Machine Learning y Big Data para la Bioinformática.

Módulo 2 - Análisis bioinformático sobre un problema en Ómicas

Breves instrucciones

Introducción a NoteBook

Dentro de este cuaderno (*NoteBook*), se le guiará paso a paso desde la carga de un conjunto de datos hasta el análisis descriptivo de su contenido.

El cuaderno de *Jupyter* (Python y R) es un enfoque que combina bloques de texto (como éste) junto con bloques o celdas de código. La gran ventaja de este tipo de celdas, es su interactividad, ya que pueden ser ejecutadas para comprobar los resultados directamente sobre las mismas. *Muy importante*: el orden de las instrucciones es fundamental, por lo que cada celda de este cuaderno debe ser ejecutada secuencialmente. En caso de omitir alguna, puede que el programa responda con un error, así que una comprobación básica para comprobar el error es comenzar a ejecutar el cuaderno desde el principio, asegurándonos de no dejar ninguna celda sin ejecutar.

Antes de nada:

Es muy muy importante que al comienzo se seleccione "*Abrir en modo de ensayo*" (draft mode), arriba a la izquierda. En caso contrario, no permitirá ejecutar ningún bloque de código, por cuestiones de seguridad. Cuando se ejecute el primero de los bloques, aparecerá el siguiente mensaje: "*Advertencia: Este cuaderno no lo ha creado Google.*". No se preocupe, deberá confiar en el contenido del cuaderno (*NoteBook*) y pulsar en "Ejecutar de todos modos".

¡Ánimo!

Haga clic en el botón "play" en la parte izquierda de cada celda de código. Las líneas que comienzan con un hashtag (#) son comentarios y no afectan a la ejecución del programa.

También puede pinchar sobre cada celda y hacer "*ctrl+enter*" (*cmd+enter* en Mac).

Cada vez que ejecute un bloque, verá la salida justo debajo del mismo. La información suele ser siempre la relativa a la última instrucción, junto con todos los `print()` (orden para imprimir) que haya en el código.

***Nota importante*: este Notebook contiene algunas celdas de código que instalan bibliotecas y funciones necesarias para ejecutar los códigos de las Cápsulas 1, 2, y 3. Estas celdas de instalación tomarán varios minutos para completar su ejecución (se indica las que requieren más tiempo).**

En cualquier caso, este cuaderno ya ha sido ejecutado previamente y se muestran los resultados de todas las celdas de código, así que os recomendamos que avancéis en la lectura del cuaderno mientras vais ejecutando las celdas. En los materiales adjuntos al curso en abierta.ugr.es también disponéis de un PDF con todas las celdas de código y los resultados obtenidos.

Si vuestra sesión expira, lo que ocurrirá cuando cerréis el navegador o tras cierto tiempo de inactividad, tendréis que volver a ejecutar las celdas de instalación de paquetes en R y Bioconductor, lo que volverá a requerir cierto tiempo. Por ello, os recomendamos que ejecutéis en la misma sesión los códigos de las tres cápsulas y que utilicéis los resultados precalculados que se muestran si queréis avanzar más rápido.

Cápsula 1 - El problema - ¿Cómo obtener y preparar los datos?

Autores:

Por **Carlos Cano Gutiérrez**

Profesor Titular de la Universidad de Granada.
Departamento de Ciencias de Computación e Inteligencia Artificial.

Por **Pedro Carmona Sáez**

Profesor Titular de la Universidad de Granada.
Departamento de Estadística e Investigación Operativa.

ÍNDICE

En este *notebook*:

2. Se describe un problema con datos -ómicos que se empleará para ilustrar distintos análisis durante el resto del curso.
3. Aprendemos a descargar de forma automática datos del proyecto TCGA (The Cancer Genome Atlas).
4. Aprendemos a reconocer cómo están organizados los datos de TCGA.
5. Aprendemos a utilizar algunas funciones útiles del lenguaje de programación R.

Contenidos:

1. NOMENCLATURA DE INTERÉS PARA DESCRIBIR UN PROBLEMA

Los protagonistas de este curso son los datos y las técnicas de análisis de datos. Es por esto que antes de empezar a manipular datos, necesitamos acordar la nomenclatura con la que nos referiremos a ciertos términos y conceptos frecuentes en el ámbito.

Cuando pensamos en grandes volúmenes de datos, típicamente se nos viene a la mente una representación de datos en forma de tabla, con un gran número de filas y columnas. Sin embargo, rara vez los datos son producidos directamente en una única tabla, ya *limpia* y preparada para su análisis. Por el contrario, típicamente los datos son producidos en distintos formatos (texto, imágenes, audio, video, varias tablas con información complementaria entre sí, etc.), son heterogéneos, están incompletos y tienen ruido.

Las técnicas de *Machine Learning* que os vamos a enseñar en el curso no aprenden directamente a partir de los datos *crudos* (denominamos un dato *crudo* como un dato que no ha sido manipulado o procesado, sino que está en su formato y escala original). Por ejemplo, los datos *crudos* son las imágenes capturadas por un escáner de tomografía computerizada (TAC) o los textos de la historia clínica de un paciente. Para analizar este tipo de datos es necesario primero convertirlos a un formato apropiado para manipularlos. Como se indicará en la siguiente cápsula, el proceso de conversión, preparación o manipulación de datos que permite su análisis posterior por técnicas computacionales se denomina *preprocesamiento*.

Definimos un **conjunto de datos** como una colección de objetos, puntos, registros, patrones, eventos, casos, muestras, observaciones, o instancias. Para unificar la nomenclatura a lo largo del curso, utilizaremos este último término, **instancia** para referirnos a cada uno de estos objetos. Por ejemplo, una instancia podría ser un paciente en un estudio clínico.

Las instancias pueden representarse como un conjunto de características, propiedades o variables que las describen. Utilizaremos este último término, **variable**, para unificar la nomenclatura a lo largo del curso. Por tanto, definimos una variable como una medida individual que caracteriza una propiedad de una instancia. Ejemplos de variables son la edad, el sexo o la presión arterial de una persona en el momento de la toma de muestras. Algunos sinónimos para variable son: característica, campo, atributo, etc.

Las variables pueden ser de distinta naturaleza:

- Cualitativa
  - Dicotómicas, tiene dos posibles valores, por ejemplo: Sexo (M/V), Fumador (S/N).
  - Nominales: pueden tener varias categorias sin orden. Por ejemplo diferentes tipos de tratamiento
  - Ordinales: varias categorias entre las que existe un orden, por ejemplo estadío tumoral o grado (T0, TI, TII, TIII, TIV), el estadío tumoral TI es el estadío inicial, y es *anterior* a TII, que a su vez *precede* a TIII, y así sucesivamente.
- Cuantitativas: valor numérico, pueden ser discretas o contínuas.

Utilizaremos la nomenclatura definida en esta sección en el resto del curso. Si la temática concreta de algún módulo invita a utilizar otra denominación para el conjunto de datos (instancias o variables) se especificá claramente al inicio del módulo que corresponda.

2. DESCRIPCIÓN DE UN PROBLEMA EN ÓMICAS

En esta sección se describe uno de los problemas modelo que se van a emplear durante el curso. Se trata del proyecto TCGA-SKCM (*TCGA-Skin Cell Melanoma*), un esfuerzo de la iniciativa Cancer Genome Atlas (TCGA) para el análisis multifactorial de cientos de muestras de melanoma de piel. Este análisis se denomina multifactorial porque incluye distintos tipos de datos -ómicos introducidos en el Módulo 1, en este caso, información de los tumores a nivel de DNA, RNA y proteína. El objetivo es crear un catálogo de mutaciones asociadas a este tumor e identificar patrones con impacto clínico para el pronóstico de la enfermedad.

Este modelo de problema es similar al de otros cientos de problemas que utilizan las Ciencias -Ómicas hoy día: a partir de una cierta hipótesis científica, los investigadores coleccionan cientos de muestras de la misma condición y utilizan sofisticadas técnicas experimentales para caracterizar en detalle estas muestras con el objetivo de descubrir patrones en los datos con potencial relevancia clínica.

En particular, el proyecto TCGA-SKCM realiza una caracterización de las muestras que incluye información genómica, transcriptómica, epigenética y características clínico-patológicas, por ejemplo, estadío tumoral, metástasis, tratamiento, tiempo hasta remisión/muerte/recidiva, etc.

Según el tipo de análisis que se realice sobre la información obtenida de las muestras, el patrón que se identifique puede utilizarse para la predicción de un pronóstico de forma más certera o precoz, la clasificación automática de la tipología del tumor o la identificación de distintos subtipos del tumor para elaborar tratamientos más específicos, entre otras aplicaciones.

En este *notebook* vamos a ilustrar el proceso de descarga y preparación de datos de la plataforma TCGA, utilizando como ejemplo el proyecto TCGA-SKCM.

2.1. Datos de expresión genética

Los denominados datos de expresión genética son muy populares en los análisis de transcriptómica. Hoy día, este tipo de datos se obtienen con las denominadas tecnologías de secuenciación de ARN (RNA-Sequencing o RNA-Seq). Estas tecnologías permiten identificar secuencias de ARN en una muestra celular y cuantificar su abundancia. Es decir, identificar qué

genes se expresan en la muestra en ese instante y cuál es su grado de expresión. Además de cuantificar la expresión de genes, el análisis de estos datos permite identificar nuevas secuencias transcritas a partir de ADN, identificar mecanismos de *splicing* alternativo o detectar expresión específica de alelo, entre otros. Además, estas tecnologías permiten caracterizar no sólo RNA mensajero (mRNA), sino también otros tipos de RNAs como los RNAs que no codifican proteínas (los llamados RNAs no codificantes o *non-coding RNAs*, ncRNAs) que incluyen los lncRNAs y los miRNAs, entre otros.

El resultado de estos experimentos suele ser una **matriz de expresión** X de tamaño $N \times M$, siendo N el número de genes y M el número de muestras, y cada valor de la matriz $x_{i,j}$ representa el valor de expresión del gene i en la muestra j.

Una característica de estos datos es que $N >> M$ ya que el número de genes suele ser del orden de decenas de miles y los casos normalmente de decenas.

Puedes consultar más detalles sobre la secuenciación de ARN en los siguientes enlaces:

- http://cshprotocols.cshlp.org/content/early/2015/04/11/pdb.top084970.abstract
- https://www.nature.com/articles/nrg2484
- https://training.galaxyproject.org/training-material/topics/transcriptomics/

Hay muchos pasos involucrados en el análisis de datos de RNA-Seq antes de obtener la matriz de expresión. Típicamente, este proceso comienza con el procesamiento de lecturas (*reads*), que se alinean contra un genoma de referencia para cuantificar el número de secuencias de ARN asociadas a cada posición del genoma. Como se conoce la posición en el genoma de un amplio repertorio de genes, se suele indicar que estas técnicas permiten cuantificar el grado de expresión de cada gen en una muestra. Esta información se dispone en una matriz numérica sobre la que podemos realizar análisis estadísticos y computacionales. En nuestro caso, y como aproximación inicial al problema, partiremos directamente de las matrices que cuantifican el número de lecturas asociadas a cada gen y nos centraremos en el análisis de estas matrices.

3. LENGUAJES DE PROGRAMACIÓN ELEGIDOS: R Y PYTHON

En esta sección justificamos la elección de los dos lenguajes de programación empleados en este curso: R y Python. Mientras que Python será el lenguaje vehicular del curso por su facilidad de uso y la disponibilidad de numerosos métodos y recursos ya programados para *Machine Learning*, emplearemos R para realizar algunas tareas relacionadas con el tratamiento de datos específicos de Bioinformática.

Tanto Python como R son software libre, por lo que cualquier usuario puede acceder a su código y hacer contribuciones al mismo en forma de *bibliotecas*. Una *biblioteca* es un conjunto de funciones o programas que permiten realizar una tarea o resolver un problema. De este modo, emplearemos un lenguaje u otro en función de la tarea que se vaya a acometer y de los recursos que uno u otro brinden para realizarla.

Python es el lenguaje de programación más popular para *Machine Learning* porque proporciona a los científicos computacionales numerosas bibliotecas ya programadas para preprocesar datos, realizar análisis exploratorios y visualizaciones e inferir y validar modelos. Algunas de

estas bibliotecas, que se emplearán en distintos módulos de este curso, son Pandas, Numpy, Matplotlib, SciPy, scikit-learn, etc.

R es el lenguaje de programación más popular en Bioinformática. Su popularidad en esta disciplina se debe a la disponibilidad de multitud de bibliotecas muy útiles para realizar análisis computacionales y estadísticos específicos sobre ciertos tipos de datos propios de la Bioinformática. Por ejemplo, en este módulo utilizaremos funciones de R ya disponibles para descargar, visualizar, procesar y normalizar datos descargados del proyecto TCGA.

Google Colab proporciona un entorno de programación listo para utilizar con Python y R. Sin embargo, tendremos que instalar algunas bibliotecas con funciones que nos serán útiles para realizar distintas tareas que acometeremos en las siguientes secciones.

3.1. Instalación de R y paquetes

En primer lugar, instalaremos R en nuestro entorno de *Google Colab*. Ten en cuenta que *Google Colab* siempre guardará tus *notebooks* y la información que desde los *notebooks* se almacene en ficheros en tu *Google Drive*. Sin embargo, todas las instalaciones de bibliotecas que realicemos en el entorno de *Google Colab* solo permanecerán activas unas pocas horas, después de las cuales las bibliotecas instaladas se eliminan. Por tanto, será necesario que vuelvas a ejecutar los códigos de instalación de bibliotecas de esta sección cada vez que tu sesión expire.

En vuestro ordenador personal con Sistema Operativo Linux, la forma habitual de realizar la descarga e instalación de las bibliotecas de R necesarias sería por medio de los siguientes comandos en una terminal de Linux (*no es necesario que ejecutéis esto en Google Colab*):

```
# 1 - Instalamos la última versión de R

!apt-get update
!apt-get install r-base

# 2 - Abrimos una terminal de R tecleando `R` (Intro) e instalamos las bibliotecas de R necesarias para que se ejecuten los análisis bioinformáticos de este módulo.

install.packages("BiocManager")
install.packages(c("scales", "pheatmap", "DT", "factoextra", "BiocManager"))
BiocManager::install(c ("NOISeq", "ComplexHeatmap", "TCGAbiolinks", "limma"))
BiocManager::install(c("clusterProfiler", "org.Hs.eg.db", "DOSE", "enrichplot"))
```

Este proceso toma unos minutos y sólo tendríamos que realizarlo una vez.

Sin embargo, dado que estamos utilizando los ordenadores de Google-Colab, vamos a proponer otra forma de instalación que resulta más rápida: montar las bibliotecas necesarias en una carpeta en tu unidad de Google Drive. Este proceso se describe en los siguientes pasos, y tendrás que seguir todos y cada uno de estos pasos para poder ejecutar los códigos del resto del Módulo en Google Colab.

Instrucciones de instalación de paquetes de R y Bioconductor en Google Colab

Para instalar las bibliotecas de R y Bioconductor en Google Colab necesitamos que sigas los siguientes pasos:

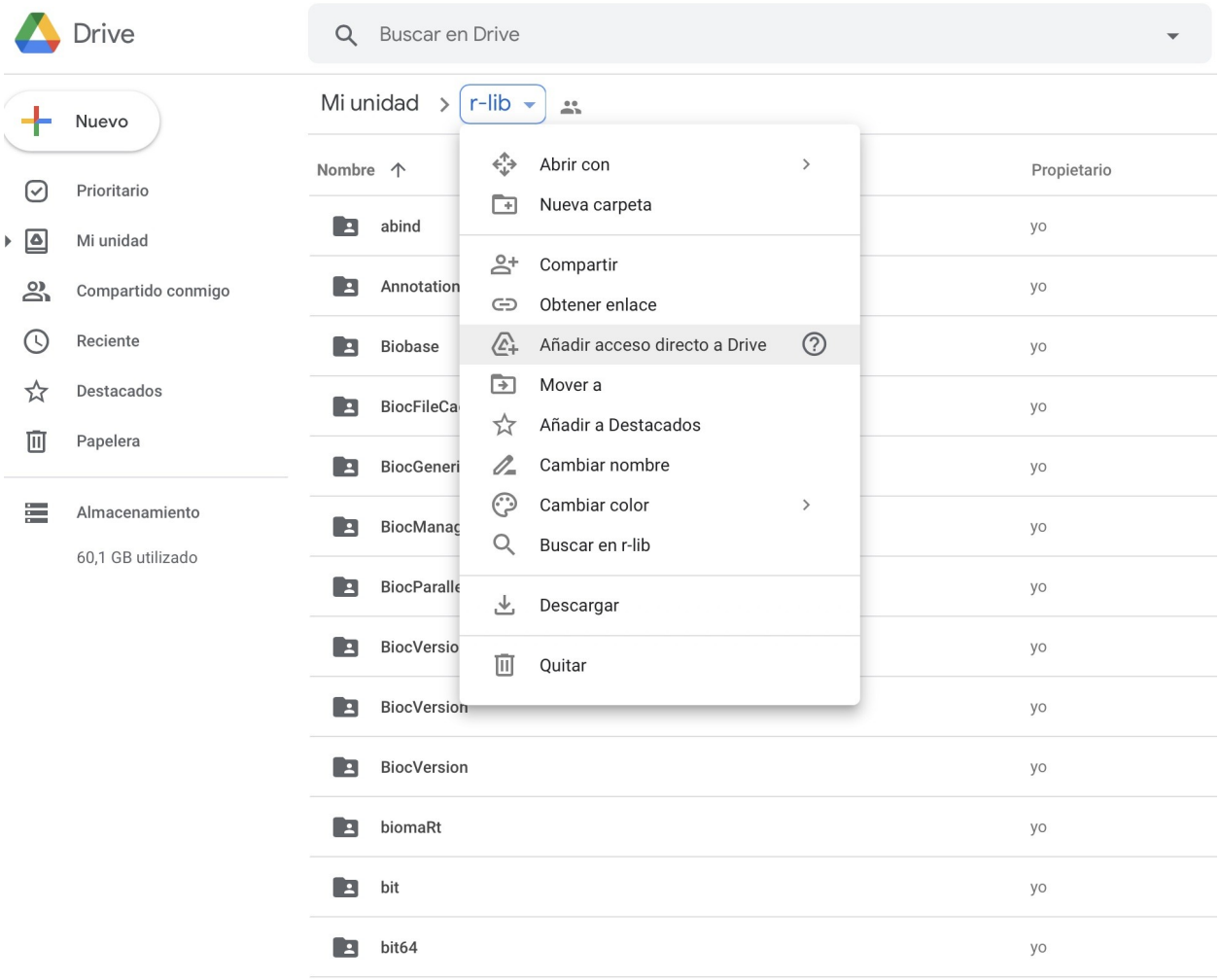
1. Ejecuta la siguiente celda para instalar algunas librerías de python que necesitamos

```
!apt-get install libcairo2-dev libjpeg-dev libgif-dev
!pip install pycairo
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libcairo2-dev is already the newest version (1.16.0-5ubuntu2).
libgif-dev is already the newest version (5.1.9-2build2).
libjpeg-dev is already the newest version (8c-2ubuntu10).
0 upgraded, 0 newly installed, 0 to remove and 43 not upgraded.
Requirement already satisfied: pycairo in
/usr/local/lib/python3.10/dist-packages (1.26.0)
```

