Búsqueda de genes de interés en datos de metagenómica shotgun

1 Datos de entrada

Los datos a trabajar en esta unidad corresponden a **4 metagenomas marinos** colectados desde la **zona mínima de oxígeno OMZ (*Oxygen Minimum Zone*)** que se encuentra en las costas de Perú y Chile. Los metagenomas fueron colectados en dos profundidades (100 y 150 metros) en una región de la OMZ frente a Valparaíso, Chile.

- Descripción de las muestras:

Muestra	Profundidad [m]	Fecha de muestreo
EC5	100	Sep-15
EC11	150	Feb-18
EC100	100	Jan-15
EC19	150	Jul-18

Los datos de entrada provienen de la secuenciación de tipo *shotgun* del ADN total extraído de las muestras de agua.

1.1 Pre-procesamiento de los datos

El primer paso en la busqueda de genes de interés es el ensamble de los *reads* en *contigs* mas largos. Eso permite trabajar con sequencias más largas, en lugar de las *reads* cortas que hacen los procesos de computación mas intensivos.

Para ensemblar las *reads*, necesitamos primero controlar la calidad, cortar los adaptadores y remover los reads de baja calidad con el programa *Prinseq*.

Elegimos los parámetros basados en el largo y calidad de las *reads* y las necesidades del análisis. Definimos un largo de *read* mínimo de 70 bases (`-min_len 70`), una calidad promedio mínima de 20 (`-min_qual_mean 20`), y una

calidad media de 10 sobre una *sliding windows* desde el 3' (`-trim_qual_window 10`), con un paso de 5 posiciones (`-trim_qual_step 5`). Filtramos las *reads* con bases indeterminadas (Ns) (`-ns_max_p 0`). También, removemos las sequencias de baja complejidad (`-lc_threshold 32`). Finalmente, removemos 10 nucleotidos a la izquierda y 20 a la derecha.

```
for i in $list_sample
do
prinseq-lite.pl -verbose -fastq $rawdir/${i}_1.fastq -fastq2
$rawdir/${i}_2.fastq -min_len 70 -min_qual_mean 20 -ns_max_p 0 -lc_method dust
-lc_threshold 32 -trim_left 10 -trim_right 10 -trim_qual_right 20 -
trim_qual_window 10 -trim_qual_step 5 -out_format 3
done
```

Despues de eso, las *reads* limpias son ensambladas usando *MetaSPAdes* (versión v3.12.0). Las opciones `-t` and `-m` permiten especificar el número de cpus y memoria RAM disponible. La opción `-k` permite especificar el largo de los k-mer por los cuales pasan las iteraciones. La elección del largo de k-mer es basada en el largo de las *reads* y la sensibilidad necesaria, lo que también puede influir en el tiempo de computación. Hacer iteraciones pasando por k-mer cortos hasta k-mer largos es una buena manera de mejorar el ensamble cuando la capacidad computacional lo permite.

```
metaspades.py -1 $filtereddir/concat_samples_1.fastq -2
$filtereddir/concat_samples_2.fastq -s
$filtereddir/concat_samples_singletons.fastq -t 80 -m 1000  -k
21,33,55,77,99,127 -o $assemblydir/concat_samples
```

Finalmente, y teniendo los metagenomas ensamblados, usamos *Prokka* para anotar cada metagenoma y predecir regiones codificantes (CDS), lo cual nos entrega un set de sequencias proteicas predichas para cada muestra.

1.2 Preparar la base de datos

Para la construccion de la base de datos de proteínas de interés, se utilizó una lista de las proteínas más relevantes en los procesos del ciclo del nitrogeno conducidos por las comunidades microbianas marinas.

Usando como referencia el articulo Nitrogen Cycle of the Open Ocean: From Genes to Ecosystems, se generó una lista de proteínas para ser buscadas en la base de datos Identical protein groups `nifH, nifD, nifK, amoC, amoA, amoB, hao, norA, norB, narD, narG, narH, narI, narJ, napA, napB, napD, napE, nirB, nirC, nirK, nirU, nirN, nirO, nirS, nosZ, nasA, nasB, nasC, nasD, narB, nrfA, nrfB, nrfC, nrfD, nrfE, gdhA, gltB`

Por cada uno de los genes, se descargaron todas las proteínas completas correspondientes (no se incluyen secuencias parciales), que se concatenaron en un FASTA común. Luego, usando *MMseq2*, se genera una base de datos local con el comando `mmseqs createdb`, la cual se agrupa por similitud de secuencias con el comando `mmseqs cluster` con las opciones `-e 1 --max-seqs 1000 -c 0.0001 --min-seq-id 0.0001 --cluster-mode 1 --kmer-per-seq 1 --alignment-mode 3`, para luego obtener la secuencias representativas de cada grupo con `mmseqs createseqfiledb` y `mmseqs result2repseq`, estas secuencias

representativas reemplazarán la base de datos local de mmseq2 para el resto de los analisis.
Para cada archivo FASTA de proteínas predichas de los 4 metagenomas, se compara contra la base de datos local utilizando mmseq2, obteniendo un resultado tabulado de cada proteína que tiene algún homólogo en la base de datos local.

