

Hands-on! Introducción al análisis de datos con R 8 y 9 de febrero de 2024

Beatriz Fernández Blanco
Maria Guaita Céspedes

Contents

1. ¿Por qué R?	1
1.1 Un poco de historia...	1
1.2 Ventajas de usar R	2
2. Instalación de R y RStudio	3
3. Exploramos RStudio ¡y escribimos las primeras líneas de código!	7
3.1. R como calculadora	7
3.2. Objetos en el entorno de R	9
3.3. Clases de objetos	12
3.4. Comentarios en el código	13
4. Análisis de un conjunto de datos	13
4.1 Importar datos de Excel (.xlsx)	13
4.2 Importar datos de un archivo tabulado (.tsv)	15
5. Cómo seguir aprendiendo por tu cuenta	44

1. ¿Por qué R?

1.1 Un poco de historia...

Conocer cómo nació R es interesante para comprender sus características. R es un lenguaje de programación creado por Ross Ihaka y Robert Gentleman en los años 90, que eran estadísticos

en la Universidad de Auckland (Nueva Zelanda) y querían crear un material mejor para dar el curso de introducción a la estadística a sus alumnos. Así, crearon el lenguaje R, basado en el lenguaje S. R siempre ha sido un proyecto de uso libre desde junio de 1995. El hecho de que fuera creado por estadísticos y no por ingenieros para el desarrollo de software como pasa con otros lenguajes de programación como C y Java, explica que uno de sus puntos fuertes sea la **interactividad** y la **visualización** de los datos.



Figure 1: Logo de R y sus creadores

El mantenimiento y desarrollo de R es realizado por el **R Development Core Team**, un equipo de especialistas en ciencias computacionales y estadística provenientes de diferentes instituciones y lugares alrededor del mundo. Este equipo mantiene la versión **base** de R que, como su nombre indica, es sobre la cual se crean otras implementaciones de R así como los paquetes que expanden su funcionalidad.

Referencias:

- <https://bookdown.org/jboscomendoza/r-principiantes4/un-poco-de-historia.html>
- <http://rafalab.dfci.harvard.edu/dsbook/getting-started.html#fn1>
- Ihaka, R., & Gentleman, R. (1996). R: a language for data analysis and graphics. *Journal of computational and graphical statistics*, 5(3), 299-314.

1.2 Ventajas de usar R

En esta sesión vamos a conocer R como una herramienta para visualizar y analizar datos. Probablemente a estas alturas hayáis empleado algunos programas para el análisis de datos, como puede ser Excel para hacer gráficos y SPSS o Graphpad para la estadística. Veamos algunas ventajas que nos proporciona R:

- Es de uso libre (forma parte del sistema GNU)
- Se puede usar en diferentes sistemas operativos: Windows, Linux, Mac.

- Hay una gran comunidad de usuarios de R activa y en continuo crecimiento y, por lo tanto, hay muchos recursos para aprender y hacer preguntas.
- Ofrece muchas utilidades en el análisis de datos, en especial en cuanto a interactividad y visualización.
- En general, conocer un lenguaje de programación os facilitará el análisis de datos, sea cuál sea vuestro ámbito de especialización. Además, mejora otras habilidades transversales, como es el pensamiento lógico. Y... el perfil bioinformático/bioestadístico está muy valorado.



Figure 2: Lo mejor de los dos mundos

2. Instalación de R y RStudio

Todo el código y documentación de R está almacenado en **CRAN** (Comprehensive R Archive Network), que es una red de servidores alrededor del mundo. Es decir, CRAN es el sitio oficial a través del cual descargaremos R. En esta sesión no vamos a instalar R, sino que lo usaremos a través de otra herramienta, pero si queréis descargarlo en casa estos son los pasos que tenéis que seguir.

Una vez instalado R podemos abrir la consola para ver qué apariencia tiene. Aquí ya podríamos trabajar.

Sin embargo, vamos a aprender a usar R dentro de RStudio, que es una interfaz que nos da muchísimas más funcionalidades que trabajar solo con la consola. La descarga de RStudio se realiza desde Posit.

cran.r-project.org

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux \(Debian, Fedora/Redhat, Ubuntu\)](#)
- [Download R for macOS](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

R for Windows

Subdirectories:

- [base](#)
- [contrib](#)
- [old.contrib](#)
- [Rtools](#)

Please do not download the source code for Windows.

You may also want to download:

- [Rtools](#)

Note: CRAN does not provide a Windows binary for Rtools.

R-4.2.2 for Windows

[Download R-4.2.2 for Windows](#) (76 megabytes, 64 bit)

[README on the Windows binary distribution](#)

[New features in this version](#)

This build requires UCRT, which is part of Windows since Windows 10 and Windows Server 2016. On older systems, UCRT has to be installed manually from [here](#).

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the [md5sum](#) of the .exe to the [fingerprint](#) on the master server.

Frequently asked questions

- [Does R run under my version of Windows?](#)
- [How do I update packages in my previous version of R?](#)

Please see the [R FAQ](#) for general information about R and the [R Windows FAQ](#) for Windows-specific information.

Other builds

R-4.2.2-win.exe

• Patches to this release are incorporated in the [r-patched snapshot build](#)

• A build of the development version (which will eventually become the next major release of R) is available in the [r-devel snapshot build](#)

• [Previous releases](#)

Note to webmasters: A stable link which will redirect to the current Windows binary release is [~CRAN/MIRROR~bin/windows/base/release.html](#)

Last change: 2022-10-31

Figure 3: Instalación de R

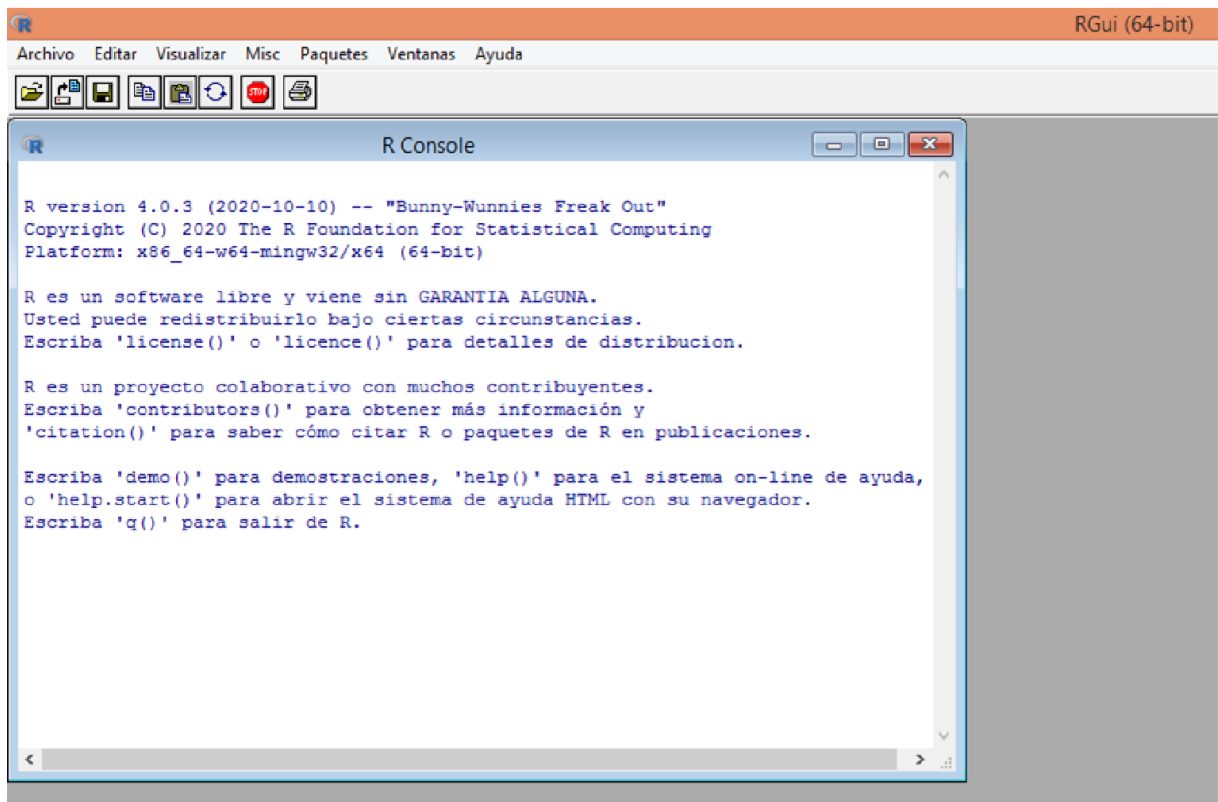


Figure 4: La consola de R


posit.co/download/rstudio-desktop/			
 PRODUCTS SOLUTIONS LEARN & SUPPORT EXPLORE MORE			
OS	Download	Size	SHA-256
Windows 10/11	RSTUDIO-2022.12.0-353.EXE	202.77 MB	FD8EA4B4
macOS 11+	RSTUDIO-2022.12.0-353.DMG	365.71 MB	FD4BEBB5
Ubuntu 18+/Debian 10+	RSTUDIO-2022.12.0-353-AMD64.DEB	131.20 MB	23CAE58F
Ubuntu 22	RSTUDIO-2022.12.0-353-AMD64.DEB	131.95 MB	8BC3F84D
Fedora 19/Red Hat 7	RSTUDIO-2022.12.0-353-X86_64.RPM	145.99 MB	A717CDAD
OpenSUSE 15	RSTUDIO-2022.12.0-353-X86_64.RPM	131.50 MB	983E7D0C

Figure 5: Instalación de RStudio

3. Exploramos RStudio ¡y escribimos las primeras líneas de código!

En esta sesión trabajaremos en RStudio en la nube a través de Posit Cloud. Únicamente necesitamos registrarnos con una cuenta de correo electrónico. Si bien es práctico para los objetivos de esta sesión, tened en cuenta que Posit Cloud tiene unos recursos limitados en la versión libre, de los cuales destaca que solo permite 25 horas de uso al mes.

Una vez nos hemos registrado podemos abrir un proyecto y empezar a explorar RStudio. Cuando abrimos RStudio vemos varios paneles. A la izquierda tenemos la consola. La consola es el espacio donde R ejecuta las órdenes que le damos. Este símbolo de “mayor que” se llama *prompt*, y significa que R está listo para que le demos una orden. Vamos a crear nuestro primer script. Un **script** es un bloque de código. Pinchamos en File > New File > R Script. Vemos que podemos crear otro tipo de documentos de R, pero no vamos a explicarlos. Vemos que se abre en el panel superior. A la derecha tenemos otros dos paneles con varias pestañas. A lo largo de la sesión veremos qué información nos aportan.

En esta sesión vamos a trabajar con datos reales, pero antes necesitamos conocer cómo escribir en R. Vamos a ver algunas líneas de código sencillas para irnos familiarizando. El uso más sencillo de R es usarlo como calculadora. Empecemos realizando operaciones sencillas:

3.1. R como calculadora

Operadores matemáticos

```
3+2
```

```
## [1] 5
```

```
3-2
```

```
## [1] 1
```

```
3*2
```

```
## [1] 6
```

```
3/2 # División con decimales
```

```
## [1] 1.5
```

```
3%/%2 # División entera
```

```
## [1] 1
```

```
3**2
```

```
## [1] 9
```

```
3^2
```

```
## [1] 9
```

Para ejecutar un comando pinchamos en “Run” o pulsamos la combinación de teclas CTRL+INTRO. Si nos encontramos en la consola es suficiente con INTRO. Observad que lo que ejecutamos en el fichero .R se visualiza en la consola: tanto la instrucción como el resultado, pero ¡atención! lo que escribimos en la consola no se queda guardado.

Operadores lógicos

Los operadores lógicos nos sirven para hacerle preguntas a R y que nos responda con verdadero o falso.

```
# IMPORTANTE ESCRIBIR DOS VECES "="  
1==2 ## 1 es igual a 2?
```

```
## [1] FALSE
```

```
1!=2 ## 1 es diferente de 2?
```

```
## [1] TRUE
```

```
1<0 ## 1 es menor que 0?
```

```
## [1] FALSE
```

```
1>0 ## 1 es mayor que 0?
```

```
## [1] TRUE
```


3.2. Objetos en el entorno de R

Existen varios tipos de objetos para almacenar la información.

Variables

Las variables son sencillamente objetos en los que almacenamos información, que puede ser de distinto tipo (números, caracteres,...). La asignación de valores a una variable se puede hacer con la combinación “<-”, como si fuera una flecha, o con el signo “=”. Por convenio se prefiere la primera. En R no importa si ponemos espacio entre la variable y su valor, cosa que puede ser distinta en otros lenguajes. Por ejemplo, podemos asignar el valor 2 a la variable a y el valor 3 a la variable b. Probad a ejecutar este código y veréis cómo quedan asignados los valores a las variables.

```
a=2
b<-3
b < -3 # OJO!
```

```
## [1] FALSE
```

```
c <- "Hola"
e <- "Adios"
```

Además fijaos en el panel superior derecho: las variables han pasado a formar parte del entorno, que es el conjunto de variables que hemos ido definiendo.

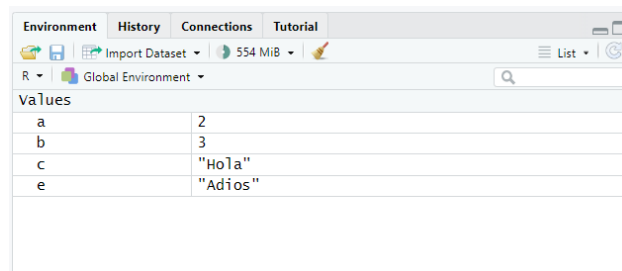


Figure 6: “El entorno de R en RStudio”

Las variables se pueden sobrescribir reasignándoles otro valor.

```
e<-3
```

Podemos nombrar a las variables como queramos pero es muy importante saber que:

- **NUNCA** debemos nombrar una variable con un nombre de una variable que ya exista en R. Por ejemplo, no sería apropiado crear la variable `sum<-2+3`, porque `sum()` es una función que existe en R y podemos crear conflictos. Ante la duda podemos ejecutar el comando `help()` para consultar si existe una variable, (`help(sum)`); si no nos devuelve nada entonces es que esa variable no está predefinida en R.
- Las variables **no contienen espacios** porque si no R las interpretará como dos objetos distintos.
- Por convención se escriben en **minúscula** y donde querríamos poner un espacio pondremos un guión bajo (o un punto). Estas convenciones pueden ser distintas en otros lenguajes de programación.
- Intentad que los nombres de vuestras variables sean **representativos** de su significado, tanto por facilitaros vuestro trabajo como por si lo lee otra persona. Puede que creéis muchas variables y necesitaréis saber qué es cada una de ellas.

Las variables que hemos definido hasta ahora solo contienen un valor, pero hay diferentes tipos de variables según el valor o valores que les asignemos. Además de números y caracteres pueden contener otro tipo de objetos.

