

RNA-seq

☰ Expositor	Leonardo Collado Torres
📅 Fecha	@February 1, 2022 → February 4, 2022
👤 Property	Daianna González

0. Generalidades del curso

[Ligas externas de relevancia](#)

[Prerrequisitos](#)

1. Introducción a R y RStudio

[1.1 R](#)

[1.2 GitHub](#)

[1.2.1 Repositorios](#)

[1.3 RStudio](#)

2. Introducción a Bioconductor

[2.1 Paquetes de Bioconductor](#)

[2.2 Estructura de un paquete](#)

[2.3 Las dos ramas de Bioconductor: release y devel](#)

3. Objetos de Bioconductor para datos de expresión

[SummarizedExperiment](#)

[3.1 Ejemplo](#)

[3.2 Ejercicio](#)

[3.3 iSEE](#)

[3.4 Ejercicio con spatialLIBD](#)

4. Datos de RNA-seq a través de `recount3`

[4.1 Usar recount3](#)

[4.2 Ejercicio](#)

[4.3 Información](#)

Diferencia de cálculo `raw_count` y `count`

[5. Bases estadísticas](#)

[Regresión lineal](#)

[Modelos estadísticos en R](#)

[5.1 Ejemplos](#)

Paquete `explorerModelMatrix`

[5.1.1 Ejercicios](#)

[5.2 Datos de SRP045638](#)

[Manejo de datos de RNA-seq](#)

[5.2 Normalización](#)

[Library size normalization](#)

[5.3 Expresión diferencial](#)

[5.4 Visualizando genes DE](#)

Ejercicio:

[6. Ejercicio](#)

[Extras](#)

[R graphics](#)



[Jeff Leek How to be a modern scientist](#)

[R Themes](#)

[Punto de corte](#)

[Paquetes](#)

[Librería purr](#)

[Genefilter](#)

0. Generalidades del curso

Ligas externas de relevancia

- Git del curso: https://lcolladotor.github.io/rnaseq_LCG-UNAM_2022
- Book del curso: [Intro RNA-seq LCG-UNAM 2022 \(lcolladotor.github.io\)](#)
- Canal en Slack: [Slack](#)
- Servidor RStudio LCG

RStudio: Browser Not Supported

Your web browser is not supported by RStudio. RStudio requires one of the following browser versions (or higher):
See the RStudio Platform Support page for more information about browser support in RStudio products.

 <http://132.248.220.108:8787/>

- Canal *YouTube de Leonardo Collado*

Leonardo Collado Torres

Welcome! Here you can find videos from the R/Bioconductor-powered Team Data Science team lead by Leonardo Collado Torres as well as videos from the LIBD rstats club. These videos will mostly involve the
 <https://www.youtube.com/channel/UCxB1IGHLzIEOikTL-aDwqFg>



- Mi repositorio del curso:
- Proyecto final del curso

paurosales/proyecto_rnaseq_2021

Proyecto final del curso de RNA-seq 2021 impartido por Leonardo Collado Torres para los estudiantes de la Licenciatura en Ciencias Genómicas de la UNAM. Repositorio del curso disponible en este

 https://github.com/paurosales/proyecto_rnaseq_2021



Prerrequisitos

Instalación de paquetes en **RStudio** versión 4.0 instalada con [CRAN](#)



Durante el curso se uso el [ambiente de la LCG](#) para no descargarlo en *Windows*

Código para la instalación de paquetes de **Bioconductor** versión 1.0.1 necesarios (disponible en el git):

```
## For installing Bioconductor packages
if (!requireNamespace("BiocManager", quietly = TRUE)) {
    install.packages("BiocManager")
}

## Install required packages
BiocManager::install(
    c(
```

```

"usethis", ## Utilities
"here",
"biocthis",
"postcards",
"pryr",
"sessioninfo",

"SummarizedExperiment", ## Main containers / vis
"iSEE",

"ExploreModelMatrix", ## RNA-seq
"limma",
"recount3",

"pheatmap", ## Visualization
"ggplot2",
"patchwork",
"RColorBrewer",

"spatialLIBD" ## Advanced
)
)

```

Material del curso en lcolladotor.github.io/rnaseq_LCG-UNAM_2021 o con el siguiente código:

```
usethis::use_course('lcolladotor/rnaseq_LCG-UNAM_2022')
```

1. Introducción a R y RStudio

1.1 R

R: es gratis, de acceso libre, utilizado para muchos campos de trabajo, fuerte en la bioinformática a través de **Bioconductor** (repositorio através del cual se comparte código de R bajo ciertos estándares y revisiones) para hacer código que analice datos de secuenciación masiva de RNA (RNAseq).

1.2 GitHub

- Permite compartir código
- Se complementa con Git que es para tener un control de versiones de tu código
- Puedes tener páginas web estáticas (*GitHub pages*) que toma archivos HTML, JavaScript, CSS directamente desde un repositorio en GitHub

1.2.1 Repositorios

Repositorio de la clase en GitHub desde RStudio

```
## Crear proyecto de R
usethis::create_project("~/rnaseq_2022_notas")
## Archivo inicial
usethis::use_r("01-notas.R")
## Creamos el archivo R/02-visualizar-mtcars.R
usethis::use_r("02-visualizar-mtcars.R")

#Contenido del archivo 02
## Se cargan paquetes al inicio
library("sessioninfo")
library("here") ## Para compartir código relativo
library("ggplot2")

## Hello world
print("Soy Daianna")

## Directorios se crean para el proyecto
dir_plots <- here::here("figuras")
dir_rdata <- here::here("processed-data")

## Crear directorio para las figuras y archivos
dir.create(dir_plots, showWarnings = TRUE) # Si existe da error
dir.create(dir_rdata, showWarnings = FALSE) # No da error(no sobreescribe)

## Hacer una imagen de ejemplo: dirección, nombre
pdf(file.path(dir_plots, "mtcars_gear_vs_mpg.pdf"),
    useDingbats = FALSE
)
## Se hace una gráfica con los datos, variables, tipo de gráfica
ggplot(mtcars, aes(group = gear, y = mpg)) +
  geom_boxplot()
dev.off()

## Guardar datos
save(mtcars, file=file.path(dir_rdata, "mtcars.Rdata"))

## Para reproducir mi código en 120 char
options(width = 120)
## Mostrar la información para reproducir los resultados
sessioninfo::session_info()

# - Session info -----
#   setting value
# version R version 4.1.1 (2021-08-10)
# os       Windows 10 x64 (build 19042)
# system  x86_64, mingw32
# ui      RStudio
# language (EN)
# collate Spanish_Mexico.1252
# ctype   Spanish_Mexico.1252
# tz     America/Mexico_City
# date   2022-02-01
```

```

# rstudio 1.2.5001 (desktop)
# pandoc NA
#
# - Packages -----
#   package      * version date (UTC) lib source
# cli           3.1.1   2022-01-20 [1] CRAN (R 4.1.2)
# crayon        1.4.2   2021-10-29 [1] CRAN (R 4.1.2)
# ellipsis      0.3.2   2021-04-29 [1] CRAN (R 4.1.2)
# fansi          1.0.2   2022-01-14 [1] CRAN (R 4.1.2)
# fs             1.5.2   2021-12-08 [1] CRAN (R 4.1.2)
# glue           1.6.1   2022-01-22 [1] CRAN (R 4.1.2)
# here            1.0.1   2020-12-13 [1] CRAN (R 4.1.2)
# lifecycle      1.0.1   2021-09-24 [1] CRAN (R 4.1.2)
# magrittr       2.0.2   2022-01-26 [1] CRAN (R 4.1.2)
# pillar          1.6.5   2022-01-25 [1] CRAN (R 4.1.2)
# pkgconfig      2.0.3   2019-09-22 [1] CRAN (R 4.1.2)
# purrr          0.3.4   2020-04-17 [1] CRAN (R 4.1.2)
# rlang           1.0.0   2022-01-26 [1] CRAN (R 4.1.2)
# rprojroot       2.0.2   2020-11-15 [1] CRAN (R 4.1.2)
# rstudioapi      0.13    2020-11-12 [1] CRAN (R 4.1.2)
# sessioninfo     1.2.2   2021-12-06 [1] CRAN (R 4.1.2)
# tibble          3.1.6   2021-11-07 [1] CRAN (R 4.1.2)
# usethis          2.1.5   2021-12-09 [1] CRAN (R 4.1.2)
# utf8            1.2.2   2021-07-24 [1] CRAN (R 4.1.2)
# vctrs           0.3.8   2021-04-29 [1] CRAN (R 4.1.2)
#
# [1] C:/Users/hp/Documents/R/win-library/4.1
# [2] C:/Program Files/R/R-4.1.1/library
#

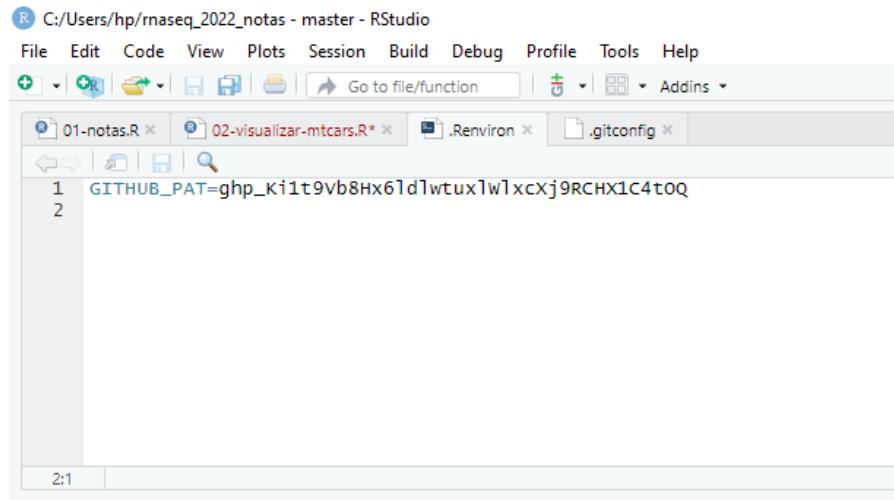
```

Conecitar computadora con GitHub

```

## Para poder conectar tu compu con GitHub
usethis::create_github_token()
## Abrirá una página web, escoje un nombre único para el token, generararlo y copiarlo
# Pegarlo
gitcreds::gitcreds_set()
# Editar el archivo Renvironment guardando el token como GITHUB_PAT=TU_CLAVE_DE_40_LETRAS
usethis::edit_r_environ()

```



```
## Configura tu usuario de GitHub
usethis::edit_git_config()
## Para inicializar el repositorio de Git
usethis::use_git()
## Para conectar tu repositorio local de Git con los servidores de GitHub
usethis::use_github()
```



Link al repo:

Página personal

Crear un repositorio [user name].github.io y clonarlo:

```
git clone https://github.com/daianna21/daianna21.github.io
```

Instalar los paquetes de *postcards* en RStudio:

```
install.packages("postcards")
```

Ir a File → New Project → New Directory → Postcards Website, nombrarlo y elegir template. Cargar foto en el proyecto y editar index.rmd, añadirlo en el repositorio.

```
cp webpage/* daianna21.github.io/
```

Hacer git add, commit y push.



Link de la webpage: [Daianna González Padilla \(daianna21.github.io\)](https://daianna21.github.io)

1.3 RStudio

- RStudio Desktop es gratis <http://www.rstudio.com/products/rstudio/download/preview/>
- Nos ayuda a realizar muchas cosas con R de forma más rápida
- Es actualizado con bastante frecuencia
- RStudio cheatsheets: <https://www.rstudio.com/resources/cheatsheets/>
 - <https://github.com/rstudio/cheatsheets/raw/master/rstudio-ide.pdf>
- RStudio projects: usalos para organizar tu código de un proyecto y ligarlo a un repositorio de github
 - https://github.com/ComunidadBioInfo/cdsb2020/blob/master/presentaciones_flujos-de-trabajo/Trabajando-con-proyectos.pdf

Con el siguiente código se puede configurar un perfil personal para RStudio cada que se abra la sesión:

```
remotes::install_github(c(  
    "gadenbuie/rsthemes"))  
remotes::install_cran("suncalc")  
rsthemes::install_rsthemes(include_base16 = TRUE)  
  
usethis::edit_r_profile()
```

Atajos:

```
# Seleccionar  
Ctrl+A  
# Comentar código seleccionado  
Ctrl+Shift+C  
# Para ver comandos usados tales iniciales  
[Primeras 3 letras d eun comando] Ctrl+↑  
# Para buscar archivos  
Ctrl+. 
```

2. Introducción a *Bioconductor*

Repositorio de paquetes de R relacionados con el análisis y comprensión de datos genómicos masivos o *high throughput*.

Repositorio abierto y comunitario de paquetes de R para el análisis de datos genómicos. Usa **R como lenguaje base**. Desarrollo colaborativo (importancia de la comunidad científica).

2.1 Paquetes de Bioconductor

▼ Tipos de paquetes de Bioconductor

Son 4 árboles principales: software, annotation, experiment, workflow

Dentro de cada árbol, un paquete puede ser parte de varias ramas

- **Software:** tipo de paquete principal; la mayoría los hacen usuarios aunque algunos los hacen gente pagada directamente por Bioconductor:
<http://bioconductor.org/packages/release/bioc/>
- **Annotation:** facilitan el interactuar con bases de datos de anotación:
<http://bioconductor.org/packages/release/data/annotation/>
- **Experiment Data:** contienen datos para algún artículo o datos que se usan en ejemplos más exhaustivos: <http://bioconductor.org/packages/release/data/experiment/>
- **Workflows:** demuestran como puedes usar varios paquetes de Bioconductor para ciertos tipos de análisis: <http://bioconductor.org/packages/release/workflows/>

2.2 Estructura de un paquete

- Usa https://bioconductor.org/packages/<pkg_name>
 - Contiene título, información como no. dependencias y descargas: las Badges (etiquetas): rápidamente podemos evaluar como está
 - Parráfo de descripción del paquete
 - Lista de autores, artículos asociados
 - Instalación
- #### ▼ Documentación
- Una liga por cada vignette en formato PDF o HTML. Es la documentación **principal!**

- Una vignette es donde los autores del paquete explican cómo usar las diferentes funciones del paquete y en qué orden

▼ Detalles

- Términos de `biocViews` (palabras clave del árbol), son necesarias en Bioconductor
- Cómo se relaciona a otros paquetes (depends, imports, linking to, suggests, depends on me, ...)
- URL: donde puedes encontrar el código fuente
- BugReports: donde puedes pedir ayuda

▼ Más detalles sobre el paquete

- Estadísticas de descargas

 Ejercicio 1: Bioconductor

2.3 Las dos ramas de Bioconductor: release y devel

▼ Dos ramas

- `release`, actualmente 3.14
- `devel`, actualmente 3.15 : versiones de desarrollo

3. Objetos de Bioconductor para datos de expresión

SummerizedExperiment

Contención de datos en una arreglo particular, principalmente compuesto por tres tablas:

- colData: info de una muestra
se\$ es sinónimo de colData(se)
- rowRanges: Genomic Ranges (objeto de clase GRanges)
- assay(s): cuentas o #reads para cada gen

Se pueden agregar datos extra como datos de los experimentos (metaData/exptData)



metaData es el que se utiliza en código

colData

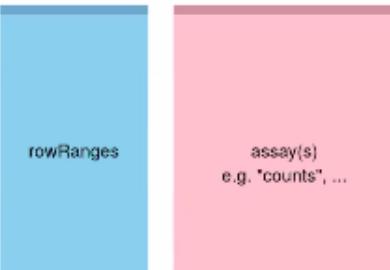
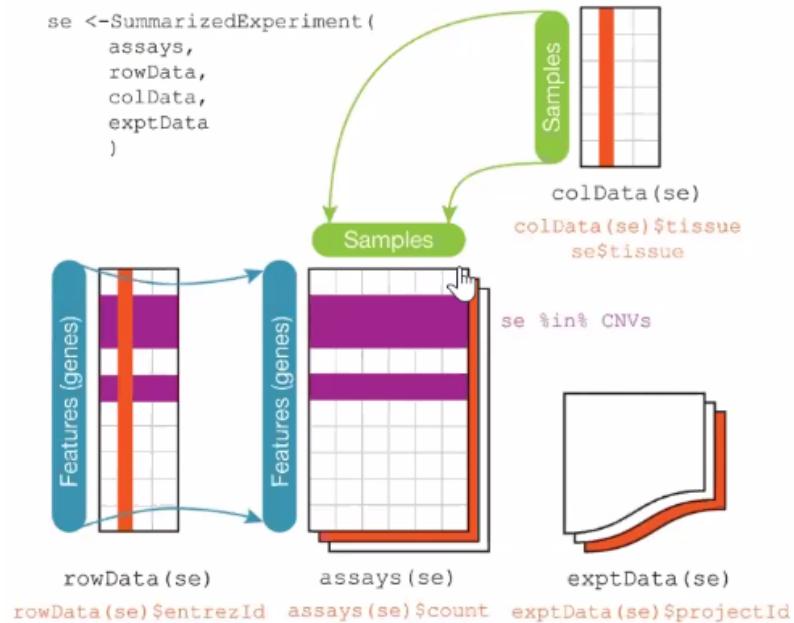


Figure 2: The integrative data container *SummarizedExperiment*.