1.3 Identificar proteinas de interés en los metagenomas

Finalmente, comparamos cada archivo conteniendo las proteínas predichas por muestra con la base de datos usando *mmseq2*, creando una tabla de cada proteína teniendo homólogo en la base de datos local. Este archivo conteniendo los identificadores de las proteínas será usado para obtener sequencias nucleotídicas, desde las cuales calcularemos la abundancia de cada gen.

1.4 Preparar FASTA con los genes presentes en los metagenomas

Una vez encontrado cuál gen ortólogo esta presente en los metagenomas, queremos encontrar las sequencias nucleotídicas correspondientes, para poder mapear las *reads*. Por eso, compilamos todos los identificadores de proteínas en un archivo y, uno por uno, buscamos sequencias nucleotídicas correspondientes usando `tblastn`. En caso de que no se encuentre una sequencia con identidad sobre 70% o cobertura sobre 80%, la proteína se descarta de la lista.
En paralelo, preparamos una tabla con el nombre de los genes correspondientes a cada identificador de los genes ortólogos y con eso, podemos luego traducir la abundancia de cada variante en abundancia de cada gen de interés, y tambien visualizar la variabilidad existente en las muestras por cada uno de los genes de interés.

1.5 Mapeo de *reads*

Usamos *Bowtie2* para mapear las *reads* sobre los genes contenidos en el archivo FASTA. Dado que no todas las sequencias nucleotídicas alcanzaron a tener 100% de identidad con las proteínas identificadas, tenemos que relajar los parametros de alineamiento, permitiendo mas *mismatch* y alineamientos incompletos.

Permitamos un *mismatch* (`-N 1`) y usamos alineamiento local (`--local`).

```
for i in $samples
do
```

```
bowtie2 -x $fasta_dir/goi -1 $read_dir/${i}_OK_1.fastq -2
$read_dir/${i}_OK_2.fastq -U $read_dir/${i}_OK_singletons.fastq -N 1 --local
|  samtools view -bh | samtools sort -o $bam_dir/${i}.bam
done
```

Después de que obtenemos las *reads* mapeadas, usamos la funcion `idxstats` de Samtools para obtener el valor de cuentas *reads* mapeadas en cada gen.

```
for i in $samples
do
samtools index $bam_dir/${i}.bam
samtools idxstats $bam_dir/${i}.bam > $bam_dir/${i}.idxstats
done
```

Ahora que estos archivos han sido creados, podemos comenzar con los análisis en R.

2 Configurar sesión de R

Primero, cargamos los paquetes de R necesarios.

Los siguientes 3 *scripts* te mostrarán una manera eficiente de instalar y cargar la lista de paquetes según su repositorio de origen, ya que cada repositorio tiene su propia función para instalar sus paquetes.

- Primero, enlistar los paquetes necesarios en diferentes vectores dependiendo de su repositorio de origen.

```
# Definir paquetes
# Repositorio CRAN
cran_packages <- c("ggplot2", "gplots", "reshape2")
# Repositorio Bioconductor
bioc_packages <- "DESeq2"
```

- Segundo, instalar los paquetes definidos arriba usando la función correspondiente a cada repositorio.

```
# Instalar paquetes CRAN
.inst <- cran_packages %in% installed.packages()
if(any(!.inst)) {
  install.packages(cran_packages[!.inst])
}
# Intalar paquetes BioConductor
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
.inst <- bioc_packages %in% installed.packages()
if(any(!.inst)) {
  BiocManager::install(bioc_packages[!.inst])
}
```

- Tercero, cargar los paquetes requeridos a la sesión actual de R.

```
# Cargar paquetes
```

```
sapply(c(cran_packages, bioc_packages), require, character.only = TRUE)
##  ggplot2   gplots reshape2   DESeq2
##     TRUE     TRUE     TRUE     TRUE
```

El paso de instalación de paquetes es necesario solamente una vez. Excepto si se quiere actualizar la versión del paquete, o bien, R ha sido desinstalado e instalado nuevamente o actualizado su versión.

Si ya tienes los paquetes instalados en tu computadora, sólo necesitas enlistar (*1er script*) y cargar (*3er script*) los paquetes. También, puedes cargarlos a la sesión actual de R usando la función `library()`, así:

```
# Cargar paquetes
library(ggplot2)
library(gplots)
library(reshape2)
library(DESeq2)
```

3 Análisis de abundancia de genes de interés

Primero, **descarga y descomprime el directorio de los datos *input* AQUÍ**.

