---
title: "ArticleFV_BSC"
output: github_document
---

1. Getting ready

```{r}
library(dada2) ; packageVersion("dada2")
```

```{r}
path <- "/home/rstudio/ADM_M1_2023/sequences"

list.files(path)
```

```{r}
fnFs <- sort(list.files(path,
                        pattern = "_1.fastq.gz",
                        full.names = TRUE))
fnFs #on a selectionner les donnee fasta _1. pour les forwards
```

La variable fnFs contient une liste ordonnée des chemins complets des fichiers qui se trouvent dans le répertoire spécifié (path) et dont le nom se termine par "_1.fastq.gz". Ces fichiers sont généralement associés aux lectures de séquence provenant de la direction avant (forward) d'une paire de lectures (paired-end reads) dans une expérience de séquençage.
Ici sont identifiés et triés les fichiers se terminant par "_1.fastq.gz" dans un répertoire spécifié, stockés les chemins complets de ces fichiers dans la variable fnFs, puis affiche cette liste.

```{r}
sample.names <- sapply(strsplit(basename(fnFs), "_"), `[`, 1)
```

Sample.names contient les identifiants d'échantillons extraits des noms de fichiers stockés dans la liste fnFs. Ces identifiants sont généralement utilisés pour étiqueter les échantillons.

```{r}
fnRs <- sort(list.files(path,
```

```
                pattern = "_2.fastq.gz",
                full.names = TRUE))
fnRs #idem mais avec les _2. pour les reverses
```
Ici il s'agit de la même chose qu'avant mais pour les séquences reverse.

2. Inspect read quality profiles
```{r, echo=FALSE}
if (!requireNamespace("BiocManager", quietly = TRUE))
   install.packages("BiocManager")

BiocManager::install("BiocParallel")

```

```{r}
plotQualityProfile(fnFs[1:2])
#en gris il s'agit d'une heatmap basé sur la frequence de chaque score de qualité pour les positions
#ligne verte = score de qualité moyen pour chaque position
#ligne rouge = echelle de proptortions des reads
```
Cela permet de visualiser la qualité des données de séquence, notamment la qualité des lectures dans des fichiers FASTQ (vu que vos fichiers se terminent par ".fastq.gz") ici des séquences forward.

Axes de la Figure :
Axe X (horizontal) : Représente la position dans la séquence. Chaque point sur cet axe correspond à la qualité des bases à cette position.
Axe Y (vertical) : Représente la qualité des bases à la position correspondante sur l'axe X. Plus la valeur est élevée, meilleure est la qualité de la base.

Courbes de Qualité :
Courbe Principale : La courbe principale représente la qualité moyenne des bases à chaque position. Elle donne une indication de la qualité globale des lectures sur toute la longueur de la séquence.
Bandelettes ou Ombres : Certains profils incluent des zones ombrées ou des bandelettes autour de la courbe principale. Elles peuvent représenter l'écart-type ou la variabilité de la qualité des bases à chaque position.

Lignes de Coupure :

Lignes de Coupure ou Seuils : Certains profils de qualité indiquent des lignes de coupure ou des seuils de qualité. Ces seuils définissent généralement des critères de qualité minimale acceptables pour différentes applications. Les bases en dessous de ces seuils peuvent être considérées comme de qualité inférieure.

Caractéristiques Spécifiques :
Pic Initial : Une bonne qualité initiale est généralement attendue au début de la séquence.
Dégradations : Des zones de dégradation de la qualité peuvent indiquer des problèmes potentiels avec la lecture ou la séquence.
Fin de Séquence : La qualité peut diminuer vers la fin de la séquence, mais cela doit rester au-dessus d'un seuil acceptable.

Interprétation Générale :
Une courbe de qualité uniformément élevée sur toute la longueur indique des lectures de haute qualité.
Des variations importantes dans la qualité le long de la séquence peuvent indiquer des problèmes potentiels, tels que des erreurs de séquençage, des contaminations, ou des régions difficiles à séquencer.

```{r}
plotQualityProfile(fnRs[1:2])

```
Ici, il semblerait que les 2 quality profil soient de plutôt bonne qualité en tout cas de la même qualité.