Obtenida del artículo original de *Bioconductor*

En exptData(se) se puede almacenar cualquier información y con metadata(se) acceder a metadatos como fechas, autores del experimento

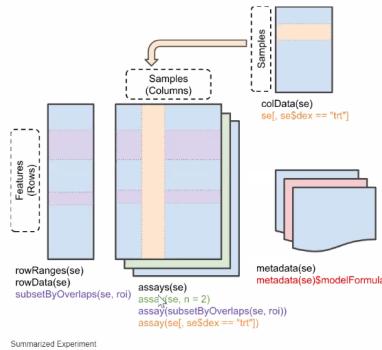
Rows son los genes

Cols son muestras

Assays son el número de tablas

```

> sce
class: SingleCellExperiment
dim: 33538 47681
metadata(1): image
assays(2): counts logcounts
rownames(33538): ENSG00000243485 ENSG00000237613 ...
ENSG00000277475 ENSG00000268674
rowData names(9): source type ... gene_search is_top_hvg
colnames(47681): AAACACCGAATAGTTC-1 AAACAAGTATCTCCA-1 ...
TTGTTTGTAAATTTC-1
colData names(73): barcode sample_name ... pseudobulk_UMAP_spatial markers_UMAP_spatial
reducedDimNames(6): PCA TSNE_perplexity50 ... TSNE_perplexity80 UMAP_neighbors15
altExpNames(0):
  
```



Resultado del objeto sce notas disponibles en le repositorio de notas del curso

GenomicRanges

Paquete que define la clase que describe la información de los genes

Permite acceder a regiones del genoma específicas

```

gr <- GRanges(
  seqnames = Rle(c("chr1", "chr2", "chr1", "chr3"), c(1, 3, 2, 4)),
  ranges = IRanges(101:110, end = 111:120, names = head(letters, 10)),
  strand = Rle(strand(c("-", "+", "*", "+", "-")), c(1, 2, 2, 3, 2)),
  score = 1:10,
  GC = seq(1, 0, length=10))
gr

## GRanges object with 10 ranges and 2 metadata columns:
##   seqnames      ranges strand |  score      GC
##   <Rle> <IRanges> <Rle> | <integer> <numeric>
##   a     chr1    101-111 - |      1  1.000000
##   b     chr2    102-112 + |      2  0.888889
##   c     chr2    103-113 + |      3  0.777778
##   d     chr2    104-114 * |      4  0.666667
##   e     chr1    105-115 * |      5  0.555556
##   f     chr1    106-116 + |      6  0.444444
##   g     chr3    107-117 + |      7  0.333333
##   h     chr3    108-118 + |      8  0.222222
##   i     chr3    109-119 - |      9  0.111111
##   j     chr3    110-120 - |     10  0.000000
##   -----
##   seqinfo: 3 sequences from an unspecified genome; no seqlengths

```

`Rle`: Run Length Encoding, objeto para almacenar la información con repeticiones (con vectores)

`IRanges`: Interval Ranges, toma posiciones de inicio y fin de los genes y nombres de los genes
`score` y `GC` son opcionales

3.1 Ejemplo

```

## Lets build our first SummarizedExperiment object
## Cargar el paquete de se
library("SummarizedExperiment")
## ?SummarizedExperiment
## Creamos los datos para nuestro objeto de tipo SummarizedExperiment para 200 genes a lo largo
## de 6 muestras
nrows <- 200
ncols <- 6
## Fijar la semilla para generar números aleatorios para que sea reproducible
set.seed(20210223)
# Generar los assays: 1200 datos en 200 renglones con ~uniforme de rango 1 a 10000
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)

## Información de nuestros genes
rowRanges <- GRanges(
  # 50 del chr1 y 150 del chr2 (seqnames)
  rep(c("chr1", "chr2"), c(50, 150)),
  # Ranges en pb (enteros) o 200 posiciones de inicio y fin
  IRanges(floor(runif(200, 1e5, 1e6)), width = 100),
  # Cadena de los genes
  strand = sample(c("+", "-"), 200, TRUE),
  #Id de los genes con 3 dígitos entre 1 y 200
  feature_id = sprintf("ID%03d", 1:200)
)
## Ver info de los genes
rowRanges

# Nombre (número) de los genes
names(rowRanges) <- paste0("gene_", seq_len(length(rowRanges)))

## Información de nuestras 6 muestras en las columnas
colData <- DataFrame(
  Treatment = rep(c("ChIP", "Input"), 3),
  # Nombres de las columnas: A, B, C, D, E, F
  row.names = LETTERS[1:6]
)
## Juntamos ahora toda la información en un solo objeto se de R
rse <- SummarizedExperiment(
  assays = SimpleList(counts = counts),
  rowRanges = rowRanges,
  colData = colData
)

## Exploraremos el objeto resultante
rse
## Número de genes y muestras
dim(rse)
# [1] 200   6
## IDs de nuestros genes y muestras
dimnames(rse)

## Nombres de tablas de cuentas que tenemos (RPKM, CPM, counts, logcounts, etc)
assayNames(rse)
## [1] "counts"

## El inicio de nuestra tabla de cuentas

```

```

## Assay es para recuperar 1 sola tabla assay(rse, "logcounts")
head(assay(rse))

## Información de los genes en un objeto de Bioconductor
rowRanges(rse)

## Tabla con información de los genes
rowData(rse) # es idéntico a 'mcols(rowRanges(rse))'

## Tabla con información de las muestras -> da un DataFrame que imprime los primeros y últimos r
englones de la tabla, data.frame todos
colData(rse)

```

3.2 Ejercicio

```

## Comando 1: solo tomar los primeros dos renglones/genes
rse[1:2, ]

## Comando 2: tomar solo las columnas A, D y F
rse[, c("A", "D", "F")]

## Vector lógico de las muestras que tienen un tratamiento Input
rse[, rse$treatment=='Input']
## Ver qué muestras son
which(rse$treatment=='Input')

```

3.3 iSEE

Interactive SE Explorer

Muestra de manera interactiva la información de un objeto `SingleCellExperiment` o `SummarizedExperiment`

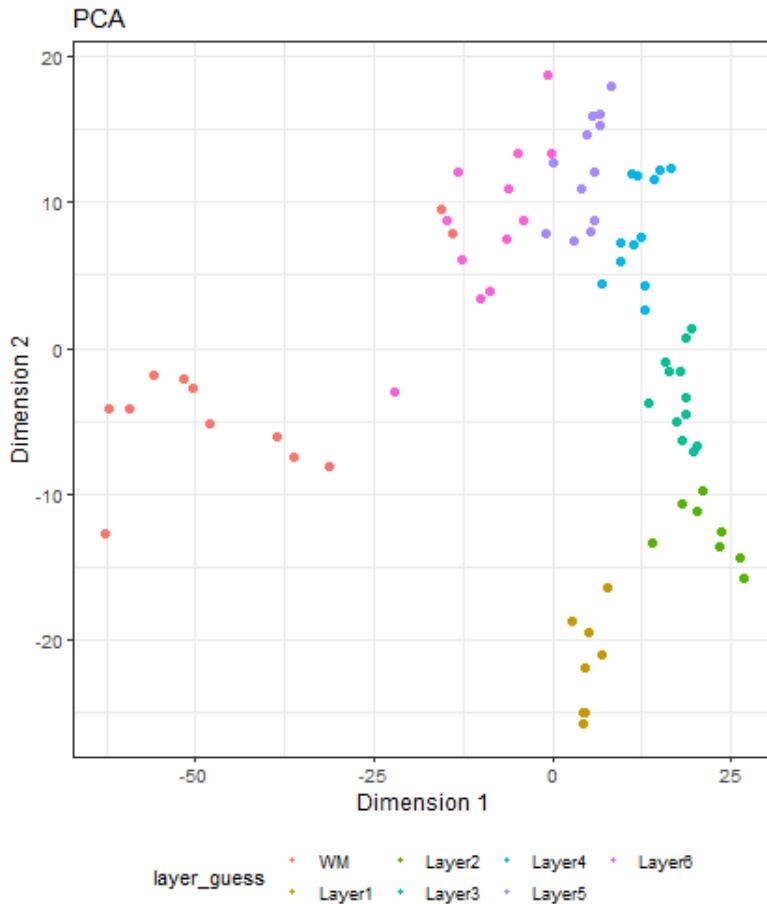
```

## Explora el objeto rse de forma interactiva
library("iSEE")
iSEE::iSEE(rse)

```

3.4 Ejercicio con spatialLIBD

`SingleCellExperiment` contiene una cuarta tabla de `reducedDims` (aparece un PCA plot en iSEE) y `rows=cell` o `gene`



```
## Descarguemos unos datos de spatialLIBD de SingleCellExperiment para 7 capas (columnas) del cerebro y rows=genes
sce_layer <- spatialLIBD::fetch_data("sce_layer")
head(assay(rse))
# Explorar los datos
iSEE::iSEE(sce_layer)
```

