

Introducción a R y Rstudio

Cristian Diaz

2024-02-16

Contents

1. Introducción a R y RStudio	2
1.1. Descarga e instalación R	2
1.2. Instalar RTools.	3
1.3. Instalar RStudio.	3
1.4. Iniciar RStudio	4
1.5. Instalación y cargue de librerías	5
2. Tipos de objetos	6
2.1. Variables	6
2.2. Vectores	7
2.3. Matrices	8
2.4. Arreglos	9
2.5. Marco de datos o dataframe	10
2.6. Listas	12
3. Conceptos básicos de R-base.	14
3.1. Operadores	14
3.2. Funciones	16
3.3. Crea Una funcion	18
4. Importar y exportar datos	19
4.1. Lectura de datos desde tu PC	19
4.2. Lectura de datos desde tu internet	20
5. Gráficas	20
5.1. Histogramas	21
5.2. Gráficas de barras	21
5.3. Boxplot (diagrama de Caja y Bigotes)	22
5.4. Diagrama de dispersión	25
5.5. Gráficos de sectores	26
6. Bucles y Condicionales	27
6.1. Condicionales	27
6.2. Bucles	29
Referencias Bibliográficas	35

1. Introducción a R y RStudio

¿Que es R?

R, también conocido como **GNU S**, representa un entorno integral y un lenguaje diseñado específicamente para el cálculo estadístico y la creación de gráficos (R Core Team, 2022). Este sistema se basa en un dialecto del elogiado lenguaje S, originario de los Laboratorios Bell y desarrollado por (“Programming with Data,” 1998).

Para aquellos menos familiarizados con este entorno, es importante destacar que R proporciona un acceso relativamente sencillo a una amplia gama de técnicas estadísticas y herramientas gráficas. Además, para usuarios con conocimientos avanzados, R brinda un lenguaje de programación completo que permite la incorporación de nuevas técnicas mediante la definición de funciones personalizadas.

Resulta fundamental señalar que S ha dejado una marca perdurable al transformar la manera en que las personas abordan el análisis, la visualización y la manipulación de datos, siendo reconocido con el premio Association of Computer Machinery Software System en 1998, otorgado a John Chambers.

En la actualidad, S y R se posicionan como los dos lenguajes más destacados y ampliamente utilizados en la investigación en estadística.

¿Que es Rstudio?

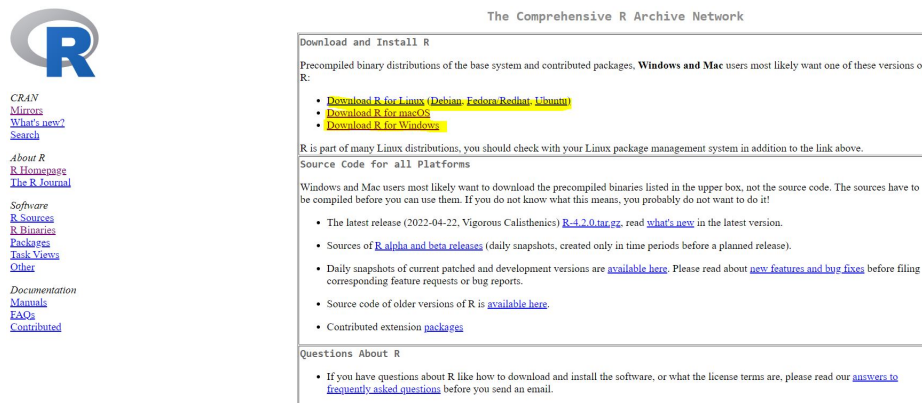
RStudio se presenta como un programa que proporciona una interfaz más amigable para interactuar con R, simplificando numerosas tareas de programación y análisis de datos en este entorno. En esencia, RStudio se clasifica como una Interfaz Gráfica de Usuario (GUI).

Desde una perspectiva más técnica, es esencial destacar que RStudio va más allá de ser simplemente una GUI; se configura como un Entorno de Desarrollo Integrado para R, denominado IDE por sus siglas en inglés (‘integrated development environment’).

Hadley Wickham, actual científico jefe de RStudio, se destaca como una figura prominente en el ámbito. Puedes obtener más información sobre él en su sitio web personal: Hadley Wickham, reconocido como uno de los desarrolladores más prolíficos de paquetes para R, ha sido el creador de un innovador enfoque de programación y análisis de datos en R conocido como **tidyverse**. Este enfoque ha dejado una marca significativa en la forma en que se aborda y realiza el análisis de datos en R.

1.1. Descarga e instalación R

Para instalar R en nuestro ordenador, vamos a la página web (<https://cran.r-project.org/>). Para descargar R, hacemos clic en **Download** del sistema operativo de nuestro ordenador, seleccionamos la opción adecuada.



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!

- The latest release (2022-04-22, Vigorous Calisthenics) [R 4.2.0](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#)
- Contributed extension [packages](#)

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

Al hacer clic sobre **Download R for Windows** iremos a la página que se reproduce más abajo. Hacemos clic sobre **install R for the first time**



CRAN
[Mirrors](#)
[What's new?](#)
[Search](#)

About R
[R Homepage](#)
[The R Journal](#)

Software
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Task Views](#)
[Other](#)

Documentation
[Manuals](#)
[FAQs](#)
[Contributed](#)

R for Windows

Subdirectories:

[base](#) Binaries for base distribution. This is what you want to [install R for the first time](#)
[contrib](#) Binaries of contributed CRAN packages (for R >= 3.4.x).
[old-contrib](#) Binaries of contributed CRAN packages for outdated versions of R (for R < 3.4.x).
[Rtools](#) Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the [R FAQ](#) and [R for Windows FAQ](#).

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

Por ultimo descargaremos la versión mas actualizada **Download R-x.x.x for xxxx**



CRAN
[Mirrors](#)
[What's new?](#)
[Search](#)
[CRAN Team](#)

About R
[R Homepage](#)
[The R Journal](#)

Software
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Task Views](#)
[Other](#)

Documentation
[Manuals](#)
[FAQs](#)
[Contributed](#)

Donations
[Donate](#)

R-4.3.2 for Windows

[Download R-4.3.2 for Windows](#) (79 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

- 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: 2023-10-31

1.2. Instalar RTools.

RTools es un conjunto de herramientas que se utiliza con el lenguaje de programación R para facilitar el desarrollo y la compilación de paquetes y programas en R que incluyen código C y C++. Estas herramientas son esenciales para construir paquetes R que contienen funciones escritas en estos lenguajes de programación de bajo nivel, el cual podemos descargarlo de la siguiente pagina web (<https://cran.r-project.org/bin/windows/Rtools/>)

RTools: Toolchains for building R and R packages from source on Windows

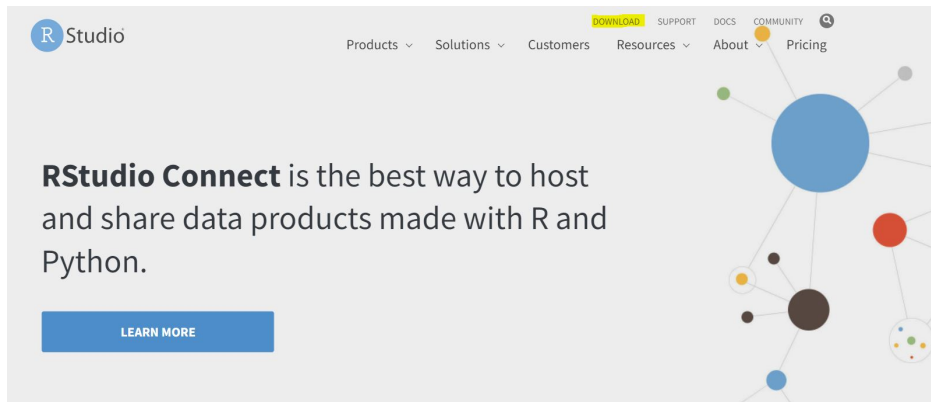
Choose your version of Rtools:

[RTools 4.3](#) for R versions from 4.3.0 (R-release and R-devel)
[RTools 4.2](#) for R versions 4.2.x (R-oldrelease)
[RTools 4.0](#) for R from version 4.0.0 to 4.1.3
[old versions of RTools](#) for R versions prior to 4.0.0

Debemos seleccionar la misma versión de Rtools que instalamos en R, siempre seleccionando la ultima

1.3. Instalar RStudio.

Una vez que hemos instalado R, descargamos RStudio desde AQUÍ (<https://www.rstudio.com/>), Al hacer clic sobre **Download**



Luego seleccionamos el sistema operativo de nuestro ordenador

RStudio Desktop 2022.02.2+485 - [Release Notes](#)

1. Install R. [RStudio requires R 3.3.0+](#)
2. Download RStudio Desktop. [Recommended for your system:](#)



Requires Windows 10/11 (64-bit)



All Installers

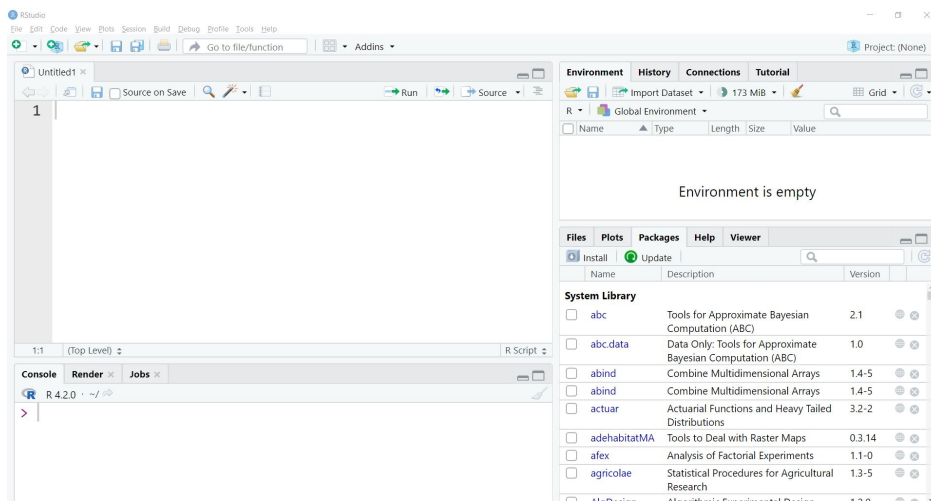
Linux users may need to [import RStudio's public code-signing key](#) prior to installation, depending on the operating system's security policy.

RStudio requires a 64-bit operating system. If you are on a 32 bit system, you can use an [older version of RStudio](#).

