Módulo III

Taller de IA y CD

Maciel-Cruz, EJ

2025-06-12

Contenidos

1 Instalar y Cargar Librerías o Paquetes

1.1 Concepto

Las librerías son una especie de caja de herramientas que se encuentran en R. Son un compilado de muchas funciones específicas y especializadas que nos permiten acelerar nuestro trabajo al tener funciones específicas.

En el R, contamos con funciones precargadas, las cuales hemos utilizado en lo que va del curso (p.e. paste, print, c, etc.), para ampliar dichas funciones, primero debemos instalar una librería de interés, luego cargarla, y ya podemos hacer uso de ellas.

> Tip
>
> No importa lo que intentes hacer, es muy probable que alguien ya lo haya hecho y exista una librería para ello, búscala en Google o alguna IA, o pregúntame por WA o correo.

1.2 Instalación y carga

Existen dos métodos (que yo conozco y utilizo), para instalar y cargar librerías, el primero es el método convencional que se encuentra en la mayoría de recursos en línea, el segundo es un método más corto y simple, que ahorra un par de pasos, pero puede que no funcione con todas las librerías (falla muy ocasionalmente).

1.2.1 Primer método

La forma habitual en que se carga es usando las siguientes líneas de código:

install.packages()

library()

El primer comando sirve para instalar la librería de interés, mientras que el segundo se usa para cargar la librería, es decir, activarla.

El primer paso se realiza solamente una vez, mientras que el segundo se realiza cada vez que iniciamos R y queremos trabajarla.

Este proceso, que puede parecer tedioso, es útil ya que solo cargamos lo que necesitamos, lo que permite hacer mejor uso de recursos, afortunadamente con las computadoras actuales (con al menos 16 gb de RAM), es poco probable que tengas inconvenientes aunque cargues varias librerías.

Solo tenemos que introducir el nombre de la librería dentro de los paréntesis, y corremos la línea. Para instalar la librería, debemos usar comillas, mientras que para la carga no es necesario.

Dentro de este método, que si bien, es diferente, hace uso de la misma base. Solo tenemos que irnos a nuestra cinta de herramientas y elegir "Tools > Install Packages..." y nos arrojará una ventana de opciones, solo buscamos la librería de interés y la instalamos.

Para el proceso de carga, puede ser igual a lo ya mencionada, o irnos a la sección de salida, irnos a "Packages", buscar el que queremos cargar, y palomeamos la casilla a la izquierda.

1.2.2 Segundo método

Este método es mi preferido, ya que nos ahorra un poco de tiempo, y como solemos trabajar con las mismas librerías de forma habitual, es más sencillo. Primeramente debemos instalar la librería "pacman" con el método 1.

Ahora, lo que tenemos que hacer es introducir el siguiente código:

pacman::p_load()

Aquí lo que hacemos es llamar a la librería aunque no esté cargada, luego, buscamos una función específica de la librería utilizando el símbolo "::" (cabe mencionar que este método es aplicada para cualquier librería). Y luego solo introducimos el nombre de la librería que queramos cargar (sin comillas), y la va a instalar (si es que no la tenemos) y al finalizar, la va a cargar. En caso de que ya se tenga dicha librería instalada, solamente la va a cargar.

Si bien, puede no representar mucha diferencia de tiempo en un inicio, la idea es que ya tenga guardado en un script de base las librerías frecuentemente utilizadas y solo copiar y pegar en cada nuevo script, p. e.

```
pacman::p_load(tidyverse,GGally,esquisse,ggThemeAssist,summarytools)
```

En el código anterior, debería instalar (si no se tienen) y luego cargar las librerías mencionadas.

> **Tip**
>
> Te sugiero guardar un archivo plantilla con lo que sueles trabajar siempre, puede ser tu nombre, fecha, título, las librerías que siempre utilizas, etc.