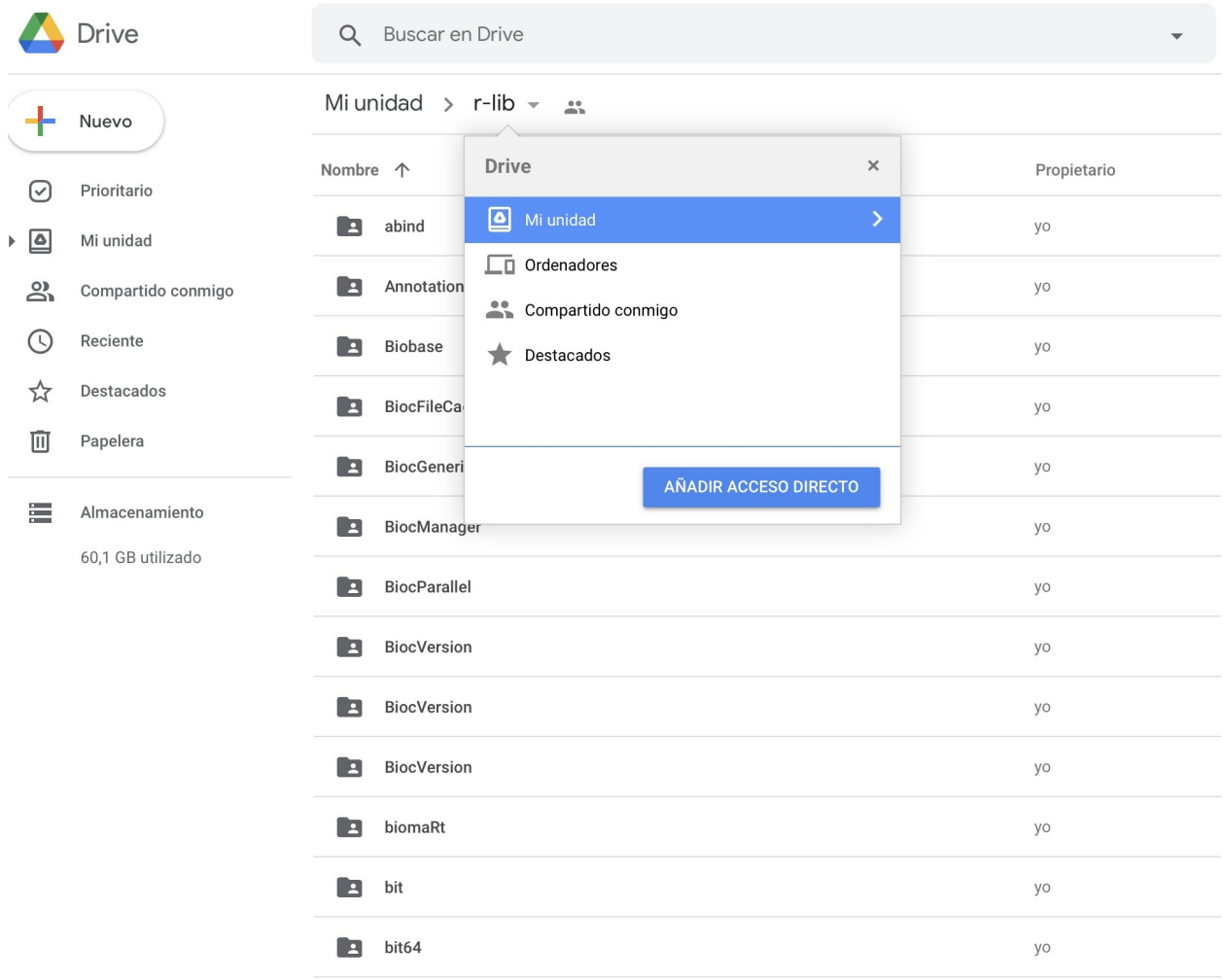
1. El siguiente enlace contiene una carpeta de Google Drive que dispone de todas las bibliotecas que necesitas para ejecutar los Notebooks de las Cápsulas 1, 2 y 3. Pincha en el siguiente enlace para abrir la carpeta de bibliotecas en tu unidad de Drive ("Shared with me" / "Compartido conmigo"): https://drive.google.com/drive/folders/1cfpAt7f-081eNOxn1wCf--VPGccuPjT8?usp=sharing

2. Pincha con el botón derecho sobre la carpeta r-lib y elige la opción "Add shortcut to drive"/"Añadir acceso directo a drive" para añadir la carpeta r-lib a "Tu Unidad" de Google Drive.



Después necesitarás indicar en qué lugar de tu Unidad/Drive deseas almacenar la carpeta compartida. Elige la carpeta "Mi unidad"/"My drive" y haz click en el botón "Añadir acceso directo"/"Add shorcut".

Nota: Si lo prefieres, puedes elegir cualquier carpeta dentro de "Mi unidad", pero tendrás que recordar la ruta a esa carpeta y modificar los ejemplos siguientes de forma acorde. Para facilitar el trabajo, te recomendamos que selecciones simplemente "Mi unidad"/"My Drive" y añadas el acceso directo.



1. Ejecuta la celda de código que aparece más abajo para conectar tu unidad de Drive a Google Colab. Google Colab te pedirá que entres con tu usuario y contraseña a Google Drive (*Conectar con Google Drive*) y que autorices a Google Colab a acceder a tu Drive (*Permitir*). Google Colab y Google Drive son dos herramientas de Google, con este proceso estás permitiendo que se comuniquen entre si (y no supone un riesgo para la seguridad de tus documentos de Google Drive).

```
from google.colab import drive
drive.mount('/content/mydrive', force_remount=True)
```

Si obtienes el mensaje `Mounted at /content/mydrive`, el proceso habrá terminado con éxito.

1. Ya tienes acceso a las bibliotecas de R que necesitamos para este Módulo. Para que las librerías se importen de la carpeta r-lib de tu Google Drive, ejecuta las siguientes celdas de código.

```
%load_ext rpy2.ipython
```

```
The rpy2.ipython extension is already loaded. To reload it, use:
  %reload_ext rpy2.ipython
```

```
%%R
#añadimos la carpeta MyDrive/r-lib a la ruta donde se buscan las
bibliotecas
.libPaths( c( "/content/mydrive/MyDrive/r-lib" , .libPaths() ) )
#vemos como queda la ruta de bibliotecas
.libPaths()
```

```
[1] "/content/mydrive/.shortcut-targets-by-id/1cfpAt7f-081eNOxn1wCf--
VPGccuPjT8/r-lib"
[2] "/usr/local/lib/R/site-library"

[3] "/usr/lib/R/site-library"

[4] "/usr/lib/R/library"
```

1. Realizamos comprobaciones cargando algunas de las bibliotecas que necesitaremos durante este Notebook. Si es la primera vez que cargas estas bibliotecas, este proceso puede tomar unos minutos, pero ya tendrás las bibliotecas listas para continuar con el resto del *notebook*.

```
%%R

library(TCGAbiolinks)
library(SummarizedExperiment)

print(sessionInfo())
```

```
R version 4.3.3 (2024-02-29)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 22.04.3 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-
r0.3.20.so;  LAPACK version 3.10.0

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

time zone: Etc/UTC
tzcode source: system (glibc)

attached base packages:
```

```
[1] stats4    tools     stats     graphics  grDevices utils
datasets
[8] methods   base

other attached packages:
 [1] SummarizedExperiment_1.32.0 GenomicRanges_1.54.1
 [3] GenomeInfoDb_1.38.8         MatrixGenerics_1.14.0
 [5] matrixStats_1.2.0           TCGAbiolinks_2.30.0
 [7] org.Hs.eg.db_3.18.0         AnnotationDbi_1.64.1
 [9] IRanges_2.36.0              S4Vectors_0.40.2
[11] Biobase_2.62.0              BiocGenerics_0.48.1
[13] clusterProfiler_4.10.1

loaded via a namespace (and not attached):
  [1] RColorBrewer_1.1-3          jsonlite_1.8.8
  [3] magrittr_2.0.3              farver_2.1.1
  [5] fs_1.6.3                    zlibbioc_1.48.2
  [7] vctrs_0.6.5                 memoise_2.0.1
  [9] RCurl_1.98-1.14             ggtree_3.10.1
 [11] S4Arrays_1.2.1              progress_1.2.3
 [13] curl_5.0.2                  SparseArray_1.2.4
 [15] gridGraphics_0.5-1          plyr_1.8.9
 [17] cachem_1.0.8                igraph_2.0.3
 [19] lifecycle_1.0.4             pkgconfig_2.0.3
 [21] Matrix_1.6-5                R6_2.5.1
 [23] fastmap_1.1.1               gson_0.1.0
 [25] GenomeInfoDbData_1.2.11     digest_0.6.34
 [27] aplot_0.2.2                 enrichplot_1.22.0
 [29] colorspace_2.1-0            patchwork_1.2.0
 [31] RSQLite_2.3.5               filelock_1.0.3
 [33] fansi_1.0.6                 httr_1.4.7
 [35] polyclip_1.10-6             abind_1.4-5
 [37] compiler_4.3.3              bit64_4.0.5
 [39] withr_3.0.0                 downloader_0.4
 [41] BiocParallel_1.36.0         viridis_0.6.5
 [43] DBI_1.2.1                   ggforce_0.4.2
 [45] biomaRt_2.58.2              MASS_7.3-60.0.1
 [47] rappdirs_0.3.3              DelayedArray_0.28.0
 [49] HDO.db_0.99.1               ape_5.7-1
 [51] scatterpie_0.2.1            glue_1.7.0
 [53] nlme_3.1-163                GOSemSim_2.28.1
 [55] grid_4.3.3                  shadowtext_0.1.3
 [57] reshape2_1.4.4              fgsea_1.28.0
 [59] generics_0.1.3              gtable_0.3.4
 [61] tzdb_0.4.0                  tidyr_1.3.1
 [63] data.table_1.15.0           hms_1.1.3
 [65] tidygraph_1.3.1             xml2_1.3.6
 [67] utf8_1.2.4                  XVector_0.42.0
 [69] ggrepel_0.9.5               pillar_1.9.0
```

```
 [71] stringr_1.5.1               yulab.utils_0.1.4
 [73] splines_4.3.3               dplyr_1.1.4
 [75] tweenr_2.0.3                BiocFileCache_2.10.1
 [77] treeio_1.26.0               lattice_0.22-5
 [79] bit_4.0.4                   tidyselect_1.2.0
 [81] GO.db_3.18.0                Biostrings_2.70.3
 [83] knitr_1.45                  gridExtra_2.3
 [85] xfun_0.41                   graphlayouts_1.1.1
 [87] stringi_1.8.3               lazyeval_0.2.2
 [89] ggfun_0.1.4                 TCGAbiolinksGUI.data_1.22.0
 [91] codetools_0.2-19            ggraph_2.2.1
 [93] tibble_3.2.1                qvalue_2.34.0
 [95] ggplotify_0.1.2             cli_3.6.2
 [97] munsell_0.5.0               Rcpp_1.0.12
 [99] dbplyr_2.4.0                png_0.1-8
[101] XML_3.99-0.16.1             parallel_4.3.3
[103] ggplot2_3.5.0               readr_2.1.5
[105] blob_1.2.4                  prettyunits_1.2.0
[107] DOSE_3.28.2                 bitops_1.0-7
[109] viridisLite_0.4.2           tidytree_0.4.6
[111] scales_1.3.0                purrr_1.0.2
[113] crayon_1.5.2                rlang_1.1.3
[115] rvest_1.0.3                 cowplot_1.1.3
[117] fastmatch_1.1-4             KEGGREST_1.42.0
```

Si obtienes un error del tipo `Error in library (...): there is no package called '...'`, significa que no se han instalado bien las bibliotecas. Vuelve a ejecutar los pasos anteriores 1 a 6 para comprobar que todos se hayan ejecutado correctamente.

En otro caso, obtendrás un listado de bibliotecas instaladas de R y su número de versión, similar al siguiente:

```
R version 4.3.3 (2024-02-29)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 22.04.3 LTS

[Y aquí se listan las bibliotecas instaladas de R y su número de versión]
```

Con esto, el proceso de instalación habrá terminado con éxito. Puedes continuar a la siguiente sección.

Recuerda: Si durante tus ejercicios del módulo 2 obtienes un error del tipo:

`Error: package '...' could not be loaded`

o

`Error in library (...): there is no package called '...'`

indicará que tu sesión ha expirado y debes volver a realizar los pasos anteriores (1 a 6) para cargar las bibliotecas de nuevo.

4. DESCARGA DE DATOS DE TCGA

Antes de comenzar, cargamos las bibliotecas de R que necesitamos para esta parte del *notebook*

```
%%R
# Cargar bibliotecas


library(TCGAbiolinks)
library(SummarizedExperiment)
library(pheatmap)
library(limma)


WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
Attaching package: ‘limma’


WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: The
following object is masked from ‘package:BiocGenerics’:

    plotMA
```

El proyecto TCGA permite a cualquier usuario acceder a sus datos y descargarlos de forma gratuita. Un ejemplo de código en R para la descarga de una matriz de expresión para muestras TCGA-SKCM es el siguiente:

```
# Descarga los datos de expresion de TCGA-SKCM. Necesitamos tres
pasos:

# 1- Lanza la consulta para recuperar todos los datos que satisfagan
los criterios de búsqueda
query <- GDCquery(project = "TCGA-SKCM",
                  data.category = "Gene expression",
                  data.type = "Gene expression quantification",
                  legacy=TRUE,
                  file.type= "normalized_results"
                  )


# 2- Descarga los datos
GDCdownload(query)
# 3- Organiza los datos y los guarda en la variable
normRSEMtranscr.counts
normRSEMtranscr.counts <- GDCprepare(query)
```

En la próxima cápsula se indicarán en detalle todos los pasos relacionados con la descarga y normalización de los datos. Sin embargo, para acelerar la descarga y ejecución de este *notebook*, te pedimos que utilices el código de la siguiente celda para descargar una matriz de expresión completa ya normalizada mediante el método RSEM.

```
%%R
# Hemos descargado previamente los datos de expresion de TCGA-SKCM con
normalización RSEM y los ponemos a tu disposición en un fichero

normRSEMtranscr.counts <-
readRDS("/content/mydrive/MyDrive/r-lib/normRSEMtranscr.counts.RDS")
```

Además de datos generados por distintas técnicas experimentales en ciencias -ómicas, TCGA pone a disposición de la comunidad los datos clínicos de las muestras de sus estudios. Existen muchos tipos de datos clínicos asociados a las muestras: el tratamiento (fármacos y dosis), estadío tumoral, recurrencia, tipo de radiación, información clínica del paciente, etc. El siguiente código ilustra cómo pueden recuperarse los datos clínicos de los pacientes (`patient`), los tratamientos (`drug`) y recidivas (`new_tumor_event`) para diez de las muestras del estudio TCGA-SKCM

```
%%R

## Descargar datos clínicos adicionales
# 1- Lanza la consulta para recuperar todos los datos que satisfagan
los criterios de búsqueda
subsetSamples=c("TCGA-EB-A5SF","TCGA-EE-A3J8","TCGA-EB-A430","TCGA-BF-
AAP1","TCGA-EE-A3J3","TCGA-EB-A550","TCGA-FR-A3YO","TCGA-YD-
A9TA","TCGA-EB-A5FP","TCGA-EB-A3XB")
query<-GDCquery(project = "TCGA-SKCM",
                data.category = "Clinical",
                data.type = "Clinical Supplement",
                data.format = "BCR XML",
                barcode = subsetSamples)
# 2- Descarga los datos
GDCdownload(query)

# 3- Organiza los datos y los guarda en sendas variables
## Selecciona el tipo de datos clínicos a descargar en el argumento
clinical.info=, puede ser:
##    drug, admin, follow_up, radiation, patient, stage_event,
new_tumor_event

clinical.patient<-GDCprepare_clinic(query, clinical.info = "patient")
#basic info
clinical.drug <- GDCprepare_clinic(query, clinical.info = "drug")
#treatment info
clinical.new_tumor_event<-GDCprepare_clinic(query, clinical.info =
"new_tumor_event") #new tumor event
```

```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
--------------------------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: o GDCquery:
Searching in GDC database

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
--------------------------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Genome of
reference: hg38

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
------------------------------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: oo
Accessing GDC. This might take a while...

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
------------------------------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: ooo
Project: TCGA-SKCM

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
--------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: oo
Filtering results

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
--------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: ooo By
data.format

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: ooo By
data.type

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: ooo By
barcode

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
----------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: oo Checking
data

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
----------------
```

```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: ooo
Checking if there are duplicated cases

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: ooo
Checking if there are results for the query

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: o Preparing
output

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Downloading
data for project TCGA-SKCM

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: GDCdownload
will download 10 files. A total of 369.949 KB

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Downloading
as: Wed_Apr_17_10_07_27_2024.tar.gz


  |
==================================================================
| 100%

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: To get the
following information please change the clinical.info argument

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =>
new_tumor_events: new_tumor_event
=> drugs: drug
=> follow_ups: follow_up
=> radiations: radiation

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Parsing
follow up version: follow_up_v2.0


  |
==================================================================
| 100%

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Adding
stage event information
```

```
  |
================================================================================
| 100%

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Updating
days_to_last_followup and vital_status from follow_up information
using last entry

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Parsing
follow up version: follow_up_v2.0


  |
================================================================================
| 100%
  |=============================================================
|  90%
  |
================================================================================
| 100%
```

La información clínica de estas muestras junto con otra información derivada de distintos estudios -ómicos sobre las mismas también está a nuestra disposición en una tabla de datos en formato Excel.

Los datos de expresión genética y la tabla de datos de información clínica será utilizada en distintos módulos de este curso.

5. ESTRUCTURA DE LOS DATOS DE TCGA

Tras la descarga de datos, en esta sección proponemos un primer acercamiento a la inspección de los mismos para comprender su estructura y naturaleza, aunque será en la siguiente cápsula en la que abordaremos un análisis exploratorio completo.

5.1. Datos de expresión

Observa que los datos descargados se han guardado en la variable `normRSEMtranscr.counts`. Echamos un vistazo a su estructura

```
%%R
normRSEMtranscr.counts

class: RangedSummarizedExperiment
dim: 19947 473
metadata(1): data_release
assays(1): normalized_count
rownames(19947): A1BG A2M ... TICAM2 SLC25A5-AS1
rowData names(3): gene_id entrezgene ensembl_gene_id
colnames(473): TCGA-D9-A4Z6-06A-12R-A266-07
  TCGA-EE-A2MQ-06A-11R-A18S-07 ... TCGA-DA-A1I2-06A-21R-A18U-07
```

```
  TCGA-D3-A2JK-06A-11R-A18S-07
colData names(179): sample patient ... subtype_DIPYRIM.C.T.n.C.T..mut
  subtype_SHATTERSEEK_Chromothripsis_calls
```

De forma muy breve, en los datos de expresión génica obtenidos mediante secuenciación masiva cuantificamos la expresión de un gen mediante el número de lecturas o reads que mapean en las coordenadas genómicas correspondientes. A partir de la cuantificación y normalización del número de lecturas se obtiene una matriz, donde cada fila representa un gen y cada columna una muestra. La estructura nos indica que esta matriz de expresión contiene, en este caso, valores de expresión de `19947` genes para `473` muestras distintas. De un vistazo, podemos apreciar el listado de nombres de genes en las *rownames* (`A1BG A2M ...`) y el de nombres de muestras en las *colnames* (`TCGA-D9-A4Z6-06A-12R-A266-07  TCGA-EE-A2MQ-06A-11R-A18S-07 ...`). Estos listados están truncados para que toda la información principal pueda aparecer en unas pocas líneas en pantalla.

Para facilitar el proceso de análisis posterior, vamos a descomponer esta estructura en tres partes: la matriz de expresión genética (se guarda en la variable `data`), la información sobre los genes (variable `genes.info`) y la información sobre las muestras (variable `sample.info`)

```
%%R
data<-assay(normRSEMtranscr.counts)
genes.info<-rowRanges(normRSEMtranscr.counts)
sample.info<-colData(normRSEMtranscr.counts)
```

También podemos almacenar los datos más relevantes que necesitemos conservar en ficheros de texto, para su recuperación en posteriores *notebooks*

```
%%R
write.table(data, file= "exprMatrix_prep_RSEM.tsv", sep="\t")
```

Recuerda que todas las celdas de código que empiezan por `%%R` contienen código del lenguaje R, y el resto de celdas contiene código en Python. Después de descargar los datos utilizando funciones y bibliotecas de R, los datos de interés han quedado almacenados en una serie de variables de R: `data`, `genes.info`, `sample.info`, etc. Si ahora quisiéramos analizar esa información utilizando bibliotecas de Python, podemos guardar las variables de R en variables de Python como ilustra la siguiente celda de código:

```
# nombre_variable_en_python = %R nombre_variable_en_R
data = %R data
genes = %R genes.info
samples = %R sample.info  #por ejemplo: la información de la variable
sample.info de R se guarda en la variable samples de python
clinica = %R clinical.patient
tratamientos = %R clinical.drug
recurrencia = %R clinical.new_tumor_event
```

El aspecto de una matriz de expresión es el que muestra la siguiente celda de código. Observa la disposición matricial con un gen por cada fila (19947 en total - numeradas desde la fila 0 a la fila 19946) y una muestra por cada columna (473 en total).

```
# Esto ya es código de Python (observa que no aparece el %%R)
# la variable "data" de python contiene la matriz de expresión
import pandas as pd
pd.DataFrame(data)
```

```
{"type":"dataframe"}
```

En la siguiente cápsula iniciaremos un análisis exploratorio de estos datos y abordaremos dos pasos imprescindibles antes de cualquier análisis computacional: el preprocesamiento y la normalización de los datos.