OS	Download	Size	SHA-256
Windows 10/11	RStudio-2022.02.2-485.exe	177.27 MB	74187a33
macOS 10.15+	RStudio-2022.02.2-485.dmg	217.09 MB	cda82e98
Ubuntu 18+/Debian 10+	rstudio-2022.02.2-485-amd64.deb	128.58 MB	508a6e9c
Fedora 19/Red Hat 7	rstudio-2022.02.2-485-x86_64.rpm	144.66 MB	7400234c
Fedora 34/Red Hat 8	rstudio-2022.02.2-485-x86_64.rpm	144.70 MB	ad00e2c5

1.4. Iniciar RStudio

En general, trabajamos con la interfaz de RStudio antes que con la de R porque la primera es “más amigable”. Para iniciar RStudio, hacemos clic en el icono de RStudio: Al abrir RStudio deberíamos ver algo parecido a:



RStudio está (normalmente) dividido en 4 paneles: Script Consola Entorno e historia Panel misceláneo: para visualizar ficheros, gráficos, paquetes, etc. Por defecto, la consola se encuentra en el panel izquierdo. Primero aparece un texto informativo y después el prompt del sistema (“>”). ¿Vemos el cursor intermitente? Aquí es donde R espera que le demos instrucciones. Para ejecutar las instrucciones y obtener el resultado pulsamos Enter.

Trabajar en la Consola es muy limitado ya que en la Consola las instrucciones generalmente se escriben y ejecutan una por una. Lo habitual es trabajar con scripts o ficheros de instrucciones. Estos ficheros tienen la extensión.R . Para crear un script, seleccionamos File > New File > R Script

Ahora el panel del script se sitúa en la parte superior-izquierda de RStudio y la Consola en el panel inferior-izquierdo. Por defecto, el nombre del nuevo script es ”Untitled1”.

1.5. Instalación y cargue de librerías

¿Qué son los paquetes en R? Un paquete son muchos archivos juntos en un mismo lugar, como funciones en R, código o datos. De esta forma, Los paquetes permiten hacer más cosas con R. Por lo general, los paquetes están guardados o alojados en repositorios en internet.

¿Qué son los repositorios? Los repositorios son lugares diseñados para guardar código de tal forma que puedan ser consultados por nosotros.

En el caso de R es común que escuches repositorios como CRAN, github o bioconductor (este último para paquetes relacionados a la bioinformática)

En esta entrada vamos a utilizar CRAN por ser el repositorio oficial de R. Los paquetes de CRAN están probados para funcionar sin problemas en la mayoría de versiones de R. Así que no tendremos ningún problema

¿Cómo instalar paquetes en R? Para instalar paquetes, podemos utilizar la consola de R o la interfaz de RStudio.

Instalar paquetes en R con la consola de R La manera más rápida de instalar un paquete de CRAN, es utilizando el comando.

```
install.packages("nombre_del_paquete")
```

Esta función de R, va al repositorio, descarga e instala todo con una línea de código.

Ahora tener instalados los paquetes no significa que ya podamos utilizarlos, primero tenemos que cargarlos en nuestra sesión de R

¿Cómo cargar instalados paquetes en R? Para «activar» o utilizar nuestros paquetes podemos ejecutar el comando

```
library("nombre_del_paquete")
```

Para esta practica instala los siguientes paquetes

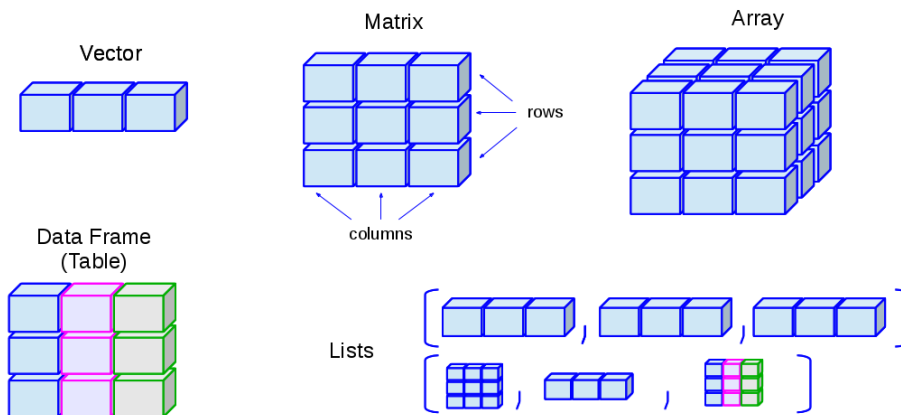
```
install.packages("DT")
install.packages("kableExtra")
install.packages("modeest")
install.packages("psych")
install.packages("readr")
install.packages("AER")
```

Para esta practica carga los siguientes paquetes

```
library("DT")
library("kableExtra")
library("modeest")
library("psych")
library("readr")
```

2. Tipos de objetos

En R existen varios tipos de objetos que permiten que el usuario pueda almacenar la información para realizar procedimientos estadísticos y gráficos. Los principales objetos en R son vectores, matrices, arreglos, marcos de datos y listas. A continuación se presentan las características de estos objetos y la forma para crearlos.



2.1. Variables

Las variables sirven para almacenar un valor que luego vamos a utilizar en algún procedimiento.

Para hacer la asignación de un valor a alguna variable se utiliza el operador `<-` entre el valor y el nombre de la variable. A continuación un **Ejemplo** sencillo.

```
x <- 5
2 * x + 3
```

```
## [1] 13
```

En el siguiente **Ejemplo** se crea la variable `pais` y se almacena el nombre Colombia, luego se averigua el número de caracteres de la variable `pais`.

```
pais <- "Colombia"
nchar(pais)
```

```
## [1] 8
```

2.2. Vectores

Los vectores son arreglos ordenados en los cuales se puede almacenar información de tipo numérico (variable cuantitativa), alfanumérico (variable cualitativa) o lógico (TRUE o FALSE), pero no mezclas de éstos. La función de R para crear un vector es `c()` y que significa concatenar; dentro de los paréntesis de esta función se ubica la información a almacenar. Una vez construido el vector se acostumbra a etiquetarlo con un nombre corto y representativo de la información que almacena, la asignación se hace por medio del operador `<-` entre el nombre y el vector.

A continuación se presenta un **Ejemplo** de cómo crear tres vectores que contienen las respuestas de cinco personas a tres preguntas que se les realizaron.

```
edad <- c(15, 19, 13, NA, 20)
deporte <- c(TRUE, TRUE, NA, FALSE, TRUE)
comic_fav <- c(NA, 'Superman', 'Batman', NA, 'Batman')
```

El vector `edad` es un vector cuantitativo y contiene las edades de las 5 personas. En la cuarta posición del vector se colocó el símbolo `NA` que significa **Not Available** debido a que no se registró la edad para esa persona. Al hacer una asignación se acostumbra a dejar un espacio antes y después del operador `<-` de asignación. El segundo vector es llamado `deporte` y es un vector lógico que almacena las respuestas a la pregunta de si la persona practica deporte, nuevamente aquí hay un `NA` para la tercera persona. El último vector `comic_fav` contiene la información del cómic favorito de cada persona, como esta variable es cualitativa es necesario usar las comillas `' '` para encerrar las respuestas.

Cuando se usa `NA` para representar una información **Not Available** no se deben usar comillas.

Es posible usar comillas sencillas `'foo'` o comillas dobles `"foo"` para ingresar valores de una variable cualitativa.

Si se desea ver lo que está almacenado en cada uno de estos vectores, se debe escribir en la consola de R el nombre de uno de los objetos y luego se presiona la tecla **enter** o **intro**, al realizar esto lo que se obtiene se muestra a continuación.

```
edad
deporte
comic_fav
```

Una variable es un vector de longitud uno.

2.2.1. ¿Cómo extraer elementos de un vector?

Para extraer un elemento almacenado dentro un vector se usan los corchetes `[]` y dentro de ellos la posición o posiciones que interesan.

Ejemplo Si queremos extraer la edad de la tercera persona escribimos el nombre del vector y luego `[3]` para indicar la tercera posición de `edad`, a continuación el código.

```
edad[3]
```

```
## [1] 13
```

Si queremos conocer el cómic favorito de la segunda y quinta persona, escribimos el nombre del vector y luego, dentro de los corchetes, escribimos otro vector con las posiciones 2 y 5 que nos interesan así `[c(2, 5)]`, a continuación el código.

```
comic_fav[c(2, 5)]
```

```
## [1] "Superman" "Batman"
```

Si nos interesan las respuestas de la práctica de deporte, excepto la de la persona 3, usamos [-3] luego del nombre del vector para obtener todo, excepto la tercera posición.

```
deporte[-3]
```

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

Si desea extraer varias posiciones de un vector NUNCA escriba esto: `mivector[2, 5, 7]`. Tiene que crear un vector con las posiciones y luego colocarlo dentro de los corchetes así: `mivector[c(2, 5, 7)]`

2.3. Matrices

Las matrices son arreglos rectangulares de filas y columnas con información numérica, alfanumérica o lógica. Para construir una matriz se usa la función `matrix()`. Por **Ejemplo**, para crear una matriz de 4 filas y 5 columnas (de dimensión 4×5) con los primeros 20 números positivos se escribe el código siguiente en la consola.

```
mimatriz <- matrix(data=1:20, nrow=4, ncol=5, byrow=FALSE)
```

El argumento `data` de la función sirve para indicar los datos que se van a almacenar en la matriz, los argumentos `nrow` y `ncol` sirven para definir la dimensión de la matriz y por último el argumento `byrow` sirve para indicar si la información contenida en `data` se debe ingresar por filas o no. Para observar lo que quedó almacenado en el objeto `mimatriz` se escribe en la consola el nombre del objeto seguido de la tecla **enter** o **intro**.

```
mimatriz
```

```
##      [,1] [,2] [,3] [,4] [,5]  
## [1,]    1    5    9   13   17  
## [2,]    2    6   10   14   18  
## [3,]    3    7   11   15   19  
## [4,]    4    8   12   16   20
```

2.3.1. ¿Cómo extraer elementos de una matriz?

Al igual que en el caso de los vectores, para extraer elementos almacenados dentro de una matriz se usan los corchetes [,] y dentro, separado por una coma, el número de fila(s) y el número de columna(s) que nos interesan.

