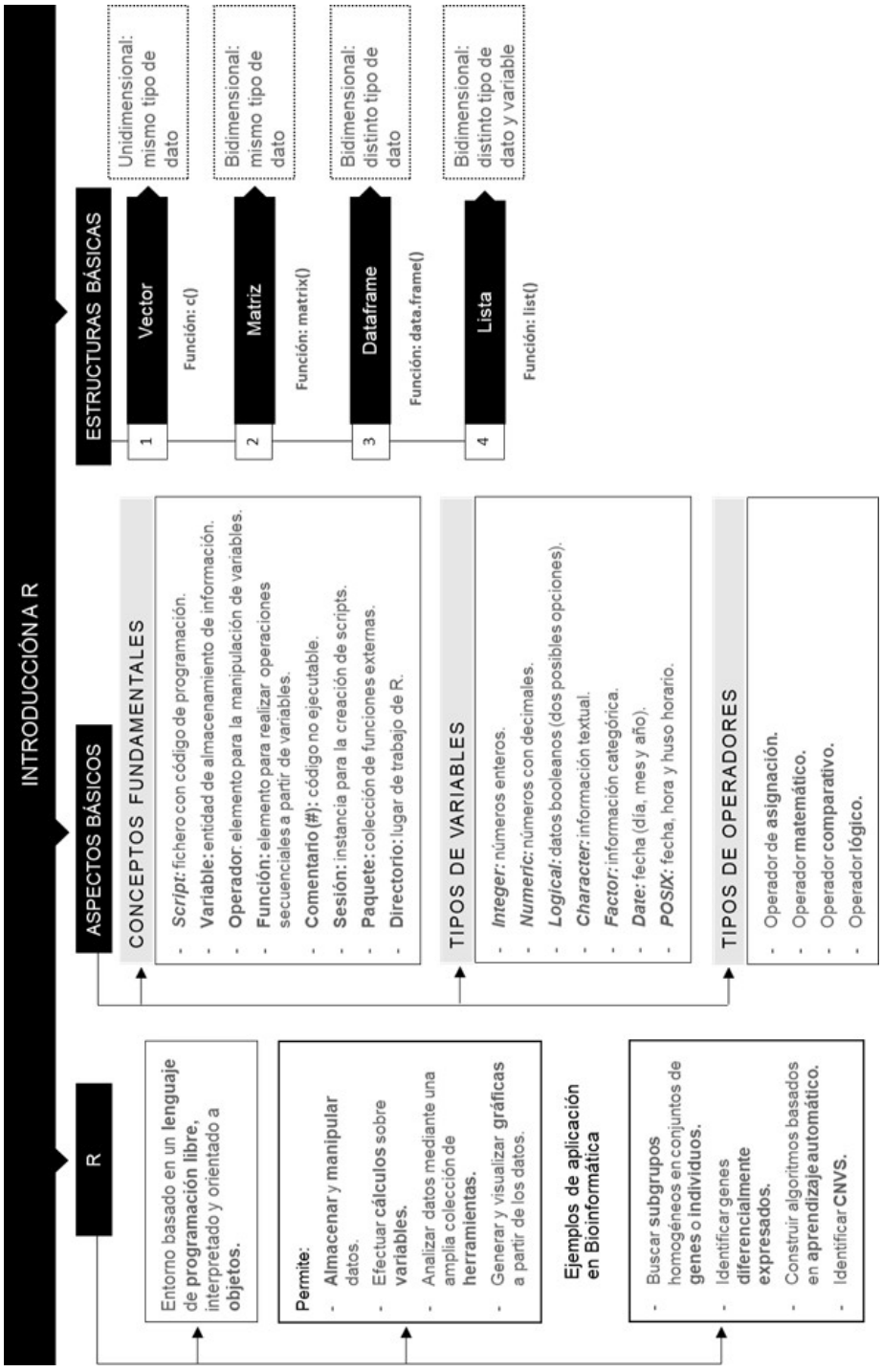
Estadística y R para Ciencias de la Salud

Tema 1. Introducción a R

Índice

Esquema



1.1. Introducción y objetivos

R es un sistema para el desarrollo de análisis estadísticos y gráficos creado por los estadísticos Ross Ihaka y Robert Gentleman en 1996 [1]. R posee una naturaleza doble de **programa** y **lenguaje de programación** y es considerado un dialecto del lenguaje S desarrollado por los Laboratorios AT&T Bell [1]. S está actualmente disponible como el programa S-PLUS comercializado por Insightful [2], existiendo asimismo diferencias importantes en lo que respecta al diseño de R y S [3].

R se encuentra distribuido bajo los términos de la **GNU General Public License,** mientras que su desarrollo y distribución son llevados a cabo por un equipo de estadísticos conocido por el nombre de **Grupo Nuclear de Desarrollo de R** (en inglés, *the R Core Team*) [4]. R se presenta en varias formas: el código fuente escrito en C, esencialmente para máquinas Unix y Linux, o en forma de archivos binarios precompilados para Windows, Linux, Macintosh y Alpha Unix. Los archivos necesarios para instalar R, bien por medio de las fuentes o bien mediante binarios precompilados, se distribuyen a través de **Comprehensive R Archive Network (CRAN)** [5].

R consiste en un conjunto integrado de programas para la manipulación de datos, cálculos y representación de gráficos. Posee, entre otras, las siguientes características [6]:

- Capacidad de almacenamiento y manipulación efectiva de datos.
- Operadores para cálculo sobre variables indexadas *(arrays),* como matrices.
- Una amplia colección de herramientas para análisis de datos.
- Posibilidades gráficas para análisis de datos.
- Un lenguaje de programación bien desarrollado, simple y efectivo, incluyendo

condicionales, ciclos, funciones recursivas y posibilidad de entradas y salidas.

