Analyses Phyloseq

##A ne pas prendre en compte

```
load("02_data-analysis-with-DADA2")
```

```
ps_connect <-url("https://raw.githubusercontent.com/spholmes/F1000_workflow/master/data/ps.rds")
ps = readRDS(ps_connect)
ps
```

```
## Loading required package: phyloseq
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 389 taxa and 360 samples ]
## sample_data() Sample Data:       [ 360 samples by 14 sample variables ]
## tax_table()   Taxonomy Table:    [ 389 taxa by 6 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 389 tips and 387 internal nodes ]
```

Bonus : Passage à Phyloseq

```
library(phyloseq)
library(ggplot2)
theme_set(theme_bw())
```

Nous pouvons construire un simple échantillon de data.frame à partir des informations encodées dans les noms de fichiers. Habituellement, cette étape consisterait plutôt à lire les données de l'échantillon à partir d'un fichier.

```
samples.out <- rownames(seqtab.nochim)
subject <- sapply(strsplit(samples.out, "D"), `[`, 1)
gender <- substr(subject,1,1)
subject <- substr(subject,2,999)
day <- as.integer(sapply(strsplit(samples.out, "D"), `[`, 2))
samdf <- data.frame(Subject=subject, Gender=gender, Day=day)
samdf$When <- "Early"
samdf$When[samdf$Day>100] <- "Late"
rownames(samdf) <- samples.out
```

Nous construisons un objet phyloseq directement à partir des sorties de dada2.

```
ps <- phyloseq(otu_table(seqtab.nochim, taxa_are_rows=FALSE),
               sample_data(samdf),
               tax_table(taxa))
ps <- prune_samples(sample_names(ps) != "Mock", ps) # Remove mock sample
```

Il est plus pratique d'utiliser des noms courts pour les variants de séquence d'amplicon (ASVs) plutôt que la séquence d'ADN complète lorsque l'on travaille avec certains tableaux et visualisations de phyloseq. Cependant, il est important de conserver les séquences d'ADN pour d'autres objectifs comme la fusion avec d'autres ensembles de données ou l'indexation dans des bases de données de référence (cf. Earth Microbiome Projet). Nous allons stocker les séquences d'ADN des ASVs dans l'extension refseq de l'objet phyloseq puis renommer les taxons en une courte chaîne. De cette façon, les noms courts des nouveaux taxons apparaîtront dans les tableaux et les graphiques, et nous pouvons toujours récupérer les séquences d'ADN correspondant à chaque ASVs avec refseq(ps).

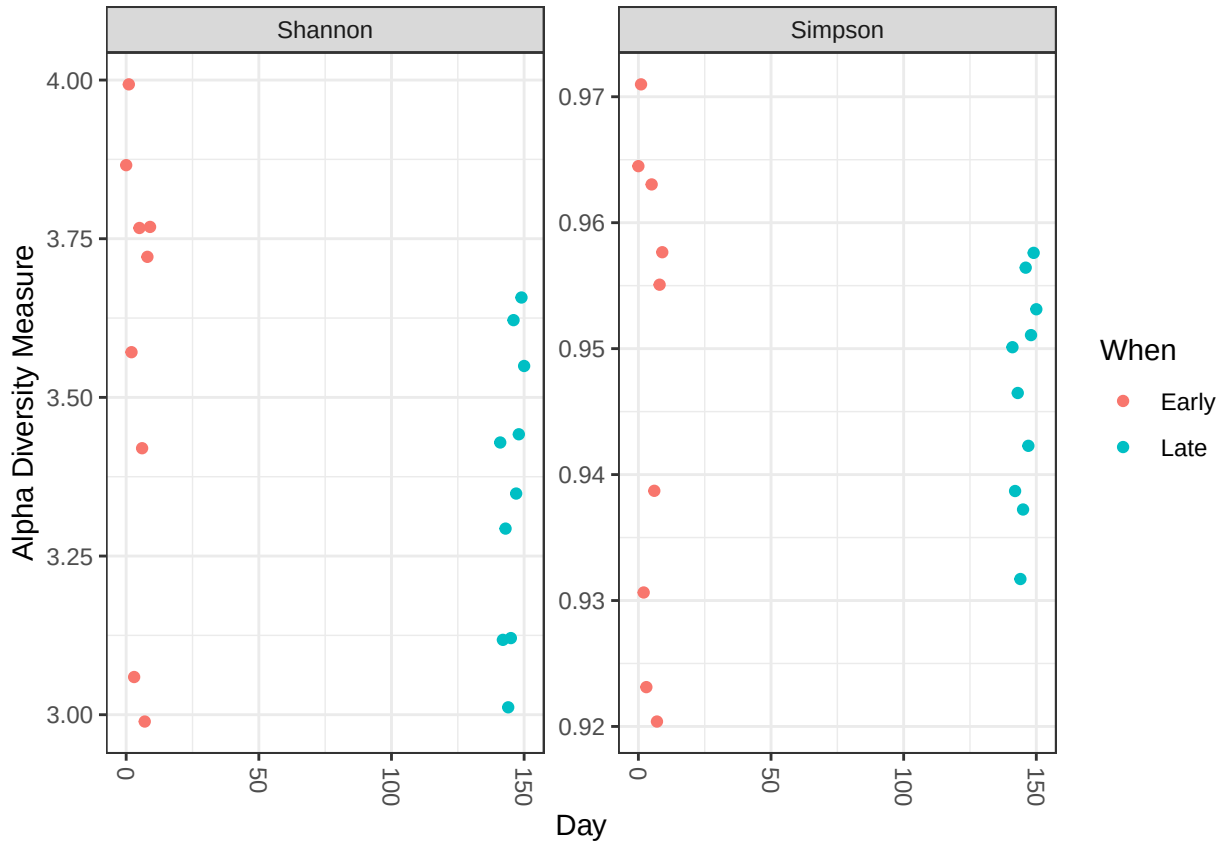
```
dna <- Biostrings::DNAStringSet(taxa_names(ps))
names(dna) <- taxa_names(ps)
ps <- merge_phyloseq(ps, dna)
taxa_names(ps) <- paste0("ASV", seq(ntaxa(ps)))
ps
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 232 taxa and 19 samples ]
## sample_data() Sample Data:       [ 19 samples by 4 sample variables ]
## tax_table()   Taxonomy Table:    [ 232 taxa by 7 taxonomic ranks ]
## refseq()      DNAStringSet:      [ 232 reference sequences ]
```

Nous pouvons maintenant visualiser la diversité alpha :

```
plot_richness(ps, x="Day", measures=c("Shannon", "Simpson"), color="When")
```

```
## Warning in estimate_richness(physeq, split = TRUE, measures = measures): The data you have provided c
## any singletons. This is highly suspicious. Results of richness
## estimates (for example) are probably unreliable, or wrong, if you have already
## trimmed low-abundance taxa from the data.
##
## We recommended that you find the un-trimmed data and retry.
```



Nous ne remarquons pas de différence systématique évidente de la diversité alpha entre les échantillons précoces et tardifs.

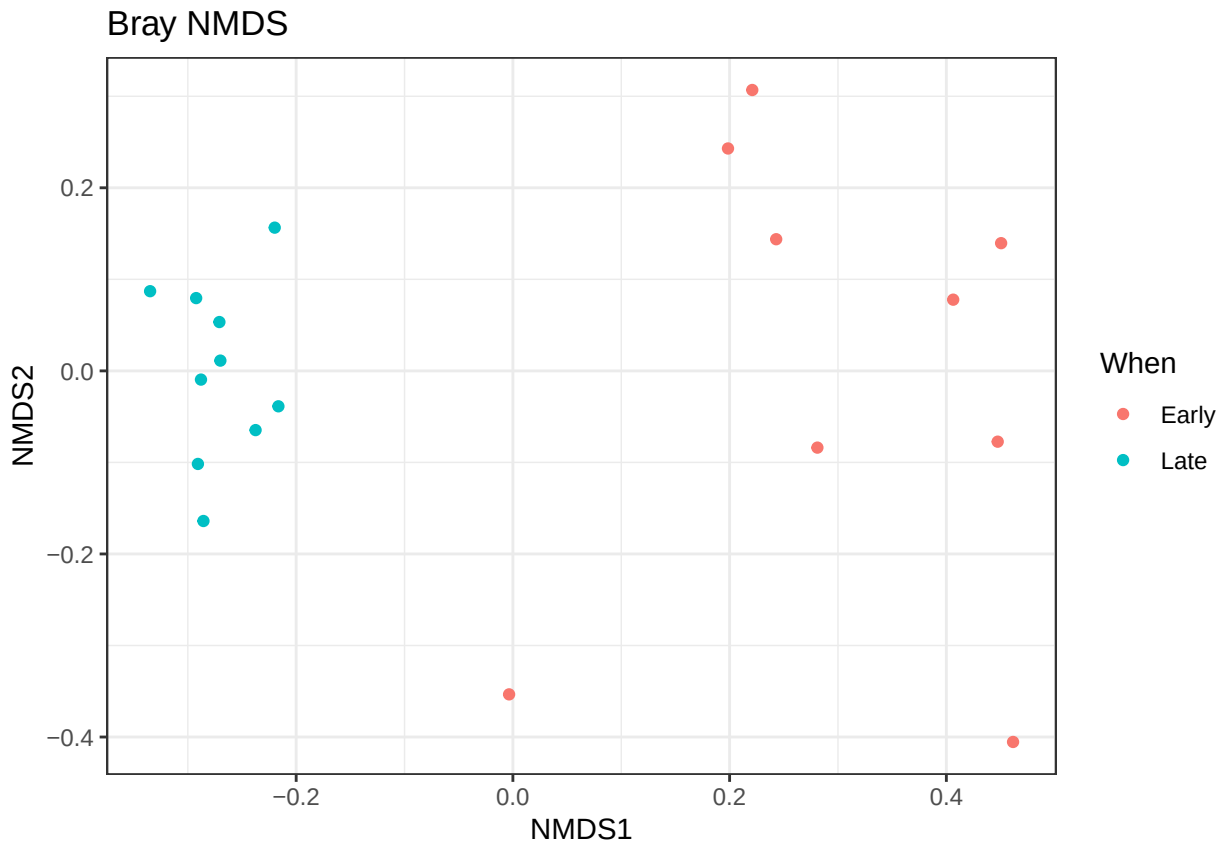
Nous allons ensuite procéder à une ordination :

```
# Transform data to proportions as appropriate for Bray-Curtis distances
ps.prop <- transform_sample_counts(ps, function(otu) otu/sum(otu))
ord.nmds.bray <- ordinate(ps.prop, method="NMDS", distance="bray")
```

```
## Run 0 stress 0.08043117
## Run 1 stress 0.08616061
## Run 2 stress 0.08616067
## Run 3 stress 0.1010632
## Run 4 stress 0.08616061
## Run 5 stress 0.08043117
## ... Procrustes: rmse 6.549346e-06  max resid 1.336386e-05
## ... Similar to previous best
## Run 6 stress 0.08076342
## ... Procrustes: rmse 0.01058592  max resid 0.03259494
## Run 7 stress 0.1212044
## Run 8 stress 0.08616061
## Run 9 stress 0.09477283
## Run 10 stress 0.08076344
## ... Procrustes: rmse 0.01060638  max resid 0.03266232
## Run 11 stress 0.08989125
## Run 12 stress 0.1212044
```

```
## Run 13 stress 0.08043117
## ... New best solution
## ... Procrustes: rmse 2.04036e-06  max resid 5.386526e-06
## ... Similar to previous best
## Run 14 stress 0.08076356
## ... Procrustes: rmse 0.01064361  max resid 0.03278527
## Run 15 stress 0.08989028
## Run 16 stress 0.1228545
## Run 17 stress 0.09477279
## Run 18 stress 0.1442037
## Run 19 stress 0.08076366
## ... Procrustes: rmse 0.01071192  max resid 0.03300791
## Run 20 stress 0.08989183
## *** Solution reached
```

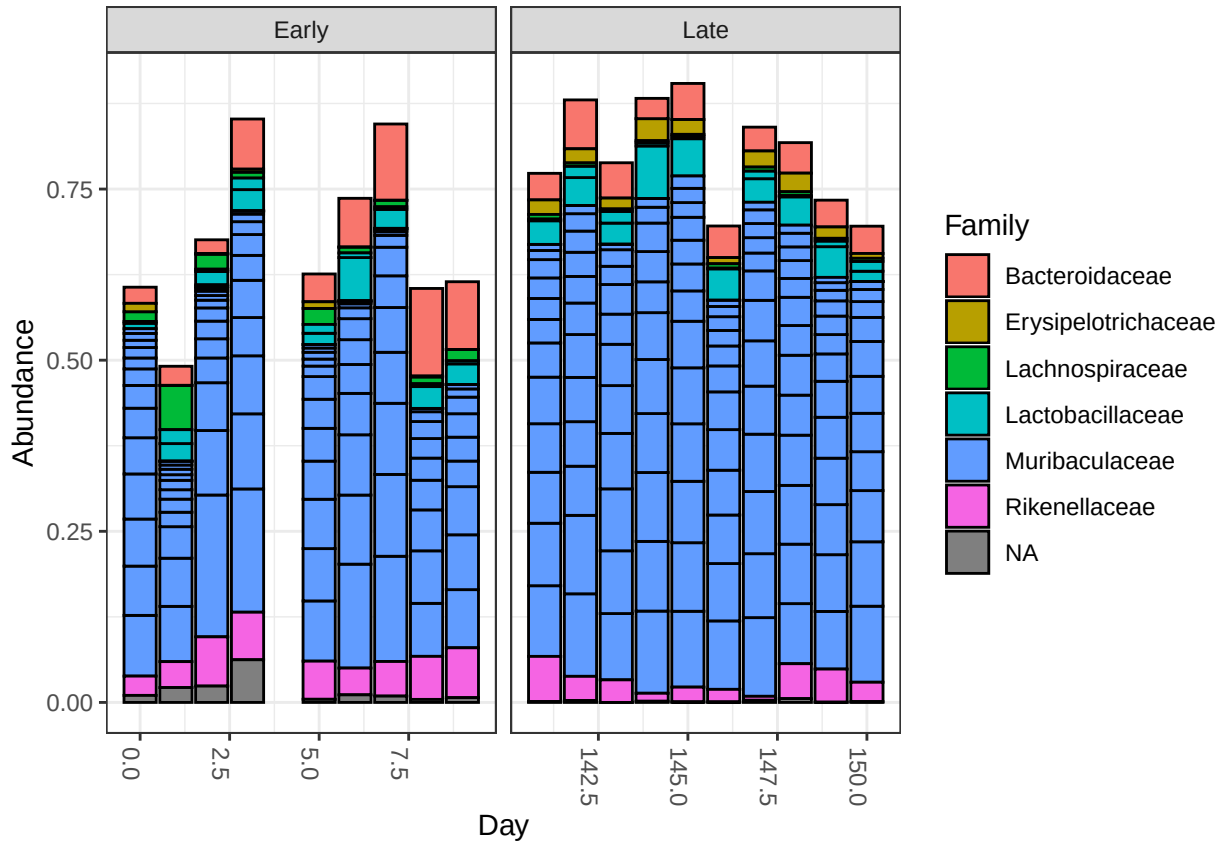
```
plot_ordination(ps.prop, ord.nmds.bray, color="When", title="Bray NMDS")
```



L'ordination a mis en évidence une séparation claire entre les échantillons tardifs et précoces.

Nous allons ensuite réaliser un diagramme en barres :

```
top20 <- names(sort(taxa_sums(ps), decreasing=TRUE))[1:20]
ps.top20 <- transform_sample_counts(ps, function(OTU) OTU/sum(OTU))
ps.top20 <- prune_taxa(top20, ps.top20)
plot_bar(ps.top20, x="Day", fill="Family") + facet_wrap(~When, scales="free_x")
```



Rien ne ressort de façon évidente au niveau de la distribution taxonomique des 20 premières séquences pour expliquer la différenciation entre les échantillons précoces et tardifs.

Utilisation de Phyloseq

```
set.seed(100)
```

Tutoriel Phyloseq

On définit la variable du chemin suivant pour qu'il va vers le répertoire extrait sur R :

```
miseq_path <- "./MiSeq_SOP" # CHANGE to the directory containing the fastq files after unzipping.
list.files(miseq_path)
```

```
##  [1] "F3D0_S188_L001_R1_001.fastq"   "F3D0_S188_L001_R2_001.fastq"
##  [3] "F3D1_S189_L001_R1_001.fastq"   "F3D1_S189_L001_R2_001.fastq"
##  [5] "F3D141_S207_L001_R1_001.fastq" "F3D141_S207_L001_R2_001.fastq"
##  [7] "F3D142_S208_L001_R1_001.fastq" "F3D142_S208_L001_R2_001.fastq"
##  [9] "F3D143_S209_L001_R1_001.fastq" "F3D143_S209_L001_R2_001.fastq"
## [11] "F3D144_S210_L001_R1_001.fastq" "F3D144_S210_L001_R2_001.fastq"
## [13] "F3D145_S211_L001_R1_001.fastq" "F3D145_S211_L001_R2_001.fastq"
```

```
## [15] "F3D146_S212_L001_R1_001.fastq" "F3D146_S212_L001_R2_001.fastq"
## [17] "F3D147_S213_L001_R1_001.fastq" "F3D147_S213_L001_R2_001.fastq"
## [19] "F3D148_S214_L001_R1_001.fastq" "F3D148_S214_L001_R2_001.fastq"
## [21] "F3D149_S215_L001_R1_001.fastq" "F3D149_S215_L001_R2_001.fastq"
## [23] "F3D150_S216_L001_R1_001.fastq" "F3D150_S216_L001_R2_001.fastq"
## [25] "F3D2_S190_L001_R1_001.fastq"   "F3D2_S190_L001_R2_001.fastq"
## [27] "F3D3_S191_L001_R1_001.fastq"   "F3D3_S191_L001_R2_001.fastq"
## [29] "F3D5_S193_L001_R1_001.fastq"   "F3D5_S193_L001_R2_001.fastq"
## [31] "F3D6_S194_L001_R1_001.fastq"   "F3D6_S194_L001_R2_001.fastq"
## [33] "F3D7_S195_L001_R1_001.fastq"   "F3D7_S195_L001_R2_001.fastq"
## [35] "F3D8_S196_L001_R1_001.fastq"   "F3D8_S196_L001_R2_001.fastq"
## [37] "F3D9_S197_L001_R1_001.fastq"   "F3D9_S197_L001_R2_001.fastq"
## [39] "filtered"                      "HMP_MOCK.v35.fasta"
## [41] "Mock_S280_L001_R1_001.fastq"   "Mock_S280_L001_R2_001.fastq"
## [43] "mouse.dpw.metadata"            "mouse.time.design"
## [45] "stability.batch"               "stability.files"
```

Filtrage et rabotage

Nous commençons par filtrer les lectures de faible qualité afin de les réduire à une longueur constante. Bien que les paramètres de filtrage et de rognage généralement recommandés servent de point de départ, il n'y a pas deux ensembles de données identiques et il est donc toujours utile d'inspecter la qualité des données avant de continuer.

Tout d'abord, nous lisons les noms des fichiers fastq, et effectuons une manipulation des chaînes pour obtenir les listes des fichiers fastq forward et reverse dans l'ordre correspondant :

```
# Sort ensures forward/reverse reads are in same order
fnFs <- sort(list.files(miseq_path, pattern="_R1_001.fastq"))
fnRs <- sort(list.files(miseq_path, pattern="_R2_001.fastq"))
# Extract sample names, assuming filenames have format: SAMPLENAME_XXX.fastq
sampleNames <- sapply(strsplit(fnFs, "_"), `[`, 1)
# Specify the full path to the fnFs and fnRs
fnFs <- file.path(miseq_path, fnFs)
fnRs <- file.path(miseq_path, fnRs)
fnFs[1:3]
```

```
## [1] "./MiSeq_SOP/F3D0_S188_L001_R1_001.fastq"
## [2] "./MiSeq_SOP/F3D1_S189_L001_R1_001.fastq"
## [3] "./MiSeq_SOP/F3D141_S207_L001_R1_001.fastq"
```

```
fnRs[1:3]
```

```
## [1] "./MiSeq_SOP/F3D0_S188_L001_R2_001.fastq"
## [2] "./MiSeq_SOP/F3D1_S189_L001_R2_001.fastq"
## [3] "./MiSeq_SOP/F3D141_S207_L001_R2_001.fastq"
```

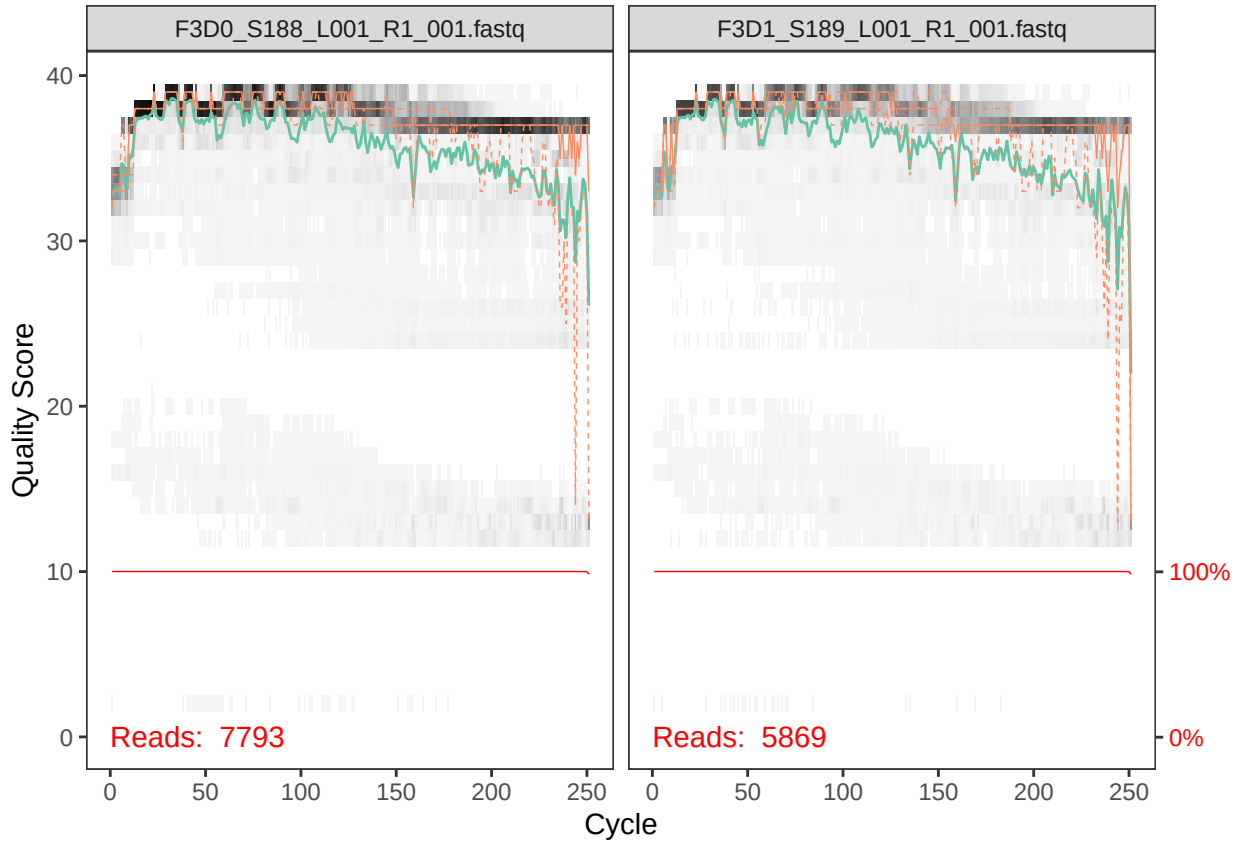
La plupart des données de séquençage Illumina montrent une tendance à la baisse de la qualité moyenne vers la fin des lectures de séquençage.