Ejemplo

Si queremos extraer el valor almacenado en la fila 3 y columna 4 usamos el siguiente código.

```
mimatriz[3, 4]
```

```
## [1] 15
```

Si queremos recuperar **toda** la fila 2 usamos el siguiente código.

```
mimatriz[2, ] # No se escribe nada luego de la coma
```

```
## [1]  2  6 10 14 18
```

Si queremos recuperar **toda** la columna 5 usamos el siguiente código.

```
mimatriz[, 5] # No se escribe nada antes de la coma
```

```
## [1] 17 18 19 20
```


Si queremos recuperar la matriz original sin las columnas 2 y 4 usamos el siguiente código.

```
mimatriz[, -c(2, 4)] # Las columnas como vector
```

```
##      [,1] [,2] [,3]
## [1,]    1    9   17
## [2,]    2   10   18
## [3,]    3   11   19
## [4,]    4   12   20
```

Si queremos recuperar la matriz original sin la fila 1 ni columna 3 usamos el siguiente código.

```
mimatriz[-1, -3] # Signo de menos para eliminar
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    6   14   18
## [2,]    3    7   15   19
## [3,]    4    8   16   20
```

2.4. Arreglos

Un arreglo es una matriz de varias dimensiones con información numérica, alfanumérica o lógica. Para construir una arreglo se usa la función `array()`. Por **Ejemplo**, para crear un arreglo de $3 \times 4 \times 2$ con las primeras 24 letras minúsculas del alfabeto se escribe el siguiente código.

```
miarray <- array(data=letters[1:24], dim=c(3, 4, 2))
```

El argumento `data` de la función sirve para indicar los datos que se van a almacenar en el arreglo y el argumento `dim` sirve para indicar las dimensiones del arreglo. Para observar lo que quedó almacenado en el objeto `miarray` se escribe en la consola lo siguiente.

```
miarray
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,] "a"  "d"  "g"  "j"
## [2,] "b"  "e"  "h"  "k"
## [3,] "c"  "f"  "i"  "l"
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,] "m"  "p"  "s"  "v"
## [2,] "n"  "q"  "t"  "w"
## [3,] "o"  "r"  "u"  "x"
```

2.4.1. ¿Cómo extraer elementos de un arreglo?

Para recuperar elementos almacenados en un arreglo se usan también corchetes, y dentro de los corchetes, las coordenadas del objeto de interés.

Ejemplo

Si queremos extraer la letra almacenada en la fila 1 y columna 3 de la segunda capa de `miarray` usamos el siguiente código.

```
miarray[1, 3, 2] # El orden es importante
```

```
## [1] "s"
```

Si queremos extraer la segunda capa completa usamos el siguiente código.

```
miarray[, , 2] # No se coloca nada en las primeras posiciones
```

```
##      [,1] [,2] [,3] [,4]
## [1,] "m"  "p"  "s"  "v"
## [2,] "n"  "q"  "t"  "w"
## [3,] "o"  "r"  "u"  "x"
```

Si queremos extraer la tercera columna de todas las capas usamos el siguiente código.

```
miarray[, 3,] # No se coloca nada en las primeras posiciones
```

```
##      [,1] [,2]
## [1,] "g"  "s"
## [2,] "h"  "t"
## [3,] "i"  "u"
```

2.5. Marco de datos o dataframe

El marco de datos o *data frame* es uno de los objetos más utilizados porque permite agrupar vectores con información de diferente tipo (numérica, alfanumérica o lógica) en un mismo objeto, la única restricción es que los vectores deben tener la misma longitud. Para crear un marco de datos se usa la función `data.frame()`, como **Ejemplo** vamos a crear un marco de datos con los vectores `edad`, `deporte` y `comic_fav` definidos anteriormente.

```
mimarco <- data.frame(edad, deporte, comic_fav)
```

Una vez creado el objeto `mimarco` podemos ver el objeto escribiendo su nombre en la consola, a continuación se muestra lo que se obtiene.

```
mimarco

##   edad deporte comic_fav
## 1   15    TRUE    <NA>
## 2   19    TRUE  Superman
## 3   13     NA    Batman
## 4   NA   FALSE    <NA>
## 5   20    TRUE    Batman
```

De la salida anterior vemos que el marco de datos tiene 3 variables (columnas) cuyos nombres coinciden con los nombres de los vectores creados anteriormente, los números consecutivos al lado izquierdo son sólo de referencia y permiten identificar la información para cada persona en la base de datos.

2.5.1. ¿Cómo extraer elementos de un marco de datos?

Para recuperar las variables (columnas) almacenadas en un marco de datos se puede usar el operador `$`, corchetes simples `[]` o corchetes dobles `[][]`. A continuación algunos **Ejemplos** para entender las diferencias entre estas opciones.

Ejemplo

Si queremos extraer la variable `deporte` del marco de datos `mimarco` como un vector usamos el siguiente código.

```
mimarco$deporte # Se recomienda si el nombre es corto
```

```
## [1] TRUE TRUE NA FALSE TRUE
```

Otra forma de recuperar la variable `deporte` como vector es indicando el número de la columna donde se encuentra la variable.

```
mimarco[, 2] # Se recomienda si recordamos su ubicación
```

```
## [1] TRUE TRUE NA FALSE TRUE
```

Otra forma de extraer la variable `deporte` como vector es usando `[[]]` y dentro el nombre de la variable.

```
mimarco[["deporte"]]
```

```
## [1] TRUE TRUE NA FALSE TRUE
```

Si usamos `mimarco["deporte"]` el resultado es la variable `deporte` pero en forma de marco de datos, no en forma vectorial.

```
mimarco["deporte"]
```

```
##  deporte
## 1    TRUE
## 2    TRUE
## 3     NA
## 4  FALSE
## 5    TRUE
```

Si queremos extraer un marco de datos sólo con las variables `deporte` y `edad` podemos usar el siguiente código.

```
mimarco[c("deporte", "edad")]
```

```
##  deporte edad
## 1    TRUE  15
## 2    TRUE  19
## 3     NA   13
## 4  FALSE  NA
## 5    TRUE  20
```

Por otra, si queremos la `edad` de las personas que están en las posiciones 2 hasta 4 usamos el siguiente código.

```
mimarco[2:4, 1]
```

```
## [1] 19 13 NA
```

2.5.2. ¿Cómo extraer subconjuntos de un marco de datos?

Para extraer partes de un marco de datos se puede utilizar la función `subset(x, subset, select)`. El parámetro `x` sirve para indicar el marco de datos original, el parámetro `subset` sirve para colocar la condición y el parámetro `select` sirve para quedarnos sólo con algunas de las variables del marco de datos. A continuación varios **Ejemplos** de la función `subset` para ver su utilidad.

Ejemplo

Si queremos el marco de datos `mimarco` sólo con las personas que SI practican deporte usamos el siguiente código.

```
subset(mimarco, subset=deporte == TRUE)
```

```
##  edad deporte comic_fav
## 1   15    TRUE      <NA>
## 2   19    TRUE  Superman
## 5   20    TRUE   Batman
```

Si queremos el marco de datos `mimarco` sólo con las personas mayores o iguales a 17 años usamos el siguiente código.

```
subset(mimarco, subset=edad >= 17)
```

```
##   edad deporte comic_fav
## 2   19     TRUE  Superman
## 5   20     TRUE   Batman
```

Si queremos el submarco con deporte y comic de las personas menores de 20 años usamos el siguiente código.

```
subset(mimarco, subset=edad < 20, select=c('deporte', 'comic_fav'))
```

```
##   deporte comic_fav
## 1     TRUE      <NA>
## 2     TRUE  Superman
## 3      NA   Batman
```

Si queremos el marco de datos mimarco sólo con las personas menores de 20 años y que SI practican deporte usamos el siguiente código.

```
subset(mimarco, subset=edad < 20 & deporte == TRUE)
```

```
##   edad deporte comic_fav
## 1   15     TRUE      <NA>
## 2   19     TRUE  Superman
```

Ejemplo

Leer la base de datos medidas del cuerpo disponible en este enlace https://raw.githubusercontent.com/fhernanb/datos/master/medidas_cuerpo. Extraer de esta base de datos una sub-base o subconjunto que contenga sólo la edad, peso, altura y sexo de aquellos que miden más de 185 cm y pesan más de 80 kg.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/medidas_cuerpo'
dt1 <- read.table(url, header=T)
dim(dt1) # Para conocer la dimensión de la base original
```

```
## [1] 36 6
```

```
dt2 <- subset(x=dt1, subset=altura > 185 & peso > 80,
              select=c('sexo', 'edad', 'peso', 'altura'))
dt2 # Para mostrar la base de datos final
```

```
##      sexo edad peso altura
## 1  Hombre  43  87.3  188.0
## 6  Hombre  33  85.9  188.0
## 15 Hombre  30  98.2  190.5
```

Al almacenar la nueva base de datos en el objeto dt2 se puede manipular este nuevo objeto para realizar los análisis de interés.

2.6. Listas

Las listas son otro tipo de objeto muy usado para almacenar objetos de diferente tipo. La instrucción para crear una lista es `list()`. A continuación vamos a crear una lista que contiene tres objetos: un vector con 5 números aleatorios llamado `mivector`, una matriz de dimensión 6×2 con los primeros doce números enteros positivos llamada `matriz2` y el tercer objeto será el marco de datos `mimarco` creado en el apartado anterior. Las instrucciones para crear la lista requerida se muestran a continuación.

```
set.seed(12345)
mivector <- runif(n=5)
matriz2 <- matrix(data=1:12, ncol=6)
milista <- list(E1=mivector, E2=matriz2, E3=mimarco)
```

La función `set.seed` de la línea número 1 sirve para fijar la semilla de tal manera que los números aleatorios generados en la segunda línea con la función `runif` sean siempre los mismos. En la última línea del código anterior se construye la lista, dentro de la función `list` se colocan los tres objetos `mivector`, `matriz2` y `mimarco`. Es posible colocarle un nombre especial a cada uno de los elementos de la lista, en este **Ejemplo** se colocaron los nombres `E1`, `E2` y `E3` para cada uno de los tres elementos. Para observar lo que quedó almacenado en la lista se escribe `milista` en la consola y el resultado se muestra a continuación.

```
milista

## $E1
## [1] 0.7209039 0.8757732 0.7609823 0.8861246 0.4564810
##
## $E2
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    5    7    9   11
## [2,]    2    4    6    8   10   12
##
## $E3
##   edad deporte comic_fav
## 1   15     TRUE      <NA>
## 2   19     TRUE  Superman
## 3   13      NA    Batman
## 4   NA    FALSE      <NA>
## 5   20     TRUE    Batman
```

2.6.1. ¿Cómo extraer elementos de una lista?

Para recuperar los elementos almacenados en una lista se usa el operador `$`, corchetes dobles `[[]]` o corchetes sencillos `[]`. A continuación unos **Ejemplos** para entender cómo extraer elementos de una lista.

Ejemplo

Si queremos la matriz almacenada con el nombre de `E2` dentro del objeto `milista` se puede usar el siguiente código.

```
milista$E2

##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    5    7    9   11
## [2,]    2    4    6    8   10   12
```

Es posible indicar la posición del objeto en lugar del nombre, para eso se usan los corchetes dobles.

```
milista[[2]]

##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    5    7    9   11
## [2,]    2    4    6    8   10   12
```

El resultado obtenido con `milista$E2` y `milista[[2]]` es **exactamente** el mismo. Vamos ahora a solicitar la posición 2 pero usando corchetes sencillos.