Vectores

Los vectores son objetos en los que podemos almacenar más de un valor, aunque estos valores han de ser siempre del mismo tipo. Por ejemplo, no puedo almacenar números y caracteres juntos. Los vectores se crean con la función `c()`, que significa concatenar, separando los elementos con una coma.

```
numeros <- c(1,2,3,4,5)
saludos<-c("Hola","Buenos días", "Buenas tardes")
```

Para acceder a los elementos de un vector empleamos los corchetes. Los elementos en R empiezan a contarse desde el 1 (otros lenguajes empiezan en 0). Así accedemos al segundo elemento:

```
saludos[2]
```

```
## [1] "Buenos días"
```

Este índice que R reconoce lo podemos usar para ordenar los datos. Vamos a decirle que queremos ver el vector “saludos” pero que nos muestre primero el tercer elemento, luego el primero y finalmente el segundo.

```
saludos[c(3,1,2)]
```

```
## [1] "Buenas tardes" "Hola"          "Buenos días"
```

Conjuntos de datos

Los conjuntos de datos son tablas con un número de filas y de columnas. Se crean con la función `data.frame()`, indicando el nombre de cada variable (cada columna) y los valores que contiene (es decir, las observaciones para cada una).

```
meses_dias<-data.frame(meses=c("enero","febrero","marzo","abril","mayo","junio",  
                               "julio","agosto","septiembre","octubre",  
                               "noviembre","diciembre"),  
                      dias=c(31,28,31,30,31,30,31,31,30,31,30,31))
```

Funciones

Las funciones las podemos imaginar como máquinas a las que les damos unos datos de entrada y nos devuelven unos datos de salida. R base tiene algunas ya definidas. ¡¡La función `help()` la más útil de todas!! Otras forman parte de paquetes con otras funcionalidades.

```
log(10)
```

```
## [1] 2.302585
```

```
sum(a,b)
```

```
## [1] 5
```

```
help(sum)
```

```
## starting httpd help server ... done
```

```
sum(meses_dias$dias) # Sumatorio
```

```
## [1] 365
```

```
mean(meses_dias$dias) # Media
```

```
## [1] 30.41667
```

```
median(meses_dias$dias) # Mediana
```

```
## [1] 31
```

```
sd(meses_dias$dias) # Desviación estandar
```

```
## [1] 0.9003366
```

También podemos crear nosotros nuestras propias funciones. Simplemente para ilustrar, tenemos la función `media.dos()`, que es una función muy sencilla para calcular la media de dos números. Mirad que nos devuelve lo mismo que la función de R `mean()`.

```
media.dos<-function(x,y){  
  (x+y)/2  
}  
media.dos(3,5)
```

```
## [1] 4
```

```
mean(c(3,5))
```

```
## [1] 4
```

3.3. Clases de objetos

Cualquier objeto que definamos pertenece a una clase. Podemos conocer la clase a la que pertenece un objeto con la función `class()`.

```
c="4"  
class(c)
```

```
## [1] "character"
```

Imaginad que realizo la siguiente operación:

```
# d<-a+c # No permite sumar n° y letras  
# d<-c+e # No permite sumar caracteres
```

¿Por qué me da error? Esto es porque estoy intentando aplicar una acción a un objeto al que no se le puede aplicar: la variable `c` es de tipo `character`, no `numérica`. Quizás ahora no le veáis la utilidad, pero hay clases de objetos más complicados y ocasiones en que tras muchas líneas de código ya no se recuerda de qué tipo son algunas variables.

3.4. Comentarios en el código

Además de las variables, véis que hay unas líneas que empiezan por `#`. Esto son comentarios y son líneas que R no ejecuta. Son muy útiles para irnos poniendo notas de lo que hace nuestro código. El código se puede comentar con la combinación de teclas `Ctrl + MAYUS + C`.

En dos horas que dura este taller es imposible hacer un recopilatorio de todas las operaciones básicas en R, estos son solo algunos ejemplos. Tampoco hay nadie que se sepa todo lo que se puede hacer en R. Lo que sí nos parece importante es que conozcáis el gran libro de R, que no es otra cosa que Google. Lanzad una pregunta de algo que queréis hacer en R pero no sabéis cómo. Por ejemplo yo quiero obtener los valores del 1 al 100 con un intervalo de 2 pero no sé cómo hacerlo. Vamos a buscar “how to create range of values with interval in r”. Este es un ejemplo muy sencillo que casi cualquier libro introductorio de R va a cubrir. Pero hay ocasiones en que la cosa se complica más y veréis que las respuestas que da la gente son de lo más variadas. Normalmente no hay una única forma de hacer las cosas, por eso no hay código que esté bien o mal (siempre y cuando haga lo que nosotros queremos).

4. Análisis de un conjunto de datos

Ahora que sabemos un poco de cómo funciona R vamos a seguir profundizando en los conjuntos de datos, que creemos que es lo que más podéis usar en esta etapa (TFG, TFM, ect.). Para ello vamos a usar datos reales. Los cursos de R suelen hacer estas explicaciones con conjuntos de datos que están en librerías de R que podéis instalar, pero preferimos daros algo que os podáis imaginar que es vuestro.

4.1 Importar datos de Excel (.xlsx)

Las hojas de cálculo como Excel son muy prácticas para guardar datos y seguramente así lo hagáis. Para leer datos de R en Excel usamos el paquete `readxl`. Si no está instalado hay que instalarlo y luego cargar el paquete al entorno.

```
# install.packages("readxl") # Descomentar esta línea para instalarlo
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 4.3.1
```

Os proporcionamos unos datos de absorbancia de un ensayo BCA para medir concentración de proteínas. Tenemos las funciones `readxl()` y `readxlsx()`. En este caso usamos la función `readxls()` porque nuestro archivo es `.xlsx`.

```
bca<-read_xlsx("BCA_assay.xlsx",sheet=1,skip = 1)
str(bca)
```

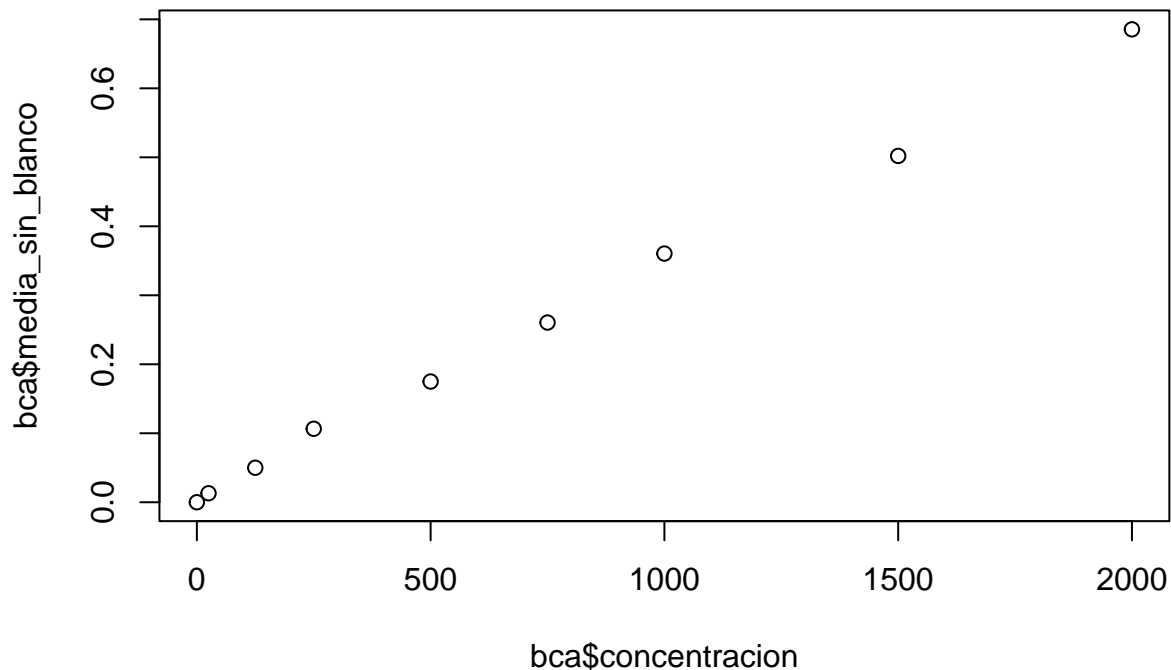
```
## tibble [11 x 4] (S3: tbl_df/tbl/data.frame)
## $ nombre      : chr [1:11] "A" "B" "C" "D" ...
## $ medida1     : num [1:11] 0.824 0.639 0.477 0.372 0.272 0.215 0.162 0.12 0.114 0.2
## $ medida2     : num [1:11] 0.771 0.589 0.468 0.373 0.302 0.222 0.162 0.13 0.11 0.24
## $ concentracion: chr [1:11] "2000" "1500" "1000" "750" ...
```

Ahora vamos a ver un bloque de código de cómo construir la recta patrón y calcular la concentración de proteínas de cada muestra.

```
# 1. Calculamos la media de absorbancia de cada muestra
bca$media<-rowMeans(bca[,c(2,3)])
# 2. Restamos a cada media la señal del blanco
bca$media_sin_blanco<-bca$media-bca$media[bca$nombre=="I"]
# 3. Necesitamos que la concentración R la lea como un número
bca$concentracion<-as.numeric(bca$concentracion)
```

```
## Warning: NAs introducidos por coerción
```

```
# 4. Construimos la recta patrón con una regresión lineal (y= a +bx)
plot(bca$concentracion,bca$media_sin_blanco)
```



```
linear.model<-lm(media_sin_blanco ~ concentracion, data=bca[1:9,])
a<-linear.model$coefficients[1]
b<-linear.model$coefficients[2]
# 5. Calculamos la concentración de cada muestra
bca$concentracion[c(10,11)]<-(bca$media_sin_blanco[c(10,11)]-a)/b
```

4.2 Importar datos de un archivo tabulado (.tsv)

El conjunto de datos con el que vamos a trabajar ahora lo hemos obtenido a través del portal **cBioPortal**. cBioPortal contiene datos de genómica en cáncer y su creación y mantenimiento es del Memorial Sloan Kettering Cancer Center. El conjunto de datos está disponible como “TMB and immunotherapy (MSK,Nat Genet 2019)” (https://www.cbioportal.org/study/summary?id=tmb_mskcc_2018). Podéis descargarlo desde el repositorio de github o en cBioPortal en la pestaña “Clinical Data” seleccionando todas las variables y descargando los datos en formato “.tsv”.

Ahora subimos los datos a Posit Cloud e importamos el documento como “tmb”. En R base podemos hacerlo con las funciones `read.delim()` o `read.table()`. La función `read.delim()` suele ser más lista que nosotros y detectar el tipo de separador que delimita las columnas.

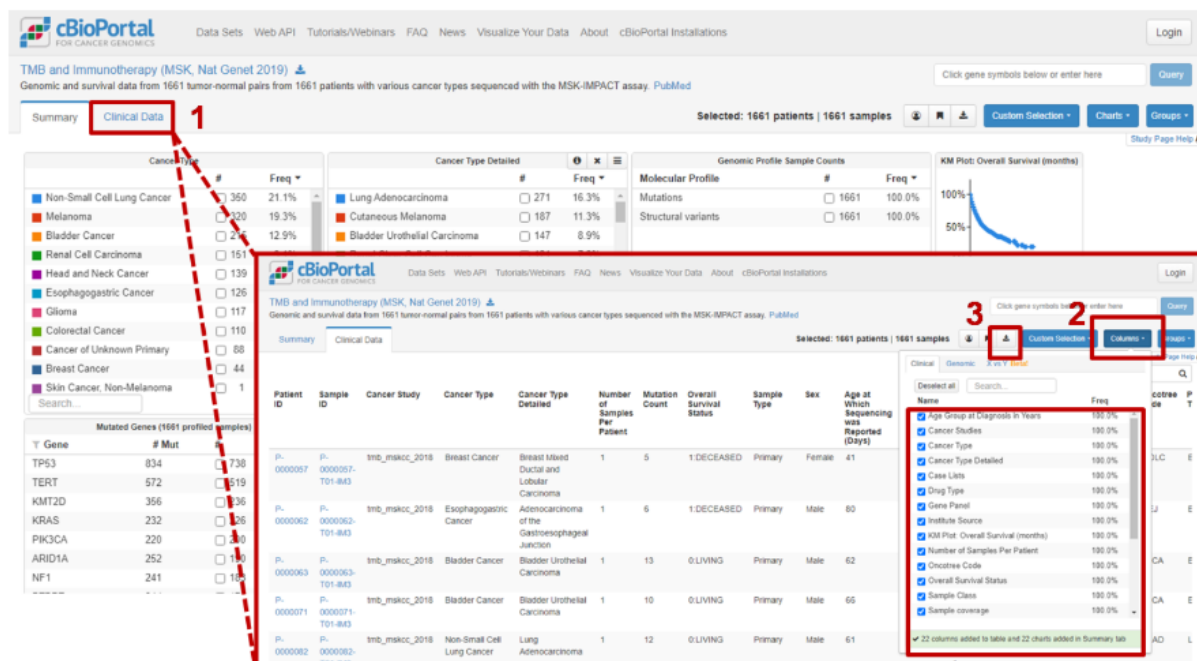


Figure 7: “Descarga de los datos en cBioPortal”

Dos opciones igual de válidas:

```
tmb<-read.delim("tmb_mskcc_2018_clinical_data.tsv")
```

Con read.table obligatorio poner el separador

```
tmb<-read.table("tmb_mskcc_2018_clinical_data.tsv",header=TRUE,sep="\t")
```

Aunque aquí solo vamos a ver los ficheros .xlsx y .tsv también podemos encontrarnos ficheros .csv (delimitados por comas) y .txt (formato plano). Tenéis que tener esto en cuenta para leer cada archivo correctamente. Además hay paquetes fuera de R base que también sirven para leer archivos, como **readr** y **openslx**.

Este es un conjunto de datos genómicos y de supervivencia dentro del proyecto Mutation Profiling of Actionable Cancer Targets (MSK-IMPACT) del Memorial Sloan Kettering Cancer Center (MSKCC). En él se recogen datos pacientes de cáncer en estadios avanzados a los que se les ha realizado una secuenciación de exoma con el objetivo de relacionar la carga mutacional con la respuesta a inhibidores de puntos de control inmunitarios (ICI). Los ICI son anticuerpos contra determinadas dianas, como CTLA4 y PD-1/PDL.1 y supusieron una revolución en el tratamiento de pacientes en estadios avanzados. Sin embargo, la respuesta a largo plazo solo es efectiva en algunos pacientes, por lo que se necesitan biomarcadores para predecir qué pacientes se pueden beneficiar de ellos. En concreto, este estudio tenía por objetivo estudiar en una cohorte amplia de pacientes si la carga mutacional estaba relacionada con la respuesta a estos tratamientos, algo que se había observado en cohortes más pequeñas.

