practico3

December 6, 2023

---

1 Práctico 3

En este práctico estaremos realizando el preprocesamiento de datos, identificación de ASVs, y anotación taxonómica con DADA2, además de generar algunas visualizaciones relevantes en todo este proceso.

Vamos primero que nada vamos a instalar las dependencias que precisamos en este flujo de trabajo.

- cutadapt
- BiocManager
- DADA2
- ShortRead
- Biostrings
- tidyverse
- doParallel

cutadapt, la puedes instalar simplemente con el comando `sudo apt install cutadapt` o `pip install cutadapt`. Existen otras opciones que puedes ver acá. En en el servidor del CURE ya está instalado, por lo que no tendrás que instalarla.

Veamos las instalaciones de todos los paquetes de R.

Primero BiocManager, el cual nos permitirá instalar paquetes de bioconductor.

[1]:
```
# install.packages("BiocManager")
```

Ahora podemos instalar DADA2, Biostrings y ShortRead.

[2]:
```
# BiocManager::install("dada2")
```

[3]:
```
# BiocManager::install("Biostrings")
```

[4]:
```
# BiocManager::install("ShortRead")
```

tidyverse y doParallel los podemos instalar directo del CRAN.

[5]:
```
# install.packages("tidyverse")
```

[6]:
```
# install.packages("doParallel")
```

Una vez instaladas todas las funciones, las cargamos a nuestro ambiente:

[7]:
```
library(dada2)
library(Biostrings)
library(ShortRead)
library(tidyverse)
library(doParallel)
```

```
Loading required package: Rcpp

Loading required package: BiocGenerics


Attaching package: ‘BiocGenerics’


The following objects are masked from ‘package:stats’:

    IQR, mad, sd, var, xtabs


The following objects are masked from ‘package:base’:

    anyDuplicated, aperm, 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 objects are masked from ‘package:base’:
```

```
    expand.grid, I, unname


Loading required package: IRanges

Loading required package: XVector

Loading required package: GenomeInfoDb


Attaching package: 'Biostrings'


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

    strsplit


Loading required package: BiocParallel

Loading required package: Rsamtools

Loading required package: GenomicRanges

Loading required package: GenomicAlignments

Loading required package: SummarizedExperiment

Loading required package: MatrixGenerics

Loading required package: matrixStats


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


  Attaching core tidyverse packages

             tidyverse 2.0.0
  dplyr     1.1.2      readr     2.1.4
  forcats   1.0.0      stringr   1.5.0
  ggplot2   3.4.2      tibble    3.2.1
  lubridate 1.9.2      tidyr     1.3.0
  purrr     1.0.1
  Conflicts

          tidyverse_conflicts()
  lubridate::%within%() masks
IRanges::%within%()
  dplyr::collapse()     masks
Biostrings::collapse(), IRanges::collapse()
  dplyr::combine()      masks
Biobase::combine(), BiocGenerics::combine()
```

```
  purrr::compact()      masks
XVector::compact()
  purrr::compose()      masks
ShortRead::compose()
  dplyr::count()        masks
matrixStats::count()
  dplyr::desc()         masks
IRanges::desc()
  tidyr::expand()       masks
S4Vectors::expand()
  dplyr::filter()       masks
stats::filter()
  dplyr::first()        masks
GenomicAlignments::first(), S4Vectors::first()
  dplyr::id()           masks
ShortRead::id()
  dplyr::lag()          masks
stats::lag()
  dplyr::last()         masks
GenomicAlignments::last()
  ggplot2::Position()   masks
BiocGenerics::Position(), base::Position()
  purrr::reduce()       masks
GenomicRanges::reduce(), IRanges::reduce()
  dplyr::rename()       masks
S4Vectors::rename()
  lubridate::second()   masks
GenomicAlignments::second(), S4Vectors::second()
  lubridate::second<-() masks
S4Vectors::second<-()
  dplyr::slice()        masks
XVector::slice(), IRanges::slice()
  tibble::view()        masks
ShortRead::view()
  Use the conflicted package
(<http://conflicted.r-lib.org/>) to force all conflicts to
become errors
Loading required package: foreach


Attaching package: 'foreach'


The following objects are masked from 'package:purrr':

    accumulate, when
```

```
Loading required package: iterators

Loading required package: parallel
```

doParallel nos permite correr procesos en paralelo permitiendo reducir el tiempo de cómputo. Para esto, luego cargar la librería, debemos especificar cuántos hilos vamos utilizar, lo que realizamos con el siguiente comando.

[8]:
```
registerDoParallel(cores = 4)
```

Puedes averiguar cuántos hilos tienes en tu computadora con la función detectCores.

Vamos a definirnos una funciones en R, muy útiles para la identificación y eliminación de *primers*. Debo aclarar que el código que aparece abajo es una modificación de lo que aparece en el tutorial de DADA2.

Función 1: `all_orients`
Esta función lo que hace es, para cada secuencia de entrada (en nuestro caso un *primer*), genera su versión complementaria, reversa y reversa complementaria.

[9]:
```
all_orients <- function(PRIMER) {
  require(Biostrings)
  DNA <- DNAString(PRIMER)
  orients <- c(Forward = DNA,
               Complement = complement(DNA),
               Reverse = reverse(DNA),
               RevComp = reverseComplement(DNA))
  return(sapply(orients, toString))
}
```

Función 2: `primer_hits`
Esta función cuenta el número de veces en que una secuencia problema (en nuestro caso un *primer*) es encontrada en una muestra de amplicones en formato fastq.

[10]:
```
primer_hits <- function(PRIMER, INPUT) {
  require(ShortRead)
  require(Biostrings)
  nhits <- vcountPattern(pattern = PRIMER,
                         subject = sread(readFastq(INPUT)),
                         fixed = FALSE)
  return(sum(nhits > 0))
}
```

Función 3: `run_cutadapt`
Como su nombre lo indica, esta función lo que hace es correr `cutadapt` desde R, lo que se conoce como *wrapper*.

Para definir esta función, primero vamos a averiguar dónde quedó instalada la herramienta `cutadapt` en nuestra computadora. Esto lo deberíamos correr desde la línea de comando, pero

también lo podemos hacer desde R utilizando la función system2, como aparece abajo.

[11]:
```
cutadapt <- system2("which", "cutadapt")
```

En el caso de haber instalado `cutadapt` con `sudo apt install cutadapt`, el comando con la función `system2` es lo mismo que:

[12]:
```
cutadapt  <- "/usr/bin/cutadapt"
```

[13]:
```
cutadapt_runner <- function(primer_fwd = PRIMER_FWD,
                            primer_rev = PRIMER_REV,
                            input_r1 = INPUT_R1,
                            input_r2 = INPUT_R2,
                            output_r1 = OUTPUT_R1,
                            output_r2 = OUTPUT_R2,
                            nslots = NSLOTS) {

  primer_fwd_rc <- dada2::rc(primer_fwd)
  primer_rev_rc <- dada2::rc(primer_rev)
  # Trim FWD and the reverse-complement of REV off of R1 (forward reads)
  R1_flags <- paste("-g", primer_fwd, "-a", primer_rev_rc)
  # Trim REV and the reverse-complement of FWD off of R2 (reverse reads)
  R2_flags <- paste("-G", primer_rev, "-A", primer_fwd_rc)
  # Run Cutadapt
  for(i in seq_along(input_r1)) {
    system2(cutadapt,
            args = c(R1_flags, R2_flags,
                     "-n", 2,
                     "--minimum-length 1",
                     "--cores", nslots,
                     "-o", output_r1[i],
                     "-p", output_r2[i],
                     input_r1[i],
                     input_r2[i]))
  }
}
```

Para entender estás primeras tres funciones, puedes consultar la ayuda de `cutadapt`.

Función 4: `count_seqs`
Esta función permite contar el número de reads por secuencia.