```
milista[2]

## $E2
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    5    7    9   11
## [2,]    2    4    6    8   10   12
```

La apariencia de este último resultado es similar, no igual, al encontrado al usar \$ y [[]]. Para ver la diferencia vamos a pedir la clase a la que pertenecen los tres últimos objetos usando la función `class`. A continuación el código usado.

```
class(milista$E2)
```

```
## [1] "matrix" "array"
```

```
class(milista[[2]])
```

```
## [1] "matrix" "array"
```

```
class(milista[2])
```

```
## [1] "list"
```

De lo anterior se observa claramente que cuando usamos \$ o [[]] el resultado es el objeto almacenado, una matriz. Cuando usamos [] el resultado es una **lista** cuyo contenido es el objeto almacenado.

Al manipular listas con \$ y [[]] se obtienen los objetos ahí almacenados, al manipular listas con [] se obtiene una lista.

3. Conceptos básicos de R-base.

Usando R como una calculadora

```
1 + 100
```

```
## [1] 101
```

R te mostrará la respuesta, precedido de un “[1]”. No te preocupes por esto por ahora, lo explicaremos más adelante. Por ahora piensa en eso como parte de la salida.

Al igual que bash, si escribes un comando incompleto R esperará a que lo completes:

```
“1 +”
```

3.1. Operadores

Si usas R desde la línea de comandos en lugar de estar dentro de RStudio, debes usar Ctrl + C en lugar de Esc para cancelar el comando. ¡Esto se aplica también a los usuarios de Mac!

La cancelación de un comando no sólo es útil para matar comandos incompletos: también puedes usarlo para decirle a R que deje de ejecutar el código (por **Ejemplo**, si tarda mucho más de lo que esperabas), o para deshacerte del código que estás escribiendo actualmente.

Operadores					
Aritméticos		Comparativos		Lógicos	
+	Adición	==	Igual a	&	Y lógico
-	Substracción	!=	Diferente de	!	NO lógico
*	Multiplicación	<	Menor que		O lógico
/	División	>	Mayor que	is.na(x)	Ausente?
^	Potencia	<=	Menor o Igual que		
%/%	División Entera	>=	Mayor o Igual que		

3.1.2. Operadores aritméticos

Los operadores aritméticos de R nos permiten realizar operaciones matemáticas, como sumas, divisiones o multiplicaciones, entre otras. La siguiente tabla resume todos los operadores aritméticos de R base.

Ejemplo

```
# Operaciones básicas
```

```
3 + 5
```

```
## [1] 8
```

```
8 - 3
```

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

```
7 * 5
```

```
## [1] 35
```

```
1/2
```

```
## [1] 0.5
```

```
4 ^ 4
```

```
## [1] 256
```

```
4 ** 4
```

```
## [1] 256
```

```
5 %% 3
```

```
## [1] 2
```

```
5 %/% 3
```

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

3.1.3 Operadores comparativos

Los operadores de comparación o relacionales están diseñados para comparar objetos. El resultado de estas comparaciones son de tipo booleano. La siguiente tabla resume los operadores relacionales de R.

Puedes comparar valores enteros con estos operadores de la siguiente manera.

Ejemplo

```
# Operaciones básicas
```

```
3 > 5
```

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

```
3 < 5
```

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

```
3 >= 5
```

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

```
3 <= 5
```

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

```
3 == 5
```

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

```
3 != 5
```

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

3.1.4. Operadores lógicos

Los operadores booleanos o lógicos en R se utilizan para especificar múltiples condiciones entre objetos. Estas comparaciones devuelven valores TRUE o FALSE.

Ejemplo

```
# Operaciones básicas
```

```
40 & 5 > 30
```

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

```
40 | 5 > 30
```

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

```
# Vectores
```

```
x <- c(3, 4, 5)
```

```
y <- c(3, 5, 1)
```

```
x & y
```

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

```
x | y
```

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

```
!x
```

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

3.2. Funciones

Funciones			
Matemáticas		Estadísticas	
sqrt(x)	Raíz de x	mean(x)	Media
exp(x)	Exponencial de x	sd(x)	Cuasidesviación
log(x)	Logaritmo natural de x	var(x)	Varianza
log10(x)	Logaritmo base 10	median(x)	Mediana
length(x)	Número de elementos	quantile(x,p)	Quantiles
sum(x)	Suma los elementos de x	cor(x,y)	Correlación
prod(x)	Producto de los elementos	max(x)	El máximo
sin(x)	Seno	min(x)	El mínimo
cos(x)	Coseno	range(x)	Retorna el máximo y mínimo
tan(x)	Tangente	sort(x)	Ordena las componentes de x
round(x,n)	redondea a n dígitos	which(condición)	los índices que cumplen la condición
cumsum(x)	calcula las sumas acumuladas $x_1, x_1 + x_2, \dots, x_1 + x_2 + x_3, \dots, x_1 + x_2 + \dots + x_n$	summary	Resumen de las variables
		choose(n, k)	número combinatorio de n sobre k

3.2.1. Funciones matemáticas

Existen funciones básicas muy utilizadas en matemáticas son: sin, cos, tan, asin, acos, atan, atan2, log, logb, log10, exp, sqrt, abs. A continuación algunos **Ejemplos** de las anteriores funciones.

Ejemplo

```
# **Ejemplo**s de medidas trigonométricas
angulos <- c(0, pi/2, pi)
sin(angulos)

## [1] 0.000000e+00 1.000000e+00 1.224606e-16

tan(angulos)

## [1] 0.000000e+00 1.633124e+16 -1.224647e-16

# **Ejemplo**s de logaritmos
log(100)

## [1] 4.60517

logb(125, base=5)

## [1] 3

# **Ejemplo**s de exponencial
exp(1)

## [1] 2.718282

exp(2)

## [1] 7.389056

exp(1:3)

## [1] 2.718282 7.389056 20.085537

# **Ejemplo**s de raíces
sqrt(49) # Raiz cuadrada de 49

## [1] 7

27 ^ (1/3) # Raiz cúbica de 273

## [1] 3

#**Ejemplo**s de valor absoluto
abs(2.5)

## [1] 2.5

abs(-3.6)

## [1] 3.6
```

3.2.2. Funciones estadísticas

Las funciones que aquí vamos a mostrar, se aplican generalmente a los valores de una matriz o vector de manera que nos devolverán los valores de estos:

Nota : Las Medidas de tendencia central Representan un valor alrededor del cual tienden a concentrarse las observaciones. Las más utilizadas son la media aritmética o promedio, la mediana y la moda.

Nota : Las medidas de posición Son valores que indican la ubicación de los datos en relación con el resto de las observaciones ordenadas, y resultan muy útiles para identificar el comportamiento de un conjunto de datos. Las medidas de posición se llaman generalmente cuantiles, y se pueden clasificar en tres grupos: cuartiles, deciles y percentile

Nota : las medidas de dispersión Permiten cuantificar la variabilidad de una característica de interés. La variación puede definirse, como la diferencia que existe entre las unidades de estudio respecto a la variable analizada (Isaza, 2012). Entre las medidas de dispersión se encuentran: el rango, el rango intercuartil, la varianza, la desviación estándar y el coeficiente de variación.

x: variable cuantitativa sobre la que se va a realizar el cálculo.

na.rm: indica si se deben excluir los datos faltantes (valores NA), en caso de que la variable de interés los tenga. Por defecto: na.rm = FALSE

Ejemplo

```
x<- c(18,19,20,18,24,17,22,15,22,25) # Vector
min <- min(x, na.rm = TRUE) # Mínimo
q1 <- quantile(x, probs = 0.25, na.rm = TRUE) # Quantil Q1
media <- mean.default(x, na.rm = TRUE) # Media
media_rec <- mean.default(x, trim = 0.025, na.rm = TRUE) # Media recortada
mediana <- median.default(x, na.rm = TRUE) # Mediana
moda <- mode(x) # Moda
var <- var(x, na.rm = TRUE) # Varianza
desvest <- sd(x, na.rm = TRUE) # Desviación Estándar
q3 <- quantile(x, probs = 0.75, na.rm = TRUE) # Quantil Q3
max <- max(x, na.rm = TRUE) # Máximo
s <- skew(x) # Simetría
c <- kurtosi(x) # Curtosis

Est_descriptivos<- as.numeric(c(min, q1, media, media_rec, mediana, moda,
var, desvest, q3, max, s, c))

nombres <- c("Mínimo", "Q1", "Media", "Media recortada", "Mediana", "Moda",
"Varianza", "Desviación Estándar", "Q3", "Máximo", "Simetría",
"Curtosis")

Est_descriptivos<-data.frame("Estadística" = nombres,
                             Valor =round(Est_descriptivos,2) )

kableExtra::kable(Est_descriptivos)

summary(x) # resumen de las variables
```

3.3. Crea Una funcion

Puedes crear funciones para encapsular un conjunto de instrucciones que realizan una tarea específica. Las funciones te permiten reutilizar código, mejorar la legibilidad y modularizar tu programa. Una función en R consta de los siguientes elementos:

1. **Nombre de la función:** Debes elegir un nombre descriptivo para tu función, que sea único dentro de tu entorno de trabajo.
2. **Argumentos:** Los argumentos son los valores que se pasan a la función para que los procese. Pueden ser obligatorios u opcionales, y puedes definir valores predeterminados para los opcionales.

3. **Cuerpo de la función:** Es el conjunto de instrucciones que la función ejecuta cuando se llama. Puede incluir operaciones aritméticas, llamadas a otras funciones, bucles, etc.
4. **Valor de retorno:** Es el resultado que devuelve la función después de ejecutar las instrucciones en su cuerpo. Puede ser cualquier objeto de R, como un vector, una matriz, un marco de datos, etc.