5.2. Datos clínicos y resultados de estudios -ómicos

Además de los datos de expresión genética, también estaban disponibles datos clínicos de las muestras y otros resultados derivados de distintos estudios -ómicos que pueden ser interesantes para identificar patrones o relaciones novedosas en los datos.

El siguiente código ilustra como descargar una tabla en formato excel desde una URL, guardar la tabla en una variable (`base_datos`) y visualizar su contenido para una primera exploración

```
# Importamos la librería pandas con el alias 'pd'
import pandas as pd

# Almacenamos el enlace a nuestros datos en la variable 'url_datos'
url_datos = 'https://drive.google.com/uc?id=1Wjyktizno4tUt8bjnBxpX-
XUZDAWXWa4'

# El método read_excel permite leer un libro de Excel
# El parámetro 'sheet_name' indica la hoja que nos interesa y
'usecols' nos permite
# especificar el conjunto de columnas que queremos leer (ambos son
parámetros opcionales)
base_datos = pd.read_excel(url_datos, sheet_name='Supplemental Table
S1D', header=1, na_values='-')

# Mostramos la tabla completa
base_datos
```

```
{"type":"dataframe","variable_name":"base_datos"}
```

Estos datos serán utilizados en siguientes módulos para ilustrar los distintos tipos de técnicas de análisis de este curso.

REFERENCIAS BIBLIOGRÁFICAS

- The Cancer Genome Atlas (TCGA) https://www.cancer.gov/about-nci/organization/ccg/research/structural-genomics/tcga

- TCGA Biolinks https://bioconductor.org/packages/release/bioc/html/TCGAbiolinks.html
- Cancer Genome Atlas Network. Genomic Classification of Cutaneous Melanoma. Cell. 2015;161(7):1681-1696. doi:10.1016/j.cell.2015.05.044 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4580370/
- Recursos para el aprendizaje de R y python https://guides.library.cmu.edu/bioinfo/r-and-python

---

Cápsula 2 - Preprocesamiento y análisis de datos.

Autores:

Por **Carlos Cano Gutiérrez**

Profesor Titular de la Universidad de Granada.
Departamento de Ciencias de Computación e Inteligencia Artificial.

Por **Pedro Carmona Sáez**

Profesor Titular de la Universidad de Granada.
Departamento de Estadística e Investigación Operativa.

ÍNDICE

En este *notebook* se establence una introducción a algunos de los pasos en el análisis de datos omicos y qué problemas se plantean así cómo técnicas que se aplican, sobre las cuales se profundizará en los siguientes módulos. Estos son:

1. Preprocesamiento y normalización.
2. Manejo de datos de expresión genética.
3. Uso de algunas funciones útiles del lenguaje de programación R para realizar representaciones gráficas de los datos.
4. Tipos de análisis y representaciones gráficas.

Contenidos:

ANTES DE EMPEZAR

Este *notebook* utiliza las bibliotecas y los datos que se han descargado en el *notebook* anterior (Módulo 2, Cápsula 1). Si estás empezando ahora una nueva sesión en Google Colab, o tu sesión de la Cápsula anterior ha caducado, necesitarás re-ejecutar las celdas del *notebook* anterior antes de ejecutar este.

Para saber si tu sesión es nueva o la anterior ha caducado, prueba a ejecutar este código

```
%%R

print(sessionInfo())
```

```
R version 4.3.3 (2024-02-29)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 22.04.3 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3;
LAPACK version 3.10.0

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

time zone: Etc/UTC
tzcode source: system (glibc)

attached base packages:
[1] stats4    tools     stats     graphics  grDevices utils
datasets
[8] methods   base

other attached packages:
 [1] limma_3.58.1                pheatmap_1.0.12
 [3] SummarizedExperiment_1.32.0 Biobase_2.62.0
 [5] GenomicRanges_1.54.1        GenomeInfoDb_1.38.8
 [7] IRanges_2.36.0              S4Vectors_0.40.2
 [9] BiocGenerics_0.48.1         MatrixGenerics_1.14.0
[11] matrixStats_1.2.0           TCGAbiolinks_2.30.0

loaded via a namespace (and not attached):
 [1] tidyselect_1.2.0            dplyr_1.1.4
 [3] blob_1.2.4                  R.utils_2.12.3
 [5] filelock_1.0.3              Biostrings_2.70.3
 [7] bitops_1.0-7                fastmap_1.1.1
 [9] RCurl_1.98-1.14             BiocFileCache_2.10.1
[11] XML_3.99-0.16.1             digest_0.6.34
[13] lifecycle_1.0.4             statmod_1.5.0
[15] KEGGREST_1.42.0             RSQLite_2.3.5
[17] magrittr_2.0.3              compiler_4.3.3
[19] rlang_1.1.3                 progress_1.2.3
```

```
[21] utf8_1.2.4                   data.table_1.15.0
[23] knitr_1.45                   prettyunits_1.2.0
[25] S4Arrays_1.2.1               bit_4.0.4
[27] curl_5.0.2                   DelayedArray_0.28.0
[29] RColorBrewer_1.1-3           plyr_1.8.9
[31] xml2_1.3.6                   abind_1.4-5
[33] withr_3.0.0                  purrr_1.0.2
[35] R.oo_1.26.0                  grid_4.3.3
[37] fansi_1.0.6                  colorspace_2.1-0
[39] ggplot2_3.5.0                scales_1.3.0
[41] biomaRt_2.58.2               cli_3.6.2
[43] crayon_1.5.2                 generics_0.1.3
[45] httr_1.4.7                   tzdb_0.4.0
[47] DBI_1.2.1                    cachem_1.0.8
[49] stringr_1.5.1                zlibbioc_1.48.2
[51] rvest_1.0.3                  AnnotationDbi_1.64.1
[53] TCGAbiolinksGUI.data_1.22.0 XVector_0.42.0
[55] vctrs_0.6.5                  Matrix_1.6-5
[57] jsonlite_1.8.8               hms_1.1.3
[59] bit64_4.0.5                  tidyr_1.3.1
[61] glue_1.7.0                   stringi_1.8.3
[63] gtable_0.3.4                 munsell_0.5.0
[65] tibble_3.2.1                 pillar_1.9.0
[67] rappdirs_0.3.3               GenomeInfoDbData_1.2.11
[69] R6_2.5.1                     dbplyr_2.4.0
[71] lattice_0.22-5               readr_2.1.5
[73] R.methodsS3_1.8.2            png_0.1-8
[75] memoise_2.0.1                Rcpp_1.0.12
[77] SparseArray_1.2.4            downloader_0.4
[79] xfun_0.41                    pkgconfig_2.0.3
```

Si obtienes un error del tipo `UsageError: Cell magic %%R not found.`, significa que tu sesión es nueva o ha caducado, y tienes que volver a ejecutar el código del Módulo 2 Cápsula 1.

Si la celda se ejecuta correctamente y obtienes un mensaje que comienza así:

```
R version 4.2.2 Patched (2022-11-10 r83330)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.5 LTS

[Se omite listado de bibliotecas instaladas de R y su número de
versión]
```

significa que tu sesión continúa con las bibliotecas activas. Puedes seguir con el resto del *notebook*.

1. PREPROCESAMIENTO Y NORMALIZACIÓN DE LOS DATOS DE TCGA

El **preprocesamiento** de datos es una etapa de todo proceso computacional donde los datos se preparan para su posterior tratamiento y análisis. Esta etapa incluye cualquier tipo de transformación, re-estructuración, filtrado, eliminación de ruido, manejo de valores perdidos, etc. En machine learning, este tipo de transformaciones sobre los datos suele denominarse manipulación de datos o **Data *wrangling***. Algunas formas de preprocesamiento habituales en datos ómicos son el filtrado de datos de mala calidad, filtrado de genes con poca variacibilidad, imputación o filtrado de casos/genes con valores perdidos, etc.

La **normalización** es un proceso de transformación de los datos con el que se pretende estadarizar o eliminar efectos o sesgos debidos a la variabilidad asociada a la experimentación o inherente a la muestra para el posterior análisis de estos datos. Por ejemplo, es habitual transformar el valor de variables numéricas continuas de distinta magnitud a la escala $[0,1)$ para que puedan combinarse o compararse entre sí. También se suele aplicar una tranformación a escala logarítmica de datos de expresión genética, ya que las propiedades de las distribuciones logarítmicas resultan más convenientes para representar esta información y facilita análisis posteriores.

Las técnicas de normalización permiten corregir en parte la variabilidad o ruido inherente a las técnicas experimentales en ciencias -ómicas. De este modo, estas técnicas resultan fundamentales antes de combinar datos obtenidos en distintos experimentos, incluso si han sido producidos en el mismo laboratorio, por el mismo equipo técnico y utilizando los mismos instrumentos.

La visualización de los datos obtenidos resultará fundamental para el denominado **Control de Calidad**, etapa en la que se identifican patrones anormales que señalan que existen sesgos experimentales (*batch effects*) que no han sido corregidos por los métodos de normalización.

En este *notebook* proponemos una serie de pasos para el preprocesamiento y normalización de la matriz de expresión genética descargada para muestras TCGA-SKCM en línea con lo propuesto en el artículo de investigación original que puso estos datos a disposición de la comunidad científica. Los pasos propuestos en este artículo son los siguientes:

1. Normalización del número de copias de RNAs con el método RSEM (ya aplicado sobre los datos descargados) y transformación logarítmica.
2. Centrado del valor de expresión de los genes en su mediana.
3. Selección de los 1500 genes con mayor variabilidad.
4. Cambios en el nombre de las muestras para que coincidan con los identificadores de muestra de las tablas con información clínica.

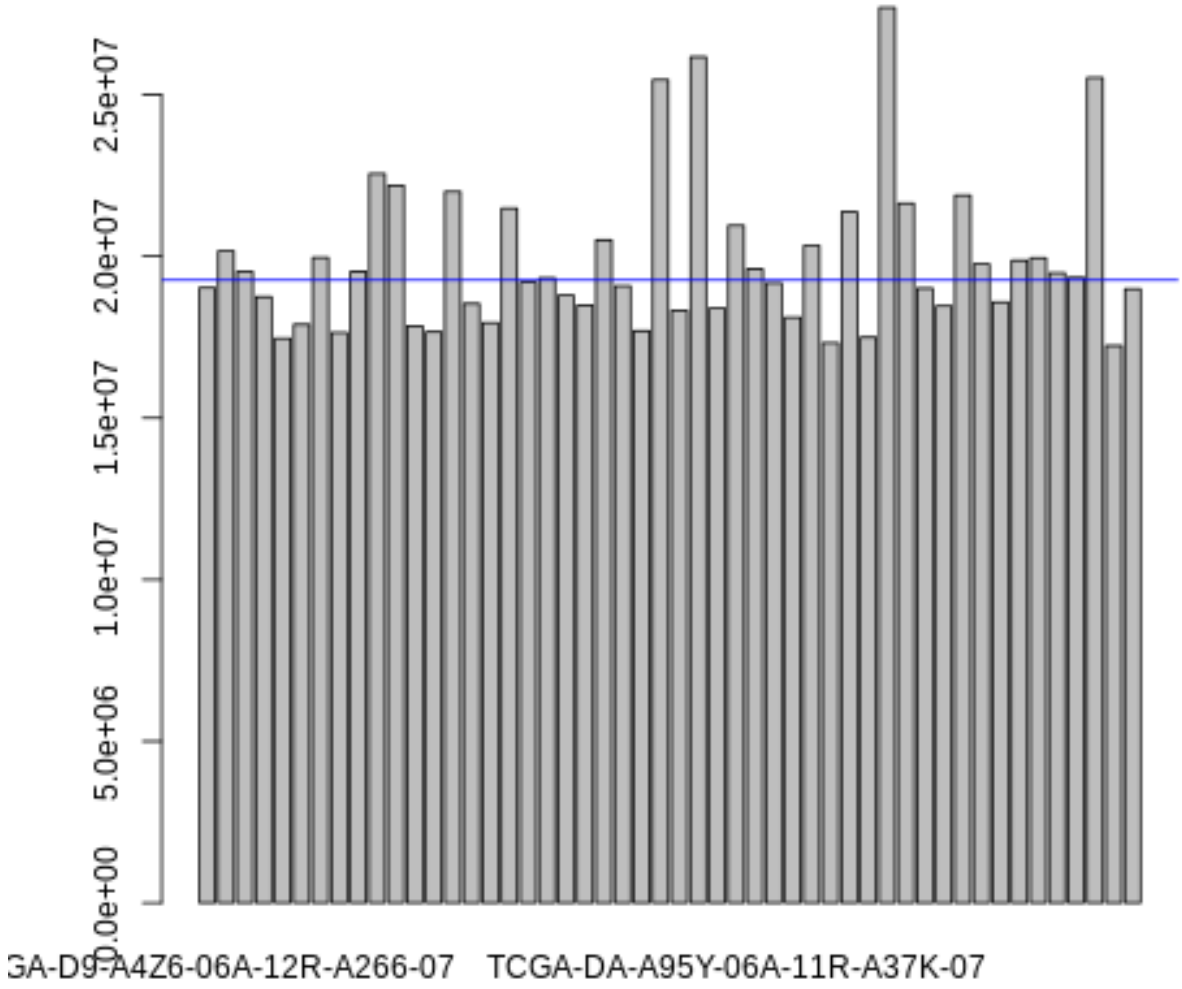
Utilizaremos distintas representaciones gráficas de datos a lo largo de este proceso para ilustar su utilidad para entender las transformaciones que se llevan a cabo en los distintos pasos.

Respecto a los datos de variables clínicas y derivadas de otros análisis -ómicos, en otros módulos se procederá a su preprocesamiento y normalización conforme sea necesario para la aplicación de las distintas técnicas. En este mismo cuaderno se propone en la sección 1.4 un pipeline alternativo de análisis para estos datos de expresión.

###1.1. Primera inspección visual de los datos

Una forma de comenzar el preprocesamiento de los datos es mediante la inspección visual de los mismos para identificar potenciales sesgos o patrones anormales. Por ejemplo, es esperable que el número total de copias de todos los genes en cada muestra sea similar. La función `colSums` permite sumar por columnas los datos de la matriz (cada columna es una muestra) y `barplot` pinta un diagrama de barras

```
%%R
#histograma con el número total de lecturas de las 50 primeras muestras
barplot(colSums(data)[1:50])
abline(h=median(colSums(data)[1:50]),col="blue") #la línea azul representa la media del número total de lecturas de las primeras 50 muestras
```



Podemos observar a simple vista que el número total de lecturas varía respecto a la media (línea azul) en algunas de las muestras. Vamos a realizar un análisis más detallado.

La función de R `TCGAanalyze_Preprocessing` calcula la correlación lineal entre pares de muestras (Coeficiente de correlación de Pearson) y representa gráficamente estas correlaciones con un mapa de calor. La casilla de fila i columna j representa el coeficiente de correlación de la muestra i y la muestra j. Como este coeficiente es simétrico ($corr(i,j) = corr(j,i)$) la matriz es también simétrica y con diagonal igual a 1.

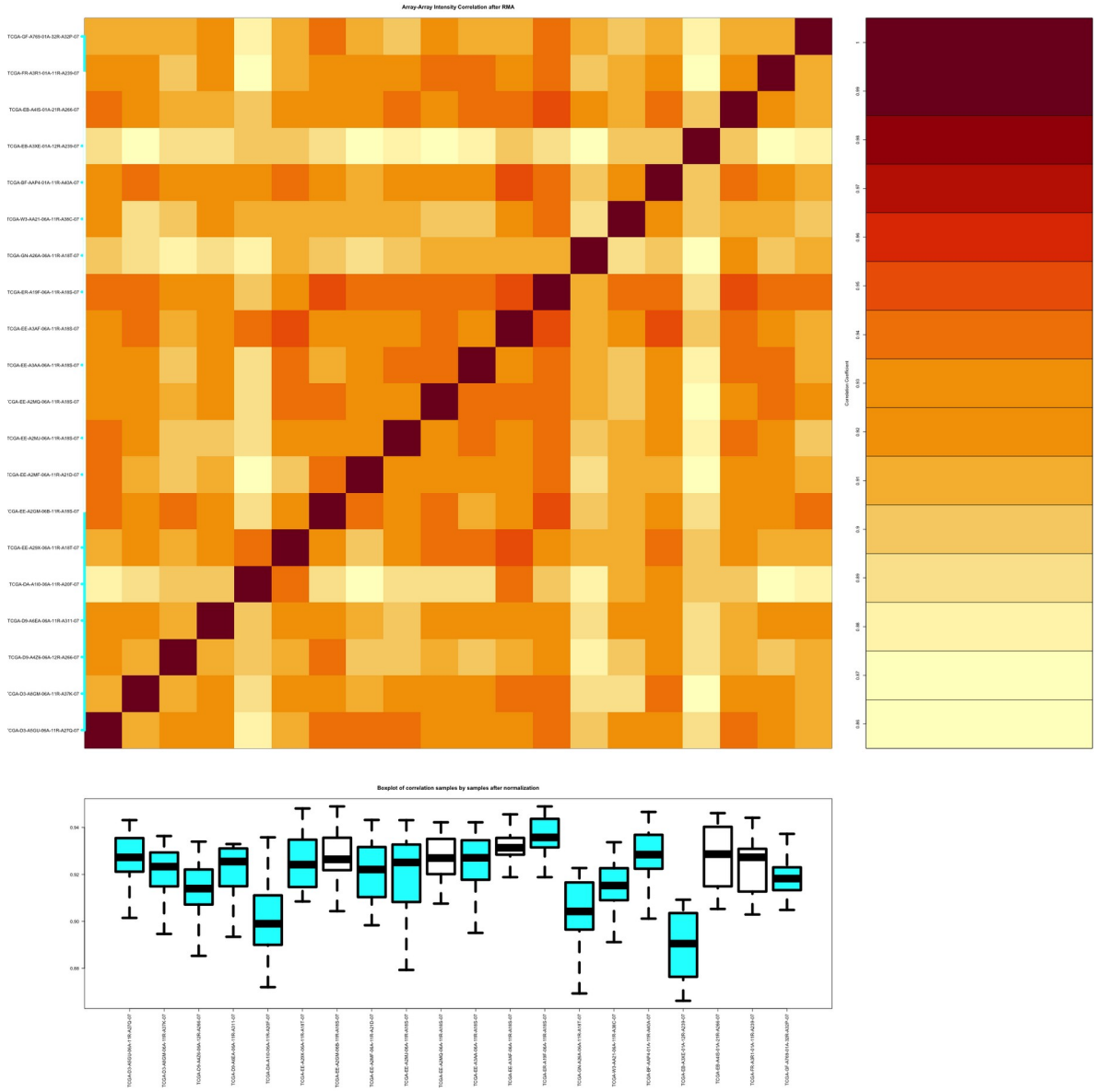
Además de la matriz de correlaciones también se representa un diagrama de cajas o *box plot* con las distribuciones de estas correlaciones para cada muestra.

Para mostrar la correlación entre muestras y boxplots de correlación se puede ejecutar el comando

TCGAanalyze_Preprocessing(normRSEMtranscr.counts, filename="sample_correlation.png", width = 2000, height = 2000)

Este comando requiere cierto tiempo de ejecución, por lo que no se ha incluído en la ejecución de este cuaderno.