[14]:
```
count_seqs <- function(p) {
  n <- readFastq(p) %>% sread %>% as.character() %>% length()
  sample <- basename(p) %>% sub(x = ., pattern = "_R1.*", replacement = "")
  output <- data.frame(sample = sample, nseq = n)
  return(output)
}
```

Dado que estamos trabajando en el ambiente de R, debemos antes de realizar cualquier análisis, poder ubicar los datos en nuestra computadora.
Para esto vamos a definir tres variables en R:

`INPUT_DIR`: el directorio donde se encuentran los archivos fastq.
`PATTERN_R1`: patrón en los nombres de los *reads* R1.
`PATTERN_R2`: patrón en los nombres de los *reads* R2.

La función getwd nos puede ser útil para determinar la ruta relativa hacia nuestros datos (i.e., `INPUT_DIR`).

[15]:
```
INPUT_DIR <- "./data/raw/"
```

[16]:
```
PATTERN_R1 <- "_L001_R1_001_redu.fastq"
```

[17]:
```
PATTERN_R2 <- "_L001_R2_001_redu.fastq"
```

Con la función list.files vamos crearnos dos vectores (*reads* R1 y R2, resp.), donde se incluye la ruta para las 30 muestras de nuestro se de datos.

[18]:
```
rawR1 <- sort(list.files(INPUT_DIR, pattern = PATTERN_R1,
                         full.names = T))
rawR2 <- sort(list.files(INPUT_DIR, pattern = PATTERN_R2,
                         full.names = T))
```

[19]:
```
rawR1
```

1. './data/raw//1_S1_L001_R1_001_redu.fastq.gz' 2. './data/raw//13_S13_L001_R1_001_redu.fastq.gz'
3. './data/raw//14_S14_L001_R1_001_redu.fastq.gz' 4. './data/raw//15_S15_L001_R1_001_redu.fastq.gz'
5. './data/raw//16_S16_L001_R1_001_redu.fastq.gz' 6. './data/raw//17_S17_L001_R1_001_redu.fastq.gz'
7. './data/raw//18_S18_L001_R1_001_redu.fastq.gz' 8. './data/raw//2_S2_L001_R1_001_redu.fastq.gz'
9. './data/raw//21_S21_L001_R1_001_redu.fastq.gz' 10. './data/raw//34_S33_L001_R1_001_redu.fastq.gz'
11. './data/raw//35_S34_L001_R1_001_redu.fastq.gz' 12. './data/raw//36_S35_L001_R1_001_redu.fastq.gz'
13. './data/raw//37_S36_L001_R1_001_redu.fastq.gz' 14. './data/raw//38_S37_L001_R1_001_redu.fastq.gz'
15. './data/raw//39_S38_L001_R1_001_redu.fastq.gz' 16. './data/raw//42_S41_L001_R1_001_redu.fastq.gz'
17. './data/raw//61_S56_L001_R1_001_redu.fastq.gz' 18. './data/raw//62_S57_L001_R1_001_redu.fastq.gz'
19. './data/raw//63_S58_L001_R1_001_redu.fastq.gz' 20. './data/raw//64_S59_L001_R1_001_redu.fastq.gz'
21. './data/raw//65_S60_L001_R1_001_redu.fastq.gz' 22. './data/raw//66_S61_L001_R1_001_redu.fastq.gz'
23. './data/raw//70-RE170207_S70_L001_R1_001_redu.fastq.gz'
24. './data/raw//76-RA170504_S76_L001_R1_001_redu.fastq.gz'
25. './data/raw//77-RA170503_S77_L001_R1_001_redu.fastq.gz'
26. './data/raw//78-RA170502_S78_L001_R1_001_redu.fastq.gz'
27. './data/raw//79-RA170501_S79_L001_R1_001_redu.fastq.gz'
28. './data/raw//84_S75_L001_R1_001_redu.fastq.gz' 29. './data/raw//85-RA170801_S85_L001_R1_001_red
30. './data/raw//87-RA171104_S87_L001_R1_001_redu.fastq.gz'

¿Puedes corroborar que las variables `rawR1` y `rawR2` hayan quedado definidas correctamente?

Los *primers* que fueron utilizados para generar los datos en el trabajo de Griffero et al son los siguientes:

Primer Forward: "GTGYCAGCMGCCGCGGTAA"
Primer Reverse: "CCGYCAATTYMTTTRAGTTT".

Vamos a crearnos dos variables donde pondremos los *primers*.

[20]:
```
PRIMER_FWD <- "GTGYCAGCMGCCGCGGTAA"
```

[21]:
```
PRIMER_REV <- "CCGYCAATTYMTTTRAGTTT"
```

Estas variables deberán ser definidas en cada caso particular (no todos los estudios utilizan los mismos *primers* ;)).
Si los *reads* se generaron con un protocolo estándar de secuenciación de Illumina, entonces el *primer forward* estará presente en el extremo 5' del *read* 1 y el *primer reverse* en el extremo 5' del *read* 2.
Para eliminar los primers, comúnmente se utilizan dos estrategias:
La primera consiste, simplemente, en recortar del extremo 5' de los *reads*, un número de bases igual al largo de los *primers*, es decir, recortar los primers (esto sólo tendrá sentido si todos los *primers* tienen el mismo largo). Con `DADA2` podemos realizar dicha tarea utilizando la función filterAndTrim.
La segunda estrategia, consiste en utilizar la herramienta cutadapt, la cual busca la secuencia de los primers y los elimina.

En este práctico estaremos aplicando la segunda estrategía ya que ésta es más robusta (como te podrás imaginar dado que nos tomamos el trabajo de definir algunas funciones para correr `cutadapt`).

Primero, utilizando nuestra función `all_orients`, vamos a crear las secuencias complementaria, reversa y reversa complementaria para cada primer.

[22]:
```
FWD_orients <- all_orients(PRIMER_FWD)
REV_orients <- all_orients(PRIMER_REV)
```

Veamos qué nos devuelve esta función.

[23]:
```
FWD_orients
```

Forward 'GTGYCAGCMGCCGCGGTAA' **Complement** 'CACRGTCGKCGGCGCCATT'
Reverse 'AATGGCGCCGMCGACYGTG' **RevComp** 'TTACCGCGGCKGCTGRCAC'

[24]:
```
REV_orients
```

Forward 'CCGYCAATTYMTTTRAGTTT' **Complement** 'GGCRGTTAARKAAAYTCAAA'
Reverse 'TTTGARTTTMYTTAACYGCC' **RevComp** 'AAACTYAAAKRAATTGRCGG'

Sumamente práctica esta función, ¿no crees?

Ahora con nuestra segunda función `primer_hits`, vamos a buscar todas estas versiones de nuestros *primers* en el primer archivo que tenemos listado en `rawR1` y `rawR2`.

[25]:
```
counts_log <- rbind(all_orients_PRIMER_FWD_vs_rawR1=sapply(FWD_orients,
                                                           primer_hits,
                                                           INPUT = rawR1[1]),
```

```
                    all_orients_PRIMER_FWD_vs_rawR2=sapply(FWD_orients,
                                                           primer_hits,
                                                           INPUT = rawR2[1]),
                    all_orients_PRIMER_REV_vs_rawR1=sapply(REV_orients,
                                                           primer_hits,
                                                           INPUT = rawR1[1]),
                    all_orients_PRIMER_REV_vs_rawR2=sapply(REV_orients,
                                                           primer_hits,
                                                           INPUT = rawR2[1]))
```

Veamos cómo se ve la salida:

[26]:
```
counts_log
```

A matrix: 4 × 4 of type int

	Forward	Complement	Reverse	RevC
all_orients_PRIMER_FWD_vs_rawR1	2593	0	0	0
all_orients_PRIMER_FWD_vs_rawR2	0	0	0	2
all_orients_PRIMER_REV_vs_rawR1	0	0	0	7
all_orients_PRIMER_REV_vs_rawR2	2583	0	0	0

Aparece un bajo conteo donde no debería para los *primers* reverso complemento. A pesar de esto, los datos se ven bien.