1.3 Librerías más utilizadas y sus funciones

Ya que se mencionó el método de carga, vamos mencionando algunas de las librerías más útiles que podemos utilizar en R.

1.3.1 ggplot2

Es un librería que tiene como objetivo crear gráficos impresionantes (al menos, en mi opinión), ya que son altamente personalizables, y al ser de libre uso y de código abierto, muchos colaboradores han estado actualizando la librería para agregar más funciones, hacer más amigables las ya existentes, y agregar nuevos gráficos.

Además, existen muchas páginas que se dedican a mostrar algunos tipos de gráficos (con su código ejemplo, obvio), que podemos hacer en R.

Unos ejemplos es The R Graph Gallery o r-charts.

Más adelante en el curso haremos varios gráficos con esta librería.

1.3.2 dplyr

Es una excelente herramienta de transformación y manipulación de datos, que tiene como objetivo utilizar una gramática lógica acorde a la palabra para producir una función, por lo que es fácil de aprender para las cuestiones más básicas. Recordemos que el lenguaje de programación en R está basado en el idioma inglés, por lo que es altamente recomendable practir este idioma en el aspecto técnico de ciencia de datos. Es especialmente útil para crear nuevas variables, seleccionar un subconjunto, filtrar, resumir y reordenar nuestras bases de datos.

1.3.3 tidyr

Es otra herramienta de transformación y manipulación de datos, parecido a dplyr, pero con diferente objetivo. Esta librería se enfoca en hacer bases de datos más legibles y entibles, agrupando las variables de forma en que sean más útles para hacer análisis. Con tidyr podemos reorganizar valores, dividir varibales (para obtener datos atómicos, p. e.), manejar valores omitidos

1.3.4 readr y readxl

La librería readr tiene como objetivo facilitar la importación de bases de datos con extensión .csv o .tsv, mientras que readxl lo hace para archivos de Excel.

1.3.5 ggally

Es una extensión de ggplot2 que tiene como objetivo facilitar el trabajo y ahorrar algunos pasos, además de generar un método mucho más eficiente para obtener correlaciones; además, permite hacer una tabla de comparaciones entre las variables deseadas utilizando métodos gráficos y estadísticos. Es de gran utilidad para hacer exploraciones rápidas de datos y ver si puede existir una tendencia.

1.3.6 summarytools

Tal como su nombre lo dice, es una herramienta para ayudar a hacer resúmenes de datos, enfocado a estadística descriptiva. Sirve para cualquier tipo de dato. También es de excelente utilidad para explorar datos.

1.3.7 esquisser

Es una librería que generar un interfaz gráfica para realizar gráficos, es decir, abre una ventana y podemos usar clics para hacer gráficos mediante ggplot2. Es muy útil para hacer un gráfico rápido, sin embargo, se recomienda aprender primeramente la sintaxis de ggplot2 para entender mejor su funcionamiento y mejorar la estética.

1.3.8 ggThemeAssist

Otra herramienta estilo interfaz gráfica. En este caso, nos permite mejorar la estética de un gráfico de ggplot2 mediante una ventana interactiva. Si bien, es muy útil para un principiante, se sugiere empezar trabajando con la sintáxis del código.

1.3.9 tidyverse

Estrictamente no es una librería como tal, realmente es un recopilación de librerías. El paquete de tidyverse junta muchas librerías de trabajo como ggplot2 y readr en una, lo que lo hace más fácil de cargar. Contiene otras librerías (un poco más especializadas a mi punto de vista), que podrían ser útiles para algunos proyectos.