R se define asimismo como un **entorno**, al tratarse de un sistema completamente diseñado y coherente, en el que se hallan implementadas numerosas técnicas estadísticas, algunas incluidas en el entorno base de R y otras disponibles en forma de **bibliotecas** (en inglés, ***packages***). Además, R también puede ser utilizado como un vehículo para el desarrollo de nuevos métodos de análisis interactivo de datos [6].

R, además de poseer diversas funciones para análisis estadísticos, también permite la visualización de gráficos, tanto en su propia ventana como por medio de archivos previamente guardados, a partir de los mismos, en múltiples formatos posibles (jpg, png, bmp, ps, pdf, emf, pictex, xfig…) [7]. Los resultados de análisis estadísticos también se muestran en pantalla, mientras que algunos resultados intermedios (p-valores, coeficientes de regresión, residuales…) pueden ser guardados, exportados a un fichero o usados en procedimientos posteriores [7].

R, asimismo, permite el análisis de conjuntos sucesivos de datos mediante **bucles** (en inglés, ***loops***) y combinar en un solo programa distintas funciones estadísticas para el desarrollo de análisis complejos, entre otras posibles aplicaciones en función de las necesidades del usuario [7].

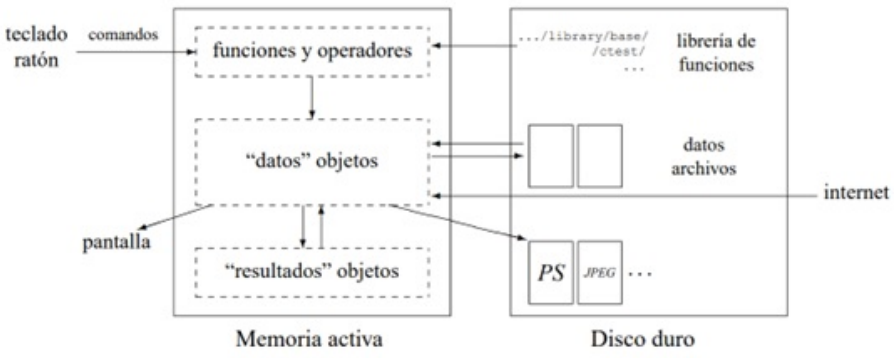


Figura 1. Visión esquemática del funcionamiento de R. Fuente: [7].

En lo que respecta al lenguaje de programación de R, este se trata de **un lenguaje**

orientado a objetos, lo cual es responsable de la flexibilidad y simplicidad de R en comparación con otros lenguajes [7]. R, de hecho, se considera un lenguaje **interpretado**, de manera que los comandos escritos en el teclado son ejecutados sin necesidad de construir ejecutables [7]. Además, la sintaxis de R resulta muy simple e intuitiva.

Orientado a objetos significa que las variables, datos, funciones y resultados, entre otros elementos, se guardan en la memoria activa de la máquina en forma de **objetos** identificados mediante un nombre específico [7]. Estos objetos, a su vez, pueden ser modificados o manipulados por parte del usuario mediante el empleo de **operadores** (aritméticos, lógicos y comparativos) y **funciones** (los cuales son, al mismo tiempo, también objetos) [7]. Las funciones en R siguen esta estructura:

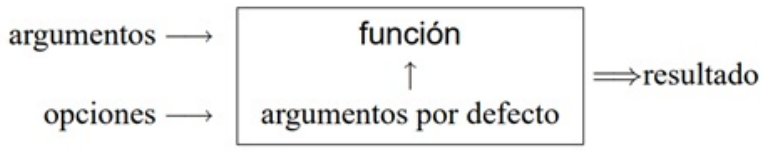


Figura 2. Funcionamiento de una función en R. Fuente: [7].

Donde los argumentos pueden ser objetos bien definidos por defecto en la función o bien modificados por el usuario con opciones [7]. Todas las acciones en R se realizan con objetos guardados en la memoria activa del ordenador, sin usar archivos temporales. La lectura y escritura de archivos solo se realiza para la entrada y salida de datos y resultados. El usuario ejecuta las funciones con la ayuda de comandos definidos. Los resultados se pueden visualizar directamente en la pantalla, guardar en un objeto o escribir directamente en el disco (particularmente para gráficos) [7]. Debido a que los resultados son objetos, pueden ser también considerados como datos y analizados como tal. Todos aquellos archivos que contengan datos pueden ser leídos directamente desde el disco local o por medio de un servidor remoto a través de la red [7].

Ideas clave

Dentro del campo de la **bioinformática**, prácticamente todos los análisis estadísticos pueden ejecutarse en R, siendo este entorno por tanto fundamental en el avance de este campo. Concretamente, en R pueden realizarse análisis basados en *clustering* y sus variantes (*k-means*, modelos mixtos…) o clasificación y discriminación (análisis de discriminantes, árboles de clasificación, *bagging*, empleo de máquinas de vectores de soporte…) gracias a la existencia de un amplio abanico de librerías de R a disposición del usuario.

Por consiguiente, R permite la puesta en marcha de diversas tareas que son comunes en el ámbito de la bioinformática, entre las cuales se podrían destacar las siguientes:

- Búsqueda de subgrupos homogéneos en conjuntos de genes o individuos.
- Identificación de genes diferencialmente expresados (junto con el conveniente ajuste por comparaciones múltiples).
- Construcción de algoritmos basados en aprendizaje automático para la clasificación de pacientes en función de sus perfiles genéticos.
- Identificación de regiones genómicas con alteraciones del número de copias (en inglés, *copy number variations* (CNVs)).

Adicionalmente, cabe destacar que R se trata de **software libre,** permitiendo a los usuarios como indica el propio término, libre acceso al código original, así como su modificación. La bioinformática, de hecho, se fundamenta en el *software libre* y no podría avanzar sin la existencia de este. De hecho, numerosos otros campos científicos serían incapaces de desarrollarse si no fuese por el libre acceso al conocimiento previo; en bioinformática, ocurre lo mismo en lo que respecta al **software libre.**

En este tema, comenzaremos viendo cómo instalar R y demás programas

fundamentales para su apropiado empleo. Posteriormente, se detallará cómo instalar algunas librerías imprescindibles para los análisis estadísticos en los que se profundizará en temas posteriores de esta asignatura, se incidirá en varios aspectos básicos acerca de la interacción del usuario con R y finalizaremos descubriendo los principales tipos de estructuras de datos utilizados en R.