4.2.1 Análisis descriptivo

En cualquier análisis de datos, antes de hacer cualquier análisis estadístico debemos hacer un análisis descriptivo, una exploración de nuestros datos. Vamos a comenzar explorando el conjunto de datos y también vamos a ir analizando algunas variables para ir incorporando nuevos conceptos. Podemos comprobar en primer lugar que R ha reconocido este objeto como `data.frame`. También podemos ver la estructura del mismo.

```
class(tmb) # vemos que R lo reconoce como data.frame
```

```
## [1] "data.frame"
```

```
str(tmb) # estructura del conjunto de datos
```

```
## 'data.frame':    1661 obs. of  24 variables:
## $ Study.ID          : chr  "tmb_mskcc_2018" "tmb_mskcc_2018"
## $ Patient.ID        : chr  "P-0000057" "P-0000062" "P-0000062"
## $ Sample.ID         : chr  "P-0000057-T01-IM3" "P-0000062-T01-IM3"
## $ Age.at.Which.Sequencing.was.Reported..Days.: int  41 80 62 66 61 63 47 44 67 60 ...
## $ Age.Group.at.Diagnosis.in.Years           : chr  "31-50" ">71" "61-70" "61-70" ...
## $ Cancer.Type                               : chr  "Breast Cancer" "Esophagogastric Cancer"
## $ Cancer.Type.Detailed                     : chr  "Breast Mixed Ductal and Lobular"
## $ Drug.Type                                : chr  "PD-1/PDL-1" "PD-1/PDL-1" "PD-1/PDL-1"
## $ Gene.Panel                               : chr  "IMPACT341" "IMPACT341" "IMPACT341"
## $ Institute.Source                          : chr  "MSKCC" "MSKCC" "MSKCC" "MSKCC"
## $ Metastatic.Site                          : chr  NA NA NA NA ...
## $ Mutation.Count                           : int  5 6 13 10 12 12 6 3 5 8 ...
## $ Oncotree.Code                            : chr  "MDLC" "GEJ" "BLCA" "BLCA" ...
## $ Overall.Survival..Months.                 : int  0 1 42 43 57 12 18 4 1 8 ...
## $ Overall.Survival.Status                   : chr  "1:DECEASED" "1:DECEASED" "0:LIVING"
## $ Primary.Tumor.Site                       : chr  "Breast" "Esophagus" "Bladder" "Bladder"
## $ Sample.Class                             : chr  "Tumor" "Tumor" "Tumor" "Tumor"
## $ Number.of.Samples.Per.Patient             : int  1 1 1 1 1 1 1 1 1 1 ...
## $ Sample.coverage                           : int  835 1176 900 795 905 783 997 506
## $ Sample.Type                              : chr  "Primary" "Primary" "Primary" "Primary"
## $ Sex                                       : chr  "Female" "Male" "Male" "Male" ...
## $ Somatic.Status                           : chr  "Matched" "Matched" "Matched" "Matched"
## $ TMB..nonsynonymous.                      : num  5.55 6.65 15.53 9.98 13.31 ...
## $ Tumor.Purity                             : chr  "25" "30" "70" "30" ...
```

PREGUNTA DIRIGIDA 1. Sabiendo cómo se accede a las variables y recordando que la media se puede calcular con la función `mean()`, ¿sabríais decirme cuál es la media de la variable `Age.at.Which.Sequencing.was.Reported..Days`?

Filtrar una variable Supongamos que me interesa conocer cuál ha sido la media de supervivencia de los pacientes vivos y los exitus.

```
mean(tmb$Overall.Survival..Months.)
```

```
## [1] 14.07827
```

```
mean(tmb$Overall.Survival..Months.[tmb$Overall.Survival.Status=="1:DECEASED"])
```

```
## [1] 9.4375
```

```
mean(tmb$Overall.Survival..Months.[tmb$Overall.Survival.Status=="0:LIVING"])
```

```
## [1] 18.73583
```

PREGUNTA DIRIGIDA 2. Ahora que sabemos cómo filtrar una variable, ¿cuál es la media de la variable Overall.Survival..Months. en mujeres?

Función summary() Esta pregunta la podríamos haber resuelto también con la función summary(), que nos devuelve un resumen de las características de una variable.

```
summary(tmb$Overall.Survival..Months.)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      0.00   4.00   11.00   14.08   20.00   80.00
```

```
summary(tmb$Overall.Survival..Months.[tmb$Overall.Survival.Status=="1:DECEASED"])
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      0.000   3.000   7.000   9.438  13.000   68.000
```

```
summary(tmb$Overall.Survival..Months.[tmb$Overall.Survival.Status=="0:LIVING"])
```

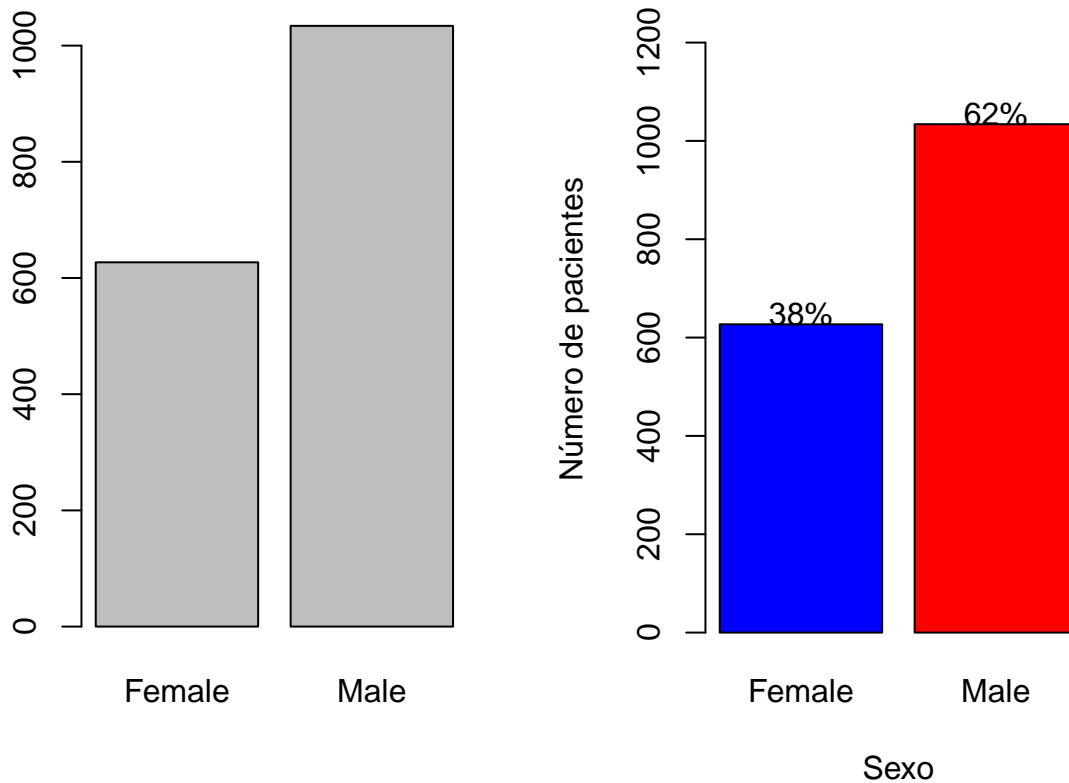
```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      0.00   8.00   16.00   18.74   25.00   80.00
```

```
range(tmb$Overall.Survival..Months.[tmb$Overall.Survival.Status=="0:LIVING"])
```

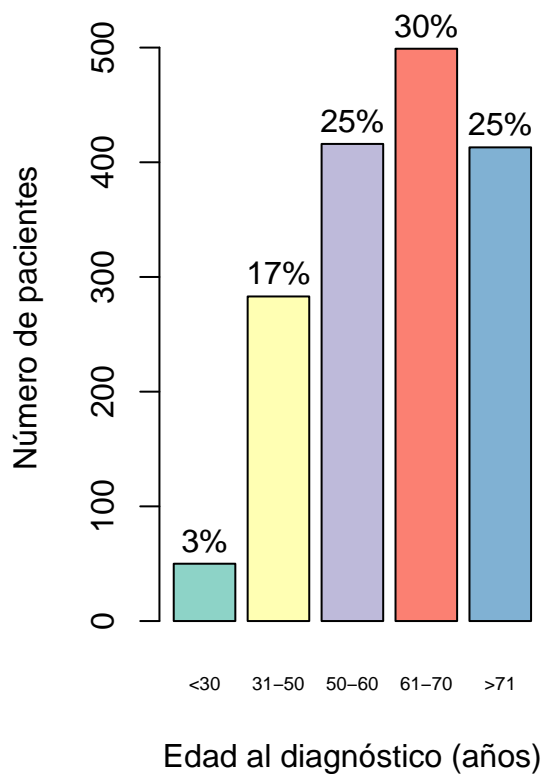
```
## [1] 0 80
```

Gráficos de frecuencia: funciones table() y barplot() Cuando hacemos una exploración de datos es muy habitual que queramos obtener una tabla de frecuencias. Esto lo conseguimos con la función table(). Esta tabla la podemos usar para crear un gráfico de frecuencias con la función barplot.

```
par(mfrow=c(1,2))
# Sexo
sex.freq<-table(tmb$Sex)
barplot(sex.freq) # Lo más sencillo
sex.bar<-barplot(sex.freq,
  col=c("blue","red"),
  xlab="Sexo",
  ylab="Número de pacientes",
  ylim = c(0,max(sex.freq)+200),
  ) # Un poco más bonito
text(x=sex.bar, y=sex.freq+20,
  labels=paste0(round(sex.freq/sum(sex.freq)*100),"%") ,cex=1)
```



```
# Edad
age.freq<-table(tmb$Age.Group.at.Diagnosis.in.Years)[c(1,3,4,5,2)]
age.bar<-barplot(age.freq,
  col=brewer.pal(5, "Set3"),
  xlab="Edad al diagnóstico (años)",
  ylab="Número de pacientes",
  ylim = c(0,max(age.freq)+30),
  cex.names=0.6
)
text(age.bar, age.freq+20, paste0(round(age.freq/sum(age.freq)*100), "%") ,cex=1)
```



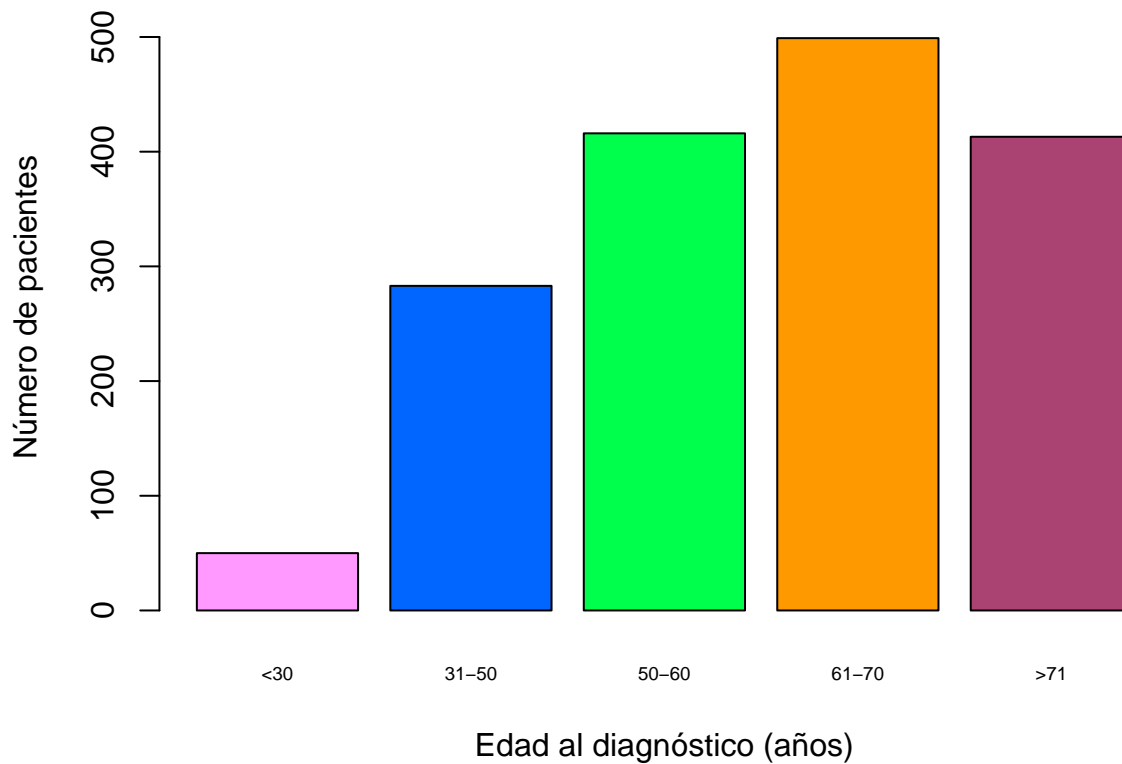
El color en los gráficos Para cambiar los colores de nuestros gráficos R cuenta con varios recursos. Si hacéis una búsqueda en internet de “R color sheet” os encontraréis un pequeño manual con todas las posibilidades. Vamos a ver algunas de ellas.

```
age.bar<-barplot(age.freq,
  col=c("#FF99FF", "#0066FF", "#00FF4D", "#FF9900", "#AA4371"), # 1 color por grupo,
```

```

xlab="Edad al diagnóstico (años)",
ylab="Número de pacientes",
ylim = c(0,max(age.freq)+30),
cex.names=0.6
)

```



Colores predeterminados:

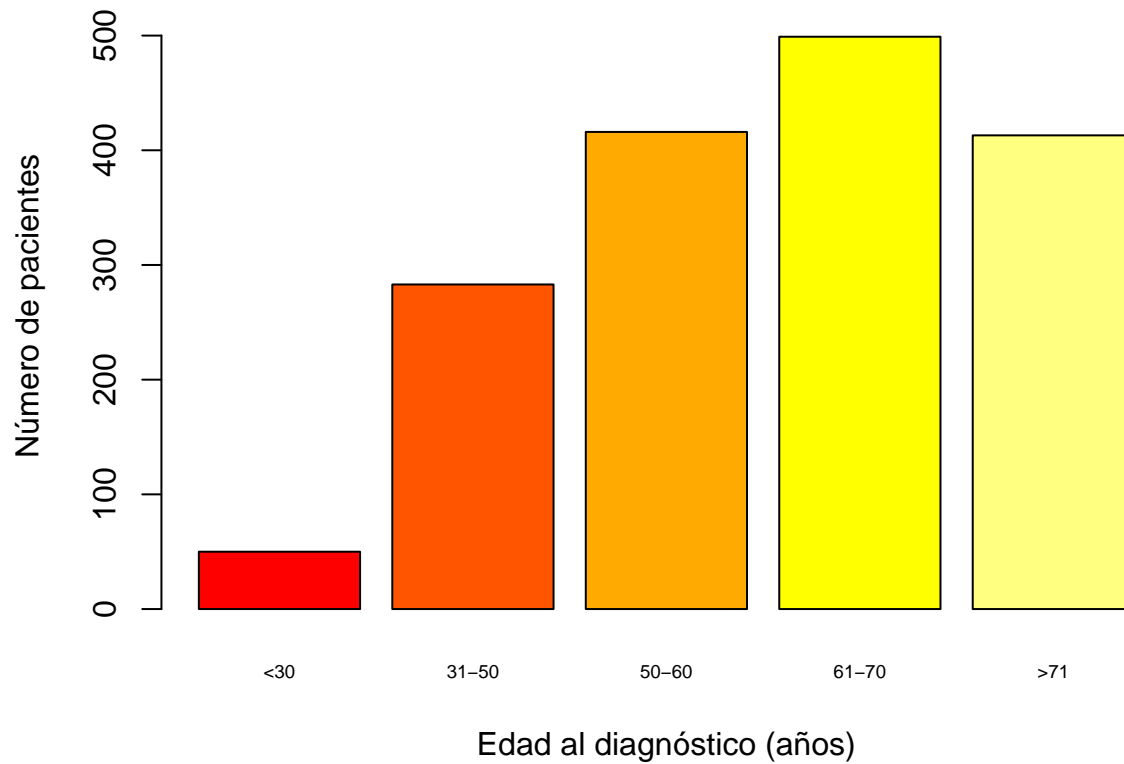
- heat.colors
- terrain.colors
- topo.colors

```

age.bar<-barplot(age.freq,
  col=heat.colors(5), ## Indicamos el n° de colores que necesitamos
  xlab="Edad al diagnóstico (años)",
  ylab="Número de pacientes",
  ylim = c(0,max(age.freq)+30),