Les deux premiers forward se lisent comme suit :

```
library(dada2)
```

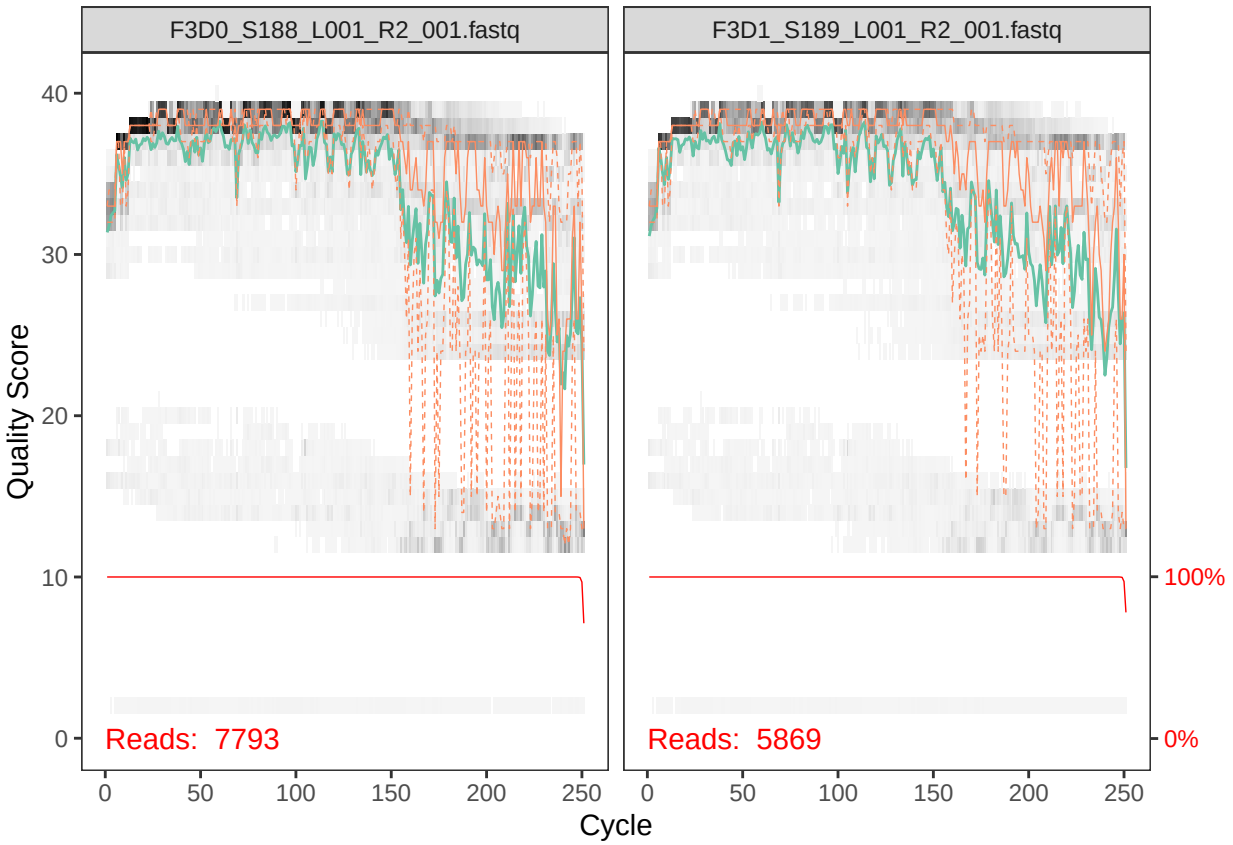
```
## Loading required package: Rcpp
```

```
plotQualityProfile(fnFs[1:2])
```



Les deux premiers reversent se lisent comme suit :

```
plotQualityProfile(fnRs[1:2])
```



Nous pouvons voir que les lectures forward conservent une haute qualité tout au long, tandis que la qualité des lectures reverse diminue considérablement à la position 160 environ. Par conséquent, nous choisissons de tronquer les lectures forward à la position 245, et les lectures reverse à la position 160. Nous choisissons également de découper les 10 premiers nucléotides de chaque lecture sur la base d'observations empiriques dans de nombreux ensembles de données Illumina puisque ces positions de base sont particulièrement susceptibles de contenir des erreurs pathologiques.

Nous définissons ensuite les noms des fichiers fastq.gz filtrés :

```
filt_path <- file.path(miseq_path, "filtered") # Place filtered files in filtered/ subdirectory
if(!file_test("-d", filt_path)) dir.create(filt_path)
filtFs <- file.path(filt_path, paste0(sampleNames, "_F_filt.fastq.gz"))
filtRs <- file.path(filt_path, paste0(sampleNames, "_R_filt.fastq.gz"))
```

Nous combinons ces paramètres de rognage avec des paramètres de filtrage standard, le plus important étant l'exécution d'un maximum de 2 erreurs attendues par lecture. Le découpage et le filtrage sont effectués conjointement sur des lectures appariées, c.-à-d. que les deux lectures doivent passer le filtre pour que la paire passe.

Les lectures forward et reverse sont ainsi filtrées avec la fonction filterAndTrim :

```
out <- filterAndTrim(fnFs, filtFs, fnRs, filtRs, truncLen=c(240,160),
              maxN=0, maxEE=c(2,2), truncQ=2, rm.phix=TRUE,
              compress=TRUE, multithread=TRUE) # On Windows set multithread=FALSE
head(out)
```

```
##                              reads.in reads.out
```

```
## F3D0_S188_L001_R1_001.fastq       7793      7113
## F3D1_S189_L001_R1_001.fastq       5869      5299
## F3D141_S207_L001_R1_001.fastq     5958      5463
## F3D142_S208_L001_R1_001.fastq     3183      2914
## F3D143_S209_L001_R1_001.fastq     3178      2941
## F3D144_S210_L001_R1_001.fastq     4827      4312
```

Inférer les variantes de séquence

Après filtrage, le séquençage typique d'amplicon se lit en unités taxonomiques opérationnelles (OTU) : groupes de lectures de séquençage qui diffèrent en dessous d'un seuil de dissemblance. Ici, nous utilisons plutôt la méthode DADA2 à haute résolution pour inférer exactement les variantes de séquence amplicon (ASVs), sans imposer de menace arbitraire, et résoudre ainsi les variantes qui diffèrent aussi peu que d'un nucléotide.

Les données de séquence sont importées dans R à partir de fichiers fastq démultiplexés (c.-à-d. un fastq pour chaque échantillon) et simultanément déréglées pour supprimer la redondance. Nous nommons les objets derep-class résultants par leur nom d'échantillon.

Déréplication

La déréplication combine toutes les lectures de séquence identiques en «séquences uniques» avec une «abondance» correspondante : le nombre de lectures avec cette séquence unique. La déréplication réduit considérablement le temps de calcul en éliminant les comparaisons redondantes.

```
derepFs <- derepFastq(filtFs, verbose=TRUE)
```

```
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D0_F_filt.fastq.gz

## Encountered 1979 unique sequences from 7113 total sequences read.

## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D1_F_filt.fastq.gz

## Encountered 1639 unique sequences from 5299 total sequences read.

## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D141_F_filt.fastq.gz

## Encountered 1477 unique sequences from 5463 total sequences read.

## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D142_F_filt.fastq.gz

## Encountered 904 unique sequences from 2914 total sequences read.

## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D143_F_filt.fastq.gz

## Encountered 939 unique sequences from 2941 total sequences read.

## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D144_F_filt.fastq.gz

## Encountered 1267 unique sequences from 4312 total sequences read.
```

```
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D145_F_filt.fastq.gz
## Encountered 1756 unique sequences from 6741 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D146_F_filt.fastq.gz
## Encountered 1438 unique sequences from 4560 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D147_F_filt.fastq.gz
## Encountered 3590 unique sequences from 15637 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D148_F_filt.fastq.gz
## Encountered 2762 unique sequences from 11413 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D149_F_filt.fastq.gz
## Encountered 3021 unique sequences from 12017 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D150_F_filt.fastq.gz
## Encountered 1566 unique sequences from 5032 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D2_F_filt.fastq.gz
## Encountered 3707 unique sequences from 18075 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D3_F_filt.fastq.gz
## Encountered 1479 unique sequences from 6250 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D5_F_filt.fastq.gz
## Encountered 1195 unique sequences from 4052 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D6_F_filt.fastq.gz
## Encountered 1832 unique sequences from 7369 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D7_F_filt.fastq.gz
## Encountered 1183 unique sequences from 4765 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D8_F_filt.fastq.gz
## Encountered 1382 unique sequences from 4871 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D9_F_filt.fastq.gz
## Encountered 1709 unique sequences from 6504 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/Mock_F_filt.fastq.gz
## Encountered 897 unique sequences from 4314 total sequences read.
```

```
derepRs <- derepFastq(filtRs, verbose=TRUE)
```

```
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D0_R_filt.fastq.gz
## Encountered 1660 unique sequences from 7113 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D1_R_filt.fastq.gz
## Encountered 1349 unique sequences from 5299 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D141_R_filt.fastq.gz
## Encountered 1335 unique sequences from 5463 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D142_R_filt.fastq.gz
## Encountered 853 unique sequences from 2914 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D143_R_filt.fastq.gz
## Encountered 880 unique sequences from 2941 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D144_R_filt.fastq.gz
## Encountered 1286 unique sequences from 4312 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D145_R_filt.fastq.gz
## Encountered 1803 unique sequences from 6741 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D146_R_filt.fastq.gz
## Encountered 1265 unique sequences from 4560 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D147_R_filt.fastq.gz
## Encountered 3414 unique sequences from 15637 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D148_R_filt.fastq.gz
## Encountered 2522 unique sequences from 11413 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D149_R_filt.fastq.gz
## Encountered 2771 unique sequences from 12017 total sequences read.
## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D150_R_filt.fastq.gz
```

```
## Encountered 1415 unique sequences from 5032 total sequences read.

## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D2_R_filt.fastq.gz

## Encountered 3290 unique sequences from 18075 total sequences read.

## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D3_R_filt.fastq.gz

## Encountered 1390 unique sequences from 6250 total sequences read.

## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D5_R_filt.fastq.gz

## Encountered 1134 unique sequences from 4052 total sequences read.

## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D6_R_filt.fastq.gz

## Encountered 1635 unique sequences from 7369 total sequences read.

## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D7_R_filt.fastq.gz

## Encountered 1084 unique sequences from 4765 total sequences read.

## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D8_R_filt.fastq.gz

## Encountered 1161 unique sequences from 4871 total sequences read.

## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/F3D9_R_filt.fastq.gz

## Encountered 1502 unique sequences from 6504 total sequences read.

## Dereplicating sequence entries in Fastq file: ./MiSeq_SOP/filtered/Mock_R_filt.fastq.gz

## Encountered 732 unique sequences from 4314 total sequences read.
```

```
# Name the derep-class objects by the sample names
names(derepFs) <- sampleNames
names(derepRs) <- sampleNames
```

La méthode DADA2 s'appuie sur un modèle paramétré d'erreurs de substitution pour distinguer les erreurs de séquençage de la variation biologique réelle.

L'apprentissage des paramètres est computationnellement intensif, car il nécessite de multiples itérations de l'algorithme d'inférence de séquence, et il est donc souvent utile d'estimer les taux d'erreur à partir d'un sous-ensemble de données (suffisamment important).

```
errF <- learnErrors(filtFs, multithread=TRUE)
```

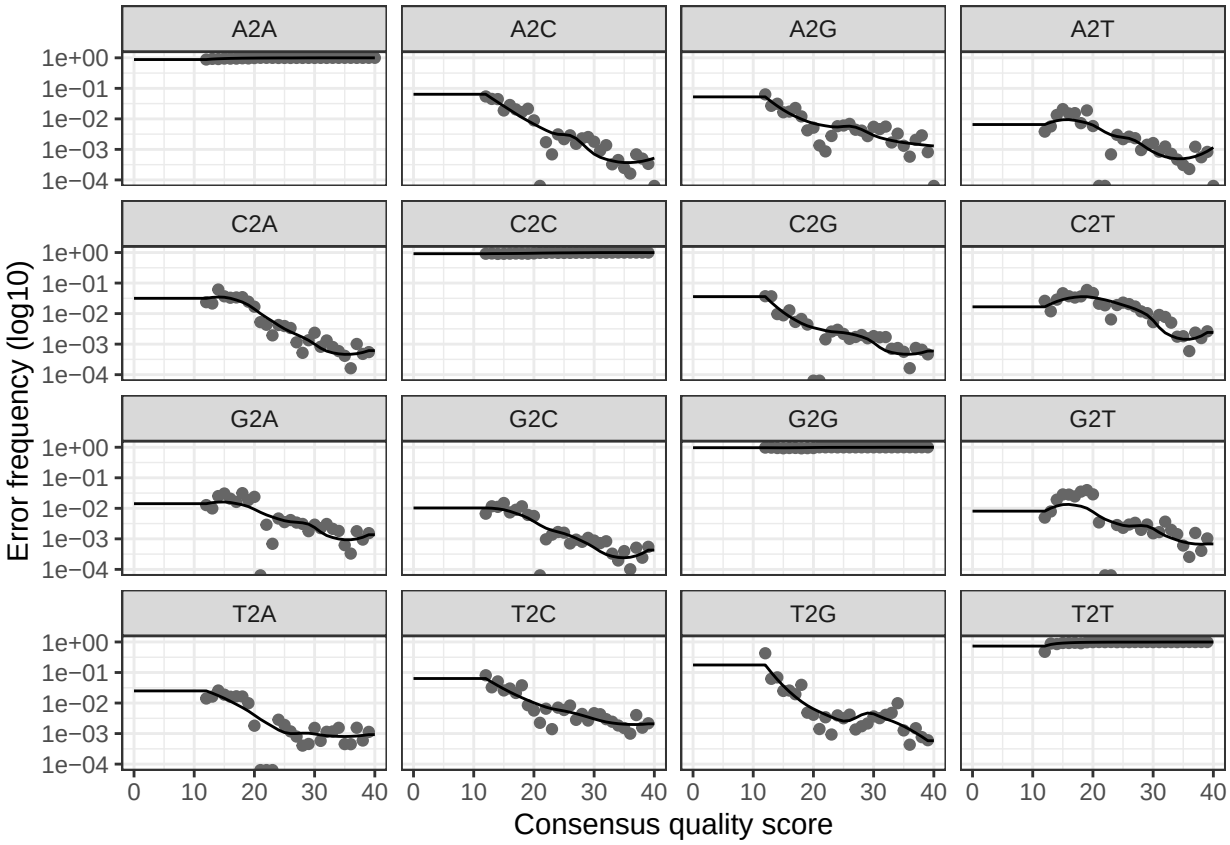
```
## 33514080 total bases in 139642 reads from 20 samples will be used for learning the error rates.
```

```
errR <- learnErrors(filtRs, multithread=TRUE)
```

```
## 22342720 total bases in 139642 reads from 20 samples will be used for learning the error rates.
```

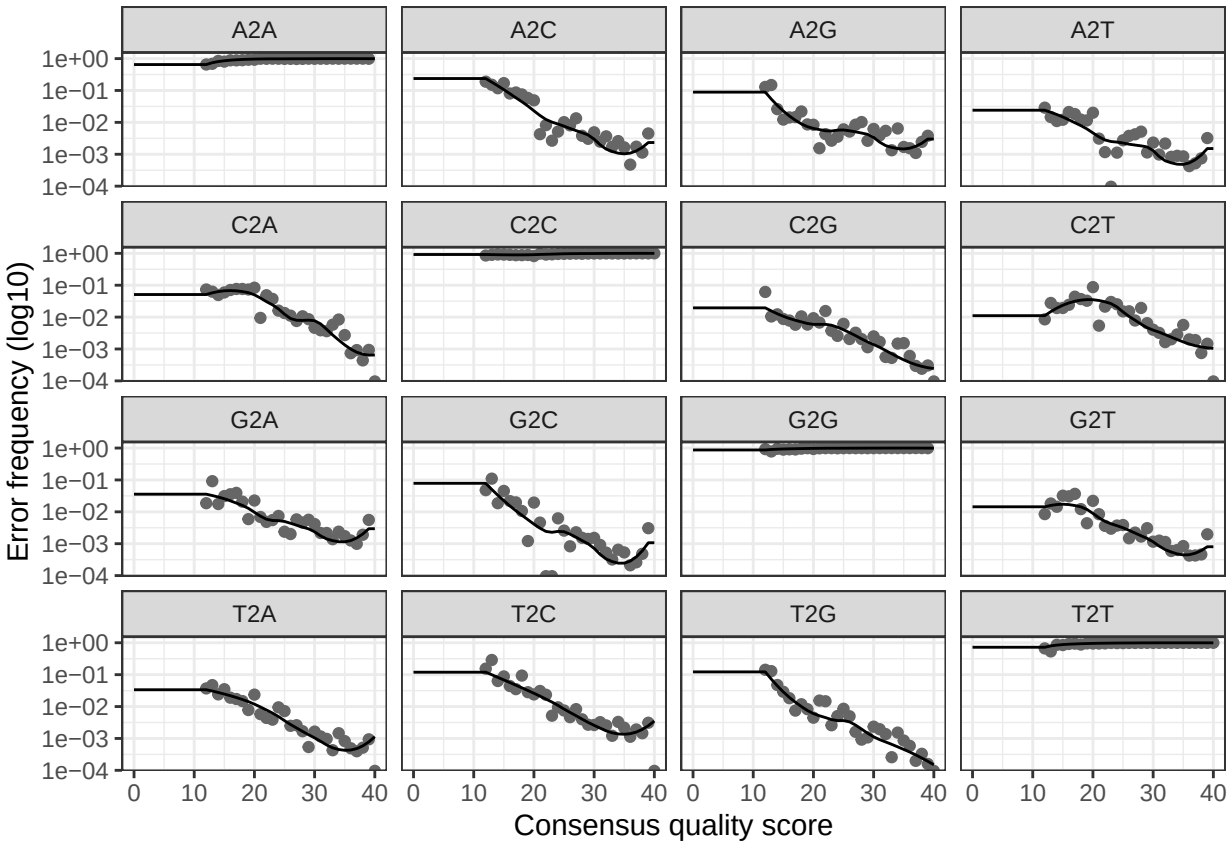
```
plotErrors(errF)
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```




```
plotErrors(errR)
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```



Afin de vérifier que les taux d'erreur ont été raisonnablement bien estimés, nous inspectons l'ajustement entre les taux d'erreur observés (points noirs) et les taux d'erreur ajustés (lignes noires) de la figure 1 . Ces figures montrent les fréquences de chaque type de transition en fonction de la qualité.

Comme cet ensemble de données n'est pas particulièrement important, nous effectuons des inférences groupées (pour les lectures forward puis les lectures reverse). Depuis la version 1.2, multithreading peut maintenant être activé avec les arguments multithread = TRUE, ce qui accélère considérablement cette étape.

```
dadaFs <- dada(derepFs, err=errF, multithread=TRUE)
```

```
## Sample 1 - 7113 reads in 1979 unique sequences.
## Sample 2 - 5299 reads in 1639 unique sequences.
## Sample 3 - 5463 reads in 1477 unique sequences.
## Sample 4 - 2914 reads in 904 unique sequences.
## Sample 5 - 2941 reads in 939 unique sequences.
## Sample 6 - 4312 reads in 1267 unique sequences.
## Sample 7 - 6741 reads in 1756 unique sequences.
## Sample 8 - 4560 reads in 1438 unique sequences.
## Sample 9 - 15637 reads in 3590 unique sequences.
## Sample 10 - 11413 reads in 2762 unique sequences.
## Sample 11 - 12017 reads in 3021 unique sequences.
## Sample 12 - 5032 reads in 1566 unique sequences.
## Sample 13 - 18075 reads in 3707 unique sequences.
## Sample 14 - 6250 reads in 1479 unique sequences.
## Sample 15 - 4052 reads in 1195 unique sequences.
```

```
## Sample 16 - 7369 reads in 1832 unique sequences.
## Sample 17 - 4765 reads in 1183 unique sequences.
## Sample 18 - 4871 reads in 1382 unique sequences.
## Sample 19 - 6504 reads in 1709 unique sequences.
## Sample 20 - 4314 reads in 897 unique sequences.
```

```
dadaRs <- dada(derepRs, err=errR, multithread=TRUE)
```

```
## Sample 1 - 7113 reads in 1660 unique sequences.
## Sample 2 - 5299 reads in 1349 unique sequences.
## Sample 3 - 5463 reads in 1335 unique sequences.
## Sample 4 - 2914 reads in 853 unique sequences.
## Sample 5 - 2941 reads in 880 unique sequences.
## Sample 6 - 4312 reads in 1286 unique sequences.
## Sample 7 - 6741 reads in 1803 unique sequences.
## Sample 8 - 4560 reads in 1265 unique sequences.
## Sample 9 - 15637 reads in 3414 unique sequences.
## Sample 10 - 11413 reads in 2522 unique sequences.
## Sample 11 - 12017 reads in 2771 unique sequences.
## Sample 12 - 5032 reads in 1415 unique sequences.
## Sample 13 - 18075 reads in 3290 unique sequences.
## Sample 14 - 6250 reads in 1390 unique sequences.
## Sample 15 - 4052 reads in 1134 unique sequences.
## Sample 16 - 7369 reads in 1635 unique sequences.
## Sample 17 - 4765 reads in 1084 unique sequences.
## Sample 18 - 4871 reads in 1161 unique sequences.
## Sample 19 - 6504 reads in 1502 unique sequences.
## Sample 20 - 4314 reads in 732 unique sequences.
```

Inspection de l'objet de classe Dada retourné par Dada :

```
dadaFs[[1]]
```

```
## dada-class: object describing DADA2 denoising results
## 128 sequence variants were inferred from 1979 input unique sequences.
## Key parameters: OMEGA_A = 1e-40, OMEGA_C = 1e-40, BAND_SIZE = 16
```

L'algorithme DADA2 a déduit 128 variantes de séquences réelles à partir des séquences uniques de 1979 dans le premier échantillon. L'objet dada-class contient plusieurs diagnostics sur la qualité de chaque variante de séquence inférée.

L'étape d'inférence de la séquence DADA2 a éliminé (presque) toutes les erreurs de substitution et d'indel des données.

Nous fusionnons maintenant les séquences forward et reverse inférées, supprimant les séquences appariées qui ne se chevauchent pas parfaitement comme un contrôle final contre les erreurs résiduelles.

Construction d'une table de séquences et élimination des chimères

La méthode DADA2 produit un tableau séquentiel qui est un analogue à plus haute résolution de la «table OTU» commune, c.-à-d. un tableau de caractéristiques des échantillons par séquence évaluée par le nombre de fois que chaque séquence a été observée dans chaque échantillon.

```
mergers <- mergePairs(dadaFs, derepFs, dadaRs, derepRs)
```

```
seqtabAll <- makeSequenceTable(mergers[!grepl("Mock", names(mergers))])
table(nchar(getSequences(seqtabAll)))
```

```
##
## 251 252 253 254 255
##   1  85 186   5   2
```

Nous supprimons maintenant les séquences chimériques en comparant chaque séquence inférée aux autres dans le tableau, et en supprimant celles qui peuvent être reproduites en assemblant deux séquences plus abondantes.

```
seqtabNoC <- removeBimeraDenovo(seqtabAll)
```

Bien que les nombres exacts varient considérablement selon les conditions expérimentales, il est typique que les chimères comprennent une fraction substantielle des variantes de séquence inférées, mais seulement une petite fraction de toutes les lectures. Les chimères représentent environ 22% des variants de séquence déduits, mais ces variants ne représentent qu'environ 4% de la séquence totale lue.

