Fundamentos de programación y acceso a base de datos en R

PID_00293465

Marta Casals Fontanet
Alícia Vila Grifo

Tiempo mínimo de dedicación recomendado: 5 horas

Universitat Oberta de Catalunya

Marta Casals Fontanet

Estadística de formación y profesión. Licenciada en ITM. Máster en Educación y nuevas tecnologías. Profesora de Matemáticas en secundaria y universidad. Profesora colaboradora en el máster de Bioinformática y bioestadística de la UOC y la UB.

Alicia Vila Grifo

Licenciada en Matemáticas por la Universidad de Valencia y profesora consultora de Probabilidad, Estadística y Análisis de Datos en diferentes estudios de la UOC. Actualmente es profesora funcionaria de Informática en el Departamento de Educación de la Generalitat de Cataluña, en los ámbitos de programación y bases de datos. También participa en la coordinación de proyectos de aplicaciones web y de análisis de datos.

El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por los profesores: Álvaro Leitao Rodríguez y David Cantón Fabà

Cómo citar este recurso de aprendizaje con el estilo Harvard:

Casals, M. y Vila A. (2024). *Fundamentos de programación y acceso a base de datos en R* [Recurso de aprendizaje textual]. 1.a ed. Fundació Universitat Oberta de Catalunya (FUOC).

Primera edición: febrero 2024

Av. Tibidabo, 39-43, 08035 Barcelona
Autoría: Marta Casals Fontanet, Alícia Vila Grifo
Producción: FUOC

Índice

1. Introducción al LAB3

El **LAB3** es un recurso didáctico complementario de la asignatura **Software para el análisis de datos (SAD)** del máster interuniversitario de Bioinformática y Bioestadística de la Universitat Oberta de Catalunya (UOC) y la Universitat de Barcelona (UB).

Este **LAB3** forma parte de un conjunto de laboratorios prácticos que conjugan contenidos teóricos con ejemplos, ejercicios y casos prácticos reales del ámbito de conocimiento del máster.

El **LAB3** explica los conceptos básicos de programación en el lenguaje **R** y su aplicación al ámbito biosanitario. En realidad, aunque **R** es un lenguaje de programación, no suele utilizarse para desarrollar grandes proyectos sino más bien para realizar paquetes de funciones o bloques de programas que puedan ser reutilizados con un objetivo específico. Por ello, las aplicaciones más habituales suelen estar dirigidas a la extracción, manipulación, tratamiento e interpretación de conjuntos de datos con fines específicos.

En este **LAB3**, una vez introducidos los primeros elementos del lenguaje **R** (tipos de datos, operadores y estructuras de datos), trabajaremos las instrucciones condicionales y bucles, y la funciones y librerías.

El **LAB3** se complementará con las instrucciones de acceso a datos trabajando conceptos de gestión de bases de datos, utilizando su lenguaje propio, Structure Query Language (SQL).

2. Instrucciones del LAB3

El **LAB3** contiene una serie de apartados con introducciones teóricas, ejemplos y casos prácticos, además de otros ejercicios que se propondrán para que sean resueltos por el estudiante. La temporización y las pautas para trabajar el **LAB3** serán indicadas en el aula de la asignatura. Del **LAB3** se presentará una propuesta de solución orientativa que servirá para que el estudiante pueda autoevaluar las soluciones realizadas por él mismo.

En ocasiones, para realizar los ejercicios, se utilizarán conjuntos de datos de paquetes propios de **RStudio**, como es el caso de *biopsy*, *anorexia* y *birthwt* del paquete *MASS* o *airquality* y *esoph* del paquete *datasetsDb*, así como también otros conjuntos de datos abiertos de repositorios externos. Todos los ejercicios se realizarán en el entorno de desarrollo integrado para **R**, **RStudio** (https://cran.rstudio.com/), como lenguaje de programación para la informática estadística y los gráficos.

3. Contenidos del LAB3

En este **LAB3** trabajaremos los siguientes contenidos:

- Elementos básicos de **R**: operadores, variables y tipos de datos.
- Estructuras de datos.
- Instrucciones básicas del lenguaje **R**.

- Funciones en **R**.
- Acceso a bases de datos en **R**.
- Ejercicios y casos prácticos en **R**.
- Soluciones de los ejercicios y casos prácticos en **R**.

4. Elementos básicos de R: operadores, variables y tipos de datos

Recapitulamos algunos conceptos vistos en el **LAB1**, revisando nuevamente los elementos básicos de **R**, como es el caso de los operadores, las variables y los tipos.

4.1. Operadores en R

La sintaxis en **R** de los operadores matemáticos más comunes es:

Operador	Sintaxis	Ejemplo
Suma	+	x+y
Resta	-	x-y
Multiplicación	*	x*y
División	/	x/y
Módulo división entera	%%	x%%y
Exponencial	**	x**y

La sintaxis en **R** de los operadores lógicos más comunes es:

Operador	Sintaxis	Ejemplo
Menor	<	x<y
Menor o igual	<=	x<=y
Mayor	>	x>y
Mayor o igual	>=	x>=y
Igual	==	x==y
Distinto	!=	x!=y
Negación	!	!x
O	|	x|y
Y	&	x&y

4.2. Definición y asignación de variables

A diferencia de otros lenguajes de programación, en **R** no es necesario definir el tipo de variable para poder crearla y utilizarla. La asignación de un valor a una determinada variable se realiza utilizando la notación **<-**, por ejemplo, x<-6, significa que la variable x toma el valor 6.