1.4 Objetos y Variables

Las variables, tal como su nombre lo dicen, son aquellos objetos cuyo contenido puede variar. En R, las variables son todo objeto al que le asignemos un valor o contenido. Vamos a crear las siguientes variables, una por una, y analicemos el contenido y proceso de cada una:

```
var1 <- 1
var2 <- c(1:5)
var3 <- as.logical(c(1,0,1,0))
var4 <- c(T,F,F,T)
var5 <- c("bueno","malo","regular")
var6 <- as.factor(c("bueno","malo","regular"))
var7 <- as.character(c(1,0,1,0))
var1 <- var7
```

En R, al ser una programación orientada a objetos, podemos "guardar" o asignar prácticamente cualquier cosa a una cadena de caracteres y así crear un objeto. Las cosas que podemos guardar van desde números, caracteres, funciones, vectores, bases de datos, gráficos, tablas, etc.

Un objeto, a modo de resumen, es cualquier cadena de texto no separada (es decir, evitando espacios), al que nosotros podemos asignarle algo (desde un solo número o letra, hasta gráficos o incluso documentos completos).

> Tip
>
> Al crear objetos, considera que se sobre escriben y se pieden por completo los datos, por lo que debes tener cuidado.

1.5 Sintaxis de R

1.5.1 Asignación de valores

El método simple para asignar es con el operador de asignación que habíamos mencionado. Al utilizar el "Alt + -" ingresamos el operador de asignación. La sintaxis para asignar variables consiste en lo siguiente:

"Objeto a crear" + "operador de asignación" + "valor o contenido a asignar"

Cabe mencionar que el método es universal, es decir, lo que queramos asignar, usamos la misma sintaxis, o sea, que es válido para algo tan simple como un número hasta gráficos o bases de datos. Como lo hemos estado manejando en todo el curso.

1.5.2 Nombres reservados

Si bien, podemos asignar contenido a cualquier cadena de texto, se sugiere (o más bien, es obligatorio), respetar una serie de nombres que se encuentran reservados para funciones nativas de R. Por ejemplo, habíamos menciona la función de "print", si nosotros la sobre escribimos, nos quedamos sin esa función, lo que es mala idea. Podemos revertir el proceso en cualquier momento borrando dicha asginación en nuesto panel de ambiente, pero es mejor utilizar otras cadenas de texto.

A continuación, se muestra una imagen con las principales palabras reservadas en R.

```
break <-1 # palabra reservada (no usar)
else <-2 # palabra reservada (no usar)
FALSE <-3 # palabra reservada (no usar)
for <-4 # palabra reservada (no usar)
function <-5 # palabra reservada (no usar)
if <-6 # palabra reservada (no usar)
in <-7 # palabra reservada (no usar)
Inf <-8 # palabra reservada (no usar)
NA <-9 # palabra reservada (no usar)
NaN <- 10 # palabra reservada (no usar)
next <- 11 # palabra reservada (no usar)
NULL <- 12 # palabra reservada (no usar)
repeat <- 13 # palabra reservada (no usar)
TRUE <- 14 # palabra reservada (no usar)
while <- 15 # palabra reservada (no usar)
```

Así mismo, también contamos con algunas constantes numéricas, como es "e" para exponentes, "pi" para el número Π, o "NaN" que se reserva para valores omitidos.

> **Tip**
>
> Es poco probable que cometamos este error; sin embargo, sugiero que uses nombres en español para tus variables para tener mayor cuidado.

1.5.3 Consulta y borrado de variables

Para consultar nuestras variables creadas, basta con echar un vistazo a nuestro ambiente para visualizar todo lo que tenemos. En esa misma ventana, si utilizamos el formato "grid", podemos tener un resumen del tipo de dato, longitud, tamaño, valores, etc. Además, hay una casilla a la izquierda de cada valor para poder seleccionar algunos.

Si deseamos borrar alguna variable, tenemos dos opciones, usar la función "rm", que proviene de "remove", o seleccionar la casilla en el ambiente y usar la escoba.

```
rm(var1)
```

> **Tip**
>
> Si utilizas la escoba, corres el riesgo de borrar toda tu información del ambiente, aunque suele venir una advertencia antes de proseguir.