```
# Definir la función
sumar <- function(a, b) {
  resultado <- a + b # Suma los dos números
  return(resultado) # Devuelve el resultado
}

# Llamar a la función y almacenar el resultado en una variable
resultado_suma <- sumar(5, 3)

# Imprimir el resultado
print(resultado_suma)

## [1] 8
```

4. Importar y exportar datos

R puede importar datos de una amplia variedad de tipos de archivo con las funciones en base además de que esta capacidad es ampliada con el uso de paquetes específicos.

Cuando importamos un archivo, estamos guardando su contenido en nuestra sesión como un objeto. Dependiendo del procedimiento que usemos será el tipo de objeto creado.

De manera análoga, podemos exportar nuestros objetos de R a archivos en nuestra computadora.

Tipos de datos que veremos

Introduciremos funciones para importar/exportar datos de los siguientes formatos: * Datos en formato texto (o tabulares)

- CSV: .csv (comma separated values o , en castellano, datos separados por comas) otros datos en formato texto como .txt

Formatos de otros programas (software propietario)

- EXCEL: .xls y .xlsx
- SPSS: .sav y .por
- STATA: .dta
- SAS: .sas

Formatos propios de R

- R objects: .RData o .rda
- Serialized R objects: .rds

4.1. Lectura de datos desde tu PC

Para leer archivos de tu PC poniendo la ruta del archivo, cambiando \ por /, los archivos tienes que proporcionarle a R la ruta del archivo con el nombre. Lo puedes hacer tú poniendo toda la ruta con el nombre y la extensión. Tipo:

Nota: Los argumentos de la función read.table son:

- file: nombre o ruta donde están alojados los datos. Puede ser un url o una dirección del computador. Es también posible usar file.choose() para que se abra un ventana y adjuntar el archivo deseado manualmente.
- header: valor lógico, se usa TRUE si la primera línea de la base de datos tiene los nombres de las variables, caso contrario se usa FALSE.
- sep: tipo de separación interna para los datos dentro del archivo. Los valores usuales para este parámetros son:
 - sep = ',' si el archivo tiene extensión .csv.
 - sep = ' ' si el archivo es bloc de notas con espacios por la barra espaciadora.
 - sep = '\t' si el archivo es bloc de notas con espacios por la barra tabuladora.

Ejemplo

```
# Lectura de datos .csv
tusDatos <- read_csv(file="la_ruta_del_archivo", header = TRUE)

# Lectura de datos .txt
tusDatos <- read.table(file="la_ruta_del_archivo", header =TRUE, sep =';')
```

Ejemplo

```
library("readxl") # Lectura de datos .xlsx o .xls

tusDatos <- read_excel(file="la_ruta_del_archivo",
                      sheet = "el nombre de la Hoja que quieres leer")

# Puedes leer la hoja número 1. La primera de todas
misDatos <- read_excel(file="la_ruta_del_archivo", sheet = 1)
```

4.2. Lectura de datos desde tu internet

Para leer archivos desde internet, se debe poner la ruta (URL) de la web donde hay datos.

Ejemplo

```
tusDatos <- read.csv("http://www.sthda.com/upload/boxplot_format.txt", sep="\t")
```

5. Gráficas

R cuenta con un sistema de generación de gráficas poderoso y flexible. Si nembargo, tener estar cualidades hace que este sistema sea un tanto complejo para aprender.

Para realizar los gráficos de este tutorial vamos a utilizar datos que se encuentran accesibles en el paquete AER . Por tanto, cargamos la librería AER (si no la tenemos hay que instalar el paquete).

El paquete AER contiene más de 100 bases de datos (data sets). ¿Recordamos cómo podemos obtener el listado de datos que contiene un paquete?

Así pues, tenemos un total de 554 observaciones y las siguientes 11 variables:

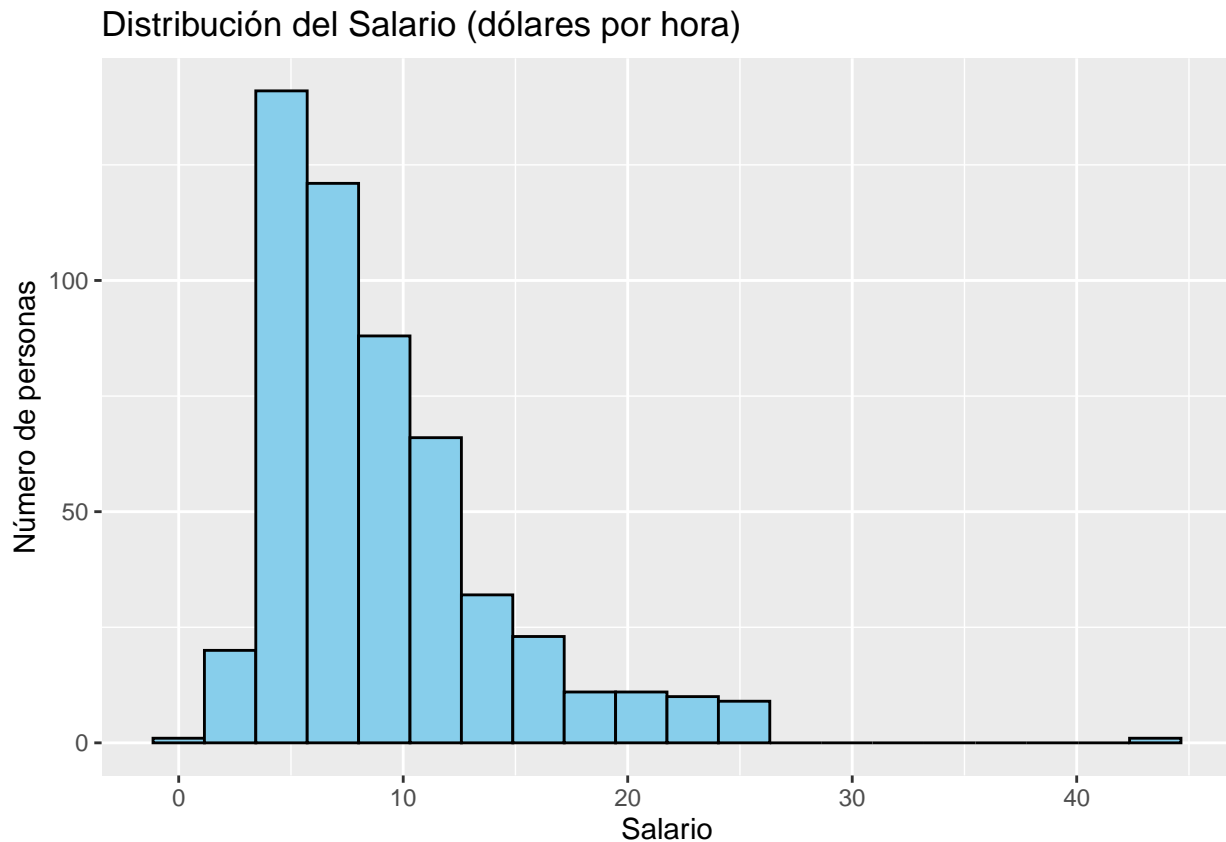
- Wage : Salario (dólares por hora).
- Education : Número de años de educación.
- Experiencia : Número de años de experiencia laboral.
- Age : Edad (años).
- Ethnicity : Raza (1=Otro, 2=Hispano, 3=Blanco).

- Region : Variable indicadora para la Región Sur (1=Persona vive en Sur, 0=Persona vive en otro lugar).
- Gender : Variable indicadora de sexo (1=Mujer, 0=Hombre).
- Occupation : Categoría ocupacional (1=Administración, 2=Ventas, 3=Administrativo, 4=Servicio, 5=Profesional, 6=Otro).
- Sector : Sector (0=Otro, 1=Fabricación, 2=Construcción).
- Union : Variable indicadora de afiliación sindical (1=Sindicalista, 0=No afiliado).
- Married : Estado Civil (0=Soltero, 1=Casado)

5.1. Histogramas

Un histograma es una gráfica que nos permite observar la distribución de datos numéricos usando barras. Cada barra representa el número de veces (frecuencia) que se observaron datos en un rango determinado.

```
ggplot(datos, aes(x = wage)) +
  geom_histogram(bins = 20, fill = "skyblue", color = "black") +
  labs(
    title = "Distribución del Salario (dólares por hora)",
    x = "Salario",
    y = "Número de personas"
  )
```



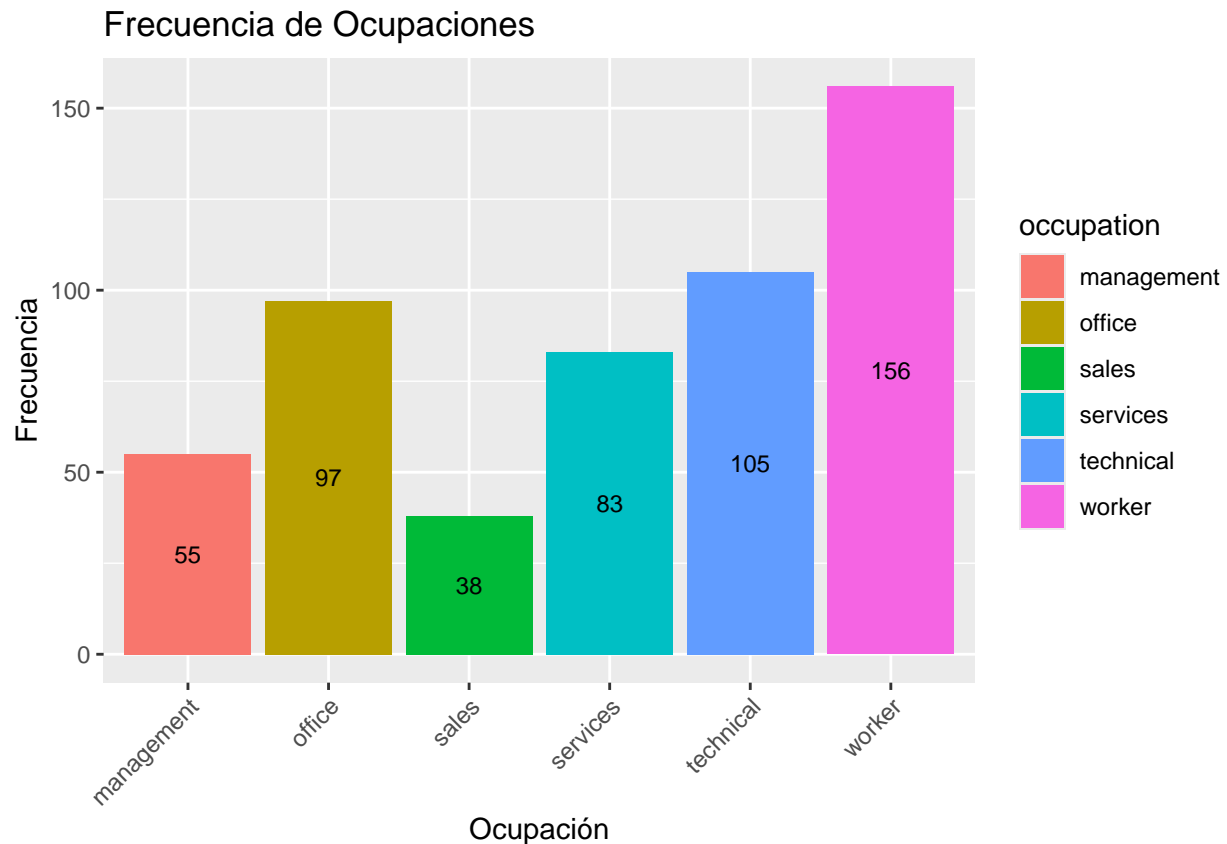
5.2. Gráficas de barras

Este es quizás el tipo de gráfico mejor conocido de todos. Una gráfica de este tipo nos muestra la frecuencia con la que se han observado los datos de una variable discreta, con una barra para cada categoría de esta variable.