```

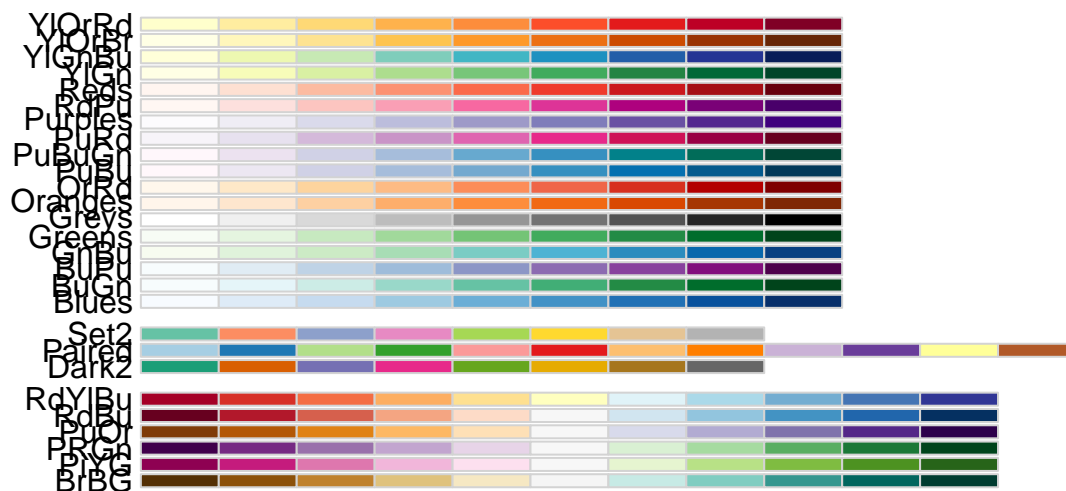
```
cex.names=0.6  
)
```



```
# install.packages("RColorBrewer") # descomentar para instalar  
library(RColorBrewer)
```

Paletas predefinidas Otras opciones de colores dentro de BrewerColors: “Set1” “Set2” “Paired” “Dark2” “RdYlBu” “BrBG”

```
display.brewer.all(colorblindFriendly = TRUE)
```



```
# Edad
age.freq<-table(tmb$Age.Group.at.Diagnosis.in.Years)[c(1,3,4,5,2)]
age.bar<-barplot(age.freq,
  col=brewer.pal(5, "Set3"),
  xlab="Edad al diagnóstico (años)",
  ylab="Número de pacientes",
  ylim = c(0,max(age.freq)+30),
  cex.names=0.6
)
text(age.bar, age.freq+20, paste0(round(age.freq/sum(age.freq)*100), "%") ,cex=1)
```

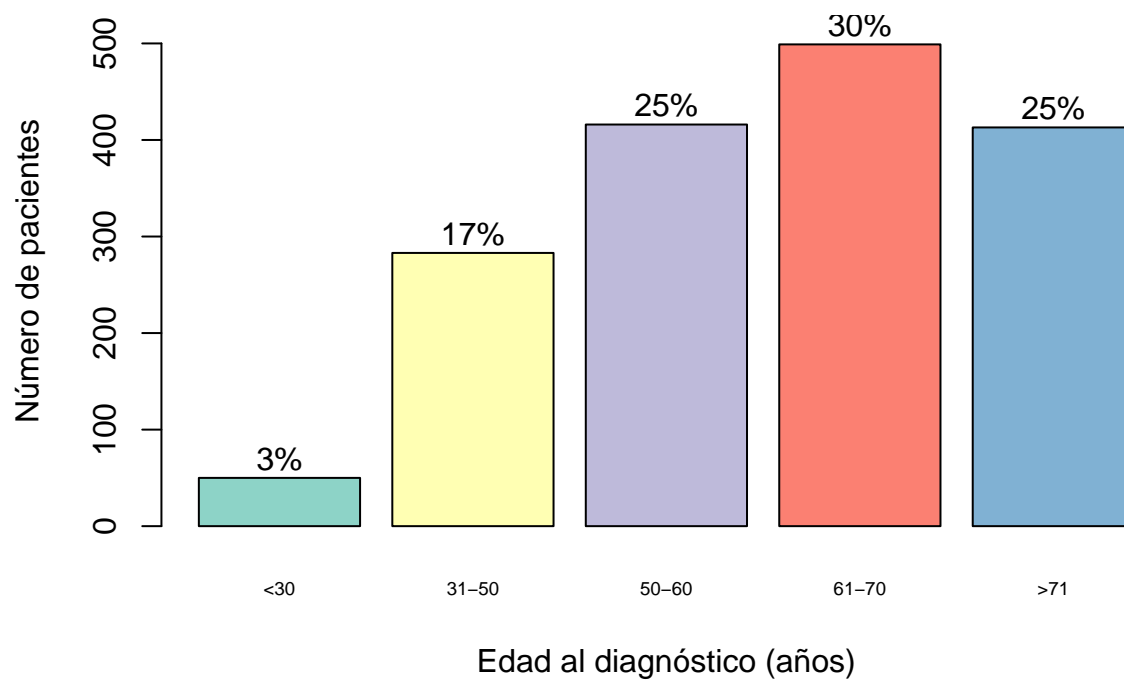
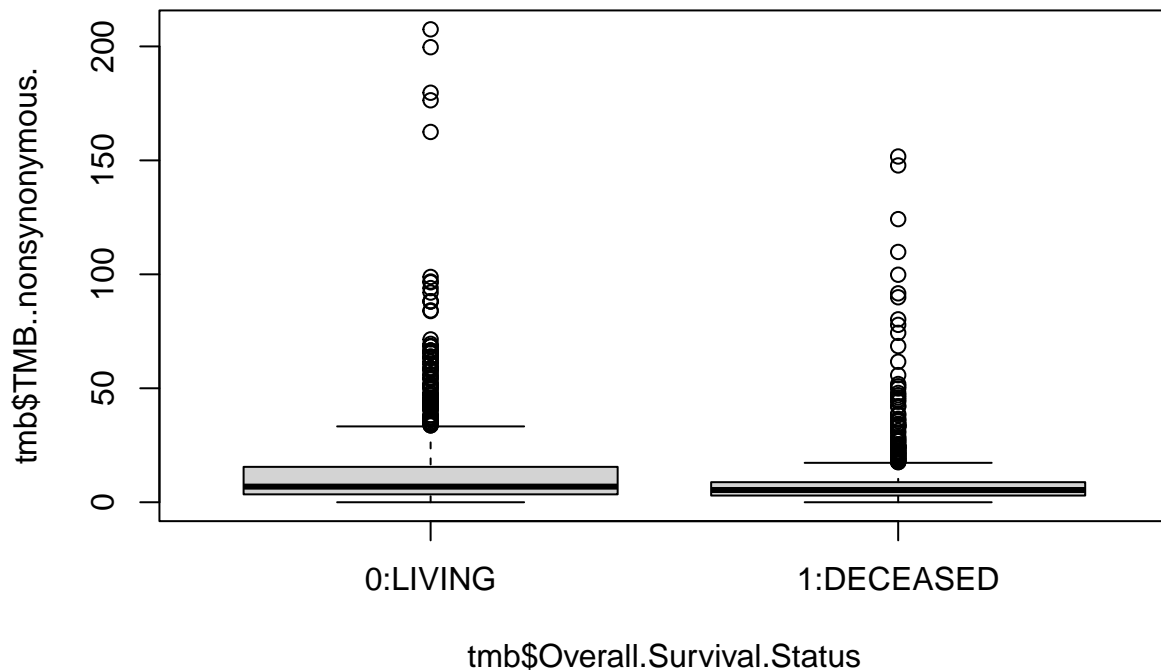


Gráfico de cajas y bigotes: función `boxplot()` Otro gráfico importante son los gráficos de cajas y bigotes para ver la dispersión y simetría de nuestros datos.

```
boxplot(tmb$TMB..nonsynonymous.~tmb$Overall.Survival.Status)
```

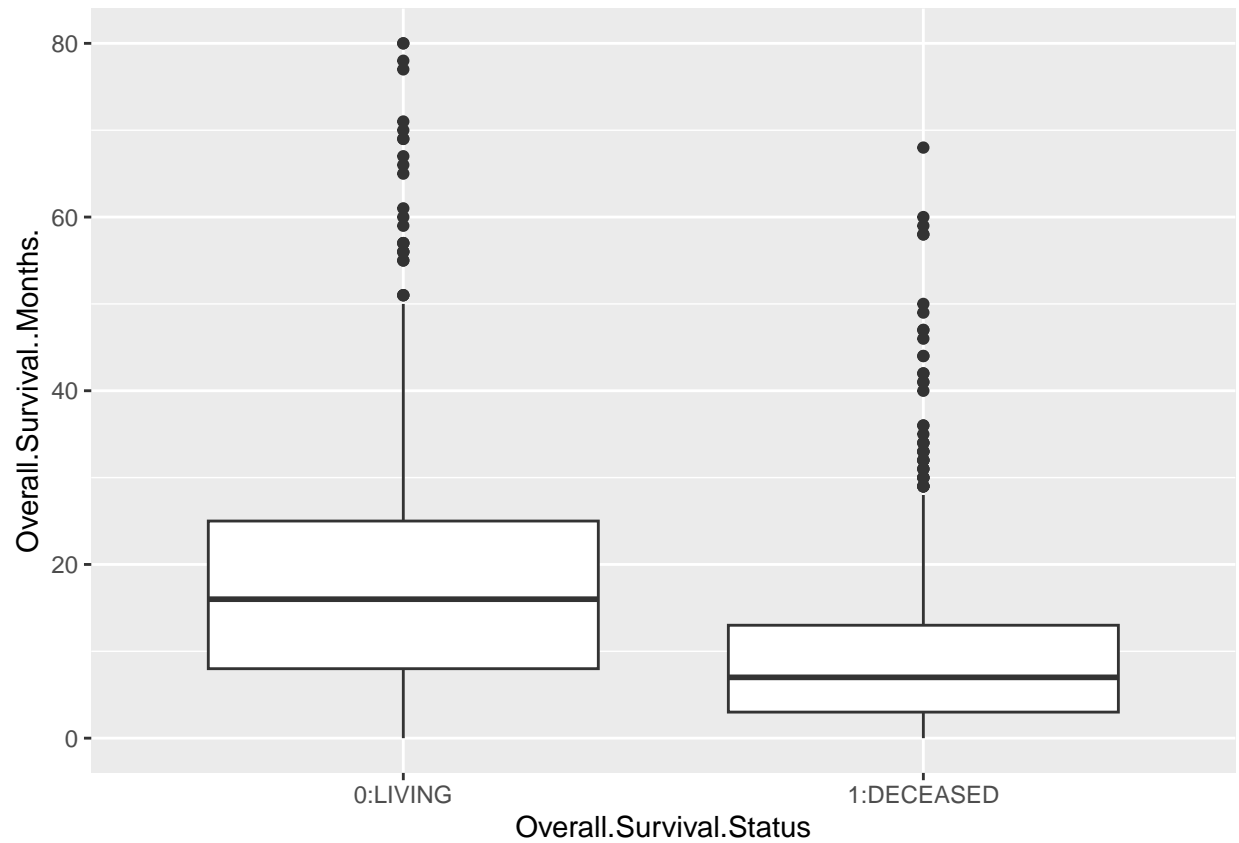



Gráficos con ggplot2 Como veis, estamos usando R base para hacer los gráficos. Sin embargo, podemos añadir más características usando ggplot2. En esta plantilla se explica de forma resumida las principales funciones de ggplot. Instalamos la librería para poder usarla.

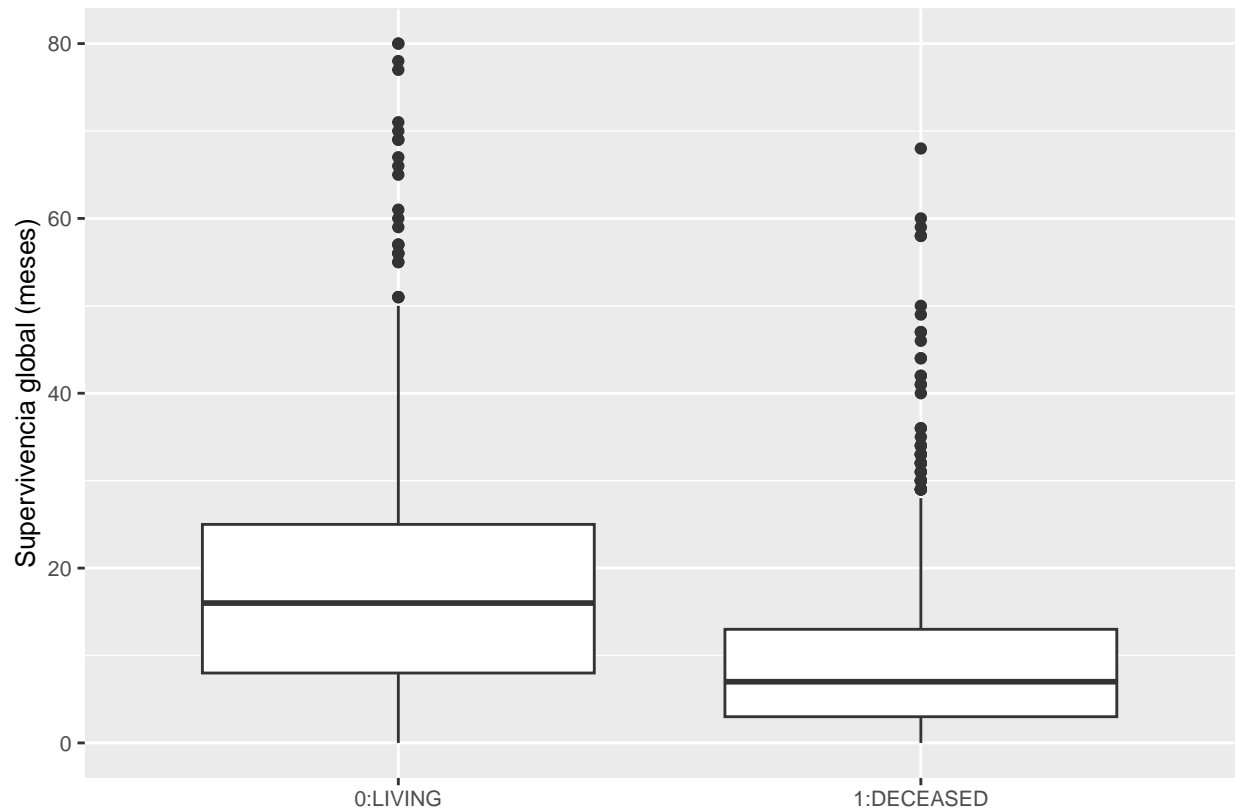
```
# install.packages("ggplot2") # Descomentar esta línea para instalarla
library(ggplot2)
```

Vamos a ver cómo hacer tres tipos de gráficos con ggplot2. ##### ggplot2: boxplot