Assignation d'une taxonomie

L'un des avantages d'utiliser des locus marqueurs bien classés comme le gène de l'ARNr 16S est la capacité de classer taxonomiquement les variantes de séquence. Le paquet dada2 implémente la méthode naïve du classificateur bayésien à cette fin. Ce classificateur compare les variantes de séquence à un ensemble de séquences d'entraînement classifiées, et ici nous utilisons l'ensemble d'entraînement RDP v16.

On importe maintenant des nouvelles données pour réaliser l'assignation d'une taxonomie à différentes familles bactériennes :

```
wget https://zenodo.org/record/801828/files/rdp_train_set_16.fa.gz
```

```
## --2020-12-04 20:02:07--  https://zenodo.org/record/801828/files/rdp_train_set_16.fa.gz
## Resolving zenodo.org (zenodo.org)... 137.138.76.77
## Connecting to zenodo.org (zenodo.org)|137.138.76.77|:443... connected.
## HTTP request sent, awaiting response... 200 OK
## Length: 3828450 (3.7M) [application/octet-stream]
## Saving to: 'rdp_train_set_16.fa.gz.19'
##
##      0K ..........  ..........  ..........  ..........  ..........   1% 7.81M 0s
##     50K ..........  ..........  ..........  ..........  ..........   2% 15.0M 0s
##    100K ..........  ..........  ..........  ..........  ..........   4% 8.17M 0s
##    150K ..........  ..........  ..........  ..........  ..........   5% 15.4M 0s
##    200K ..........  ..........  ..........  ..........  ..........   6% 70.2M 0s
##    250K ..........  ..........  ..........  ..........  ..........   8% 15.1M 0s
##    300K ..........  ..........  ..........  ..........  ..........   9% 15.1M 0s
##    350K ..........  ..........  ..........  ..........  .......... 10%  103M 0s
##    400K ..........  ..........  ..........  ..........  .......... 12% 15.0M 0s
##    450K ..........  ..........  ..........  ..........  .......... 13% 83.5M 0s
##    500K ..........  ..........  ..........  ..........  .......... 14% 18.1M 0s
##    550K ..........  ..........  ..........  ..........  .......... 16%  102M 0s
```

```
##    600K ..........  .......... .......... .......... .......... 17% 20.2M 0s
##    650K ..........  .......... .......... .......... .......... 18% 18.7M 0s
##    700K ..........  .......... .......... .......... .......... 20% 52.5M 0s
##    750K ..........  .......... .......... .......... .......... 21% 21.2M 0s
##    800K ..........  .......... .......... .......... .......... 22% 47.6M 0s
##    850K ..........  .......... .......... .......... .......... 24% 23.0M 0s
##    900K ..........  .......... .......... .......... .......... 25% 43.1M 0s
##    950K ..........  .......... .......... .......... .......... 26% 16.1M 0s
##   1000K ..........  .......... .......... .......... .......... 28%  107M 0s
##   1050K ..........  .......... .......... .......... .......... 29% 18.0M 0s
##   1100K ..........  .......... .......... .......... .......... 30%  100M 0s
##   1150K ..........  .......... .......... .......... .......... 32% 9.78M 0s
##   1200K ..........  .......... .......... .......... .......... 33%  104M 0s
##   1250K ..........  .......... .......... .......... .......... 34% 17.6M 0s
##   1300K ..........  .......... .......... .......... .......... 36%  131M 0s
##   1350K ..........  .......... .......... .......... .......... 37% 15.3M 0s
##   1400K ..........  .......... .......... .......... .......... 38%  126M 0s
##   1450K ..........  .......... .......... .......... .......... 40% 13.8M 0s
##   1500K ..........  .......... .......... .......... .......... 41%  139M 0s
##   1550K ..........  .......... .......... .......... .......... 42% 16.2M 0s
##   1600K ..........  .......... .......... .......... .......... 44%  115M 0s
##   1650K ..........  .......... .......... .......... .......... 45% 16.1M 0s
##   1700K ..........  .......... .......... .......... .......... 46%  131M 0s
##   1750K ..........  .......... .......... .......... .......... 48% 14.9M 0s
##   1800K ..........  .......... .......... .......... .......... 49% 84.7M 0s
##   1850K ..........  .......... .......... .......... .......... 50%  150M 0s
##   1900K ..........  .......... .......... .......... .......... 52% 16.6M 0s
##   1950K ..........  .......... .......... .......... .......... 53%  125M 0s
##   2000K ..........  .......... .......... .......... .......... 54% 22.1M 0s
##   2050K ..........  .......... .......... .......... .......... 56% 44.8M 0s
##   2100K ..........  .......... .......... .......... .......... 57%  104M 0s
##   2150K ..........  .......... .......... .......... .......... 58% 14.6M 0s
##   2200K ..........  .......... .......... .......... .......... 60%  135M 0s
##   2250K ..........  .......... .......... .......... .......... 61% 16.3M 0s
##   2300K ..........  .......... .......... .......... .......... 62% 86.6M 0s
##   2350K ..........  .......... .......... .......... .......... 64%  100M 0s
##   2400K ..........  .......... .......... .......... .......... 65% 16.2M 0s
##   2450K ..........  .......... .......... .......... .......... 66% 95.7M 0s
##   2500K ..........  .......... .......... .......... .......... 68% 94.4M 0s
##   2550K ..........  .......... .......... .......... .......... 69% 16.9M 0s
##   2600K ..........  .......... .......... .......... .......... 70% 92.6M 0s
##   2650K ..........  .......... .......... .......... .......... 72% 16.2M 0s
##   2700K ..........  .......... .......... .......... .......... 73% 82.0M 0s
##   2750K ..........  .......... .......... .......... .......... 74% 26.2M 0s
##   2800K ..........  .......... .......... .......... .......... 76% 21.5M 0s
##   2850K ..........  .......... .......... .......... .......... 77% 59.0M 0s
##   2900K ..........  .......... .......... .......... .......... 78% 82.8M 0s
##   2950K ..........  .......... .......... .......... .......... 80% 19.3M 0s
##   3000K ..........  .......... .......... .......... .......... 81% 66.7M 0s
##   3050K ..........  .......... .......... .......... .......... 82% 24.0M 0s
##   3100K ..........  .......... .......... .......... .......... 84% 45.1M 0s
##   3150K ..........  .......... .......... .......... .......... 85% 80.9M 0s
##   3200K ..........  .......... .......... .......... .......... 86% 24.9M 0s
##   3250K ..........  .......... .......... .......... .......... 88% 39.4M 0s
```

```
##   3300K ..........  .......... .......... .......... .......... 89% 22.7M 0s
##   3350K ..........  .......... .......... .......... .......... 90% 45.1M 0s
##   3400K ..........  .......... .......... .......... .......... 92% 20.4M 0s
##   3450K ..........  .......... .......... .......... .......... 93% 62.6M 0s
##   3500K ..........  .......... .......... .......... .......... 94% 46.6M 0s
##   3550K ..........  .......... .......... .......... .......... 96% 19.0M 0s
##   3600K ..........  .......... .......... .......... .......... 97% 51.8M 0s
##   3650K ..........  .......... .......... .......... .......... 98% 22.7M 0s
##   3700K ..........  .......... .......... ........              100% 62.3M=0.1s
##
## 2020-12-04 20:02:08 (27.0 MB/s) - 'rdp_train_set_16.fa.gz.19' saved [3828450/3828450]
```

Dada2 contient des fastas de formation formatés pour l'ensemble de formation RDP, GreenGenes regroupés à 97% d'identité, et la base de données de référence Silva disponible. Pour la taxonomie fongique, les fichiers de version General Fasta de la base de données UNITE ITS peuvent être utilisés tels quels. Pour suivre ce flux de travail, téléchargez le fichier rdp_train_set_16.fa.gz et placez-le dans le répertoire contenant les fichiers fastq.

```
fastaRef <- "./rdp_train_set_16.fa.gz"
taxTab <- assignTaxonomy(seqtabNoC, refFasta = fastaRef, multithread=TRUE)
unname(head(taxTab))
```

```
##      [,1]       [,2]            [,3]          [,4]
## [1,] "Bacteria" "Bacteroidetes" "Bacteroidia" "Bacteroidales"
## [2,] "Bacteria" "Bacteroidetes" "Bacteroidia" "Bacteroidales"
## [3,] "Bacteria" "Bacteroidetes" "Bacteroidia" "Bacteroidales"
## [4,] "Bacteria" "Bacteroidetes" "Bacteroidia" "Bacteroidales"
## [5,] "Bacteria" "Bacteroidetes" "Bacteroidia" "Bacteroidales"
## [6,] "Bacteria" "Bacteroidetes" "Bacteroidia" "Bacteroidales"
##      [,5]                 [,6]
## [1,] "Porphyromonadaceae" NA
## [2,] "Porphyromonadaceae" NA
## [3,] "Porphyromonadaceae" NA
## [4,] "Porphyromonadaceae" "Barnesiella"
## [5,] "Bacteroidaceae"     "Bacteroides"
## [6,] "Porphyromonadaceae" "Barnesiella"
```

Construction d'un arbre phylogénétique

La relation phylogénétique est couramment utilisée pour éclairer les analyses en aval, en particulier le calcul des distances phylogénétiques entre les communautés microbiennes. La méthode d'inférence de séquence DADA2 est sans référence, donc nous devons construire l'arbre phylogénétique reliant les variantes de novo de séquence inférées. Nous commençons par effectuer un alignement multiple en utilisant le package DECIPHER :

```
library(DECIPHER)
```

```
## Loading required package: Biostrings
```

```
## Loading required package: BiocGenerics
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, basename, cbind, colnames,
##     dirname, do.call, duplicated, eval, evalq, Filter, Find, get, grep,
##     grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##     rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,
##     union, unique, unsplit, which.max, which.min

## Loading required package: S4Vectors

## Loading required package: stats4

##
## Attaching package: 'S4Vectors'

## The following object is masked from 'package:base':
##
##     expand.grid

## Loading required package: IRanges

##
## Attaching package: 'IRanges'

## The following object is masked from 'package:phyloseq':
##
##     distance

## Loading required package: XVector

##
## Attaching package: 'Biostrings'

## The following object is masked from 'package:base':
##
##     strsplit

## Loading required package: RSQLite
```

```
seqs <- getSequences(seqtabNoC)
names(seqs) <- seqs # This propagates to the tip labels of the tree
alignment <- AlignSeqs(DNAStringSet(seqs), anchor=NA,verbose=FALSE)
```

Le package phangorn R est ensuite utilisé pour construire un arbre phylogénétique. Ici, nous construisons d'abord un arbre de neighbor-joining, puis nous ajustons un arbre de probabilité maximum GTR+G+I (Generalized time-reversible with Gamma rate variation) en utilisant l'arbre neighbor-joining comme point de départ.

```
library(phangorn)
```

```
## Loading required package: ape
```

```
##
## Attaching package: 'ape'
```

```
## The following object is masked from 'package:Biostrings':
##
##     complement
```

```
phangAlign <- phyDat(as(alignment, "matrix"), type="DNA")
dm <- dist.ml(phangAlign)
treeNJ <- NJ(dm) # Note, tip order != sequence order
fit = pml(treeNJ, data=phangAlign)
```

```
## negative edges length changed to 0!
```

```
fitGTR <- update(fit, k=4, inv=0.2)
fitGTR <- optim.pml(fitGTR, model="GTR", optInv=TRUE, optGamma=TRUE,
        rearrangement = "stochastic", control = pml.control(trace = 0))
detach("package:phangorn", unload=TRUE)
```

Combiner les données dans un objet Phyloseq

Le paquet phyloseq organise et synthétise les différents types de données d'une expérience typique de séquençage d'amplicon en un seul objet de données qui peut être facilement manipulé. Le dernier bit d'information nécessaire est l'échantillon de données contenues dans un fichier . csv. Il peut être téléchargé à partir de github :

```
samdf <- read.csv("https://raw.githubusercontent.com/spholmes/F1000_workflow/master/data/MIMARKS_Data_c
samdf$SampleID <- paste0(gsub("00", "", samdf$host_subject_id), "D", samdf$age-21)
samdf <- samdf[!duplicated(samdf$SampleID),] # Remove dupicate entries for reverse reads
rownames(seqtabAll) <- gsub("124", "125", rownames(seqtabAll)) # Fix discrepancy
all(rownames(seqtabAll) %in% samdf$SampleID) # TRUE
```

```
## [1] TRUE
```

```
rownames(samdf) <- samdf$SampleID
keep.cols <- c("collection_date", "biome", "target_gene", "target_subfragment",
"host_common_name", "host_subject_id", "age", "sex", "body_product", "tot_mass",
"diet", "family_relationship", "genotype", "SampleID")
samdf <- samdf[rownames(seqtabAll), keep.cols]
```

La série complète de données pour cette étude – le tableau des caractéristiques des échantillons par séquence, les métadonnées des échantillons, les taxonomies des séquences et l'arbre phylogénétique – peuvent maintenant être combinés en un seul objet

```
ps <- phyloseq(otu_table(seqtabNoC, taxa_are_rows=FALSE),
               sample_data(samdf),
               tax_table(taxTab),phy_tree(fitGTR$tree))
ps <- prune_samples(sample_names(ps) != "Mock", ps) # Remove mock sample
ps
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 218 taxa and 19 samples ]
## sample_data() Sample Data:       [ 19 samples by 14 sample variables ]
## tax_table()   Taxonomy Table:    [ 218 taxa by 6 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 218 tips and 216 internal nodes ]
```

Chargement des données

```
ps_connect <-url("https://raw.githubusercontent.com/spholmes/F1000_workflow/master/data/ps.rds")
ps = readRDS(ps_connect)
ps
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 389 taxa and 360 samples ]
## sample_data() Sample Data:       [ 360 samples by 14 sample variables ]
## tax_table()   Taxonomy Table:    [ 389 taxa by 6 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 389 tips and 387 internal nodes ]
```

Filtration taxonomique

Dans de nombreux milieux biologiques, l'ensemble de tous les organismes de tous les échantillons est bien représenté dans la base de données de référence taxonomique disponible. Lorsque (et seulement lorsque) c'est le cas, il est raisonnable ou même conseillé de filtrer les caractéristiques taxonomiques pour lesquelles une taxonomie de haut rang ne pourrait pas être attribuée. De telles caractéristiques ambiguës dans ce cadre sont presque toujours des artefacts séquentiels qui n'existent pas dans la nature. Il devrait être évident qu'un tel filtre n'est pas approprié pour les échantillons de spécimens mal caractérisés ou nouveaux, du moins jusqu'à ce qu'il y ai la possibilité d'une nouveauté taxonomique qui puisse être rejetée de façon satisfaisante.

Pour commencer, nous créons un tableau des comptages de lectures pour chaque phylas présents dans l'ensemble de données.

```
# Show available ranks in the dataset
rank_names(ps)
```

```
## [1] "Kingdom" "Phylum"  "Class"   "Order"   "Family"  "Genus"
```

```
# Create table, number of features for each phyla
table(tax_table(ps)[, "Phylum"], exclude = NULL)
```

```
##
##              Actinobacteria               Bacteroidetes
##                          13                          23
## Candidatus_Saccharibacteria   Cyanobacteria/Chloroplast
##                           1                           4
##         Deinococcus-Thermus                  Firmicutes
##                           1                         327
##                Fusobacteria              Proteobacteria
##                           1                          11
##                 Tenericutes             Verrucomicrobia
##                           1                           1
##                        <NA>
##                           6
```

Cela montre quelques phylas pour lesquels une seule caractéristique a été observée. Il est à noter que dans ce cas, six fonctionnalités ont été annotées avec un Phyla de NA. Ces caractéristiques sont probablement des artefacts dans un ensemble de données comme celui-ci, et devraient être supprimés.

Ce qui suit garantit que les fonctions avec une annotation de phyla ambiguë soient également supprimées:

```
ps <- subset_taxa(ps, !is.na(Phylum) & !Phylum %in% c("", "uncharacterized"))
```

Une prochaine étape utile consiste à explorer la prévalence des caractéristiques dans l'ensemble de données, que nous définirons ici comme le nombre d'échantillons dans lesquels un taxon apparaît au moins une fois :

```
# Compute prevalence of each feature, store as data.frame
prevdf = apply(X = otu_table(ps),
               MARGIN = ifelse(taxa_are_rows(ps), yes = 1, no = 2),
               FUN = function(x){sum(x > 0)})
# Add taxonomy and total read counts to this data.frame
prevdf = data.frame(Prevalence = prevdf,
                    TotalAbundance = taxa_sums(ps),
                    tax_table(ps))
```

On peut se demander s'il y a des phylums composés principalement de caractéristiques à faible prévalence. Pour y répondre, on peut calculer la prévalence totale et moyenne des caractéristiques dans chaque phyla :

```
plyr::ddply(prevdf, "Phylum", function(df1){cbind(mean(df1$Prevalence),sum(df1$Prevalence))})
```

```
##                       Phylum         1     2
## 1              Actinobacteria 120.15385  1562
## 2               Bacteroidetes 265.52174  6107
## 3  Candidatus_Saccharibacteria 280.00000   280
## 4    Cyanobacteria/Chloroplast  64.25000   257
## 5          Deinococcus-Thermus  52.00000    52
## 6                  Firmicutes 179.24771 58614
## 7                Fusobacteria   2.00000     2
```

```
## 8               Proteobacteria  59.09091   650
## 9                  Tenericutes 234.00000   234
## 10             Verrucomicrobia 104.00000   104
```

Ce sont les prévalences totales et moyennes des caractéristiques dans chaque phylum.

Deinococcus-Thermus est apparu dans un peu plus d'1 % des échantillons, et les Fusobactéries sont apparues dans seulement 2 échantillons au total. Dans certains cas, il pourrait être intéressant d'explorer ces deux phyla plus en détail malgré cela (mais probablement pas les deux échantillons de Fusobacteria).

Pour les besoins de cet exemple, cependant, ils seront filtrés à partir de l'ensemble de données.

```
# Define phyla to filter
filterPhyla = c("Fusobacteria", "Deinococcus-Thermus")
# Filter entries with unidentified Phylum.
ps1 = subset_taxa(ps, !Phylum %in% filterPhyla)
ps1
```

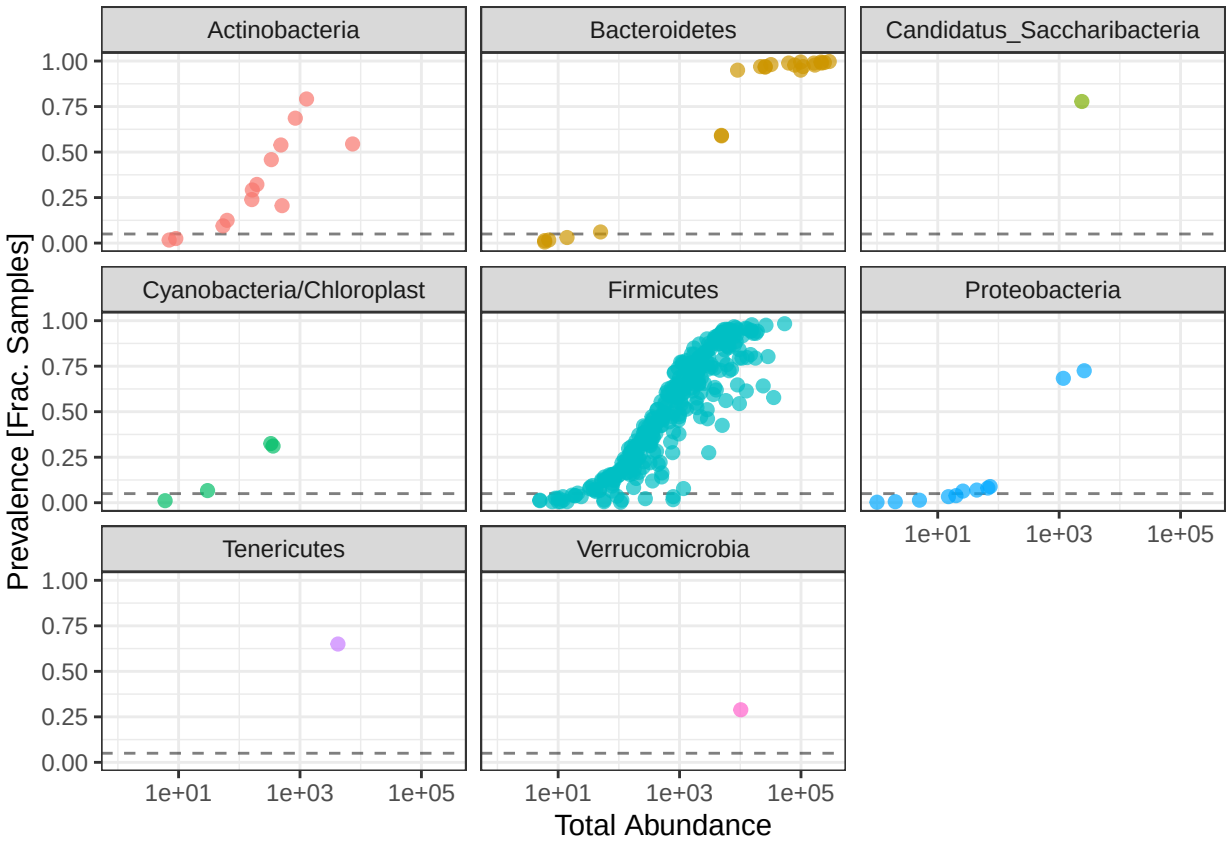
```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 381 taxa and 360 samples ]
## sample_data() Sample Data:       [ 360 samples by 14 sample variables ]
## tax_table()   Taxonomy Table:    [ 381 taxa by 6 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 381 tips and 379 internal nodes ]
```

Filtration de la prévalence

Les étapes de filtrage précédentes sont considérées comme supervisées, car elles reposent sur des informations préalables externes à cette expérience (une base de données de référence taxonomique). Cette prochaine étape de filtrage est complètement non supervisée, en s'appuyant uniquement sur les données de cette expérience, et un paramètre que nous choisirons après avoir exploré les données. Ainsi, cette étape de filtrage peut être appliquée même dans des paramètres où l'annotation taxonomique n'est pas disponible ou n'est pas fiable.

Tout d'abord, nous allons étudier la relation entre la prévalence et le nombre total de lectures pour chaque caractéristique. Parfois, cela révèle des valeurs aberrantes qui devraient être probablement supprimées, et cela fournit également un aperçu des intervalles de l'une ou l'autre des caractéristiques qui pourraient être utiles.

```
# Subset to the remaining phyla
prevdf1 = subset(prevdf, Phylum %in% get_taxa_unique(ps1, "Phylum"))
ggplot(prevdf1, aes(TotalAbundance, Prevalence / nsamples(ps),color=Phylum)) +
  # Include a guess for parameter
  geom_hline(yintercept = 0.05, alpha = 0.5, linetype = 2) +  geom_point(size = 2, alpha = 0.7) +
  scale_x_log10() +  xlab("Total Abundance") + ylab("Prevalence [Frac. Samples]") +
  facet_wrap(~Phylum) + theme(legend.position="none")
```



Ces graphiques présent la prévalence des taxons par rapport aux dénombrements totaux. Chaque point représente un taxon différent. L'exploration des données de cette façon est souvent utile pour sélectionner les paramètres de filtrage, comme les critères de prévalence minimum que nous utiliserons pour filtrer les données ci-dessus. Parfois, une séparation naturelle dans l'ensemble de données se révèle, ou du moins, un choix conservateur qui se trouve dans une région stable pour laquelle de petits changements auraient eu un effet mineur ou nul sur l'interpréation biologique (stabilité). Ici, aucune séparation naturelle n'est immédiatement évidente, mais il semble que nous pourrions raisonnablement définir un seuil de prévalence dans une fourchette de 0 à 10 % environ. Il est important que ce choix n'introduise pas de biais dans une analyse en aval de l'association de l'abondance différentielle.