```
frecuencias <- table(datos$occupation) # con variable categórica

# Convertir las frecuencias a un dataframe
df_frecuencias <- data.frame(occupation = names(frecuencias),
                             frecuencia = as.numeric(frecuencias))

# Crear el gráfico de barras con ggplot2
ggplot(df_frecuencias, aes(x = occupation, y = frecuencia, fill = occupation)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = frecuencia), position = position_stack(vjust = 0.5), size = 3) +
  labs(title = "Frecuencia de Ocupaciones", x = "Ocupación", y = "Frecuencia") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



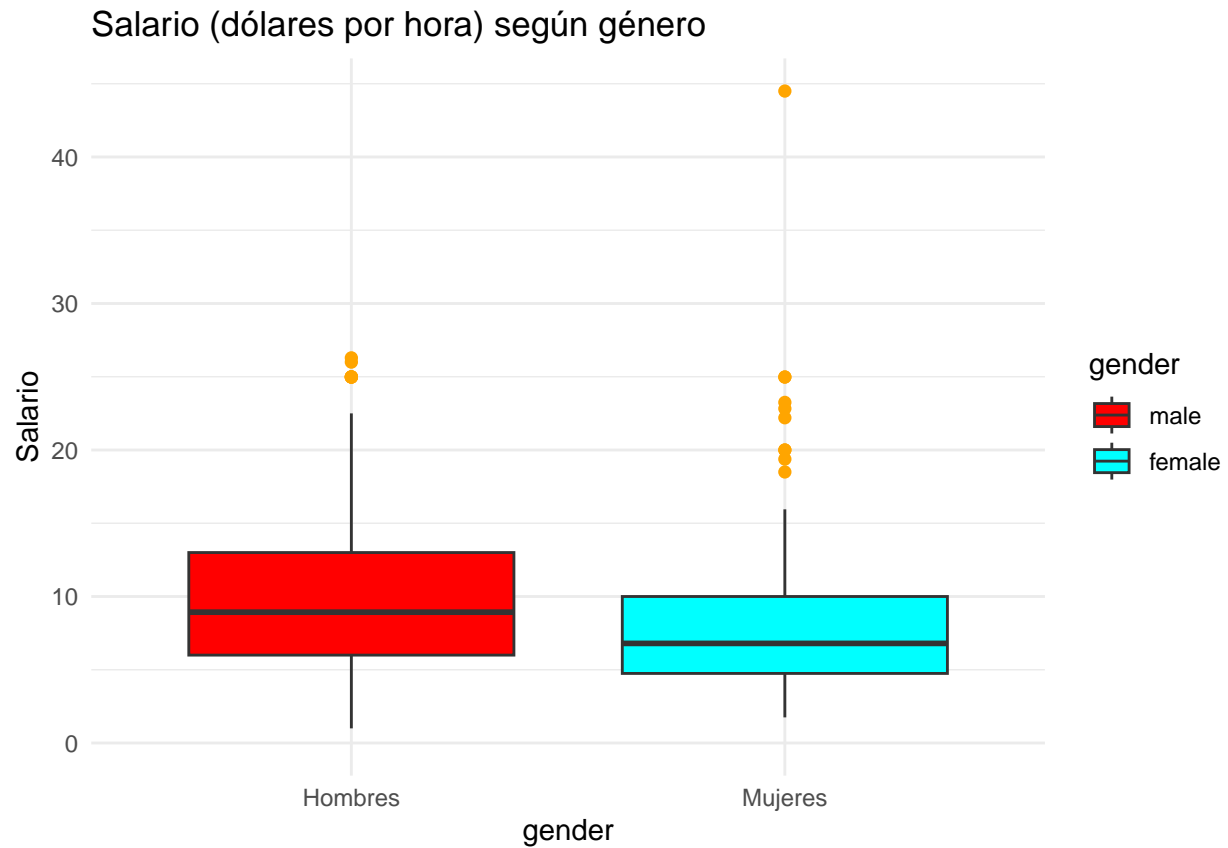
5.3. Boxplot (diagrama de Caja y Bigotes)

los gráficos Boxplot (o Box and Whiskers, más conocidos como Caja y Bigotes) se representan los cinco números (mínimo, cuartil 1, mediana, cuartil 3 y máximo), lo que nos puede ayudar a hacernos una idea de la asimetría de la distribución. Los gráficos de Caja y Bigotes también permiten detectar valores atípicos y outliers.

Boxplot según el salario entre Hombre (Male) y Mujeres (Female)

```
ggplot(datos, aes(x = gender, y = wage, fill = gender)) +
  geom_boxplot(outlier.colour = "orange", outlier.shape = 16, outlier.size = 2) +
  labs(title = "Salario (dólares por hora) según género", y = "Salario") +
  scale_x_discrete(labels = c("Hombres", "Mujeres")) +
  scale_fill_manual(values = rainbow(2)) +
```

```
coord_cartesian(ylim = c(0, max(datos$wage))) +
theme_minimal()
```

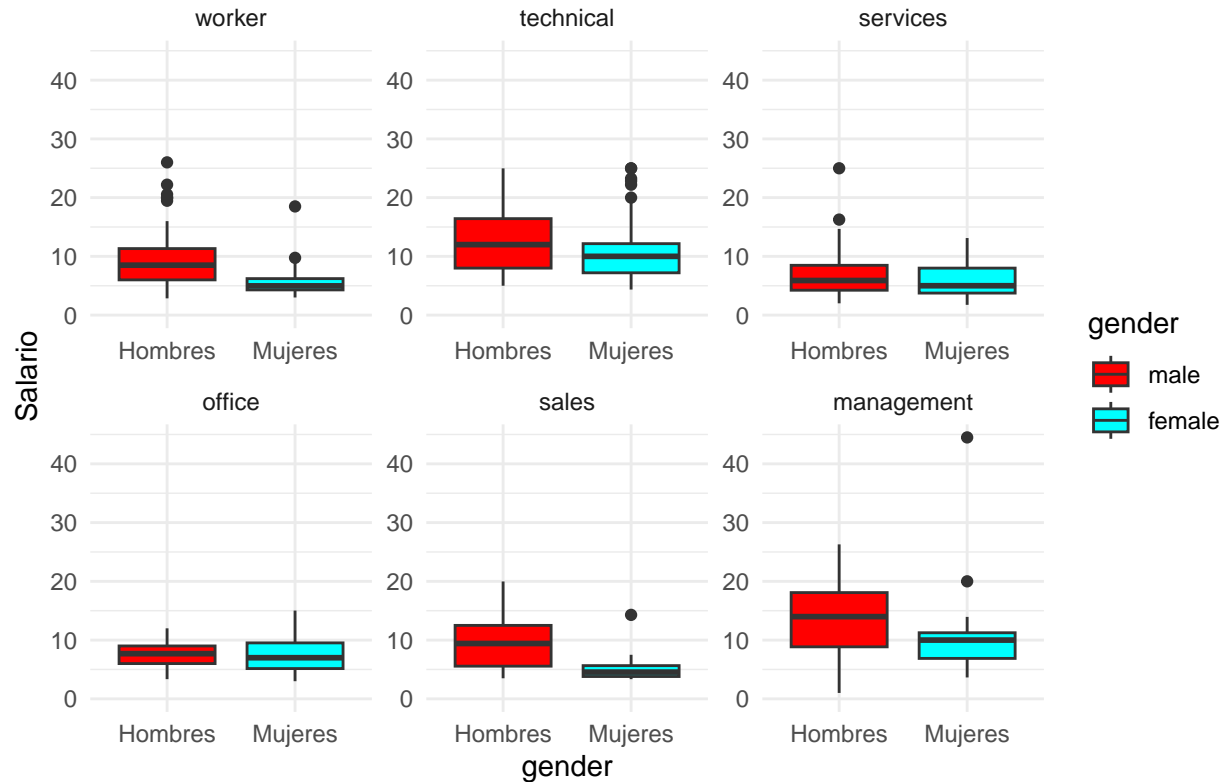


Boxplot según su trabajo entre Hombre (Male) y Mujeres (Female)

```
ggplot(datos, aes(x = gender, y = wage, fill = gender)) +
  geom_boxplot() +
  labs(title = "Salario (dólares por hora) según género", y = "Salario") +
  scale_x_discrete(labels = c("Hombres", "Mujeres")) +
  scale_fill_manual(values = rainbow(2)) +

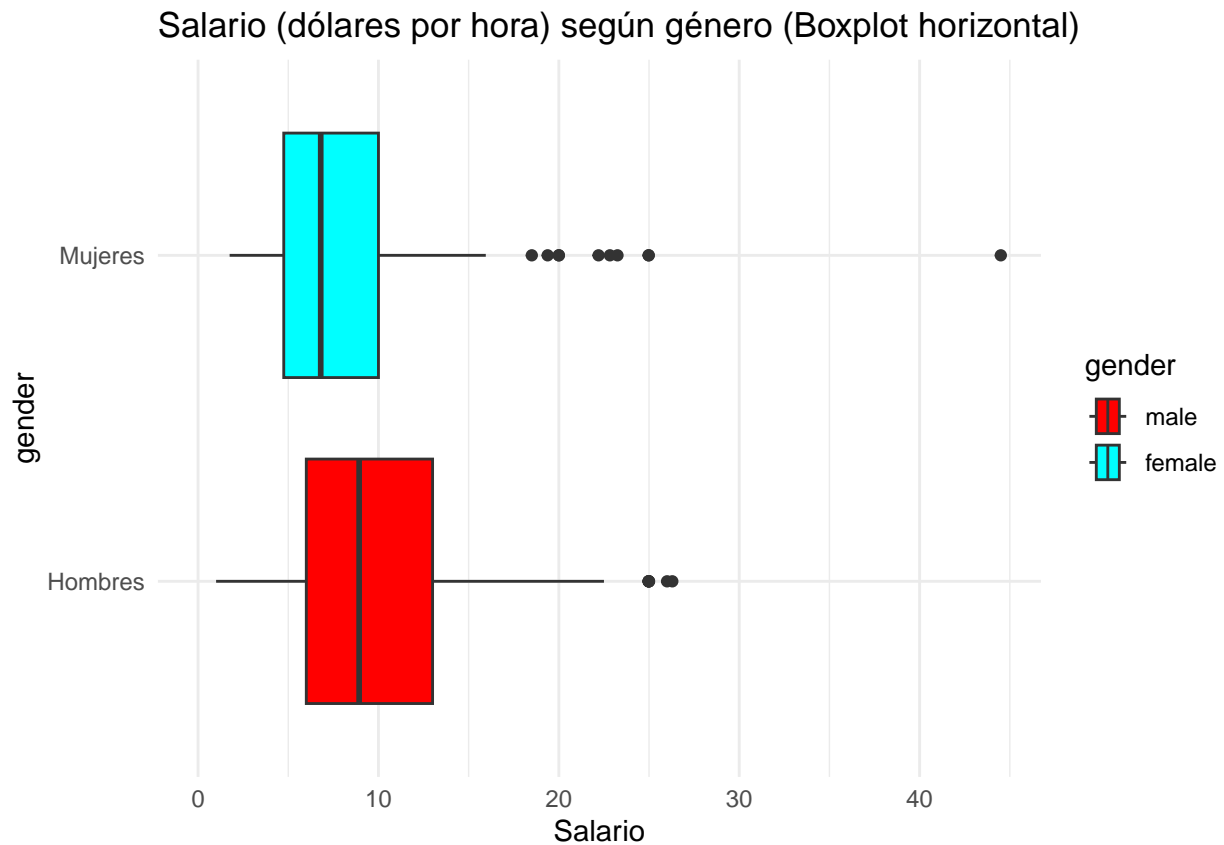
  facet_wrap(~ occupation, scales = "free", nrow = 2) +
  coord_cartesian(ylim = c(0, max(datos$wage))) +
  theme_minimal()
```