- Explora en con un *heatmap* la expresión de los genes `MOBP`, `MBP` y `PCP4`. Si hacemos un *clustering* (agrupamos los genes), ¿cuáles genes se parecen más? 14 y 71 (se agrupan juntos en cluster)

```
# ENSG00000168314
assay(sce_layer)[ "ENSG00000168314", ]
iSEE::iSEE(sce_layer[ "ENSG00000168314", ])

# ENSG00000183036
assay(sce_layer)[ "ENSG00000183036", ]
```

```

iSEE::iSEE(sce_layer["ENSG00000183036",])

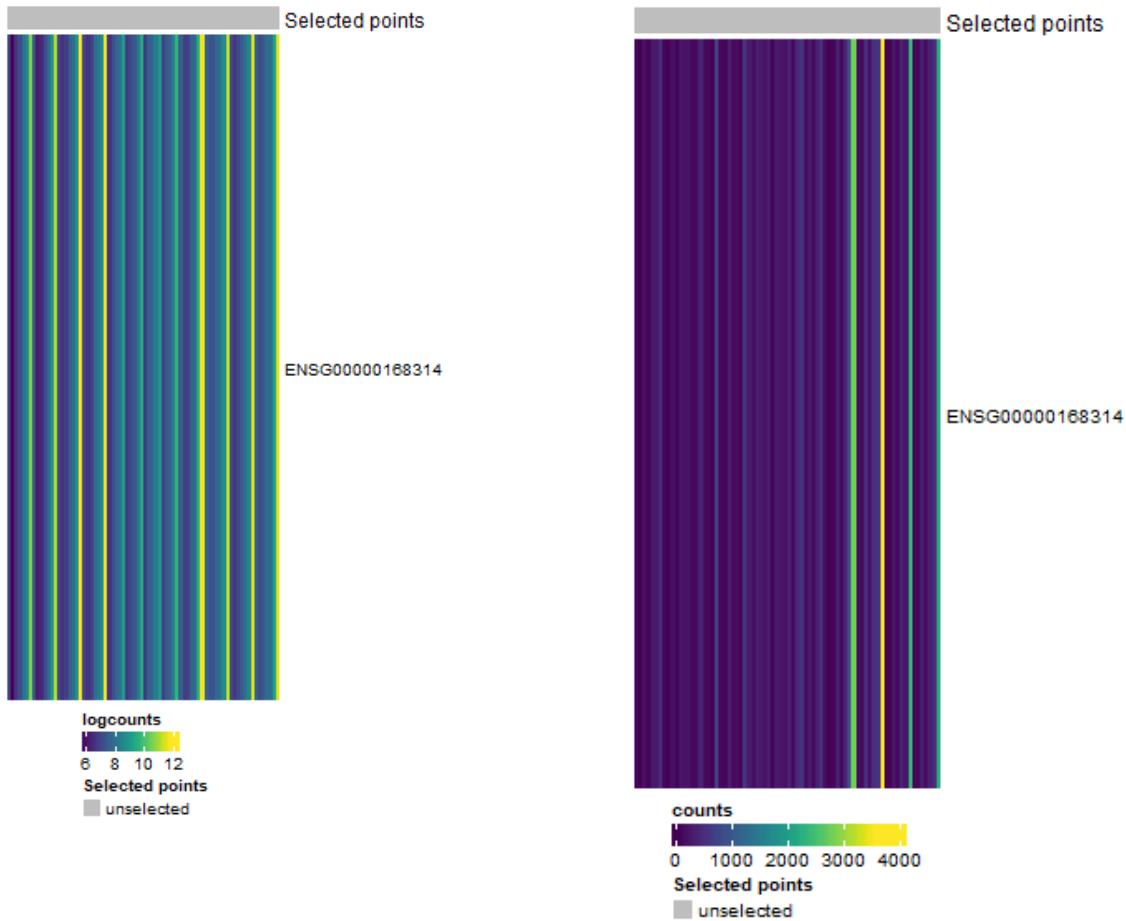
# ENSG00000197971
assay(sce_layer)["ENSG00000197971",]
iSEE::iSEE(sce_layer["ENSG00000197971",])

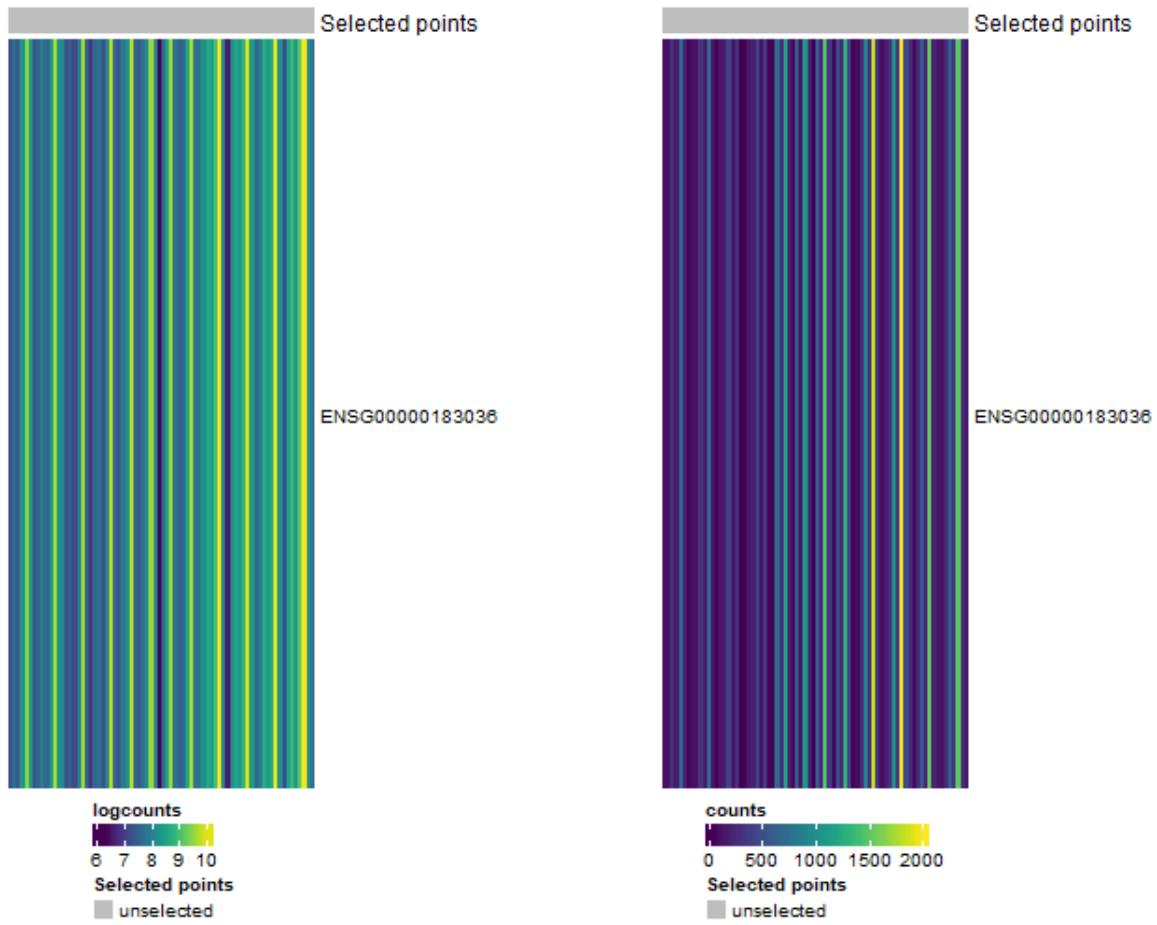
# Clustering
heatmap(assay(sce_layer[c("ENSG00000168314", "ENSG00000197971", "ENSG00000183036"), ]))

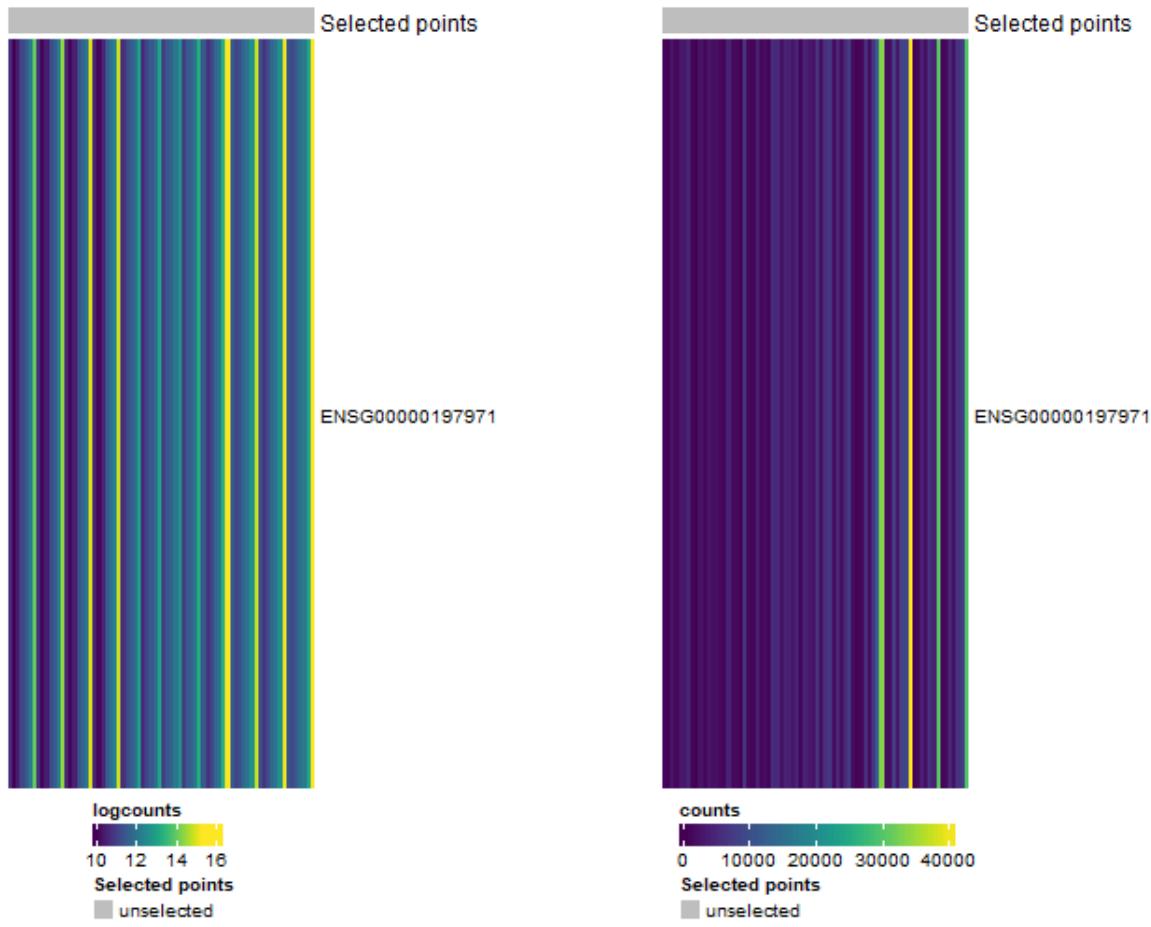
# Mayor expresión
which.max(assay(sce_layer["ENSG00000168314"]))
# [1] 62
colnames(assay(sce_layer["ENSG00000168314"]))[62]
# [1] "151674_WM"
## Obtener índices de las capas según la expresión del gen de manera descendiente
order(assay(sce_layer["ENSG00000168314"])), decreasing = TRUE)

```

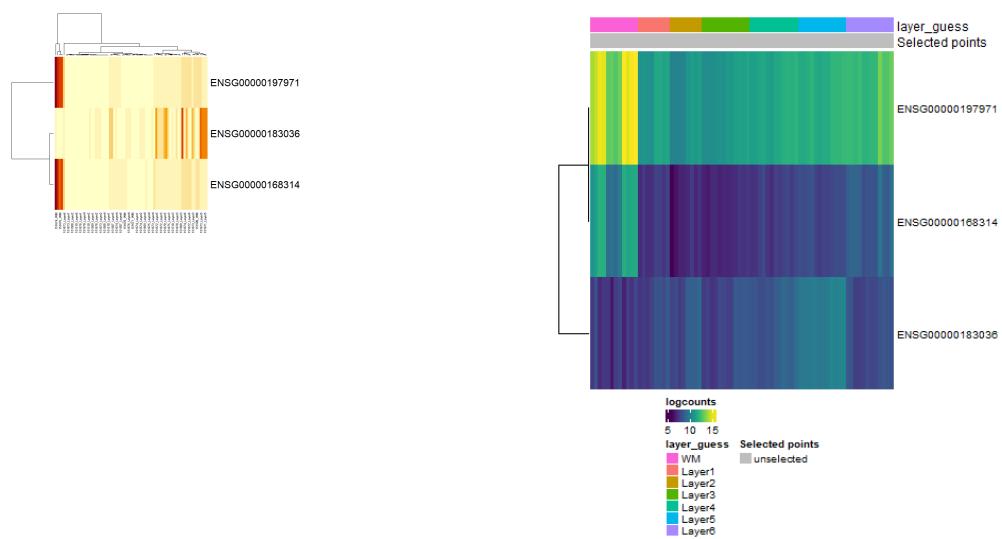
Resultados:



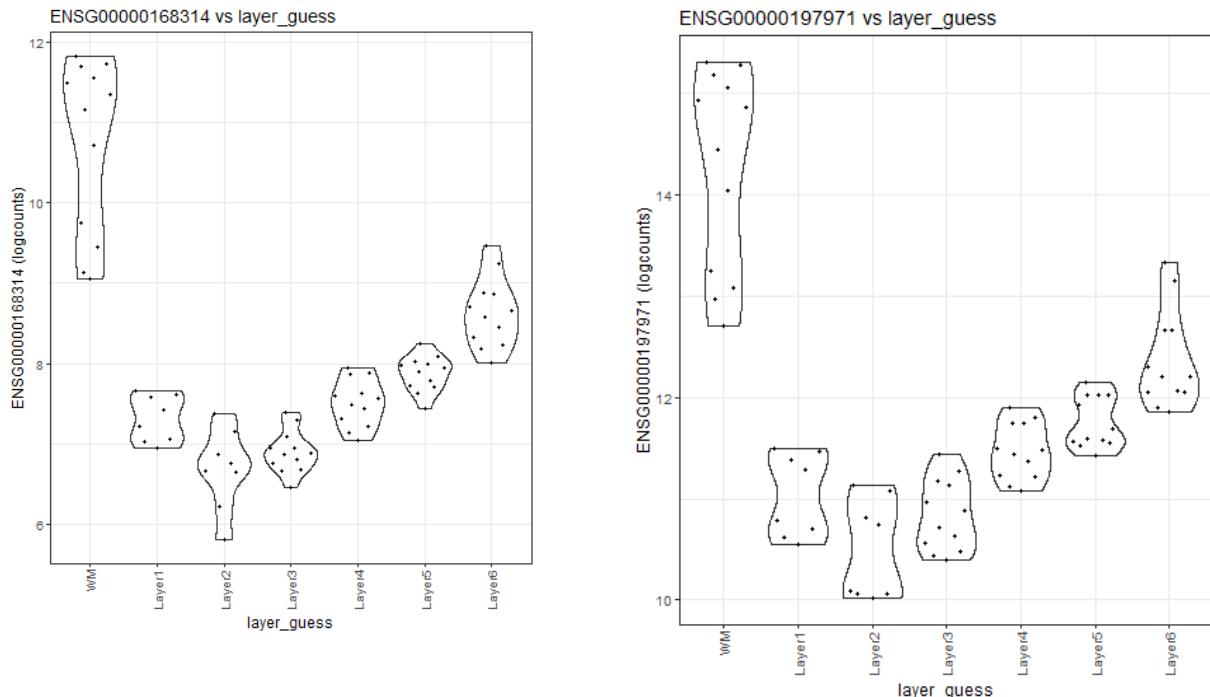




Clustering:



- ¿En qué capas se expresan más los genes *MOBP* y *MBP*?



4. Datos de RNA-seq a través de `recount3`

Es un proyecto que ha tenido 3 fases y contiene datos de RNA-seq para estudios de diversa índole.

4.1 Usar `recount3`

Primero cargamos el paquete de R que automáticamente carga todas las dependencias incluyendo a `SummarizedExperiment` o con `recount3: uniformly_processed RNA-seq`

```
## Load recount3 R package
library("recount3")
```

Después tenemos que identificar un estudio de interés y determinar si queremos accesar la información a nivel de genes, exones, etc. Sabiendo el estudio de interés, podemos descargar los datos usando la función `create_rse()` como mostramos a continuación. `create_rse()` tiene argumentos con los cuales podemos especificar la **anotación** que queremos usar (las opciones dependen del organismo).

```

## Revisemos todos los proyectos con datos de humano en recount3
## Por default da los de humano
human_projects <- available_projects()
## Encuentra tu proyecto de interés. Aquí usaremos
## SRP009615 de ejemplo, buscálo en el Study Explorer
proj_info <- subset(
  human_projects,
  project == "SRP009615" & project_type == "data_sources"
)
## Crea un objeto de tipo RangedSummarizedExperiment (RSE)
## con la información a nivel de genes
rse_gene_SRP009615 <- create_rse(proj_info)

## Explora el objeto RSE
rse_gene_SRP009615

```

Una vez que tenemos las cuentas, podemos

usar `transform_counts()` o `compute_read_counts()` para convertir en los formatos esperados por otras herramientas.

```

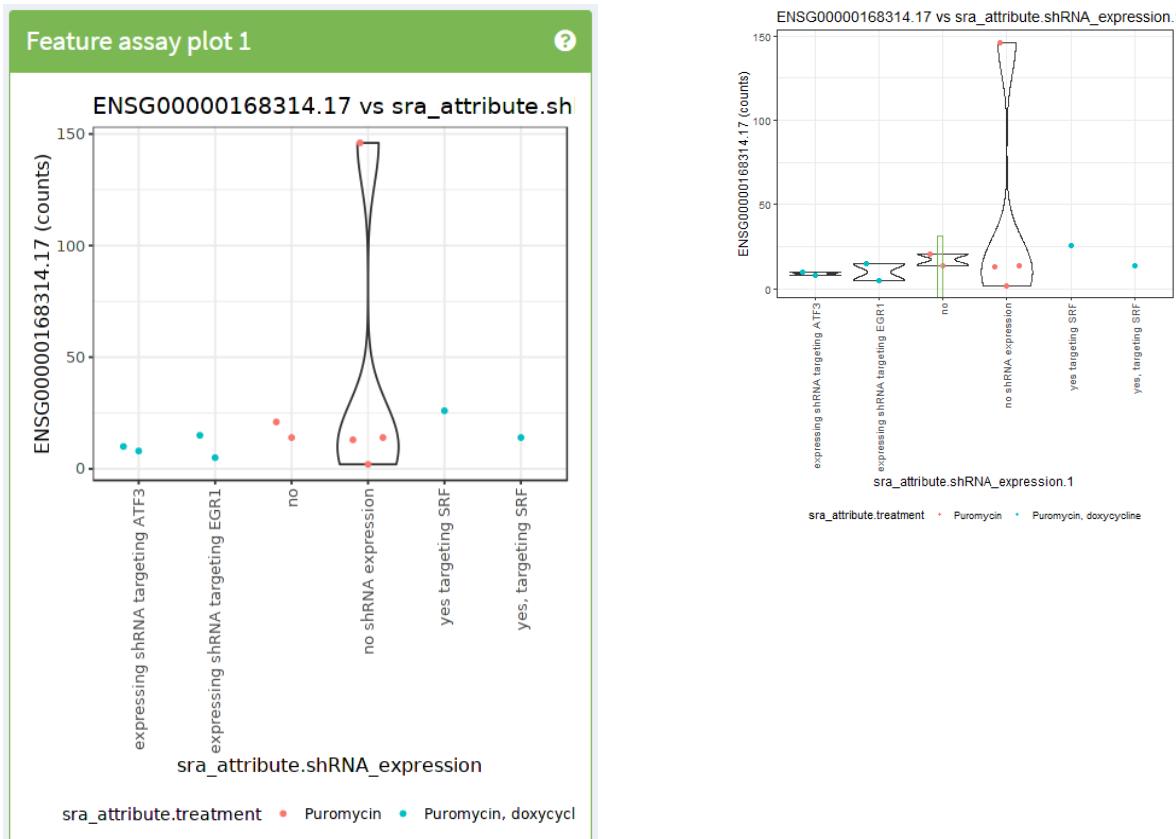
## Convirtamos las cuentas por nucleotido a cuentas por lectura
## usando compute_read_counts() o sea de raw-counts a counts
assay(rse_gene_SRP009615, "counts") <- compute_read_counts(rse_gene_SRP009615)

## Para este estudio en específico, hagamos más fácil de usar la
## información del experimento, se agregan columnas: Cells, shRNA expression. source name y trea
tment
rse_gene_SRP009615 <- expand_sra_attributes(rse_gene_SRP009615)
colData(rse_gene_SRP009615)[
  ,
  grepl("^sra_attribute", colnames(colData(rse_gene_SRP009615)))
]