4.3. Tipos de datos

R permite manejar una gran variedad de tipos de datos. A continuación, especificamos los tipos principales más utilizados.

4.3.1. Datos de tipo numérico

Tipo de dato	Descripción	Ejemplo
Double	Doble precisión	num<-3.0
Numeric	Números decimales	num<-3.0
Integer	Números enteros	num<-3
Character	Cadenas de texto	num<-"cadena texto"
Complex	Números complejos	num<-5+3i
Logical	Verdadero (TRUE) o falso (FALSE). Resultado de operación lógica	num1<-3;num2<-4;num1<num2
Raw	Datos en bytes sin procesar. Cualquier dato se puede transformar a este tipo con la función correspondiente	num<-5+3i
Factor	No es un tipo de dato, pero habitualmente se utiliza para describir estructuras de datos para representar un vector, por ejemplo.	colores<-c("Rojo","Verde");factor(colores) mostrará los elementos del vector

4.3.2. Comprobación de tipos y conversores

En ocasiones, a la hora de realizar algún tipo de operación entre variables, es necesario comprobar el tipo para garantizar que se pueda realizar la operación y que el resultado sea correcto. Para ello, **R** ofrece funciones específicas para comprobar si un objeto es de un tipo de datos particular. Por ejemplo, *is.numeric()* verifica si un objeto es numérico, *is.character()* verifica si es una cadena de caracteres, y así sucesivamente.

Ejemplo 1:

```
cad <- "Cadena"
if (is.character(cad)) {
  print("cad es una cadena de caracteres.")
}

## [1] "cad es una cadena de caracteres."
```

La función *typeof()* permite obtener el tipo de datos interno de un objeto en **R**.

Ejemplo 2:

```
num <- 34
if (typeof(num) == "double") {
  print("num es un número de punto flotante.")
}

## [1] "num es un número de punto flotante."
```

En ocasiones, es interesante la combinación de las funciones *is.*()* con *class()* o *typeof()* para realizar comprobaciones más específicas.

Ejemplo 3:

```
num <- 42
if (is.numeric(num) && typeof(num) == "double") {
  print("num es un número de punto flotante.")
}

## [1] "num es un número de punto flotante."
```

Más adelante cuando se expliquen estructuras de datos como vectores o data frames, se podrán utilizar las funciones *is.()** que se pueden aplicar a cada elemento del vector o a cada columna del data frame para verificar el tipo de datos de manera más granular.

Ejemplo 4:

```
vector_numérico <- c(1, 2, 3)
if (all(is.numeric(vector_numérico))) {
  print("Todos los elementos son numéricos.")
}

## [1] "Todos los elementos son numéricos."
```

Estas son algunas de las formas más comunes de realizar comprobaciones de tipos de datos en **R**.

Las funciones de conversión de tipos son funciones en **R** que permiten cambiar el tipo de datos de una variable de una clase a otra. Esto es útil cuando es necesario transformar datos de un tipo en otro para realizar operaciones específicas o para ajustarlos a un formato determinado. **R** ofrece varias funciones de conversión de tipos que permiten realizar estas transformaciones, las más comunes son las siguientes:

- **as.character()**: conversión a caracteres

La función *as.character()* se utiliza para convertir un objeto en un carácter.

Ejemplo 5:

```
número <- 42
cadena <- as.character(número)
cadena
```

```
## [1] "42"
```

- **as.numeric()**: conversión a numérico

La función *as.numeric()* se utiliza para convertir un objeto en un número decimal.

Ejemplo 6:

```
cadena <- "3.1416"
número <- as.numeric(cadena)
número

## [1] 3.1416
```

- **as.integer()**: conversión a entero

La función *as.integer()* se utiliza para convertir un objeto en un número entero.

Ejemplo 7:

```
decimal <- 8.99
entero <- as.integer(decimal)
entero

## [1] 8
```

- **as.logical():** conversión a lógico

La función *as.logical()* se utiliza para convertir un objeto en un valor lógico (TRUE o FALSE).

Ejemplo 8:

```
número <- 0
lógico <- as.logical(número)
lógico

## [1] FALSE
```

- **factor()**: conversión a factor

La función *factor()* se utiliza para convertir un vector de caracteres en un factor. Los factores son útiles cuando se trabaja con variables categóricas.

Ejemplo 9:

```
categorías <- c("Rojo", "Verde", "Azul")
factores <- factor(categorías)
factores

## [1] Rojo  Verde Azul
## Levels: Azul Rojo Verde
```

- **as.Date()**: conversión a fecha

La función *as.Date()* se utiliza para convertir un objeto en una fecha. Podéis especificar el formato de fecha deseado utilizando el argumento format.