Ce qui suit utilise 5 % de tous les échantillons comme seuil de prévalence :

```
# Define prevalence threshold as 5% of total samples
prevalenceThreshold = 0.05 * nsamples(ps)
prevalenceThreshold
```

```
## [1] 18
```

```
## [1] 18
# Execute prevalence filter, using `prune_taxa()` function
keepTaxa = rownames(prevdf1)[(prevdf1$Prevalence >= prevalenceThreshold)]
ps2 = prune_taxa(keepTaxa, ps)
```

Agglomération des taxons

Lorsqu'on sait qu'il y a beaucoup d'espèces ou de sous-espèces redondantes au niveau fonctionnel dans une communauté microbienne, il pourrait être utile d'agglomérer les caractéristiques des données correspondant à des taxons étroitement apparentés. Idéalement, nous connaîtrions les redondances fonctionnelles parfaitement à l'avance, auquel cas nous regrouperions les taxons en utilisant ces relations définies et la fonction dans phyloseq. Ce genre de données fonctionnelles n'est généralement pas disponible, et différentes paires de microbes auront différents ensembles de fonctions qui se chevauchent, ce qui complique la question de définir des critères de regroupement appropriés.

Bien qu'il ne s'agisse pas nécessairement des critères les plus utiles ou les plus précis sur le plan fonctionnel pour regrouper les caractéristiques microbiennes (parfois loin d'être précis), l'agglomération taxonomique a l'avantage d'être beaucoup plus facile à définir à l'avance. En effet, les taxonomies sont habituellement définies à l'aide d'une structure graphique arborescente relativement simple qui comporte un nombre fixe de nœuds internes, appelés « rangs ». Cette structure est assez simple pour que le paquet phyloseq représente les taxonomies comme table d'étiquettes taxonomiques. L'agglomération taxonomique regroupe toutes les « feuilles » de la hiérarchie qui descendent du rang d'agglomération prescrit par l'utilisateur, ce qui est parfois appelé « glomming ».

L'exemple de code suivant montre comment combiner toutes les caractéristiques qui descendent du même genre.

```
# How many genera would be present after filtering?
length(get_taxa_unique(ps2, taxonomic.rank = "Genus"))
```

```
## [1] 49
```

```
ps3 = tax_glom(ps2, "Genus", NArm = TRUE)
```

Si la taxonomie n'est pas disponible ou n'est pas fiable, l'agglomération arborescente est une solution de rechange « sans taxonomie » pour combiner des caractéristiques de données correspondant à des taxons étroitement apparentés. Dans ce cas, plutôt que le rang taxonomique, l'utilisateur spécifie une hauteur d'arbre correspondant à la distance phylogénétique entre les caractéristiques qui devrait définir leur groupement. Ceci est très similaire à ce qu'on appelle « l'OTU Clustering », sauf que dans de nombreux algorithmes de OTU Clustering, la distance de séquence utilisée n'a pas la même (ou aucune) définition évolutive.

```
h1 = 0.4
ps4 = tip_glom(ps2, h = h1)
```

Ici, la fonction plot_tree() de phyloseq compare les données originales non filtrées, l'arbre après agglomération taxononmique et l'arbre après agglomération phylogénétique. Ils sont stockés sous forme d'objets graphiques séparés, puis regroupés dans un graphique en utilisant la fonction gridExtra::grid.arrange.

```
multiPlotTitleTextSize = 15
p2tree = plot_tree(ps2, method = "treeonly",
                   ladderize = "left",
                   title = "Before Agglomeration") +
  theme(plot.title = element_text(size = multiPlotTitleTextSize))
p3tree = plot_tree(ps3, method = "treeonly",
                   ladderize = "left", title = "By Genus") +
  theme(plot.title = element_text(size = multiPlotTitleTextSize))
p4tree = plot_tree(ps4, method = "treeonly",
                   ladderize = "left", title = "By Height") +
  theme(plot.title = element_text(size = multiPlotTitleTextSize))
```

Ces figures montrent les différents types d'agglomération. On peut voir l'arbre original à gauche, l'agglomération taxonomique au rang du genre au milieu et l'agglomération phylogénétique à une distance fixe de 0,4 à droite.

Transformation de la valeur de l'abondance

Il est généralement nécessaire de transformer les données de comptage du microbiome pour tenir compte des différences dans la taille de la bibliothèque, variance, scale, etc. Le paquet phyloseq fournit une interface flexible pour définir de nouvelles fonctions pour accomplir ces transformations des valeurs d'abondance avec la fonction transform_sample_counts(). Le premier argument de cette fonction est l'objet phyloseq que vous voulez transformer, et le second argument est une fonction R qui définit la transformation. La fonction R est appliquée par échantillonnage, en s'attendant à ce que le premier argument sans nom soit un vecteur de comptages de taxons dans le même ordre que l'objet phyloseq. Des arguments supplémentaires sont transmis à la fonction spécifiée dans le second argument, fournissant un moyen explicite d'inclure des valeurs pré-compilées, des paramètres/seuils définis précédemment, ou tout autre objet qui pourrait être approprié pour calculer les valeurs transformées d'intérêt.

Cet exemple commence par définir une fonction plot personnalisée, plot_abundance(), qui utilise la fonction de phyloseq pour définir un graphique d'abondance relative. Nous l'utiliserons pour comparer plus facilement les différences d'échelle et de distribution des valeurs d'abondance dans notre objet phyloseq avant et après la transformation.

```
plot_abundance = function(physeq,title = "",
                          Facet = "Order", Color = "Phylum"){
  # Arbitrary subset, based on Phylum, for plotting
  p1f = subset_taxa(physeq, Phylum %in% c("Firmicutes"))
  mphyseq = psmelt(p1f)
  mphyseq <- subset(mphyseq, Abundance > 0)
  ggplot(data = mphyseq, mapping = aes_string(x = "sex",y = "Abundance",
                                color = Color, fill = Color)) +
    geom_violin(fill = NA) +
    geom_point(size = 1, alpha = 0.3,
               position = position_jitter(width = 0.3)) +
    facet_wrap(facets = Facet) + scale_y_log10()+
    theme(legend.position="none")
}
```

La transformation dans ce cas convertit les comptages de chaque échantillon en leurs fréquences, souvent appelées proportions ou abondance relative. Cette fonction peut être définie dans l'appel de la fonction transform_sample_counts().

```
# Transform to relative abundance. Save as new object.
ps3ra = transform_sample_counts(ps3, function(x){x / sum(x)})
```

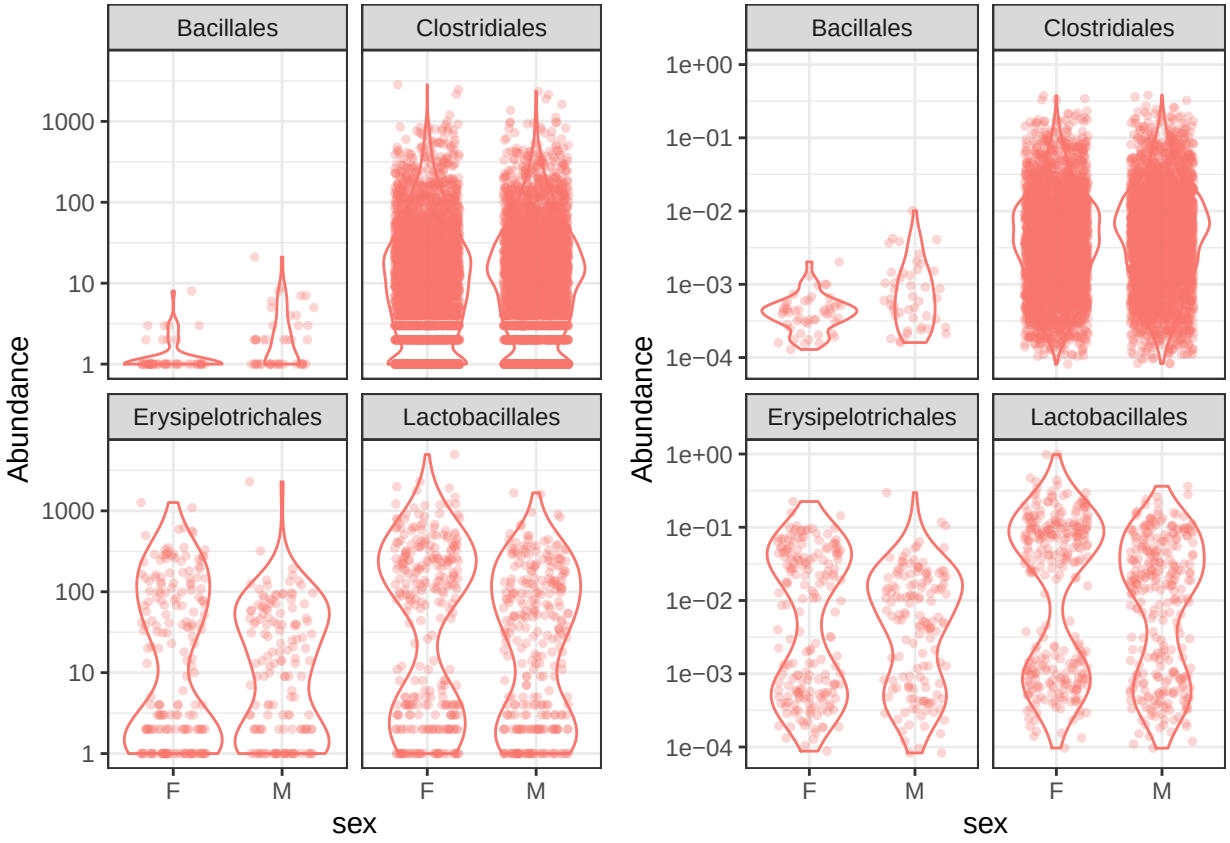
Maintenant nous traçons les valeurs d'abondance avant et après la transformation :

```
library(ggplot2)
library(phyloseq)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:BiocGenerics':
##
##     combine
```

```
plotBefore = plot_abundance(ps3,"")
plotAfter = plot_abundance(ps3ra,"")
# Combine each plot into one graphic.
grid.arrange(nrow = 1,  plotBefore, plotAfter)
```

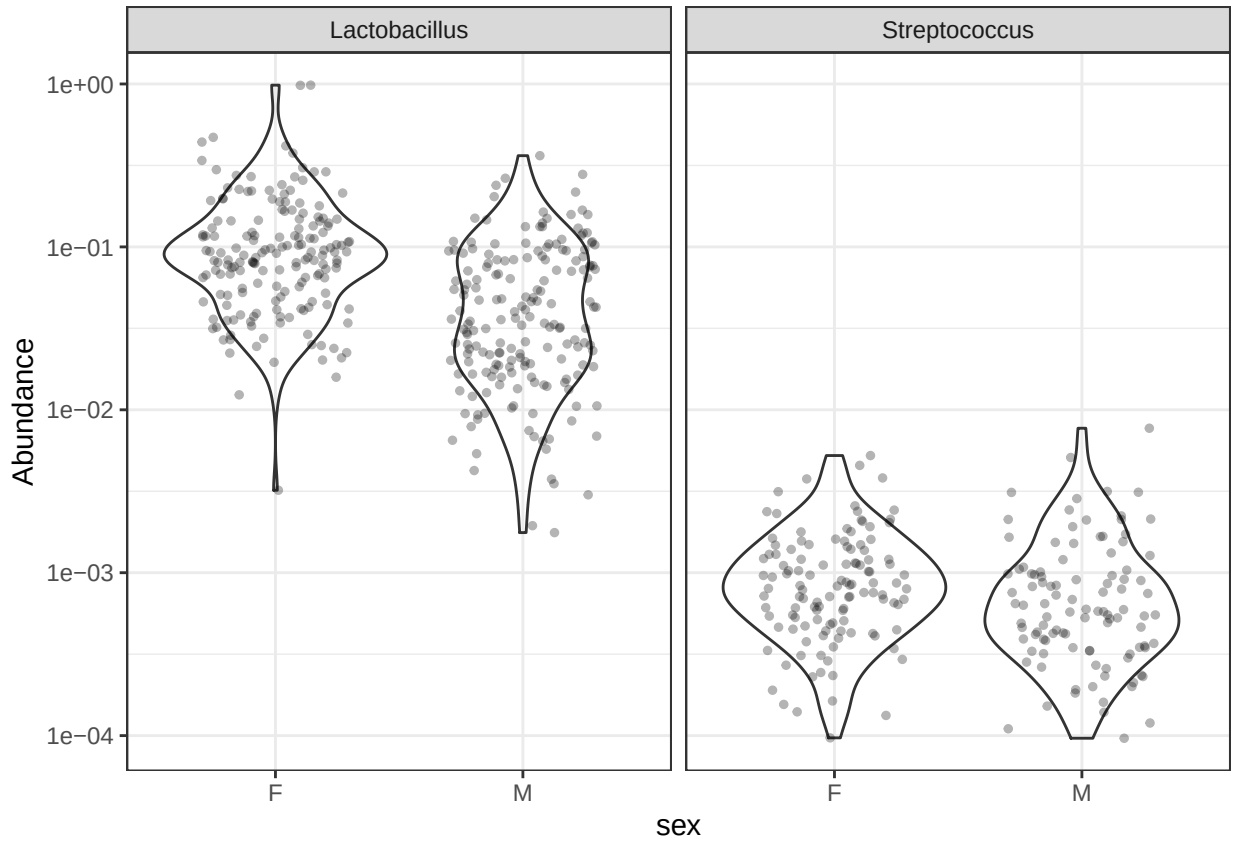


Ces graphiques représentent la comparaison des abondances d'origine (en haut) avec les données transformées (en bas).

Sous-ensemble par taxonomie

Il est à noter que sur la figure précédente, les Lactobacillales semblent être un ordre taxonomique avec un profil d'abondance bimodal dans les données. Nous pouvons le vérifier en traçant juste ce sous-ensemble taxonomique des données. Pour cela, nous sous-utilisons la fonction, puis spécifierons un rang taxonomique plus précis à l'argument de la fonction que nous avons défini ci-dessus.

```
psOrd = subset_taxa(ps3ra, Order == "Lactobacillales")
plot_abundance(psOrd, Facet = "Genus", Color = NULL)
```



Ce graphique montre les abondances relatives de Lactobacillales, regroupées par sexe et genre d'hôte. Ici, il est clair que la distribution biomodale apparente de Lactobacillales sur la figure précédente était le résultat d'un mélange de deux genres différents, avec l'abondance relative typique de Lactobacillales beaucoup plus grande que Streptococcus.

À ce stade, après avoir converti les lectures brutes en abondances d'espèces interprétables, et après avoir filtré et transformé ces abondances pour se concentrer sur des quantités scientifiquement significatives, nous sommes en mesure d'envisager une analyse statistique plus minutieuse. L'avantage d'effectuer ce flux de travail complet dans R est que de permettre une transition de la bioinformatique à la statistique facilement.

Nous appuyons ces affirmations en illustrant plusieurs analyses sur les données de souris (cf data import). Nous expérimentons plusieurs types d'ordination avant de passer à des tests et à une modélisation plus formels, expliquant les paramètres dans lesquels les différents points de vue sont les plus appropriés. Enfin, nous fournissons des exemples d'analyses de données multitables, en utilisant une étude dans laquelle des mesures d'abondance métabolique et microbienne ont été recueillies sur les mêmes échantillons, pour démontrer que le flux de travail général présenté ici peut être adapté au paramètre multitable.

```
.cran_packages <- c( "shiny","miniUI", "caret", "pls", "e1071", "ggplot2", "randomForest", "dplyr", "ggf
                     "reshape2", "PMA", "structSSI", "ade4",
                     "ggnetwork", "intergraph", "scales")
.github_packages <- c("jfukuyama/phyloseqGraphTest")
.bioc_packages <- c("genefilter", "impute")
# Install CRAN packages (if not already installed)
.inst <- .cran_packages %in% installed.packages()
if (any(!.inst)){
  install.packages(.cran_packages[!.inst],repos = "http://cran.rstudio.com/")
}
```

```
.inst <- .github_packages %in% installed.packages()
if (any(!.inst)){
  devtools::install_github(.github_packages[!.inst])
}
```

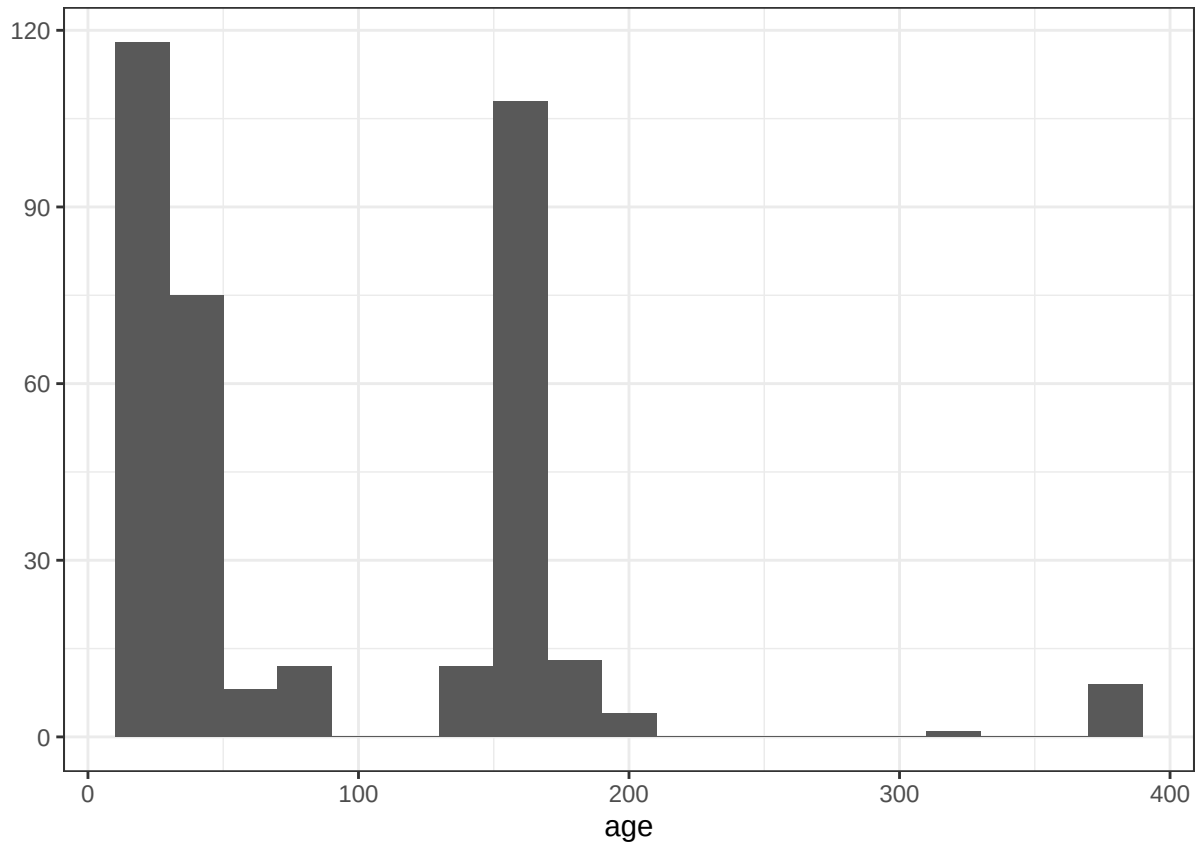
```
## Skipping install of 'phyloseqGraphTest' from a github remote, the SHA1 (3fb6c274) has not changed sin
##   Use 'force = TRUE' to force installation
```

```
.inst <- .bioc_packages %in% installed.packages()
if(any(!.inst)){
  source("http://bioconductor.org/biocLite.R")
  biocLite(.bioc_packages[!.inst])
}
```

Pré-traitement

Avant de faire les projections multivariées, nous ajouterons quelques colonnes à nos données d'échantillons, qui peuvent ensuite être utilisées pour annoter les tracés.

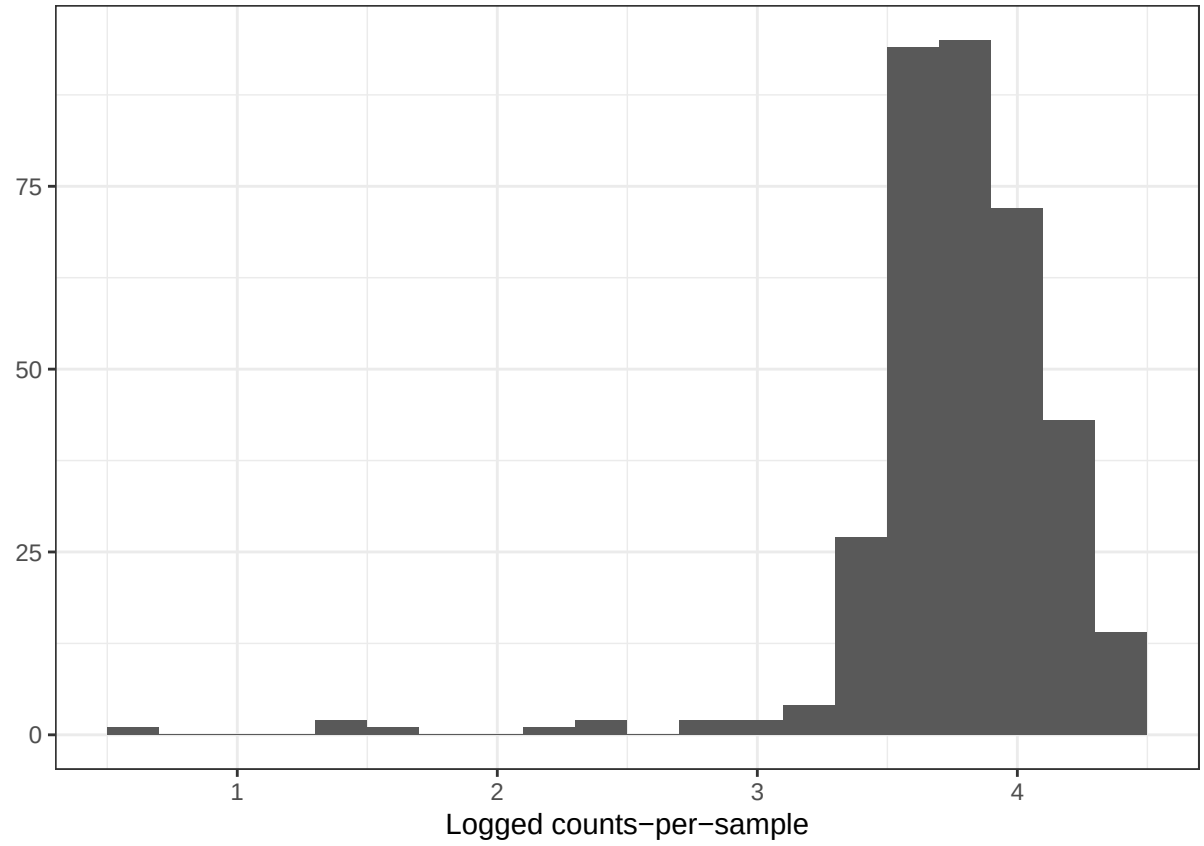
```
qplot(sample_data(ps)$age, geom = "histogram",binwidth=20) + xlab("age")
```



La figure montre l'histogramme des groupes d'âge des souris et que covariable d'âge appartient à trois noeuds distincts. Nous voyons que les âges des souris sont répartis en trois groupes, et nous créons donc une variable catégorique correspondant aux souris jeunes, d'âge moyen et âgées. Nous enregistrons également le nombre

total de comptages observés dans chaque échantillon et transformons les données en une transformation stabilisatrice approximative de la variance.

```
qplot(log10(rowSums(otu_table(ps))),binwidth=0.2) +
  xlab("Logged counts-per-sample")
```