Por tanto, los **objetivos** que se busca cumplir tras la finalización de este tema son:

- Aprender a instalar R y RStudio en función del sistema operativo.
- Comprender el funcionamiento de RStudio para el análisis estadístico de datos.
- Entender las características básicas del lenguaje de programación R.
- Conocer las principales estructuras de datos disponibles en R y su manipulación.
- Aprender a generar informes dinámicos para la presentación de los resultados de un análisis estadístico desarrollado mediante RStudio.

1.2. Instalación de R y RStudio

Vamos a comenzar viendo cómo instalar el entorno **R**. Posteriormente, también se detallará el proceso de descarga e instalación del programa **RStudio**, que permite una interacción más intuitiva entre el usuario y el lenguaje R. Por último, se mostrará cómo instalar **Rtools**, un elemento fundamental para la correcta instalación y creación de librerías de R.

Instalación de R

Para la instalación del entorno **R**, entraremos a **CRAN** (https://cran.r-project.org). En caso de que nuestra intención sea descargar exclusivamente el código fuente, recurriremos a https://cran.r-project.org/sources.html independientemente del sistema operativo. Si, por el contrario, deseamos descargar directamente un ejecutable para la instalación de R, accederemos a https://cran.r-project.org/bin/, seleccionando el sistema operativo que corresponda. Suponiendo que esta es la opción deseada, vamos a detallar brevemente qué hacer para cada sistema operativo.

Windows

En este caso, debemos instalar *base* a partir de https://cran.r-project.org/bin/windows/base/. En esta página, podremos descargar directamente la versión más reciente de R, consultar las preguntas más frecuentes y, en caso de ser necesario, descargar otra versión anterior de R.

Linux

La mayoría de las distribuciones de Linux permiten la instalación de R directamente en la consola. En https://cran.r-project.org/bin/linux/, en función de la distribución, podemos consultar detalladamente cómo realizar la instalación para cada caso.

MacOS

En https://cran.r-project.org/bin/macosx/, podemos consultar qué fichero descargar, relativo a la última versión disponible de R, en función de la versión de macOS que tengamos instalada. En caso de encontrarnos con problemas, podemos consultar las preguntas más frecuentes aquí.

Instalación de RStudio

Para esta asignatura, utilizaremos la versión *desktop* de RStudio, la cual puede ser descargada en este enlace, en el que podemos encontrar el archivo a descargar para nuestro sistema operativo. Nótese que RStudio requiere que R esté previamente instalado en nuestro ordenador, que la versión de éste sea igual o superior a **3.3.0** y que nuestro sistema operativo sea de 64 bits, aunque hay versiones anteriores que también funcionan en un sistema de 32 bits (https://docs.posit.co/previous-versions/). En caso de no saber el tipo de sistema de nuestro ordenador, podemos consultarlo de la siguiente manera para cada sistema operativo:

- Windows
- Linux
- macOS

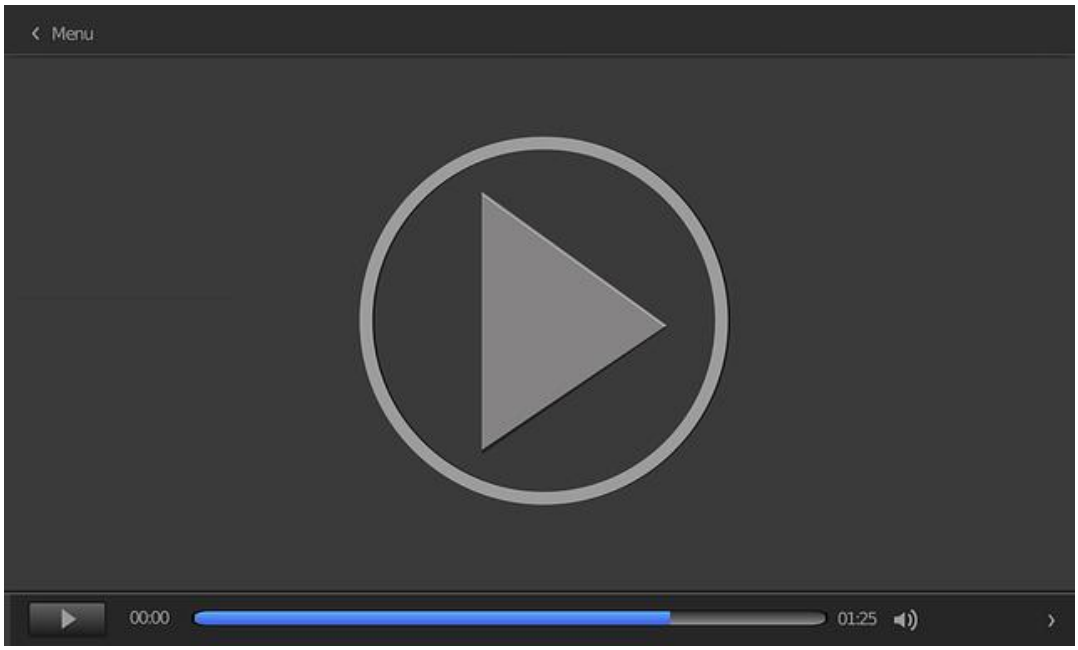
Instalación de Rtools

Solamente en el caso de descargar R y RStudio en Windows, conviene descargar Rtools (https://cran.r-project.org/bin/windows/Rtools/) para poder instalar librerías adecuadamente o, en el caso de necesitarlo en un futuro, desarrollar nuestras propias librerías.