3. Filter and trim

```{r}
filtFs <- file.path(path, "filtered", paste0(sample.names, "_F_filt.fastq.gz"))
filtRs <- file.path(path, "filtered", paste0(sample.names, "_R_filt.fastq.gz"))
names(filtFs) <- sample.names
names(filtRs) <- sample.names
```

```{r}
getOption("timeout")
```

```{r}
out <- filterAndTrim(fnFs, filtFs, fnRs, filtRs, truncLen=c(240,160),
          maxN=0, maxEE=c(2,2), truncQ=2, rm.phix=TRUE,
```

```
        compress=TRUE, multithread=FALSE)
head(out)
```

fnFs et fnRs : Les fichiers d'entrée pour les lectures avant (forward) et arrière (reverse) avant le processus de filtrage et d'élagage.

filtFs et filtRs : Les fichiers de sortie après le processus de filtrage et d'élagage.

truncLen=c(240,160) : Spécifie les longueurs de troncature des lectures avant (forward) et arrière (reverse). Dans cet exemple, les lectures avant sont tronquées à 240 bases et les lectures arrière à 160 bases.

maxN=0 : Aucune base avec la lettre N (non déterminée) n'est autorisée dans les lectures résultantes.

maxEE=c(2,2) : Les lectures sont filtrées en fonction de la qualité attendue (EE). Les lectures avec une qualité attendue supérieure à 2 sont exclues. Cela peut être utilisé pour éliminer des lectures de qualité inférieure.

truncQ=2 : Les bases avec une qualité inférieure à 2 sont tronquées.

rm.phix=TRUE : Les séquences similaires à PhiX (un génome de contrôle couramment utilisé en séquençage) sont supprimées. Cela est souvent nécessaire pour éliminer les contaminations de PhiX.

compress=TRUE : Les fichiers de sortie sont compressés.

multithread=FALSE : L'utilisation du multithreading pour accélérer le processus est désactivée.

out : Le résultat de la fonction filterAndTrim est stocké dans la variable out. Cette fonction peut renvoyer plusieurs informations sur le processus de filtrage et d'élagage.

head(out) : Les premières lignes de la sortie out sont affichées. Cela peut inclure des informations sur les lectures avant et arrière, le nombre de lectures retenues après le filtrage, etc.

Analyse du résulat :

reads.in : Le nombre initial de lectures avant tout processus de filtrage.

reads.out : Le nombre de lectures restantes après le processus de filtrage et d'élagage.

Le tableau indique ces deux valeurs pour chaque échantillon. Prenons l'exemple de la première ligne :
SRR6222309_1.fastq.gz : Nom de l'échantillon.
reads.in : 1065360 - Le nombre initial de lectures avant le filtrage.
reads.out : 670053 - Le nombre de lectures restantes après le filtrage.

Cela signifie que, pour l'échantillon SRR6222309_1.fastq.gz, après le processus de filtrage et d'élagage, 670,053 lectures ont été conservées sur un total initial de 1,065,360.

Une réduction du nombre de lectures après le filtrage peut être attendue, car certaines lectures de qualité inférieure, contaminées ou ne répondant pas aux critères spécifiés dans le code ont été éliminées. Cela peut être une étape importante pour garantir la qualité des données dans une analyse ultérieure.

4. Learn the Error Rates
```{r}
errF <- learnErrors(filtFs, multithread=TRUE)
```
étape d'apprentissage des erreurs à partir des lectures avant (forward) filtrées. Apprendre les erreurs est une étape importante dans le traitement des données de séquençage pour comprendre les caractéristiques de séquençage spécifiques à vos données, et cela peut être utilisé dans d'autres étapes de l'analyse pour corriger ou prendre en compte ces erreurs.

```{r}
errR <- learnErrors(filtRs, multithread=TRUE)
```
idem mais avec les reverse

```{r}
plotErrors(errF, nominalQ=TRUE)
```
Ces graphiques sont utiles pour comprendre les caractéristiques spécifiques des erreurs dans nos données de séquençage, ce qui peut être important pour les étapes ultérieures de l'analyse, telles que la détection et la correction d'erreurs.