Ejemplo 10:

```
fecha <- "2023-08-08"
fecha_date <- as.Date(fecha, format = "%Y-%m-%d")
fecha

## [1] "2023-08-08"
```

Es importante tener en cuenta que la conversión de tipos puede provocar pérdida de información en algunos casos (por ejemplo, la conversión de números a enteros elimina los decimales), por lo que es necesario asegurar que la conversión es coherente.

5. Estructuras de datos

En **R**, las estructuras de datos son fundamentales para almacenar y manipular información. Estas estructuras se utilizan para organizar datos de manera eficiente y realizar análisis de datos de manera efectiva. A continuación, exploraremos las estructuras de datos más comunes en **R**.

5.1. Vectores

Los vectores son la estructura de datos más básica en **R** y pueden contener elementos del mismo tipo de datos, como números, caracteres o lógicos. Se crean utilizando la función *c()* (concatenar).

Ejemplo 11. Cómo crear vectores

```
#Vectores de datos de pacientes: edad, presión arterial sistólica y
diastólica
edades <- c(35, 42, 28, 56, 65)
presión_sistólica <- c(120, 130, 118, 140, 155)
presión_sistólica

## [1] 120 130 118 140 155

presión_diastólica <- c(80, 85, 72, 90, 98)
presión_diastólica

## [1] 80 85 72 90 98
```

5.2. Listas

Las listas son estructuras de datos que pueden contener elementos de diferentes tipos (números, caracteres, otros vectores, listas, etc.). Se crean utilizando la función *list()*.

Ejemplo 12. Cómo crear listas

```
#Lista de detalles de paciente y resultados de análisis
paciente_1 <- list(nombre = "María", edad = 35, género = "Femenino")
análisis_1 <- list(presión_sistólica = 120, presión_diastólica = 80)

paciente_2 <- list(nombre = "Juan", edad = 42, género = "Masculino")
análisis_2 <- list(presión_sistólica = 130, presión_diastólica = 85)

estudio <- list(paciente_1 = paciente_1, análisis_1 = análisis_1,
                paciente_2 = paciente_2, análisis_2 = análisis_2)
estudio
```

```
## $paciente_1
## $paciente_1$nombre
## [1] "María"
##
## $paciente_1$edad
## [1] 35
##
## $paciente_1$género
## [1] "Femenino"
##
##
## $análisis_1
## $análisis_1$presión_sistólica
## [1] 120
##
## $análisis_1$presión_diastólica
## [1] 80
##
##
## $paciente_2
## $paciente_2$nombre
## [1] "Juan"
##
## $paciente_2$edad
## [1] 42
##
## $paciente_2$genero
## [1] "Masculino"
##
##
## $analisis_2
## $analisis_2$presion_sistolica
## [1] 130
##
## $analisis_2$presion_diastolica
## [1] 85
```

5.3. Matrices

Las matrices son estructuras bidimensionales que contienen elementos del mismo tipo de datos (generalmente numéricos). Se crean utilizando la función *matrix()*. Las matrices se caracterizan por tener filas y columnas.

Ejemplo 13. Cómo crear matrices

```
#Matriz de recuento de células por tipo en diferentes muestras
muestras <- c("Muestra_A", "Muestra_B", "Muestra_C")
tipos_células <- c("Linfocitos", "Neutrófilos", "Eosinófilos")

matriz_células <- matrix(c(1200, 950, 320,
                           1050, 890, 280,
                           1300, 1100, 310), nrow = 3,
                         dimnames = list(muestras, tipos_células))
matriz_células

##           Linfocitos Neutrófilos Eosinófilos
## Muestra_A       1200        1050        1300
## Muestra_B        950         890        1100
## Muestra_C        320         280         310
```

Es importante tener en cuenta que la elección entre listas y matrices en **R** depende de la naturaleza de los datos y del tipo de operaciones para realizar. Las listas son flexibles y versátiles, ideales para datos heterogéneos y manipulaciones personalizadas, mientras que las matrices son eficientes y adecuadas para datos homogéneos y cálculos matriciales. En muchos casos, es conveniente utilizar ambas estructuras en conjunto para aprovechar sus ventajas respectivas.

5.4. Data frames

Los data frames son estructuras de datos similares a las tablas de una base de datos y se utilizan para almacenar datos en forma de filas y columnas. Cada columna de un data frame puede contener datos de diferentes tipos. Se crean utilizando la función *data.frame()*. Los data frames son ideales para trabajar con conjuntos de datos estructurados.