- Define la ruta de directorios en tu computadora hasta el directorio *input* `gene_search_input_files/`:

```
base_path <- "/path/to/gene_search_input_files/"
base_path <- "./"
```

- Ahora, definimos la ruta a cada directorio que contiene datos *input*:

```
reads_directory <- paste(base_path , "reads/", sep = "")
fasta_directory <- paste(base_path , "fasta/", sep = "")
bam_directory <- paste(base_path , "bam/", sep = "")
```

3.1 Obtener la metadata

- Leemos la tabla que contiene los datos asociados a cada muestra (*metadata*) en el objeto llamado `metadata`.

```
path <- paste(base_path, "metadata.csv", sep = "")
metadata <- read.table(path, sep = "\t", header=T)
metadata
##   Sample Depth_m Sampling_date
## 1    EC5     100        Sep-15
## 2   EC11     150        Feb-18
## 3  EC100     100        Jan-15
## 4   EC19     150        Jul-18
```

3.2 Preparar tabla de cuentas

- Comenzamos enlistando los archivos `.idxstats` presentes en el directorio. Debería haber un archivo por muestra.

```
stat_list <- list.files(bam_directory, pattern = "\\.idxstats$")
stat_list
## [1] "EC100.idxstats" "EC11.idxstats"  "EC19.idxstats"  "EC5.idxstats"
```

Después, vamos a cargar estos archivos, uno a la vez, y construiremos la tabla de cuentas.

Los archivos `.idxstats` contienen el identificador del gen, seguido por el largo del gen, el número de *reads* mapeados y número de *reads* no mapeados. Empezamos con leer la tabla y añadir nombre a las columnas. Dado que el tamaño del gen varía, necesitamos normalizar los valores de cuentas por el largo del gen, dado que genes mas largos permiten mapear más *reads* que genes más cortos con la misma abundancia.

Una vez normalizados los valores de abundancia, necesitamos añadir este valor a una tabla general. Multiplicamos el valor normalizado por 1000000 para tener valores en un rango mas manejable.

Una de las líneas de los archivos `.idxstat` comienza con `*` y esta usada para añadir información que no pertenece a la tabla, por lo mismo, la removemos.

```
for (i in c(1:length(stat_list)))
{
sample_stat <- read.table(paste(bam_directory,stat_list[i], sep=""), header =
F)
colnames(sample_stat) <- c("id","length","mapped","unmapped")
sample_stat <- sample_stat[! sample_stat$id == "*",]
sample_stat$count <- sample_stat$mapped / sample_stat$length *1000000
colnames(sample_stat)[5] <- strsplit(stat_list[i], "\\.")[[1]][1]
if (i == 1)
 {
  table_stat <- sample_stat[,c(1,5)]
 }
else
 {
  table_stat <- merge(table_stat, sample_stat[,c(1,5)])
 }
}
```

Ahora, hemos normalizado las cuentas de las *reads* por el largo del gen, pero no hemos tomado en cuenta el tamaño de la librería (*número de secuencias por muestra*). Una mayor profundidad de secuenciación implica un mayor número de *reads* mapeando en contra de genes de interés, incrementando su abundancia de forma artificial.

- Para normalizar por el tamaño de la librería, usamos la función `echo bash` para calcular el número de *reads* totales por muestras: número de líneas del archivo FASTQ dividido por 4 = número de *reads*.

```
samples <- vapply(strsplit(stat_list, ".", fixed = TRUE), "[", "", 1)

lib_size <- vector()

for (i in c(1:length(samples)))
{
echo_command <- paste("echo $(cat ", reads_directory, samples[i],
"_OK_1.fastq|wc -l)/4|bc", sep="")
lib_size <- c(lib_size,system(echo_command, intern = TRUE))
}

lib_size <- as.numeric(lib_size)
```

Ahora que tenemos el número de *reads* por muestra, vamos a crear la tabla de cuentas normalizada. Después, nuevamente, multiplicamos el valor normalizado por 100000 para tener valores manejables.

```
table_normalized <- table_stat

for (i in c(1:4))
{
n = i+1
table_normalized[,n] <- table_stat[,n]/lib_size[i]*100000
}
```

- Ahora que tenemos los valores de abundancia normalizados por tamaño de librería, necesitamos generar la tabla de correspondencia entre el ID del gen y el nombre del gen de interés. Debido a que algunos genes no presentan match en los metagenomas, removemos aquellas filas sin ID correspondiente.

```
path <- paste(fasta_directory, "gene_table.tsv", sep="")
gene_names <- read.table(path, header = F)

gene_names <- gene_names[! is.na(gene_names$V2),]
```