```

4.2 Ejercicio

Utiliza `iSEE` para reproducir la siguiente imagen



4.3 Información

Procesamiento uniforme de datos públicos RNA-seq, permite:

- "Democratización de datos"
- Acceso para cualquier persona que quiera, independientemente de su acceso a clústers o *high-performance*
- Comparación de datos de expresión a partir de el análisis de las mismas herramientas
- Fácil de usar
- Datos de humano y ratón

Fase 1 (2011) `ReCount`

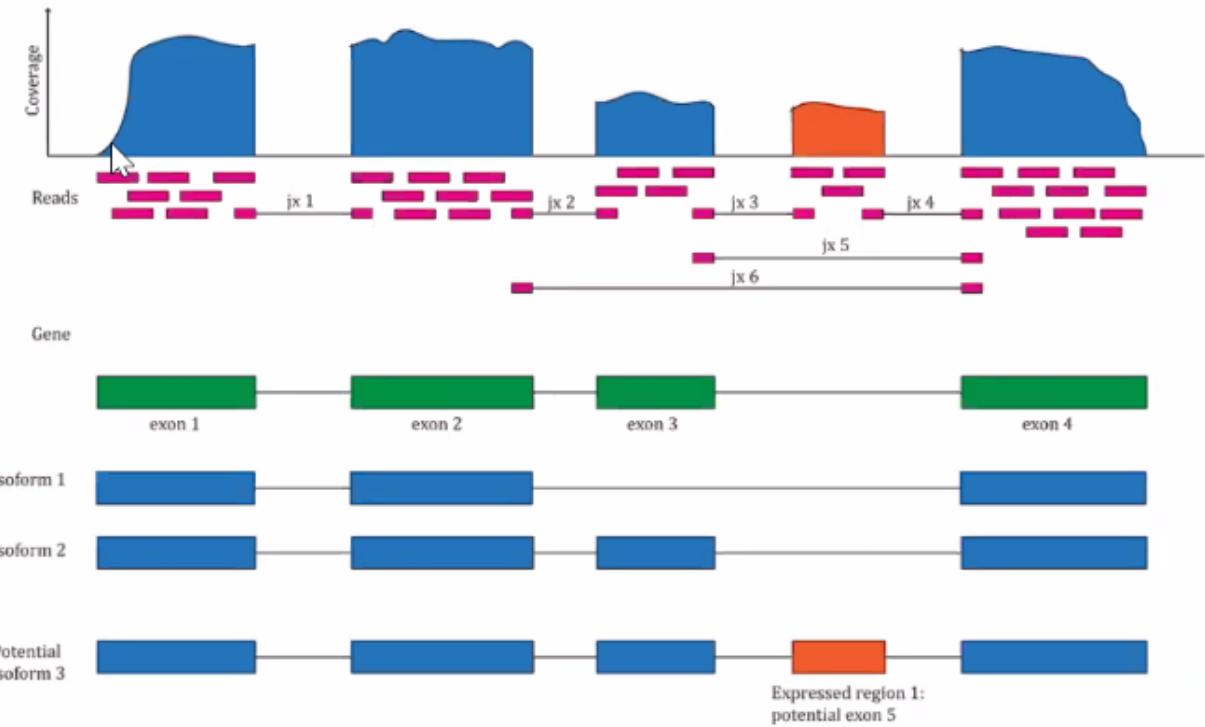
20 muestras

Fase 2 (2017) `recount2`

70 mil muestras

Fase 3 (en aprobación) `recount3`

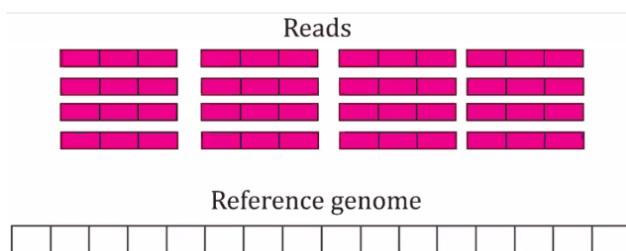
700 mil muestras

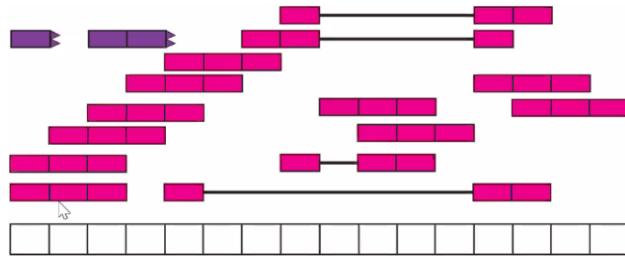


Diferencia de cálculo `raw_count` y `count`

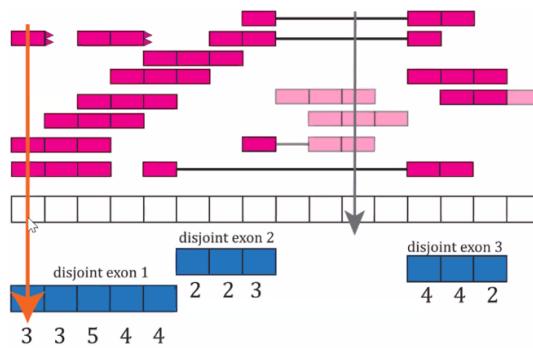
Procedimiento llevado a cabo por `recount`

Comprimen la información y se calcula la covertura de cada base.

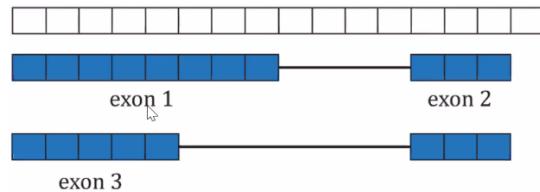




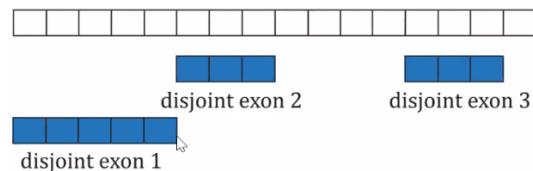
Cuantificar las lecturas para cada base de los exones del genoma, contando lecturas cortadas

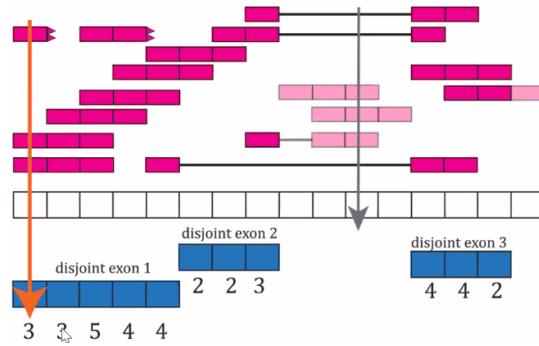


La notación indica cuáles son las coordenadas de los genes en las lecturas

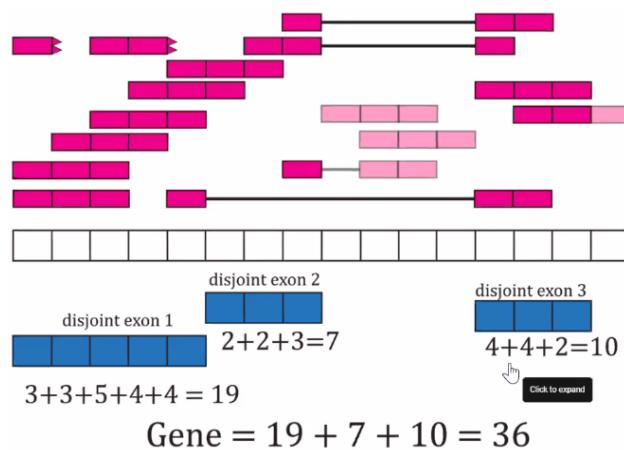


Encuentran las regiones diferentes de exones disjuntos (no sobrelapantes)





Calcular valores para cada segmento exónico → gene

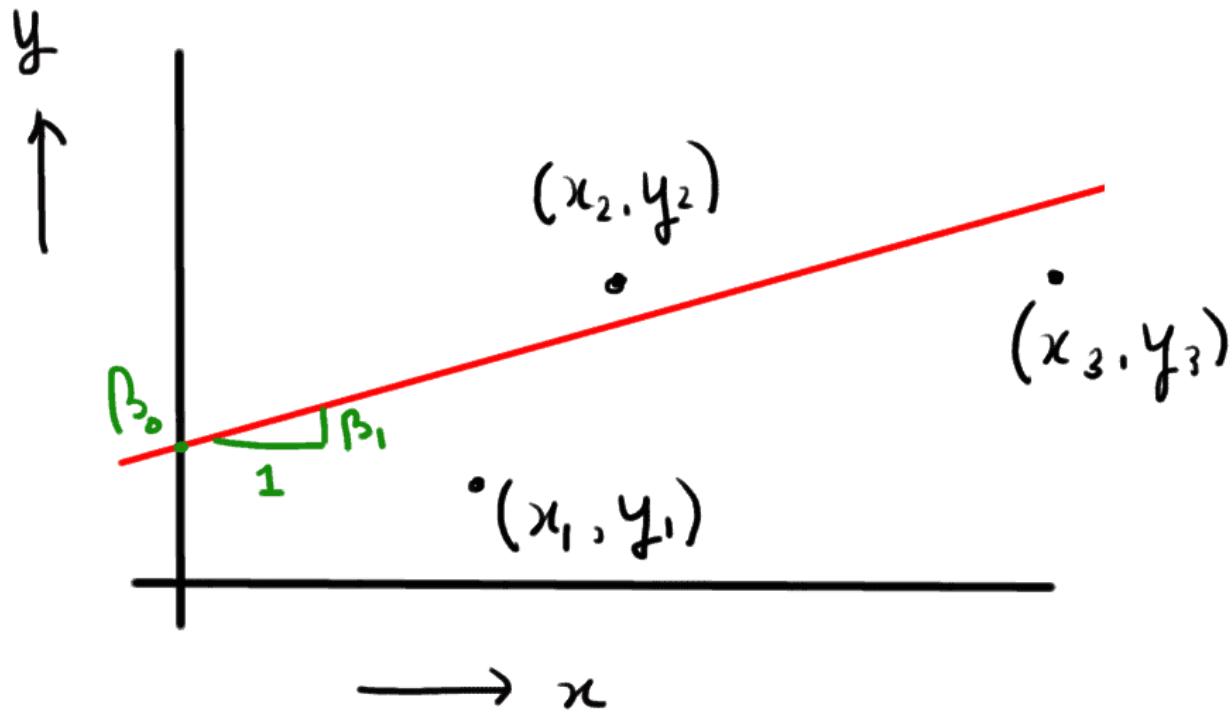


El 36 es el valor de `raw_count` que `recount` calcula por **base**. NO el valor de **lecturas sobrelapantes** que normalmente se calcula como `count`

5. Bases estadísticas

Para análisis de expresión diferencial, si tenemos n genes, se hacen n regresiones lineales y se extrae información de 1 sola variable o parámetro. Y son los datos de expresión

Regresión lineal



$$Y = \beta_0 - \beta_1 X + \epsilon$$

con

$$\epsilon \sim N(0, \sigma^2)$$

$Y \rightarrow$ response variable

$X \rightarrow$ explanatory variable

β_i es el cambio promedio en Y con respecto a X

$\epsilon \rightarrow$ corresponde al **término de residuos**, que representa la diferencia entre el valor observado y el estimado

First, notice that our linear model is $E[Y] = \beta_0 + \beta_1 X$, then we provide two values for X : little x and little x plus one, and compute the difference between them:

$$E[Y|X=x+1] - E[Y|X=x]$$

$$E[Y|X=x+1] - E[Y|X=x]$$

$$= (\beta_0 + \beta_1(x+1)) - (\beta_0 + \beta_1x)$$

$$\begin{aligned}
 &= (\beta_0 + \beta_1(x+1)) - (\beta_0 + \beta_1x) \\
 &= \beta_0 + \beta_1x + \beta_1 - \beta_0 - \beta_1x \\
 &= \beta_0 + \beta_1x + \beta_1 - \beta_0 - \beta_1x \\
 &= (\beta_0 - \beta_0) + (\beta_1x - \beta_1x) + \beta_1 \\
 &= (\beta_0 - \beta_0) + (\beta_1x - \beta_1x) + \beta_1 \\
 &= \beta_1
 \end{aligned}$$

Modelos estadísticos en R

Con `model.matrix()`

Funciona con **variables categóricas y numéricas**.