Se genera una imagen resultante con el nombre `sample_correlation.png`, que se muestra aquí (corresponde a las 20 primeras muestras)

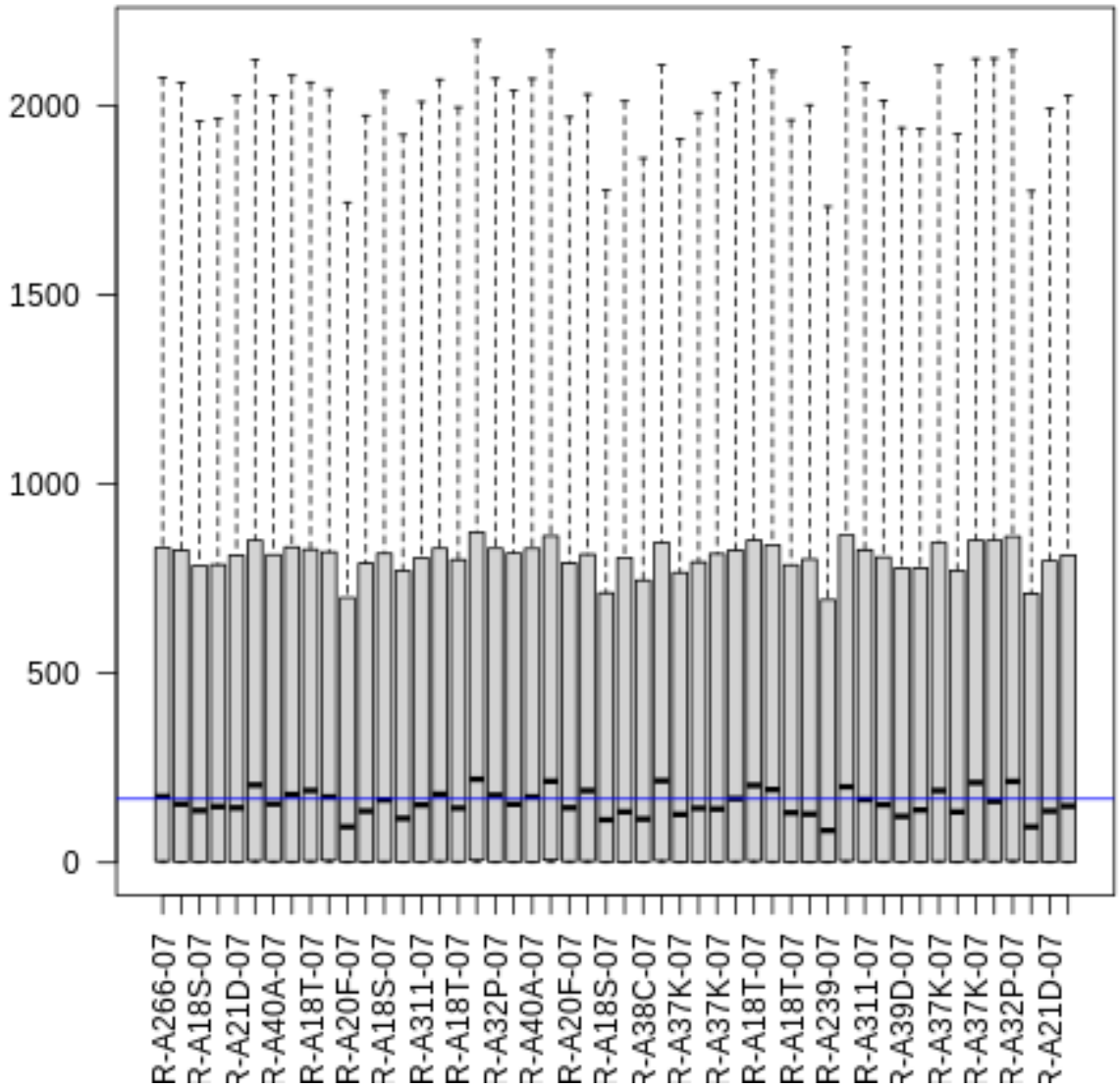


El análisis de la imagen resultante indica que existen tres muestras con un grado de correlación con el resto sensiblemente más bajo que la media. Es decir, estas tres muestras tendrían un

comportamiento diferente al resto. Visualmente, podemos identificar estas muestras en el heatmap y en el diagrama de cajas: en el eje de la X, de izquierda a derecha, las muestras de las posiciones 5, 14, 17. Especialmente la muestra 17. Es probable que, más adelante, cuando estudiemos cómo se agrupan estas muestras (Módulo 6, Cápsula de Clustering) identifiquemos que una o varias de estas muestras tienen, efectivamente, un comportamiento de *outliers* (sensiblemente diferente al resto).

Los diagramas de cajas (boxplots) también se utilizan para representar las distribuciones de valores de expresión por cada muestra para identificar segos o valores extremos (*outliers*) y validar el efecto de la normalización sobre los datos. De inicio, el diagrama de cajas sobre los datos de expresión crudos muestra lo siguiente

```
%%R
# Boxplot con las distribuciones de valores de expresión de todos los
genes en las 50 primeras muestras
boxplot(data[,1:50], outline=FALSE, las=2)
abline(h=median(data),col="blue")
```

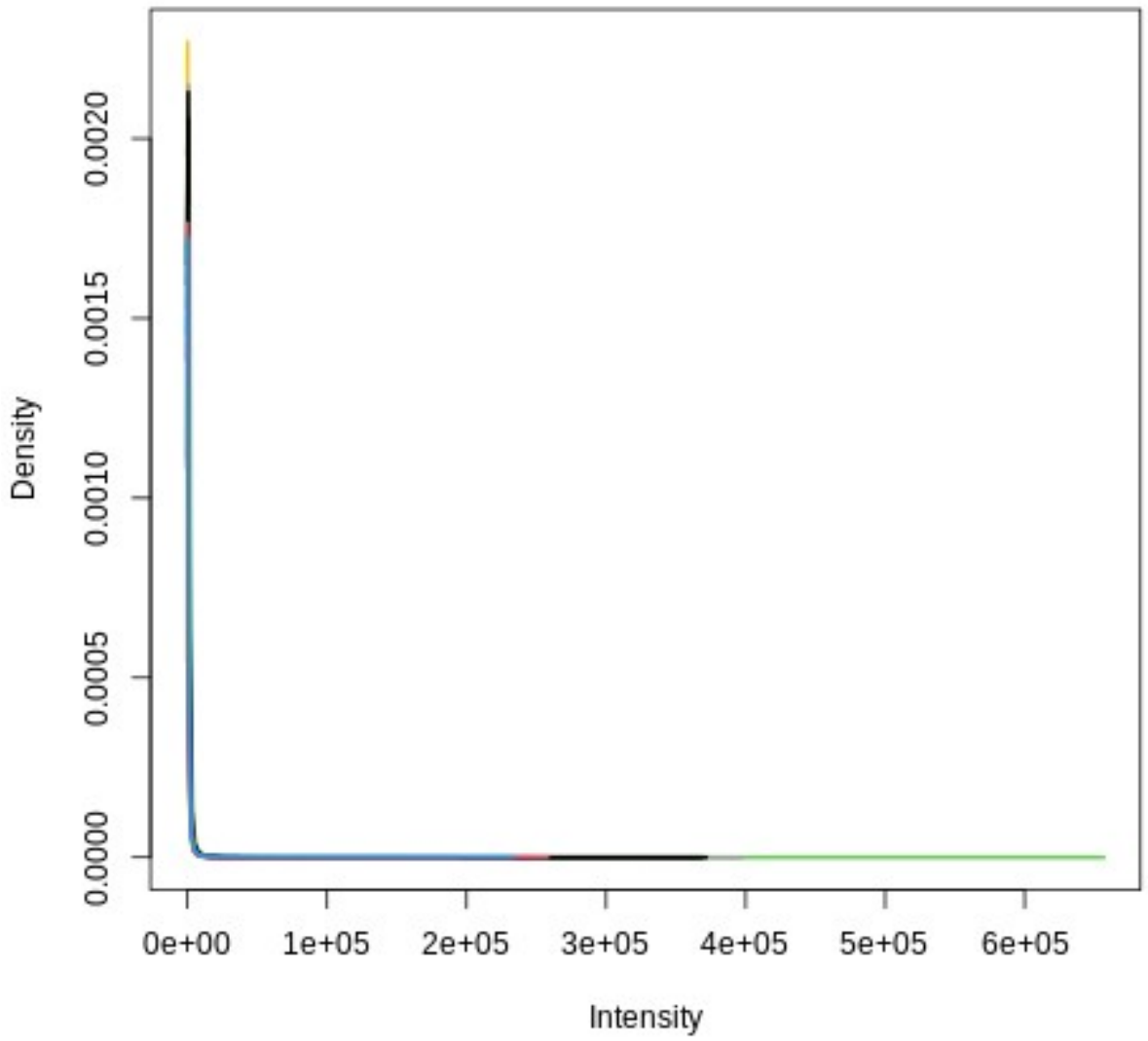


Con estos boxplots podemos observar que la distribución de los conteos de lecturas por genes presenta cierta variabilidad entre unas muestras y otras, pero son suficientemente parecidas para considerarlas en un análisis conjunto.

Si una muestra tuviera una caja muy desplazada respecto al resto (por ejemplo, su mediana estuviera mucho más alta o más baja que la línea azul horizontal), necesitaríamos prestar atención a esta muestra para comprobar si la normalización que se aplica consigue corregir esta desviación respecto al comportamiento del resto de muestras del conjunto.

También puede pintarse la distribución de densidad de la expresión para cada muestra, comparando perfiles entre las mismas

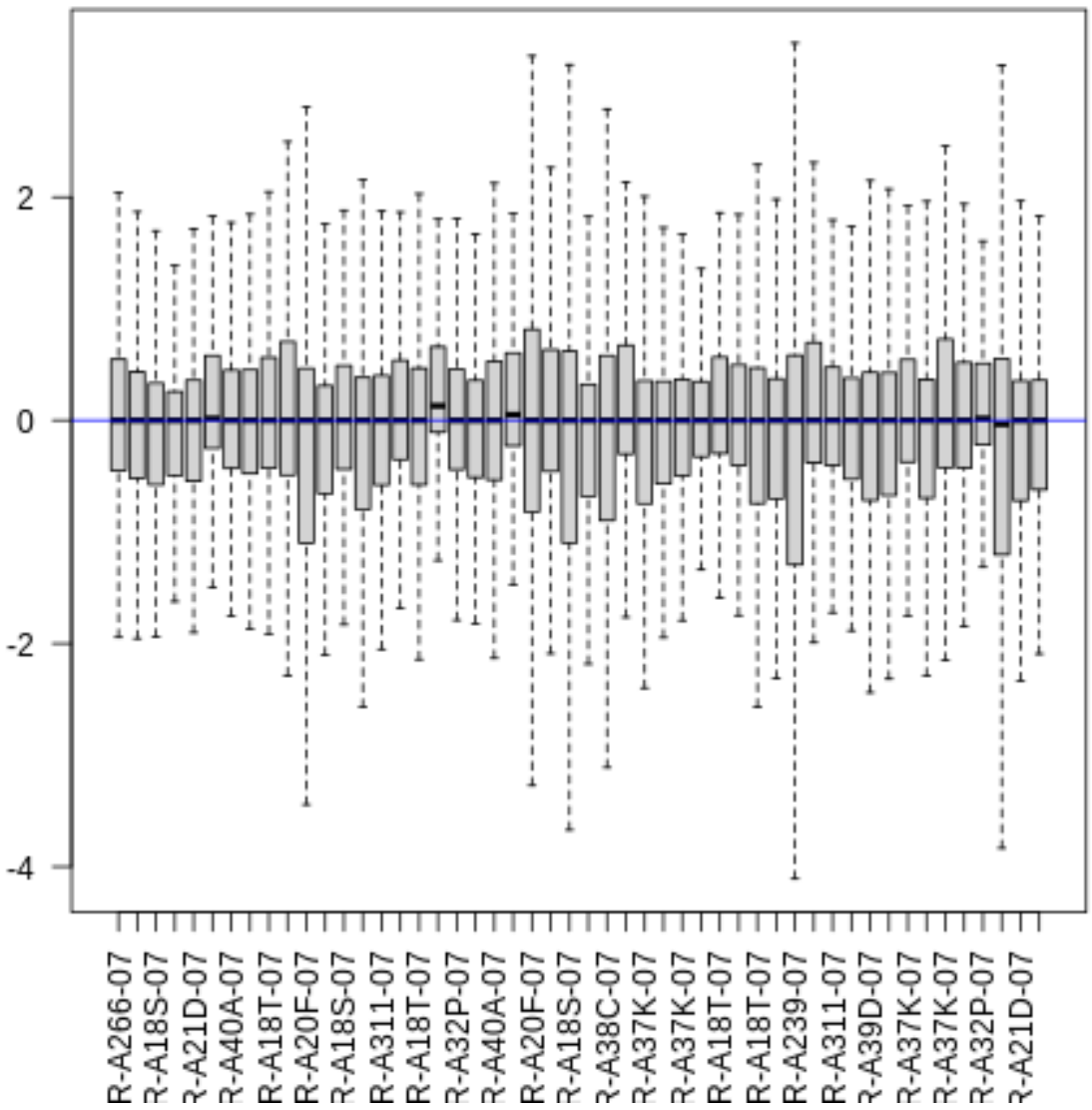
```
%%R
# Perfil de densidad para las 10 primeras muestras
plotDensities(data[,1:10], legend=FALSE)
```



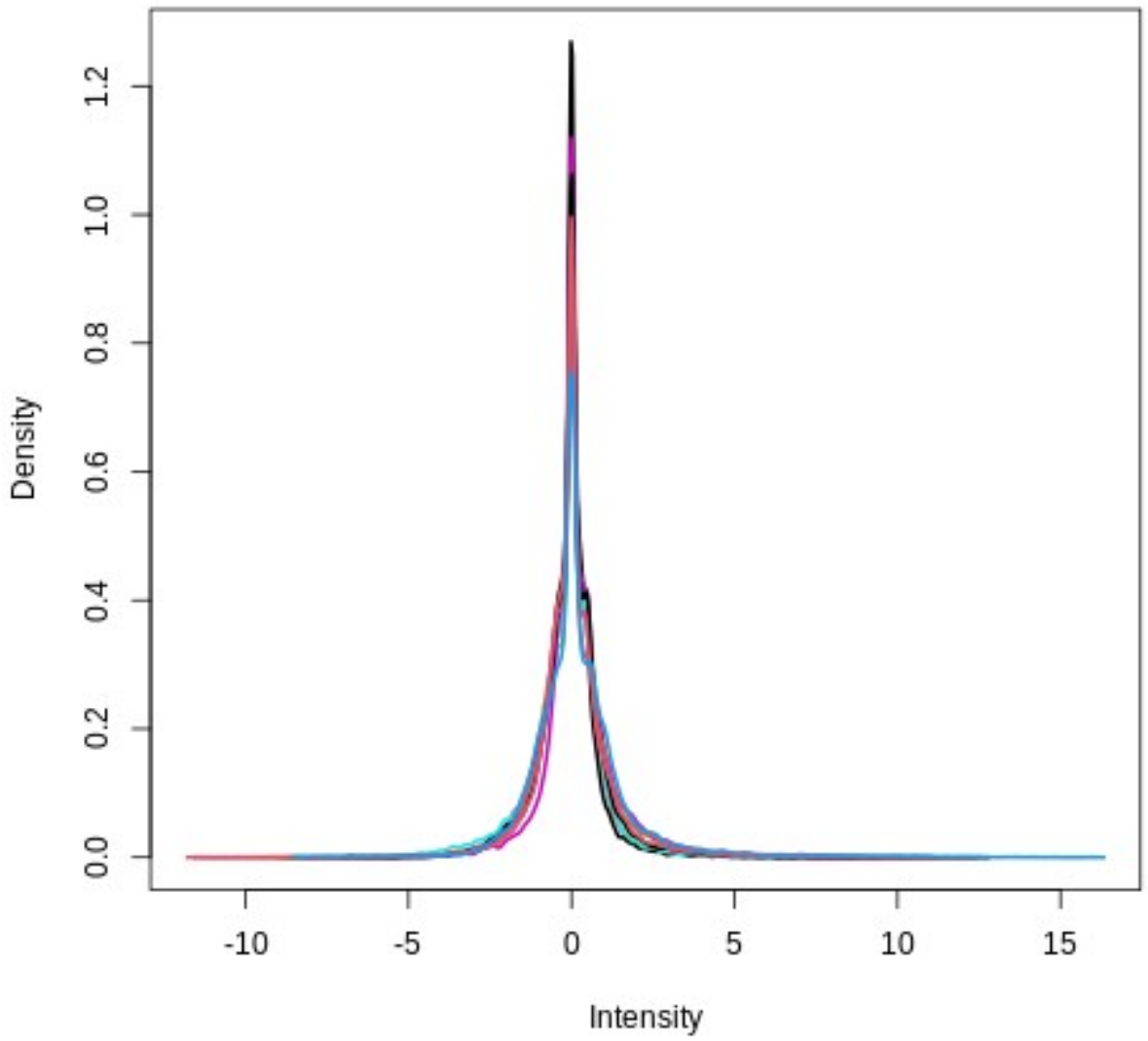
###1.2. Preprocesamiento y Normalización

Procedemos con el segundo (`log2`) y tercer (`median`) paso indicado arriba para el preprocesamiento y normalización de los datos. Después de cada paso representamos gráficamente las distribuciones de valores de expresión para ir comprobando el efecto de estas transformaciones

```
%%R
# Preprocesamiento y normalización
# 1- Aplicamos log2
log2datamasuno<-log2(data+1)  #sumamos 1 a toda la matriz para evitar log(0)

# 2- Aplicamos centrado en mediana
medianbygene<-apply(log2datamasuno, 1, median)
normdata<-log2datamasuno-medianbygene

# Observamos el efecto de esta normalización mostrando las distribuciones de valores de expresión y comparando con la gráfica anterior
boxplot(normdata[,1:50], outline=FALSE, las=2)
abline(h=median(normdata),col="blue")
```




```
%%R
# Densidad
plotDensities(normdata[,1:10], legend=FALSE)
```



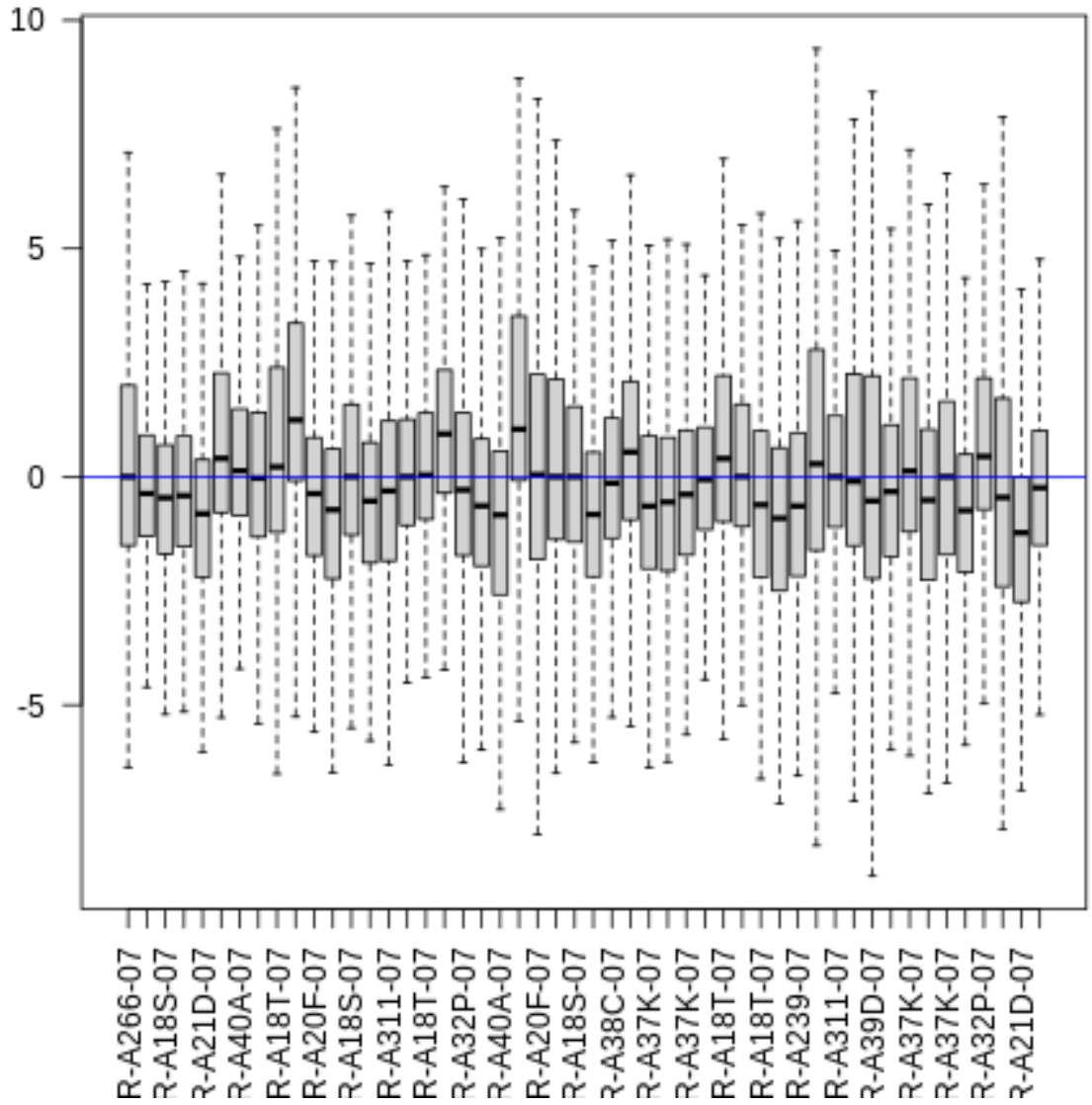
Podemos comprobar con el boxplot que, en efecto, las distribuciones se han centrado sobre 0 y son más homogéneas que las distribuciones de partida.