Salario (dólares por hora) según género



Boxplot horizontal con variables categóricas invertidas

```
ggplot(datos, aes(x = wage, y = gender, fill = gender)) +
  geom_boxplot() +
  labs(title = "Salario (dólares por hora) según género (Boxplot horizontal)",
       x = "Salario") +
  scale_y_discrete(labels = c("Hombres", "Mujeres")) +
  scale_fill_manual(values = rainbow(2)) +
  coord_cartesian(xlim = c(0, max(datos$wage))) +
  theme_minimal()
```

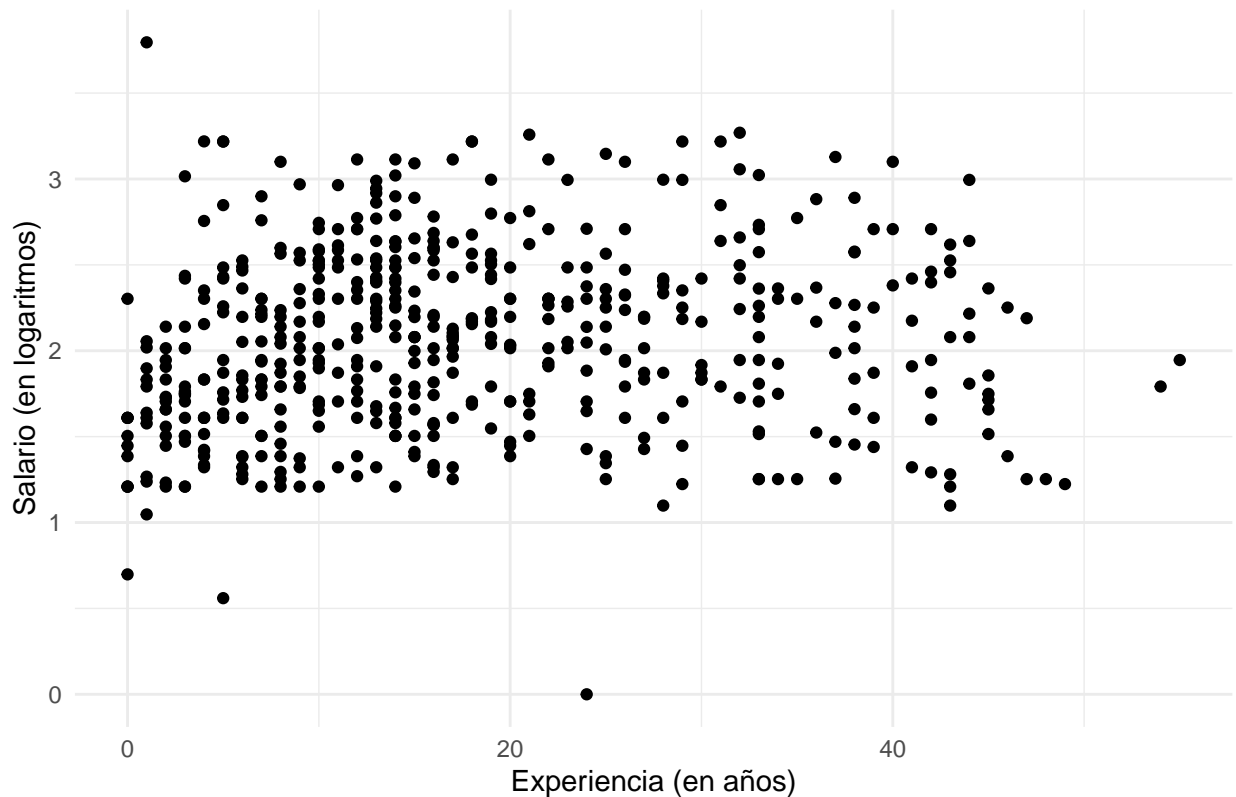



5.4. Diagrama de dispersión

Los diagramas de dispersión permiten realizar un diagnóstico visual de la posible relación (funcional) entre dos variables de naturaleza cuantitativa. La función que permite realizar este tipo de gráfico es `plot()`.

```
ggplot(datos, aes(x = experience, y = log(wage))) +
  geom_point() +
  labs(title = "Salario (en logaritmos) en función de la experiencia",
        x = "Experiencia (en años)",
        y = "Salario (en logaritmos)") +
  theme_minimal()
```

Salario (en logaritmos) en función de la experiencia



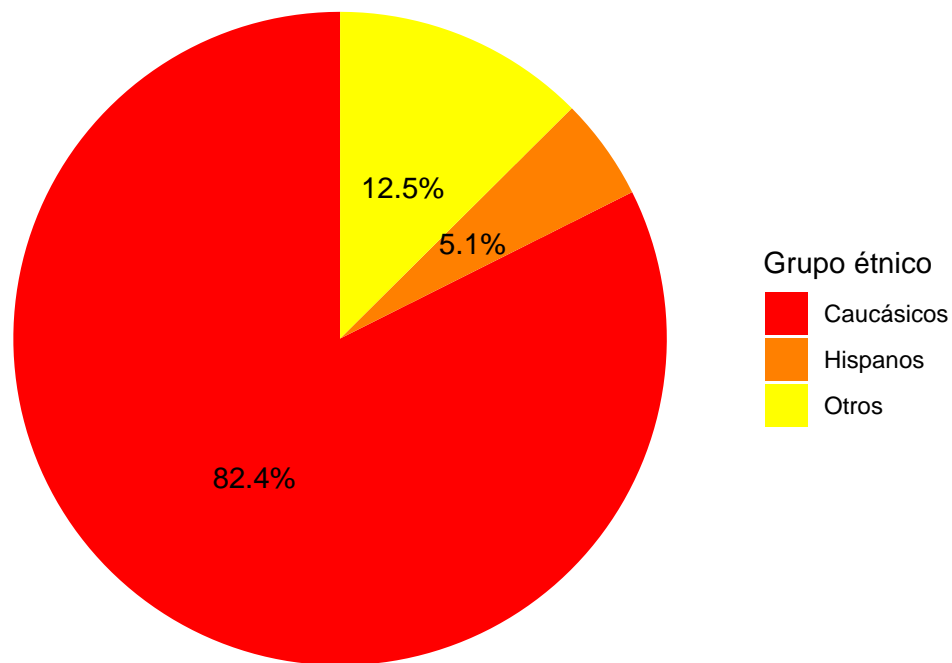
5.5. Gráficos de sectores

Un gráfico de sectores, también conocido como gráfico circular, gráfico de pastel, de tarta, de torta o pie chart, por su nombre en inglés, es un gráfico circular que representa proporciones o porcentajes en sectores, donde el área y la longitud del arco de cada sector es proporcional a la cantidad representada. Existen variaciones de este tipo de gráfico, como los gráficos de anillos, de donut, de waffles y gráficos spie. En este tutorial revisaremos cómo hacer un gráfico de sectores en R base.

```
# Crear un nuevo conjunto de datos con etiquetas de porcentaje
datos_porcentajes <- data.frame(ethnicity = names(prop.table(table(datos$ethnicity)) * 100), porcentaje = prop.table(table(datos$ethnicity)) * 100)

ggplot(datos_porcentajes, aes(x = "", y = porcentaje.Freq, fill = ethnicity)) +
  geom_bar(width = 1, stat = "identity") +
  coord_polar(theta = "y") +
  labs(title = "Gráfico de sectores", fill = "Grupo étnico") +
  scale_fill_manual(values = heat.colors(3), labels = c("Caucásicos", "Hispanos", "Otros")) +
  geom_text(aes(label = paste0(round(porcentaje.Freq, 1), "%")),
            position = position_stack(vjust = 0.5)) +
  theme_void()
```

Gráfico de sectores



6. Bucles y Condicionales

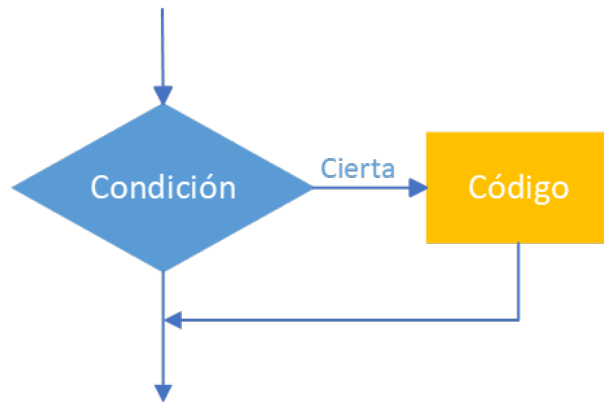
Los bucles y las sentencias se utilizan para controlar el flujo de ejecución de un programa. Aquí tienes una descripción de los bucles y las sentencias más comunes en RStudio.

6.1. Condicionales

Los condicionales en RStudio se utilizan para tomar decisiones en función del cumplimiento o no de una condición. Los condicionales más comunes son *if*, *else if* y *else*.

1. **if**: Se utiliza para ejecutar un bloque de código si una condición es verdadera.

Estructura Condicional simple



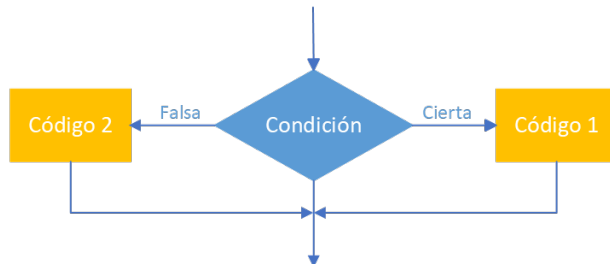
```
edad<-18
if(edad >=18){
  cat("\n Es mayor de edad")
}
```

##

Es mayor de edad

2. **else**: Se ejecuta si ninguna de las condiciones anteriores es verdadera.