Ejemplo 14. Cómo crear data frames

```
#Data frame de datos de pacientes
pacientes <- data.frame(ID = c(1, 2, 3, 4, 5),
                        Nombre = c("Ana", "Luis", "María", "Carlos",
"Elena"),
                        Edad = c(45, 32, 28, 55, 40),
                        Diagnóstico = c("Diabetes", "Hipertensión", "Sano",
"Hipertensión", "Asma"))

#Acceso a columnas del data frame
nombres_pacientes <- pacientes$Nombre
nombres_pacientes
```

```
## [1] "Ana"    "Luis"   "María"  "Carlos" "Elena"

edades_pacientes <- pacientes$Edad
edades_pacientes

## [1] 45 32 28 55 40

diagnósticos <- pacientes$Diagnóstico
diagnósticos

## [1] "Diabetes"     "Hipertensión" "Sano"         "Hipertensión" "Asma"
```

Estas estructuras de datos son esenciales en **R** y se utilizan en una variedad de aplicaciones, desde análisis de datos simples hasta manipulación de datos compleja. Comprender cómo trabajar con vectores, listas, matrices y data frames es fundamental para el análisis de datos en **R**.

6. Instrucciones básicas del lenguaje R

6.1. Instrucciones condicionales

Las estructuras de control de flujo condicional if y else se utilizan para tomar decisiones en función de una condición dada.

En **R**, las instrucciones condicionales se definen según el siguiente estándar: *if(condition_1) { statements } else { statements }*.

Ejemplo 15. Instrucciones condicionales simples

```
#Ejemplo de condición (if, else)
edad_paciente <- 60
presión_arterial <- 140

if (edad_paciente > 50 && presión_arterial > 130) {
  diagnóstico <- "El paciente debe estar vigilado periódicamente."
} else {
  diagnóstico <- "El paciente está en buen estado de salud."
}

cat("Diagnóstico:", diagnóstico, "\n")

## Diagnóstico: El paciente debe estar vigilado periódicamente.
```

En este ejemplo, se evalúa la edad y la presión arterial de un paciente. Si el paciente tiene más de 50 años y su presión arterial es mayor que 130, se emite un mensaje de monitoreo cercano; de lo contrario, se considera que está en buen estado de salud.

Ejemplo 16. Cálculo IMC

Calculamos el índice de masa corporal de un individuo (IMC):

```
peso<-50 #asignamos el peso en kg
altura<-1.60 #asignamos altura en metros
IMC<-peso/((altura)^2) #valor del índice de masa corporal
IMC

## [1] 19.53125
```

Si quisiéramos catalogar el estado del individuo en función de su IMC haríamos:

```
if (IMC<19) {
  print ("Usted está por debajo del peso recomendado")
} else {
  if (IMC<25)
  print("Su peso es correcto")
    else if (IMC<=27)
      print("Usted padece sobrepeso")
    else if (IMC<=34)
      print ("Usted padece obesidad leve")
    else if (IMC<=39)
      print ("Usted padece obesidad severa")
    else if(IMC<=50)
        print("Usted padece obesidad mórbida")
      else
        print("Usted padece obesidad extrema")
  }

## [1] "Su peso es correcto"
```

Nota.- Los índices y las catalogaciones se han extraído de https://hospitalcruzrojacordoba.es/obesidad-nutricion-cordoba/calcular-imc-indice-de-masa-corporal-sobrepeso/

En el caso de que sea necesario tomar una decisión en función del valor de una variable, utilizaremos la instrucción condicional switch.

switch(expression, value_1, ... value_n)

Ejemplo 17. Uso múltiples condiciones

Siguiendo con el ejemplo anterior, en función de la catalogación del IMC de un individuo, se recomendaría un determinado tratamiento:

```
diagnóstico<-"obesidad"
switch(diagnóstico,"normal"="Correcto","sobrepeso"="Vigilar
dieta","obesidad"="Requiere tratamiento médico")

## [1] "Requiere tratamiento médico"
```

Ejemplo 18. Uso múltiples condiciones

```
#Ejemplo de función switch()
diagnóstico <- "Diabetes"
tratamiento <- switch(diagnóstico,
  "Diabetes" = "Insulina y control de la dieta",
  "Hipertensión" = "Medicamentos para la presión arterial",
  "Asma" = "Inhaladores y control de alérgenos",
  "Sano" = "Mantener hábitos saludables",
  "Otro" = "Evaluar caso por caso"
)

cat("Tratamiento para el diagnóstico", diagnóstico, "es:", tratamiento, "\n")

## Tratamiento para el diagnóstico Diabetes es: Insulina y control de la
dieta
```

6.2. Instrucciones iterativas (bucles)

Para ejecutar un conjunto de instrucciones repetidamente en **R**, principalmente utilizaremos *for* y *while* siguiendo los siguientes estándares:

for(variable in objeto iterable) { * # código }*

Ejemplo 19. Uso instrucciones repetitivas (for)

```
#Ejemplo de bucle (for)
pacientes <- c("Ana", "Luis", "María", "Carlos", "Elena")
for (paciente in pacientes) {
  cat("Realizando seguimiento para el paciente:", paciente, "\n")
}

## Realizando seguimiento para el paciente: Ana
## Realizando seguimiento para el paciente: Luis
## Realizando seguimiento para el paciente: María
## Realizando seguimiento para el paciente: Carlos
## Realizando seguimiento para el paciente: Elena
```