La tabla contiene los nombres de los genes en la primera columna, y en la segunda columna, el ID de los genes separados por `,`. Desde esto nos interesa extraer la siguiente información: una tabla de correspondencia uno a uno entre el nombre de los genes y el ID de los genes, y una tabla con el número de variantes por gen.
Vamos a crear la estructura para ambas tablas, para después leer la tabla original línea por línea e ir rellenando las tablas nuevas.

```
id_gene <- as.data.frame(matrix(ncol = 2, nrow=length(table_normalized$id) ,
NA))
colnames(id_gene) <- c("id","gene")
gene_versions <- as.data.frame(matrix(ncol = 2, nrow=length(gene_names$V1) ,
NA))
colnames(gene_versions) <- c("gene","version_nb")
gene_names$V2 <- as.character(gene_names$V2)
gene_names$V1 <- as.character(gene_names$V1)
n <- 0
```

```
for (i in c(1:length(gene_names$V1)))
{
 gene <- gene_names$V1[i]
 id_vect <- as.vector(unlist(strsplit(gene_names$V2[i], split=",") ) )
 if(length(id_vect) == 1)
 {
   n = n+1
   id_gene$id[n] <- gene_names$V2[i]
   id_gene$gene[n] <- gene_names$V1[i]
   gene_versions$gene[i] <- gene_names$V1[i]
   gene_versions$version_nb[i] <- 1
 } else
 {
   gene_versions$gene[i] <- gene_names$V1[i]
   gene_versions$version_nb[i] <- length(id_vect)
   for (j in c(1:length(id_vect)))
   {
    n = n+1
    id_gene$id[n] <- id_vect[j]
    id_gene$gene[n] <- gene_names$V1[i]
   }
 }
}
```

3.3 Abundancia de genes por muestra

- Comenzaremos por la visualización de las *read count* para los diferentes genes y versiones. Para ello, agregamos el nombre de los genes a la tabla que contiene los valores de cuentas por ID de los genes.

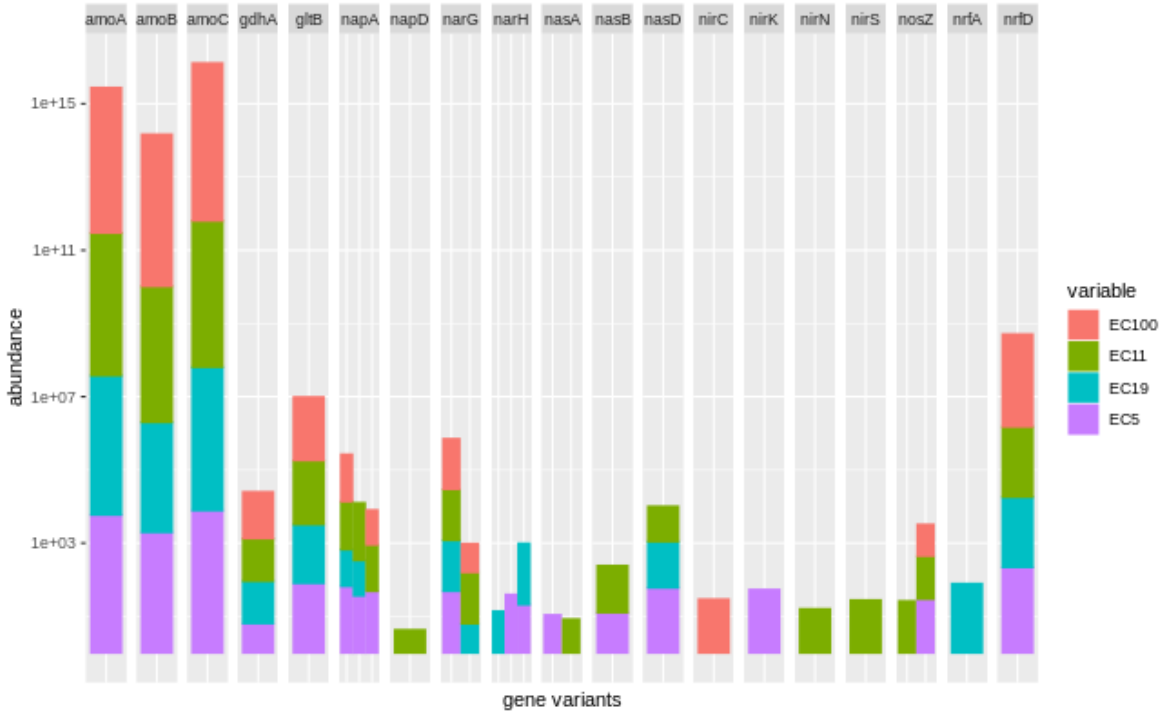
```
table_all <- merge(table_normalized, id_gene, by.all=id)
```

- Generamos un gráfico de barras (*barplot*) para visualizar la abundancia de los genes en cada muestra. Para tomar en cuenta las variantes del gen, vamos a graficar una barra por cada variante, y luego, agrupamos por gen. Para ello, debemos preparar la tabla de datos para que tenga todos los valores en una misma columna, y otra información en columnas separadas (una para el nombre de las muestras, otra para el nombre de los genes, y otra para el ID de la variante). Hacemos esto usando la función `melt` para colapsar las columnas que contienen el valor de las cuentas, conserbando la columna con el ID como referencia, y luego, agregar el nombre del gen.

```
tmp_table <- melt(data = table_all[,c(1:5)], id.vars = "id")
plot_table <- merge(tmp_table,  id_gene, by.all=id)
```