Estructura Condicional doble



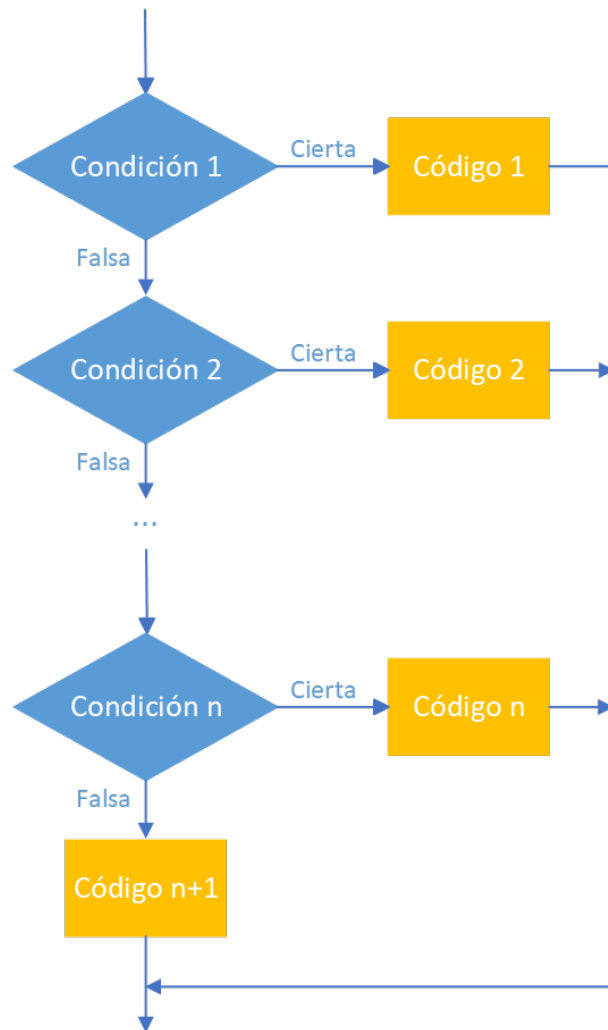
```
edad<-17
if(edad >=18){
  cat("\n Es mayor de edad")
} else {
  cat("\n Es menor de edad")
}
```

##

Es menor de edad

3. **else if**: Permite evaluar otra condición si la primera no se cumple.

Estructura Condicional múltiple



```
nota<-4.5
if(nota<3){
  cat("\n Reprobado")
} else if(nota<4 ){
  cat("\n Aprobado")
} else {
  cat("\n Excelele")
}
```

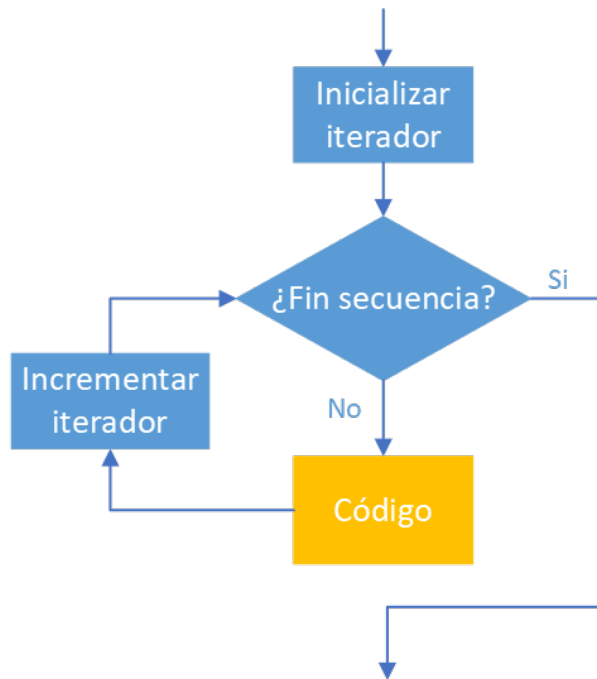
```
##
## Excelele
```

6.2. Bucles

los bucles son secuencias repetitivas hasta culminar el contador, de los cuales tenemos **for** y **while**

1. **Bucle for:** Un bucle for se utiliza para iterar sobre una secuencia de elementos (como un vector o una lista) y ejecutar un bloque de código para cada elemento en la secuencia.

Bucle iterativo



```
for (variable in secuencia) {  
  # Código a ejecutar  
}
```

1.1. Bucle for con vectores numéricos

```
# Crear un vector numérico  
numeros <- c(1, 2, 3, 4, 5)  
  
# Iterar sobre cada elemento del vector e imprimirlo  
for (num in numeros) {  
  print(num)  
}
```

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

1.2. Bucle for con listas

```
# Crear una lista con diferentes tipos de datos  
mi_lista <- list(nombre = "Juan", edad = 30, casado = TRUE)  
  
# Iterar sobre cada elemento de la lista e imprimirlo  
for (elemento in mi_lista) {  
  print(elemento)  
}
```

```
## [1] "Juan"
```

```
## [1] 30
## [1] TRUE
```

1.3. Bucle for con matrices

```
# Crear una matriz 2x2
matriz <- matrix(1:4, nrow = 2)

# Iterar sobre cada elemento de la matriz e imprimirlo
for (fila in 1:nrow(matriz)) {
  for (columna in 1:ncol(matriz)) {
    print(matriz[fila, columna])
  }
}
```

```
## [1] 1
## [1] 3
## [1] 2
## [1] 4
```

1.4. Bucle for con data frames

```
# Crear un data frame con información de estudiantes
estudiantes <- data.frame(nombre = c("María", "Pedro", "Luis"),
                          edad = c(25, 28, 30),
                          calificación = c(85, 90, 88))

# Iterar sobre cada fila del data frame e imprimir información
for (i in 1:nrow(estudiantes)) {
  cat("Nombre:", estudiantes[i, "nombre"], "\n")
  cat("Edad:", estudiantes[i, "edad"], "\n")
  cat("Calificación:", estudiantes[i, "calificación"], "\n\n")
}
```

```
## Nombre: María
## Edad: 25
## Calificación: 85
##
## Nombre: Pedro
## Edad: 28
## Calificación: 90
##
## Nombre: Luis
## Edad: 30
## Calificación: 88
```

1.5. Bucle for condicionado

La instrucción **break** es una herramienta esencial en la programación para interrumpir la ejecución de un bucle y salir de él. Esta instrucción se emplea tanto en bucles iterativos como en bucles condicionales. Su función principal radica en detener la repetición del bloque de código del bucle en el momento en que se cumple una condición específica. Una vez que se alcanza esta condición, el flujo del programa se interrumpe y se sale del bucle, permitiendo que la ejecución continúe fuera de él. Esta capacidad de control de flujo es fundamental para optimizar el rendimiento y la lógica de los programas, ya que permite gestionar eficientemente el flujo de ejecución en función de condiciones específicas.

La instrucción **next** es una herramienta fundamental en la programación para controlar el flujo de ejecución dentro de un bucle. Se utiliza para interrumpir la ejecución del bloque de código del bucle, pero en lugar

de salir del bucle, avanza a la siguiente iteración. En el caso de bucles iterativos, el iterador se desplaza al siguiente elemento de la secuencia de iteración, mientras que en los bucles condicionales, la evaluación de la condición de repetición se reinicia.

```
# Crear un vector de números del 1 al 10
numeros <- 1:10

# Iterar sobre cada número en el vector
for (num in numeros) {
  # Imprimir el número actual
  cat("Número actual:", num, "\n")

  # Comprobar si el número es par
  if (num %% 2 == 0) {
    cat("Este número es par.\n")
  } else {
    cat("Este número es impar.\n")
  }

  # Si el número es mayor que 5, salir del bucle
  if (num > 5) {
    cat("El número es mayor que 5. Salir del bucle.\n")
    break
  }

  # Si el número es divisible por 3, pasar al siguiente número
  if (num %% 3 == 0) {
    cat("Este número es divisible por 3. Continuar con el siguiente número.\n")
    next
  }

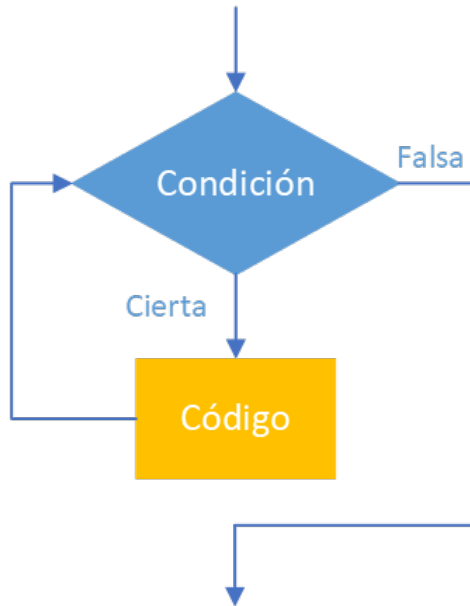
  # Imprimir un espacio en blanco para separar cada iteración
  cat("\n")
}
```

```
## Número actual: 1
## Este número es impar.
##
## Número actual: 2
## Este número es par.
##
## Número actual: 3
## Este número es impar.
## Este número es divisible por 3. Continuar con el siguiente número.
## Número actual: 4
## Este número es par.
##
## Número actual: 5
## Este número es impar.
##
## Número actual: 6
## Este número es par.
## El número es mayor que 5. Salir del bucle.
```

2. Bucle while: Un bucle **while** es una estructura de control de flujo que repite un bloque de código

mientras una condición especificada sea evaluada como verdadera. Mientras la condición sea verdadera, el bloque de código se ejecutará repetidamente.

Bucle condicional



```
# Contador hasta cierto límite
contador<-0
limite <- 10

while (contador < limite) {
  print(paste("El contador es:", contador))
  contador <- contador + 1
}
```

```
## [1] "El contador es: 0"
## [1] "El contador es: 1"
## [1] "El contador es: 2"
## [1] "El contador es: 3"
## [1] "El contador es: 4"
## [1] "El contador es: 5"
## [1] "El contador es: 6"
## [1] "El contador es: 7"
## [1] "El contador es: 8"
## [1] "El contador es: 9"
```

```
# Generación de números pares
numero <- 0
limite <- 10
while (numero < limite) {
  if (numero %% 2 == 0) {
    print(paste("Número par:", numero))
  }
  numero <- numero + 1
}
```

}

Referencias Bibliográficas

Programming with data: A guide to the s language. (1998). *Technometrics*, 41(3), 266–267.

R Core Team. (2022). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. <https://www.R-project.org/>