Ahora que sabemos que los *primers* están donde deberían de estar, vamos a correr nuestra función `run_cutadapt`.

Para cada fastq que estaremos preprocesando mediante la eliminación de primers, vamos a crear un archivo de salida, por lo que debemos definir dónde guardamos estos archivos y sus nombres.

Creamos un directorio desde R:

[27]:
```
OUTPUT_DIR<-"./data/rm_primers"
dir.create(OUTPUT_DIR)
```

```
Warning message in dir.create(OUTPUT_DIR):
"'./data/rm_primers' already exists"
```

Definimos los nombres.

[28]:
```
sample.names <- basename(rawR1) |>
                sub(pattern = paste(PATTERN_R1, ".gz", sep = ""),
                    replacement = "")
```

[29]:
```
sample.names
```

1. '1_S1' 2. '13_S13' 3. '14_S14' 4. '15_S15' 5. '16_S16' 6. '17_S17' 7. '18_S18' 8. '2_S2' 9. '21_S21' 10. '34_S33' 11. '35_S34' 12. '36_S35' 13. '37_S36' 14. '38_S37' 15. '39_S38' 16. '42_S41' 17. '61_S56' 18. '62_S57' 19. '63_S58' 20. '64_S59' 21. '65_S60' 22. '66_S61' 23. '70-RE170207_S70' 24. '76-RA170504_S76' 25. '77-RA170503_S77' 26. '78-RA170502_S78' 27. '79-RA170501_S79' 28. '84_S75' 29. '85-RA170801_S85' 30. '87-RA171104_S87'

Unimos la ruta al directorio con los nombres de los archivos.

[30]:
```
rmprimerR1 <- file.path(OUTPUT_DIR, paste(sample.names,
                                          "R1_rmprimer.fastq.gz", sep = "_"))
rmprimerR2 <- file.path(OUTPUT_DIR, paste(sample.names,
                                          "R2_rmprimer.fastq.gz", sep = "_"))
```

[31]:
```
rmprimerR1
```

1. './data/rm_primers/1_S1_R1_rmprimer.fastq.gz' 2. './data/rm_primers/13_S13_R1_rmprimer.fastq.gz'
3. './data/rm_primers/14_S14_R1_rmprimer.fastq.gz' 4. './data/rm_primers/15_S15_R1_rmprimer.fastq.gz'
5. './data/rm_primers/16_S16_R1_rmprimer.fastq.gz' 6. './data/rm_primers/17_S17_R1_rmprimer.fastq.gz'
7. './data/rm_primers/18_S18_R1_rmprimer.fastq.gz' 8. './data/rm_primers/2_S2_R1_rmprimer.fastq.gz'
9. './data/rm_primers/21_S21_R1_rmprimer.fastq.gz' 10. './data/rm_primers/34_S33_R1_rmprimer.fastq.gz'
11. './data/rm_primers/35_S34_R1_rmprimer.fastq.gz' 12. './data/rm_primers/36_S35_R1_rmprimer.fastq.gz'
13. './data/rm_primers/37_S36_R1_rmprimer.fastq.gz' 14. './data/rm_primers/38_S37_R1_rmprimer.fastq.gz'
15. './data/rm_primers/39_S38_R1_rmprimer.fastq.gz' 16. './data/rm_primers/42_S41_R1_rmprimer.fastq.gz'
17. './data/rm_primers/61_S56_R1_rmprimer.fastq.gz' 18. './data/rm_primers/62_S57_R1_rmprimer.fastq.gz'
19. './data/rm_primers/63_S58_R1_rmprimer.fastq.gz' 20. './data/rm_primers/64_S59_R1_rmprimer.fastq.gz'
21. './data/rm_primers/65_S60_R1_rmprimer.fastq.gz' 22. './data/rm_primers/66_S61_R1_rmprimer.fastq.gz'
23. './data/rm_primers/70-RE170207_S70_R1_rmprimer.fastq.gz'
24. './data/rm_primers/76-RA170504_S76_R1_rmprimer.fastq.gz'
25. './data/rm_primers/77-RA170503_S77_R1_rmprimer.fastq.gz'
26. './data/rm_primers/78-RA170502_S78_R1_rmprimer.fastq.gz'
27. './data/rm_primers/79-RA170501_S79_R1_rmprimer.fastq.gz'
28. './data/rm_primers/84_S75_R1_rmprimer.fastq.gz' 29. './data/rm_primers/85-RA170801_S85_R1_rmprim
30. './data/rm_primers/87-RA171104_S87_R1_rmprimer.fastq.gz'

Ejcutamos la función *wrapper* de `cutadapt`.

[32]:
```
cutadapt_runner(primer_fwd = PRIMER_FWD,
                primer_rev = PRIMER_REV,
                input_r1 = rawR1,
                input_r2 = rawR2,
                output_r1 = rmprimerR1,
                output_r2 = rmprimerR2,
                nslots = 4)
```

Veamos los archivos que tenemos en `OUTPUT_DIR`.

[33]:
```
list.files(OUTPUT_DIR) |> head()
```

1. '1_S1_R1_rmprimer.fastq.gz' 2. '1_S1_R2_rmprimer.fastq.gz'
3. '1_S1.gz_R1_rmprimer.fastq.gz' 4. '1_S1.gz_R2_rmprimer.fastq.gz'
5. '13_S13_R1_rmprimer.fastq.gz' 6. '13_S13_R2_rmprimer.fastq.gz'

¿Puedes correr nuevamente nuestra función `primer_hits` sobre los archivos fastq sin primers para verificar que efectivamente hayamos eliminado los *primers*?

DADA2 cuenta con una función muy práctica para visualizar la calidad de nuestras secuencias: plotQualityProfile (otra herramienta muy usada para esto es FastQC).

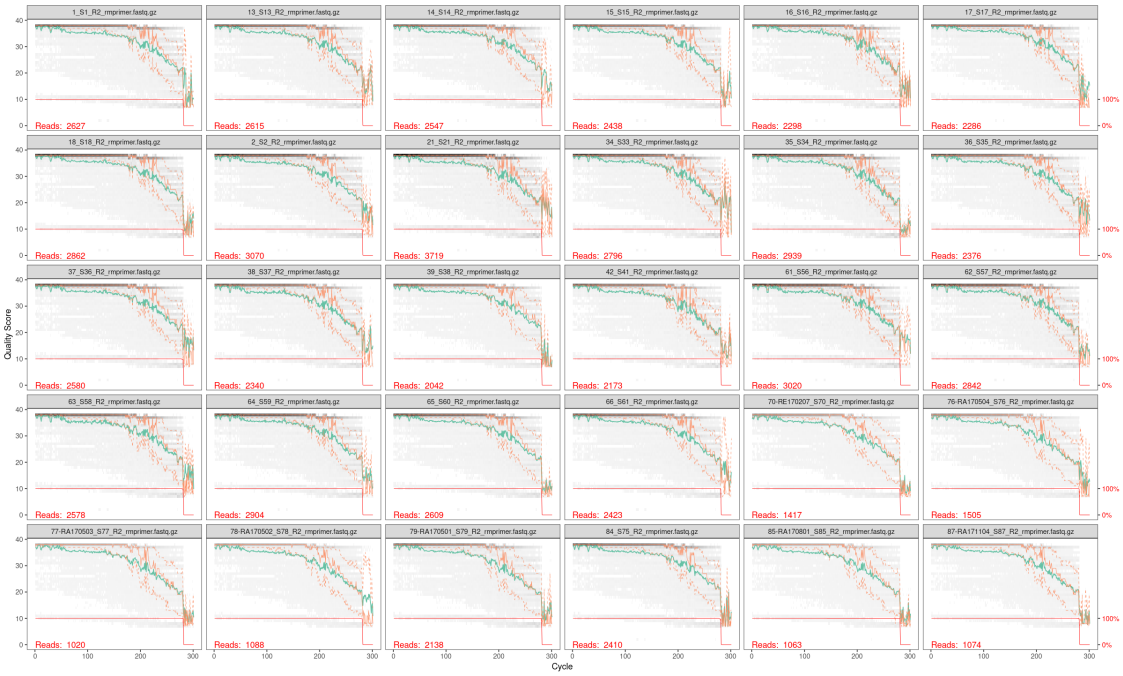
Para utilizarla, con todos sus parámetros por defecto, simplemente le damos como entrada la ubicación de los archivos que queremos analizar. En nuestro caso, vamos a analizar los *reads* sin *primers*.