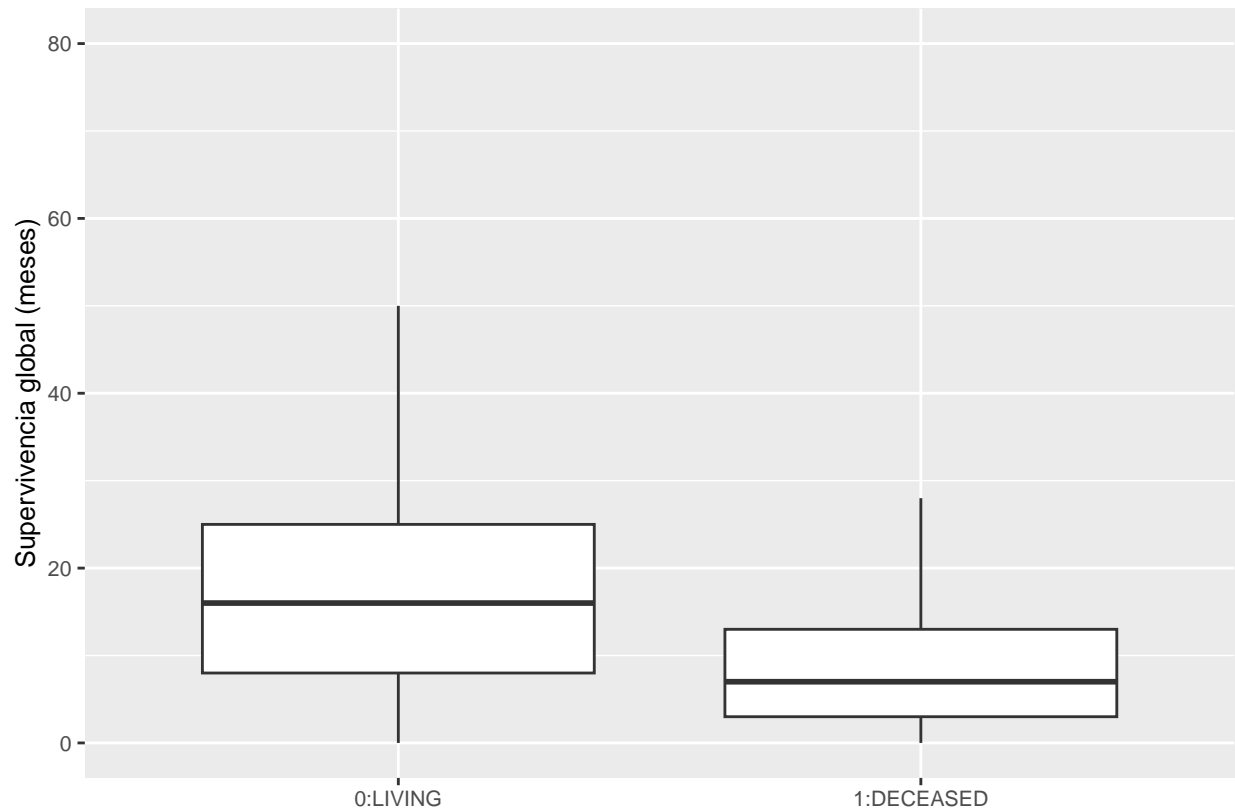
```
ggplot(data=tmb,
       aes(x=Overall.Survival.Status,y=Overall.Survival..Months.))+
  geom_boxplot()
```



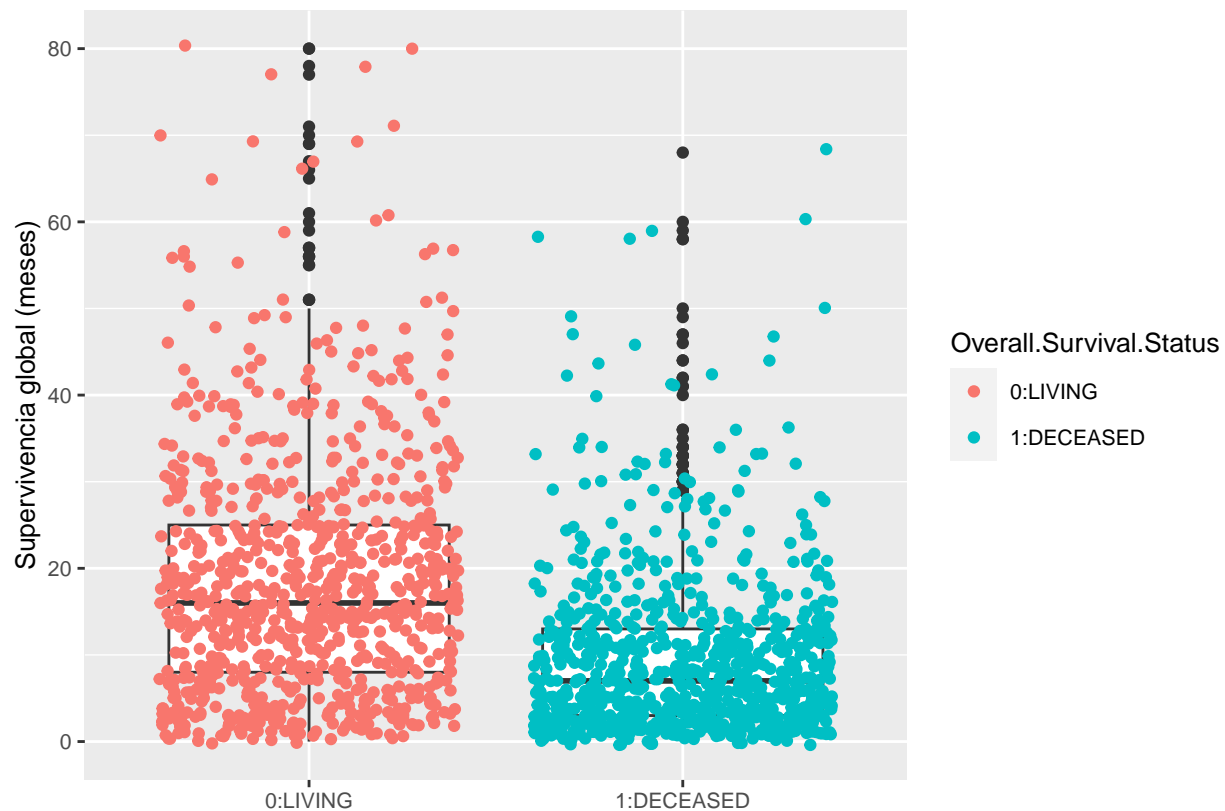
```
ggplot(data=tmb,
       aes(x=Overall.Survival.Status,y=Overall.Survival..Months.))+
  geom_boxplot() +
  labs(y="Supervivencia global (meses)",x=" ") +
  theme(text=element_text(size=10))
```



```
ggplot(data=tmb,  
       aes(x=Overall.Survival.Status,y=Overall.Survival..Months.))+  
  geom_boxplot(outlier.shape = NA) +  
  labs(y="Supervivencia global (meses)",x=" ") +  
  theme(text=element_text(size=10))
```



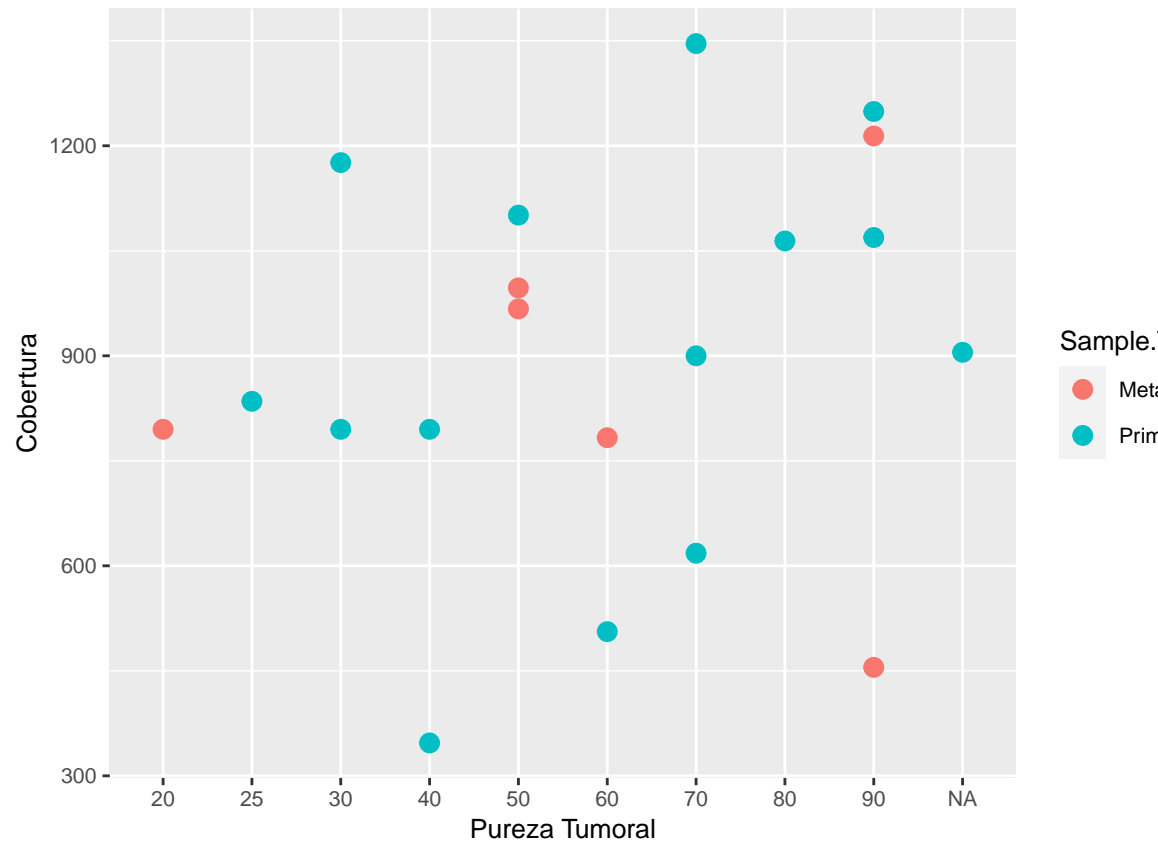
```
ggplot(data=tmb,  
       aes(x=Overall.Survival.Status,y=Overall.Survival..Months.),  
       fill=factor(Overall.Survival.Status))+  
geom_boxplot()+  
geom_jitter(aes(colour=Overall.Survival.Status))+  
theme(text=element_text(size=10))+  
labs(y="Supervivencia global (meses)",x=" ")
```



```
ggplot(data=tmb,
       aes(x=Overall.Survival.Status,y=Overall.Survival..Months.),
       fill=factor(Overall.Survival.Status))+
  geom_boxplot(outlier.shape = NA)+
  geom_jitter(aes(colour=Overall.Survival.Status))+
  theme(text=element_text(size=10))+
  labs(y="Supervivencia global (meses)",x=" ")
```

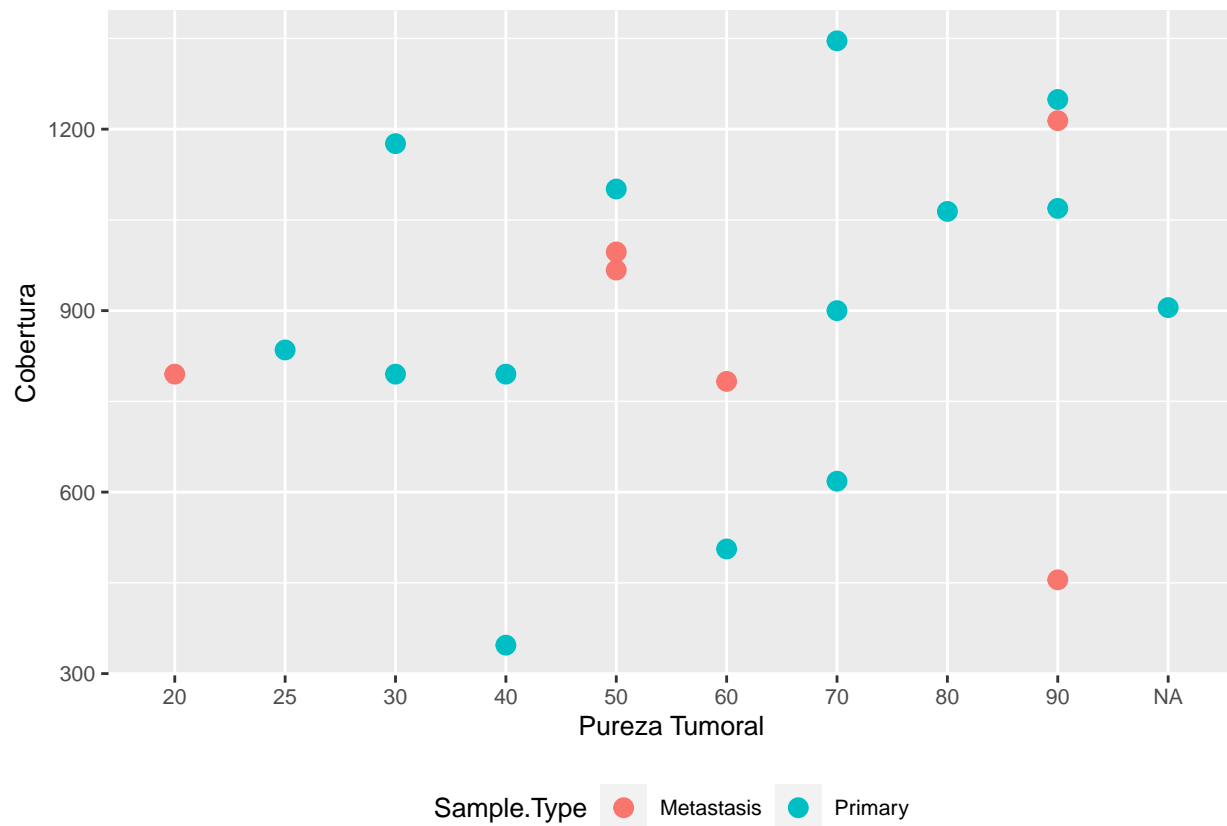


```
## Mostrar los datos filtrados:
ggplot(data=tmb[1:20,],
       aes(y=Sample.coverage, x=Tumor.Purity, color=Sample.Type)) +
  geom_point(size=3) +
  theme(text=element_text(size=10))+
  labs(y="Cobertura", x="Pureza Tumoral")
```