Instalados todos los programas necesarios, ya podemos entrar de lleno en el funcionamiento del lenguaje R y las principales estructuras de datos que podemos construir por medio del mismo.

Profundización

Para ver detalladamente el proceso de instalación del software tratado en este apartado, puedes consultar el vídeo ***Instalación de R y RStudio***:



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=fb106c69-fb98-4fed-95ad-b08900db9d10

1.3. Aspectos básicos

Conceptos fundamentales

Tras haber finalizado la instalación de RStudio, vamos a pasar a conocer algunos elementos básicos que componen R y este programa:

Ejecución de código

R se trata de un **lenguaje interpretado** [7]. Al igual que con otros lenguajes de programación, al trabajar con R se generan archivos de código compuestos por instrucciones que nuestro ordenador va ejecutando progresivamente. No obstante, dado que nuestros procesadores no poseen la capacidad de entender estos lenguajes directamente, es necesario llevar a cabo una **traducción** de este lenguaje al denominado **código máquina** [7].

Esta traducción puede ser realizada de dos maneras [7]:

- Por medio del empleo de un **compilador** que traduce de una sola vez todo el archivo de código a lenguaje de máquina y pasa posteriormente a ejecutarlo.
- A través de un **intérprete** que traduce el archivo de código línea por línea durante el proceso de ejecución: se traduciría primero la línea 1 y luego se ejecutaría, a continuación la 2 y se ejecutaría, y así sucesivamente.

En el caso de RStudio, este incorpora un intérprete de R a través de su consola. La consola de RStudio nos permite introducir las instrucciones que deseemos para que sean traducidas y ejecutadas inmediatamente [8].

Script

Un **script** se corresponde con aquel fichero que contiene código de programación de R. Este fichero, con extensión **.R**, se encontrará guardado en la ubicación que

nosotros definamos o en aquella que utilice RStudio por defecto para guardar los archivos derivados del uso de este programa.

Variable

Las **variables** son entidades fundamentales de cualquier script y sirven para el almacenamiento de información. Las variables pueden ser de distintos tipos, como una tabla, un vector numérico o incluso una imagen. Las variables pueden ser nombradas como queramos, eso sí, considerando que deben empezar siempre con una letra y que solo pueden incluir letras, números, puntos (.) y guion bajo ( ). Además, los nombres de las variables son sensibles a las mayúsculas y minúsculas, de manera que los nombres v1 y V1 se corresponden con dos variables completamente distintas. El contenido de una variable también puede ser modificado simplemente definiéndola de nuevo.

Operador

Los **operadores** son elementos imprescindibles para la manipulación de objetos. Estos operadores también forman parte del código y permiten realizar diversas acciones utilizando las variables que consideremos.

Función

Al igual que los operadores, las **funciones** también permiten la realización de operaciones a partir de variables. Las funciones, sin embargo, permiten la puesta en marcha de una o más tareas de forma secuencial. Estas, además, pueden recibir argumentos de entrada y devolver los resultados oportunos. Las funciones en R presentan la siguiente estructura: `nombre_de_la_función(argumentos)` **.** Por ejemplo, existe la función `median()` , que recibe un conjunto de números y devuelve la mediana.

Las funciones de R, además, pueden tener los siguientes tipos de origen:

- Pueden venir incluidas con la propia instalación de R (base).

- Pueden estar contenidas en librerías externas que debamos descargar e instalar.
- Pueden ser creadas, definidas y programadas por el propio usuario.

Comentario

Los **comentarios** son aquellas partes del código que no queremos que sean ejecutadas por nuestro ordenador. Son utilizados para explicar y ordenar nuestros scripts, de manera que sean más fáciles de entender. En R, una línea de código puede ser comentada mediante el símbolo #.

Sesión

Cada vez que iniciamos R, se crea asimismo una instancia del entorno de este lenguaje de programación. Por un lado, estamos activando el intérprete de R para que este traduzca todas las instrucciones que generemos al lenguaje máquina. Por otro lado, también estamos reservando un espacio específico de la memoria RAM de nuestro ordenador exclusivamente para el alojamiento de las variables y funciones que vayamos definiendo.

Estas instancias que, por tanto, posibilitan nuestra interacción como usuarios con R y la generación de scripts se denominan **sesiones** de R. Un mismo ordenador puede poseer varias sesiones, si bien las variables y funciones que se definan para cada una solo pertenecen a dicha sesión de manera independiente.

Paquete

En numerosas ocasiones se da la situación en la que necesitamos utilizar funciones concretas que no encontramos en R tras simplemente instalarlo. Estas funciones estarán contenidas en **librerías** que tendremos que descargar e instalar por separado.

Las **librerías** o **paquetes**, por tanto, son colecciones de funciones desarrolladas por

una comunidad de programadores y que se ofrecen gratuitamente a todo el público que precise de ellas. La mayoría de estas **librerías** se encuentran en los servidores de **CRAN**, de donde pueden ser descargadas. Sin embargo, muchas otras, sobre todo aquellas relacionadas con tareas bioinformáticas, están disponibles en otro repositorio denominado **BioConductor** [9].

En el caso de necesitar instalar un paquete disponible en CRAN, como por ejemplo *car* y sus dependencias, ejecutaremos lo siguiente en la consola de RStudio, indicando siempre el nombre de la librería **entre comillas**:

```
install.packages("car")
```

Si buscamos instalar más de una librería, esto puede realizarse de la siguiente manera: concatenando los paquetes que sean de nuestro interés mediante C.

```
install.packages(c("gamm4", "varSelRF"))
```

Además, no basta con solo instalar el paquete en caso de necesitar utilizarlo, también debemos importarlo para que las funciones contenidas en este pasen a formar parte de la sesión que tengamos abierta en dicho momento. Para ello, usamos la función `library()`, esta vez con el nombre de la librería **sin comillas**. Por ejemplo:

```
library(car)
```

Directorio de trabajo

Cuando iniciemos una sesión de R y desarrollemos un script, es muy posible que deseemos guardar este o el contenido de alguna variable en una ubicación en concreto de nuestro ordenador. El directorio de trabajo es, efectivamente, la carpeta en la que se encuentran alojados los ficheros de datos, archivos de código y otros documentos que estemos usando.