[34]:
```
quality_plot_R1 <- plotQualityProfile(rmprimerR1)
quality_plot_R2 <- plotQualityProfile(rmprimerR2)
```

```
Warning message:
"The `<scale>` argument of `guides()` cannot be `FALSE`. Use "none"
instead as of ggplot2 3.3.4.
  The deprecated feature was likely used in the dada2
package.
  Please report the issue at
<https://github.com/benjjneb/dada2/issues>."
```

[35]:
```
options(repr.plot.width=20, repr.plot.height=12)
quality_plot_R1
```

[36]:
```
options(repr.plot.width=20, repr.plot.height=12)
quality_plot_R2
```



Estos plots muestran la distribución de los puntajes de calidad en cada posición (mapa de calor de gris a negro).
Las línea verde es la media, el naranja es la mediana y las líneas naranjas discontinuas corresponden al primer y tercer cuartil.

Como sucede generalmente, observamos que la calidad de las secuencias decrece a lo largo de los *reads* y hay un diferencia notoria entre la calidad de los *reads* R1 y R2.
¿Sabes por qué es esto? Acá puedes encontrar una respuesta.

Ahora vamos a realizar un análisis de calidad que puede resultar muy útil para identificar muestras problemáticas.
Lo que vamos a hacer es comparar, para cada muestra, la calidad promedio (de una submuestra) de *reads* vs. el número de *reads*.

Primero vamos a contar el número *reads* con la función que definimos arriba `count_seqs` y para hacerlo más rápido, utilizaremos las facilidades de `doParallel` (i.e., foreach with %dopar%).

[37]:
```
seq_counts_df <- foreach(i = rmprimerR1, .combine=rbind) %dopar% {
  count_seqs(i)
}
```

Luego, computamos los promedios de calidad de R1. Para esto utilizamos la función qa de ShortRead.
Para correr esta función, similarmente a cómo lo hicimos arriba, debemos especificar un directorio de entrada y un patrón para identificar los archivos que vamos a analizar.

[38]:
```
INPUT_DIR <- "./data/rm_primers"
```

[39]:
```
PATTERN_R1 <- "_R1_rmprimer.fastq.gz"
```

[40]:
```
qa_r1 <- qa(dirPath = INPUT_DIR, pattern = PATTERN_R1, sample = T, n = 5000)
qa_r1_df <- qa_r1[["perCycle"]][["quality"]]
qa_means_r1 <- qa_r1_df %>%
               group_by(lane) %>%
               summarize(mean_q = sum(Score*Count)/sum(Count))
qa_means_r1$lane <- qa_means_r1$lane %>% sub(x = ., pattern = "_R1.*",
                                             replacement = "")
```

Y también para R2.

[41]:
```
PATTERN_R2 <- "_R2_rmprimer.fastq.gz"
```

[42]:
```
qa_r2 <- qa(dirPath = INPUT_DIR, pattern = PATTERN_R2, sample = T, n = 5000)
qa_r2_df <- qa_r2[["perCycle"]][["quality"]]
qa_means_r2 <- qa_r2_df %>%
               group_by(lane) %>%
               summarize(mean_q = sum(Score*Count)/sum(Count))
qa_means_r2$lane <- qa_means_r2$lane %>% sub(x = ., pattern = "_R2.*",
                                             replacement = "")
```

Siguiente paso es cruzar los resultados de promedios de calidad con el número de secuencias por muestra, lo que podemos hacer fácilmente con la función left_join de `tidyverse`.

[43]:
```
qa_means1counts <- left_join(x = qa_means_r1, y = seq_counts_df,
                             by = c("lane" = "sample"))
qa_means2counts <- left_join(x = qa_means_r2, y = seq_counts_df,
                             by = c("lane" = "sample"))
```

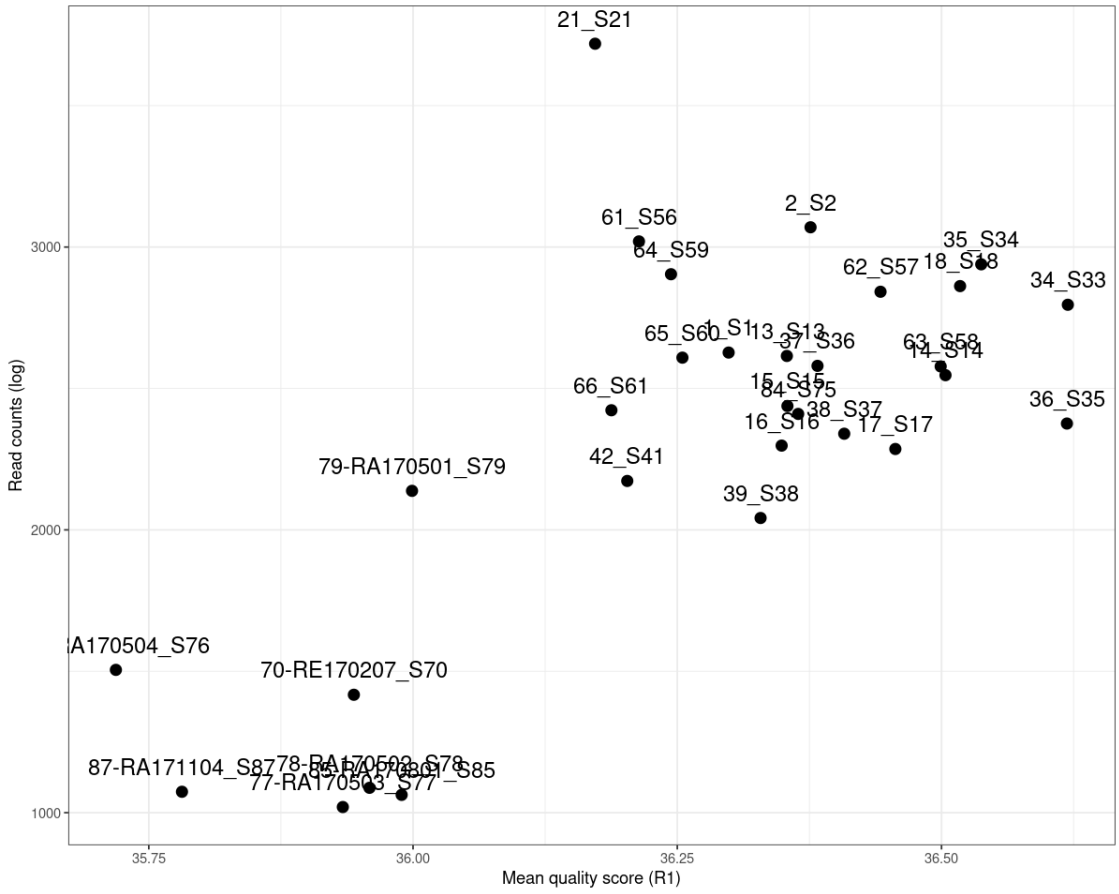
Finalmente, realizamos los plots para R1 y R2.