Ligne ou Courbe de Qualité Nominale : la qualité nominale des bases telles qu'assignées par le séquenceur.

Ligne ou Courbe d'Erreur Estimée : la probabilité estimée d'erreur pour chaque position dans la séquence.

Distribution d'Erreurs : voir comment la probabilité d'erreur varie le long des lectures de séquence.

5. Sample Inference
```{r}
dadaFs <- dada(filtFs, err=errF, multithread=TRUE)
```

effectuer le traitement de données de séquençage avec la méthode DADA2

En DADA2, le processus de traitement des données, généralement effectué par la fonction dada, comprend des étapes telles que :

Correction des Erreurs : Utilisation du modèle d'erreurs (errF) pour corriger les erreurs de séquençage présentes dans les lectures.

Débruitage (Denoising) : Identification et suppression des erreurs et bruits de séquençage pour obtenir des séquences débruitées.

Inférence des Amplicons : Inférence des amplicons, qui sont les séquences biologiques réelles après élimination des erreurs.

Estimation des Abondances : Estimation des abondances relatives des différentes séquences.

```{r}
dadaRs <- dada(filtRs, err=errR, multithread=TRUE)
```
idem mais pour les reverse

```{r}
dadaFs[[1]]
```

ces résultats indiquent que, à partir des séquences d'entrée, DADA2 a identifié et débruité 4 255 variants de séquence en utilisant les paramètres spécifiés. Ces variants de séquence sont

essentiellement les séquences biologiques réelles, nettoyées des erreurs de séquençage. Ces résultats peuvent être utilisés pour des analyses ultérieures, telles que la création de tableaux de comptage, l'analyse de diversité, etc

6. Merge paired reads

```{r}
mergers <- mergePairs(dadaFs, filtFs, dadaRs, filtRs, verbose=TRUE)
```

fonction mergePairs pour fusionner les lectures appariées après le processus de débruitage

7872 paired-reads (in 1267 unique pairings) successfully merged out of 650967 (in 44929 pairings) input.

7872 lectures appariées ont été fusionnées avec succès.

Ces lectures appariées provenaient initialement de 650967 paires d'entrée.

Il y a eu 1267 paires uniques résultantes après la fusion.

47024 paired-reads (in 1683 unique pairings) successfully merged out of 862912 (in 31022 pairings) input.

47024 lectures appariées ont été fusionnées avec succès.

Ces lectures appariées provenaient initialement de 862912 paires d'entrée.

Il y a eu 1683 paires uniques résultantes après la fusion.

... et ainsi de suite pour chaque ligne.

la fusion réussie de lectures appariées est une étape importante dans le traitement des données de séquençage, en particulier dans les technologies de séquençage de nouvelle génération où les lectures sont souvent générées en paire (forward et reverse). La fusion des lectures appariées permet de prolonger la longueur des séquences et d'améliorer la qualité des données pour certaines analyses

```{r}
head(mergers[[1]])
```
utilisé pour afficher les premières lignes des résultats de fusion pour la première paire d'échantillons traitée

représente les identifiants ou les positions des séquences dans le résultat de la fusion des lectures appariées (mergers)

7. Construct sequence table

```{r}
seqtab <- makeSequenceTable(mergers)
dim(seqtab)
```
(15) représente le nombre d'échantillons ou de groupes dans lesquels les séquences ont été regroupées

(13127) représente le nombre de séquences uniques identifiées dans l'ensemble des échantillons

```{r}
table(nchar(getSequences(seqtab)))
```

Il y a 45 séquences ayant une longueur de 240 caractères.
Il y a 16 séquences ayant une longueur de 241 caractères.
Il y a 23 séquences ayant une longueur de 242 caractères.
Et ainsi de suite...

utile pour comprendre la variabilité de longueur dans vos données

8. Remove chimeras

```{r}
seqtab.nochim <- removeBimeraDenovo(seqtab, method="consensus", multithread=TRUE, verbose=TRUE)
dim(seqtab.nochim)
```
La suppression des chimères est une étape importante dans le traitement des données de séquençage pour s'assurer que les séquences utilisées dans les analyses ultérieures ne sont pas artificiellement créées lors du processus de séquençage. La méthode de consensus utilisée dans ce cas s'appuie probablement sur le fait qu'une chimère est moins probable d'être présente dans plusieurs échantillons, et donc elle est éliminée si elle n'est pas consensuelle dans l'ensemble des échantillons.