1.5.4 Operadores Aritméticos y Booleanos

Los operadores aritméticos tienen la función de hacer operaciones matemáticas básicas, mientas que los Booleanos llevan a cabo operaciones lógicas.

Los operadores aritméticos en R emplean los mismos símbolos que usamos habitualmente en las calculadoras.

```
2+3
```

```
[1] 5
```

```
3-2
```

```
[1] 1
```

```r
2*3
```

```
[1] 6
```

```r
2/2 #División con decimales
```

```
[1] 1
```

```r
2^2
```

```
[1] 4
```

```r
2**2 #Nótese que "^" es lo mismo que "**"
```

```
[1] 4
```

```r
3%/%2 #División entera
```

```
[1] 1
```

Además, podemos llevar a cabo estas operaciones directamente en la consola, por lo que no tenemos necesidad de ponerlas en el script, con lo que también podemos usar R como una calculadora.

Respecto a los operadores Booleanos, estos tienen como objetivo llevar logicidad en nuestras funciones. Son especialmente útiles cuando queremos transformar datos, como cuando queramos obtener un resultado a partir de una serie de condiciones.

Por ejemplo, si tenemos las variables "peso", "diabetes" y "colesterol", podemos pedir una nueva columna llamada "riesgo", y con operadores Booleanos elegir puntos de corte para cada variable y que nos arroje un resultado (habitualmente positivo o negativo, o verdadero y falso), si se cumplen o no las series de condiciones que solicitemos. Veremos ejemplos más detallados en la sección de pre procesado de datos.

Los operadores Booleanos se dividen en relacionales (útiles para comparaciones) y los lógicos, los más comunes son:

```r
x1 <- c(1,2,3)
x1>2
```

```
[1] FALSE FALSE  TRUE
```

```r
x1>=2
```

```
[1] FALSE  TRUE  TRUE
```

```r
x1<2
```

```
[1]  TRUE FALSE FALSE
```

```r
x1<=2
```

```
[1]  TRUE  TRUE FALSE
```

```r
x1==2 #Igual a, algunos comandos aceptan un = sencillo
```

```
[1] FALSE  TRUE FALSE
```

```r
x1 != 3 #Diferente de
```

```
[1]  TRUE  TRUE FALSE
```

```r
0 %in% x1 #Se encuentra en
```

```
[1] FALSE
```

```r
x2 <- 10
x3 <- 15
x2>5&x3<17
```

```
[1] TRUE
```

```r
any(c(x1,x2,x3)>12)
```

```
[1] TRUE
```

```r
any(c(x1,x2,x3)==12)
```

```
[1] FALSE
```

```r
all(c(x1,x2,x3)>5)
```

```
[1] FALSE
```

```r
all(c(x1,x2,x3)==12)
```

```
[1] FALSE
```

Además de estos operadores, existen un par de condicionales bastante útiles en R que son "if", "else", "case_when" estos indican que sí se da un evento haga una acción (if), y si no se cumplen las condiciones, se lleve a cabo una segunda acción (else), o que se realice una sola acción al encontrar la condición (case_when).

Se puede ver en el siguiente ejemplo:

```r
if (any(c(x1,x2,x3)>12)) {
  print("Aquí hay un valor mayor que 12")
} else {
  print("Aquí no hay valores mayores que 12")
}
```

```
[1] "Aquí hay un valor mayor que 12"
```

En el código anterior, tenemos lo siguiente: se genera un condicional "if" que contiene una operación lógico que consiste en que si existe al menos una variable que cumpla la condición (mayor que 12), nos otorga un cadena texto que dice "Aquí un valor mayor que 12". Si no se cumple la condición, es decir, que en cualquier otro evento, se usa el "else", y nos arrojará la cadena de texto "Aquí no hay valores mayores que 12".