- Entonces, usamos ggplot2 para graficar: graficar los valores por ID, apilar valores para cada muestra y agruparlos por nombre del gen. También, utilizamos una escala logarítmica para poder apreciar la variación de genes con menor abundancia.

```
ggplot(plot_table[which(plot_table$value > 0),] , aes(x = id,y = value)) +
  geom_bar(aes(color=variable, fill=variable), stat="identity",
position="stack")  +
  facet_grid(~gene, scales="free_x" ) +
  scale_y_log10() +
  theme( axis.text.x=element_blank(), axis.ticks.x=element_blank()) +
  labs(x = "gene variants", y="abundance")
```



Basándonos en el gráfico, vemos que varios genes tales como *amoA*, *amoB*, y *amoC* parecen estar presentes en todas las muestras en abundancia similar. Mientras que otros genes tales como *nirC*, *nirK* y *nirN* parecen ser específicos para algunas muestras. Los genes con varias variantes, tales como *napA* y *narG*, muestran que aunque algunas variantes están igualmente presentes en todas las muestras, otras son específicas para una o más muestras. Esto es interesante, por ejemplo en el caso de *napA*, las muestras EC19 y EC100 tienen una abundancia total similar de este gen, pero provienen de versiones diferentes del gen.

Podemos ver que muestras, tales como EC11, tienen muchos de los genes de interés mientras que muestras, tales como EC19 tienen un número menor de estos genes.

Ahora bien, si miramos la metadata notamos que todas las muestras fueron colectadas en diferentes tiempos lo que podría explicar la alta especificidad por muestra de los datos.

3.4 Test estadístico de abundancia diferencial

Para ver si la abundancia de algunos genes es conducida por la profundidad de las muestras, usamos el paquete *DESeq2*. DESeq2 evalúa si las diferencias en abundancia de un gen entre dos condiciones es significativa o no. Aquí, vamos a definir la profundidad como condición para evaluar la abundancia de los genes de interés.

DESeq2 toma como *input* las *read count* y realiza el proceso de normalización. Sin embargo, asume que el total de *read count* es proporcional al tamaño de la librería, lo que no es necesariamente verdadero en este caso en el que estamos analizando un grupo pequeño de genes seleccionados. Por lo tanto, usaremos los valores de abundancia que previamente normalizamos por tamaño de librería como *input* para DESeq2.

- Para correr DESeq2, necesitamos: una tabla de cuentas por cada gen y por cada muestra (aquí `table_normalized`) con los ID de los genes como *row names*, y la metadata con la profundidad definida como variable categórica (`factor`).
- Debido a que DESeq2 espera una tabla de cuentas como *input* (*integer values*), es necesario redondear los valores en la tabla `table_normalized` para que sean números enteros.

```
## [1] "100" "150" "150" "100"
## converting counts to integer mode
## estimating size factors
## estimating dispersions
## gene-wise dispersion estimates
## mean-dispersion relationship
## final dispersion estimates
## fitting model and testing
## [1] "Intercept"          "Depth_m_150_vs_100"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2120  0.6295  0.6364  0.6761  0.7366  0.8168
```

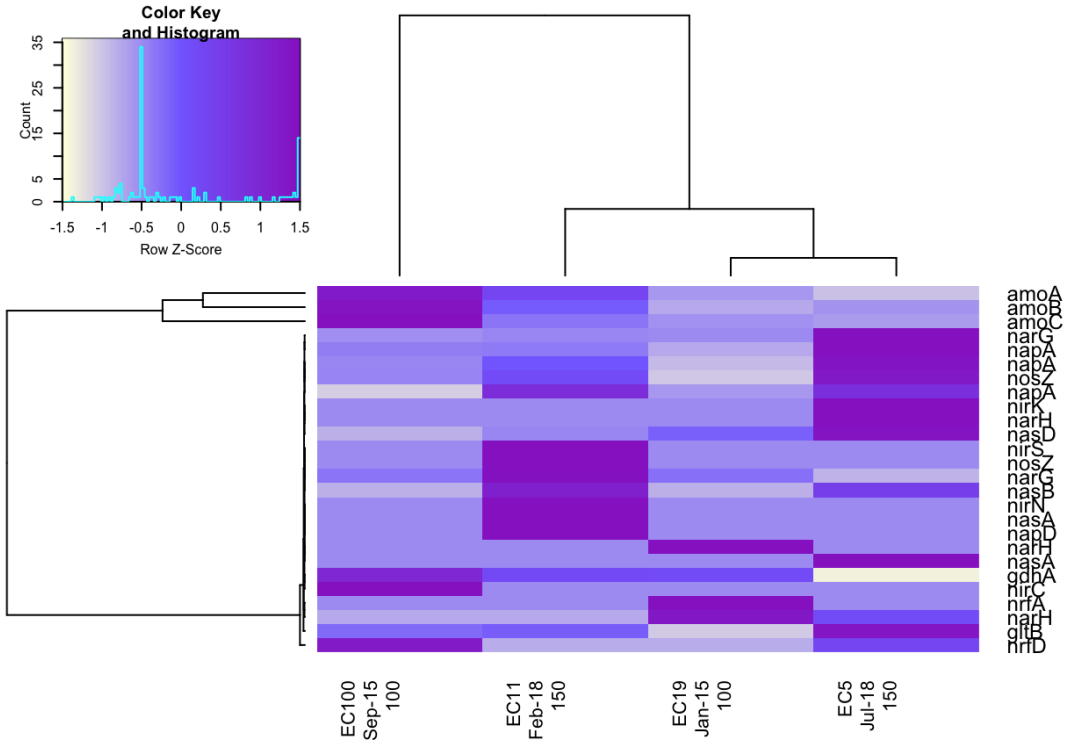
Podemos ver en los valores de p-values ajustados que no hay valores menores a 0.2, indicando que ninguna de las diferencias entre muestras de diferente profundidad es significativa. Esto se puede deber a varios factores, podría significar que las diferencias en profundidad comprendida en las muestras no tienen influencia en el metabolismo de el nitrógeno en la OMZ, pero hay muchos otros factores a considerar. Primero, el número de muestras es muy bajo y un mayor número de muesras y réplicas conducen a identificar diferencias con mayor precisión. Otro factor a considerar es que todas las muestras fueron colectadas en fechas diferentes, lo que significa que en el caso de que el metabolismo del nitrógeno varíe por profundidad y por fecha, las diferencias por fecha podrían estar enmascarando cualquier diferencia debido a la profundidad.