###1.3. Eliminar genes con poca variabilidad

Un paso habitual para simplificar el análisis es eliminar los genes que apenas se expresan en ninguna muestra (los genes que tienen bajo número de copias en las muestras, se suelen denominar *flat patterns*, genes con perfiles *planos*). Un enfoque habitual para hacer este filtrado es establecer un umbral mínimo del número de copias de un gen por millón de lecturas (CPM). Se considera que los genes no lleguen a ese valor umbral de CPM apenas se expresan y serán descartados del análisis.

Otro método muy empleado para realizar este filtrado es mantener únicamente los genes que exhiben una mayor varianza en los valores de expresión entre muestras. Esta técnica de selección permite eliminar genes *planos* y centrar el análisis en los genes con mayor potencial para discriminar unas muestras de otras.

```
%%R

# Tomar los 1500 genes expresados con mas variabilidad.
varianza<-apply(normdata, 1, var)
varianza<-sort(varianza, decreasing=TRUE) #decreasing a TRUE para
coger los 1500 genes de más varianza
milquinientosgenes<-varianza[1:1500]
genes<-names(milquinientosgenes)
milquinientosgenesdata<-normdata[genes,]

# Observamos el efecto de este filtrado mostrando las distribuciones
de valores de expresión y comparando con la gráfica anterior

boxplot(milquinientosgenesdata[,1:50], outline=FALSE, las=2)
abline(h=median(milquinientosgenesdata),col="blue")
```



###Transformaciones adicionales

Por último, realizamos transformaciones en los nombres de las muestras de la matriz de expresión para que éstos coincidan con los nombres de muestras utilizados en la tabla de información clínica descargada en la Cápsula 1. Se trata de un paso que puede carecer del interés técnico de otras celdas de código pero esta *carpintería de datos* es muy habitual para preparar los datos para su análisis computacional

```
%%R

# ID tricks to match colnames(milquinientosgenesdata) with IDs in the
mmc2 table from the Supl mat. from the paper
# remove last char from sample.info$sample so the ID matches the one
```

```
in the clinical data table from the paper
sample.info$sample <- as.factor(substr(sample.info$sample, 1,
nchar(as.vector(sample.info$sample))-1))
#given that
rownames(sample.info) == colnames(milquinientosgenesdata)
#replace colnames(milquinientosgenesdata) with sample.info$sample
colnames(milquinientosgenesdata)<-sample.info$sample
```

Guardamos la matriz de datos resultante de este proceso en un fichero de nombre `exprMatrix_prep_RSEM_log2_median_1500maxvar.tsv` (el nombre del fichero resulta de resumir las transformaciones principales aplicadas sobre los datos: matriz de expresión preprocesada por el método RSEM+log2+mediana, en la que se han seleccionado los 1500 genes de máxima varianza)

```
%%R
write.table(milquinientosgenesdata, file=
"exprMatrix_prep_RSEM_log2_median_1500maxvar.tsv", sep="\t")
```

###1.4. Pipeline alternativo de preprocesamiento y normalización (Necesario para ejecutar 2.3. Expresión diferencial)

Existen un amplio abanico de métodos de normalización y transformaciones posibles sobre datos de expresión genética. Para un análisis más detallado, se recomienda explorar los manuales de TCGA-Biolinks o la literatura especializada al respecto (consultar las referencias).

A modo ilustrativo, se presenta un *pipeline* de normalización alternativo al propuesto por los investigadores de la iniciativa TCGA-SKCM y la representación de las distribuciones de valores de expresión obtenidas. Este *pipeline* descarga los datos de expresión *crudos*, sin ningún tipo de normalización preliminar, para un subconjunto de muestras, cuantifica el número de lecturas asociadas a cada gen con STAR y aplica una normalización de tipo `gcContent`, además de filtrado de genes planos por 1er. quartil (`quantile`) y normalización TMM. Los *boxplots* resultantes muestran el efecto de la normalización sobre la distribución de valores de expresión. Se puede apreciar que en este caso las distribuciones de las distintas muestras resultan más homogéneas que con la normalización anterior.

Nota: la ejecución de este *pipeline* tomará unos minutos, ya que requiere la descarga de nuevos datos de expresión y el preprocesamiento y normalización de los mismos. Los datos procesados con este *pipeline* se utilizarán en la cápsula 2, sección 2.3 "Expresión diferencial" y cápsula 3 "Análisis de enriquecimiento de anotaciones funcionales".

```
%%R
library(TCGAbiolinks)
library(SummarizedExperiment)
library(DT)
library(NOISeq)
```

```
# Descarga de datos "crudos" (RNA counts)
# 1- Lanza la consulta para recuperar todos los datos que satisfagan
los criterios de búsqueda
subsetSamples=c("TCGA-EB-A5SF-01A-11R-A311-07","TCGA-EE-A3J8-06A-11R-
A20F-07","TCGA-EB-A430-01A-11R-A24X-07","TCGA-BF-AAP1-01A-11R-A39D-
07","TCGA-EE-A3J3-06A-11R-A20F-07","TCGA-EB-A550-01A-61R-A27Q-
07","TCGA-FR-A3YO-06A-11R-A239-07","TCGA-YD-A9TA-06A-11R-A39D-
07","TCGA-EB-A5FP-01A-11R-A27Q-07","TCGA-EB-A3XB-01A-11R-A239-
07","TCGA-WE-A8ZO-06A-11R-A37K-07","TCGA-EB-A44Q-06A-11R-A266-
07","TCGA-EB-A431-01A-11R-A266-07","TCGA-DA-A960-01A-11R-A37K-
07","TCGA-BF-A3DJ-01A-11R-A20F-07","TCGA-D3-A8GP-06A-11R-A37K-
07","TCGA-D9-A3Z1-06A-11R-A239-07","TCGA-YG-AA3N-01A-11R-A38C-
07","TCGA-EB-A44P-01A-11R-A266-07","TCGA-D3-A51N-06A-11R-A266-
07","TCGA-FW-A5DY-06A-11R-A311-07","TCGA-WE-A8ZM-06A-11R-A37K-
07","TCGA-YD-A89C-06A-11R-A37K-07","TCGA-EB-A5UM-01A-11R-A311-
07","TCGA-D3-A8GB-06A-11R-A37K-07","TCGA-EE-A3JH-06A-11R-A21D-
07","TCGA-W3-AA1Q-06A-11R-A38C-07","TCGA-D3-A51F-06A-11R-A266-
07","TCGA-D3-A3ML-06A-11R-A21D-07","TCGA-FW-A3R5-06A-11R-A239-
07","TCGA-RP-A695-06A-11R-A311-07","TCGA-FR-A7UA-06A-32R-A352-
07","TCGA-D3-A5GU-06A-11R-A27Q-07",",CGA-GF-A6C8-06A-12R-A311-
07","TCGA-BF-A5ER-01A-12R-A27Q-07","TCGA-EB-A6QZ-01A-12R-A32P-
07","TCGA-WE-A8ZQ-06A-41R-A37K-07","TCGA-BF-A5EP-01A-12R-A27Q-
07","TCGA-OD-A75X-06A-12R-A32P-07","TCGA-EB-A5SE-01A-11R-A311-
07","TCGA-EB-A553-01A-12R-A27Q-07","TCGA-D3-A51J-06A-11R-A266-
07","TCGA-RP-A694-06A-11R-A311-07","TCGA-EB-A42Y-01A-12R-A24X-
07","TCGA-D3-A51K-06A-11R-A266-07","TCGA-EB-A3HV-01A-11R-A21D-
07","TCGA-D3-A8GV-06A-11R-A37K-07","TCGA-FS-A4F2-06A-11R-A24X-
07","TCGA-EB-A44O-01A-11R-A266-07","TCGA-D9-A4Z2-01A-11R-A24X-
07","TCGA-FR-A3R1-01A-11R-A239-07","TCGA-D9-A6E9-06A-12R-A311-
07","TCGA-YD-A9TB-06A-12R-A40A-07","TCGA-FS-A1ZA-06A-11R-A18T-
07","TCGA-ER-A19A-06A-21R-A18U-07","TCGA-FS-A1ZB-06A-12R-A18S-
07","TCGA-EE-A2GD-06A-11R-A18T-07","TCGA-D3-A2J9-06A-11R-A18T-
07","TCGA-EE-A2M7-06A-11R-A18U-07","TCGA-FS-A1Z7-06A-11R-A18T-
07","TCGA-EE-A2GI-06A-11R-A18T-07","TCGA-ER-A193-06A-12R-A18S-
07","TCGA-EE-A17X-06A-11R-A18S-07","TCGA-EE-A2MI-06A-11R-A18U-
07","TCGA-DA-A1HV-06A-21R-A18S-07","TCGA-FS-A1Z0-06A-11R-A18T-
07","TCGA-EE-A2GR-06A-11R-A18S-07","TCGA-FS-A1ZY-06A-11R-A18S-
07","TCGA-D3-A2JG-06A-11R-A18T-07","TCGA-D3-A1Q1-06A-21R-A18T-
07","TCGA-EE-A2MC-06A-12R-A18S-07")
query.raw <- GDCquery(project = "TCGA-SKCM", data.category =
"Transcriptome Profiling", data.type = "Gene Expression
Quantification", workflow.type = "STAR - Counts",
barcode=subsetSamples)
#query.raw <- GDCquery(project = "TCGA-SKCM", data.category =
"Transcriptome Profiling", data.type = "Gene Expression
Quantification", workflow.type = "STAR - Counts")

# 2- Descarga los datos
GDCdownload(query.raw)
```

```r
# 3- Organiza los datos y los guarda en una variable en la memoria RAM
de tu ordenador
SKCM.counts <- GDCprepare(query = query.raw,
                          summarizedExperiment = TRUE)
rm(query.raw)

# Matriz de expresión
data2<-assay(SKCM.counts)

# Pre-procesamiento
# 1- Función TCGAanalyze_Preprocessing
# dataPrep<-TCGAanalyze_Preprocessing(object=SKCM.counts,
#                                     cor.cut = 0.6)
# 2- Función TCGAanalyze_Normalization
dataNorm<-TCGAanalyze_Normalization(tabDF=data2,
                                    geneInfo =
TCGAbiolinks::geneInfoHT,
                                   method="gcContent")
# 3- Función TCGAanalyze_Filtering
dataFilt<-TCGAanalyze_Filtering(tabDF=dataNorm,
                                  method="quantile",
                                qnt.cut = 0.25)
# 4- Método TMM
dataTMMnorm<- tmm(dataFilt)
```

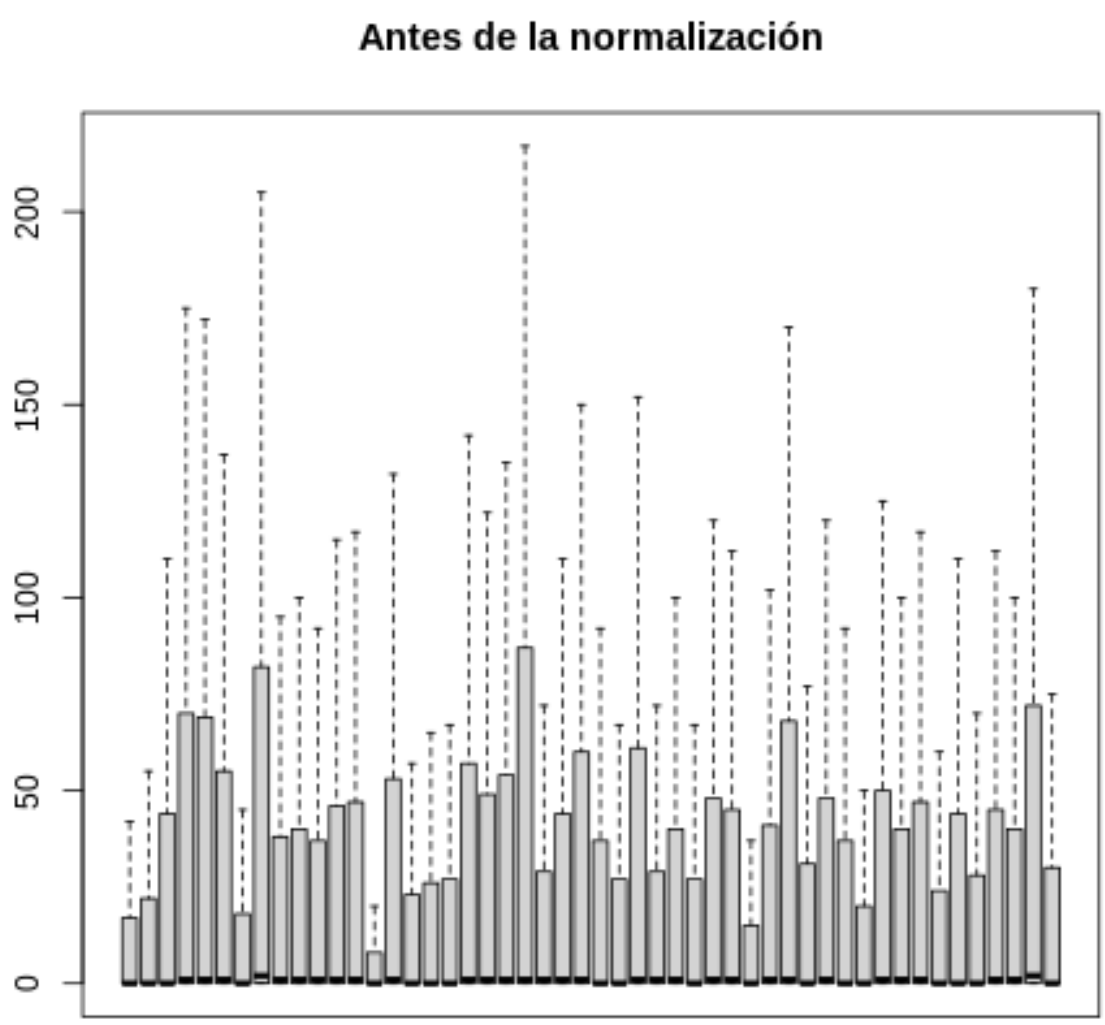
```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Loading
required package: splines

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Loading
required package: Matrix

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
Attaching package: ‘Matrix’


WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: The
following object is masked from ‘package:S4Vectors’:

    expand


WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
---------------------------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: o GDCquery:
Searching in GDC database

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
---------------------------------------
```

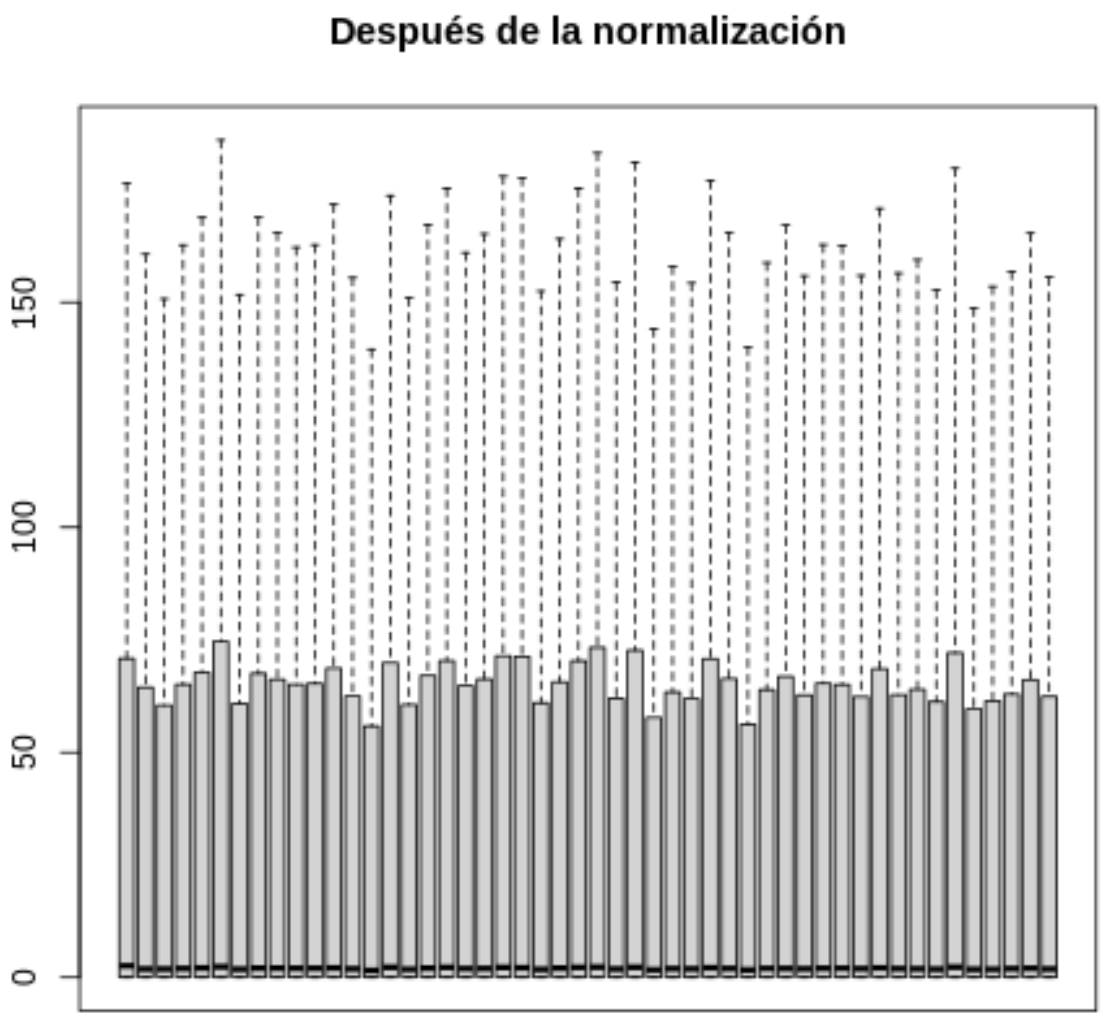
```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Genome of
reference: hg38

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
-----------------------------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: oo
Accessing GDC. This might take a while...

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
-----------------------------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: ooo
Project: TCGA-SKCM

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
--------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: oo
Filtering results

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
--------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: ooo By
data.type

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: ooo By
workflow.type

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: ooo By
barcode

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
----------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: oo Checking
data

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
----------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: ooo
Checking if there are duplicated cases

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: ooo
Checking if there are results for the query

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
-------------------
```

```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: o Preparing
output

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Downloading
data for project TCGA-SKCM

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: GDCdownload
will download 70 files. A total of 296.311254 MB

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Downloading
as: Tue_Apr_16_10_17_04_2024.tar.gz


|==================================================|100%
Completed after 17 s

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Starting to
add information to samples

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:  => Add
clinical information to samples

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:  => Adding
TCGA molecular information from marker papers

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:  =>
Information will have prefix 'paper_'

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: skcm
subtype information from:doi:10.1016/j.cell.2015.05.044

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Available
assays in SummarizedExperiment :
  => unstranded
  => stranded_first
  => stranded_second
  => tpm_unstrand
  => fpkm_unstrand
  => fpkm_uq_unstrand

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: I Need
about  53 seconds for this Complete Normalization Upper Quantile
[Processing 80k elements /s]

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Step 1 of
4: newSeqExpressionSet ...

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Step 2 of
```

```
4: withinLaneNormalization ...

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Step 3 of
4: betweenLaneNormalization ...

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Step 4 of
4: .quantileNormalization ...
```

```
%%R
# Boxplots

boxplot(data2[,1:50], outline=FALSE, main="Antes de la normalización",
xaxt="n")
boxplot(dataTMMnorm[,1:50], outline=FALSE, main="Después de la
normalización", xaxt="n")
```


Antes de la normalización
0
50
100
150
200



2. ANÁLISIS DE DATOS Y REPRESENTACIONES GRÁFICAS

Una vez filtrados y normalizados, dependiendo del tipo de información que se quiera obtener y el diseño experimental, se aplican diferentes tipos de técnicas.