Las variables **categóricas** las convierte o interpreta como *dummy variables* o variables indicativas (binarias), o sea que las convierte en 0s y 1s.

$$y_i = \beta_0 + \beta_1 Z_i + e_i$$

where:

- y_i is outcome score of i^{th} unit,
- β_0 is coefficient for the *intercept*,
- β_1 is coefficient for the *slope*,
- Z_i is:
 - 1 if the i^{th} unit is in the treatment group;
 - 0 if the i^{th} unit is in the control group;
- e_i is residual for the i^{th} unit.

$$y_i = \beta_0 + \beta_1 Z_i + e_i$$

First, determine effect for each group:

For control group ($Z_i = 0$):

$$y_C = \beta_0 + \beta_1(0) + 0$$

$$y_C = \beta_0$$

e_i averages to 0 across the group

For treatment group ($Z_i = 1$):

$$y_T = \beta_0 + \beta_1(1) + 0$$

$$y_T = \beta_0 + \beta_1$$

- Con R, usamos mucho la función `model.matrix()` y la sintaxis de fórmula `Y ~ X1 + X2` tal como en el siguiente ejemplo.

```
## ?model.matrix
## model.matrix(log(Y) ~ log(X_1) * log(X_2)) para variables X dependientes
## model.matrix(log(Y) ~ log(X_1) + log(X_2)) para variables X independientes
## with() evalua una expresión en datos de un df
mat <- with(trees, model.matrix(log(Volume) ~ log(Height) + log(Girth)))
mat
colnames(mat)
## Modelo lineal creado para una fórmula dada de la forma Y~X1+X2
## Se visualizan los datos con summary
lm(log(Volume) ~ log(Height) + log(Girth), data = trees)
summary(lm(log(Volume) ~ log(Height) + log(Girth), data = trees))
```

Coefficients:					
	β estimado	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-6.63162	0.79979	-8.292	5.06e-09 ***	
log(Height)	1.11712	0.20444	5.464	7.81e-06 ***	
log(Girth)	1.98265	0.07501	26.432	< 2e-16 ***	

```
##      (Intercept) log(Height) log(Girth)
## 1          1     4.248495   2.116256
## 2          1     4.174387   2.151762
## 3          1     4.143135   2.174752
```

Columnas:

`Intercept` es el β_0 o nivel basal con X y $Y = 0$

`log(X)` es el β_1 de la variable X

```
## 4      1    4.276666  2.351375
## 5      1    4.394449  2.370244
```

5.1 Ejemplos

Paquete `explorerModelMatrix`

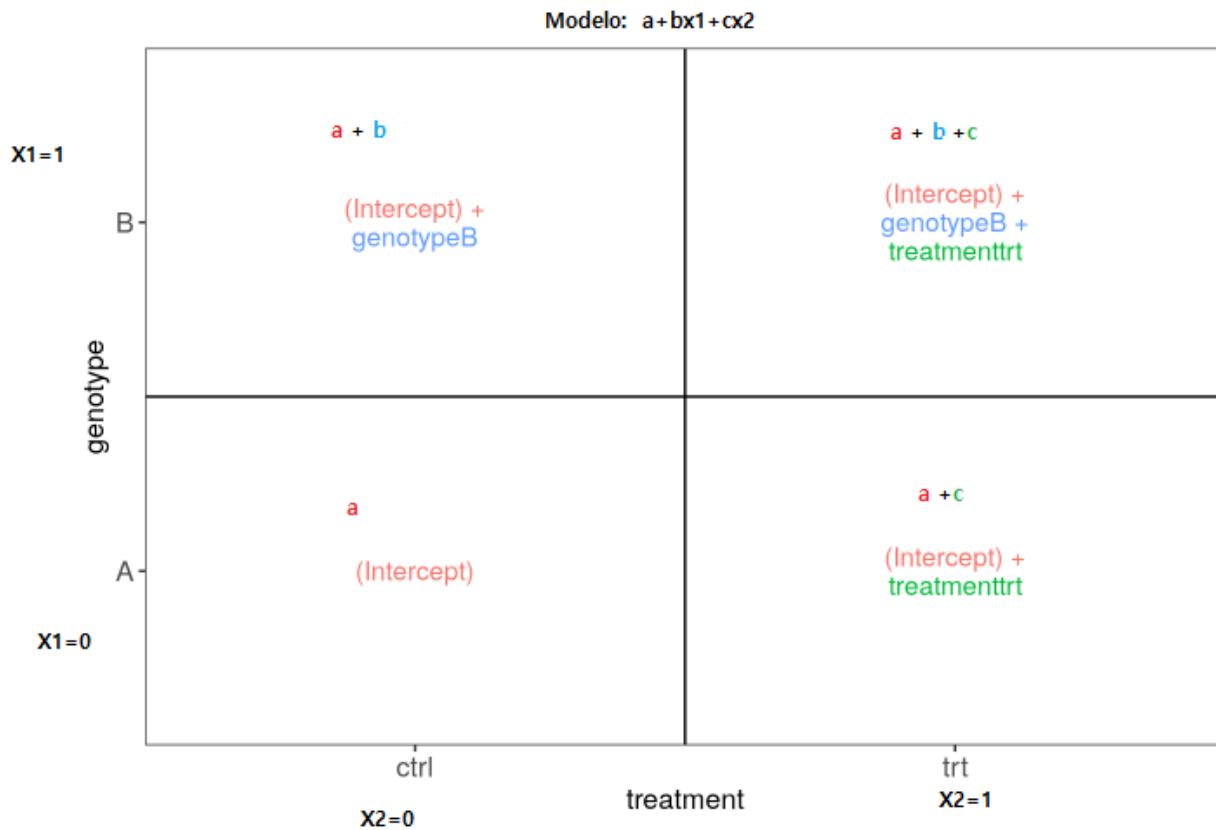
Paquete de *Bioconductor*

```
## Datos de ejemplo
(sampleData <- data.frame(
  genotype = rep(c("A", "B"), each = 4),
  treatment = rep(c("ctrl", "trt"), 4)
))
```

```
##   genotype treatment
## 1       A      ctrl
## 2       A      trt
## 3       A      ctrl
## 4       A      trt
## 5       B      ctrl
## 6       B      trt
## 7       B      ctrl
## 8       B      trt
```

```
## Creamos las imágenes usando ExploreModelMatrix
vd <- ExploreModelMatrix::VisualizeDesign(
  sampleData = sampleData,
  # Misma sintaxis que regresión lineal pero sin Y porque la Y cambia de acuerdo al gen
  designFormula = ~ genotype + treatment,
  textSizeFitted = 4
)

## Veamos las imágenes
cowplot::plot_grid(plotlist = vd$plotlist)
```



Podemos crear una matriz con *dummy variables*, de acuerdo a variables de referencia generadas automáticamente.

```
mod <- model.matrix(~ genotype + treatment, data = sampleData)
mod
```

```
##   genotypeB treatmenttrt
## 1       0       0
## 2       0       1
## 3       0       0
## 4       0       1
## 5       1       0
## 6       1       1
## 7       1       0
## 8       1       1
```

Para cambiar la variable de referencia:

```
## Cambiar la variable de referencia por comandos
sampleData$genotype
## [1] "A" "A" "A" "A" "B" "B" "B" "B"
```

```
factor(sampleData$genotype)
## [1] A A A A B B B B
## Levels: A B
factor(sampleData$genotype, levels = c("B", "A"))
## [1] A A A A B B B B
## Levels: A B
```

En el ambiente interactivo se pude cambiar la variable de referencia:

```
## Usaremos shiny otra ves
app <- ExploreModelMatrix(
  sampleData = sampleData,
  designFormula = ~ genotype + treatment
)
if (interactive()) shiny::runApp(app)
```



Ejercicios más avanzados en el repositorio de GitHub

5.1.1 Ejercicios

- Interpreta `ResponseResistant.Treatmentpre` del [ejercicio 2](#). Puede ser útil tomar un *screenshot* (captura de pantalla) y anotarla con líneas de colores. Si haces eso, puedes incluir la imagen en tus notas.

Es una variable que describe si el estudio de los pacientes fue antes o después del tratamiento con:

`post=0`

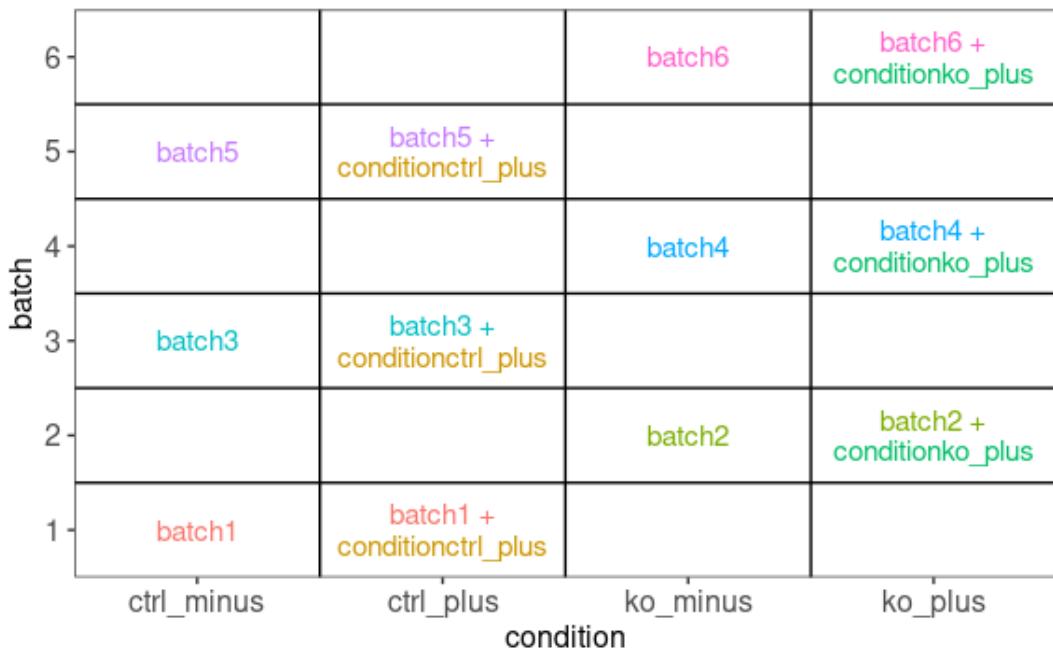
`pre=1`

Es la diferencia entre `col2 -col1`

- ¿Por qué es clave el `0` al inicio de la fórmula en el [ejercicio 3](#)?

Para que el intercept se vaya a 0.

Se debe tomar a los minus como 0 y (no se estiman sus coeficientes) plus como 1



5.2 Datos de SRP045638

```

library("recount3")
## Obtener datos
human_projects <- available_projects()
rse_gene_SRP045638 <- create_rse(
  subset(
    human_projects,
    project == "SRP045638" & project_type == "data_sources"
  )
)
assay(rse_gene_SRP045638, "counts") <- compute_read_counts(rse_gene_SRP045638)

## sra-sample_attributes: una muestra tiene info que no viene en otras entonces se borra con gsub
## b, también se borra su pipe
rse_gene_SRP045638$sra.sample_attributes <- gsub("dev_stage;;Fetal\\|", "", rse_gene_SRP045638$sra.sample_attributes)
rse_gene_SRP045638$sra.sample_attributes[1:3]
## Agregar atributos extras de interés
rse_gene_SRP045638 <- expand_sra_attributes(rse_gene_SRP045638)

colData(rse_gene_SRP045638)[
  ,
  grepl("^sra_attribute", colnames(colData(rse_gene_SRP045638)))
]
## Pasar de character a numeric o factor
rse_gene_SRP045638$sra_attribute.age <- as.numeric(rse_gene_SRP045638$sra_attribute.age)
rse_gene_SRP045638$sra_attribute.disease <- factor(rse_gene_SRP045638$sra_attribute.disease)
## RIN: medida de la calidad del RNA
rse_gene_SRP045638$sra_attribute.RIN <- as.numeric(rse_gene_SRP045638$sra_attribute.RIN)

```

```
rse_gene_SRP045638$sra_attribute.sex <- factor(rse_gene_SRP045638$sra_attribute.sex)

## Resumen de las variables de interés
summary(as.data.frame(colData(rse_gene_SRP045638)[
  ,
  grepl("^sra_attribute.[age|disease|RIN|sex]", colnames(colData(rse_gene_SRP045638)))
]))
```

sra_attribute.age	sra_attribute.disease	sra_attribute.isolate	sra_attribute.RIN	sra_attribute.sex
Min. :-0.4986	control:62	Length:66	Min. :5.30	female:22
1st Qu.: 0.3424	Control: 4	Class :character	1st Qu.:8.00	male :44
Median :14.9000		Mode :character	Median :8.30	
Mean :22.6286			Mean :8.15	
3rd Qu.:41.2900			3rd Qu.:8.70	
Max. :73.9100			Max. :9.60	

Ahora crearemos un par de variables para que las podamos usar en nuestro análisis.

```
## Encontraremos diferencias entre muestra prenatalas vs postnatales
rse_gene_SRP045638$prenatal <- factor(ifelse(rse_gene_SRP045638$sra_attribute.age < 0, "prenatal",
"postnatal"))
table(rse_gene_SRP045638$prenatal)

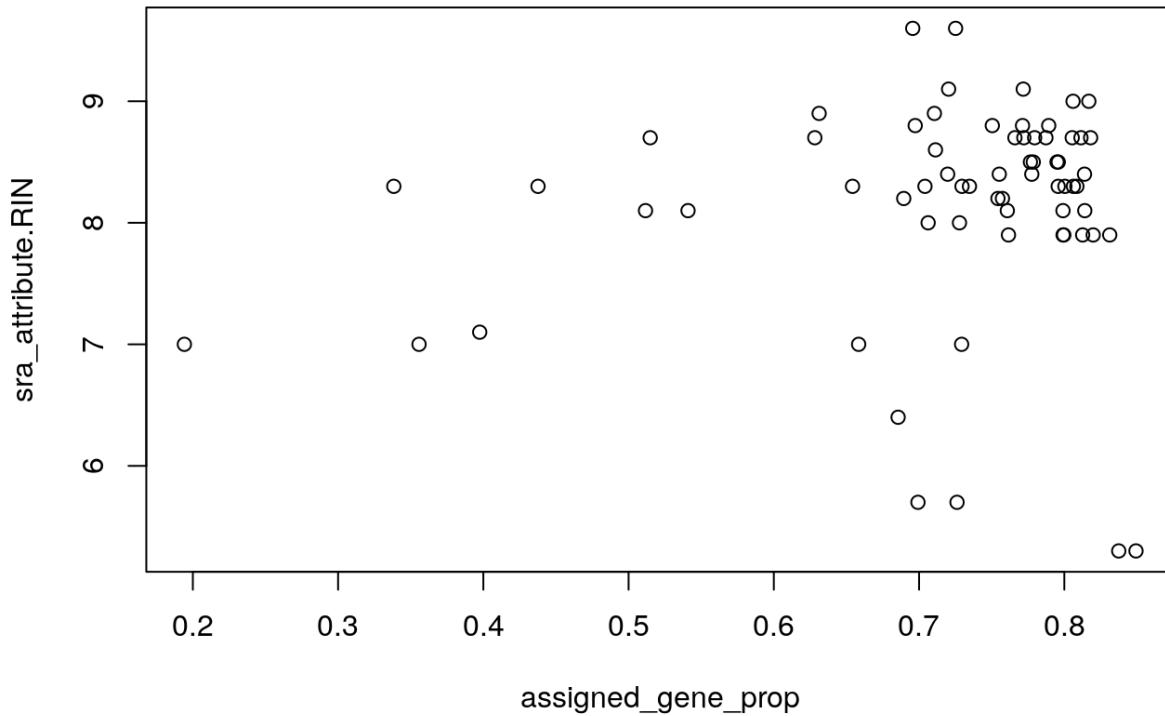
## postnatal prenatal
##      56      10

## Calcular proporción de reads asignadas a genes para análisis de calidad
## count_all.assigned es el numero de lecturas asignadas a genes y total el no. total de reads
rse_gene_SRP045638$assigned_gene_prop <- rse_gene_SRP045638$recount_qc.gene_fc_count_all.assigned / rse_gene_SRP045638$recount_qc.gene_fc_count_all.total
summary(rse_gene_SRP045638$assigned_gene_prop)

##      Min. 1st Qu. Median Mean 3rd Qu. Max.
## 0.1942 0.7004 0.7591 0.7170 0.7991 0.8493

## Ver correlación entre RNA Integrity Number y proporción de reads asignadas
with(colData(rse_gene_SRP045638), plot(assigned_gene_prop, sra_attribute.RIN))

## Hm... veamos si hay una diferencia entre los grupos
with(colData(rse_gene_SRP045638), tapply(assigned_gene_prop, prenatal, summary))
```



```

## Guardemos nuestro objeto entero por si luego cambiamos de opinión
rse_gene_SRP045638_unfiltered <- rse_gene_SRP045638

## Eliminemos a muestras malas
## Valor de corte como 3 desviaciones a la izq de la mediana
hist(rse_gene_SRP045638$assigned_gene_prop)
table(rse_gene_SRP045638$assigned_gene_prop < 0.3)
## FALSE TRUE
## 65 1
rse_gene_SRP045638 <- rse_gene_SRP045638[, rse_gene_SRP045638$assigned_gene_prop > 0.3]

## Eliminar genes con niveles de expresión debajo del primer quartil
## Calculemos los niveles medios de expresión de los genes en nuestras muestras.
## Ojo: en un análisis real probablemente haríamos esto con los RPKMs o CPMs en vez de las cuentas.
gene_means <- rowMeans(assay(rse_gene_SRP045638, "counts"))
summary(gene_means)

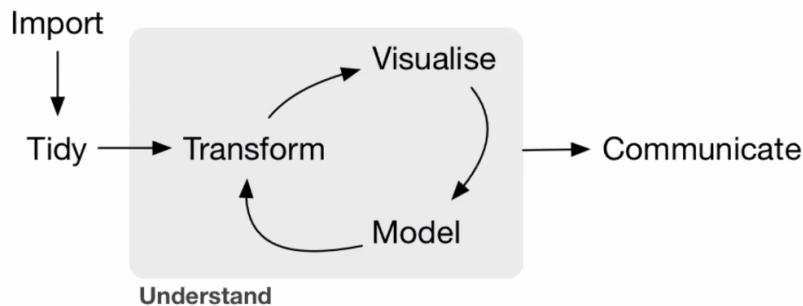
## Eliminamos genes con niveles medios de expresión <=0.1
rse_gene_SRP045638 <- rse_gene_SRP045638[gene_means > 0.1, ]

## Dimensiones finales
dim(rse_gene_SRP045638)
## [1] 46932 65