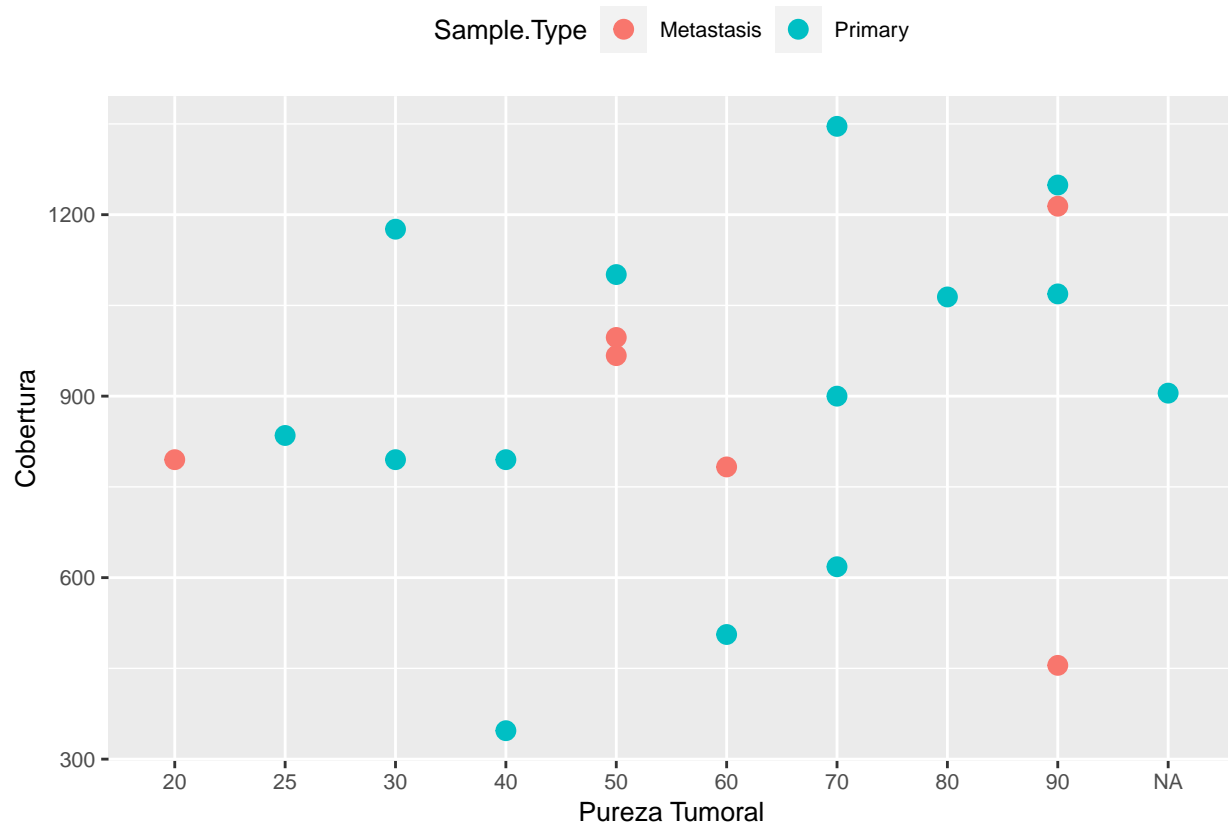


ggplot2: dotplot

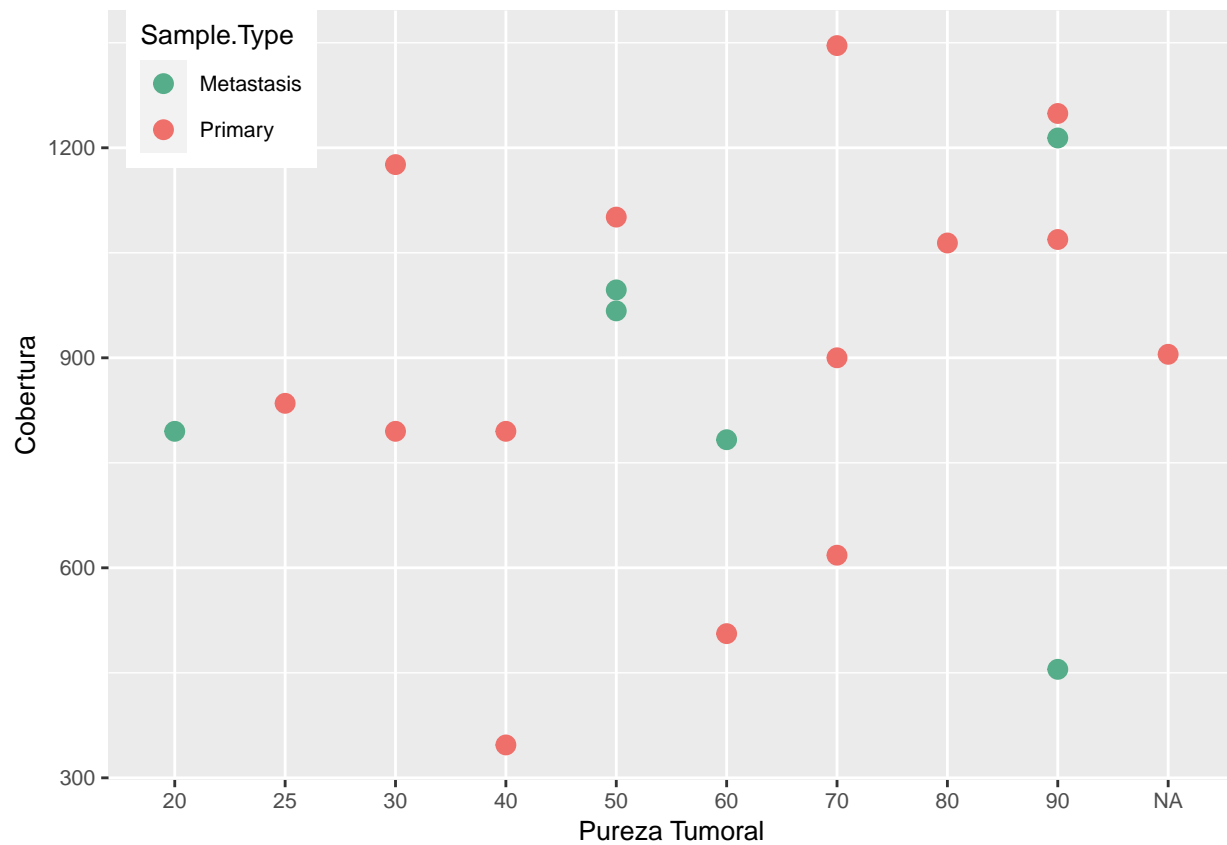
```
ggplot(data=tmb[1:20,],
       aes(y=Sample.coverage, x=Tumor.Purity, color=Sample.Type)) +
  geom_point(size=3) +
  theme(text=element_text(size=10), legend.position = "bottom")+
  labs(y="Cobertura",x="Pureza Tumoral")
```



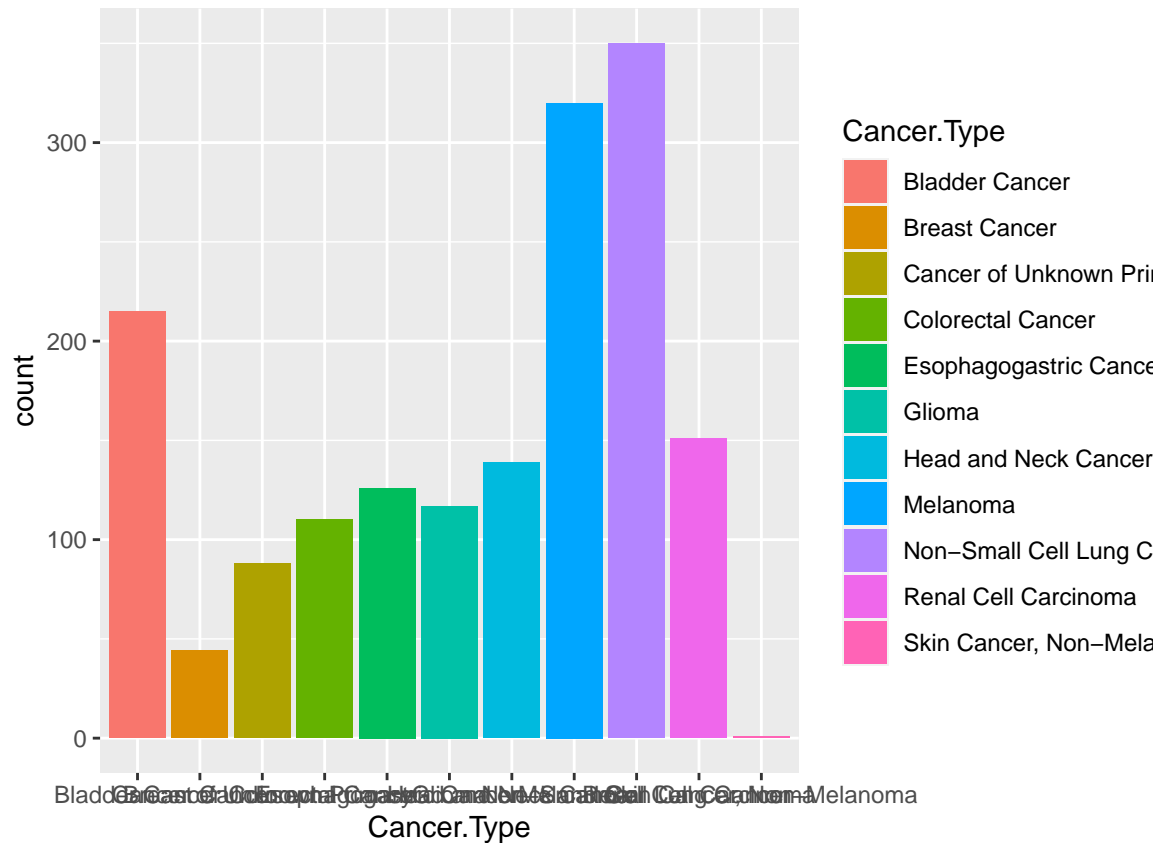
```
ggplot(data=tmb[1:20,],
       aes(y=Sample.coverage, x=Tumor.Purity, color=Sample.Type)) +
  geom_point(size=3) +
  theme(text=element_text(size=10), legend.position = "top")+
  labs(y="Cobertura",x="Pureza Tumoral")
```

```
cols <- c("#55AD89", "#EF6F6A") ## cambiamos el color
ggplot(data=tmb[1:20,],
  aes(y=Sample.coverage, x=Tumor.Purity, color=Sample.Type)) +
  geom_point(size=3) +
  scale_color_manual(values = cols) +
  theme(text=element_text(size=10), legend.position = c(0.1, 0.9))+
  labs(y="Cobertura",x="Pureza Tumoral")
```



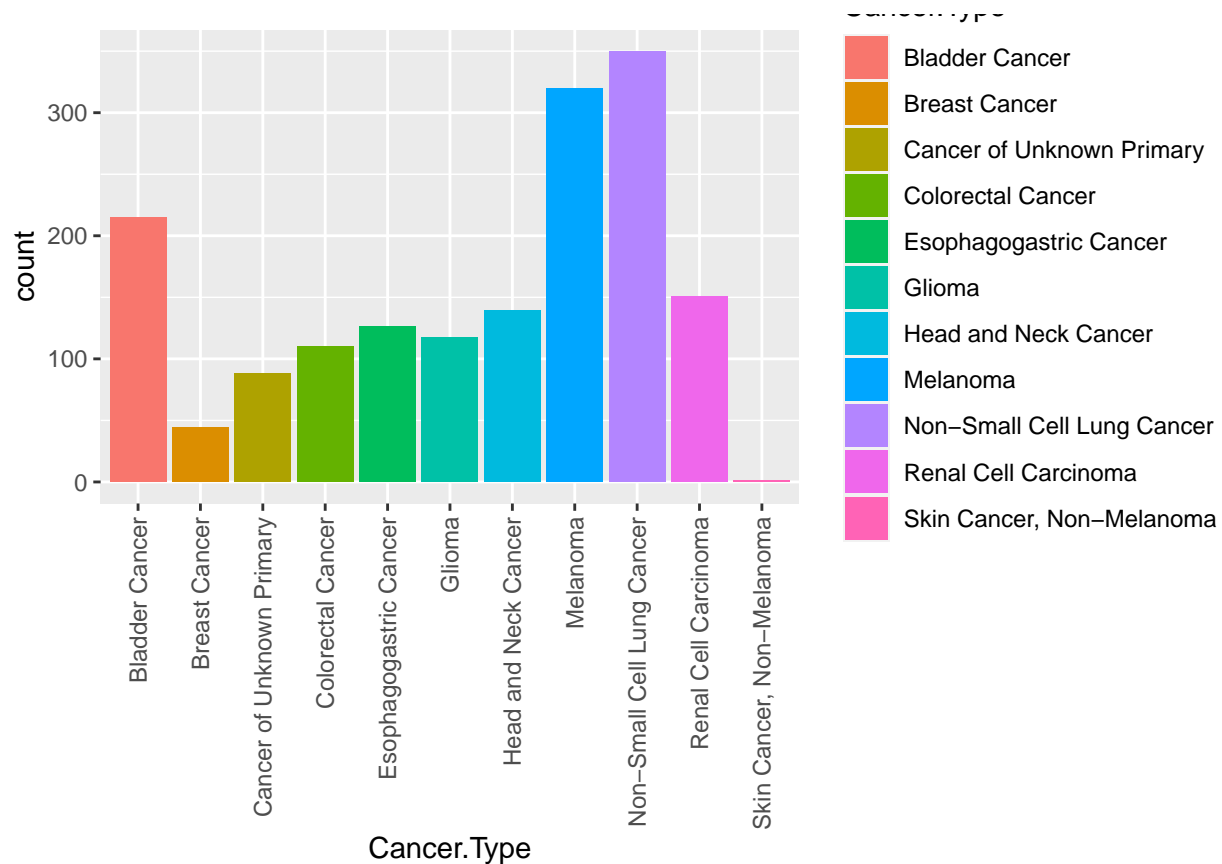
```
ggplot(data=tmb,
       aes(x=Cancer.Type, fill=Cancer.Type)) +
  geom_bar()
```