[44]:
```
qual_vs_nseq_r1 <- ggplot(data = qa_means1counts, aes(x = mean_q, y = nseq)) +
  geom_point(size = 3) +
  theme_bw() +
  #scale_y_log10() +
  ylab("Read counts (log)") +
  xlab("Mean quality score (R1)") +
  geom_text(aes(label=as.character(lane)), hjust=0.5, vjust=-1, size = 5)
```

[45]:
```
qual_vs_nseq_r2 <- ggplot(data = qa_means2counts, aes(x = mean_q, y = nseq)) +
  geom_point(size = 3) +
  theme_bw() +
  #scale_y_log10() +
  ylab("Read counts (log)") +
  xlab("Mean quality score (R1)") +
  geom_text(aes(label=as.character(lane)), hjust=0.5, vjust=-1, size = 5)
```

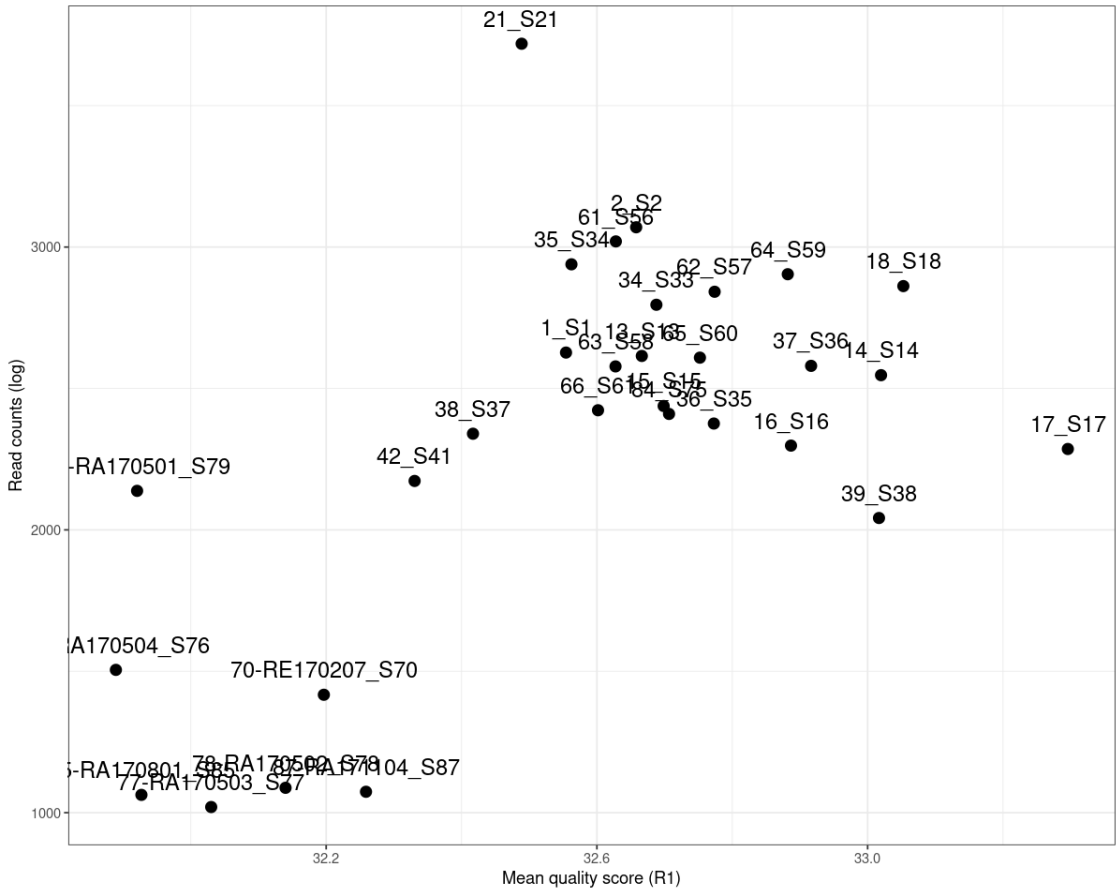
[46]:
```
options(repr.plot.width=10, repr.plot.height=8)
qual_vs_nseq_r1
```

```
Warning message:
"Removed 30 rows containing missing values (`geom_point()`)."
Warning message:
"Removed 30 rows containing missing values (`geom_text()`)."
```



[47]:
```
options(repr.plot.width=10, repr.plot.height=8)
qual_vs_nseq_r2
```

```
Warning message:
"Removed 30 rows containing missing values (`geom_point()`)."
Warning message:
"Removed 30 rows containing missing values (`geom_text()`)."
```



Es interesante que parecen formarse dos grupos: uno de mayor calidad y mayor número de secuencias, y otro de menor calidad y menor número de secuencias.
Este plot puede ser un insumo para definir si es necesario descartar algunas muestras.
En nuestro caso, el grupo de menor calidad aún es suficientemente bueno (tenemos puntajes por arriba de 32 y un número de *reads* comparable). En consecuencia, no estaremos descartando ninguna muestra.

Como última tarea en el control de calidad, vamos a recortar las regiones 5' de los *reads*. Para esto, en base a los plots `quality_plot_R1` y `quality_plot_R2` vamos a definir el punto de corte. Esta decisión es algo subjetiva, y es conveniente probar qué resultados obtenemos con distintos valores. En este práctico, a los efectos de mostrar cómo funciona esta función, estaremos utilizando dos únicos valores, uno para los *reads* R1 y otro para los R2.

[48]:
```
TRUNC_R1 <- 250
TRUNC_R2 <- 225
```

Al igual a cómo lo hicimos anteriormente, vamos a definir el directorio y nombre de los archivos de salida, ya que queremos que estos archivos filtrados por la calidad queden guardados en nuestra computadora.

[49]:
```
OUTPUT_DIR <- "./data/quality_filtered"
dir.create(OUTPUT_DIR)
```

```
Warning message in dir.create(OUTPUT_DIR):
"'./data/quality_filtered' already exists"
```

[50]:
```
filtR1 <- file.path(OUTPUT_DIR, paste(sample.names, "R1_filt.fastq.gz",
                                      sep = "_"))
filtR2 <- file.path(OUTPUT_DIR, paste(sample.names, "R2_filt.fastq.gz",
                                      sep = "_"))
```

[51]:
```
filterAndTrim_log <- filterAndTrim(fwd = rmprimerR1, filt = filtR1,
                                   rev = rmprimerR2, filt.rev = filtR2,
                                   truncLen = c(TRUNC_R1, TRUNC_R2),
                                   maxN = 0, maxEE = c(2,2), truncQ = 2,
                                   rm.phix = TRUE,
                                   compress=TRUE,
                                   multithread = 4)
```

Puedes consultar la ayuda de la función `filterAndTrim` para ver qué significan estos parámetros.

Veamos la salida de esta función.