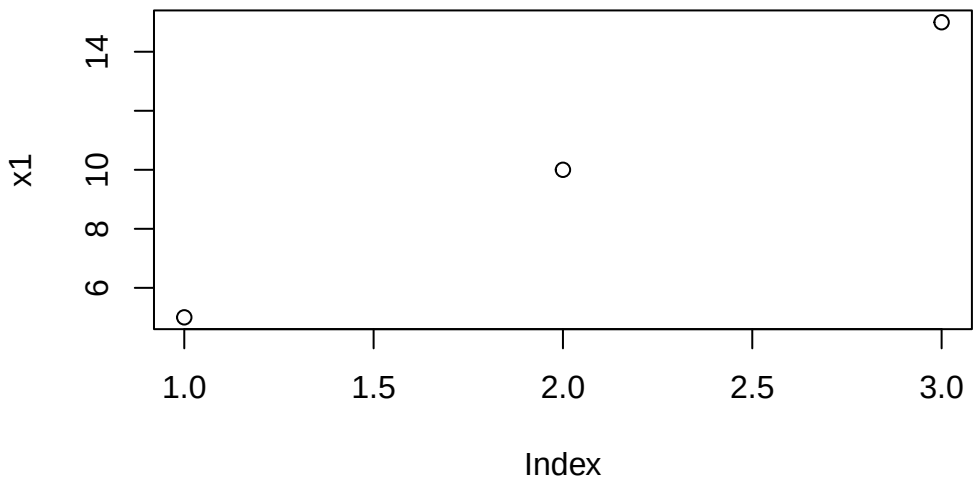
Cabe mencionar que podemos modificar cualquier sección para que se cumplan las características que necesitamos, o que arroje diferentes cadenas de texto. Incluso podemos anidar más "else" con "else if", lo que nos permite hacer diferentes condicionales.

1.5.5 Funciones

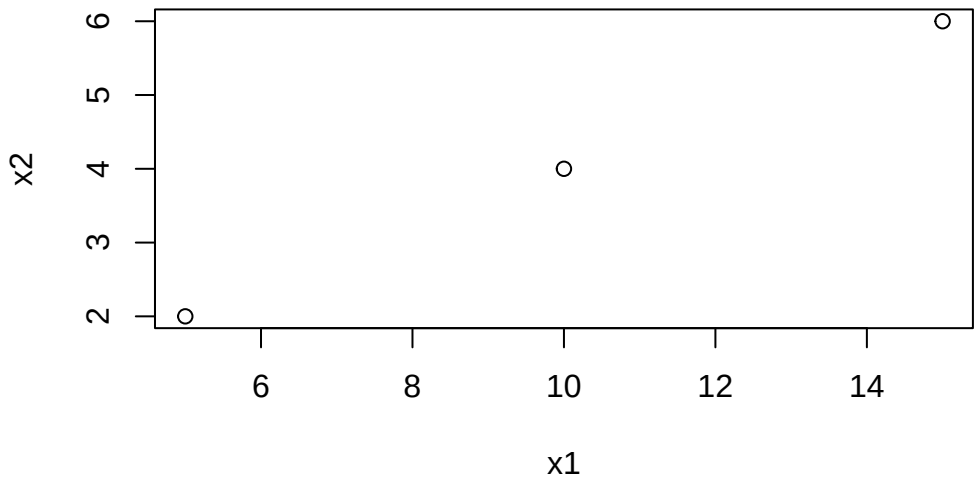
Ya hemos utilizado muchas funciones en R base, como print, if, else, etc. Es importante mencionar la sintaxis de las funciones. Las funciones se encuentran guardadas en un objeto, que habitualmente no se muestran en el ambiente.