En caso de no conocer el directorio de trabajo actual utilizado por R, podemos determinarlo mediante la siguiente función sin indicar ningún argumento:

```
getwd()
```

Asimismo, si nos interesa que este directorio se corresponda con otra ubicación de nuestro ordenador, podemos especificar la ruta deseada mediante esta otra función, esta vez indicando esta ruta **entre comillas**:

```
setwd()
```

Proyecto de RStudio

En ciertas ocasiones, por ejemplo cuando estemos trabajando con conjuntos de datos de distinto origen y deseemos realizar análisis con ellos por separado, resulta muy útil la creación de **proyectos** de RStudio para poder trabajar de forma ordenada. Estos proyectos pueden ser alojados en el directorio que elijamos y los archivos contenidos en él pasan a formar parte de dicho proyecto [8].

Entorno de RStudio

RStudio posee una interfaz específica la cual presenta los siguientes paneles o regiones:

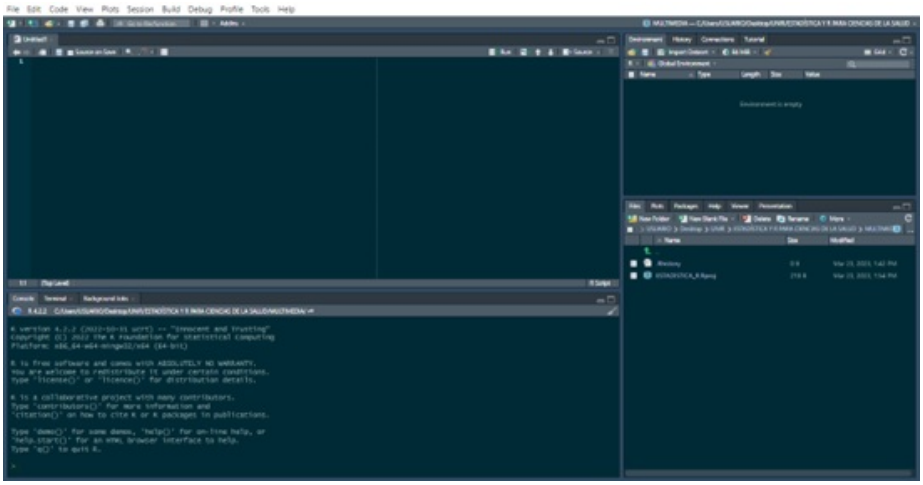


Figura 3. Interfaz de RStudio. Fuente: elaboración propia.

Se pueden observar, además del menú superior, cuatro regiones principales numeradas tal y como se indica en la figura anterior:

- **1**: se trata del **editor**, en el que visualizamos y editamos los scripts.
- **2**: es la **consola**. Es el lugar en el que se interpreta y ejecuta el código R, pudiendo observar además los resultados producto de cada comando.
- **3**: es el **entorno** de **variables**. Encontramos una pestaña denominada Environment, en la que podemos ver todas las funciones y variables definidas en la sesión actual. Además, encontramos otra pestaña denominada History, en la que podemos ver el historial de comandos que hemos ejecutado previamente.
- **4**: es un conjunto de **utilidades** de RStudio. En primer lugar, apreciamos una pestaña llamada Files en la que podemos visualizar todos los ficheros y carpetas presentes en el directorio utilizado por R (el que, si recordamos, podemos conocer por medio de `getwd()`), pudiendo abrirlos sin ninguna limitación. Posteriormente, hallamos la pestaña Plots, que muestra los gráficos que generemos en la consola o en el editor. En la pestaña Packages, encontramos todos los paquetes instalados en nuestro equipo y cuáles han sido importados. Finalmente, en la pestaña Help, podemos hallar la documentación de todas las funciones y paquetes de R, detallándose su modo de empleo, así como ejemplos de uso (esto también podría realizarse en la consola por medio de la función `help()` o **?**; por ejemplo: `help(mean)` o `?mean`).

Tipos de variables

Las variables que definamos en R pueden ser de distintos tipos:

- **Integer**: números enteros (0, 3, -5, 2, -19).
- **Numeric**: números con decimales (2,5, -6,723, 3,03).

- **Logical**: datos booleanos, esto es, con dos posibles valores: *true* o *false*.
- **Character**: información textual.
- **Factor**: información categórica, es decir, que existe un rango de posibles valores (por ejemplo, caso vs control).
- **Date**: fecha (día, mes y año).
- **POSIX**: fecha, hora y huso horario.
- **NA**: valor no disponible.
- **NULL**: elemento vacío.
- **Inf**: infinito.

En ciertos casos, una variable almacena únicamente un solo dato. Por ejemplo, si generamos una variable que contenga el valor 7, esta variable será de tipo **integer**. En otras situaciones, sin embargo, nos interesa que nuestras variables alberguen más de un dato, lo cual es posible mediante variables estructurales.

Operadores

A la hora de manejar las variables, podemos utilizar los operadores para la puesta en marcha de tareas básicas. Los operadores se clasifican de la siguiente manera:

Operadores de asignación

Operador	Definición	Ejemplo
<-	Asigna un valor a una variable.	`x <- 8`
=	Asigna un valor a una variable (es más recomendable usarlo para los argumentos de las funciones).	`median(x = c(3,7,4,1))`

Tabla 1. Operadores de asignación. Fuente: elaboración propia.

Operadores matemáticos

Operador	Definición	Ejemplo	Resultado
+	Suma	4+4	8
-	Resta	11-3	8
*	Producto	4*7	28
/	División	12/4	3
^	Potencia	3^3	27
%%	Resto de dividir un número por el otro	10%%4	2

Tabla 2. Operadores matemáticos. Fuente: elaboración propia.