Otro formato para representar las instrucciones repetitivas es el siguiente:

while(condition) { # código }

Ejemplo 20. Uso instrucciones repetitivas (while)

```
#Ejemplo de bucle (while)
contador <- 1
while (contador <= length(pacientes)) {
  cat("Realizando seguimiento para el paciente:", pacientes[contador], "\n")
  contador <- contador + 1
}
```

```
## Realizando seguimiento para el paciente: Ana
## Realizando seguimiento para el paciente: Luis
## Realizando seguimiento para el paciente: María
## Realizando seguimiento para el paciente: Carlos
## Realizando seguimiento para el paciente: Elena
```

Ejemplo 21:

Consideremos ahora el conjunto de datos *anorexia* del paquete *MASS* y realicemos algunas consultas sobre estos datos, por ejemplo, de los 72 pacientes tratados, ¿cuántos de ellos han perdido peso después de realizar algún tratamiento?

```
library(MASS)
data("anorexia")
prewt<-c(anorexia$Prewt)
postwt<-c(anorexia$Postwt)
result<-c(postwt-prewt)
dif_peso<-0
for(i in 1:length(result))
  if (result[i] < 0)
    dif_peso<-dif_peso+1
 cat("La diferencia de peso es", dif_peso, "\n")

## La diferencia de peso es 29
```

7. Funciones en R

Las funciones son bloques de instrucciones que realizan una determinada acción, de manera que pueden ser reutilizadas en diversos programas. En **R**, usualmente, se utilizan funciones predefinidas para propósitos específicos. También es posible definir funciones personalizadas por el usuario. Veamos, brevemente, los dos tipos.

7.1. Funciones personalizadas por el usuario

En **R**, es posible definir funciones personalizadas para realizar tareas específicas. Para definir una función, se utiliza la palabra clave **function**. Las funciones pueden tener argumentos que actúan como parámetros de entrada. Estos argumentos pueden ser utilizados en el cuerpo de la función para realizar operaciones. Algunos argumentos pueden tener valores predeterminados (argumentos opcionales). Por otra parte, las funciones pueden devolver valores utilizando la palabra clave **return** y pueden devolver un único valor o múltiples valores en forma de lista o data frame.

La notación estándar sería:

namefunction <- function(arg1, arg2, ...) { statements return(object) }.

Posteriormente, para utilizar la función creada, es suficiente con invocarla pasando los parámetros especificados en la definición *(namefunction(arg1,arg2,..))*.

Ejemplo 22. Uso de funciones

```
#Función que calcula el IMC (Índice de Masa Corporal)
calcular_imc <- function(peso, altura) {
  imc <- peso / (altura^2)
  return(imc)
}

#Llamada a la función con argumentos
peso <- 70
altura <- 1.75
mi_imc <- calcular_imc(peso, altura)
cat("Mi IMC es:", mi_imc, "\n")

## Mi IMC es: 22.85714
```

El objetivo de este tipo de funciones es, básicamente, reutilizar código, hacer que este sea más modular y legible, así como encapsular bloques de código. Las funciones también permiten automatizar tareas repetitivas.

Ejemplo 23. Uso de funciones con diferentes tipos de retorno

```
#Función para calcular el IMC
calcular_imc <- function(peso, altura) {
  imc <- peso / (altura^2)
  return(imc)
}

#Función para clasificar el IMC en categorías
clasificar_imc <- function(imc) {
  if (imc < 18.5) {
    return("Bajo peso")
  } else if (imc >= 18.5) {
    return("Normal")
  } else if (imc < 29.9) {
    return("Sobrepeso")
  } else {
    return("Obesidad")
  }
}

#Datos de pacientes (peso en kg y altura en metros)
pacientes <- data.frame(
  Nombre = c("Ana", "Luis", "María", "Carlos", "Elena"),
  Peso = c(58, 70, 80, 90, 75),
  Altura = c(1.65, 1.75, 1.70, 1.80, 1.68)
)

#Calcular y clasificar el IMC para cada paciente
pacientes$IMC <- sapply(pacientes$Peso, function(x) {
```

```
  altura <- pacientes$Altura[pacientes$Peso == x]
  return(calcular_imc(x, altura))
})

pacientes$Categoría_IMC <- sapply(pacientes$IMC, clasificar_imc)

#Mostrar el data frame con los resultados
print(pacientes)

##   Nombre Peso Altura      IMC Categoría_IMC
## 1    Ana   58   1.65 21.30395        Normal
## 2   Luis   70   1.75 22.85714        Normal
## 3  María   80   1.70 27.68166        Normal
## 4 Carlos   90   1.80 27.77778        Normal
## 5  Elena   75   1.68 26.57313        Normal
```

7.2. Funciones propias de R

R ofrece una amplia variedad de funciones propias que cubren una amplia gama de tareas, desde operaciones numéricas hasta manipulación de caracteres y estadísticas. A continuación, veremos la lista de algunas de las funciones más importantes y útiles en **R**, agrupadas en categorías:

7.3.1. Funciones Numéricas

- **sum()**: suma de elementos en un vector.
- **mean()**: media aritmética de un vector numérico.
- **median()**: mediana de un vector numérico.
- **min()** y **max()**: Valor mínimo y máximo en un vector numérico.
- **abs()**: valor absoluto de un número o un vector.
- **sqrt()**: raíz cuadrada de un número o un vector.
- **round()**: redondea números a un número específico de decimales.