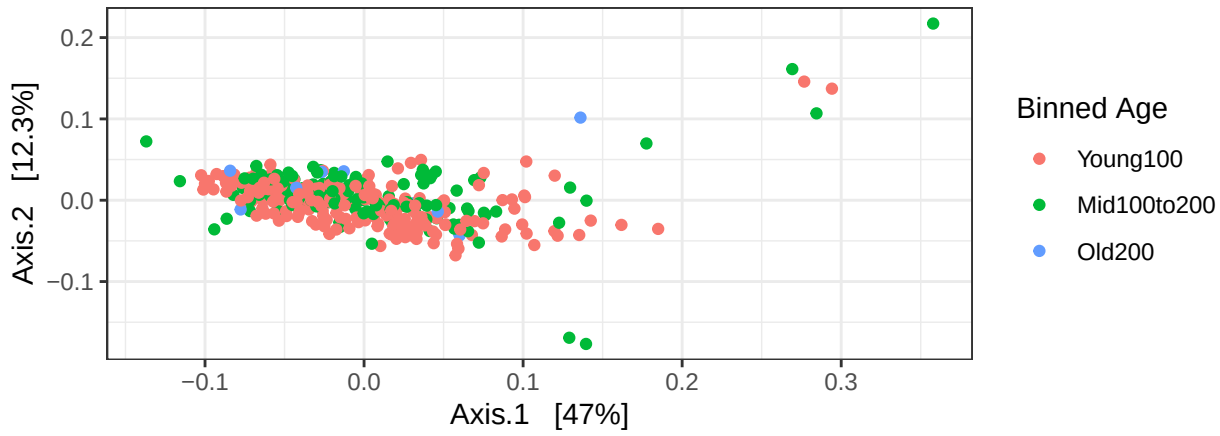


Cette figure représente les histogrammes comparant les profondeurs de lectures brutes et transformées

```
sample_data(ps)$age_binned <- cut(sample_data(ps)$age,
                          breaks = c(0, 100, 200, 400))
levels(sample_data(ps)$age_binned) <- list(Young100="(0,100]", Mid100to200="(100,200]", Old200="(200,40
sample_data(ps)$family_relationship=gsub(" ","",sample_data(ps)$family_relationship)
pslog <- transform_sample_counts(ps, function(x) log(1 + x))
out.wuf.log <- ordinate(pslog, method = "MDS", distance = "wunifrac")
```

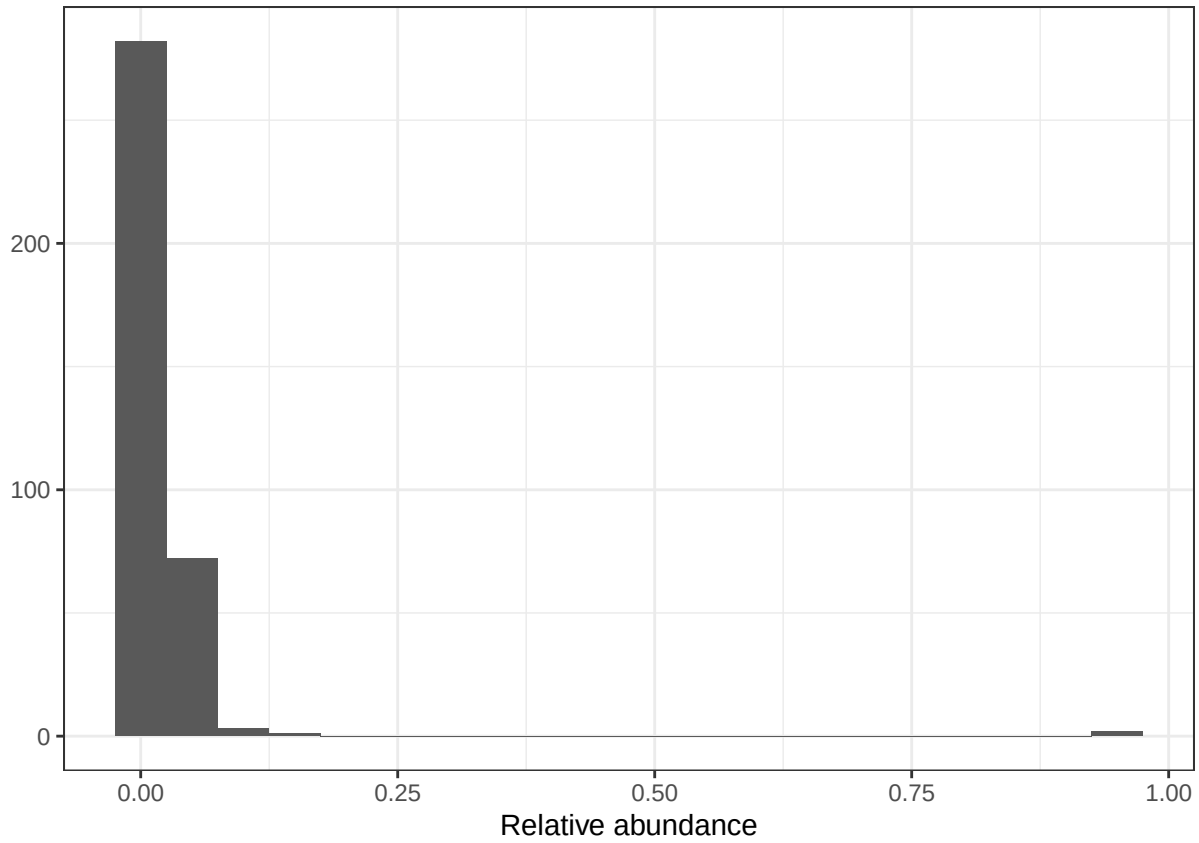
```
## Warning in UniFrac(physeq, weighted = TRUE, ...): Randomly assigning root as --
## GCAAGCGTTGTCCGGATTTACTGGGTGTAAAGGGCGTGCAGCCGGAGAGACAAGTCAGATGTGAAATCCACGGGCTCAACCCGTGAACTGCATTTGAAAC
## -- in the phylogenetic tree in the data you provided.
```

```
evals <- out.wuf.log$values$Eigenvalues
plot_ordination(pslog, out.wuf.log, color = "age_binned") +
  labs(col = "Binned Age") +
  coord_fixed(sqrt(evals[2] / evals[1]))
```



Analyse par PCoA avec la dissimilarité de Bray-Curtis.La figure represente l'analyse d'ordination exploratoire avec le log des abondances. Il y a la presence de quelques valeurs aberrantes.

```
rel_abund <- t(apply(otu_table(ps), 1, function(x) x / sum(x)))
qplot(rel_abund[, 12], geom = "histogram",binwidth=0.05) +
  xlab("Relative abundance")
```



Les échantillons aberrants sont dominés par un seul ASV.

Differentes projections d'ordination

Comme nous l'avons vu, une première étape importante dans l'analyse des données sur le microbiome consiste à effectuer des analyses non supervisées. Ceci est simple à faire dans phyloseq, qui fournit de nombreuses distances et méthodes d'ordination.

Après avoir documenté les valeurs aberrantes, nous allons calculer les ordinations sans ces valeurs aberrantes (qui ont été suprimées) et étudier plus attentivement le résultat.

```
outliers <- c("F5D165", "F6D165", "M3D175", "M4D175", "M5D175", "M6D175")
ps <- prune_samples(!(sample_names(ps) %in% outliers), ps)
```

Nous allons prélever des échantillons avec moins de 1000 lectures :

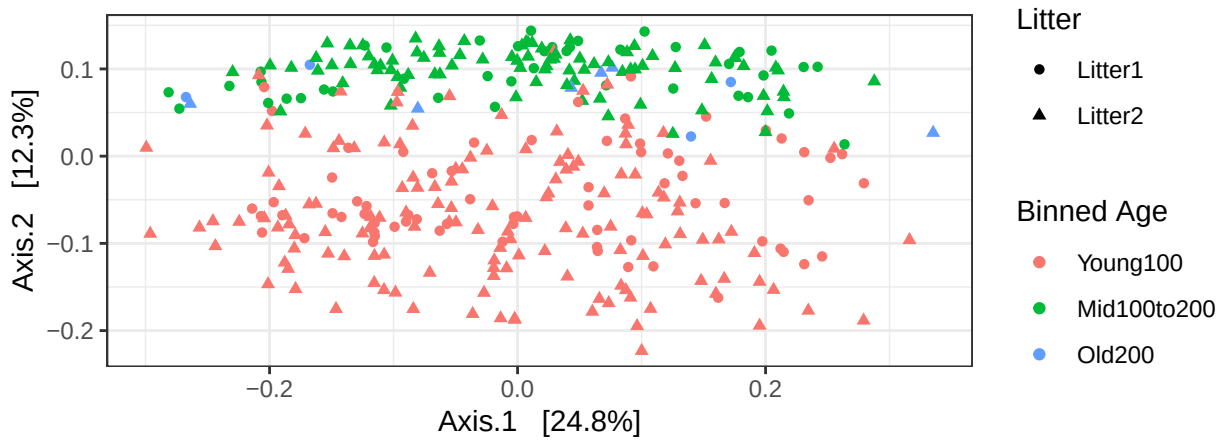
```
which(!rowSums(otu_table(ps)) > 1000)
```

```
## F5D145 M1D149   M1D9 M2D125  M2D19 M3D148 M3D149   M3D3   M3D5   M3D8
##     69    185    200    204    218    243    244    252    256    260
```

```
ps <- prune_samples(rowSums(otu_table(ps)) > 1000, ps)
pslog <- transform_sample_counts(ps, function(x) log(1 + x))
```

Réalisation d'une PCoA en utilisant la dissimilarité de Bray-Curtis :

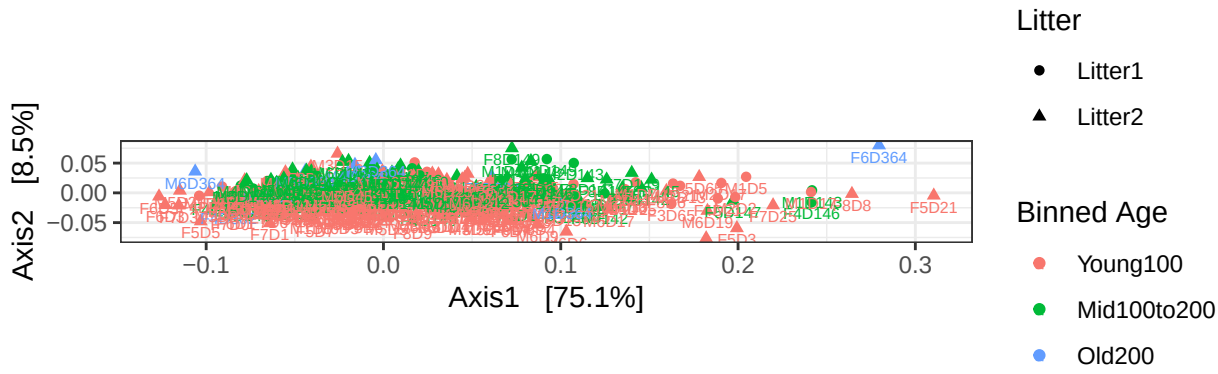
```
out.pcoa.log <- ordinate(pslog,  method = "MDS", distance = "bray")
evals <- out.pcoa.log$values[,1]
plot_ordination(pslog, out.pcoa.log, color = "age_binned",
                  shape = "family_relationship") +
  labs(col = "Binned Age", shape = "Litter")+
  coord_fixed(sqrt(evals[2] / evals[1]))
```



On voit qu'il y a un effet de l'âge assez important qui est cohérent entre toutes les souris, males et femelles, et de portées différentes.

Réalisation d'une DPCoA (analyse des coordonnées principales doubles) qui est une méthode d'ordination phylogénétique et qui fournit une représentation biplotique des échantillons et des catégories taxonomiques :

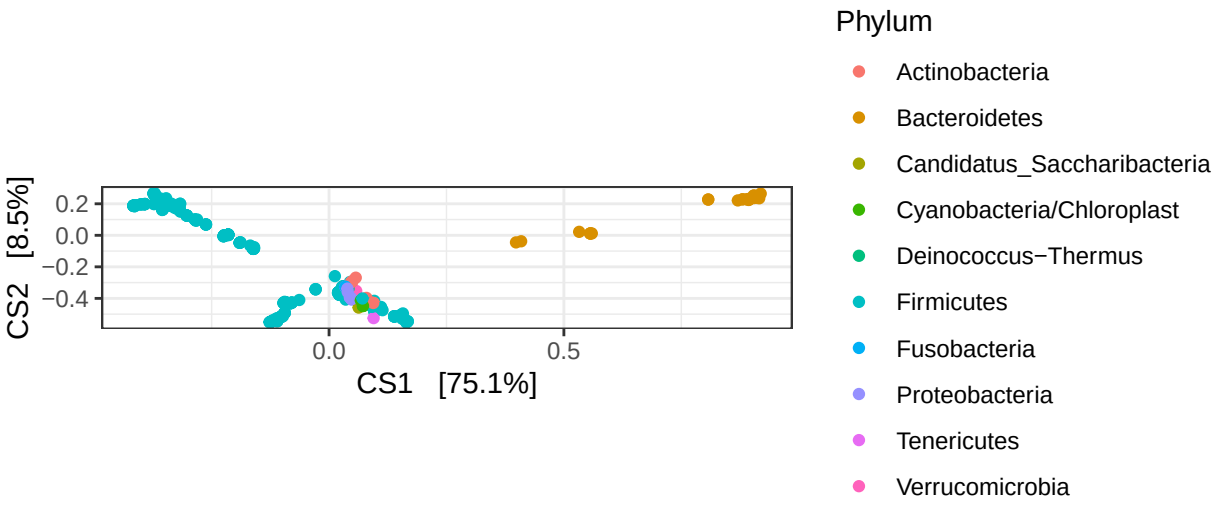
```
library(phyloseq)
library(ggplot2)
out.dpcoa.log <- ordinate(pslog, method = "DPCoA")
evals <- out.dpcoa.log$eig
plot_ordination(pslog, out.dpcoa.log, color = "age_binned", label= "SampleID",
                  shape = "family_relationship") +
  labs(col = "Binned Age", shape = "Litter")+
  coord_fixed(sqrt(evals[2] / evals[1]))
```



Ce graphique représente une DPCoA qui incorpore des informations phylogénétiques, qui sont dominées par le premier axe qui exprime 75 % de variabilité, environ 9 fois celle du second axe, ce qui se traduit par la forme allongée du graphique d'ordination. Nous voyons à nouveau que le deuxième axe correspond aux jeunes vs. vieilles souris, et le biplot suggère une interprétation du deuxième axe : les échantillons qui ont des scores plus élevés sur le deuxième axe ont plus de taxons de Bacteroidetes et un sous-ensemble de Firmicutes.

Enfin, nous pouvons examiner les résultats de PCoA avec Unifrac pondéré :

```
plot_ordination(pslog, out.dpcoa.log, type = "species", color = "Phylum") +
  coord_fixed(sqrt(evals[2] / evals[1]))
```

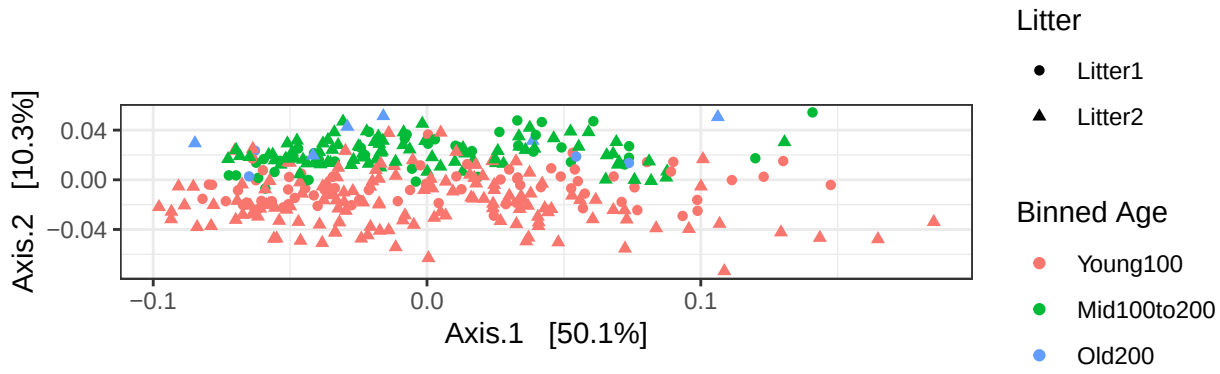


Ce graphe représente les taxons responsables des axes 1 et 2. Comme précédemment, nous constatons que le deuxième axe est associé à un effet d'âge, assez similaire au DPCoA. Cela n'est pas surprenant, car les deux sont des méthodes d'ordination phylogénétique prenant en compte l'abondance. Cependant, lorsque nous comparons les biplots, nous voyons que le DPCoA a donné une interprétation beaucoup plus claire du deuxième axe, par rapport à l'Unifrac pondéré (présenté en-dessous).

```
out.wuf.log <- ordinate(pslog, method = "PCoA", distance ="wunifrac")
```

```
## Warning in UniFrac(physeq, weighted = TRUE, ...): Randomly assigning root as --
## GCAAGCGTTAATCGGAATTACTGGGCGTAAAGCGCGCGTAGGTGGTTTGTTAAGTTGGATGTGAAATCCCCGGGCTCAACCTGGGAACTGCATTCAAAAC
## -- in the phylogenetic tree in the data you provided.
```

```
evals <- out.wuf.log$values$Eigenvalues
plot_ordination(pslog, out.wuf.log, color = "age_binned",
                  shape = "family_relationship") +
  coord_fixed(sqrt(evals[2] / evals[1])) +
  labs(col = "Binned Age", shape = "Litter")
```



Ce graphique représente les positions d'échantillons produits par un PCoA en utilisant Unifrac pondéré

Pourquoi les plans d'ordination sont-ils présentés de cette façon ?

Les graphiques précédents de PCoA ne sont pas représentés sous forme de "carré" comme c'est souvent le cas dans la littérature. Cela s'explique par le fait que la représentation des distances entre les échantillons doit être la plus fidèle possible. Il faut tenir compte du fait que la deuxième valeur est toujours plus petite que la première, parfois considérablement. Ainsi une normalisation est réalisée. Cela garantit que la variabilité représentée est effectuée de manière fidèle.

Analyse en composantes principales (PCA) sur les rangs

Les données sur l'abondance microbienne sont souvent très détaillées, et il peut parfois être difficile d'identifier une normalisation. Dans ces cas, il peut être plus sûr d'ignorer les abondances brutes et de travailler à la place avec les rangs. Nous démontrons cette idée en utilisant une version transformée de la PCA. Tout d'abord, nous créons une nouvelle matrice, représentant les abondances par leurs rangs, où le microbe le plus petit dans un échantillon est représenté au rang 1, deuxième plus petit au rang 2, etc.

```
library(phyloseq)
abund <- otu_table(pslog)
abund_ranks <- t(apply(abund, 1, rank))
```

L'utilisation aléatoire de ces rangs pourrait faire des différences entre les paires de microbes à abondance faible et élevée, comparables. Dans le cas où de nombreuses bactéries sont absentes ou présentes à des

quantités infimes, une différence de rang artificiellement importante pourrait se produire pour les taxons peu abondants. Pour éviter cela, tous ces microbes dont le rang est inférieur à un certain seuil sont fixés à égalité à 1. Les rangs pour les autres microbes sont décalés vers le bas, de sorte qu'il n'y a pas de grand écart entre les rangs.

```
abund_ranks <- abund_ranks - 329
abund_ranks[abund_ranks < 1] <- 1
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:gridExtra':
##
##     combine

## The following objects are masked from 'package:Biostrings':
##
##     collapse, intersect, setdiff, setequal, union

## The following object is masked from 'package:XVector':
##
##     slice

## The following objects are masked from 'package:IRanges':
##
##     collapse, desc, intersect, setdiff, slice, union

## The following objects are masked from 'package:S4Vectors':
##
##     first, intersect, rename, setdiff, setequal, union

## The following objects are masked from 'package:BiocGenerics':
##
##     combine, intersect, setdiff, union

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

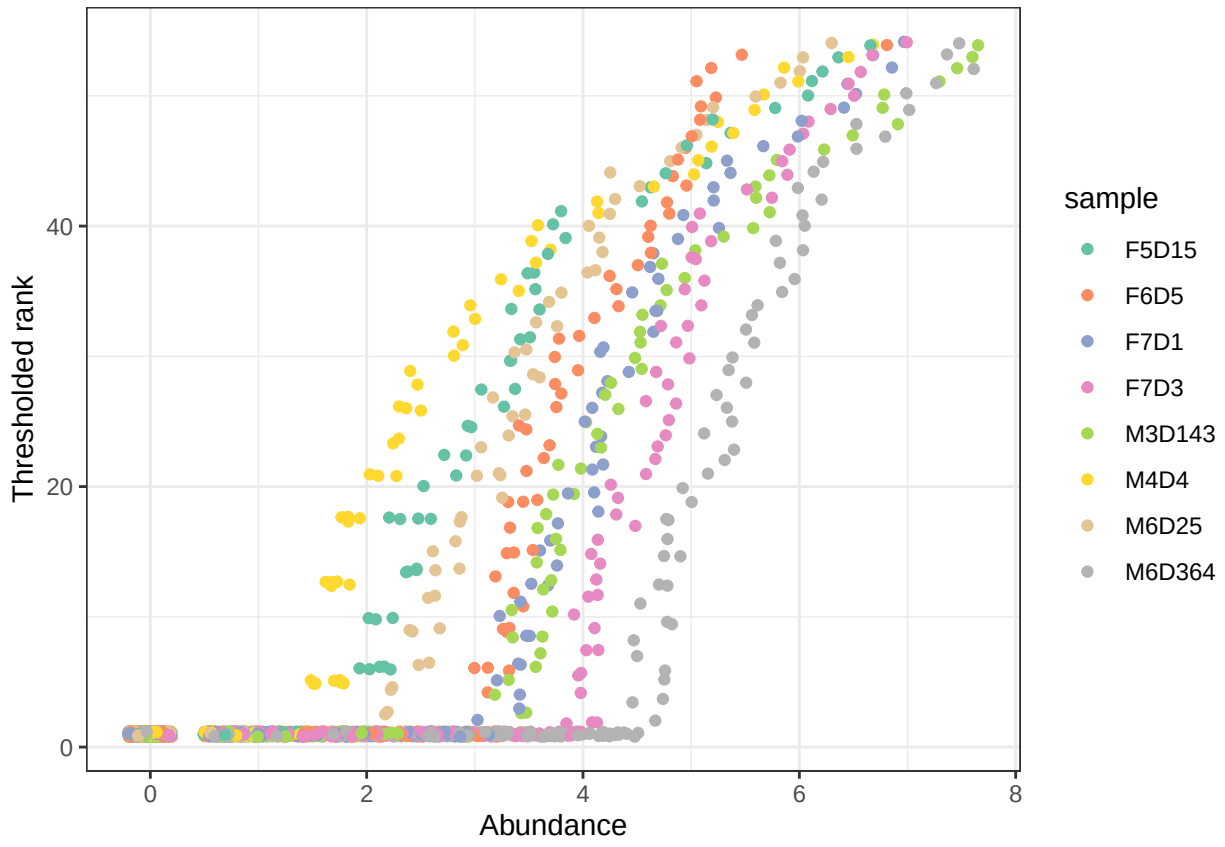
```
library(reshape2)
library(phyloseq)
library(ggplot2)
abund_df <- melt(abund, value.name = "abund") %>%
  left_join(melt(abund_ranks, value.name = "rank"))
```

```
## Joining, by = c("Var1", "Var2")
```

```
colnames(abund_df) <- c("sample", "seq", "abund", "rank")

abund_df <- melt(abund, value.name = "abund") %>%
  left_join(melt(abund_ranks, value.name = "rank"))
```

```
## Joining, by = c("Var1", "Var2")
```

```
colnames(abund_df) <- c("sample", "seq", "abund", "rank")

sample_ix <- sample(1:nrow(abund_df), 8)
ggplot(abund_df %>%
         filter(sample %in% abund_df$sample[sample_ix])) +
  geom_point(aes(x = abund, y = rank, col = sample),
             position = position_jitter(width = 0.2), size = 1.5) +
  labs(x = "Abundance", y = "Thresholded rank") +
  scale_color_brewer(palette = "Set2")
```



Ce graphique représente la transformation au seuil de rang. L'association entre l'abondance et le rang, pour quelques échantillons choisis au hasard est présentée ici.

Nous pouvons maintenant effectuer une PCA et étudier le biplot résultant, représenté dans la figure ci-dessous.