Operadores comparativos

Este tipo de operadores se utilizan para comprobar si las variables cumplen o no cierta condición que definamos previamente.

Operador	Definición	Ejemplo	Resultado
<	Menor que	3<4	TRUE
<=	Menor o igual que	5<=5	TRUE
>	Mayor que	10>12	FALSE
>=	Mayor o igual que	4.5>=4.6	FALSE
==	Igual que	7==9	FALSE
!=	Distinto que	7!=9	TRUE

Tabla 3. Operadores comparativos. Fuente: elaboración propia.

Operadores lógicos

Operador	Definición	Ejemplo	Resultado
&	Sentencia AND. Ambos elementos deben ser TRUE para que se obtenga TRUE	TRUE & FALSE	FALSE
|	Sentencia OR. Al menos un elemento debe ser TRUE para que se obtenga TRUE	TRUE | FALSE	TRUE
!	Se invierte el dato booleano	!TRUE	FALSE

Tabla 4. Operadores lógicos. Fuente: elaboración propia.

En el apartado **A fondo**, podrás encontrar una guía de **buenas prácticas de programación en R**, recomendables para el correcto manejo de estos operadores y otros elementos introducidos en este apartado.

Lectura y guardado de datos

En algunas ocasiones puede que nos encontremos con datos externos componiendo, por ejemplo, un fichero .txt, que nos interese importar en R. Para ello, podemos utilizar la función `read.table()` .

```
data <- read.table(
   "table-500-text.txt"
)
```

El código anterior, sin embargo, funciona solo si el fichero que deseamos importar se encuentra en el mismo directorio de trabajo utilizado por R. De lo contrario, tendremos que cambiar el directorio usado por R a la ruta donde se halle el fichero de interés (mediante `setwd()`) o indicar la ruta absoluta del mismo:

```
data <- read.table(
   "C:/tmp/table-500-text.txt"
)
```

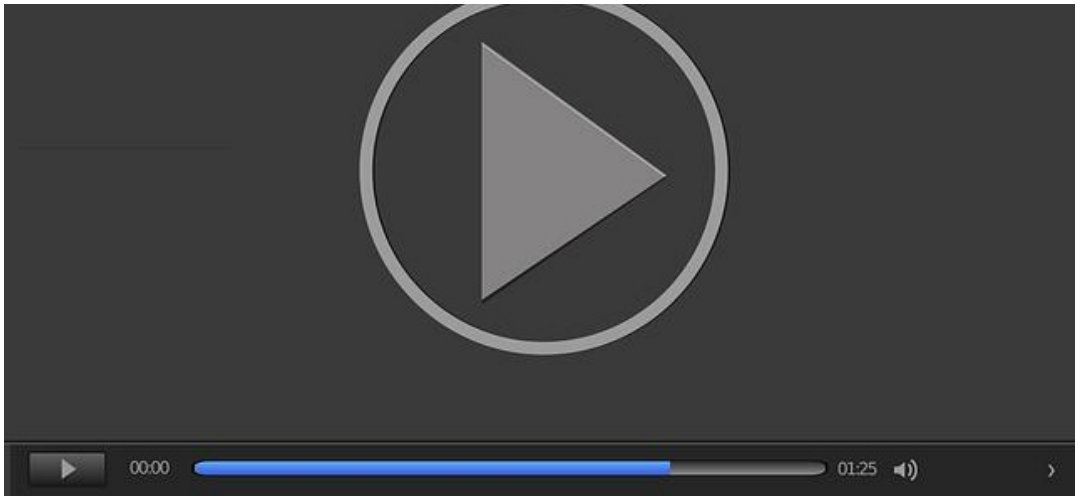
Si, por otro lado, nos interesa guardar datos o resultados que hayamos obtenido en R, podemos guardarlos en el directorio de trabajo utilizado por R mediante la función `write.table()` ¸indicando el nombre con el que queramos guardar dichos datos.

```
write.table(data, file = "saved table.txt")
```

Generación de informes dinámicos

En numerosas ocasiones, puede resultar muy útil presentar nuestros resultados en **informes dinámicos** que podemos construir en RStudio gracias a ficheros **R Markdown**, que usan la extensión ***.Rmd***. Se profundizará en su uso en el vídeo titulado *Generación de informes dinámicos mediante R Markdown*, que puedes consultar a continuación:

Ideas clave



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=76537e40-8a64-4763-8c88-b08900db9cf3

Introducidos los elementos básicos, el entorno, las variables y los operadores de R, vamos a pasar a ver las principales variables estructurales de R.

1.4. Estructuras de datos básicas de R

Las variables estructurales permiten el almacenamiento de conjuntos de datos, pudiendo usar una o más dimensiones, y mezclar distintos tipos de datos en una misma variable. R contiene los siguientes tipos de variables estructurales:

Vectores

Los vectores son colecciones de **una única dimensión** que almacenan datos de un **mismo tipo**.

Definición

Para definir un vector, usamos la letra **c** seguida de los distintos elementos, separados por **comas** entre **paréntesis**.

```
vector_numerico <- c(1, 2, 3, 4)

Vector_de_texto <- c("uno", "dos", "tres")
```

También podemos recurrir a la función `seq()` para generar vectores con una estructura definida:

```
seq(from = 1, to = 10)

# [1] 1 2 3 4 5 6 7 8 9 10

seq(from = 1, to = 10, by = 2)

# [1] 1 3 5 7 9
```

Indexación

Para acceder a una posición o varias posiciones del vector o modificarla/s, usamos los **corchetes**.

```
# Acceso a la primera posición del vector

vector_numerico[1]

# Acceso a las posiciones 2, 3 y 4 del vector

vector_numerico[c(2, 3, 4)]

# Acceso a todas las posiciones del vector excepto la primera

vector_numerico[-1]

# Modificación de la primera posición del vector

vector_numerico[1] <- 2

# Modificación de las posiciones 2, 3 y 4 del vector

vector_numerico[c(2, 3, 4)] <- c(7, 8, 9)
```

Nombrado

Podemos acceder a los nombres de un vector mediante la función `names` (ejemplo: `names(vector_numerico)`).