En este contexto, hay numerosas técnicas estadísticas y computaciones que se aplican dependiendo del problema que se quiere abordar y las preguntas o hipótesis que se plantean. Desde el punto de vista de *machine learning* podemos hablar de:

- Técnicas de **aprendizaje no supervisado**. No se usan clases previas o predefinidas. Entre estas técnicas se incluyen métodos de análisis multivariante; clustering o

agrupamiento, reducción de dimensionalidad, extracción de reglas de asociación, etc.

- Técnicas de **aprendizaje supervisado**. Utilizan clases predefinidas sobre los datos. Dos de las técnicas más populares en este campo son la construcción de clasificadores o la detección de biomarcadores mediante la selección de variables que muestran diferencias significativas en los valores de expresión media entre distintos tipos de muestras.

Estas técnicas se verán en detalle en los módulos posteriores. A continuación, comentaremos brevemente, a modo de introducción, algunas funciones de R muy útiles para aplicar conjuntamente con estos tipos de técnicas.

###2.1. Clustering y visualizaciones con Heatmaps

Los métodos de clustering o agrupación son una de las técnicas más ampliamente usadas para el análisis de expresión génica. Estos métodos se pueden aplicar para descubrimento de grupos de genes o de muestras que muestran similitudes en sus perfiles de expresión y han tenido aplicaciones muy útiles para establecer, por ejemplo, nuevas clasificaciones de enfermedades basadas en patrones moleculares.

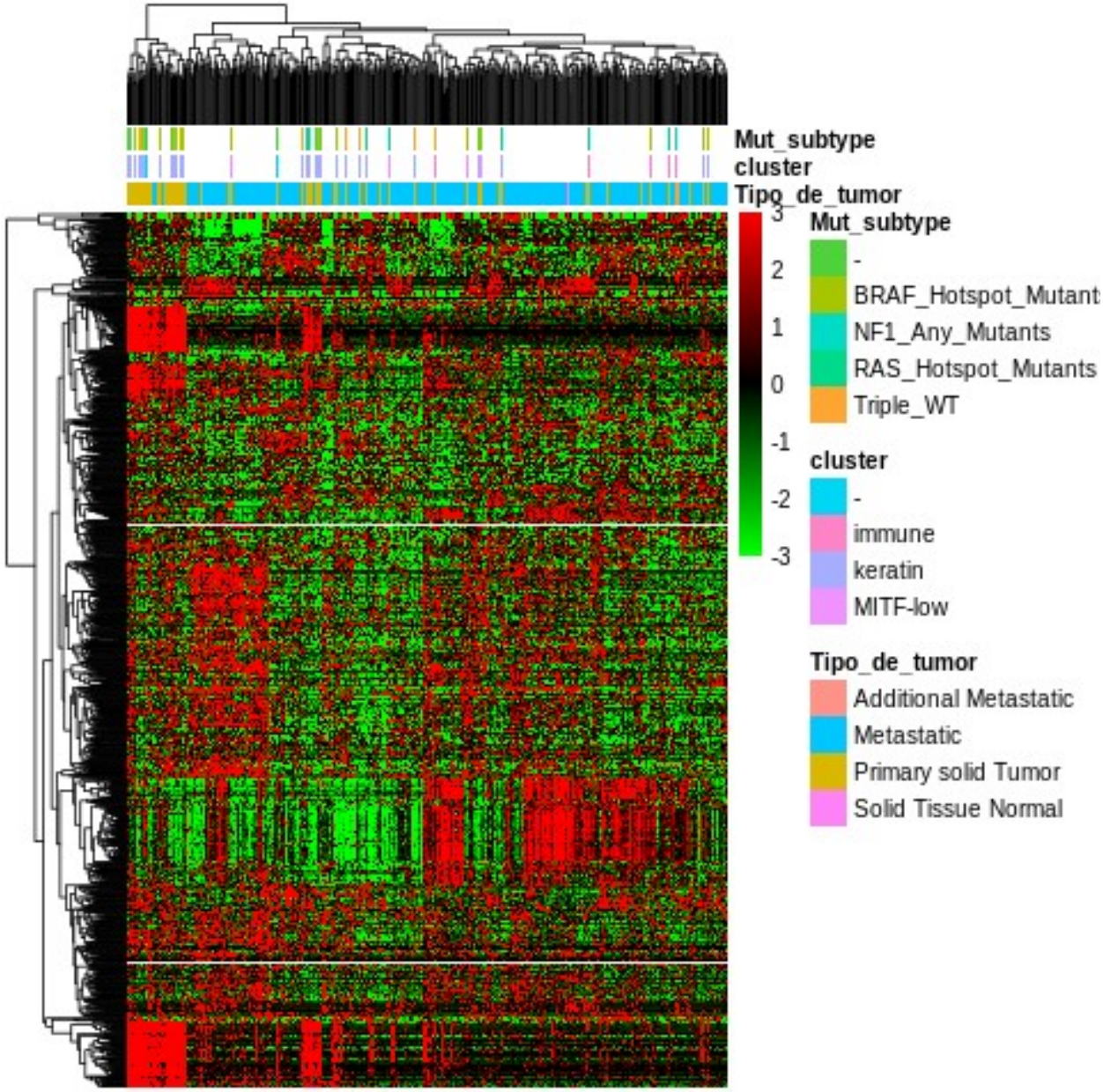
Los algoritmos de clustering jerárquico y *consensus clustering* son algunos de los más usados en este contexto. Estos métodos establecen un dendrograma que sirve para identificar grupos de elementos que muestran alta similitud entre sí, así como base para establecer una división posterior en grupos y subgrupos de distintos tamaño.

Los mapas de calor o *heatmaps* son visualizaciones frecuentemente utilizadas en combinación con las técnicas de clustering para representar visualmente la matriz de expresión ordenada por las similitudes entre elementos. En estas representaciones usualmente se reordenan las filas y columnas de la matriz de forma que objetos similares (genes en las filas y muestras en las columnas) se colocan en posiciones colindantes. De este modo, permiten apreciar de un vistazo los perfiles de expresión de genes (filas) y muestras (columnas) e identificar agrupamientos o clusters en los datos, como veremos con detalle en el Módulo 6. Cápsula 2.

R tiene funciones muy potentes para representar *heatmaps* y enriquecerlos con anotaciones que se añaden como leyendas coloreadas para las muestras o los genes. Estas visualizaciones nos permiten confrontar la estructura que se deriva del análisis de los datos (los clusters) con la información conocida para las muestras. Esta imagen ilustra este tipo de funcionalidad. En la misma, se representa un *heatmap* de la matriz de expresión de TCGA-SKCM anterior con dendrogramas para las filas (genes) y columnas (muestras) y anotaciones en color para los tipos de tumores. De nuevo, el Clustering y este tipo de representaciones se estudiarán en profundidad en la Cápsula 2 del Módulo 6.

```
%%R

definitiondata<-
data.frame( row.names=rownames(sample.info),Tipo_de_tumor=sample.info@
listData[["definition"]])
subtypedata<-data.frame(row.names=rownames(sample.info),
cluster=sample.info@listData[["subtype_RNASEQ.CLUSTER_CONSENHIER"]])
```

```
subtypemutationdata<-data.frame(row.names=rownames(sample.info),
Mut_subtype=sample.info@listData[["subtype_MUTATIONSUBTYPES"]])
anotacionesfila<-data.frame(cbind(definitiondata, subtypedata,
subtypemutationdata))

my_colors=c("green", "black", "red")
my_colors=colorRampPalette(my_colors)(100)
pheatmap(milquinientosgenesdata, color= my_colors, show_rownames = F,
show_colnames = F, breaks = seq(-3,3,length.out = 100), annotation_col
= anotacionesfila)
TCGAvisualize_Heatmap(milquinientosgenesdata)

png
  2
```



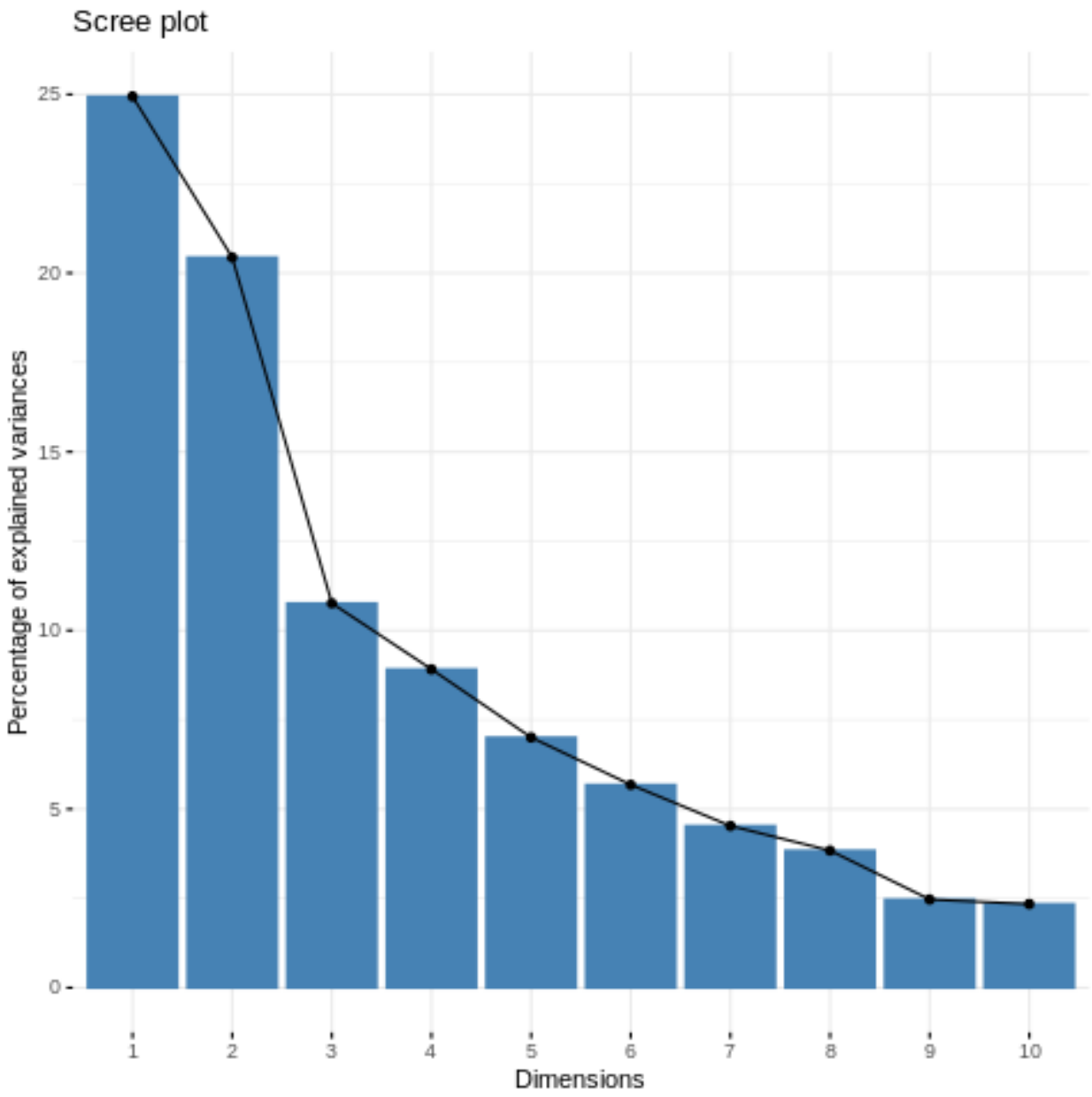
###2.2. Reducción de dimensionalidad

Como se ha indicado previamente, en este tipo de estudios se suele tener decenas de miles de variables cuantificadas en unas pocas decenas de muestras. Por tanto, es frecuente también el uso de técnicas de reducción de dimensionalidad como el análisis de componentes principales (PCA) mediante la cual se sustituyen las observaciones originales por *n* combinaciones lineales, siendo *n* mucho menor las dimensiones originales. Se seleciona *n* componentes principales de forma que representen una proporción razonable de la variación total.

La función de R `prcomp` calcula las componentes principales de una matriz dada como argumento. El siguiente código ilustra como identificar sobre la submatriz `milquinientosgenesdata[1:50,1:20]` los perfiles genéticos de máxima varianza. Primero, consideramos que cada gen es una observación y las muestras son las variables.

```
%%R
# 1- PCA aplicado sobre los genes
library(factoextra)
pca <- prcomp(milquinientosgenesdata[1:50,1:20])
fviz_eig(pca)
```

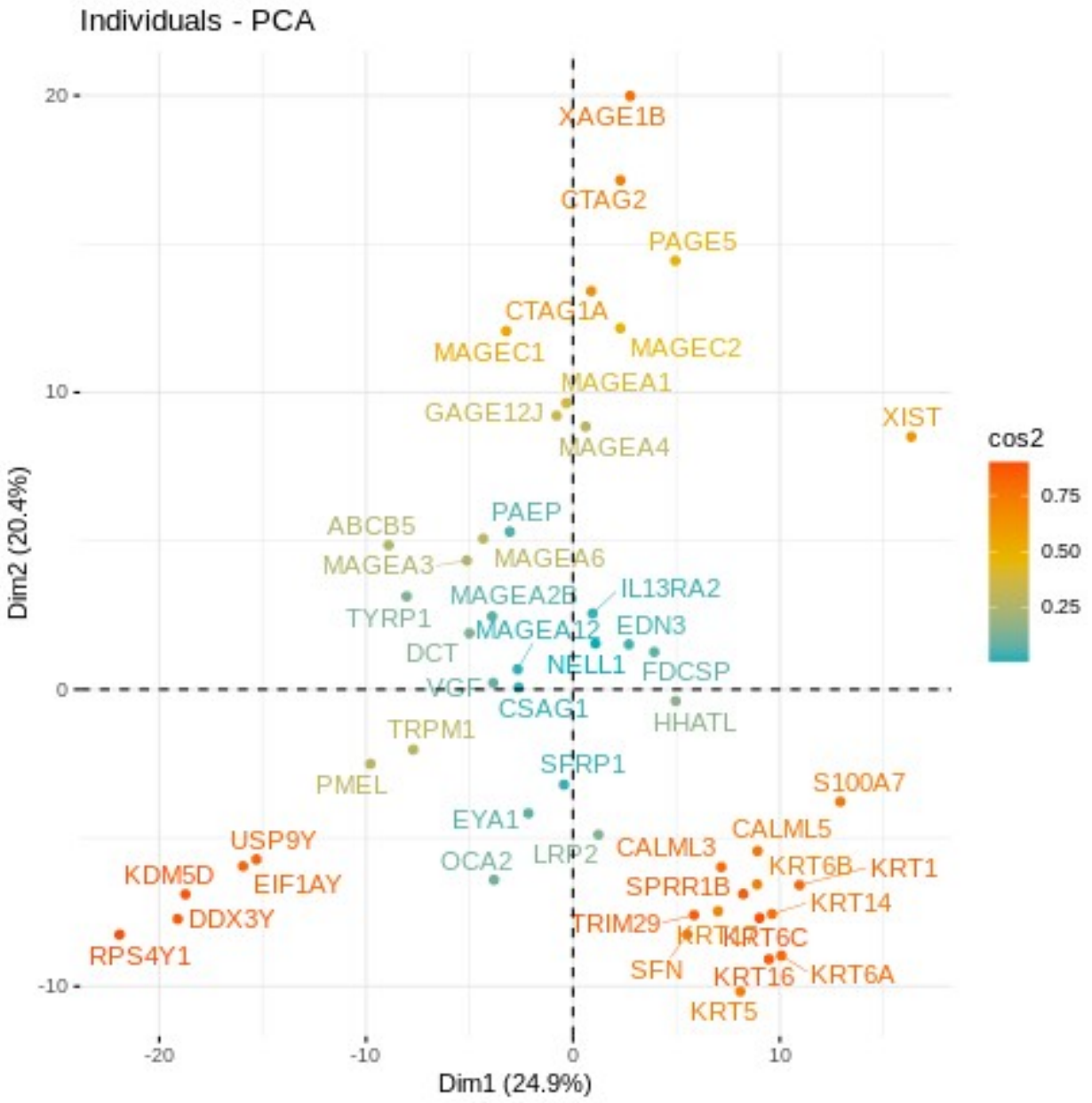
```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Loading
required package: ggplot2

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Welcome!
Want to learn more? See two factoextra-related books at
https://goo.gl/ve3WBa
```



Este gráfico representa cuánta variabilidad está explicada por cada una de las componentes principales calculadas. En este caso, comprobamos que el 45% de la varianza en los datos está explicada por las dos primeras componentes principales. El siguiente gráfico muestra cómo se representan los genes en un espacio definido por estas dos componentes principales

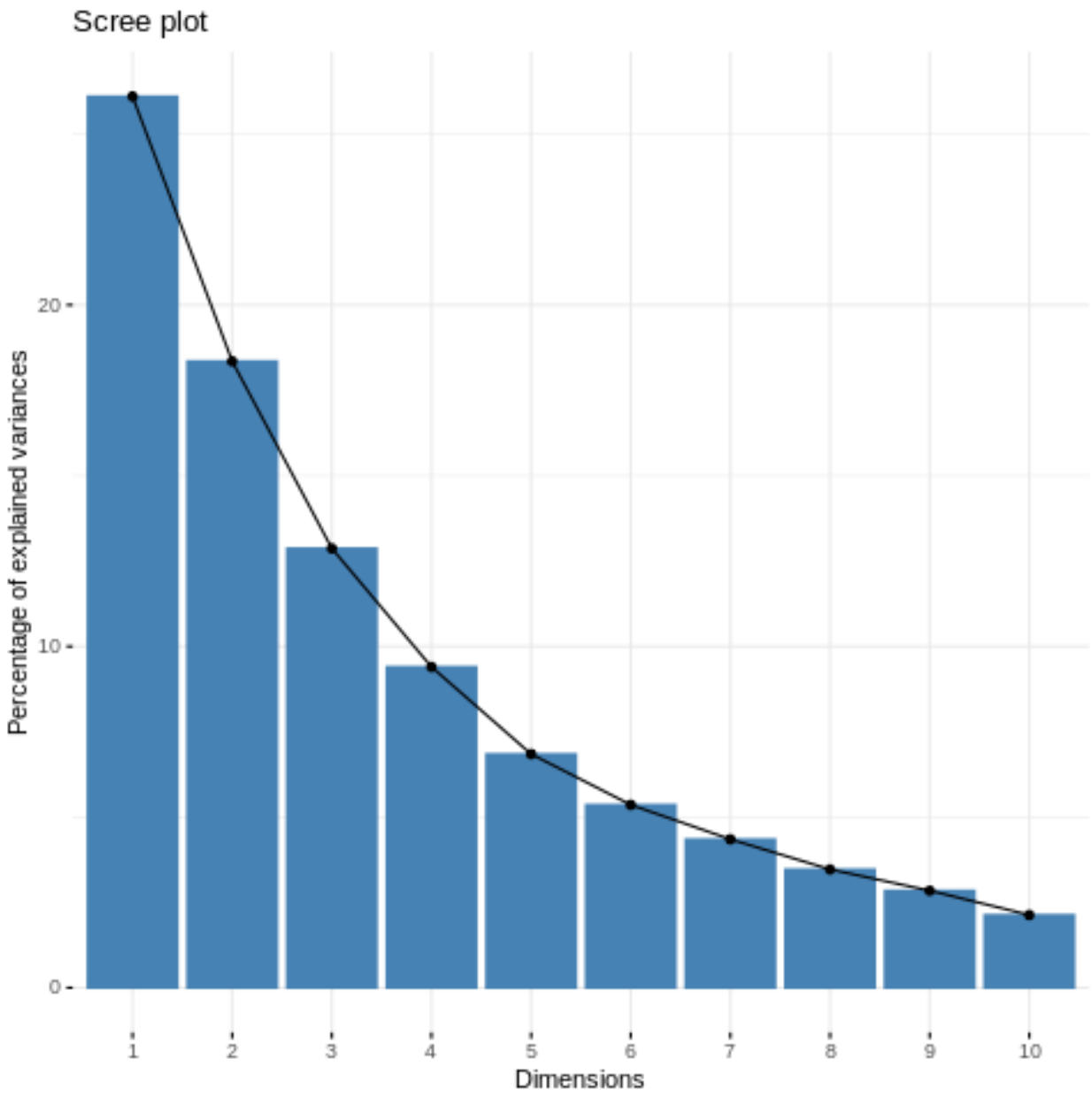
```
%%R
library(scales)
fviz_pca_ind(pca,
             col.ind = "cos2", # Color by the quality of
representation
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
```

```
                repel = TRUE     # Avoid text overlapping
)
```