```
library(ade4)
```

```
##
## Attaching package: 'ade4'
```

```
## The following object is masked from 'package:Biostrings':
##
##     score
```

```
## The following object is masked from 'package:BiocGenerics':
##
##     score
```

```
library(base)
ranks_pca <- dudi.pca(abund_ranks, scannf = F, nf = 3)
row_scores <- data.frame(li = ranks_pca$li,
                         SampleID = rownames(abund_ranks))
col_scores <- data.frame(co = ranks_pca$co,
                         seq = colnames(abund_ranks))
tax <- tax_table(ps) %>%
  data.frame(stringsAsFactors = FALSE)
tax$seq <- rownames(tax)
main_orders <- c("Clostridiales", "Bacteroidales", "Lactobacillales",
                 "Coriobacteriales")
tax$Order[!(tax$Order %in% main_orders)] <- "Other"
tax$Order <- factor(tax$Order, levels = c(main_orders, "Other"))
tax$otu_id <- seq_len(ncol(otu_table(ps)))
row_scores <- row_scores %>%
  left_join(sample_data(pslog))
```

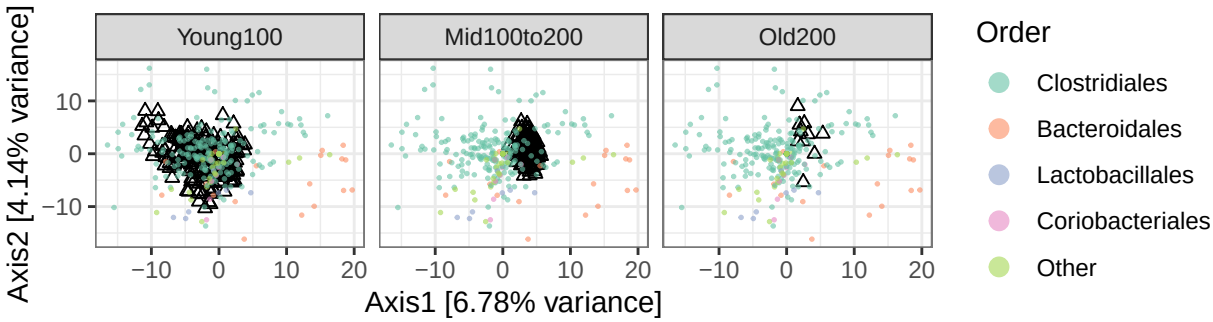
```
## Joining, by = "SampleID"
```

```
## Warning in class(x) <- c(setdiff(subclass, tibble_class), tibble_class): Setting
## class(x) to multiple strings ("tbl_df", "tbl", ...); result will no longer be an
## S4 object
```

```
col_scores <- col_scores %>%
  left_join(tax)
```

```
## Joining, by = "seq"
```

```
evals_prop <- 100 * (ranks_pca$eig / sum(ranks_pca$eig))
ggplot() +
  geom_point(data = row_scores, aes(x = li.Axis1, y = li.Axis2), shape = 2) +
  geom_point(data = col_scores, aes(x = 25 * co.Comp1, y = 25 * co.Comp2, col = Order),
             size = .3, alpha = 0.6) +
  scale_color_brewer(palette = "Set2") +
  facet_grid(~ age_binned) +
  guides(col = guide_legend(override.aes = list(size = 3))) +
  labs(x = sprintf("Axis1 [%s%% variance]", round(evals_prop[1], 2)),
       y = sprintf("Axis2 [%s%% variance]", round(evals_prop[2], 2))) +
  coord_fixed(sqrt(ranks_pca$eig[2] / ranks_pca$eig[1])) +
  theme(panel.border = element_rect(color = "#787878", fill = alpha("white", 0)))
```



Ce graphique présente le biplot résultant de l'ACP après la transformation de classement tronqué. Les résultats sont similaires aux analyses PCoA calculées sans appliquer une transformation de classement tronqué, renforçant notre confiance dans l'analyse sur les données d'origine.

Correspondance canonique

L'analyse de correspondance canonique (CCpnA) est une approche d'ordination d'une espèce par tableau d'échantillons qui incorpore des informations supplémentaires sur les échantillons. Comme précédemment, le but de la création de biplots est de déterminer quels types de communautés bactériennes sont les plus importants dans différents types d'échantillons de souris. Il peut être plus facile d'interpréter ces biplots lorsque l'ordre entre les échantillons reflète les caractéristiques de l'échantillon – les variations d'âge ou le portage dans les données de la souris, par exemple – et ce qui est au cœur de la conception de la CCpnA.

La fonction nous permet de créer des biplots où les positions des échantillons sont déterminées par la similitude des signatures d'espèces et des caractéristiques environnementales. Par contre, l'analyse des composantes principales ou l'analyse de correspondance ne s'intéressent qu'aux signatures d'espèces.

Comme la PCoA et la DPCoA, cette méthode peut être exécutée en utilisant ordinate à partir du package phyloseq. Afin d'utiliser les données d'échantillons supplémentaires, il est nécessaire de fournir un argument supplémentaire, en spécifiant les caractéristiques à prendre en compte. Sinon, par défaut, l'utilisation de toutes les mesures lors de la production de l'ordination.

```
ps_ccpna <- ordinate(pslog, "CCA", formula = pslog ~ age_binned + family_relationship)
```

Pour accéder aux positions du biplot, nous pouvons utiliser la fonction ordinate dans phyloseq. Sur les 23 ordres taxonomiques totaux, nous annotons seulement explicitement les quatre plus abondants, ce qui rend le biplot plus facile à lire.

```
library(ggrepel)
library(phyloseq)
library(ggplot2)
library(dplyr)
ps_scores <- vegan::scores(ps_ccpna)
sites <- data.frame(ps_scores$sites)
sites$SampleID <- rownames(sites)
sites <- sites %>%
  left_join(sample_data(ps))
```

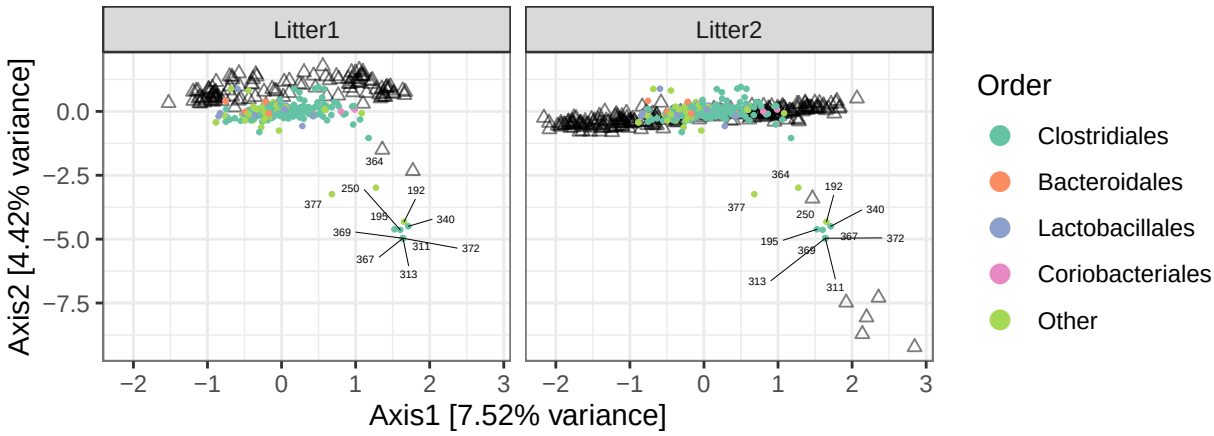
```
## Joining, by = "SampleID"
```

```
## Warning in class(x) <- c(setdiff(subclass, tibble_class), tibble_class): Setting
## class(x) to multiple strings ("tbl_df", "tbl", ...); result will no longer be an
## S4 object
```

```
species <- data.frame(ps_scores$species)
species$otu_id <- seq_along(colnames(otu_table(ps)))
species <- species %>%
  left_join(tax)
```

```
## Joining, by = "otu_id"
```

```
evals_prop <- 100 * ps_ccpna$CCA$eig[1:2] / sum(ps_ccpna$CA$eig)
ggplot() +
  geom_point(data = sites, aes(x = CCA1, y = CCA2), shape = 2, alpha = 0.5) +
  geom_point(data = species, aes(x = CCA1, y = CCA2, col = Order), size = 0.5) +
  geom_text_repel(data = species %>% filter(CCA2 < -2),
                  aes(x = CCA1, y = CCA2, label = otu_id),
            size = 1.5, segment.size = 0.1) +
  facet_grid(. ~ family_relationship) +
  guides(col = guide_legend(override.aes = list(size = 3))) +
  labs(x = sprintf("Axis1 [%s%% variance]", round(evals_prop[1], 2)),
       y = sprintf("Axis2 [%s%% variance]", round(evals_prop[2], 2))) +
  scale_color_brewer(palette = "Set2") +
  coord_fixed(sqrt(ps_ccpna$CCA$eig[2] / ps_ccpna$CCA$eig[1])*0.45   ) +
  theme(panel.border = element_rect(color = "#787878", fill = alpha("white", 0)))
```



Ce graphique montre les scores des souris et des bactéries générés par CCpnA.

Apprentissage supervisé

Puisque nous avons vu que les signatures de microbiome changent avec l'âge, nous allons appliquer des techniques supervisées pour essayer de prédire l'âge à partir de la composition de microbiome. La première étape consiste à diviser les données en ensembles, avec des assignations par souris, plutôt que par échantillon, pour s'assurer que l'ensemble du test simule de manière réaliste la collecte de nouvelles données. Une fois les données divisées, nous pouvons utiliser la fonction train pour adapter le modèle PLS (Partial Least Squares).

```
library(caret)
```

```
## Loading required package: lattice
```

```
library(igraph)
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union
```

```
## The following objects are masked from 'package:ape':
##
##     edges, mst, ring

## The following object is masked from 'package:Biostrings':
##
##     union

## The following object is masked from 'package:XVector':
##
##     path

## The following object is masked from 'package:IRanges':
##
##     union

## The following object is masked from 'package:S4Vectors':
##
##     union

## The following objects are masked from 'package:BiocGenerics':
##
##     normalize, path, union

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union
```

```
sample_data(pslog)$age2 <- cut(sample_data(pslog)$age, c(0, 100, 400))
dataMatrix <- data.frame(age = sample_data(pslog)$age2, otu_table(pslog))
# take 8 mice at random to be the training set, and the remaining 4 the test set
trainingMice <- sample(unique(sample_data(pslog)$host_subject_id), size = 8)
inTrain <- which(sample_data(pslog)$host_subject_id %in% trainingMice)
training <- dataMatrix[inTrain,]
testing <- dataMatrix[-inTrain,]
plsFit <- train(age ~ ., data = training,
                method = "pls", preProc = "center")
```

Ensuite, nous pouvons prédire les étiquettes de classe sur l'ensemble du test en utilisant la fonction predict et comparer aux valeurs réelles. On peut voir que la méthode donne de bons résultats dans la prévision de l'âge.

```
plsClasses <- predict(plsFit, newdata = testing)
table(plsClasses, testing$age)
```

```
##
## plsClasses  (0,100] (100,400]
##   (0,100]        64         2
##   (100,400]       5        44
```

Comme autre exemple, nous pouvons essayer des données aléatoires de forêts. Ceci est exécuté exactement de la même manière que PLS, en changeant l'argument. Les forêts aléatoires donnent également de bons résultats à la tâche de prévision sur cet ensemble de tests.

```
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:gridExtra':
##
##     combine

## The following object is masked from 'package:BiocGenerics':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
rfFit <- train(age ~ ., data = training, method = "rf",
               preProc = "center", proximity = TRUE)
rfClasses <- predict(rfFit, newdata = testing)
table(rfClasses, testing$age)
```

```
##
## rfClasses   (0,100] (100,400]
##   (0,100]        68         6
##   (100,400]       1        40
```

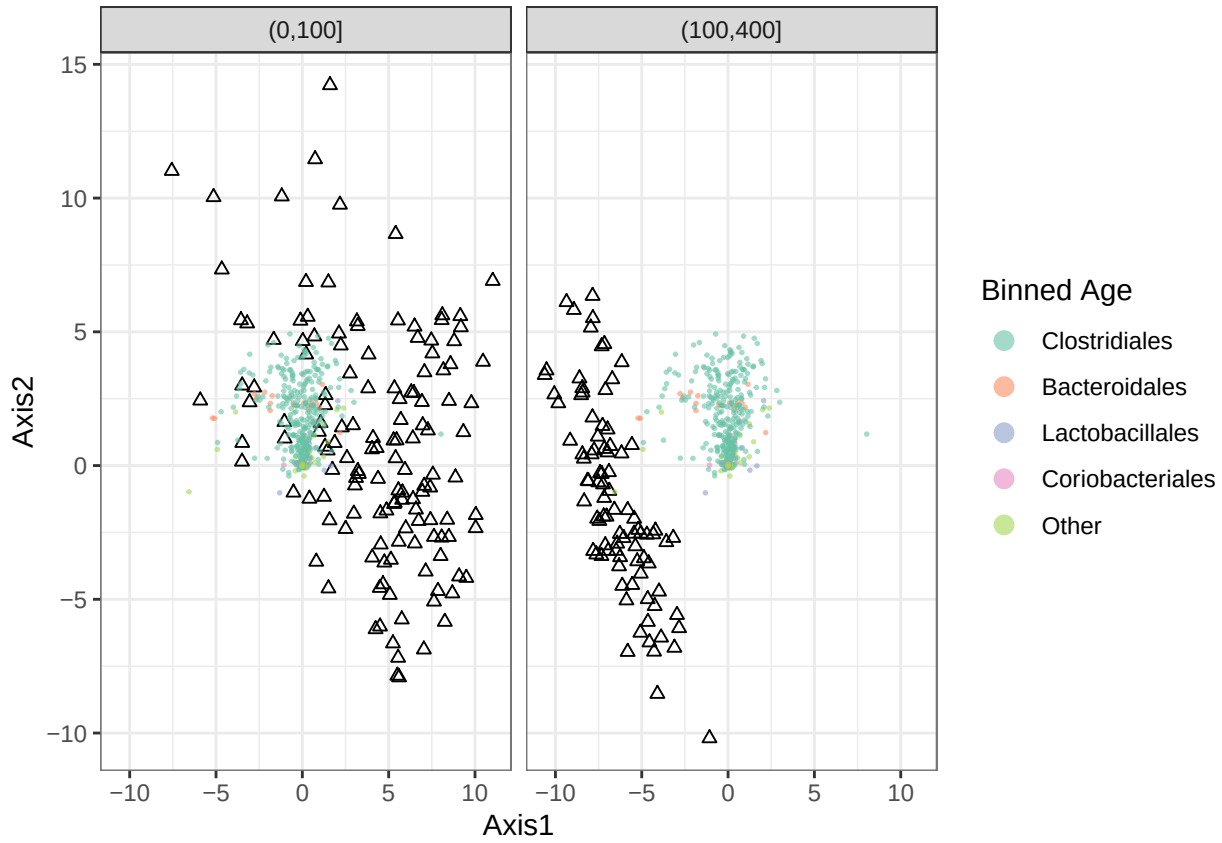
Pour interpréter ces résultats du PLS et des forêts aléatoires, il est normal de produire des biplots et des biplots de proximité. Le code ci-dessous extrait les coordonnées et fournit l'annotation pour les points à inclure sur le biplot PLS.

```
library(phyloseq)
library(ggplot2)
library(vegan)
```

```
## Loading required package: permute

##
## Attaching package: 'permute'
```

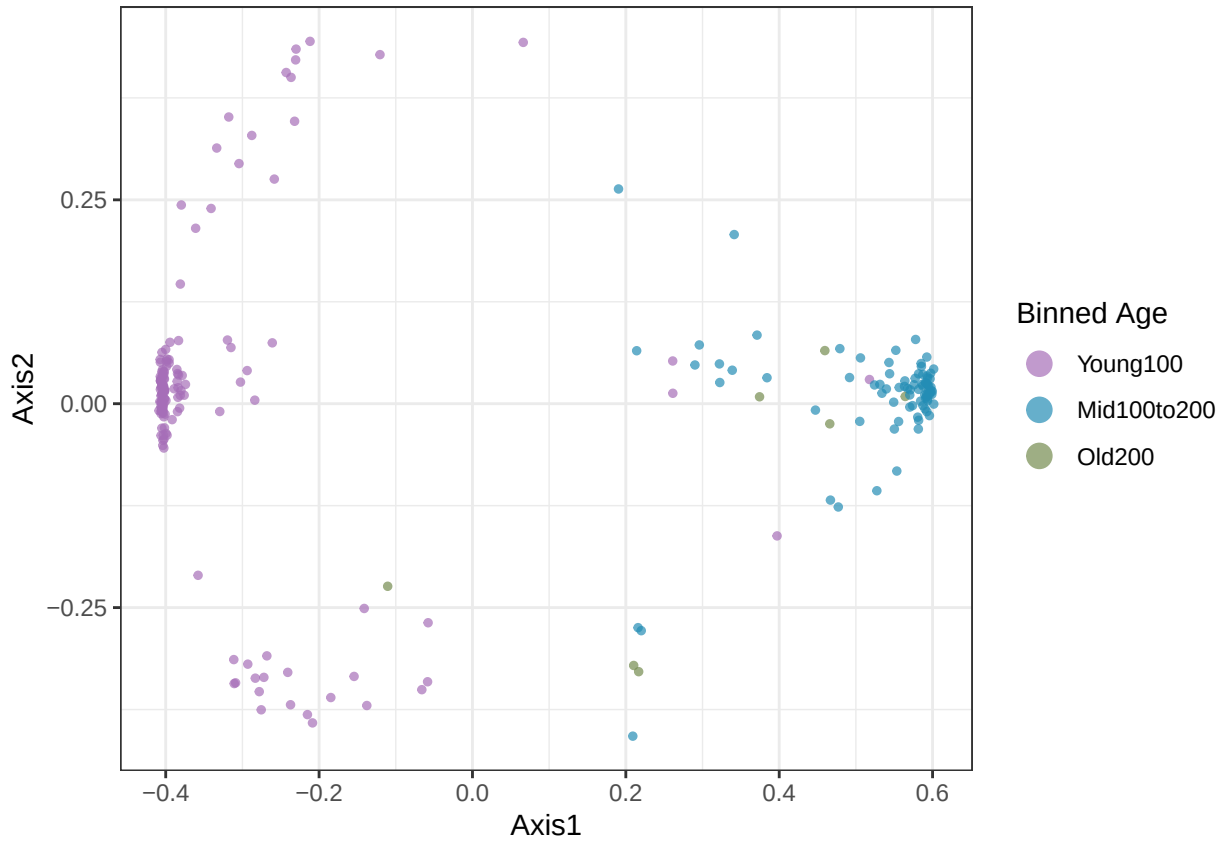
```
## The following object is masked from 'package:igraph':
##
##     permute

## This is vegan 2.5-6

##
## Attaching package: 'vegan'

## The following object is masked from 'package:igraph':
##
##     diversity

## The following object is masked from 'package:caret':
##
##     tolerance
```

```
pls_biplot <- list("loadings" = loadings(plsFit$finalModel),
                   "scores" = scores(plsFit$finalModel))
class(pls_biplot$scores) <- "matrix"

pls_biplot$scores <- data.frame(sample_data(pslog)[inTrain, ],
                                pls_biplot$scores)

tax <- tax_table(ps)@.Data %>%
  data.frame(stringsAsFactors = FALSE)
main_orders <- c("Clostridiales", "Bacteroidales", "Lactobacillales",
                 "Coriobacteriales")
tax$Order[!(tax$Order %in% main_orders)] <- "Other"
tax$Order <- factor(tax$Order, levels = c(main_orders, "Other"))
class(pls_biplot$loadings) <- "matrix"
pls_biplot$loadings <- data.frame(tax, pls_biplot$loadings)
ggplot() +
  geom_point(data = pls_biplot$scores,
             aes(x = Comp.1, y = Comp.2), shape = 2) +
  geom_point(data = pls_biplot$loadings,
             aes(x = 25 * Comp.1, y = 25 * Comp.2, col = Order),
             size = 0.3, alpha = 0.6) +
  scale_color_brewer(palette = "Set2") +
  labs(x = "Axis1", y = "Axis2", col = "Binned Age") +
  guides(col = guide_legend(override.aes = list(size = 3))) +
  facet_grid( ~ age2) +
  theme(panel.border = element_rect(color = "#787878", fill = alpha("white", 0)))
```



Ce graphe représente un PLS qui produit une représentation biplot conçue pour séparer les échantillons par une variable. Il peut être interprété de la même manière que les diagrammes d'ordination antérieurs, à l'exception que la projection est choisie avec une référence explicite à la variable d'âge compartimentée. Plus précisément, le PLS identifie un sous-espace pour maximiser la discrimination entre les classes, et le biplot affiche des projections d'échantillon et des coefficients de ASV à l'égard de ce sous-espace.