Concatenación

Mediante la función **c**, podemos unir varias variables obteniendo un nuevo vector, producto de dicha unión (ejemplo: `names(vector_numerico)`).

Factores: un tipo especial de vector

Los factores son necesarios para diferenciar entre vectores de caracteres y vectores de variables categóricas. Por ejemplo, si estamos participando en un estudio donde recojamos el sexo de los participantes, podría interesarnos analizar los datos del estudio en base a esta variable. Para que R sepa que esta es categórica, usamos los factores, empleando la función `factor()` .

```
sex <- factor(c("Female", "Female", "Female", "Male", "Male"))
sex
# # [1] "Female" "Female" "Female" "Male" "Male"
# # Levels: Female Male
```

Matrices

Las matrices son colecciones de datos **bidimensionales** que almacenan datos de un **mismo tipo**.

Definición

Para definir una matriz, usamos la función `matrix()`, indicando el número de filas y columnas, así como el vector utilizado para la construcción.

```
matriz_numerica <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
```

Indexación

Para acceder a una posición o varias posiciones de la matriz, así como modificarla o modificarlas en caso de requerirlo, utilizamos los **corchetes** indicando además la posición de ambas dimensiones **(primero filas y después columnas).**

```
# Acceso al elemento situado en la fila 1 y la columna 2
matriz_numerica[1, 2]

# Acceso a todos los elementos que se encuentran en las columnas 1 y 2, independientemente de su fila

matriz_numerica[, c(1, 2)]

# Modificación del elemento situado en la fila 1 y la columna 2

matriz_numerica[1, 2] <- 2
```

Nombrado

Podemos acceder a los nombres de las filas y de las columnas mediante las

funciones rownames() y colnames() (ejemplos: rownames(matriz_numerica), colnames(matriz_numerica)).

Concatenación

Podemos concatenar distintas matrices mediante las funciones rbind() y cbind() :

- Si deseamos que las matrices se coloquen una encima de la otra, utilizamos rbind() . El número de filas, por tanto, aumenta mientras que el número de columnas se mantiene igual.
- Si, por el contrario, deseamos que las matrices se coloquen una al lado de la otra, utilizamos cbind() . En dicho caso, el número de filas es el que no cambia, mientras que el número de columnas incrementa.

```
# Ejemplo de rbind():

v1 <- c(1, 2, 3, 4, 5)
v2 <- c(11, 12, 13, 14, 15)
rbind(v1, v2)

##    [,1]   [,2]   [,3]   [,4]   [,5]

## V1    1    2    3    4    5

## V2    11    12    13    14    15

# Ejemplo de cbind():

cbind(v1, v2)

##      V1    v2

## [1,] 1     11

## [2,] 2     12

## [3,] 3     13
```

```
## [4,] 4    14

## [5,] 5    15
```

Dataframes

Los dataframes son colecciones de datos **bidimensionales** que almacenan datos de **distinta naturaleza**. Cada columna está definida por un vector de un determinado tipo (*numeric*, *date*, etc.). El número de filas es igual a la longitud de dichos vectores.

Definición

Para definir un dataframe, utilizamos la función `data.frame()`, asignando a cada nombre de columna su vector correspondiente.

```
df <- data.frame(texto = c("trece", "siete", "doce", "cuatro"), numero = c(13, 19, 89, 57))
```

Indexación

Para acceder a una posición o varias posiciones del dataframe, así como modificarla/s en caso de requerirlo, utilizamos los **corchetes,** indicando además la posición de ambas dimensiones **(primero filas y después columnas)**. También podemos acceder a una columna en concreto usando el símbolo **$** (operador en programación).

```
# Acceso al elemento situado en la fila 1 y la columna llamada "texto"
df[1, "texto"]
# Acceso a todos los datos de la columna "numero"
df$numero
# Modificación de los elementos de todas las filas y la columna 1
df[, 1] <- 10
```

Nombrado

Podemos acceder a los nombres de las filas y de las columnas mediante las funciones `rownames()` y `colnames()` (ejemplo: `rownames(df)`, `colnames(df)`).

Concatenación

Podemos concatenar distintos dataframes mediante las funciones `rbind()` y `cbind()`:

- Si deseamos que los dataframes se unan generando nuevas filas, utilizamos `rbind()`. Los dataframes deben tener las mismas columnas.
- Si por el contrario, deseamos que los dataframes se coloquen uno al lado del otro, utilizamos `cbind()`. En dicho caso, los dataframes deben tener las mismas filas.

Listas

Las listas son colecciones de datos **bidimensionales** que admiten **datos y variables de distinta naturaleza**.

Definición

Las listas son definidas mediante la función `list()`, introduciendo los distintos elementos a incluir entre el paréntesis.

```
lista <- list(df, "texto", c(7, 8, 10, 43))
```

Indexación

En el caso de las listas, también podemos acceder a una posición o varias posiciones, así como modificarla, mediante el empleo de **corchetes**, aunque debemos considerar lo siguiente:

- Si buscamos acceder a un elemento concreto de la lista, debemos usar **doble corchete**.
- Si queremos, en cambio, obtener una lista de elementos, debemos utilizar **corchete simple**.

También podemos acceder a un elemento en concreto usando el símbolo **$**.

```
# Acceso a un elemento de la lista
lista[[2]]

# Acceso a una lista que contiene solamente un elemento de la lista original

lista[2]

# Acceso al elemento df de "lista"

lista$df
```

Nombrado

Podemos acceder a los nombres mediante la función `names()` . (Ejemplo: `names(lista)`).