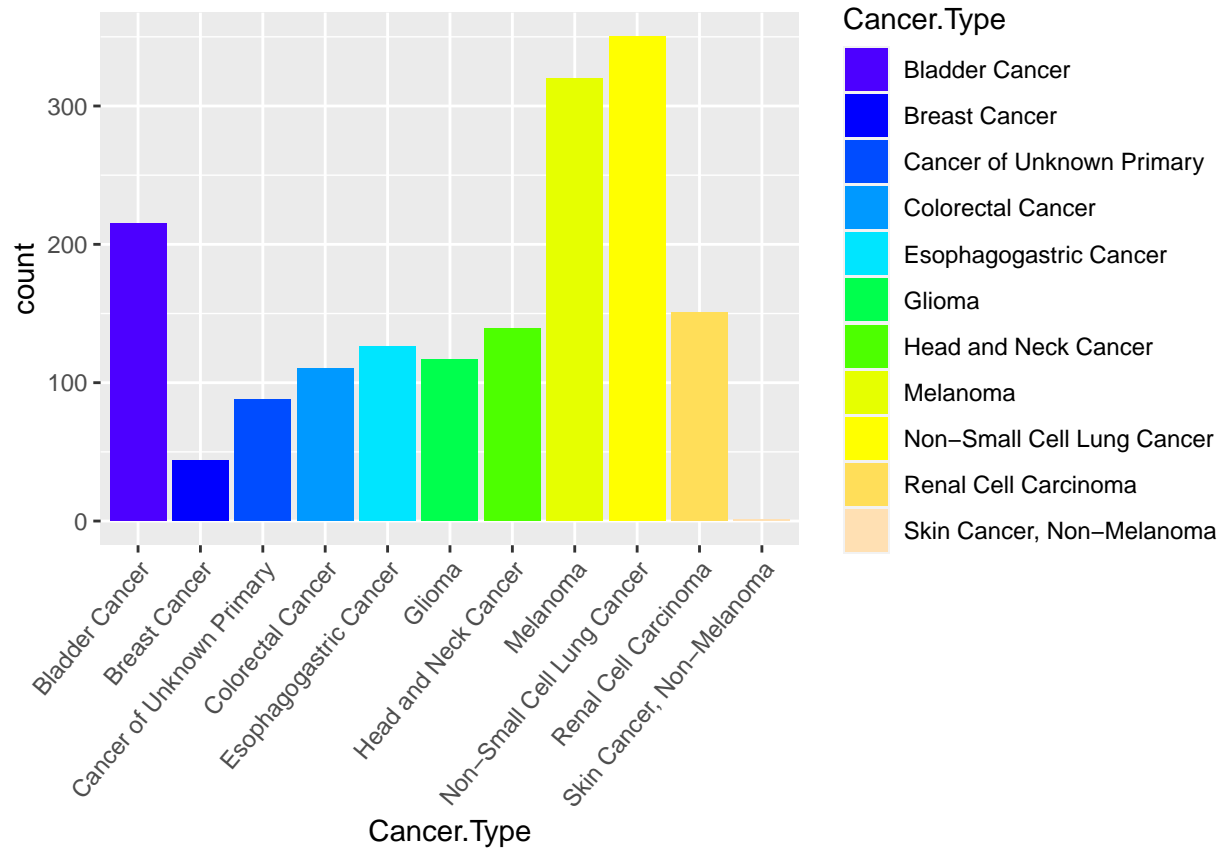
ggplot2: barplot

```
# The value of hjust (horizontal) and vjust (vertical) are only defined between 0 and 1
# 0 means left-justified
# 1 means right-justified

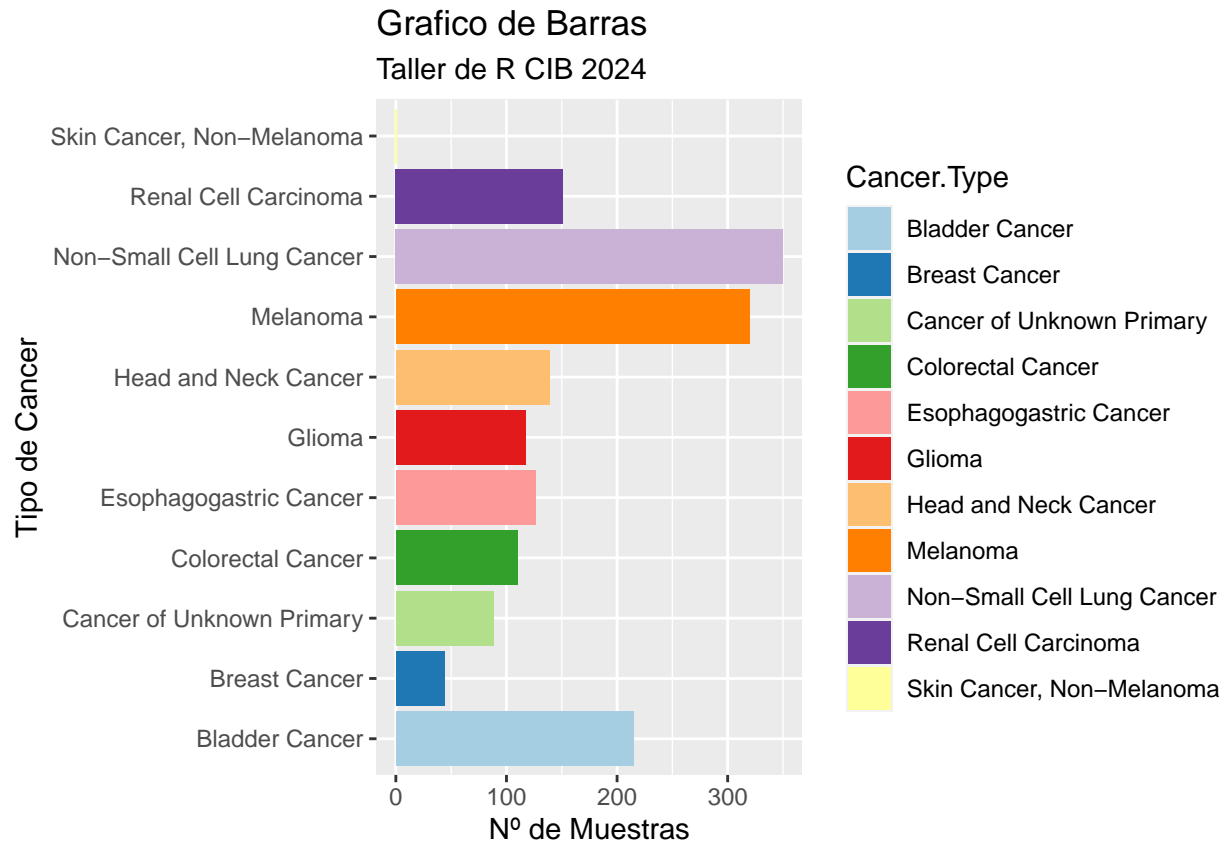
ggplot(data=tmb,
       aes(x=Cancer.Type, fill=Cancer.Type)) +
  geom_bar() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



```
ggplot(data=tmb,
       aes(x=Cancer.Type, fill=Cancer.Type)) +
  geom_bar() +
  theme(axis.text.x = element_text(angle = 50, hjust=1)) +
  scale_fill_manual(values = topo.colors(11))
```



```
ggplot(data=tmb,
       aes(x=Cancer.Type, fill=Cancer.Type)) +
  geom_bar() +
  coord_flip() +
  scale_fill_brewer(palette = "Paired") +
  xlab("Tipo de Cancer") +
  ylab("N° de Muestras") +
  ggtitle("Grafico de Barras", subtitle= "Taller de R CIB 2024")
```



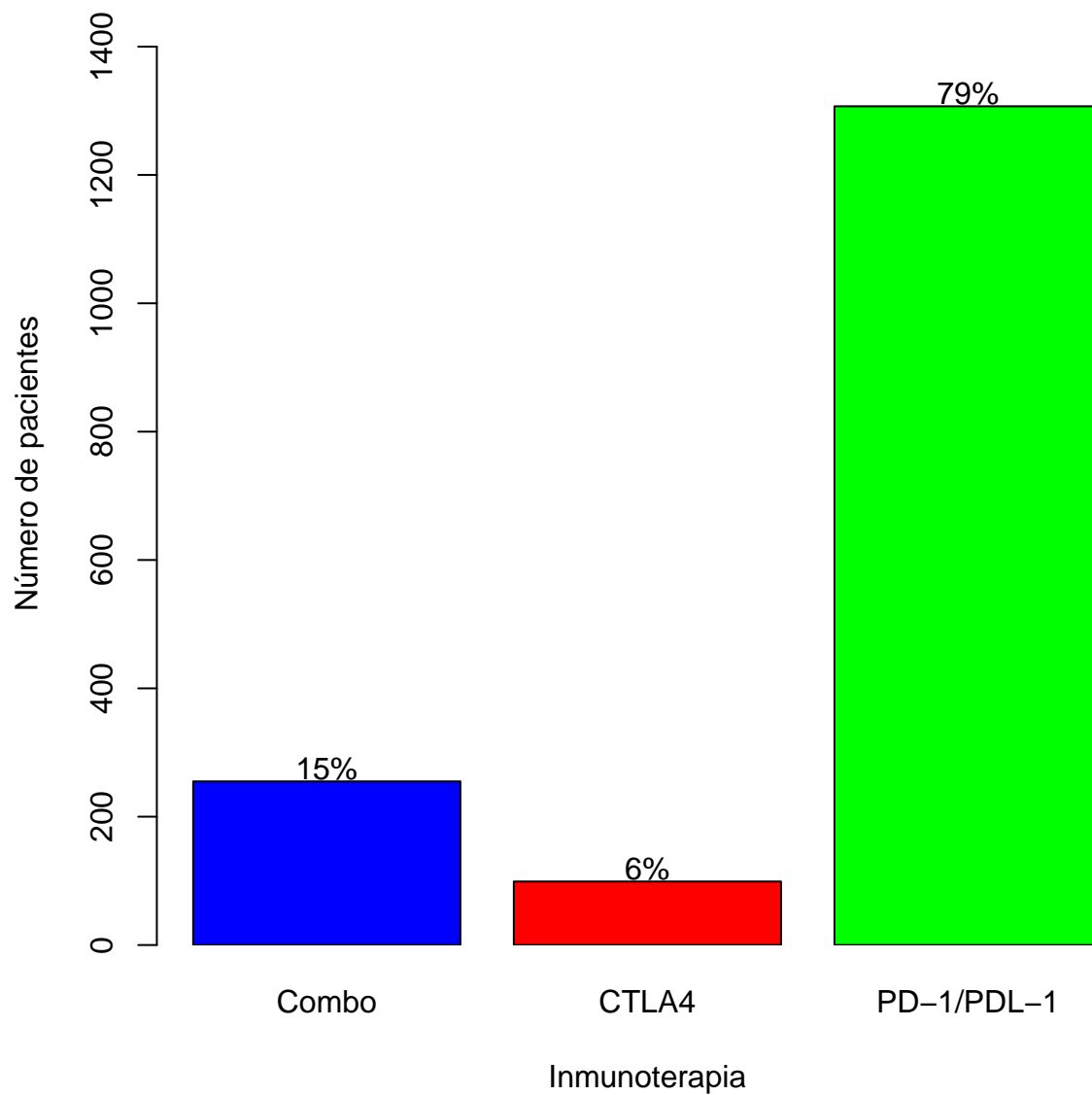
Guardar gráficos: `ggsave()` Para guardar nuestros gráficos nos vamos a la pestaña “plots” y desde ahí lo podemos exportar en el formato que queramos. Pero puede que no queramos estar exportando nuestros gráficos manualmente, sino implementarlo dentro de un bloque de código. La función `ggsave()` nos permite exportar gráficos creados con `ggplot2` en varios formatos (pdf, jpeg, tiff, png, ...). Le podremos especificar el ancho, la altura y las unidades (in, cm, mm, px), así como la resolución (dpi).

```
# No ejecutar
# ruta_file_name = ""
# ggsave(ruta_file_name,
#       plot=last_plot(),
#       device = "png")
```

PREGUNTA DIRIGIDA 3. Os dejamos un rato para que hagáis un gráfico de barras para la variable `Drug.Type`. Queremos enseñar de forma gráfica cuál fue la inmunoterapia de elección más frecuente en esta cohorte.

```
# Inmunoterapia
drug.freq<-table(tmb$Drug.Type)
drug.bar<-barplot(drug.freq,
```

```
col=c("blue","red","green"),
ylim=c(0,1500),
xlab="Inmunoterapia",
ylab="Número de pacientes",
xpd=FALSE
)
text(drug.bar, drug.freq+20, paste0(round(drug.freq/sum(drug.freq)*100),"%"),cex=1)
```

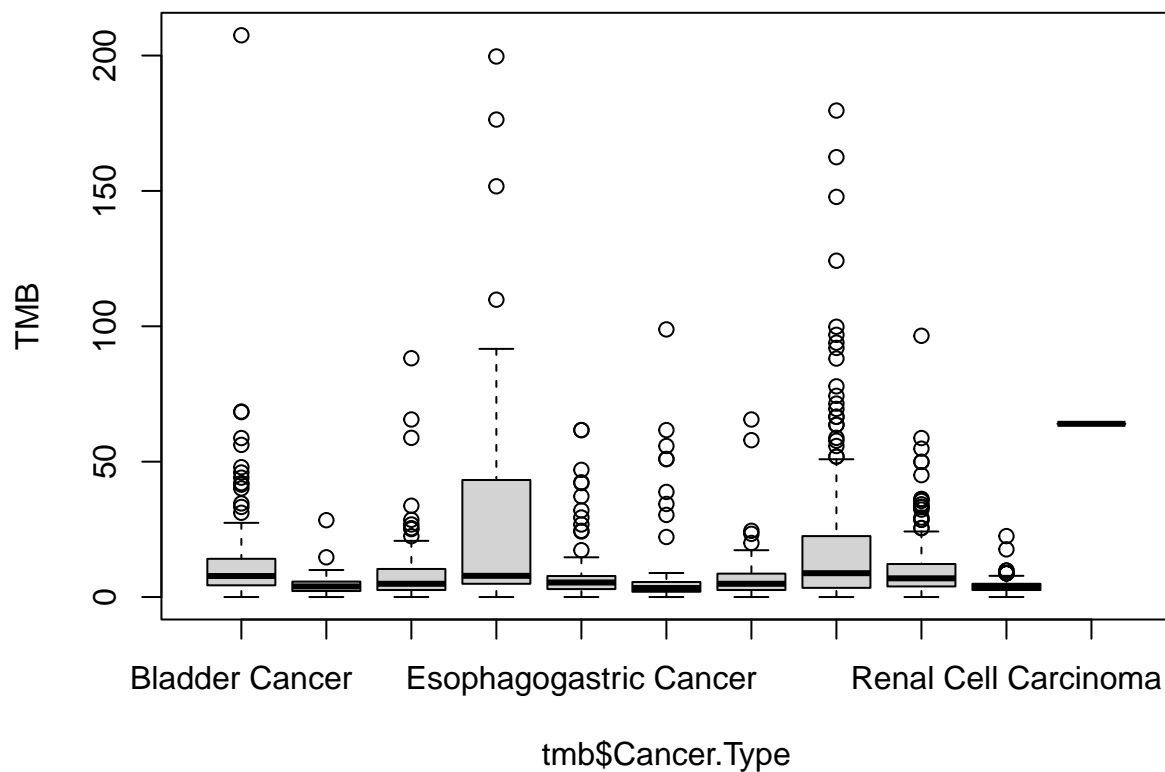


PREGUNTA DIRIGIDA 4. Intentad hacer un gráfico de cajas y bigotes para la vari-

able TMB..nonsynonymous. según el tipo de tumor. Primero hacedlo sencillo ¿Veis alguna tendencia?