```
rf_prox <- cmdscale(1 - rfFit$finalModel$proximity) %>%
  data.frame(sample_data(pslog)[inTrain, ])

ggplot(rf_prox) +
  geom_point(aes(x = X1, y = X2, col = age_binned),
             size = 1, alpha = 0.7) +
  scale_color_manual(values = c("#A66EB8", "#238DB5", "#748B4F")) +
  guides(col = guide_legend(override.aes = list(size = 4))) +
  labs(col = "Binned Age", x = "Axis1", y = "Axis2")
```



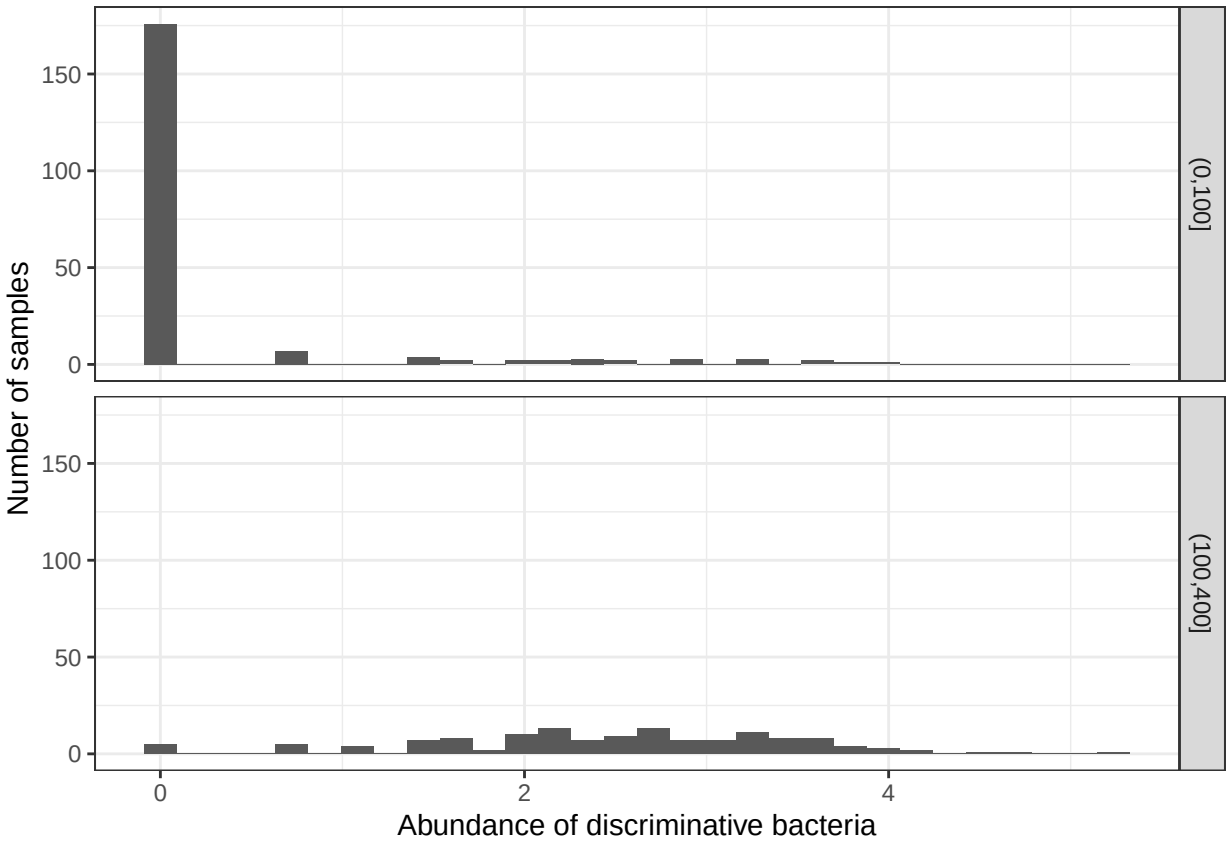
Ce graphique représente un modèle de foret aléatoire qui determine une distance entre les échantillons, qui peut être entrée dans une PCoA pour produire un graphique de proximité. Pour générer cette représentation, une distance est calculée entre les échantillons en fonction de la fréquence à laquelle l'échantillon se produit dans la même branche d'arbre dans la procédure de bootstrapping de la forêt aléatoire. Si une paire d'échantillons se produit fréquemment dans la même branche, la paire est assignée à une faible distance. Les distances résultantes sont ensuite entrées dans la PCoA, donnant un aperçu du mécanisme de classification qui est complexe des forêts aléatoires. La séparation entre les classes est claire, et des points d'inspection manuelle révéleraient quels types d'échantillons sont plus faciles ou plus difficiles à classer.

```
as.vector(tax_table(ps)[which.max(importance(rfFit$finalModel)), c("Family", "Genus")])
```

```
## [1] "Lachnospiraceae" "Roseburia"
```

```
impOtu <- as.vector(otu_table(pslog)[,which.max(importance(rfFit$finalModel))])
maxImpDF <- data.frame(sample_data(pslog), abund = impOtu)
ggplot(maxImpDF) +   geom_histogram(aes(x = abund)) +
  facet_grid(age2 ~ .) +
  labs(x = "Abundance of discriminative bacteria", y = "Number of samples")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Pour mieux comprendre le modèle de forêt aléatoire ajusté, nous identifions le microbe ayant le plus d'influence dans la prédiction aléatoire de la forêt. Il s'agit d'un microbe de la famille des Lachnospiracées et du genre Roseburia. La figure représente son abondance dans les échantillons. On voit qu'il est uniformément très bas de 0 à 100 jours et beaucoup plus élevé de 100 à 400 jours

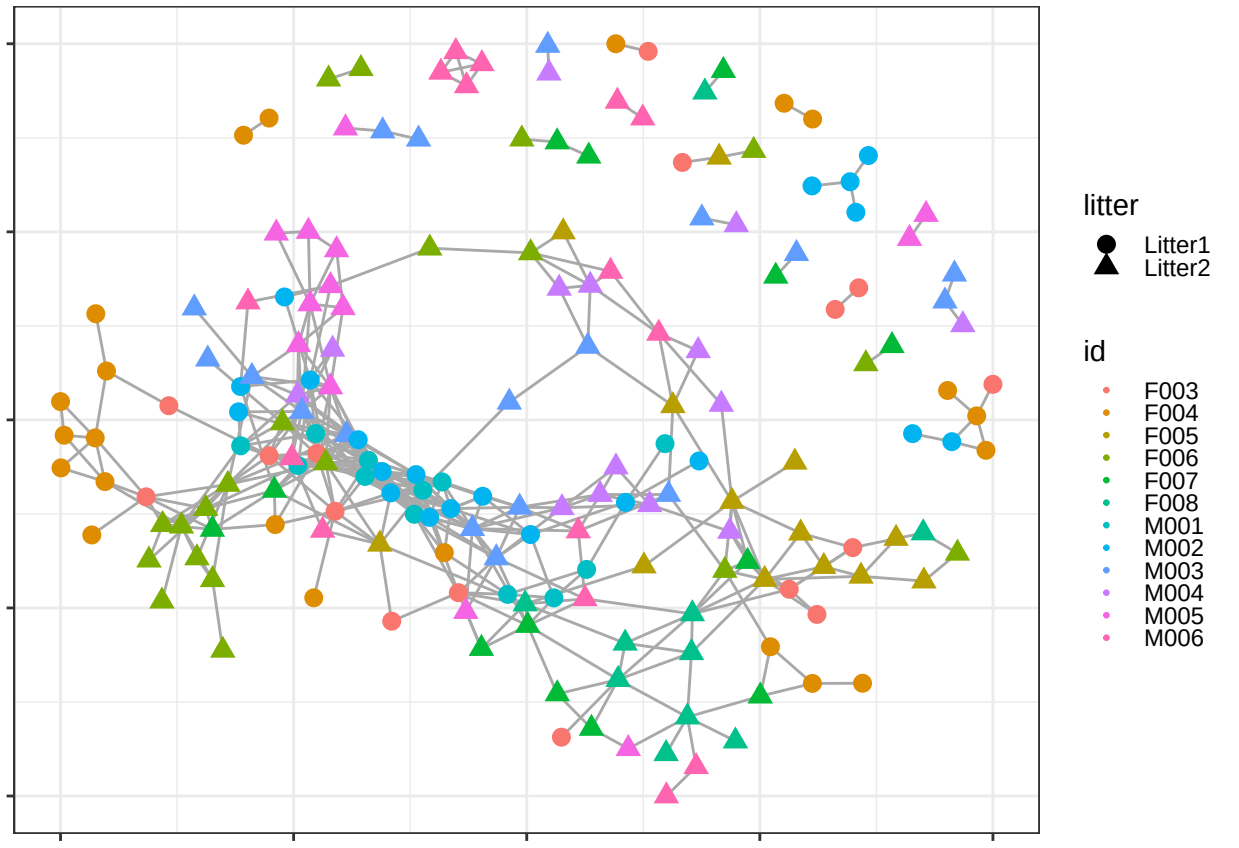
Analyses graphiques

Créer et tracer des graphiques

Phyloseq a une fonctionnalité pour créer des graphiques basés sur le seuil d'une matrice de distance, et les réseaux résultants peuvent être tracés à l'aide du package ggnetwork. Ce package se superpose à la syntaxe ggplot, donc vous pouvez utiliser la fonction ggplot sur un objet igraph et ajouter la fonction geoms pour tracer le réseau. Pour pouvoir colorer les nœuds ou les bords d'une certaine façon, nous devons ajouter ces attributs à l'objet igraph. Ci-dessous, nous créons un réseau en multipliant la dissimilarité de Jaccard (la distance par défaut pour la fonction make_network) à . 35. Puis nous ajoutons un attribut aux sommets indiquant de quelle souris provient l'échantillon et dans quelle portée se trouve la souris. Ensuite, nous pouvons tracer le réseau avec la coloration par souris et la forme par portée.

```
library("phyloseqGraphTest")
library("igraph")
library("ggnetwork")
net <- make_network(ps, max.dist=0.35)
sampledata <- data.frame(sample_data(ps))
V(net)$id <- sampledata[names(V(net)), "host_subject_id"]
```

```r
V(net)$litter <- sampledata[names(V(net)), "family_relationship"]

net_graph <- ggnetwork(net)

ggplot(net_graph, aes(x = x, y = y, xend = xend, yend = yend), layout = "fruchtermanreingold") +
  geom_edges(color = "darkgray") +
  geom_nodes(aes(color = id, shape = litter),  size = 3 ) +
  theme(axis.text = element_blank(), axis.title = element_blank(),
        legend.key.height = unit(0.5,"line")) +
  guides(col = guide_legend(override.aes = list(size = .5)))
```



On peut voir sur ce graphique un réseau de la matrice de dissimilarité de Jaccard. Les couleurs de la figure représentent de quelle souris provient l'échantillon et la forme représente la portée dans laquelle se trouvait la souris. Nous pouvons voir qu'il existe un regroupement des échantillons par souris et par portée.

Tests à deux échantillons sur les graphiques

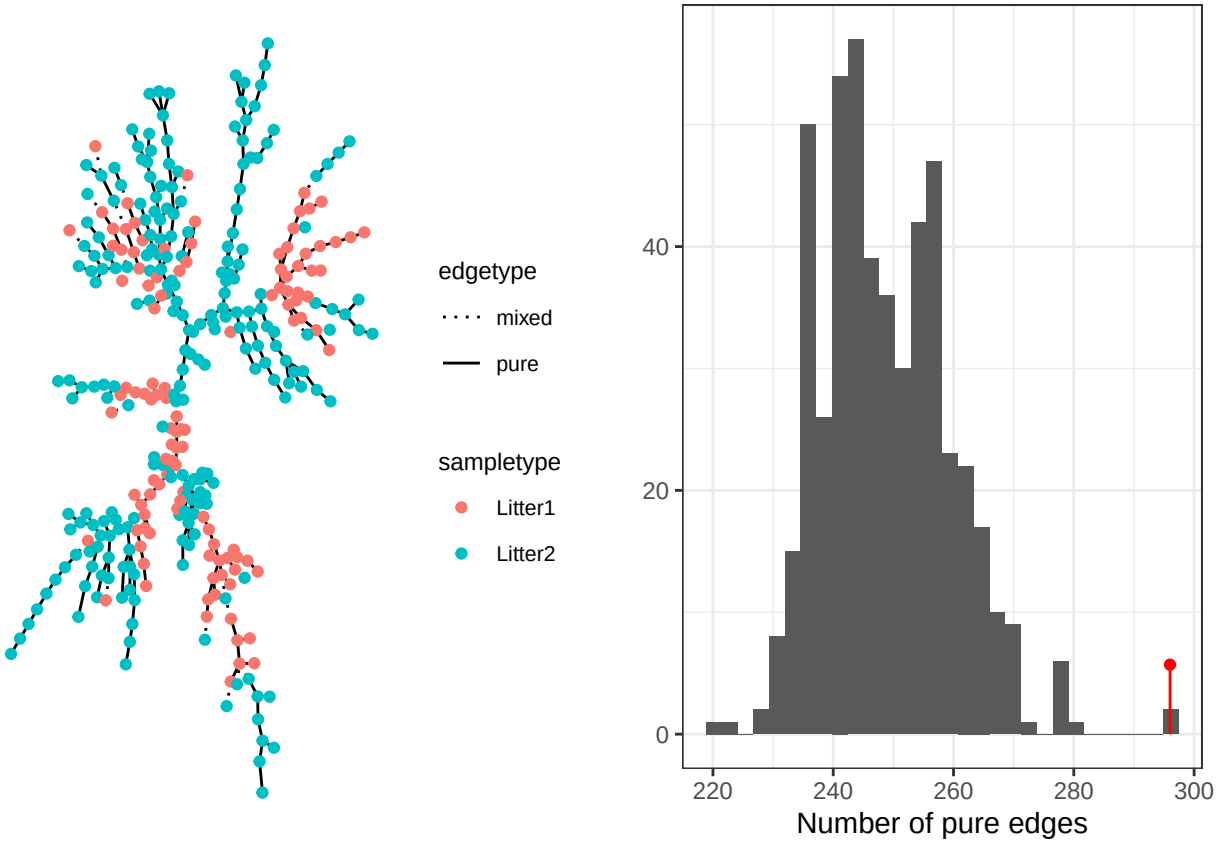
Arbre de recouvrement minimal (MST)

Nous effectuons d'abord un test à l'aide d'un arbre de recouvrement minimal (MST) avec la dissimilarité de Jaccard. Nous voulons savoir si les deux portées (family_relationship) proviennent de la même distribution. Comme il y a un regroupement dans les données par individu (host_subject_id), nous ne pouvons pas simplement permuter tous les étiquettes, nous devons maintenir cette structure imbriquée. C'est ce que fait l'argument grouping. Ici nous permutons les étiquettes mais nous maintenons la structure intacte :

```
gt <- graph_perm_test(ps, "family_relationship", grouping = "host_subject_id",
                      distance = "jaccard", type = "mst")
gt$pval
```

```
## [1] 0.006
```

```
library(phyloseq)
library(ggplot2)
library("gridExtra")
plotNet1=plot_test_network(gt) + theme(legend.text = element_text(size = 8),
        legend.title = element_text(size = 9))
plotPerm1=plot_permutations(gt)
grid.arrange(ncol = 2,  plotNet1, plotPerm1)
```



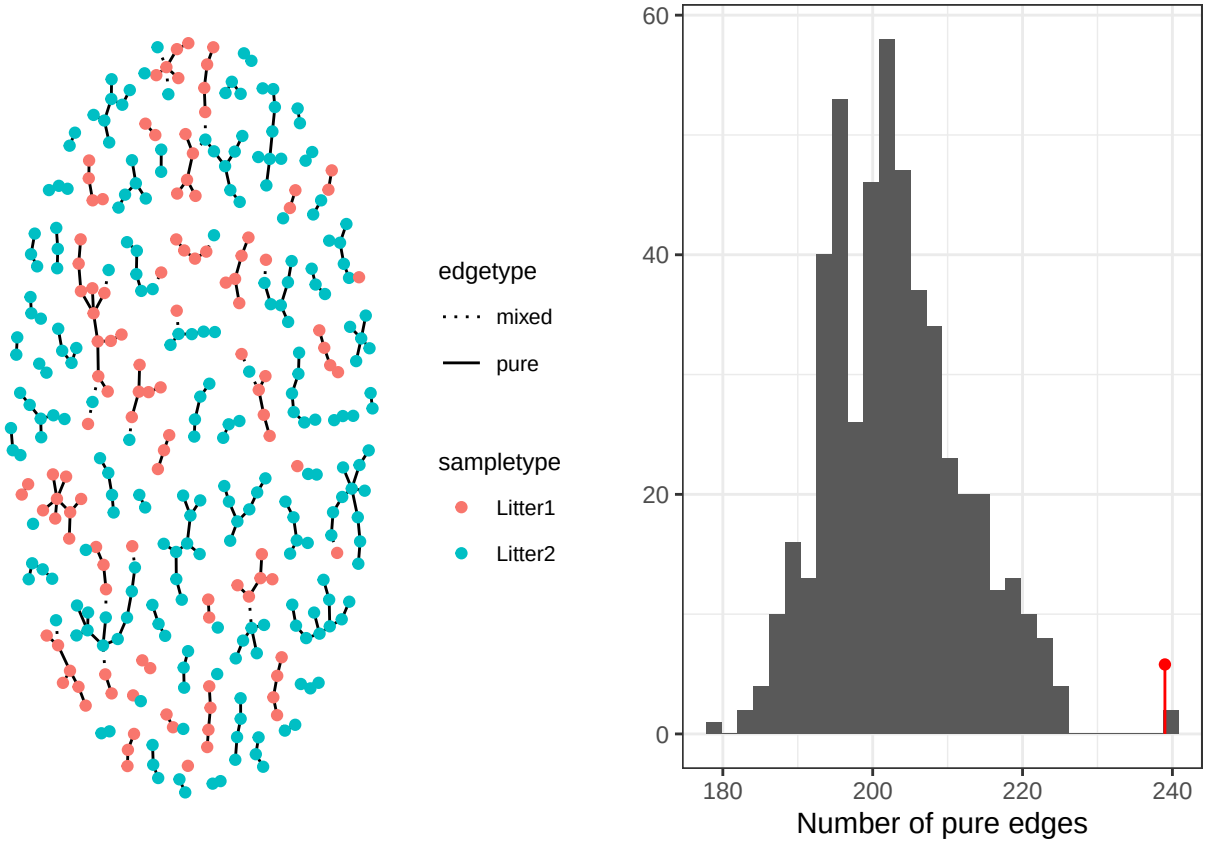
Cette figure représente le graphique et l'histogramme de permutation obtenus à partir de l'arbre de recouvrement minimal avec la similitude de Jaccard. Ce test a un petit p-value, et nous rejetons l'hypothèse nulle que les deux échantillons proviennent de la même distribution. À partir du graphique de l'arbre , nous voyons que les échantillons sont regroupés par portée.

Voisins proches

Le graphique des voisins k les plus proches est obtenu en plaçant une bordure entre deux échantillons chaque fois que l'un d'eux est dans l'ensemble des voisins k les plus proches de l'autre.

```
gt <- graph_perm_test(ps, "family_relationship", grouping = "host_subject_id",
                      distance = "jaccard", type = "knn", knn = 1)
```

```
plotNet2=plot_test_network(gt) + theme(legend.text = element_text(size = 8),
        legend.title = element_text(size = 9))
plotPerm2=plot_permutations(gt)
grid.arrange(ncol = 2,  plotNet2, plotPerm2)
```



Cette figure représente le réseau voisin le plus proche et l'histogramme de permutation. Nous voyons à partir de la figure que si une paire d'échantillons est proche entre eux, ils sont susceptibles d'être de la même portée.

Modélisation linéaire

Il est souvent intéressant d'évaluer dans quelle mesure la diversité de la communauté microbienne reflète les caractéristiques de l'environnement à partir duquel elle a été échantillonnée. Contrairement à l'ordination, l'objectif de cette analyse n'est pas de développer une représentation de nombreuses bactéries par rapport aux caractéristiques de l'échantillon, mais plutôt de décrire comment une mesure unique de la structure globale de la communauté est associée aux caractéristiques de l'échantillon. Il s'agit d'un objectif statistique un peu plus simple qui peut être atteint par la modélisation linéaire, pour laquelle il existe une gamme d'approches en R. Par exemple, nous utiliserons un modèle d'effets pour étudier la relation entre la diversité de la communauté microbienne des souris et les variables de l'âge et de la portée qui ont été notre cible jusqu'à présent. Ce choix était motivé par l'observation que les souris plus jeunes avaient des diversités de Shannon sensiblement inférieures, mais que les souris différentes avaient des diversités de base différentes. Le modèle des effets mixtes est un point de départ pour formaliser cette observation.

Un calcul de la diversité de Shannon associée à chaque échantillon est d'abord effectuée puis elle sera jointe avec l'annotation d'échantillon.

```
library("nlme")
```

```
##
## Attaching package: 'nlme'
```

```
## The following object is masked from 'package:dplyr':
##
##     collapse
```

```
## The following object is masked from 'package:Biostrings':
##
##     collapse
```

```
## The following object is masked from 'package:IRanges':
##
##     collapse
```

```
library("reshape2")
library("Biostrings")
ps_alpha_div <- estimate_richness(ps, split = TRUE, measure = "Shannon")
ps_alpha_div$SampleID <- rownames(ps_alpha_div) %>%
  as.factor()
ps_samp <- sample_data(ps) %>%
  unclass() %>%
  data.frame() %>%
  left_join(ps_alpha_div, by = "SampleID") %>%
  melt(measure.vars = "Shannon",
       variable.name = "diversity_measure",
       value.name = "alpha_diversity")

# reorder's facet from lowest to highest diversity
diversity_means <- ps_samp %>%
  group_by(host_subject_id) %>%
  summarise(mean_div = mean(alpha_diversity)) %>%
  arrange(mean_div)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
ps_samp$host_subject_id <- factor(ps_samp$host_subject_id)
#                                   diversity_means$host_subject_id)
```

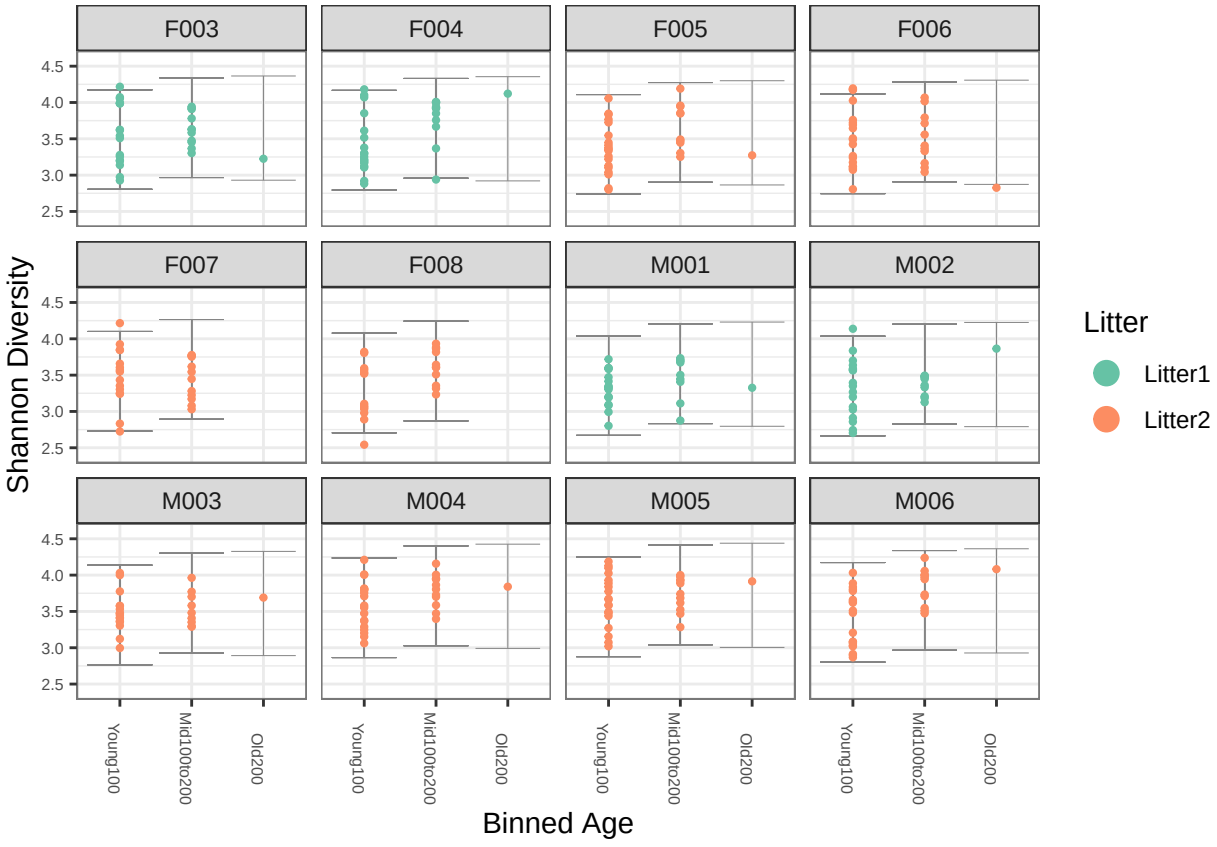
```
alpha_div_model <- lme(fixed = alpha_diversity ~ age_binned, data = ps_samp,
                       random = ~ 1 | host_subject_id)
```

```
new_data <- expand.grid(host_subject_id = levels(ps_samp$host_subject_id),
                        age_binned = levels(ps_samp$age_binned))
new_data$pred <- predict(alpha_div_model, newdata = new_data)
X <- model.matrix(eval(eval(alpha_div_model$call$fixed)[-2]),
```

```
                  new_data[-ncol(new_data)])
pred_var_fixed <- diag(X %*% alpha_div_model$varFix %*% t(X))
new_data$pred_var <- pred_var_fixed + alpha_div_model$sigma ^ 2
```

```
# fitted values, with error bars
ggplot(ps_samp %>% left_join(new_data)) +
  geom_errorbar(aes(x = age_binned, ymin = pred - 2 * sqrt(pred_var),
                    ymax = pred + 2 * sqrt(pred_var)),
                col = "#858585", size = .1) +
  geom_point(aes(x = age_binned, y = alpha_diversity,
                 col = family_relationship), size = 0.8) +
  facet_wrap(~host_subject_id) +
  scale_y_continuous(limits = c(2.4, 4.6), breaks = seq(0, 5, .5)) +
  scale_color_brewer(palette = "Set2") +
  labs(x = "Binned Age", y = "Shannon Diversity", color = "Litter") +
  guides(col = guide_legend(override.aes = list(size = 4))) +
  theme(panel.border = element_rect(color = "#787878", fill = alpha("white", 0)),
        axis.text.x = element_text(angle = -90, size = 6),
        axis.text.y = element_text(size = 6))
```

```
## Joining, by = c("host_subject_id", "age_binned")
```



Tests multiples hiérarchiques

Des tests d'hypothèse peuvent être utilisés pour identifier des microbes individuels dont l'abondance se rapporte à des variables d'échantillon d'intérêt. Une approche standard consiste à calculer une statistique d'essai pour chaque bactérie individuellement, en mesurant son association avec les caractéristiques de l'échantillon, puis à ajuster conjointement les valeurs p pour assurer une limite supérieure du taux de découverte erronée.Cependant, cette procédure n'exploite aucune structure parmi les hypothèses testées. Par exemple, il est probable que si une espèce de Ruminococcus est fortement associée à l'âge, d'autres le sont aussi.