[52]:
```
list.files(OUTPUT_DIR)
```

1. '1_S1_R1_filt.fastq.gz' 2. '1_S1_R2_filt.fastq.gz' 3. '13_S13_R1_filt.fastq.gz'
4. '13_S13_R2_filt.fastq.gz' 5. '14_S14_R1_filt.fastq.gz' 6. '14_S14_R2_filt.fastq.gz'
7. '15_S15_R1_filt.fastq.gz' 8. '15_S15_R2_filt.fastq.gz' 9. '16_S16_R1_filt.fastq.gz'
10. '16_S16_R2_filt.fastq.gz' 11. '17_S17_R1_filt.fastq.gz' 12. '17_S17_R2_filt.fastq.gz'
13. '18_S18_R1_filt.fastq.gz' 14. '18_S18_R2_filt.fastq.gz' 15. '2_S2_R1_filt.fastq.gz'
16. '2_S2_R2_filt.fastq.gz' 17. '21_S21_R1_filt.fastq.gz' 18. '21_S21_R2_filt.fastq.gz'
19. '34_S33_R1_filt.fastq.gz' 20. '34_S33_R2_filt.fastq.gz' 21. '35_S34_R1_filt.fastq.gz'
22. '35_S34_R2_filt.fastq.gz' 23. '36_S35_R1_filt.fastq.gz' 24. '36_S35_R2_filt.fastq.gz'
25. '37_S36_R1_filt.fastq.gz' 26. '37_S36_R2_filt.fastq.gz' 27. '38_S37_R1_filt.fastq.gz'
28. '38_S37_R2_filt.fastq.gz' 29. '39_S38_R1_filt.fastq.gz' 30. '39_S38_R2_filt.fastq.gz'
31. '42_S41_R1_filt.fastq.gz' 32. '42_S41_R2_filt.fastq.gz' 33. '61_S56_R1_filt.fastq.gz'
34. '61_S56_R2_filt.fastq.gz' 35. '62_S57_R1_filt.fastq.gz' 36. '62_S57_R2_filt.fastq.gz'
37. '63_S58_R1_filt.fastq.gz' 38. '63_S58_R2_filt.fastq.gz' 39. '64_S59_R1_filt.fastq.gz'
40. '64_S59_R2_filt.fastq.gz' 41. '65_S60_R1_filt.fastq.gz' 42. '65_S60_R2_filt.fastq.gz'
43. '66_S61_R1_filt.fastq.gz' 44. '66_S61_R2_filt.fastq.gz' 45. '70-RE170207_S70_R1_filt.fastq.gz'
46. '70-RE170207_S70_R2_filt.fastq.gz' 47. '76-RA170504_S76_R1_filt.fastq.gz'
48. '76-RA170504_S76_R2_filt.fastq.gz' 49. '77-RA170503_S77_R1_filt.fastq.gz'
50. '77-RA170503_S77_R2_filt.fastq.gz' 51. '78-RA170502_S78_R1_filt.fastq.gz'
52. '78-RA170502_S78_R2_filt.fastq.gz' 53. '79-RA170501_S79_R1_filt.fastq.gz'
54. '79-RA170501_S79_R2_filt.fastq.gz' 55. '84_S75_R1_filt.fastq.gz'
56. '84_S75_R2_filt.fastq.gz' 57. '85-RA170801_S85_R1_filt.fastq.gz'
58. '85-RA170801_S85_R2_filt.fastq.gz' 59. '87-RA171104_S87_R1_filt.fastq.gz'
60. '87-RA171104_S87_R2_filt.fastq.gz'

Todo parece correcto, pero por mayor control puedes abrir alguno de estos archivos desde la línea de comando en BASH.

Como recordarás del teórico, DADA2 realiza una inferencia de ASVs en base al modelado de los errores en los *reads* para cada puntaje de calidad.

Esto nos da, por ejemplo, la probabilidad de observar una transición para un puntaje de calidad de 35: p(A→C, 35).

El 3x y 4x que aparecen arriba de los *reads* representan su abundancia.

Para modelar los errores, DADA2 cuenta con la función learnErros, la cual estarmos utilizando luego de cargar los datos.
Notar que ahora vamos a trabajar con los archivos sin *primers* y filtrados por calidad, los cuales tenemos en los vectores `filtR1` y `filtR2`.

[53]:
```
errR1 <- learnErrors(filtR1, multithread=4)
errR2 <- learnErrors(filtR2, multithread=4)
```

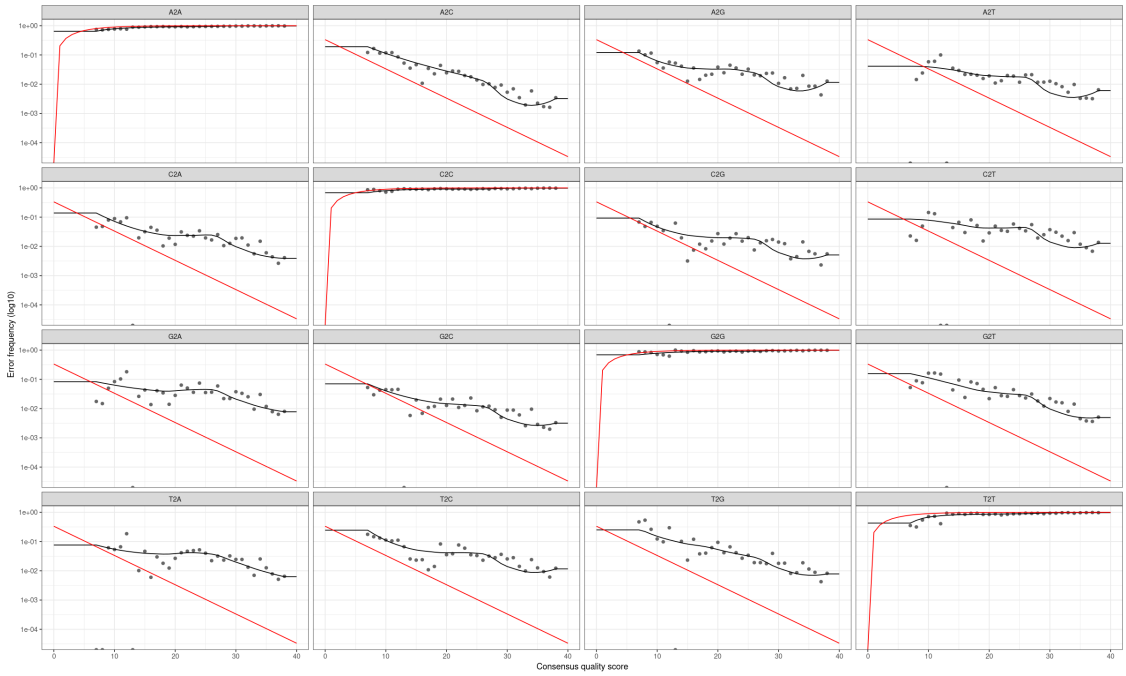
```
13700500 total bases in 54802 reads from 30 samples will be used for learning
the error rates.
12330450 total bases in 54802 reads from 30 samples will be used for learning
the error rates.
```

Muy convenientemente, DADA2, incluye una función para visualizar el modelado de errores: plotErrors.

[54]:
```
error_plot_R1 <- plotErrors(errR1, nominalQ=TRUE)
error_plot_R2 <- plotErrors(errR2, nominalQ=TRUE)
```