```
# Carga mutacional
```

```
boxplot(tmb$TMB..nonsynonymous.~tmb$Cancer.Type,ylab="TMB")
```



Estudiemos ahora algunas de las variables para cada tipo de tumor. Vemos que, excepto en el caso de cáncer de mama, que solo se da en mujeres, y en los tumores primarios de origen desconocido, cáncer colorectal y cáncer de pulmón de célula no pequeña, la mayoría de los casos se dan en hombres, lo que es especialmente notable en el caso del cáncer de vejiga, renal, esofagagástrico y cáncer de piel no melanoma.

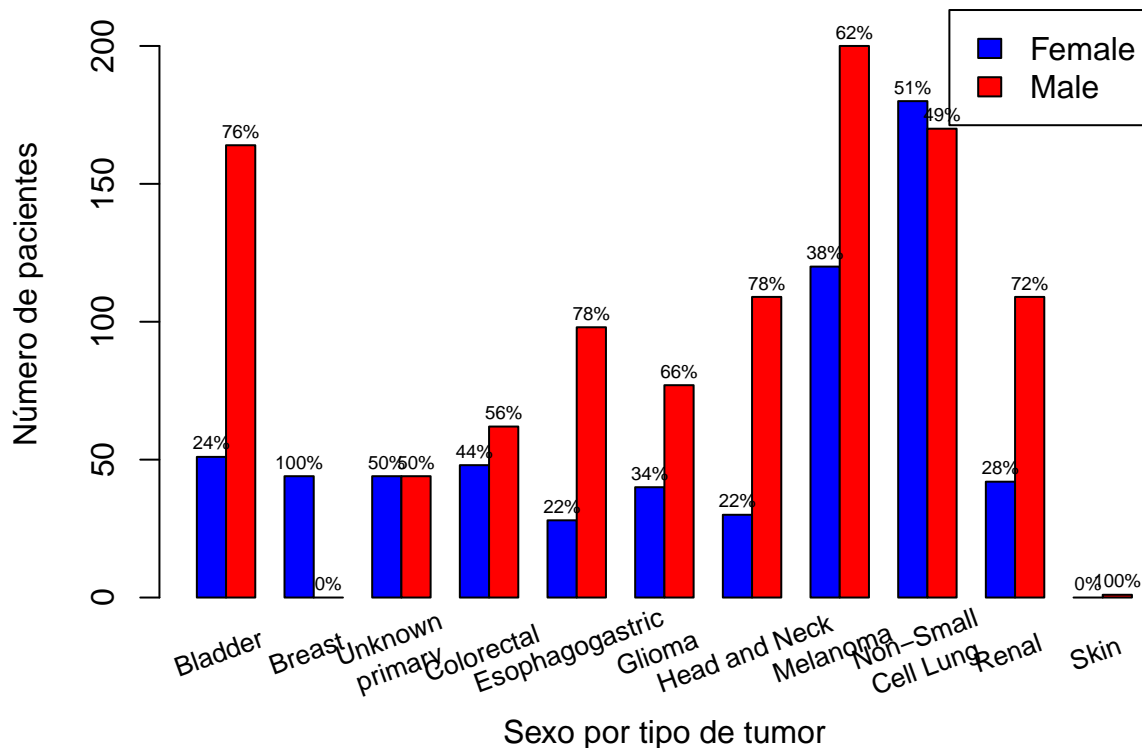
```
sex.freq.tumor<-table(tmb$Sex,tmb$Cancer.Type)
sex.tumor.bar<-barplot(sex.freq.tumor,
  beside=TRUE,
  legend=rownames(sex.freq.tumor),
  col=c("blue","red"),
  xlab="Sexo por tipo de tumor",
  ylab="Número de pacientes",
```



```

ylim = c(0,max(sex.freq.tumor)+20),
xaxt="n",
cex.names=0.5)
text(sex.tumor.bar, sex.freq.tumor+5 ,
     paste0(apply(sex.freq.tumor,2,function(x){round(x/sum(x)*100)}), "%") ,cex=0.6)
tumor.labels <- c("Bladder","Breast","Unknown\nprimary","Colorectal",
                  "Esophagogastric","Glioma","Head and Neck","Melanoma",
                  "Non-Small\nCell Lung","Renal","Skin")
text(cex=0.8, x=colMeans(sex.tumor.bar)-.25, y=-20,
     tumor.labels,xpd=TRUE, srt=20)

```



En cuanto a la edad, llama la atención que en el cáncer colorectal hay un porcentaje aparentemente mayor de pacientes de 31-50 años en comparación con los otros tipos.

```

age.freq.tumor<-table(tmb$Age.Group.at.Diagnosis.in.Years,
                      tmb$Cancer.Type)[c(1,3,4,5,2),]
age.tumor.bar<-barplot(age.freq.tumor,
                       beside=TRUE,
                       legend=rownames(age.freq.tumor),

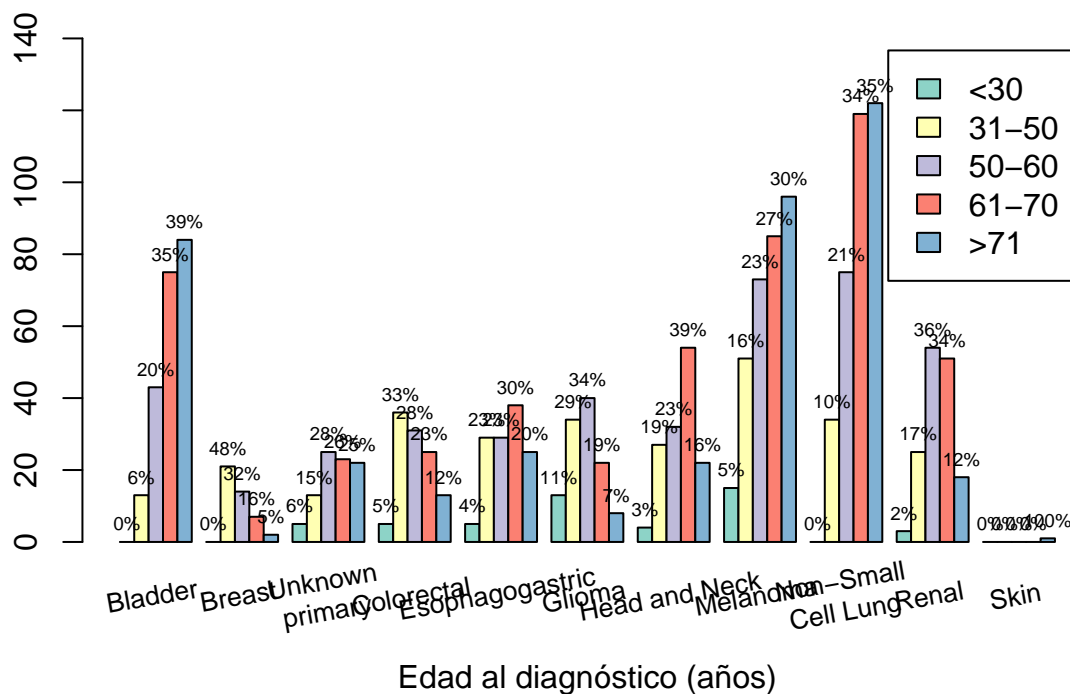
```

```

col=brewer.pal(5, "Set3"),
xlab="Edad al diagnóstico (años)",
ylim = c(0,max(age.freq.tumor)+20),
xaxt="n",
cex.names=0.5)
text(age.tumor.bar, age.freq.tumor+5,
     paste0(apply(age.freq.tumor,2,function(x){round(x/sum(x)*100)}), "%"),
     cex=0.6)

text(cex=0.8, x=colMeans(age.tumor.bar)-.25, y=-15,
     tumor.labels,
     xpd=TRUE, srt=10)

```

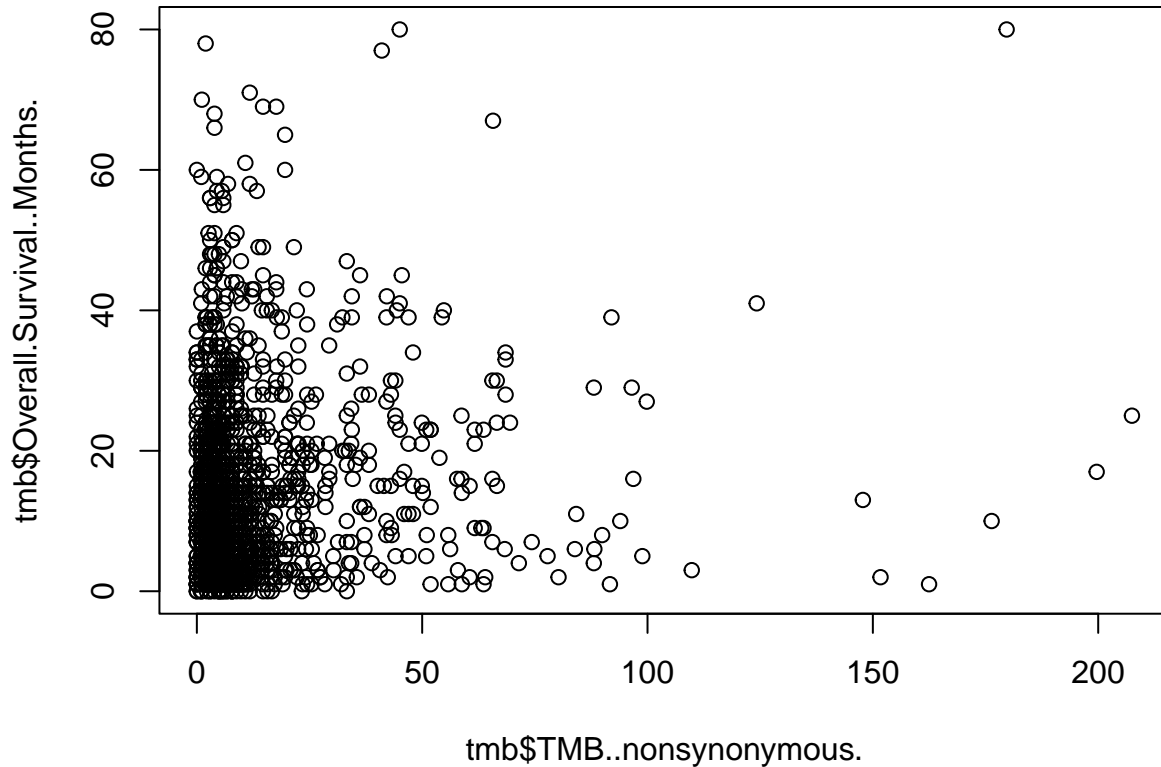


Nos hemos entretenido mirando cada una de las variables, pero lo importante de este artículo es ver si hay una relación entre carga mutacional y eficacia del tratamiento. La eficacia la podemos evaluar en términos de supervivencia global. Exploramos si la carga mutacional tiene relación con la supervivencia, ya que esto determinaría si los inhibidores de los puntos de control autoinmunes funcionan mejor en función de la carga mutacional. Sin embargo, vemos que no hay correlación entre estas dos variables, ni en la cohorte en general, ni en ningún tipo de cáncer en concreto, ya que los coeficientes de correlación son muy bajos.

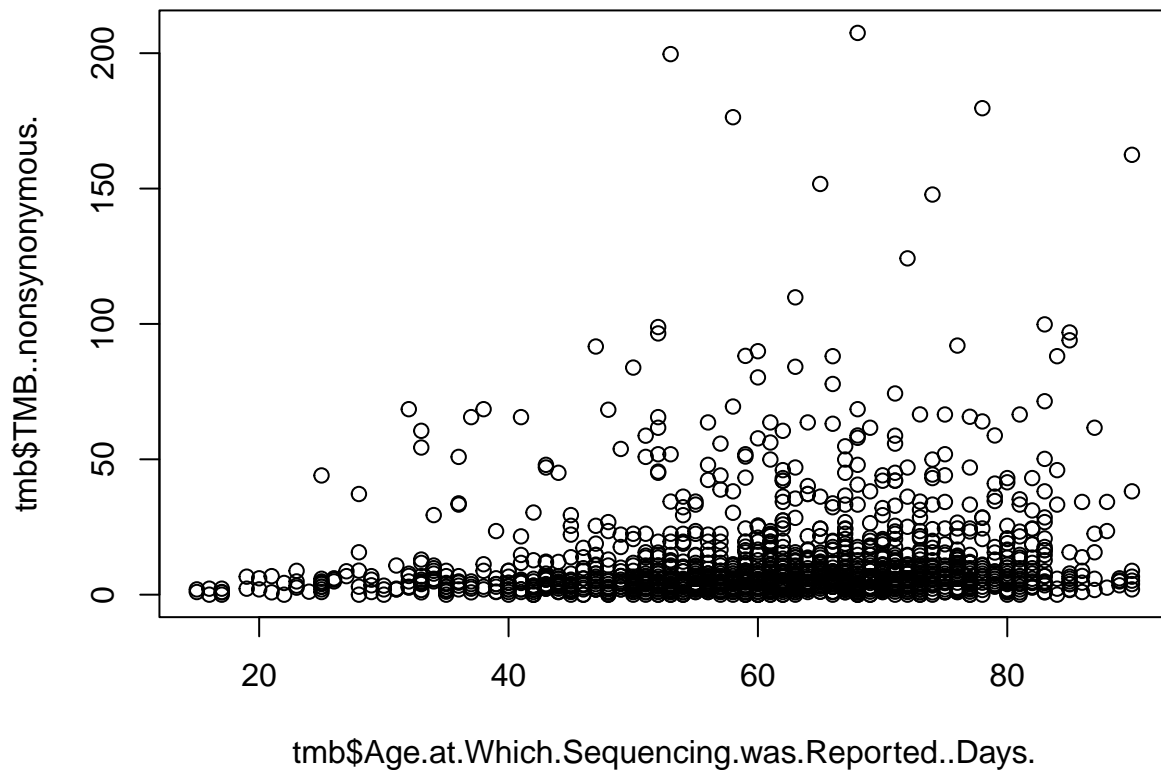
```
cor(tmb$TMB,tmb$Overall.Survival..Months.)
```

```
## [1] 0.1066418
```

```
cor.tmb.cancer<-sapply(unique(tmb$Cancer.Type),function(x){  
  cor(tmb$TMB..nonsynonymous.[tmb$Cancer.Type==x],  
      tmb$Overall.Survival..Months.[tmb$Cancer.Type==x])  
})  
plot(tmb$TMB..nonsynonymous.,tmb$Overall.Survival..Months.)
```



```
plot(tmb$Age.at.Which.Sequencing.was.Reported..Days.,tmb$TMB..nonsynonymous.)
```



Para acabar, os queremos mostrar algunas publicaciones que mencionan que sus figuras están hechas con ggplot2:

- <https://mednexus.org/doi/full/10.1097/JBR.0000000000000110>: creo que no tiene supplementary
- <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7685753/>
- <https://www.sciencedirect.com/science/article/pii/S0092867420311417>

5. Cómo seguir aprendiendo por tu cuenta

Un recurso muy sencillo, gratis y eficaz es swirl, que es un paquete para aprender R dentro de R. La instalación de swirl se realiza con `install.packages()`. Podéis encontrar las instrucciones en este enlace.

```
# install.packages("swirl")
# library(swirl)
# swirl()
```

También hay mucho material disponible en internet. Algunas recomendaciones:

- “An introduction to R” (31/10/2022). <https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf>
- Los libros de Rafael Irizarry, profesor de bioestadística en la Universidad de Harvard. Uno de sus libros es Introduction to Data Science, disponible gratuitamente en Leanpub (<https://leanpub.com/datasciencebook>). También hay la versión en español (<http://rafalab.dfci.harvard.edu/dslibro/>). Echad también un ojo a sus cursos en edX (<https://www.edx.org/es/bio/rafael-irizarry>). Aunque son de pago si queréis el certificado, el material está accesible gratuitamente durante bastante tiempo.
- Cursos gratuitos en Datacamp: <https://www.datacamp.com/courses/free-introduction-to-r>
- Esta página es muy resolutive en temas de estadística: <http://www.sthda.com/english/wiki/what-is-r-and-why-learning-r-programming>

La forma de aprender a usar R es usándolo. Buscad algo que os motive. Un buen ejercicio puede ser usar datos de vuestros proyectos que sean sencillos y podáis comparar vuestros análisis con R y con la herramienta que hayáis empleado hasta ahora. En vuestra cuenta de Posit Cloud ahora tenéis guardado lo que habéis hecho en esta sesión. Os animamos a que instaléis R y RStudio en casa, a que os entretengáis un rato a repasar lo que hemos visto, a que os salgan errores, a intentar solucionarlo, . . . y a que poco a poco os vayáis animando a utilizarlo para analizar vuestros propios datos. Veréis que cuando os deis cuenta de que R es como un folio en blanco en el que podéis pintar con absoluta libertad, ¡no querréis volver a usar otra cosa!