```

```
## Porcentaje de genes que retuvimos  
round(nrow(rse_gene_SRP045638) / nrow(rse_gene_SRP045638_unfiltered) * 100, 2)  
## [1] 73.5
```

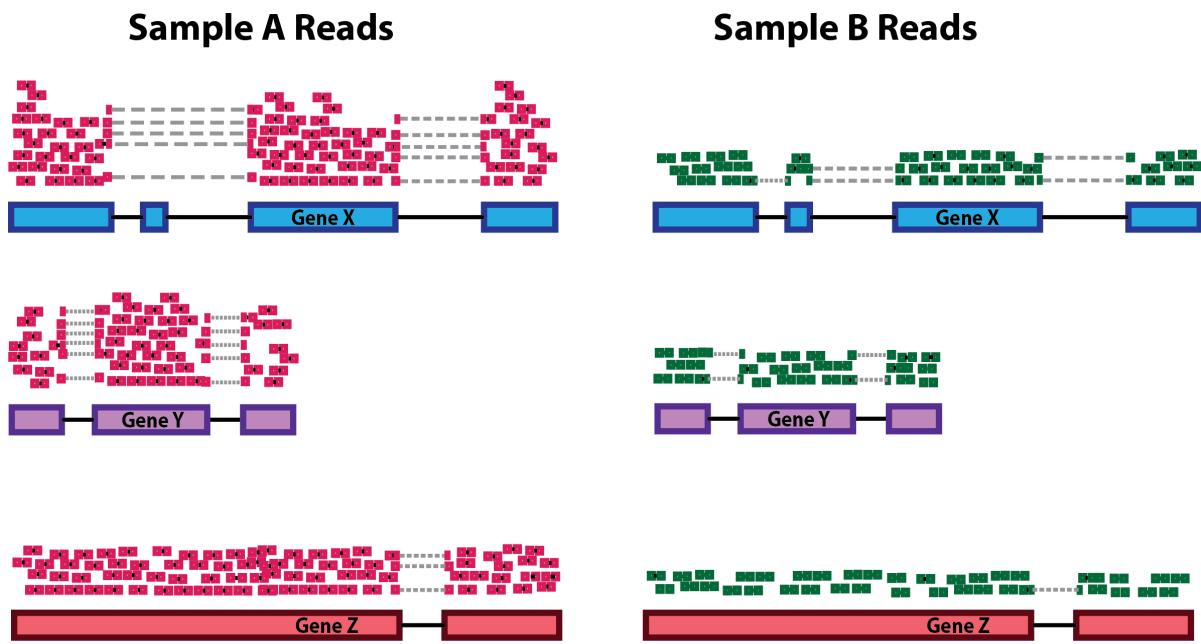
Manejo de datos de RNA-seq



5.2 Normalización

The main factors often considered during normalization are:

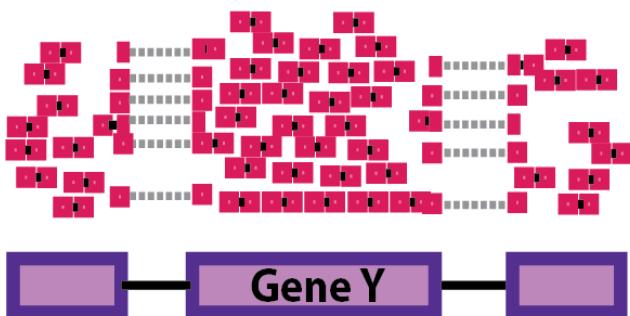
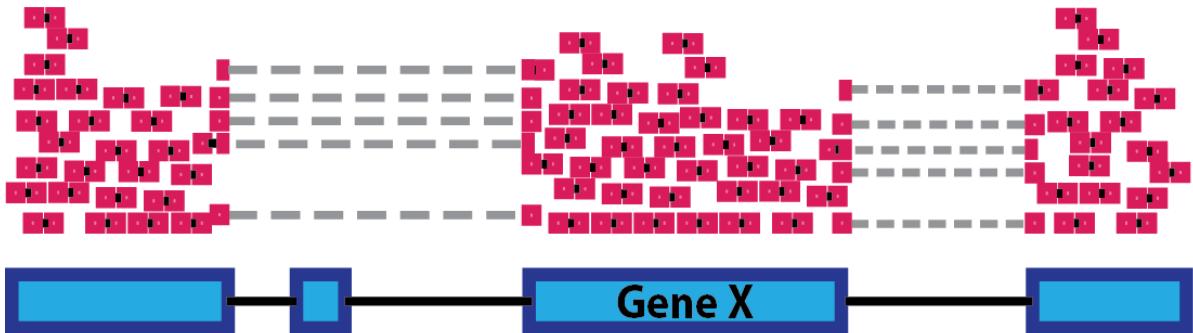
- **Sequencing depth:** Accounting for sequencing depth is necessary for comparison of gene expression between samples. In the example below, each gene appears to have doubled in expression in *Sample A* relative to *Sample B*, however this is a consequence of *Sample A* having double the sequencing depth.



NOTE: In the figure above, each pink and green rectangle represents a read aligned to a gene. Reads connected by dashed lines connect a read spanning an intron.

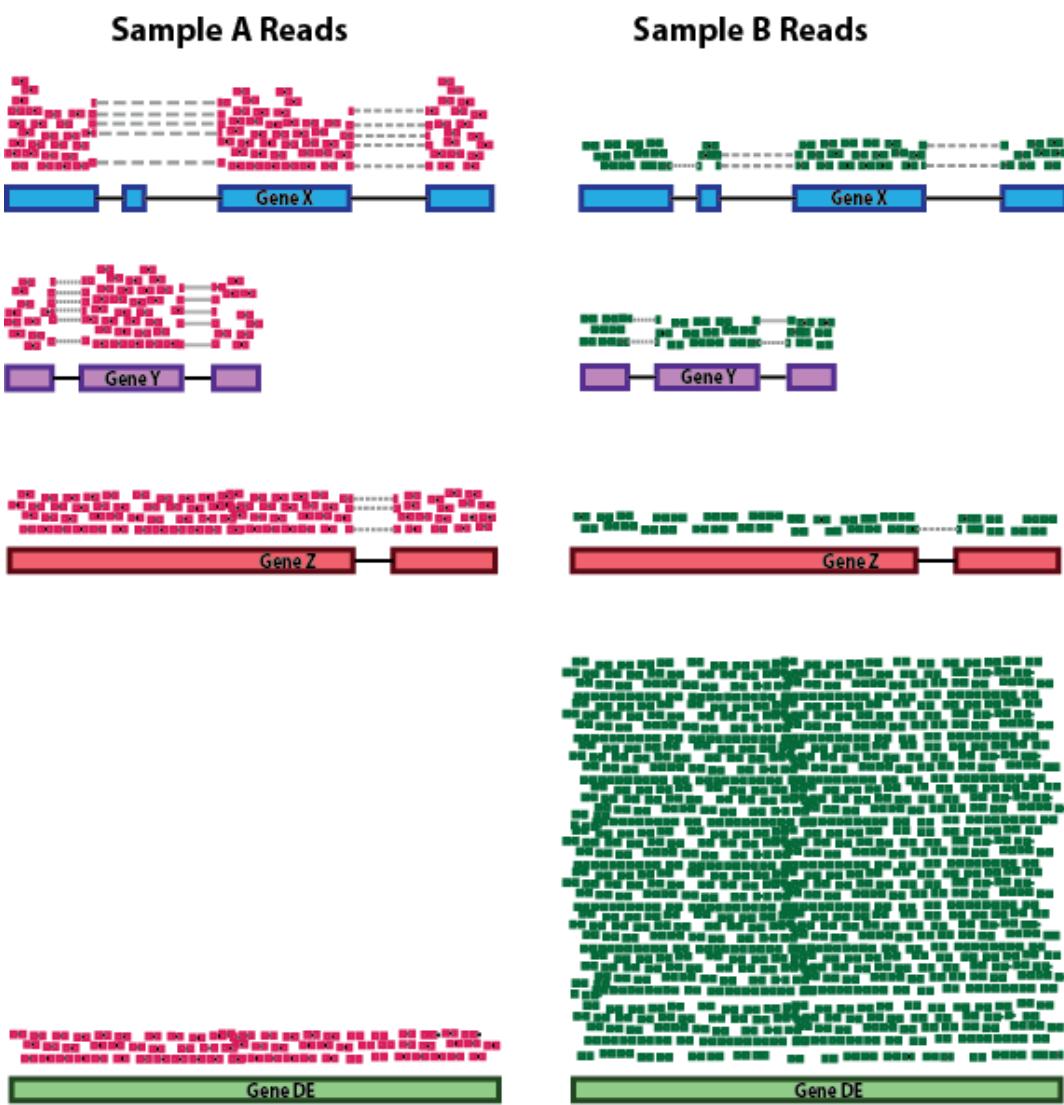
- **Gene length:** Accounting for gene length is necessary for comparing expression between different genes within the same sample. In the example, Gene X and Gene Y have similar levels of expression, but the number of reads mapped to Gene X would be many more than the number mapped to Gene Y because Gene X is longer.

Sample A Reads



- **RNA composition:** A few highly differentially expressed genes between samples, differences in the number of genes expressed between samples, or presence of contamination can skew some types of normalization methods. Accounting for RNA composition is recommended for accurate comparison of expression between samples, and is particularly important when performing differential expression analyses [1].

In the example, if we were to divide each sample by the total number of counts to normalize, the counts would be greatly skewed by the DE gene, which takes up most of the counts for *Sample A*, but not *Sample B*. Most other genes for *Sample A* would be divided by the larger number of total counts and appear to be less expressed than those same genes in *Sample B*.



Con base en los niveles de RNA de cada muestra y el número total de lecturas, se deben considerar los genes abarcados, asumiendo que la mayoría de los genes no están DE

Library size normalization

Toma en cuenta la suma de los niveles de expresión de todos los genes de todas las muestras y las compara. Normalizando de acuerdo a dicha comparación.

Librería `edgeR` :

Tiene su propia clase de objetos, pero es fácil transformar un objeto `SummarizedExperiment` al objeto utilizado en `edgeR`

```
library("edgeR")# BiocManager::install("edgeR", update = FALSE)
dge <- DGEList(
```

```

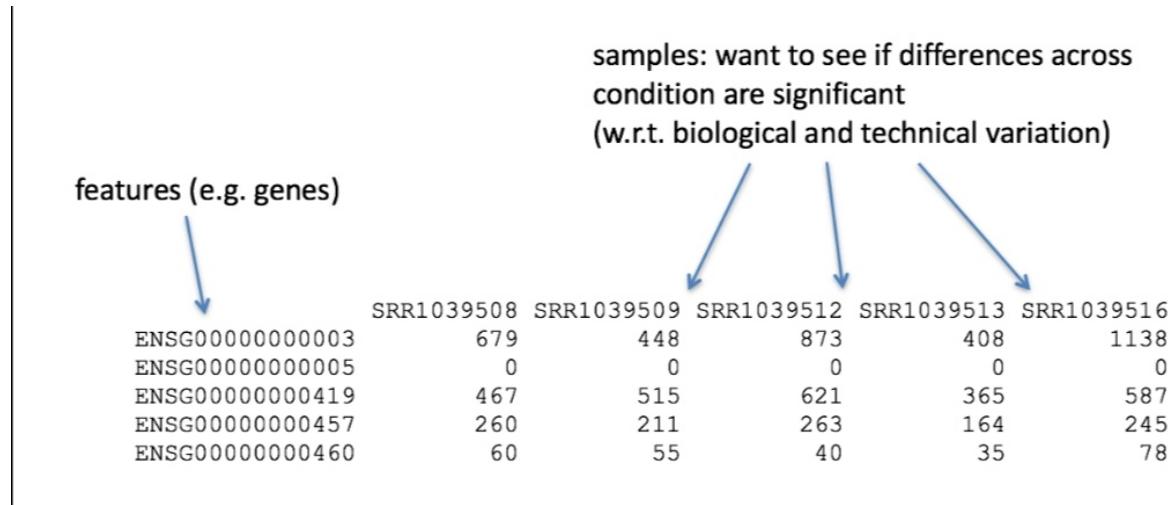
counts = assay(rse_gene_SRP045638, "counts"),
genes = rowData(rse_gene_SRP045638)
)
dge <- calcNormFactors(dge)

```

	Condition A	Condition B	A norm lib size	B norm lib size	A / B
Gene1	8	8	0.075471698	0.150943	0.5
Gene2	33	33	0.311320755	0.622642	0.5
Gene3	12	12	0.113207547	0.226415	0.5
Gene4	13	0	0.122641509	0	#DIV/0!
Gene5	6	0	0.056603774	0	#DIV/0!
Gene6	34	0	0.320754717	0	#DIV/0!
Total	106	53			

5.3 Expresión diferencial

The goal of differential expression testing is to determine which genes are expressed at different levels between conditions. These genes can offer biological insight into the processes affected by the condition(s) (of the sample) of interest.



Se hace un modelo de regresión para cada gen considerando las variables de interés de las muestras y con él se calcula el valor de expresión de cada gen en cada muestra.