Nosotros llamamos una función, se abren paréntesis, y adentro de ellos, introducimos las variables que queremos que trabaje la función. Algunas funciones pueden requerir de una sola variable, otras dos, otras tres, etc. Además, cada función puede presentar argumentos u opciones, es decir, pequeñas modificaciones a nuestra función de base, para ajustar lo que necesitamos. Por ejemplo:

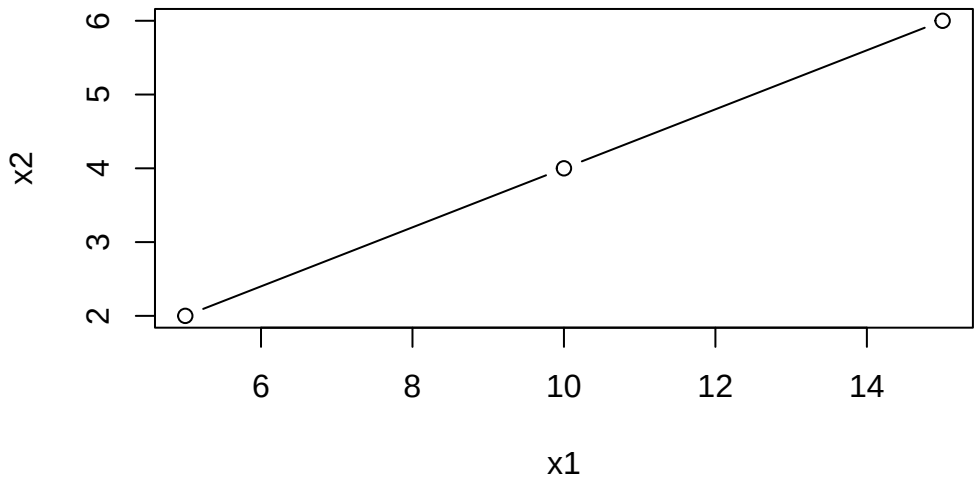
```
x1 <- c(5,10,15)
x2 <- c(2,4,6)
plot(x1) #Una sola variable, genera un gráfico que muestras la posición de los valores de x1
```



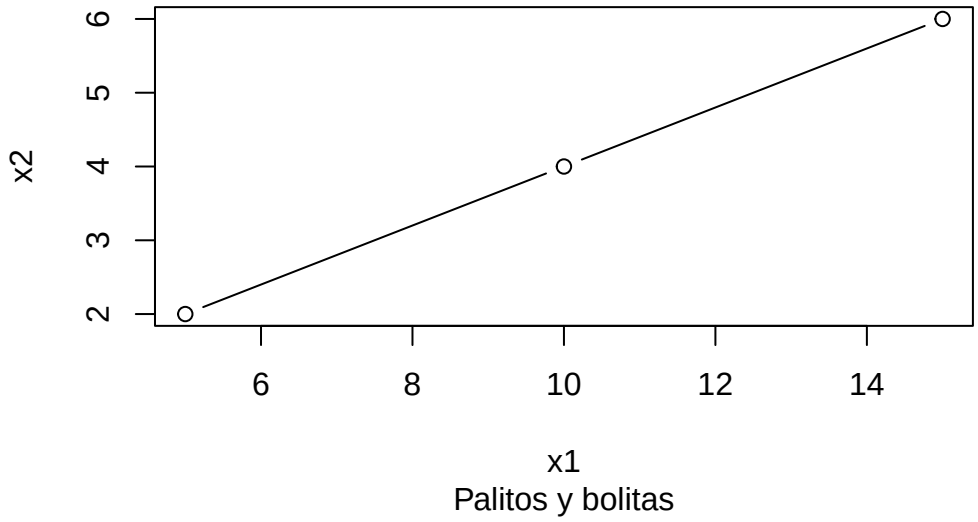

```
plot(x1,x2) #Dos variables, muestra la intersección entre los valores
```




```
plot(x1,x2, type = "b") #Agregamos un argumento, que es "type", le decimos que cambie el esqu
```



```r
plot(x1,x2, type = "b", sub = "Palitos y bolitas") #Agregamos un subtítulo al gráfico, querer
```