3.5 Evaluación cualitativa de abundancia de genes

A pesar de que las muestras no nos entregan suficiente información para las pruebas estadísticas, podemos hacer una evaluación cualitativa de la abundancia de genes por muestra. Para ello, podemos graficar un *heatmap* junto con un *hierarchical clustering* de los genes, de acuerdo a sus valores de abundancia a través de las muestras, y de las muestras, de acuerdo a sus valores de abundancia de los genes de interés.

Esto nos permitirá ver qué muestras son similares y qué genes se encuentran generalmente juntos.

```
my_palette <- colorRampPalette(c("lightyellow", "lightslateblue",
"darkorchid"))(104)


x_labels <- paste(colnames(table_all[,c(2:5)]), metadata$Sampling_date,
metadata$Depth_m, sep="\n")

heatmap.2(as.matrix(table_all[,c(2:5)]), scale="row", labRow = table_all$gene,
labCol = x_labels , cexCol=0.8 , col=my_palette, trace="none", margins = c(7,
7), key.par = list(cex=0.5),  lhei=c(2,4), lwid=c(1.5,4), keysize=0.75)
```



4 Conclusiones acerca del metabolismo del nitrógeno en OMZ

Vamos a darle un vistazo a la siguiente tabla tomada del artículo Zehr and Kudela, https://www.annualreviews.org/doi/10.1146/annurev-marine-120709-142819 que nos indica en que proceso participa cada gen.

Table 2 Major oceanic nitrogen cycling pathways and relevant genes

Reaction name	Chemical reaction	Genes
Nitrogen fixation	$N_2 + 8H^+ + 8e^- + 16\ ATP \rightarrow 2NH_3 + H_2 + 16ADP + 16\ P_i$	*nifH*, *nifD*, *nifK* in alternative nitrogenases (those that use Fe or V in place of Mo in component I); there is also a *nifG* gene (between *nifD* and *nifK*)
Ammonium oxidation	$NH_3 + O_2 + 2\ H^+ + 2e^- \rightarrow NH_2OH + H_2O$ $NH_2OH + H_2O \rightarrow HNO_2 + 4H^+ + 4e^-$ $0.5O_2 + 2H^+ + 2e^- \rightarrow H_2O$	*amoC*, *amoA*, *amoB*, *hao*
Nitrite oxidation	$2NO_2^- + H_2O \rightarrow NO_3^- + 2H^+ + 2e^-$ $2H^+ + 2e^- + 0.5O_2 \rightarrow H_2O$	*norA*, *norB*
Heterotrophic nitrification	R-NH2 $\rightarrow NO_2$ R-NH2 $\rightarrow NO_3$	The genes are not well known but may be the nitrate reductase genes involved in heterotrophic denitrification *narH*, *narJ*
Anaerobic ammonia oxidation	$HNO_2 + 4H^+ \rightarrow NH_2OH + H_2O$ $NH_2OH + NH_3 \rightarrow N_2H_4 + H_2O$ $N_2H_4 \rightarrow N_2 + 4H^+$ $>HNO_2 + NH_3 \rightarrow N_2 + 2H_2O$ $>HNO_2 + H_2O + NAD \rightarrow HNO_3 + NADH_2$	Over 200 genes involved in anammox metabolism (Strous et al. 2006), including 9 *hao-like* genes, hydrazine hydrolase (*hzf*), and hydrazine dehydrogenase
Dissimilatory nitrate reduction and denitrification	$5[CH_2O] + 4NO_3^- + 4H^+ \rightarrow 5CO_2 + 2N_2 + 7H_2O$ $5\ H_2 + 2NO_3^- + 2H^+ \rightarrow N_2 + 6H_2O$ $NO_3 \rightarrow NO_2^-$ $NO_2^- \rightarrow NO + N_2O$ $N_2O \rightarrow N_2$	*narDGHIJ*; *napA,B,D,E*; *nirB,C,K,U,N,O, S*; *norB*; *nosZ*
Assimilatory nitrate and nitrite reduction	$NAD(P)H + H^+ + NO_3^- + 2e^- \rightarrow NO_2^- + NAD(P)^+ + H_2O$ 6 ferredoxin (red) + 8 H^+ + 6 e^- + $NO_2^- \rightarrow NH_4^+$ + 6 ferredoxin (ox) + $2H_2O$	*nasA*, *nasB*, *nasC*, *nasD* (noncyanobacterial Bacteria); *narB* (cyanobacteria); *nrtA*, *nrtB*, *nrtC*, *nrtD* (or *nap*) permeases (cyanobacteria)
Dissimilatory nitrate reduction to ammonia	$NO_3^- + 2H^+ + 4H_2 \rightarrow NH_4^+ + 3H_2O$	*nir*, nar, *nap*, *nrfABCDE*
Ammonification/regeneration/remineralization	R-$NH_2 \rightarrow NH_4^+$	—
Ammonium assimilation	NH_3 + 2-oxoglutarate + NADPH + H^+ $\rightleftarrows$ glutamate + $NADP^+$ (glutamate dehyrdrogenase) NH_3 + glutamate + ATP $\rightarrow$ glutamine + ADP + Pi glutamine + 2-oxoglutarate + NADPH + $H^+ \rightarrow$ 2 glutamate + $NADP^+$ (glutamine synthetase and NADH-dependent glutamine:2-oxoglutarate amidotransferase)	*gdhA*,*gdhA*, *gltB*