Generalized linear models

- Model the expression of each gene as a linear combination of explanatory factors (eg. group, time, patient)

- $y = a + (b \cdot \text{group}) + (c \cdot \text{time}) + (d \cdot \text{patient}) + e$
 y = gene's expression
 a, b, c and d = parameters estimated from the data
 a = intercept (expression when factors are at reference level)
 e = error term

- Generalized linear model (GLM) allows the expression value distribution to be different from normal distribution

- Negative binomial distribution used for count data



```
library("ggplot2")
## Datos se convierten a df (input de ggplot) con aes se definen variables, como se quiere graficar, se cambia tema y tamaño de puntos
ggplot(as.data.frame(colData(rse_gene_SRP045638)), aes(y = assigned_gene_prop, x = prenatal)) +
  geom_boxplot() + theme_bw(base_size = 20) + ylab("Assigned Gene Prop") + xlab("Age Group")

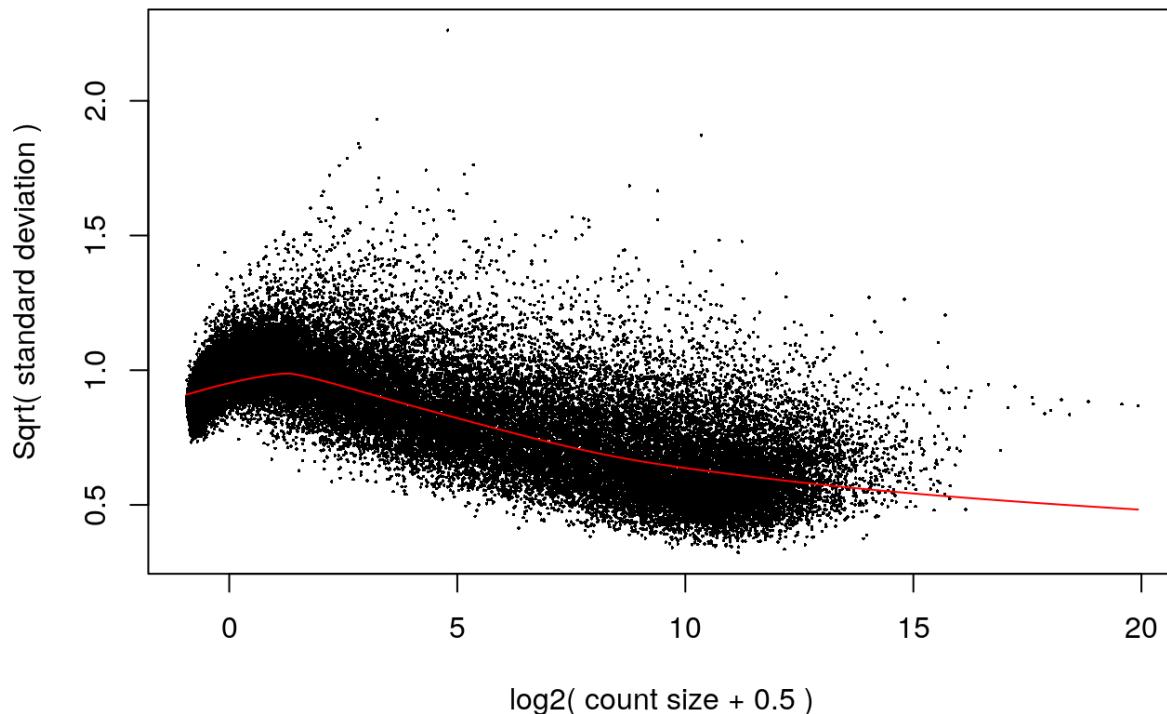
## Se genera el modelo ajustando por prenatal, RIN, sex y assigned gene prop
mod <- model.matrix(~ prenatal + sra_attribute.RIN + sra_attribute.sex + assigned_gene_prop,
  data = colData(rse_gene_SRP045638)
)
colnames(mod)

## Columna prenatal porque post al ser el primer en orden alfabetico es el de referencia
## y se muestra variable+contraste (prenatal y prenatal)
## [1] "(Intercept)"           "prenatalprenatal"       "sra_attribute.RIN"      "sra_attribute.sex"
## [5] "male"                  "assigned_gene_prop"
```

Ya teniendo el modelo estadístico, podemos usar `limma` para realizar el análisis de expresión diferencial como tal. Usa una ~binomial negativa usando valores enteros y no negativos de expresión (no. reads que sobrelapan). Hace iteraciones hasta converger a un valor.

```
##
library("limma")
## voom hace gráfica con eje x como el valor de expresión (no. cuentas+0.5 (porque log2 0 no funciona)) y en el y
## Se ve la relación entre el promedio de expresión (x) y la varianza (y), cada punto es un solo gen
vGene <- voom(dge, mod, plot = TRUE)
```

voom: Mean-variance trend



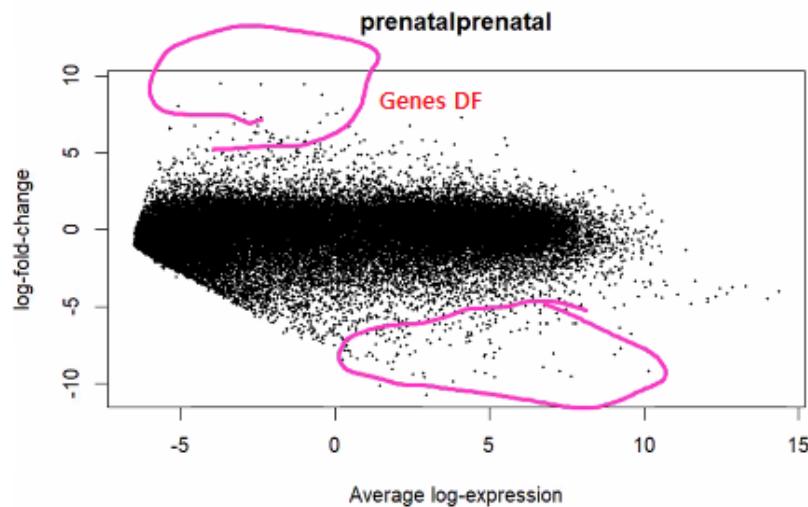
```
## Calcular y extraer valores t y p
eb_results <- eBayes(lmFit(vGene))
##
de_results <- topTable( ## por defecto regresa los 10 genes con DF ordenados por valor t y p, ca
mbiar en number y sort
  eb_results,
  coef = 2, ## Numero o nombre que es el no. de columna del modelo o la variable para la cual
queremos calcular valores t y p, el de interes
  number = nrow(rse_gene_SRP045638),
  sort.by = "none") ## no ordenar para conservar posiciones de datos
dim(de_results)
head(de_results)
## limma de el logFC (fold change) si es positivo esta up regulated
## valor t negativo es que tiene mayor exp en post que pre
## p value >0.05 es DE significativo
## Genes diferencialmente expresados entre pre y post natal con FDR < 5%
table(de_results$adj.P.Val < 0.05)
## FALSE TRUE
## 12898 34034

## Visualicemos los resultados estadisticos: relacion entre logFC y promedio de expresion de los
```

```

genes
plotMA(eb_results, coef = 2)
## Relacion logFC y p value (-10log= > los p valores más cercanos a cero están mas arriba)
volcanoplot(eb_results, coef = 2, highlight = 3, names = de_results$gene_name)
## ver resultados de genes específicos
de_results[de_results$gene_name %in% c("ZSCAN2", "VASH2", "KIAA0922"), ]

```



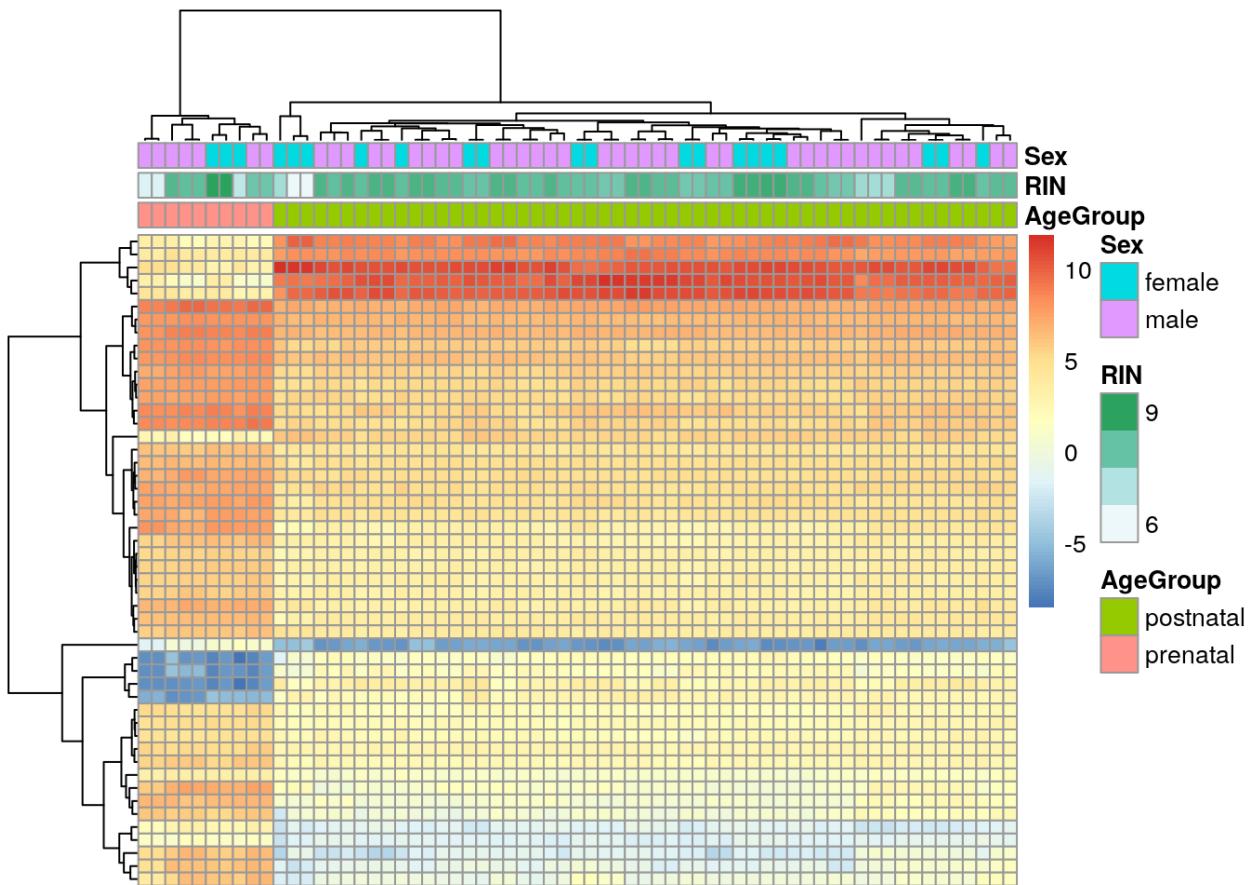
5.4 Visualizando genes DE

De `vGene$E` podemos extraer los datos normalizados por `limma-voom`. Revisemos los top 50 genes diferencialmente expresados.

```

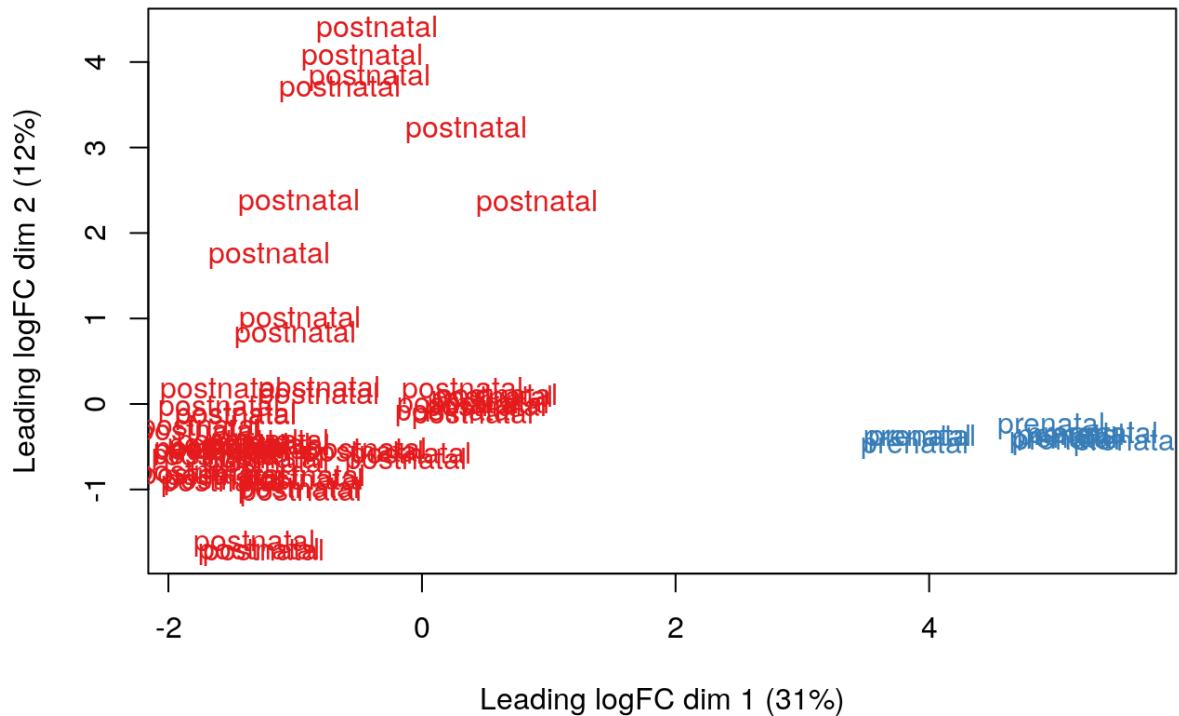
## Extraer valores de los genes de interés con mayor señal de DE
exprs_heatmap <- vGene$E[rank(de_results$adj.P.Val) <= 50, ]
## Creamos una tabla con información de las muestras y con nombres de columnas más amigables
df <- as.data.frame(colData(rse_gene_SRP045638)[, c("prenatal", "sra_attribute.RIN", "sra_attribute.sex")])
colnames(df) <- c("AgeGroup", "RIN", "Sex")
## Hagamos un heatmaplibrary("pheatmap")
pheatmap(
  # Valores de expresión normalizados
  exprs_heatmap,
  cluster_rows = TRUE,
  cluster_cols = TRUE,
  show_rownames = FALSE,
  show_colnames = FALSE,
  annotation_col = df
)