Sin embargo, cabe la posibilidad de que deseemos crear nuestras propias funciones para ahorrarnos tiempo, asumamos que queremos crear una función empleando los operadores condicionales mencionados.

```r
#Primero, copiamos y pegamos el condicional
if (any(c(x1,x2,x3)>12)) {
  print("Aquí hay un valor mayor que 12")
} else {
  print("Aquí no hay valores mayores que 12")
}
```

```
[1] "Aquí hay un valor mayor que 12"
```

```r
#Luego lo seleccionamos todo, y oprimimos "Ctrl+Alt+x", o irnos a Code>Extract Function
```

Luego debería de solicitarnos el nombre de la función, yo la llamé “mayor_12”, y debería de aparecer algo así:

```r
mayor_12 <- function(x1, x2, x3) {
  if (any(c(x1,x2,x3)>12)) {
    print("Aquí hay un valor mayor que 12")
  } else {
    print("Aquí no hay valores mayores que 12")
  }
}
```

Si corremos todo el código, debería de haberse creado la función. Ahora, si la llamamos, nos debería de auto completar con el nombre dado, y si corremos, nos debería de arrojar el resultado. Ahora solo restaría modificar los valores de "x1", "x2" y "x3", para ver qué resultados nos da. Lo mismo se puede realizar con el plot "palitos y bolitas".

> Tip
>
> La mayoría de funciones necesarias para lo que realizamos en el día a día suelen venir en alguna librería de base o terceros, es poco probable que tengamos que crearlas nosotros.

1.6 Manejo de Bases de Datos en R

1.6.1 Entrada y salida de datos

La entrada de datos, para un primer uso de R, consiste en el ingreso manual mediante la ventada del "script", como lo hemos manejo al crear variables. Esto es de utilidad cuando queremos hacer algunos pequeños ajustes, crear algunas variables, funciones o gráficas. Sin embargo, para fines de manipulación y trabajo habitual en ciencias de la salud, no es el método más amigable, ni el más eficiente.

Por norma general, siempre hacemos uso de bases de datos creadas en Excel, o posiblemente en Access, si se desea algo más estructurado y con formularios, y raramente en formato .csv o .tsv (a menos que la actividad principal sea analista de datos).

Respecto a la salida de datos, la forma habitual, es utilizando objetos. Es decir, hacemos un análisis, gráfica, transformación, descripción, etc., lo pasamos a un objeto, y de ese objeto lo guardamos en nuestra computadora. Se pueden exportar incluso tablas, o hacer reportes y convertirlos a .pdf, todo este proceso se verá con más detalle en las siguientes secciones.

Otra forma, menos "sofisticada" de sacar datos, es con el clásico copia y pega de lo que nos arroja la consola o en la sección de plots. Podemos hacer clic derecho y seleccionar la opción de copiar, o incluso de "guardar cómo…", pero es mucho más sencillo y rápido si necesitamos solo algunos datos.

1.6.2 Importación y exportación de bases de datos

Respecto a la importación, tenemos dos principales métodos, ya sea usando las líneas de código, o mediante el ambiente. Empecemos con el método del ambiente, ya que es más amigable.

Nos vamos a ambiente, en el ícono que dice "Import Dataset", y seleccionamos el método de preferencia:

- From text (base): para archivos .txt o .tsv
- From text (readr): para archivos .csv. En este método, cabe mencionar que algunos archivos con extensión .csv pueden estar separados por otros carácteres y aún así tener estar extensión. Recordemos que la extensión es una cosa (un método que indica a la computadora una forma de abrir un archivo mediante un programa), mientras que la separación es otra cosa. Por lo que verás que en las opciones para cargar, puedes utilizar diferentes separadores.
- From Excel: sorprendentemente sí, es para archivos de Excel…

Además, para aquellos que trabajan con SPSS, también tiene un importador de archivos con dicha extensión, aunque en lo personal, casi nunca lo uso.

