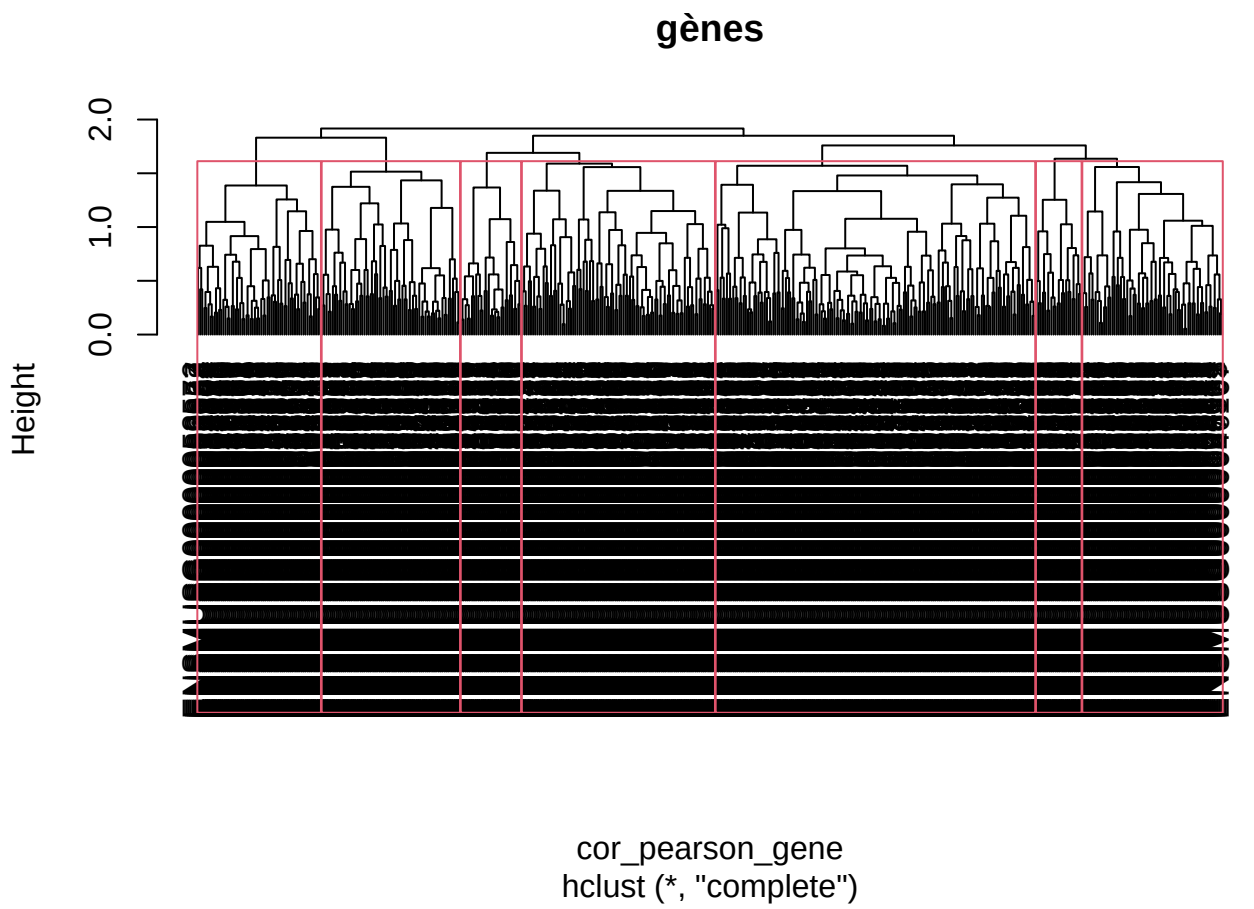
Figure 8: Sample tree with three alternative distance metrics: Euclidiant distance (left), Pearson correlation (center), Spearman correlation (right).?

```
cor_pearson_gene <- as.dist(1 - cor(t(fa_expr_selected)))
plot(hclust(cor_pearson_gene), hang = -1,
     main = "gènes")
```