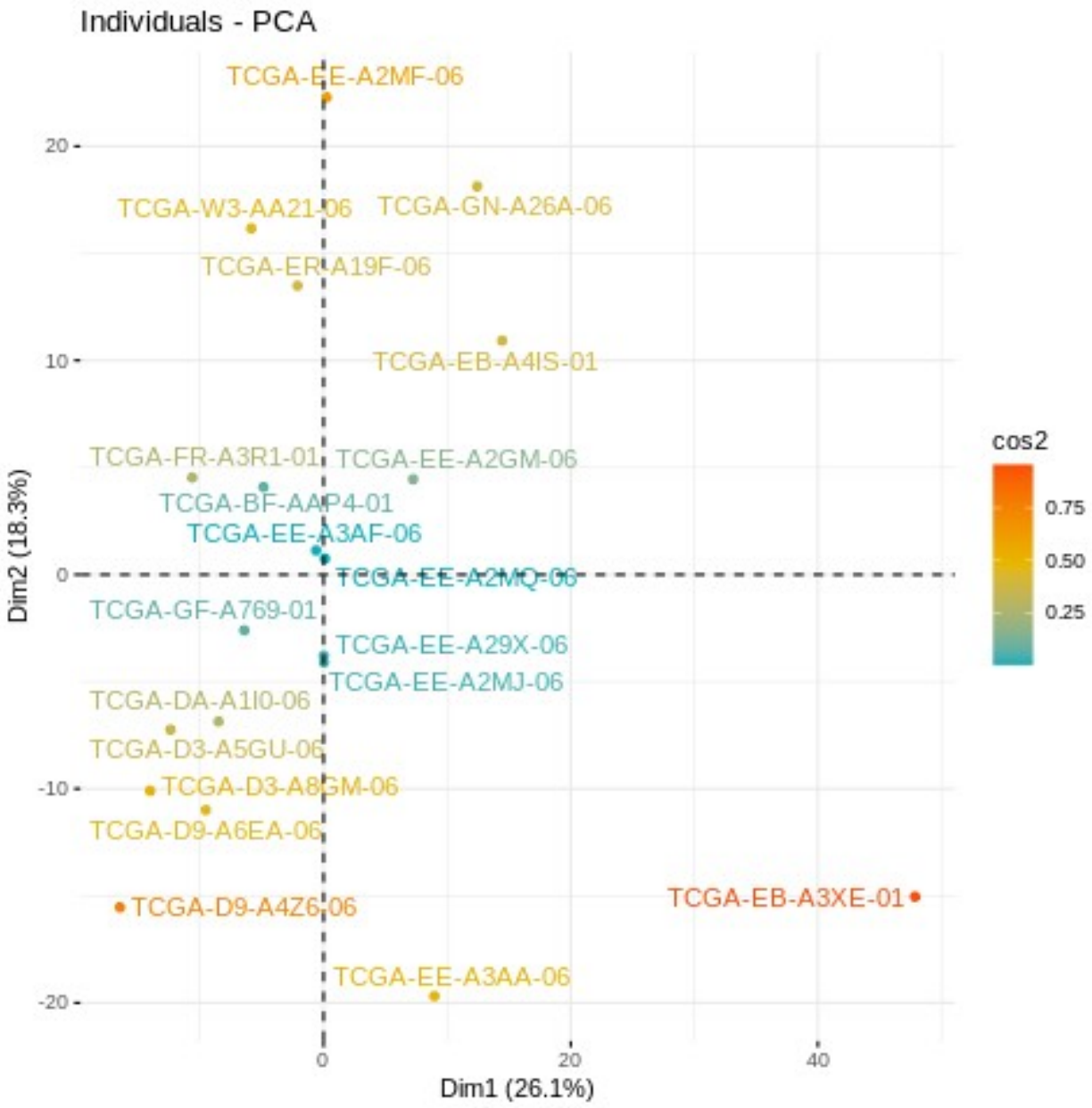


Ahora, procedemos de igual modo pero consideramos cada muestra una observación, y cada gen una variable. Para ello, trasponemos la matriz anterior con la función `t`. El resto del código no cambia:

```
%%R
# 2- PCA aplicado sobre las muestras (transponiendo la matriz original con la funcion t)
pca <- prcomp(t(milquinientosgenesdata[1:50,1:20]))
fviz_eig(pca)
```




```
%%R
fviz_pca_ind(pca,
             col.ind = "cos2", # Color by the quality of
representation
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE     # Avoid text overlapping
)
```



Estas representaciones gráficas nos permiten realizar un primer acercamiento a los datos y a su distribución en el espacio, identificando de forma preliminar muestras (genes) correladas y agrupamientos de muestras (genes) en los datos.

###2.3. Expresión diferencial

Uno de los principales objetivos de muchos estudios de transcriptómica es encontrar genes que presentan un patrón de expresión diferencial entre distintos tipos de muestras. Por ejemplo, la identificación de genes que muestran una expresión diferencial en pacientes respecto a controles sanos.

En estos estudios se evalúa si existe una diferencia significativa en la media de expresión de cada gen en dos o mas condiciones experimentales o tipos de muestras. Dado un conjunto de

muestras pertenecientes a dos grupos experimentales, *A* y *B* (por ejemplo, un grupo podría representar a los controles sanos y otro a muestras tumorales), para cada gen se lleva a cabo un contraste de hipótesis:

$H_0 : \mu_A = \mu_B$

$H_1 : \mu_A \neq \mu_B$

Aplicando el test correspondiente, se asigna a cada gen un *p-valor* que se usa para seleccionar aquellos que muestran una diferencia significativa en la expresión entre las dos condiciones.

Existe mucha bibliografía respecto a los análisis de expresión diferencial. Para introduciros en el tema, podéis consultar la referencia Costa-Silva et al., 2017 de la bibliografía.

El siguiente código ilustra cómo aplicar un análisis de expresión diferencial (DEA) entre muestras de tipo *Metástasis* y las de tipo *Tumor Sólido Primario* utilizando la función `TCGAanalyze_DEA` sobre los datos descargados anteriormente del proyecto TCGA-SKCM.

```
%%R

#Información de muestras
sample.info<-colData(SKCM.counts)
# Separar los datos en melanoma y tumor sólido primario.
TMdata<-
dataTMMnorm[,which(sample.info@listData[["definition"]]=="Metastatic")
]
PSTdata<-
dataTMMnorm[,which(sample.info@listData[["definition"]]=="Primary
solid Tumor")]

# Análisis de expresión diferencial entre metástasis y tumores sólidos
primarios.
dataDEGs <- TCGAanalyze_DEA(mat1 = TMdata,
                            mat2 = PSTdata,
                            Cond1type = "Metastatic",
                            Cond2type = "Primary solid Tumor",
                            fdr.cut = 0.01 ,
                            logFC.cut = 1,
                            method = "glmLRT")

genesexpresadosdif <- as.character(rownames(dataDEGs))
genesexpresadosdif[1:10]
```

```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Batch
correction skipped since no factors provided

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
---------------------- DEA -------------------------------

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: o 48
samples in Cond1type Metastatic
```

```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: o 22
samples in Cond2type Primary solid Tumor

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: o 35852
features as miRNA or genes

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: This may
take some minutes...

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
---------------------- END DEA -------------------------------


 [1] "ENSG00000000938" "ENSG00000001626" "ENSG00000002726"
"ENSG00000005001"
 [5] "ENSG00000005187" "ENSG00000005381" "ENSG00000006377"
"ENSG00000006432"
 [9] "ENSG00000006555" "ENSG00000006740"
```

El resultado de estos análisis es una lista de genes diferencialmente expresados, candidatos a ser estudiados en profundidad para identificar biomarcadores. Se trata de una tarea muy habitual en bioinformática. La siguiente cápsula muestra cómo abordar un análisis funcional para intentar interpretar la información de pathways or procesos biológicos que están asociados a estas listas de genes.

Referencias

1. The Cancer Genome Atlas (TCGA) https://www.cancer.gov/about-nci/organization/ccg/research/structural-genomics/tcga
2. TCGA Biolinks https://bioconductor.org/packages/release/bioc/html/TCGAbiolinks.html
3. Manual de uso TCGA-Biolinks https://www.bioconductor.org/packages/devel/bioc/vignettes/TCGAbiolinks/inst/doc/analysis.html
4. Cancer Genome Atlas Network. Genomic Classification of Cutaneous Melanoma. Cell. 2015;161(7):1681-1696. doi:10.1016/j.cell.2015.05.044 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4580370/
5. Abrams, Z.B., Johnson, T.S., Huang, K. et al. A protocol to evaluate RNA sequencing normalization methods. BMC Bioinformatics 20, 679 (2019). https://doi.org/10.1186/s12859-019-3247-x
6. Costa-Silva, J., Domingues, D., and Lopes, F.M. (2017). RNA-Seq differential expression analysis: An extended review and a software tool. PLOS ONE 12, e0190152.

Cápsula 3 - Análisis de enriquecimiento de anotaciones funcionales.

Autores:

Por **Pedro Carmona Sáez**

Profesor Titular de de la Universidad de Granada.
Departamento de Estadística e Investigación Operativa.

Por **Carlos Cano Gutiérrez**

Profesor Titular de la Universidad de Granada.
Departamento de Ciencias de Computación e Inteligencia Artificial.

INDICE

En este *notebook*:

1. Aprenderemos qué es el análisis de enriquecimiento de anotaciones funcionales en experimentos con datos -omicos.
2. Aprenderemos a utilizar los principales métodos .
3. Aprenderemos a representar los resultados de este tipo de análisis.
4. Aprendemos a manejar algunas de las principales fuentes de información de funciones biológicas y rutas metabólicas.

Contenidos:

1. ¿QUÉ ES EL ANÁLISIS DE ENRIQUECIMIENTO DE ANOTACIONES FUNCIONALES?

El resultado del análisis de datos -ómicos consiste, en la mayoría de las ocasiones, en grandes listas de genes o proteínas que están asociados a un determinado fenotipo, por ejemplo, genes diferencialmente expresados entre dos condiciones (pacientes vs. sanos), genes con patrones de metilación comunes, etc. A partir de este punto, el objetivo del estudio se centra en extraer conocimiento biológico a partir de estos genes. Este conocimiento ayudará a entender cuáles son los procesos y funciones biológicas desreguladas y servirá como punto de partida para conocer las bases moleculares de los fenotipos que se están estudiando.

Para ello, una de las aproximaciones clásicas ha consistido en evaluar si hay algún tipo de información (en forma de anotaciones) que está sobre-representada en la lista de genes respecto el resto de genes que hay en el genoma. Este tipo de análisis, denominado **análisis funcional o análisis de enriquecimiento**, se basa en anotar los genes con información disponible en diferentes bases de datos como Gene Ontology (GO: http://geneontology.org/) o Kyoto Encyclopedia of Genes and Genomes (KEGG: https://www.genome.jp/kegg/), y establecer las frecuencias de cada término en la lista de genes y el resto del genoma, de forma que se puede aplicar un test estadístico para determinar qué anotaciones funcionales están enriquecidas de forma significativa en la lista.

En este *notebook* aprenderemos a aplicar algunos de los métodos de análisis funcional más extendidos, así como algunas de las principales bases de datos de anotaciones y explorar y visualizar los resultados.

2. ANÁLISIS DE ENRIQUECIMIENTO DE ANOTACIONES INDIVIDUALES

El análisis de enriquecimiento se basa en ver si hay una sobrerrepresentación de una determinada anotación en la lista de genes de interés respecto al resto del genoma. Uno de los test estadísticos usados con más frecuencia en este contexto es el basado en la distribución hipergeométrica, donde para cada anotación se puede calcular la probabilidad de encontrar un determinado número de genes asociados a la misma:

$$P(X=i)=\frac{\binom{M}{i}\binom{N-M}{N-i}}{\binom{N}{n}}$$

Donde N es el número total de genes en el genoma, M es el número que presentan una determinada anotación, n es el número de genes en la lista e i el número de genes en la lista con la anotación.

Se calcula el p-valor, pero hay que tener en cuenta que cuando se analizan numerosos genes el p-valor se tiene que corregir para tener en cuenta el problema de las comparaciones múltiples, para lo que se suele aplicar corrección de Bonferroni o False Discovery Rate (FDR) entre las más frecuentes.

###2.1. Ejemplo 1. Enriquecimiento de términos de Gene Ontology

Gene Ontology (GO) es un recurso ampliamente utilizado que ha establecido una ontología genética que categoriza en anotaciones el conocimiento científico actual sobre las funciones de los genes de muchos organismos diferentes, desde humanos hasta bacterias. Es una de las principales fuentes de información para análisis funcional y ha sido citado en decenas de miles de publicaciones.

El proyecto se inició en 1998 cuando los investigadores que estudiaban el genoma de tres organismos modelo: *Drosophila melanogaster* (mosca de la fruta), *Mus musculus* (ratón) y *Saccharomyces cerevisiae* (levadura) aceptaron trabajar colaborativamente en un esquema de clasificación común para la función de genes.

GO ofrece dos recursos principales:

- La ontología en sí. Es decir, el vocabulario de términos y la relación entre ellos para los diferentes tipos de funciones biológicas (Función Molecular), las vías que llevan a cabo diferentes programas biológicos (Proceso Biológico) y lugares donde ocurren estos (Componente Celular).
- El corpus de las anotaciones GO para los diferentes genes de un gran número de organismos.

```
%%R
library(clusterProfiler)
library(org.Hs.eg.db)

# Asumimos que existe una lista de IDs de genes de interés (resultado de alguna técnica de análisis, podría cargarse desde un fichero)
gene<-
c("4312","8318","10874","55143","55388","991","6280","2305","9493","1062","3868","4605","9833","9133","6279","10403","8685","597","7153","23397","6278","79733","259266","1381","3627","27074","6241","55165","9787","7368","11065","55355","9582","220134","55872","51203","3669","83461","22974","10460","10563","4751","6373","8140","79019","820","10635","1844","4283","27299","55839","27338","890","9415","983","54821","10232","4085","6362","9837","5080","7850","81930","5918","81620","332","55765","79605","3832","6286","5163","2146","3002","50852","7272","2568","64151","51806","366","2842")

#Usaremos como referencia unos datos precargados en el paquete DOSE
data(geneList, package="DOSE")

# Ejecutamos análisis de enriquecimiento de términos de Gene Ontology para el listado anterior de genes para la ontología Componente Celular (CC)
GO <- enrichGO(gene = gene, universe = names(geneList), OrgDb = org.Hs.eg.db,   ont= "CC", pAdjustMethod = "BH", pvalueCutoff = 0.01, qvalueCutoff = 0.05, readable = TRUE)

head(GO)
```

```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Error: disk I/O error
```

```
Error: disk I/O error
```

###2.2. Ejemplo 2. Enriquecimiento de rutas de KEGG

KEGG (Kyoto Encyclopedia of Genes and Genomes) es un poyecto que se inició en 1995 por el programa del genoma humano japonés y se ha convertido en un recurso muy utilizado para análisis de rutas enzimáticas y redes de interacciones moleculares específicas por organismo. KEGG proporciona no sólo una representación gráfica de estas redes y cómo los genes y proteínas se interconectan, sino también la anotación de en qué vías y rutas está implicado cada gen.

```
%%R

#Utilizamos la misma lista de genes "gene" definida en la celda
```

```
anterior para hacer un enriquecimiento de rutas con KEGG
enrichKEGG <- enrichKEGG( gene = gene,
                          organism = 'hsa',
                          pvalueCutoff = 0.05)
head(enrichKEGG )
```

```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Reading
KEGG annotation online: "https://rest.kegg.jp/link/hsa/pathway"...

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Reading
KEGG annotation online: "https://rest.kegg.jp/list/pathway/hsa"...
```

```
                                    category
hsa04110                  Cellular Processes
hsa04657                  Organismal Systems
hsa04218                  Cellular Processes
hsa04061 Environmental Information Processing
hsa04114                  Cellular Processes
hsa04914                  Organismal Systems
                                 subcategory       ID
hsa04110               Cell growth and death hsa04110
hsa04657                      Immune system hsa04657
hsa04218               Cell growth and death hsa04218
hsa04061 Signaling molecules and interaction hsa04061
hsa04114               Cell growth and death hsa04114
hsa04914                    Endocrine system hsa04914
                                                                Description
hsa04110                                                         Cell cycle
hsa04657                                          IL-17 signaling pathway
hsa04218                                             Cellular senescence
hsa04061 Viral protein interaction with cytokine and cytokine receptor
hsa04114                                                    Oocyte meiosis
hsa04914                          Progesterone-mediated oocyte maturation
         GeneRatio  BgRatio       pvalue     p.adjust       qvalue
hsa04110      9/48 157/8753 1.408932e-07 1.676629e-05 1.527579e-05
hsa04657      5/48  94/8753 1.525771e-04 6.074746e-03 5.534709e-03
hsa04218      6/48 156/8753 1.924598e-04 6.074746e-03 5.534709e-03
hsa04061      5/48 100/8753 2.041932e-04 6.074746e-03 5.534709e-03
hsa04114      5/48 131/8753 7.109876e-04 1.692150e-02 1.541720e-02
hsa04914      4/48 102/8753 2.282502e-03 4.526961e-02 4.124520e-02
                                              geneID Count
hsa04110 8318/991/9133/10403/890/983/4085/81620/7272     9
hsa04657                    4312/6280/6279/6278/3627     5
hsa04218                2305/4605/9133/890/983/51806     6
hsa04061                   3627/10563/6373/4283/6362     5
hsa04114                     991/9133/983/4085/51806     5
hsa04914                          9133/890/983/4085     4
```

Las pathways se pueden visualizar con los genes asociados a las mismas, usando el comando

browseKEGG(enrichKEGG, 'hsa04110')

Ver por ejmplo www.kegg.jp/kegg-bin/show_pathway?
hsa04110/8318/991/9133/890/983/4085/7272

3. ANÁLISIS DE ENRIQUECIMIENTO DE CO-ANOTACIONES (MODULAR ENRICHMENT)

Otro tipo de análisis tiene en cuenta la relación que hay entre diferentes términos y el hecho de que un mismo gen puede estar anotado con diferentes fuentes. Encontrar relaciones entre anotaciones basadas en patrones de co-ocurrencia puede ampliar nuestra comprensión de los eventos biológicos asociados con un sistema experimental dado. Por ejemplo, un conjunto de genes expresados diferencialmente puede estar asociado con la activación de procesos biológicos que están restringidos a ciertos orgánulos celulares. La recuperación de tales asociaciones proporciona información significativa y adicional para la interpretación de los resultados experimentales. En este ejemplo usaremos la aplicación GENECODIS (https://genecodis.genyo.es/), que es capaz de integrar diferentes fuentes de información y extraer conjuntos de anotaciones que co- ocurren en un número mínimo de genes mediante una técnica de minería de datos denominada extracción de reglas asociativas que se verá en detalle en los siguientes módulos de este curso.

Puedes ejecutar un ejemplo copiando una lista de genes y haciendo un análisis en la aplicación.