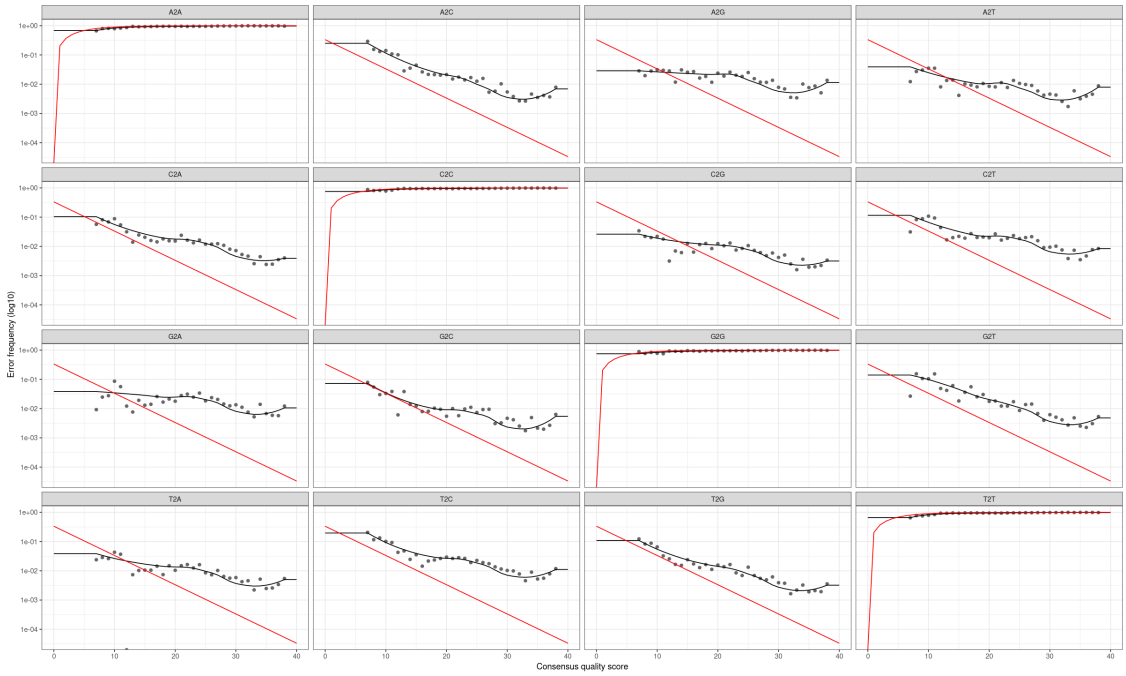
[55]:
```
options(repr.plot.width=20, repr.plot.height=12)
error_plot_R1
```

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



[56]:
```
options(repr.plot.width=20, repr.plot.height=12)
error_plot_R2
```

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



"El eje x muestra el nivel de calidad; el eje y la frecuencia de la transición. Los puntos muestran las frecuencias observadas, la línea negra el modelo de error inferido por DADA2 utilizando su ajuste loess y la línea roja las tasas esperadas dada la definición nominal del puntaje de calidad. Las puntuaciones de calidad de Illumina son bastante informativas sobre las tasas de error de sustitución, pero se observan desviaciones sistemáticas de las tasas esperadas". Texto traducido de (Callahan et al 2016).

Este paso, consiste simplemente en agrupar todas las secuencias que aparecen idénticas en nuestros datos, sin utilizar ningún modelado de errores. Como resultado, todas las secuencias iguales son agrupadas en una única secuencia, para la cual además es registrada su abundancia, como se muestra en la figura de abajo.

Para esto simplemente corremos dos comandos utilizando la función derepFastq.

[57]:
```
derepR1 <- derepFastq(filtR1)
derepR2 <- derepFastq(filtR2)
```

Está parte es, probablemente, la más fundamental en todo el flujo de trabajo: la reconstrucción de ASVs mediante el *Divisive partitioning algorithm*.

Donde el p-valor A entre los *reads* "i" y "j" se computa cómo:

Y "nj" y "ai" son las abundancias de las secuencias "j" e "i", respectivamente.

Vayamos al grano.

[58]:
```
dadaR1 <- dada(derepR1, err=errR1, multithread=4, pool = T)
dadaR2 <- dada(derepR2, err=errR2, multithread=4, pool = T)
```

```
30 samples were pooled: 54802 reads in 14086 unique sequences.
30 samples were pooled: 54802 reads in 16292 unique sequences.
```

¿Qué utilidad tiene la opción `pool`?

Esta tarea consiste en alinear los extremos de los *reads* R1 y R2 para unir las secuencias en el caso de que exista una superposición mayor a `minOverlap`.
Dado que los ASVs ya fueron identificados para R1 y R2, el alineamiento debe de ser exacto (sin *mismatches* o *gaps*). En este práctico estaremos utilizando un único valor de `minOverlap`; no obstante, si estuviéramos trabajando con datos que no conocemos, sería conveniente probar distintos valores de `minOverlap` y evaluar cómo se ve afectado el *merging* en cada caso. Notar que para definir este parámetro, también debemos tener en cuenta cuánto hemos recortado los *reads* en el control de calidad.

En nuestro caso, usaremos un `minOverlap` de 12 bases.

[59]:
```
MIN_OVERLAP <- 12
```

[60]:
```
merged <- mergePairs(dadaR1, derepR1,
                     dadaR2, derepR2,
                     minOverlap = MIN_OVERLAP)
```

El objeto `merged` tiene toda la información que precisamos, pero no es un formato muy amigable. Por este motivo, y para realizar la eliminación de bimeras, vamos a convertir el objeto `merged` en una tabla.

[61]:
```
seqtab <- makeSequenceTable(merged)
```

Veamos qué dimensiones tiene la tabla de abundancias.

[62]:
```
dim(seqtab)
```

1. 30 2. 1240

Esto es, 30 muestras (filas) por 1240 ASVs (columnas). Este es un número bajo de ASVs, pero debemos tener en cuenta que estamos trabajando con un set de datos de "juguete".

Veamos algunas filas y columnas de nuestra matriz.

[63]:
```
seqtab[1:3, 1:3]
```

A matrix: 3 × 3 of type int

	GACGAAGGGGGCAAGCGTTGTTCGGAATTACTGG
1_S1_R1_filt.fastq.gz	244
13_S13_R1_filt.fastq.gz	218
14_S14_R1_filt.fastq.gz	25

Se ve un poco raro, dado que los nombres de las columnas son las secuencias de cada ASV. No obstante, resultará muy conveniente tener estas secuencias en posteriores análisis.

La última parte en el análisis de identificación de ASVs, consiste en eliminar las bimeras. Estas se forman durante la amplifcación por PCR como se explica acá.

Aunque podriamos utilizar otras herramientas, por ejemplo vsearch, resulta especialmente conveniente seguir trabjando en R con DADA2, por lo que utilizaremos la función removeBimeraDenovo. Esto se debe no sólo al hecho de que es necesario realizar menos pasos en la manipulación de datos, sino también a que `removeBimeraDenovo`, al trabajar sobre ASVs, es particularmente sensible.
"Los algoritmos comunes de identificación de quimeras fueron diseñados bajo el supuesto de que la sensibilidad en las variantes quiméricas cercanas no era muy importante porque las variantes cercanas probablemente se clusterizarán en una misma OTU de todos modos. Sin embargo, DADA2 viola esta suposición, ya que distingue variantes que difieren tan solo en un nucleótido." Texto traducido de (Callahan et al 2016).

[64]:
```
seqtab.nochim <- removeBimeraDenovo(seqtab, multithread = 4)
```

Veamos cuantos ASVs quedaron luego de la eliminación de bimeras.

[65]:
```
dim(seqtab.nochim)
```

1. 30 2. 1047

Con el siguiente comando podemos computar el porcentaje que fue eliminado:

[66]:
```
100 -(100*dim(seqtab.nochim)[2]/dim(seqtab)[2]) |> round(3)
```

15.565

15.565% es un porcentaje bastante razonable. Comúnmente se observan valores más altos.

Esta tabla es nuestro resultado final en la identificación de ASVs, por lo que vamos a guardarla en formato csv.

[67]:
```
OUTPUT_DIR <- "./results"
dir.create(OUTPUT_DIR)
write.csv(x = t(seqtab.nochim),
          file = file.path(OUTPUT_DIR, "asv_abund.csv"))
```

```
Warning message in dir.create(OUTPUT_DIR):
"'./results' already exists"
```

En esta tarea vamos a contar el número de secuencias en las distintas etapas del preprocesamiento, identificación de ASVs y eliminación de bimeras.