Vamos a agregar esta información al *heatmap*.

- Primero, agregamos una nueva columna a la tabla para incorporar la información acerca del proceso bilógico en el que participa cada gen.

```
table_all$process <- NA
table_all$process[table_all$gene %in% c("nifH","nifD","nifK","nifG")] <-
"Nitrogen fixation"
table_all$process[table_all$gene %in% c("amoC","amoA","amoB","hao")] <-
"Ammonium oxidation"
table_all$process[table_all$gene %in% c("norA","norB")] <- "Nitrite oxidation"
table_all$process[table_all$gene %in% c("narH","narJ")] <- "Heterotrophic
oxidation"
```

```
table_all$process[table_all$gene %in% c("hfz")] <- "Anaerobic ammonia
oxidation"
```

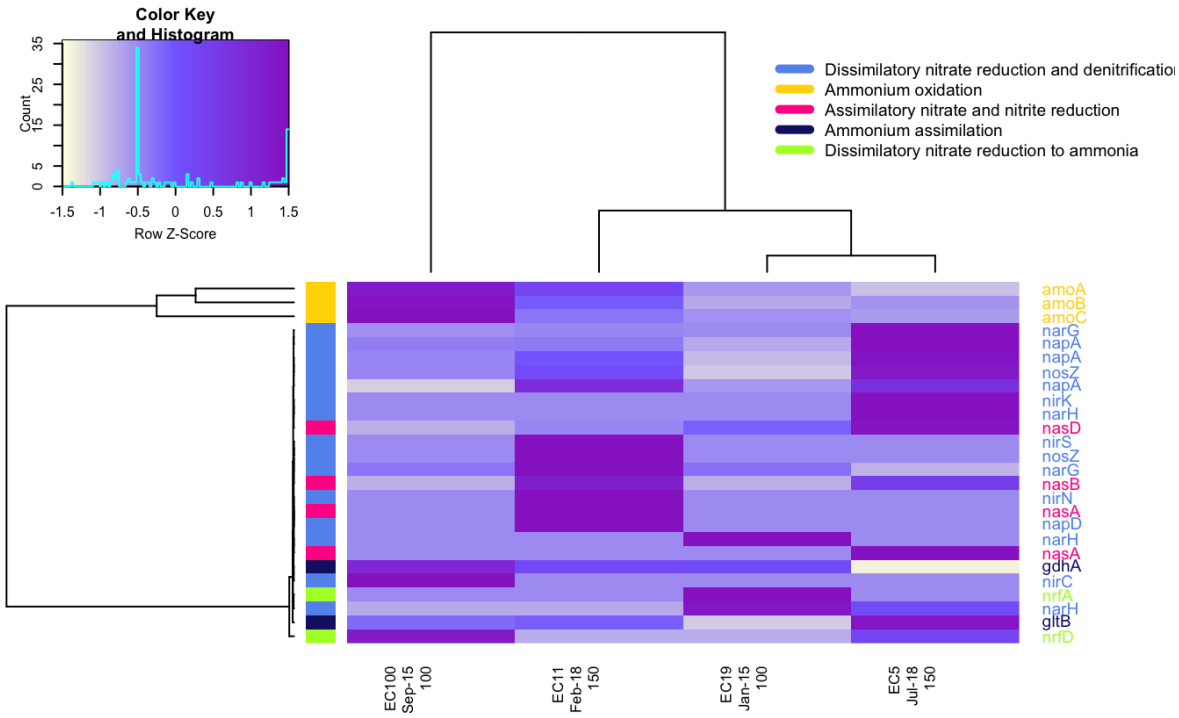
Keep filling the process in the same way.

```
unique(table_all$process)
## [1] "Dissimilatory nitrate reduction and denitrification"
## [2] "Ammonium oxidation"
## [3] "Assimilatory nitrate and nitrite reduction"
## [4] "Ammonium assimilation"
## [5] "Dissimilatory nitrate reduction to ammonia"
```