Otro punto importante a mencionar al hablar de importación de datos, es la codificación que emplea cada archivo. Cabe la posibilidad de que al guardar un archivo (p. e. de Excel), y lo guardemos como .csv o .tsv, nos arroje un error que mencione que los símbolos no son compatibles. Para corregir ese y otros errores similares, debemos ir al explorador de archivos de nuestra computadora, ir al archivo en cuestión, se debe hacer clic derecho en el archivo, elegir "Abrir con…" y seleccionamos bloc de notas. Una vez abierta la base de datos con el bloc de notas, nos vamos a "Archivo>Guardar como…" y en la parte inferior donde dice "Codificación", elegimos "UTF-8", y ya deberíamos poder importar el archivo sin problema.

El otro método para importar archivos es mediante el código de script. La "desventaja", es que debemos de conocer la localización de nuestro archivo y teclearla para que la computadora sepa en dónde está y lo pueda cargar. Este inconveniente se arregla mediante el uso de proyectos, y guardando nuestras bases de datos en dicha carpeta. Para cargar a partir del código, se utiliza la siguiente sintaxis:

> **NOTA**: Para descargar la base de datos con la estaremos trabajando, click aquí. Desde este enlace puedes guardar una copia en tu computadora.

```
pacman::p_load(tidyverse)
base_de_datos <- read_csv("base datos.csv")
```

```
Rows: 403 Columns: 1
-- Column specification --------------------------------------------------------
Delimiter: ","
```

```
chr (1): id;chol;glu;hdl;ratio chol-hdl;glyhb;location;age;gender;height;wei...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

En el código anterior, asumimos que ya se tiene cargada la librería "readr". El objeto a crear se llamará "base_de_datos", recordemos que la podemos nombrar como queramos, operador de asignación, función "read_csv" (es para archivos .csv, si fueran .tsv, sería "read_tsv", y si fuera de excel, si, es correcto, sería "read_excel"), y entre comillas ponemos el nombre del archivo con su extensión tal como está escrito en el archivo (es decir, incluyendo acentos, comas, espacios, todo). En caso de que se encuentre en otras carpetas, hacemos la ruta del archivo hasta llegar a el. Por ejemplo, si la base de datos estuviera en la carpeta "images", sería "images/base datos.csv", y listo.

Además, existen varios argumentos para modificar el formato de carga de nuestra base de datos. Podemos omitir la primera fila, usar la primera fila como nombre para las variables, elegir el tipo de dato por columna, etc.

> **Tip**
>
> Si ves que no se acomoda correctamente la tabla, prueba diferentes separadores. En este caso es "semicolon".

Vamos a cargar la base de datos llamada "base datos.csv" usando el método del ambiente, y veamos su contenido con el explorador nativo de R.

Una vez terminado, vamos a guardar la misma base de datos, pero con otro nombre. Podemos guardar la base de datos con formato .csv, .tsv, .xlsx, entre otros.

```
pacman::p_load(writexl)
write.csv(base_de_datos, "BD.csv")
write_xlsx(base_de_datos, "BD.xlsx")
write.table(base_de_datos, "BD.tsv", sep = "\t")
```

> **Tip**
>
> Para fines prácticos, siempre usaremos las siglas "BD", puedes cambiarlo si deseas, pero deberás cambiar todos los códigos.

Hay que notar que la sintaxis es muy similar en todos los casos, solicitamos guardar con la función "write", dependiendo de la extensión, es la parte siguiente (cabe mencionar que hay que cargar la librería "writexl" para guardar archivos de Excel), luego elegimos la base de nuestro ambiente a guarda, luego elegimos un nombre con extensión. Cabe mencionar que

para el caso de .tsv, hay que seleccionar el tipo de separador con el argumento “sep”, y “\t” significa símbolo de tabulador. Estos son los métodos más comunes para exportar archivos y los tipos más empleados, es de mencionarse que se pueden exportar también a SPSS u otros, para más información, clic aquí.