```



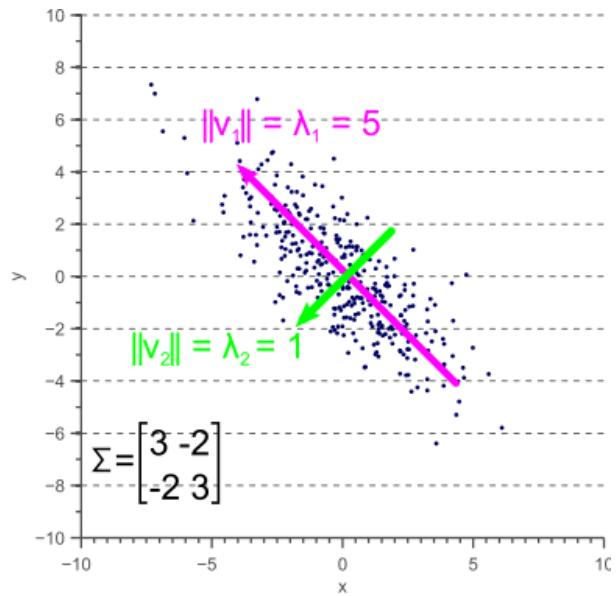
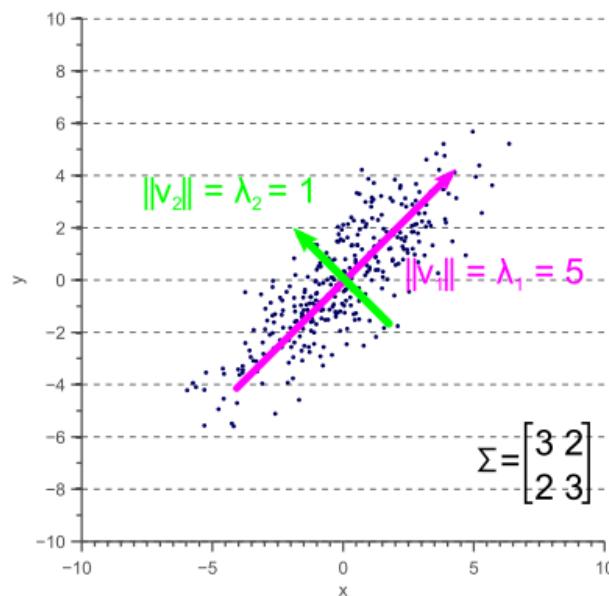
```
## Para colores
library("RColorBrewer")
## Convirtiendo los grupos de edad a colores
col.group <- df$AgeGroup
levels(col.group) <- brewer.pal(nlevels(col.group), "Set1")
```

```
col.group <- as.character(col.group)
## MDS por grupos de edad
plotMDS(vGene$E, labels = df$AgeGroup, col = col.group)
```



Para entender análisis de PCA → [Intuitive understanding of Eigenvectors: Key to PCA | by Najeeb Qazi | Towards Data Science](#)

[PCA: Eigenvectors and Eigenvalues | by Valentina Alto | Towards Data Science](#)



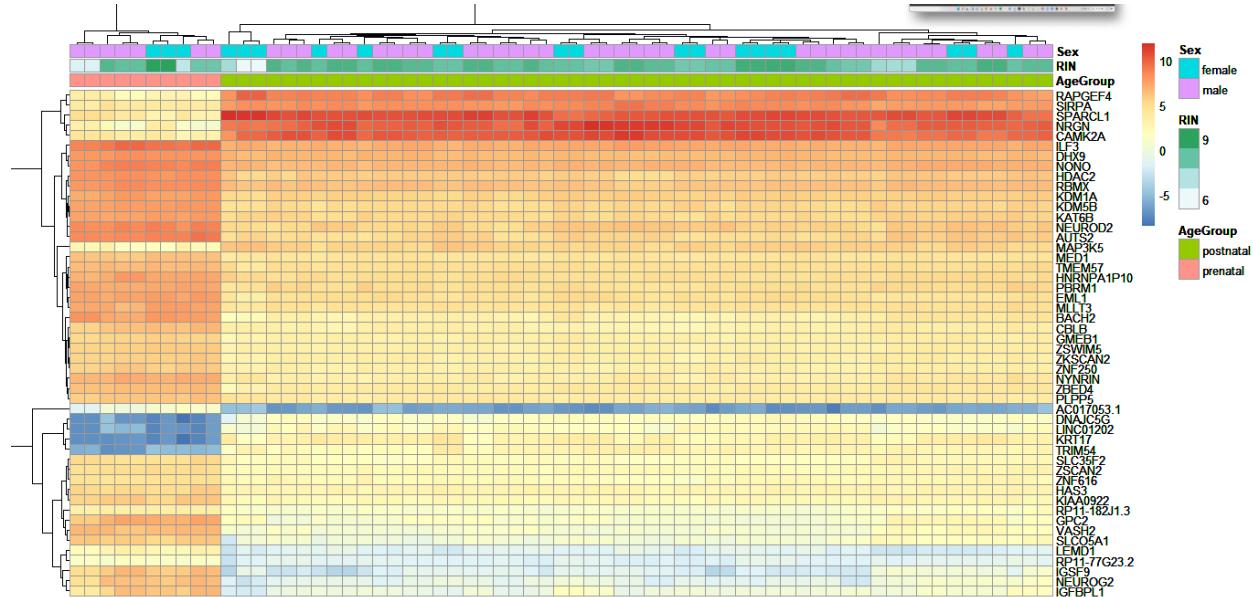
```
## Convirtiendo los valores de Sex a colores
col.sex <- as.character(col.sex)
levels(col.sex) <- brewer.pal(nlevels(col.sex), "Dark2")
```

```
col.sex <- as.character(col.sex)
## MDS por sexoplotMDS(vGene$E, labels = df$Sex, col = col.sex)
```

Ejercicio:

Agreguen los nombres de los genes a nuestro `pheatmap`.

```
rownames(exprs_heatmap) <- de_results$gene_name[match(rownames(exprs_heatmap), de_results$gene_id)]
pheatmap(
    exprs_heatmap,
    cluster_rows = TRUE,
    cluster_cols = TRUE,
    show_rownames = TRUE,
    show_colnames = FALSE,
    annotation_col = df
)
```



6. Ejercicio

- ¿Debemos explorar las relaciones entre nuestras variables con información de nuestras muestras previo a hacer un análisis de expresión diferencial?

Sí para generar el modelo estadístico

- ¿Por qué usamos el paquete `edgeR` ?

Para normalizar los datos eliminando composition bias

- ¿Por qué es importante el argumento `sort.by` en `topTable()` ?

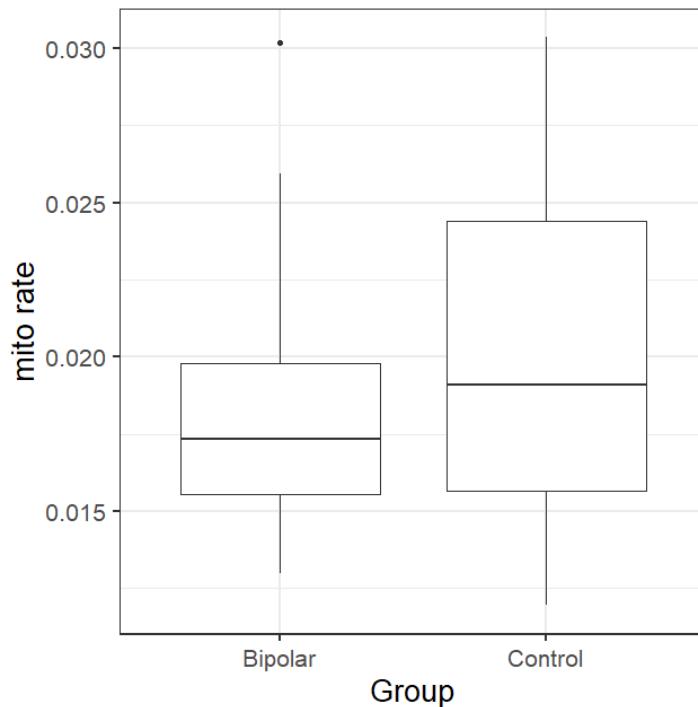
Porque se encarga de ordenar los genes con mayores diferencias en su expresión y usualmente queremos conservar las posiciones originales al ligarlo con los datos RSE para calcular los niveles de expresión de los genes

- ¿Por qué es importante el argumento `coef` en `topTable()` ?

Para escoger la variable para la cual queremos calcular valores t y p, el de interés, para hacer el DE con respecto a esta variable

- ¿Hay diferencias en `totalAssignedGene` o `mitoRate` entre los grupos de diagnosis (`PrimaryDx`)?

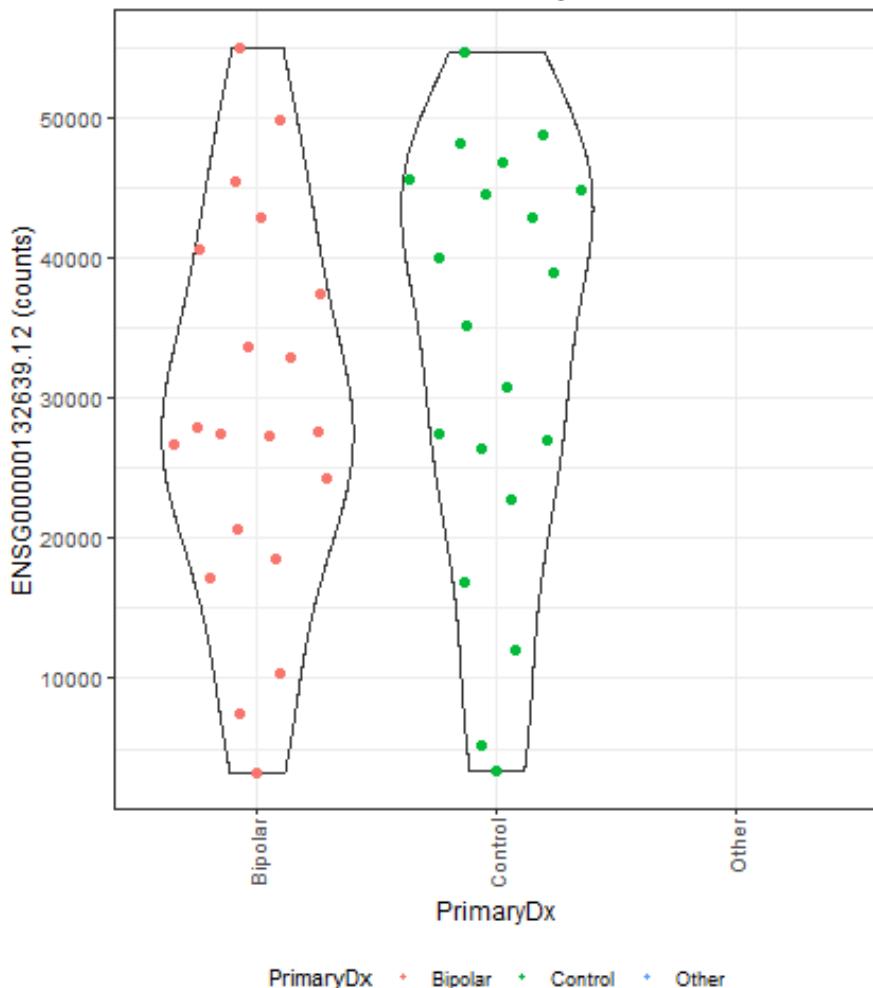
```
tapply(rse_gene$mitoRate, rse_gene$PrimaryDx, summary)
library("ggplot2")
ggplot(as.data.frame(colData(rse_gene)), aes(y = mitoRate, x = PrimaryDx)) +
  geom_boxplot() +
  theme_bw(base_size = 20) +
  ylab("mito rate") +
  xlab("Group")
```



- Grafica la expresión de *SNAP25* para cada grupo de diagnosis.

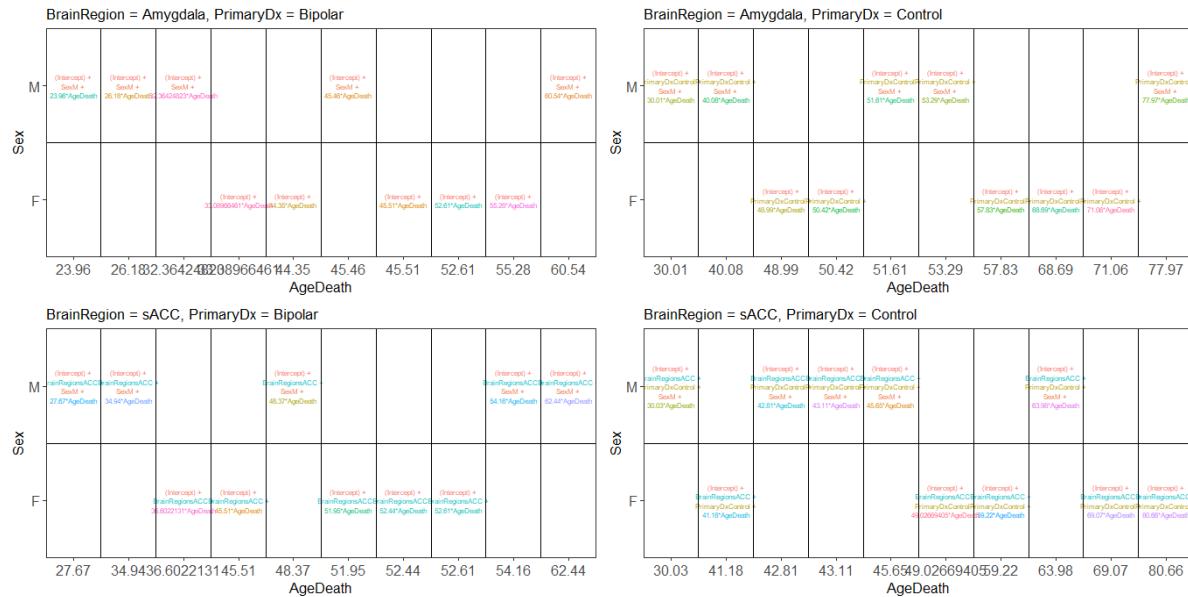
```
library("iSEE")
iSEE::iSEE(rse_gene)
```

ENSG00000132639.12 vs PrimaryDx



- Sugiere un modelo estadístico que podríamos usar en una análisis de expresión diferencial. Verifica que si sea un modelo *full rank*. ¿Cuál sería el o los coeficientes de interés?

```
mod <- model.matrix(~ BrainRegion + PrimaryDx + Sex + AgeDeath,
data = colData(rse_gene))
colnames(mod)
```



Extras

R graphics

R Graphics Cookbook, 2nd edition

This cookbook contains more than 150 recipes to help scientists, engineers, programmers, and data analysts generate high-quality graphs quickly-without having to comb through all the details of R's graphing

<https://r-graphics.org/>



R Graphics



Jeff Leek How to be a modern scientist

Jeff Leek

<http://jtleek.com/>

R Themes

```
remotes::install_github(c(
  "gadenbuie/rsthemes"
))
remotes::install_cran("suncalc")
rsthemes::install_rsthemes(include_base16 = TRUE)
```

Punto de corte

Ejemplo

⌚ https://github.com/LieberInstitute/brainseq_phase2/blob/master/expr_cutoff/pdf/suggested_expr_cutoffs_gene.pdf

Paquetes

Librería *purr*

Programación funcional

⌚ <https://github.com/ComunidadBioInfo/cdsb2019/blob/master/05-fp.pdf>

Genefilter

genefilter

DOI: 10.18129/B9.bioc.genefilter Bioconductor version: Release (3.12)
Some basic functions for filtering genes. Author: R. Gentleman, V. Carey,
W. Huber, F. Hahne Maintainer: Bioconductor Package Maintainer Citation

⌚ <https://www.bioconductor.org/packages/release/bioc/html/genefilter.html>



Differential expression analysis of dermal endothelial cells of healthy and type 2 diabetic patients

Negative Binomial