Identified 5963 bimeras out of 13127 input sequences. : Cela signifie que la fonction a identifié 5963 chimères dans les 13127 séquences d'entrée.

5 7164 indique qu'il y a maintenant 15 échantillons (ou groupes) avec 7164 séquences uniques non-chimériques

```{r}
sum(seqtab.nochim)/sum(seqtab)
```

0.7793849 indique que environ 77.94% des séquences dans la table seqtab sont considérées comme non-chimériques

9. Track reads through the pipeline

```{r}
getN <- function(x) sum(getUniques(x))
track <- cbind(out, sapply(dadaFs, getN), sapply(dadaRs, getN), sapply(mergers, getN), rowSums(seqtab.nochim))
```
pour calculer le nombre total de séquences uniques dans différentes étapes du processus d'analyse des données de séquençage

track rassemble des informations sur les résultats de l'analyse à différentes étapes, en incluant le nombre total de séquences uniques dans chaque échantillon pour les lectures avant, les lectures arrière, les lectures fusionnées et les séquences non-chimériques

```{r}
colnames(track) <- c("input", "filtered", "denoisedF", "denoisedR", "merged", "nonchim")
rownames(track) <- sample.names
head(track)
```

track qui suit l'évolution des données à travers différentes étapes du processus d'analyse

permet de visualiser comment le nombre de séquences évolue à travers différentes étapes du pipeline d'analyse pour chaque échantillon

Chaque ligne représente un échantillon avec le nombre de séquences à différentes étapes du processus

permet de suivre comment le nombre de séquences change au fur et à mesure que vous effectuez différentes étapes d'analyse

##############ERREUR#########
10. Assign taxonomy

```{r}
taxa <- assignTaxonomy(seqtab.nochim,
"~/home/rstudio/ADM_M1_2023/silva_nr_v132_train_set.fa.gz", multithread=TRUE)
```
```{r}
if (!require("BiocManager", quietly = TRUE))
   install.packages("BiocManager")

BiocManager::install("ShortRead")
```

```{r}
# Load the Bioconductor Biostrings package
if (!requireNamespace("Bioconductor", quietly = TRUE))
   install.packages("BiocManager")
BiocManager::install("Biostrings")
library(Biostrings)

# Specify the file path
file_path <- "silva_nr_v132_train_set.fa.gz"

# Read the FASTA file using readDNAStringSet
donnees_fasta <- readDNAStringSet(file_path)
```

```{r}
donnees_fasta <- readFasta("silva_nr_v132_train_set.fa.gz")
```

```{r}
taxa.print <- taxa # Removing sequence rownames for display only
rownames(taxa.print) <- NULL
head(taxa.print)
```

11. Evaluate accuracy

```{r}
unqs.mock <- seqtab.nochim["Mock",]
unqs.mock <- sort(unqs.mock[unqs.mock>0], decreasing=TRUE) # Drop ASVs absent in the Mock
cat("DADA2 inferred", length(unqs.mock), "sample sequences present in the Mock community.\n")


mock.ref <- getSequences(file.path(path, "HMP_MOCK.v35.fasta"))
match.ref <- sum(sapply(names(unqs.mock), function(x) any(grepl(x, mock.ref))))
cat("Of those,", sum(match.ref), "were exact matches to the expected reference sequences.\n")
```

12. Bonus: Handoff to phyloseq

```{r}
library(phyloseq); packageVersion("phyloseq")

library(Biostrings); packageVersion("Biostrings")

library(ggplot2); packageVersion("ggplot2")

theme_set(theme_bw())
```

```{r}
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
```

```{r}
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
```

```{r}
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
```

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

```{r}
ps.prop <- transform_sample_counts(ps, function(otu) otu/sum(otu))
ord.nmds.bray <- ordinate(ps.prop, method="NMDS", distance="bray")
```

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

```{r}
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")
```