4. GENE SET ENRICHMENT ANALYSIS (GSEA)

Gene Set Enrichment Analysis (Subramanian et al., 2005) es un algoritmo desarrollado para paliar algunas limitaciones de los análisis de enriquecimiento que se aplican tras seleccionar una lista de genes diferencialmente expresados. Estas limitaciones pueden ser:

- En algunas ocasiones, los análisis estadísticos de expresión diferencial no ofrecen genes que superen los umbrales de significancia estadística.
- En ocasiones, pueden existir genes que aunque no pasen los umbrales queden con valores muy cercanos y pueden aportar información muy útil para la interpretación funcional.

GSEA se basa en ordenar toda la lista de genes en base a la expresión diferencial entre dos condiciones sin aplicar un umbral para seleccionar un listado y evaluar la distribución de los genes asociados a una determinada anotación a lo largo de toda la lista y calcular un *Enrichment Score*. Los detalles sobre la metodología se pueden encontrar en la publicación original, pero de forma resumida:

- Dada una matriz *NxM* se ordenan los *N* genes en base a una medida de asociación con el fenotipo $r\left(g_j\right)=r\ j$ obteniendo la lista ordenada *L: {g1, . . . , gN}* y un valor *p*.
- Dado un conjunto independiente de N_H genes *S*, por ejemplo genes que están anotados con la misma función, se evalua la fracción de genes en *S* (*T*) ponderada por su correlación y la fracción de genes que no están en *S* (*F*) que hay hasta una posición dada *i* en *L*.

$$P_T(S,i) = \sum_{g_j \in S; j \le i} ¿ \frac{|r_j)^p}{N_R},$$

donde $N_R = \sum_{g_j \in S} |r_j)^p$

$$P_F(S,i) = \sum_{g_j \notin S; j \le i} ¿ \frac{1}{(N - N_H)}$$

El *enrichment score* (ES) es la máxima desviación de cero de $P_T - P_F$. Si S se distribuye aleatoriamente, ES(S) tendrá un valor pequeño, pero si se concentra en la parte superior o inferior de la lista, ES(S) tendrá un valor alto.

Se calcula la significancia calculando el ES obtenido al permutar las clases originales, repitiendo este proceso un gran número de veces y comparando el ES observado con los obtenidos en las permutaciones y corrigiendo finalmente los *p-valores* obtenidos por las comparaciones múltiples.

###4.1. Ejemplo 3. GSEA de rutas de KEGG

```
%%R
kegg <- gseKEGG(geneList = geneList, organism = 'hsa',
     nPerm      = 1000,
     minGSSize  = 120,
     pvalueCutoff = 0.05,
     verbose    = FALSE)
head(kegg)
```

```
               ID                                 Description setSize
hsa04110 hsa04110                                  Cell cycle     139
hsa05169 hsa05169            Epstein-Barr virus infection     193
hsa04613 hsa04613 Neutrophil extracellular trap formation     130
hsa05166 hsa05166 Human T-cell leukemia virus 1 infection     202
hsa04218 hsa04218                       Cellular senescence     141
hsa04510 hsa04510                            Focal adhesion     191
         enrichmentScore       NES      pvalue p.adjust     qvalue
rank
hsa04110       0.6637551  2.805914 0.002958580    0.018 0.01169591
1155
hsa05169       0.4335010  1.934008 0.003154574    0.018 0.01169591
2820
hsa04613       0.4496569  1.895076 0.002890173    0.018 0.01169591
2575
hsa05166       0.3893613  1.755545 0.003144654    0.018 0.01169591
1955
hsa04218       0.4115945  1.749876 0.002890173    0.018 0.01169591
1155
hsa04510      -0.4199193 -1.726190 0.001453488    0.018 0.01169591
2183
```

```
                             leading_edge
hsa04110  tags=36%, list=9%, signal=33%
hsa05169 tags=39%, list=23%, signal=31%
hsa04613 tags=37%, list=21%, signal=30%
hsa05166 tags=26%, list=16%, signal=22%
hsa04218  tags=17%, list=9%, signal=16%
hsa04510 tags=27%, list=17%, signal=23%

core_enrichment
hsa04110
8318/991/9133/10403/890/983/4085/81620/7272/9212/1111/9319/891/4174/92
32/4171/993/990/5347/701/9700/898/23594/4998/9134/4175/4173/10926/6502
/994/699/4609/5111/26271/1869/1029/8317/4176/2810/3066/1871/1031/9088/
995/1019/4172/5885/11200/7027/1875
hsa05169
3627/890/6890/9636/898/9134/6502/6772/3126/3112/4609/917/5709/1869/365
4/919/915/4067/4938/864/4940/5713/5336/11047/3066/54205/1871/578/1019/
637/916/3383/4939/10213/23586/4793/5603/7979/7128/6891/930/5714/3452/6
850/5702/4794/7124/3569/7097/5708/2208/8772/3119/5704/7186/5971/3135/1
380/958/5610/4792/10018/8819/3134/10379/9641/1147/5718/6300/3109/811/5
606/2923/3108/5707/1432
hsa04613
820/366/51311/64581/3015/85236/55506/8970/8357/1535/2359/5336/4688/928
15/3066/8336/292/1991/3689/8345/5603/4689/5880/10105/1184/6404/3018/68
50/5604/3014/7097/1378/8290/1536/834/5605/1183/728/2215/8335/5594/9734
/3674/5578/5582/7417/8331/6300
hsa05166
991/9133/890/4085/7850/1111/9232/8061/701/9700/898/4316/9134/3932/3559
/3126/3112/4609/3561/917/1869/1029/915/114/2005/5902/55697/1871/1031/2
224/292/1019/3689/916/3383/11200/706/3600/6513/3601/468/5604/7124/1030
/3569/4049/4055/10393/3119/5901/5971/1959/3135
hsa04218
2305/4605/9133/890/983/51806/1111/891/993/3576/1978/898/9134/4609/1869
/1029/22808/1871/5499/91860/292/1019/11200/1875
hsa04510
5595/5228/7424/1499/4636/83660/2013/7059/5295/1288/23396/3910/3371/308
2/1291/394/3791/7450/596/3685/1280/3675/595/3912/1793/2012/1278/1277/1
293/10398/55742/2317/7058/25759/56034/3693/3480/5159/857/1292/3908/390
9/63923/3913/1287/3679/7060/3479/10451/80310/1311/1101
```

5. VISUALIZACIONES

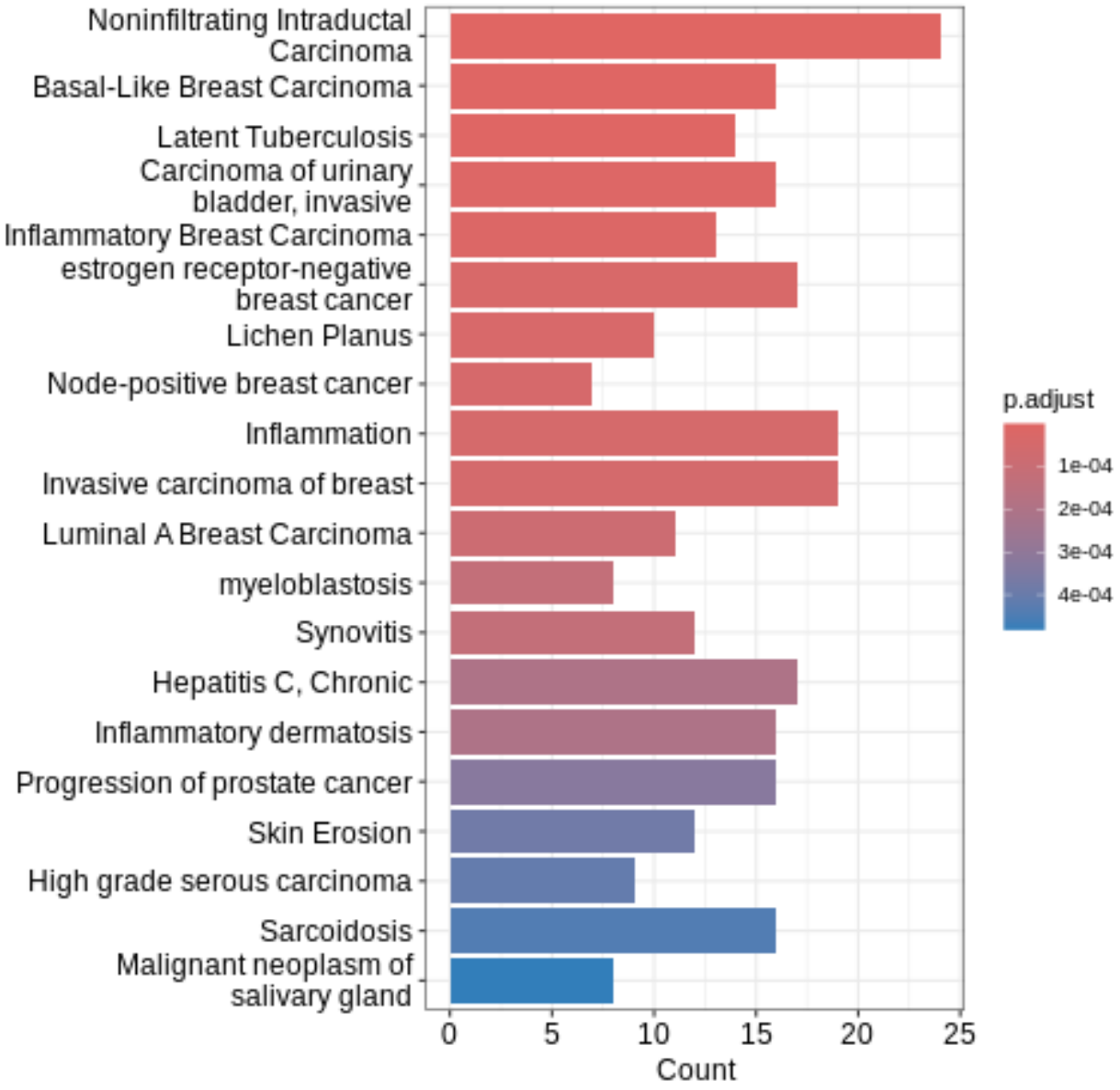
Hay varias opciones para llevar a cabo la visualización de los resultados de enriquecimiento. Algunas de ellas son las siguientes:

###5.1. Bar plot

Es el tipo de visualización más frecuente, donde se representan los términos enriquecidos y la frecuencia o pvalor de cada uno.

```
%%R
# Usaremos el paquete DOSE y unos ejemplos para ilustrar estos gráficos
library (DOSE)
data(geneList)
deGenes <- names(geneList)[abs(geneList) > 2]

edo <- enrichDGN(deGenes)
library(enrichplot)
barplot(edo, showCategory=20)
```

```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: DOSE v3.28.2  For help: https://yulab-smu.top/biomedical-knowledge-mining-book/

If you use DOSE in published research, please cite:
Guangchuang Yu, Li-Gen Wang, Guang-Rong Yan, Qing-Yu He. DOSE: an R/Bioconductor package for Disease Ontology Semantic and Enrichment analysis. Bioinformatics 2015, 31(4):608-609
```

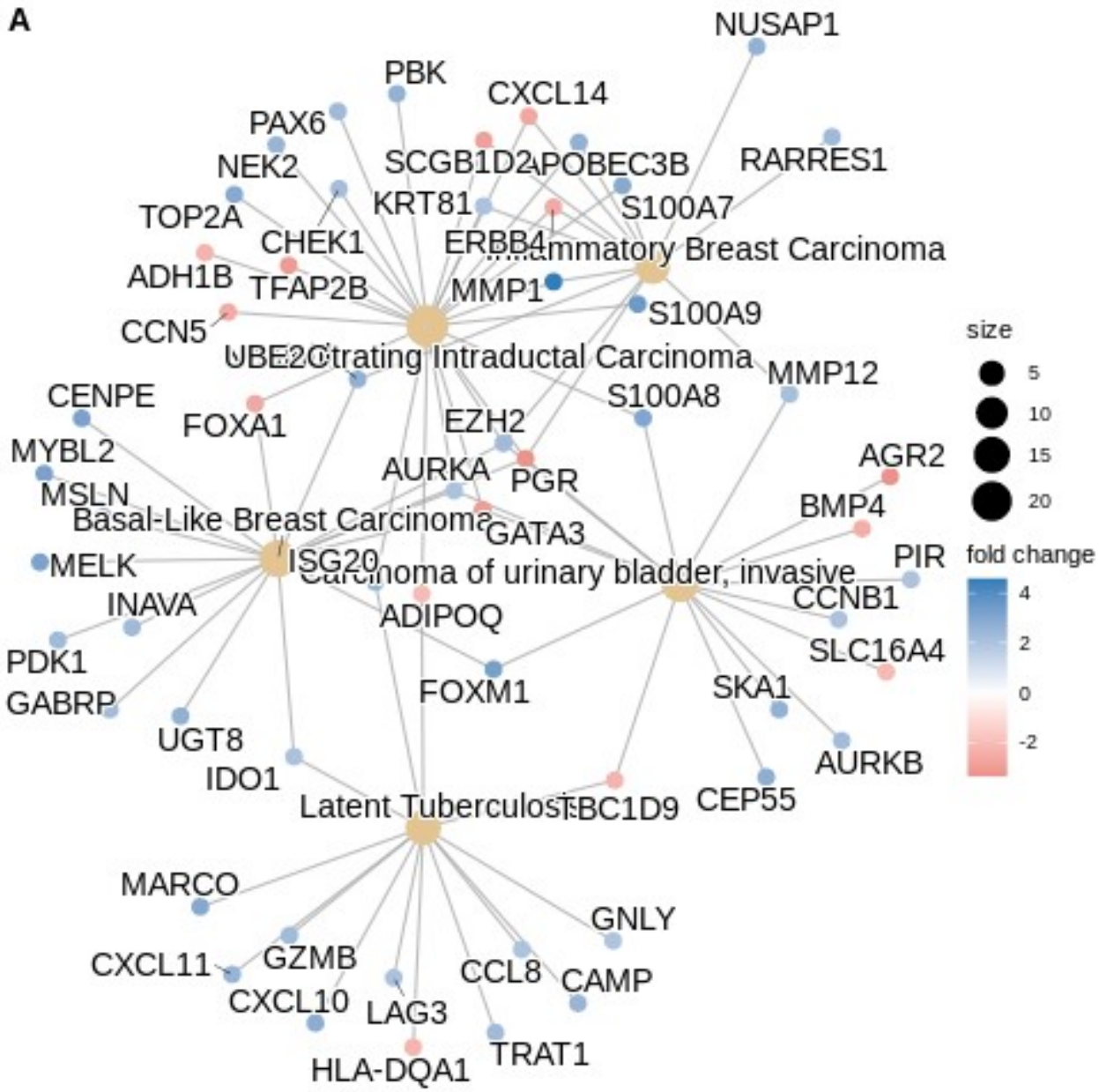


###5.2. Network de genes y términos funcionales

Los bar plots sólo muestran los términos enriquecidos, pero puede ser útil tener información de anotaciones funcionales y genes asociados a las mismas.

```
%%R

## se convierten  gene ID en gene Symbol
edox <- setReadable(edo, 'org.Hs.eg.db', 'ENTREZID')
p1 <- cnetplot(edox, foldChange=geneList)

cowplot::plot_grid(p1, ncol=1, labels=LETTERS[1], rel_widths=c(1.2))
```

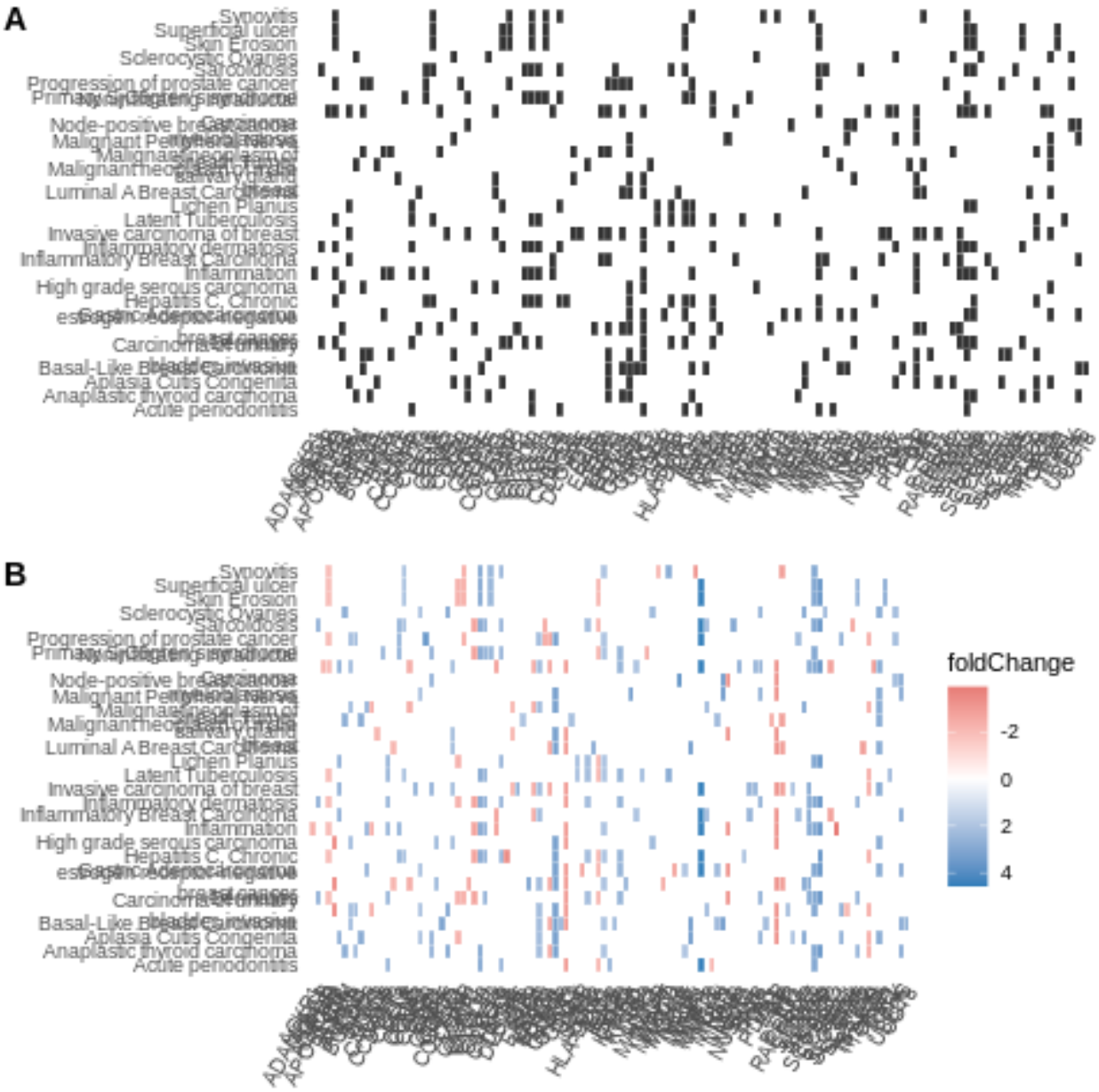
```
Scale for size is already present.
Adding another scale for size, which will replace the existing scale.
```



###5.3. Mapas de calor

Estas representaciones, muy útiles para representar datos experimentales (ej. expresión génica) también se puede utiliza para representar los términos y genes. Al igual que las redes, dan información de relación entre genes y términos. Cuando hay muchos términos y genes las redes llegan a ser muy complejas y las visualizaciones no son adecuadas, por lo que la representación en heatmap puede ser más apropiada

```
%%R
p1 <- heatplot(edox)
p2 <- heatplot(edox, foldChange=geneList)
cowplot::plot_grid(p1, p2, ncol=1, labels=LETTERS[1:2])
```



REFERENCIAS

- The Gene Ontology Consortium (2019). The Gene Ontology Resource: 20 years and still GOing strong. Nucleic Acids Research, Volume 47
- Huang, D.W., Sherman, B.T., and Lempicki, R.A. (2009). Bioinformatics enrichment tools: paths toward the comprehensive functional analysis of large gene lists. Nucleic Acids Res. 37, 1–13.

- Subramanian, A., Tamayo, P., Mootha, V.K., Mukherjee, S., Ebert, B.L., Gillette, M.A., Paulovich, A., Pomeroy, S.L., Golub, T.R., Lander, E.S., et al. (2005). Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. Proc. Natl. Acad. Sci. U. S. A. 102, 15545–15550.