Ejemplo 24. Uso de funciones numéricas

En el siguiente ejemplo, utilizamos la función *mean()*:

```
glucosa <- c(125, 140, 95, 160, 110)
media_glucosa <- mean(glucosa)
cat("La media de glucosa en sangre es:", media_glucosa, "\n")

## La media de glucosa en sangre es: 126
```

En el siguiente ejemplo, utilizamos la función *round()*:

```
#Niveles de glucosa en sangre de pacientes (en mg/dL)
glucosa <- c(125.456, 140.372, 95.879, 160.621, 110.934)

#Redondear los niveles de glucosa a 2 decimales
glucosa_redondeada <- round(glucosa, digits = 2)
```

```
cat("Niveles de glucosa redondeados:", glucosa_redondeada, "\n")

## Niveles de glucosa redondeados: 125.46 140.37 95.88 160.62 110.93
```

7.3.2. Funciones de caracteres

Algunas de las funciones más importantes aplicadas a cadenas de caracteres son las siguientes:

- **paste()**: combina vectores de caracteres en una única cadena.
- **substr()**: extrae una subcadena de un vector de caracteres.
- **nchar()**: cuenta el número de caracteres en un vector de caracteres.
- **tolower()** y **toupper()**: convierte caracteres a minúsculas o mayúsculas.
- **grep()**: busca patrones de texto en un vector de caracteres.

Ejemplo 25. Uso de funciones de caracteres

```
#Busca pacientes con un diagnóstico determinado
diagnósticos <- c("Diabetes", "Hipertensión", "Sano", "Hipertensión", "Asma")
diagnóstico_buscado <- "Hipertensión"
pacientes_hipertensión <- nombres[grep(diagnóstico_buscado, diagnósticos)]
cat("Pacientes con hipertensión:", paste(pacientes_hipertension, collapse =
", "), "\n")

## Pacientes con hipertensión: Luis, Carlos
```

7.3.3. Funciones estadísticas

Algunas de las funciones estadísticas más comunes son las siguientes:

- **cor()**: calcula la correlación entre dos vectores numéricos.
- **var()**: calcula la varianza de un vector numérico.
- **sd()**: calcula la desviación estándar de un vector numérico.
- **quantile()**: calcula los cuantiles de un vector numérico.
- **hist()**: crea un histograma a partir de un vector numérico.
- **lm()**: ajusta un modelo de regresión lineal.

Ejemplo 25. Uso de funciones estadísticas

```
varianza_glucosa <- var(glucosa)
cat("La varianza de glucosa en sangre es:", varianza_glucosa, "\n")

## La varianza de glucosa en sangre es: 634.3987
```

7.3.4. Funciones de manipulación de datos

Las funciones más utilizadas para el tratamiento y la manipulación de datos son las siguientes:

- **subset()**: subconjunto de un data frame basado en condiciones.
- **merge()**: combina data frames por columnas o filas.
- **aggregate()**: realiza agregaciones de datos en un data frame.
- **reshape()**: cambia la forma de un data frame (pivotaje).
- **transform()**: agrega nuevas columnas a un data frame.

Ejemplo 26. Uso de funciones de manipulación de datos

```
#Combina dos data frames con información de pacientes.
pacientes_df1 <- data.frame(ID = 1:5, Diagnóstico = diagnósticos)
pacientes_df2 <- data.frame(ID = 1:5, Edad = edades)
pacientes_combinados <- merge(pacientes_df1, pacientes_df2, by = "ID")
cat("Pacientes combinados:\n")

## Pacientes combinados:

print(pacientes_combinados)

##   ID  Diagnóstico Edad
## 1  1     Diabetes   45
## 2  2 Hipertensión   32
## 3  3         Sano   28
## 4  4 Hipertensión   55
## 5  5         Asma   40
```

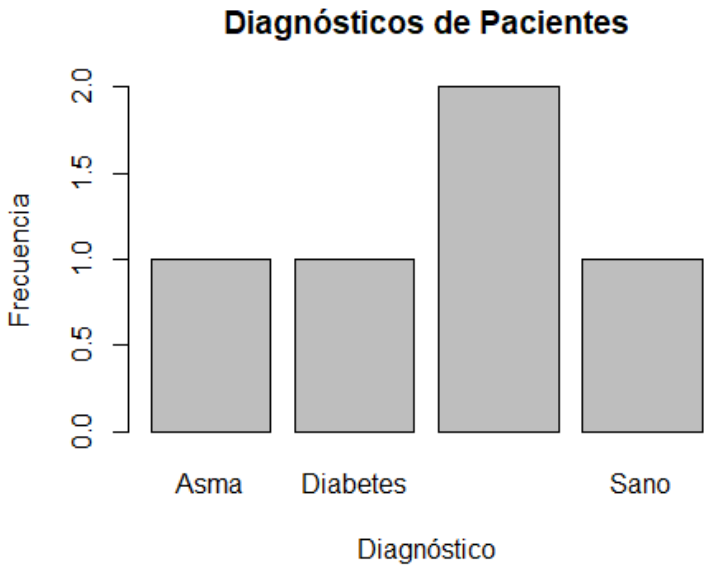
7.3.5. Funciones de gráficos

Las funciones de gráficos más comunes son las siguientes:

- **plot()**: crea gráficos básicos, como diagramas de dispersión y líneas.
- **hist()**: crea histogramas.
- **barplot()**: crea gráficos de barras.
- **boxplot()**: crea diagramas de caja.
- **ggplot2()**: un paquete popular para crear gráficos más avanzados y personalizados.

Ejemplo 27. Uso de funciones de creación de gráficos

```
#Crea un gráfico de barras para los diagnósticos de pacientes.
diagnósticos <- c("Diabetes", "Hipertensión", "Sano", "Hipertensión", "Asma")
barplot(table(diagnósticos), main = "Diagnósticos de pacientes", xlab =
"Diagnóstico", ylab = "Frecuencia")
```



7.3.6. Funciones para estadística descriptiva

Las funciones más comunes para trabajar la estadística descriptiva son las siguientes:

- **summary()**: proporciona un resumen estadístico de un objeto.
- **table()**: crea una tabla de frecuencias.
- **quantile()**: calcula los cuantiles de un vector numérico.
- **t.test()**: realiza una prueba t para comparar dos grupos.
- **chisq.test()**: realiza una prueba de ji al cuadrado para tablas de contingencia.

Ejemplo 27. Uso de funciones de estadística descriptiva

```
#Realiza una prueba t para comparar dos grupos de pacientes.
grupo_a <- c(120, 130, 118, 140, 155)
grupo_b <- c(140, 145, 132, 152, 160)
resultado_prueba_t <- t.test(grupo_a, grupo_b)
cat("Resultado de la prueba t:\n")

## Resultado de la prueba t:

print(resultado_prueba_t)

##
##  Welch Two Sample t-test
##
## data:  grupo_a and grupo_b
## t = -1.5777, df = 7.1885, p-value = 0.1575
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -32.879209   6.479209
## sample estimates:
## mean of x mean of y
##     132.6     145.8
```

7.3.7. Funciones de manipulación de tiempo y fechas

Las funciones más utilizadas en la gestión de fechas son las siguientes:

- **Sys.Date()**: obtiene la fecha actual.
- **Sys.time()**: obtiene la fecha y hora actual.
- **format()**: formatea fechas y horas.
- **difftime()**: calcula la diferencia entre dos fechas o horas.

Ejemplo 28. Uso de funciones de manipulación de fechas

```
#Crear un vector de fechas en formato de texto
fechas <- c("2023-08-01", "2023-08-10", "2023-08-15")

#Convertir las fechas de texto a objetos Date
fechas_date <- as.Date(fechas)

#Calcular la diferencia en días entre dos fechas
fecha_inicio <- as.Date("2023-08-01")
fecha_fin <- as.Date("2023-08-15")

diferencia_días <- fecha_fin - fecha_inicio

#Imprimir las fechas y la diferencia en días
cat("Fechas en formato Date:\n")

## Fechas en formato Date:

print(fechas_date)

## [1] "2023-08-01" "2023-08-10" "2023-08-15"

cat("Diferencia en días entre", fecha_inicio, "y", fecha_fin, "es:",
diferencia_días, "días\n")

## Diferencia en días entre 19570 y 19584 es: 14 días
```

Ejemplo 29. Uso de funciones en el dataset 'anorexia'

A continuación, aplicaremos algunas de estas funciones a un conjunto de datos, *anorexia* del paquete *MASS*, a partir del cual queremos saber cuál es la media de los pesos de todos los pacientes antes del tratamiento, y cómo ha variado esta media después de los tratamientos. Para ello, haríamos lo siguiente:

```
library(MASS)
##Conjunto de datos anorexia. Variación de la media de los pesos antes y
después del tratamiento
mean(anorexia$Prewt) ##media pesos antes tratamiento

## [1] 82.40833
```

```
mean(anorexia$Postwt) ##media pesos después tratamiento

## [1] 85.17222
```

Después del tratamiento realizado la media ha aumentado, es decir, del resultado se desprende que ha habido un aumento de la media de pesos después del tratamiento.