Podemos ver entonces que tenemos 5 procesos diferentes representados en nuestras muestras.

- Vamos a definir 5 colores para personalizar la visualización. Y graficamos el *heatmap* nuevamente.

```
#palette_gimp <-
c("#b3b477","#f8c47c","#81b477","#b8f87c","#77b48b","#7cf8a1","#76b4aa","#7cc7
f8","#7791b4","#7c93f8","#8177b4","#b17cf8","#a077b4","#e57cf8","#b477a5","#f8
7ca9", "#9d9d9d" , "#b47781","#f87c7c","#b49677", "#f8de7c")
#color_bar <- palette_gimp[as.numeric((as.factor(table_all$process)))]

color_bar <- table_all$process
color_bar[color_bar %in% "Dissimilatory nitrate reduction and
denitrification"] <- "cornflowerblue"
color_bar[color_bar %in% "Ammonium oxidation"] <- "gold"
color_bar[color_bar %in% "Assimilatory nitrate and nitrite reduction"] <-
"deeppink"
color_bar[color_bar %in% "Ammonium assimilation"] <- "midnightblue"
color_bar[color_bar %in% "Dissimilatory nitrate reduction to ammonia"] <-
"greenyellow"

x_labels <- paste(colnames(table_all[,c(2:5)]), metadata$Sampling_date,
metadata$Depth_m, sep="\n")

par(oma=c(0,0,3,0),xpd=TRUE)

heatmap.2(as.matrix(table_all[,c(2:5)]), scale="row", labRow = table_all$gene,
labCol = x_labels , cexCol=0.8 , col=my_palette, trace="none", margins = c(7,
7), RowSideColors=color_bar , key.par = list(cex=0.5),  lhei=c(2,4),
lwid=c(1.5,4), keysize=0.75, colRow = color_bar)

legend(x=0.65,y=1.34,
    legend = unique(table_all$process),
    col = unique(color_bar),
    lty= 1,
    lwd = 5,
    cex=.6 ,
    bty="n"
    )
```



Los primeros 3 genes *amoA*, *amoB* y *amoC* que son todos parte de proceso *ammonium oxidation*, son más abundantes en la muestra EC100. Es importante notar que los valores de abundancia están normalizados por filas en la tabla. Es decir, aunque las muestras EC19 y EC5 presenten bajo z-score, esto no significa que presenten una baja abundancia absoluta de este gen, sino que se encuentran en la fracción menor de los valores de abundancia de los genes de interés en este pequeño grupo de muestras. En efecto, si regresamos a mirar el gráfico de barras de la sección 3.3, esos gene presentan una abundancia mucho más alta en todas las muestras que cualquier otro gen de nuestra lista.

Teniendo esto en mente, aquí podemos mirar las diferencias entre las muestras.

Podemos notar que ambas muestras colectadas a 150 metros de profundidad tienen un grupo de alta abundancia de genes involucrados en *Dissimilatory nitrate reduction and denitrification* y *Assimilatory nitrate and nitrite reduction*. Aunque el mismo proceso es enriquecido en esas muestras, diferentes grupos de genes, o ortólogos son de alta abundancia en una de las otras muestras. Esto podría reflejar que aunque las comunidades microbianas podrían haber cambiado entre 2015 y 2018, las funciones de la comunidad microbiana se conservan. Por ejemplo, en el caso de *Assimilatory nitrate and nitrite reduction*, EC5 es enriquecida en *nasD* y *nasA*, mientras que EC11 es enriquecida en *nasB* y en otra variante de *nasA*. También, ambas muestras presentan alta abundancia de *nosZ*, pero con variantes diferentes; y para *napA*, varias

variantes se presentan con altos valores de abundancia en EC5 y otras en EC11.

Otra función que parece ser específica a cierta profundidad es *Dissimilatory nitrate reduction to ammonia*, para la cual EC100 esta enriquecida en *nrfD*, mientras que EC19 está enriquecida en *nrfA*.

Esto nos muestra que la ausencia de abundancia diferencial entre dos profundidades diferente no necesariamente implica que la profundidad no tenga una influencia en el metabolismo del nitrógeno, y que las similitudes funcionales puedes existir a pesar de las diferencias a nivel taxonómico en las comunidades microbianas.