Pour résoudre ce problème, une procédure de test hiérarchique, où les groupes taxonomiques ne sont testés que si des niveaux plus élevés sont associés. Dans le cas où de nombreuses espèces apparentées ont un léger signal, cette mise en commun de l'information peut augmenter la puissance.

Ici, une procédure de test hiérarchique permettant l'association entre l'abondance microbienne et l'âge est appliquée. Cela fournit une vue complémentaire des analyses précédentes, identifiant les bactéries individuelles responsables des différences entre les souris jeunes et âgées.

Mais d'abord, on effectue une normalisation par stabilisation de la variance via DESeq2 qui permet d'obtenir des résultats plus puissants (au niveau statistique) et évite d'empiéter sur les données :

```
library("reshape2")
library("DESeq2")
```

```
## Loading required package: GenomicRanges

## Loading required package: GenomeInfoDb

## Loading required package: SummarizedExperiment

## Loading required package: MatrixGenerics

## Loading required package: matrixStats

##
## Attaching package: 'matrixStats'

## The following object is masked from 'package:dplyr':
##
##     count

##
## Attaching package: 'MatrixGenerics'

## The following objects are masked from 'package:matrixStats':
##
##     colAlls, colAnyNAs, colAnys, colAvgsPerRowSet, colCollapse,
##     colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
##     colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,
##     colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,
##     colProds, colQuantiles, colRanges, colRanks, colSdDiffs, colSds,
##     colSums2, colTabulates, colVarDiffs, colVars, colWeightedMads,
##     colWeightedMeans, colWeightedMedians, colWeightedSds,
##     colWeightedVars, rowAlls, rowAnyNAs, rowAnys, rowAvgsPerColSet,
```

```
##     rowCollapse, rowCounts, rowCummaxs, rowCummins, rowCumprods,
##     rowCumsums, rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps,
##     rowMadDiffs, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
##     rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
##     rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs, rowVars,
##     rowWeightedMads, rowWeightedMeans, rowWeightedMedians,
##     rowWeightedSds, rowWeightedVars

## Loading required package: Biobase

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.


##
## Attaching package: 'Biobase'

## The following object is masked from 'package:MatrixGenerics':
##
##     rowMedians

## The following objects are masked from 'package:matrixStats':
##
##     anyMissing, rowMedians

## The following object is masked from 'package:phyloseq':
##
##     sampleNames
```

```
library("phyloseq")
#New version of DESeq2 needs special levels
sample_data(ps)$age_binned <- cut(sample_data(ps)$age,
                          breaks = c(0, 100, 200, 400))
levels(sample_data(ps)$age_binned) <- list(Young100="(0,100]", Mid100to200="(100,200]", Old200="(200,40
sample_data(ps)$family_relationship = gsub(" ", "", sample_data(ps)$family_relationship)
ps_dds <- phyloseq_to_deseq2(ps, design = ~ age_binned + family_relationship)
```

```
## converting counts to integer mode

## Warning in DESeqDataSet(se, design = design, ignoreRank): some variables in
## design formula are characters, converting to factors
```

```
# geometric mean, set to zero when all coordinates are zero
geo_mean_protected <- function(x) {
  if (all(x == 0)) {
    return (0)
  }
  exp(mean(log(x[x != 0])))
}
```

```
geoMeans <- apply(counts(ps_dds), 1, geo_mean_protected)
ps_dds <- estimateSizeFactors(ps_dds, geoMeans = geoMeans)
ps_dds <- estimateDispersions(ps_dds)
```

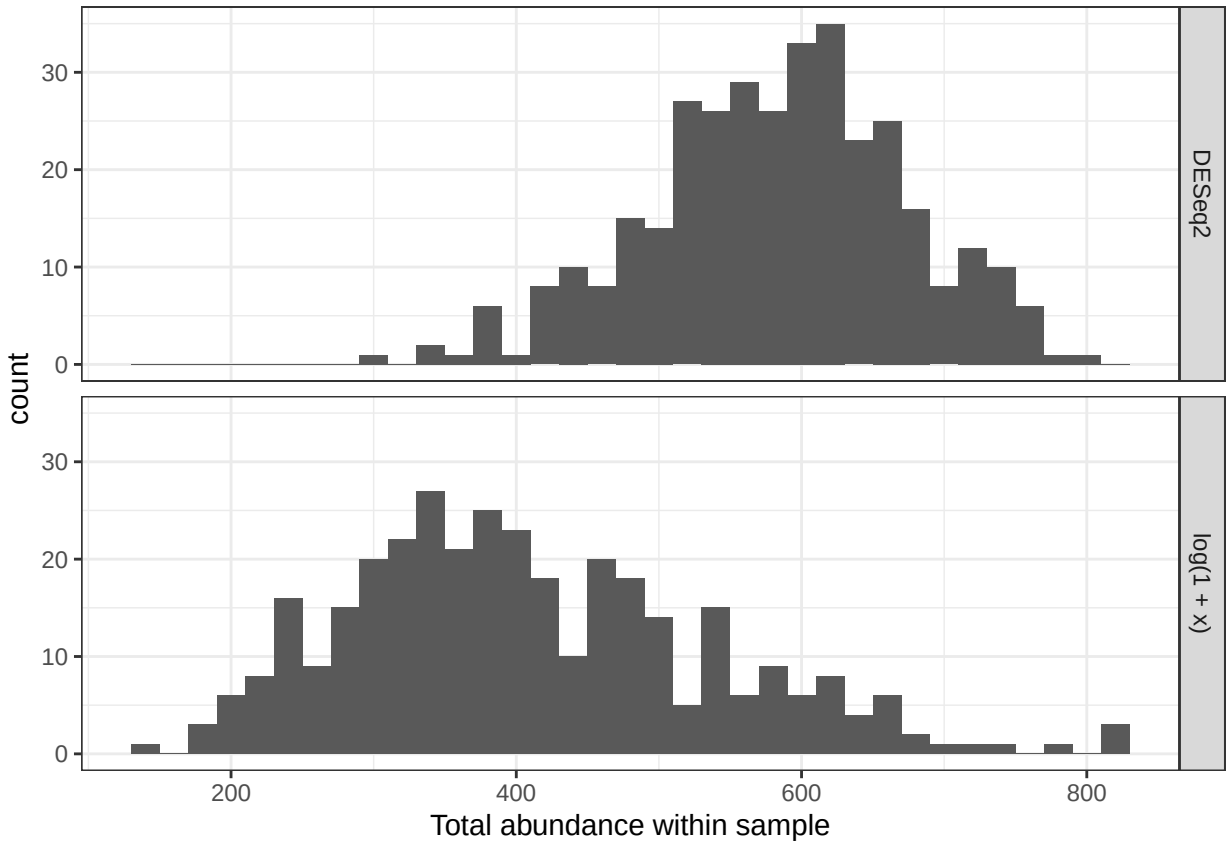
```
## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates
```

```
abund <- getVarianceStabilizedData(ps_dds)
```

Le package structSSI est utilisé pour réaliser les tests hiérarchiques. Les noms de chaque taxon/ASV est raccourci :

```
library("phyloseq")
library("ggplot2")
library("magrittr")
short_names <- substr(rownames(abund), 1, 5)%>%
  make.names(unique = TRUE)
rownames(abund) <- short_names
```

```
abund_sums <- rbind(data.frame(sum = colSums(abund),
                               sample = colnames(abund),
                               type = "DESeq2"),
                    data.frame(sum = rowSums(otu_table(pslog)),
                               sample = rownames(otu_table(pslog)),
                               type = "log(1 + x)"))

ggplot(abund_sums) +
  geom_histogram(aes(x = sum), binwidth = 20) +
  facet_grid(type ~ .) +
  xlab("Total abundance within sample")
```



Les histogrammes représentent l'abondance de la transformation par DESeq. L'histogramme sur le dessus donne l'abondance totale transformée DESeq2 dans chaque échantillon. L'histogramme du bas est le même que celui de la figure précedente, et est inclus pour faciliter la comparaison.

Contrairement aux tests standard d'hypothèses multiples, la procédure de tests hiérarchiques nécessite des tests univariés pour chaque groupe taxonomique de niveau supérieur, et pas seulement pour chaque espèce. Une fonction treePValues est disponible pour cela :

```
library("structSSI")
library("phyloseq")
el <- phy_tree(pslog)$edge
el0 <- el
el0 <- el0[nrow(el):1, ]
el_names <- c(short_names, seq_len(phy_tree(pslog)$Nnode))
el[, 1] <- el_names[el0[, 1]]
el[, 2] <- el_names[as.numeric(el0[, 2])]
unadj_p <- treePValues(el, abund, sample_data(pslog)$age_binned)
```

Nous pouvons maintenant corriger la valeur p en utilisant la procédure de test hiérarchique. Les résultats des tests sont garantis pour contrôler plusieurs variantes de contrôle FDR (false discovery rate), mais à différents niveaux.

```
hfdr_res <- hFDR.adjust(unadj_p, el, .75)
summary(hfdr_res)
```

```
## Number of hypotheses: 764
```

```
## Number of tree discoveries: 579
## Estimated tree FDR: 1
## Number of tip discoveries: 280
## Estimated tips FDR: 1
##
##  hFDR adjusted p-values:
##                 unadjp         adjp adj.significance
## GCAAG.95  1.861873e-82 3.723745e-82              ***
## GCAAG.70  1.131975e-75 2.263950e-75              ***
## GCAAG.187 5.148758e-59 1.029752e-58              ***
## GCAAG.251 3.519276e-50 7.038553e-50              ***
## GCAAG.148 1.274481e-49 2.548962e-49              ***
## GCAAG.30  9.925218e-49 1.985044e-48              ***
## GCGAG.76  1.722591e-46 3.445183e-46              ***
## GCAAG.167 6.249050e-43 1.249810e-42              ***
## 255       8.785479e-40 1.757096e-39              ***
## GCAAG.64  2.727610e-36 5.455219e-36              ***
## [only 10 most significant hypotheses shown]
## ---
## Signif. codes:  0 '***' 0.015 '**' 0.15 '*' 0.75 '.' 1.5 '-' 1
```

```
#interactive part: not run
plot(hfdr_res, height = 5000) # opens in a browser
```

Nous pouvons maintenant retrouver l'identité taxonomique des hypothèses rejetées :

```
library("magrittr")
tax <- tax_table(pslog)[, c("Family", "Genus")] %>%
  data.frame()
tax$seq <- short_names
```

```
library("phyloseq")
library("gridExtra")
library("ggplot2")
library("magrittr")
library("dplyr")
options(digits=3)
hfdr_res@p.vals$seq <- rownames(hfdr_res@p.vals)
tax %>%
  left_join(hfdr_res@p.vals) %>%
  arrange(adjp) %>% head(10)
```

```
## Joining, by = "seq"
```

```
##             Family            Genus       seq   unadjp     adjp
## 1  Lachnospiraceae             <NA>  GCAAG.95 1.86e-82 3.72e-82
## 2  Lachnospiraceae        Roseburia  GCAAG.70 1.13e-75 2.26e-75
## 3  Lachnospiraceae Clostridium_XlVa GCAAG.187 5.15e-59 1.03e-58
## 4  Lachnospiraceae             <NA> GCAAG.251 3.52e-50 7.04e-50
## 5  Lachnospiraceae Clostridium_XlVa GCAAG.148 1.27e-49 2.55e-49
## 6  Lachnospiraceae             <NA>  GCAAG.30 9.93e-49 1.99e-48
## 7  Ruminococcaceae     Ruminococcus  GCGAG.76 1.72e-46 3.45e-46
```

```
## 8  Lachnospiraceae Clostridium_XlVa GCAAG.167 6.25e-43 1.25e-42
## 9  Lachnospiraceae        Roseburia  GCAAG.64 2.73e-36 5.46e-36
## 10            <NA>             <NA>   GCAAG.1 5.22e-35 1.04e-34
##    adj.significance
## 1               ***
## 2               ***
## 3               ***
## 4               ***
## 5               ***
## 6               ***
## 7               ***
## 8               ***
## 9               ***
## 10              ***
```

Il semble que les bactéries les plus fortement associées appartiennent toutes à la famille des Lachnospiracées , ce qui est cohérent avec les résultats aléatoires de la forêt.

Techniques multitables

Afin de quantifier la variation des mesures microbiennes, génomiques et métaboliques dans différentes conditions expérimentales, nous utilisons une analyse de corrélation canonique (sparse CCA). Un nouvel ensemble de données va être utilisé. Il y a deux tableaux ici, un pour les bactéries et un autre avec des métabolites. 12 échantillons ont été obtenus, chacun avec des mesures à des valeurs de 637 m / z et 20 609 OTU; cependant, environ 96% des entrées du tableau d'abondance microbienne sont nulles.Le code ci-dessous récupère et filtre ces données :

```
metab <- read.csv("https://raw.githubusercontent.com/spholmes/F1000_workflow/master/data/metabolites.csv
microbe_connect <-url("https://raw.githubusercontent.com/spholmes/F1000_workflow/master/data/microbe.rda
load(microbe_connect)
microbe
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 20609 taxa and 12 samples ]
## tax_table()   Taxonomy Table:    [ 20609 taxa by 6 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 20609 tips and 20607 internal nodes ]
```

On peut voir que l'obhet phyloseq représente le microbe.

Nous filtrons d'abord les microbes et les métabolites d'intérêt, en supprimant ceux qui sont nuls sur de nombreux échantillons, en éliminant ceux qui sont nuls dans de nombreux échantillons. Ensuite, nous les transformons pour diminuer les queues lourdes. Nous prenons également le logarithme des métabolites.

```
library("genefilter")
```

```
##
## Attaching package: 'genefilter'
```

```
## The following objects are masked from 'package:MatrixGenerics':
##
##     rowSds, rowVars
```

```
## The following objects are masked from 'package:matrixStats':
##
##     rowSds, rowVars
```

```
keep_ix <- rowSums(metab == 0) <= 3
metab <- metab[keep_ix, ]
microbe <- prune_taxa(taxa_sums(microbe) > 4, microbe)
microbe <- filter_taxa(microbe, filterfun(kOverA(3, 2)), TRUE)
metab <- log(1 + metab, base = 10)
X <- otu_table(microbe)
X[X > 50] <- 50
dim(X)
```

```
## [1] 174  12
```

```
dim(metab)
```

```
## [1] 405  12
```

Nous voyons que X et metab ont 12 colonnes, ce sont en fait les échantillons et nous les transposerons.

Ensuite, nous pouvons appliquer la méthode CCA qui permet de comparer des ensembles d'entités dans des tables de données de grande dimension, où il peut y avoir plus de caractéristiques mesurées que d'échantillons. Dans le processus, il y aura un choix d'un sous-ensemble de fonctionnalités disponibles permettant une capture de plus de covariance. Cela permet de refléter les signaux présents sur plusieurs colonnes.

Nous appliquons ensuite une PCA à ce sous-ensemble de caractéristiques sélectionné, autrement dit CCA clairsemée comme procédure de sélection, plutôt que comme méthode d'ordination.

Les paramètres penaltyxet penaltyz sont des pénalités de parcimonie. De plus petites valeurs de penaltyx se traduiront par moins de microbes sélectionnés. De la même manière penaltyz module le nombre de métabolites sélectionnés.

```
library(PMA)
cca_res <- CCA(t(X),  t(metab), penaltyx = .15, penaltyz = .15)
```

```
## 123456789101112131415
```

```
cca_res
```

```
## Call: CCA(x = t(X), z = t(metab), penaltyx = 0.15, penaltyz = 0.15)
##
##
## Num non-zeros u's:  5
## Num non-zeros v's:  15
## Type of x:  standard
## Type of z:  standard
## Penalty for x: L1 bound is  0.15
## Penalty for z: L1 bound is  0.15
## Cor(Xu,Zv):  0.974
```

Avec ces paramètres, 5 microbes et 15 métabolites ont été sélectionnés. De plus, ces 20 caractéristiques entraînent une corrélation de 0,974 entre les deux tableaux. Nous interprétons cela comme signifiant que les données microbiennes et métaboliques reflètent des signaux sous-jacents similaires, et que ces signaux peuvent être bien approchés par les 20 caractéristiques sélectionnées.

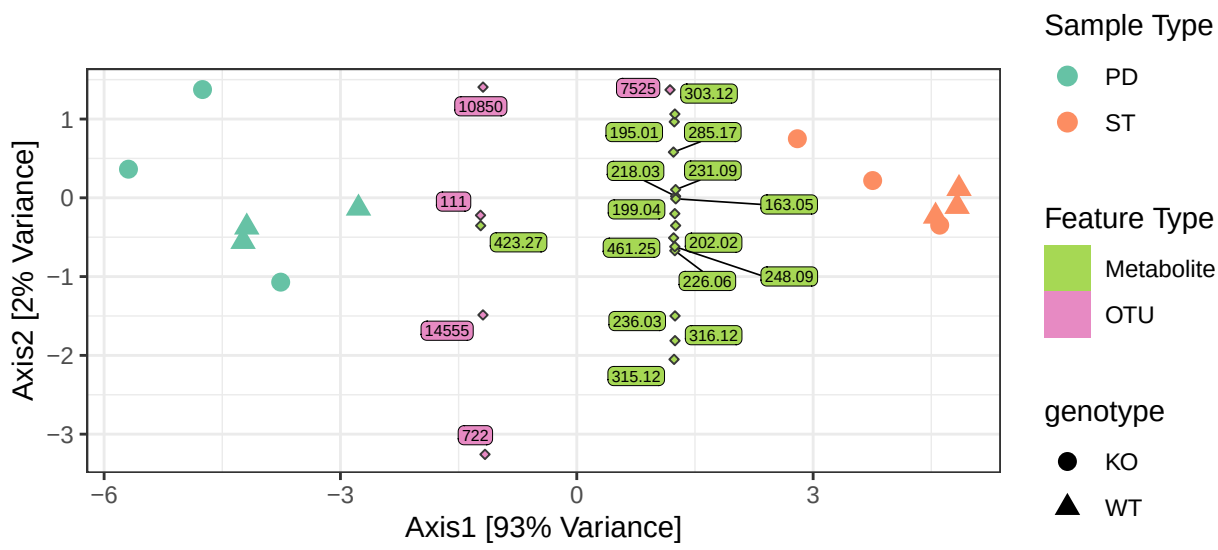
Cependant, il est possible que d'autres sous-ensembles de fonctionnalités puissent tout aussi bien expliquer les données. En effet, une CCA clairsemée a minimisé la redondance entre les fonctionnalités, mais ne garantit en aucun cas que ce sont les «vraies» fonctionnalités.

Pour relier les métabolites et les OTU récupérés à partir des caractéristiques des échantillons sur lesquels ils ont été mesurés, nous les utilisons comme données d'entrée dans une PCA ordinaire :

```
library("phyloseq")
library("ade4")
combined <- cbind(t(X[cca_res$u != 0, ]),
                  t(metab[cca_res$v != 0, ]))
pca_res <- dudi.pca(combined, scannf = F, nf = 3)
```

```
genotype <- substr(rownames(pca_res$li), 1, 2)
sample_type <- substr(rownames(pca_res$l1), 3, 4)
feature_type <- grepl("\\.", colnames(combined))
feature_type <- ifelse(feature_type, "Metabolite", "OTU")
sample_info <- data.frame(pca_res$li, genotype, sample_type)
feature_info <- data.frame(pca_res$c1,
                           feature = substr(colnames(combined), 1, 6))
```

```
library("ggplot2")
library("ggrepel")
ggplot() +  geom_point(data = sample_info,
            aes(x = Axis1, y = Axis2, col = sample_type, shape = genotype), size = 3) +
  geom_label_repel(data = feature_info,
                   aes(x = 5.5 * CS1, y = 5.5 * CS2, label = feature, fill = feature_type),
                   size = 2, segment.size = 0.3,
                   label.padding = unit(0.1, "lines"), label.size = 0) +
  geom_point(data = feature_info,
             aes(x = 5.5 * CS1, y = 5.5 * CS2, fill = feature_type),
             size = 1, shape = 23, col = "#383838") +
  scale_color_brewer(palette = "Set2") +
  scale_fill_manual(values = c("#a6d854", "#e78ac3")) +
  guides(fill = guide_legend(override.aes = list(shape = 32, size = 0))) +
  coord_fixed(sqrt(pca_res$eig[2] / pca_res$eig[2])) +
  labs(x = sprintf("Axis1 [%s%% Variance]",
                   100 * round(pca_res$eig[1] / sum(pca_res$eig), 2)),
       y = sprintf("Axis2 [%s%% Variance]",
                   100 * round(pca_res$eig[2] / sum(pca_res$eig), 2)),
       fill = "Feature Type", col = "Sample Type")
```



Ce graphique représente un triplot de la PCA produite à partir des caractéristiques sélectionnées par la CCA à partir de plusieurs types de données, les métabolites et les OTU. Cela permet une comparaison entre les échantillons mesurés; triangles pour Knockout et cercles pour le type sauvage. Cela caractérise également l'influence des différentes caractéristiques; des losanges avec des étiquettes de texte. Par exemple, nous voyons que la principale variation des données se situe entre les échantillons PD et ST, qui correspondent aux différents régimes. Ainsi, des grandes valeurs pour 15 des caractéristiques sont associées au statut ST, tandis que de petites valeurs pour 5 d'entre elles indiquent au statut PD, qui correspondent aux différents régimes alimentaires.

De plus, de grandes valeurs de 15 des fonctions sont associées au statut ST, tandis que de petites valeurs pour 5 d'entre elles indiquent le statut PD. L'avantage de la sélection clairsemée de la CCA est maintenant clair. Nous pouvons afficher la majeure partie de la variation entre les échantillons à l'aide d'un graphique relativement simple, et nous pouvons éviter de tracer les centaines de points supplémentaires qui seraient nécessaires pour afficher toutes les caractéristiques.

Conlusions

A travers ce tutoriel, nous avons pu voir comment un flux de travail complet dans R est réalisé pour débruiter, identifier et normaliser les lectures de séquençage d'amplicon de nouvelle génération à l'aide de modèles probabilistes avec des paramètres ajustés en utilisant les données disponibles. Différentes analyses ont ensuite été effectuées.