Concatenación

Mediante la función **c**, podemos unir varias listas, obteniendo una nueva lista producto de dicha unión (ejemplo: `c(lista, list(14, "texto", c(3, 9, 19)))`).

La definición de estructuras de datos biomédicos y su manipulación mediante operadores o funciones es fundamental para su análisis estadístico en R.

En el apartado **A fondo**, si te quedas con ganas de más, puedes encontrar el recurso ***Profundización en R y su aplicación en bioinformática***, en donde podrás conocer otras funciones de R y ejemplos de uso en el análisis de datos biológicos.

1.5. Referencias bibliográficas

1. Ihaka R, Gentleman R. R: A Language for Data Analysis and Graphics. J. Comput. Graph. Stat. 1996 sept; 5(3):299.

2. Wikipedia contributors. Wikipedia [Internet]. 2023 jul. 25. S-PLUS [citado 2023 sept. 7]. Disponible en: https://en.wikipedia.org/wiki/S-PLUS

3. Hornik K, et.al. Cran [Internet]. 2023 abr. 5. R faq [citado 2023 sept. 7]. Disponible en: https://cran.r-project.org/doc/FAQ/R-FAQ.html

4. Equipo de traductores al español de GNU. GNU [Internet]. 2023 jun. 1. ¿Qué es GNU? [citado 2023 sept. 7]. Disponible en: https://www.gnu.org/

5. Cran. Cran R Project [Internet]. S. f. The comprehensive R archive network [citado en 2023 sept. 7]. Disponible en: https://cran.r-project.org/

6. Venables WN, Smith DM, et. al. An Introduction to R: notes on R: a programming environment for data analysis and graphics. The R Core Team; 2023. https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

7. Paradis E. R para principiantes. Ahumada JA, traductor. Universidad de Hawaii; 2003. https://cran.r-project.org/doc/contrib/rdebuts_es.pdf

8. RStudio. Posit [Internet]. 2023 ag. 26. RStudio IDE User Guide [citado 2023 sept. 7]. Disponible en: https://docs.posit.co/ide/user/

9. Huber W, Carey VJ, Gentleman R, Anders S, Carlson M, Carvalho BS, et al. Orchestrating high-throughput genomic analysis with Bioconductor. Nat. Methods [Internet]. 2015 en. 29; 12(2):115- 121. Disponible en: https://www.nature.com/articles/nmeth.3252

Buenas prácticas de programación en R

Bernardo I. Towards Data Science [Internet]. 2021 abr. 26. Best Practices for R Programming [citado en 2023 sept. 7]. Disponible en: https://towardsdatascience.com/best-practices-for-r-programming-ec0754010b5a

A la hora de generar un script, es importante seguir unas pautas de programación, recogidas en este recurso, que pueden facilitar su comprensión no solamente por parte de otras personas que lo consulten, sino incluso por nosotros mismos en caso de necesitar evaluarlo o modificarlo en el futuro.

Profundización en R y su aplicación en bioinformática

Cran. Cran R Project [Internet]. S. f. Contributed documentation [citado en 2023 sept. 7]. Disponible en: https://cran.r-project.org/other-docs.html#nenglish

En el recurso anterior podrás encontrar documentación muy útil en caso de querer profundizar más en el lenguaje R y conocer detalles del mismo que sean más avanzados. Para descubrir más detalles a nivel general, podrías consultar «R para principiantes» (Paradis) o «Introducción a R» (R Development Core Team). En cambio, para acceder a una introducción a la aplicación de R en el análisis de datos biológicos mediante el empleo de técnicas estadísticas, también puedes consultar «Introducción al uso y programación del sistema estadístico R» (Ramón Díaz-Uriarte).

Test

1. ¿Cómo se insertan comentarios en R?

A. Mediante el símbolo =.

B. Mediante el símbolo /.

C. Mediante el símbolo #.

D. Mediante el símbolo ?.

2. ¿Cuál de estas opciones NO es un posible tipo de variable en R?

A. Numeric.

B. Float.

C. Date.

D. Integer.

3. ¿Cómo crearías una variable en R cuyo valor fuese 3 elevado a la cuarta potencia?

A. X <- 3 ^ 4 .

B. X <- 3 * 4 .

C. X = 3 / 4 .

D. X <- 3 ^^ 4 .

4. ¿Cuál de estas opciones es la correcta a la hora de generar un vector en R?

A. fruits <- list("banana", "apple", "orange") .

B. fruits <- v("banana", "apple", "orange") .

C. fruits <- c("banana", "apple", "orange") .

D. fruits <- listOf("banana", "apple", "orange") .

5. ¿Cuál de estas funciones es utilizada para la creación de un dataframe en R?

A. Df() .

B. Data.frame() .

C. Dframe() .

D. Dataframe() .

6. ¿Cuál de estas funciones es utilizada para añadir columnas adicionales en una matriz?

A. Add() .

B. Join() .

C. Append item() .

D. Cbind() .

7. ¿Cuál es el resultado del siguiente código?

```
x <- 1:4;y <- 6:9;z <- x + y;z
```

A. 7 9 11 13.

B. 7 9 11 13 14.

C. 9 11 13.

D. Todas son incorrectas.

8. ¿Cuál es el resultado del siguiente código?

```
x <- 1:4;x > 2
```

A. 1 2 3 4 .

B. FALSE FALSE TRUE TRUE .

C. FALSE TRUE TRUE FALSE .

D. 3 4.

9. ¿Cómo crearías la matriz mostrada en la siguiente imagen?

A. Matrix(c(7, 0, 0, 0, 7, 0), 3, 2) .

B. Matrix(c(7, 0, 0, 7, 0, 0), 3, 2) .

C. Matrix(c(7, 0, 0, 7, 0, 0), 2, 3) .

D. Matrix(c(7, 0, 0, 0, 7, 0), 2, 3) .

10. Relaciona cada variable estructural con su descripción.