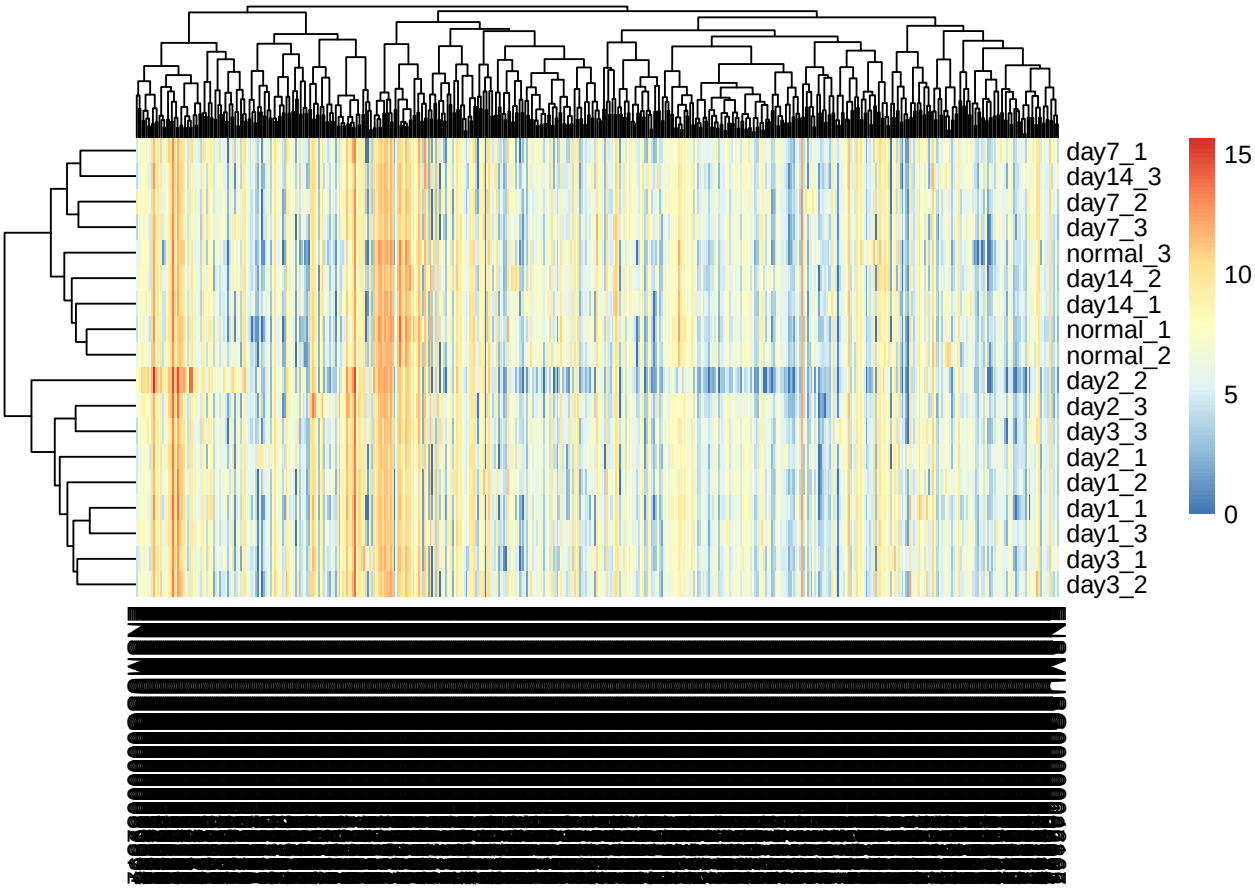
- Dessinez un arbre avec le résultat du clustering des gènes et commentez sa strcuture. Si vous deviez choisir de façon arbitraire un nombre de clusters, que choisiriez-vous ? Pourquoi ? Pas de panique, nous pouvons assumer ici que la réponse comporte une part de subjectivité.

```
plot(hclust(cor_pearson_gene), hang = -1,
     main = "gènes")
rect.hclust(hclust(cor_pearson_gene), k = 7)
```



- Dessinez une heatmap du résultat, en sélectionnant les deux résultats de clustering ci-dessus pour les gènes et les échantillons.

```
pheatmap(t(fa_expr_selected),
         clustering_distance_cols = "correlation", clustering_distance_rows = "correlation")
```



Interprétez les résultats en quelques phrases.

Enrichissement fonctionnel

Effectuez une analyse d'enrichissement fonctionnel avec les principaux clusters obtenus dans la section précédente.

Conclusions générales

Résumez en quelques phrases vos conclusions à partir des résultats obtenus.