[68]:
```
track_n_seqs <- data.frame(samples = sample.names,
                           raw = filterAndTrim_log[,1],
                           filtered = filterAndTrim_log[,2],
                           denoisedR1 = sapply(dadaR1,
                                               getUniques) %>% sapply(., sum),
                           denoisedR2 = sapply(dadaR2,
                                               getUniques) %>% sapply(., sum),
                           merged = sapply(merged,
                                           getUniques) %>% sapply(., sum),
                           nobim = rowSums(seqtab.nochim),
                           stringsAsFactors = F)
```

[69]:
```
head(track_n_seqs)
```

A data.frame: 6 × 7

	samples <chr>	raw <dbl>	filtered <dbl>	denoisedR1 <int>	denoisedR2 <int>	n <</int>
1_S1_R1_rmprimer.fastq.gz	1_S1	2627	2049	2027	2010	1
13_S13_R1_rmprimer.fastq.gz	13_S13	2615	2056	1872	1922	1
14_S14_R1_rmprimer.fastq.gz	14_S14	2547	2032	1794	1848	1
15_S15_R1_rmprimer.fastq.gz	15_S15	2438	1918	1754	1804	1
16_S16_R1_rmprimer.fastq.gz	16_S16	2298	1826	1824	1824	1
17_S17_R1_rmprimer.fastq.gz	17_S17	2286	1852	1784	1789	1

Pasamos a formato largo.

[70]:
```
track_n_seqs_long <- track_n_seqs %>%
                     pivot_longer(names_to = "var",
                                  values_to = "value",
                                  raw:nobim)
```

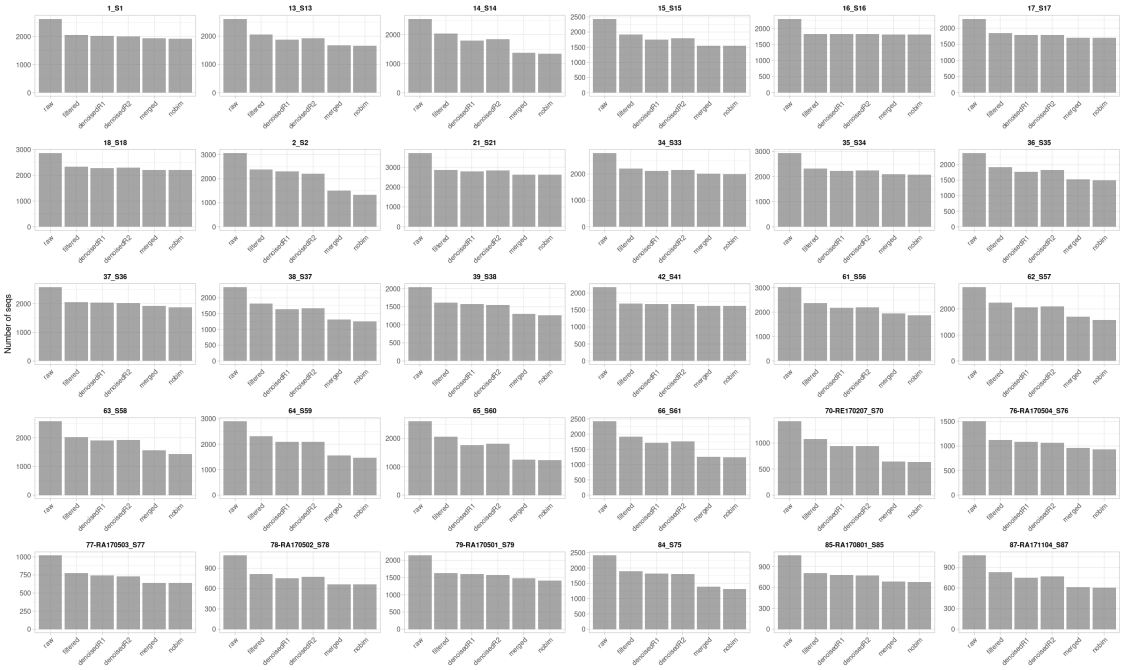
Ordenamos los nombres de las variables de acuerdo al procesamiento de los datos.

[71]:
```
track_n_seqs_long$var <- factor(track_n_seqs_long$var,
                                levels = c("raw","filtered", "denoisedR1",
                                           "denoisedR2", "merged","nobim"))
```

Creamos la visualización.

[72]:
```
nseq_barplots <- ggplot(track_n_seqs_long, aes(x = var, y = value)) +
                 facet_wrap(~ samples, ncol = 6, scales = "free") +
                 geom_bar(stat = "identity", fill = "gray50",
                          alpha = 0.7) +
                 ylab("Number of seqs") +
                 theme_light() +
                 theme(strip.background = element_blank(),
                       strip.text = element_text(color = "black",
                                                 face = "bold"),
                       axis.text.x = element_text(angle = 45, hjust = 1),
                       axis.title.x = element_blank())
```

[73]:
```
options(repr.plot.width=20, repr.plot.height=12)
nseq_barplots
```



Por último, estaremos realizando la anotación taxonómica de los ASVs. Nuevamente, existen varias herramientas que realizan esta tarea, cada una planteando sus argumentos a favor. En esta flujo de trabajo, con la finalidad de seguir trabajando en R, vamos a utilizar la función de DADA2 dedicada a la anotación taxonómica, la cual implementa el *Naive Bayes Classifier*: assignTaxonomy.

Como recordás, este algoritmo consiste en una serie de modelos entrenados para cada género. En este caso estaremos utilizando la base de datos Silva v138 (previamente formateada para su compatibilidad con `assignTaxonomy`).

La base de datos ya la hemos descargado y la puedes encontrar en la siguiente ruta (que vamos a guardar en la variable `TRAIN_DB`).

[74]:
```
TRAIN_DB <- "/home/epereira/tmp/silva_nr99_v138.1_train_set.fa.gz"
```

Ya tenemos todo para correr la anotación taxonómica:

[75]:
```
annot_taxa <- assignTaxonomy(seqs = seqtab.nochim,
                             refFasta = TRAIN_DB,
                             outputBootstraps = T,
                             multithread = 4)
```

Para tener mejor organizados los resultados, podemos cruzar la anotación taxonómica de los ASVs con la matriz de abundancias.

[76]:
```
seqtab_nochim_formatted <- seqtab.nochim %>%
    t %>%
    as.data.frame %>%
```

```
    rownames_to_column("asv")

annot_taxa_formatted <- annot_taxa %>%
    as.data.frame %>%
    rownames_to_column("asv")

seqtab_nochim_taxa <- right_join(x = annot_taxa_formatted,
                                 y = seqtab_nochim_formatted,
                                  by = "asv")
```

Con el comando `head(seqtab_nochim_taxa)` puedes inspeccionar la tabla y ver cómo quedaron organizados los resultados.

[77]:
```
head(seqtab_nochim_taxa)
```

A data.frame: 6 × 43

	asv <chr>
1	GACGAAGGGGGCAAGCGTTGTTCGGAATTACTGGGCGTAAAGCGCGTGTAG
2	TACGAACTGTGCGAACGTTATTCGGAATCACTGGGCTTAAAGGGTGCGTAGG
3	TACGGAGGGTGCAAGCGTTGTCCGGATTTATTGGGTTTAAAGGGTGCGCAGG
4	TACGTAGGGTGCAAGCGTTAATCGGAATTACTGGGCGTAAAGCGTGCGCAGG
5	TACGAAGGTGGCAAGCGTTGTTCGGATTCACTGGGCGTACAGGGAGCGTAG
6	TACGTAGGGTGCAAGCGTTAATCGGAATTACTGGGCGTAAAGCGTGCGCAGG

Esta tabla es el resultado final todo el flujo de trabajo, por lo que vamos a guardarla como archivo csv.

[78]:
```
OUTPUT_DIR <- "./results"
write.csv(x = seqtab_nochim_taxa, file = file.path(OUTPUT_DIR,
                                                   "asv_abund_annot.csv"))
```

Como último ejercicio te proponemos determinar el porcentaje de secuencias que fueron anotadas a nivel de filo y género, con un valor de *bootstrap* mayor a 80%.