Ejemplo 30. Uso de diferentes funciones

En el siguiente ejemplo definimos una función personalizada por el usuario y otra función, *matrix()*, específica para la creación de matrices:

```
#Función para calcular el IMC de un paciente
calcular_imc <- function(peso, altura) {
  imc <- peso / (altura^2)
  return(imc)
}

#Datos de pacientes (peso en kg y altura en metros)
pacientes <- c("Ana", "Luis", "María", "Carlos", "Elena")
peso <- c(58, 70, 80, 90, 75)
altura <- c(1.65, 1.75, 1.70, 1.80, 1.68)

#Crear una matriz vacía para almacenar los resultados
matriz_imc <- matrix(nrow = length(pacientes), ncol = 2)
colnames(matriz_imc) <- c("Paciente", "IMC")

#Calcular el IMC para cada paciente y almacenar los resultados en la matriz
for (i in 1:length(pacientes)) {
  nombre <- pacientes[i]
  imc <- calcular_imc(peso[i], altura[i])
  matriz_imc[i, ] <- c(nombre, imc)
}

#Mostrar la matriz con los resultados
print(matriz_imc)

##      Paciente IMC
## [1,] "Ana"    "21.3039485766759"
## [2,] "Luis"   "22.8571428571429"
## [3,] "María"  "27.681660899654"
## [4,] "Carlos" "27.7777777777778"
## [5,] "Elena"  "26.5731292517007"
```

Ejemplo 31. Aplicación del algoritmo Needleman-Wunsch

Una aplicación de las funciones trabajadas anteriormente que puede ser interesante en el contexto de alineamiento de secuencias utilizando el algoritmo Needleman-Wunsch en **R**, es el siguiente ejemplo en el que utilizaremos secuencias de caracteres en lugar de secuencias

de aminoácidos y puntuaciones de coincidencia y penalización de gaps arbitrarias para ilustrar el proceso. Veamos una posible aproximación del código en **R**:

```
#Función para realizar el alineamiento de secuencias utilizando el algoritmo
Needleman-Wunsch
needleman_wunsch <- function(seq1, seq2, match_score, mismatch_penalty,
gap_penalty) {
  n <- nchar(seq1)
  m <- nchar(seq2)

  #Crear una matriz de puntuación y una matriz de seguimiento para almacenar
las direcciones
  score_matrix <- matrix(0, n + 1, m + 1)
  direction_matrix <- matrix(0, n + 1, m + 1)

  #Inicializar la matriz de puntuación
  for (i in 1:(n + 1)) {
    score_matrix[i, 1] <- (i - 1) * gap_penalty
  }
  for (j in 1:(m + 1)) {
    score_matrix[1, j] <- (j - 1) * gap_penalty
  }

  #Llenar la matriz de puntuación y la matriz de dirección
  for (i in 2:(n + 1)) {
    for (j in 2:(m + 1)) {
      match <- ifelse(substr(seq1, i - 1, i - 1) == substr(seq2, j - 1, j -
1), match_score, mismatch_penalty)
      diagonal <- score_matrix[i - 1, j - 1] + match
      left <- score_matrix[i, j - 1] + gap_penalty
      up <- score_matrix[i - 1, j] + gap_penalty
      score_matrix[i, j] <- max(diagonal, left, up)
      direction <- which.max(c(diagonal, left, up))
      direction_matrix[i, j] <- direction
    }
  }

  # Realizar el seguimiento de la dirección para obtener el alineamiento
  i <- n + 1
  j <- m + 1
  align_seq1 <- ""
  align_seq2 <- ""

  while (i > 1 || j > 1) {
    direction <- direction_matrix[i, j]

    if (direction == 1) {
      align_seq1 <- paste(substr(seq1, i - 1, i - 1), align_seq1, sep = "")
      align_seq2 <- paste(substr(seq2, j - 1, j - 1), align_seq2, sep = "")
```

```
      i <- i - 1
      j <- j - 1
    } else if (direction == 2) {
      align_seq1 <- paste("-", align_seq1, sep = "")
      align_seq2 <- paste(substr(seq2, j - 1, j - 1), align_seq2, sep = "")
      j <- j - 1
    } else {
      align_seq1 <- paste(substr(seq1, i - 1, i - 1), align_seq1, sep = "")
      align_seq2 <- paste("-", align_seq2, sep = "")
      i <- i - 1
    }
  }

  return(list(seq1 = align_seq1, seq2 = align_seq2))
}

#Secuencias de ejemplo
seq1 <- "LAASTYV"
seq2 <- "NAASV"

#Parámetros de puntuación
match_score <- 2
mismatch_penalty <- -1
gap_penalty <- -2

#Realizar el alineamiento
alignment <- needleman_wunsch(seq1, seq2, match_score, mismatch_penalty,
gap_penalty)

#Mostrar los resultados del alineamiento
cat("Secuencia 1 alineada:", alignment$seq1, "\n")

## Secuencia 1 alineada: LAASTYV

cat("Secuencia 2 alineada:", alignment$seq2, "\n")

## Secuencia 2 alineada: NAAS--V
```

La función *needleman_wunsch* implementa el **algoritmo Needleman-Wunsch** para alinear dos secuencias de caracteres. Se han utilizado puntuaciones de coincidencia (match_score), penalización por desajuste (mismatch_penalty) y penalización por gaps (gap_penalty) para ilustrar el proceso de alineamiento. El resultado muestra las secuencias alineadas, pero hay que tener en cuenta que es solo una posible solución y que las puntuaciones y penalizaciones utilizadas son arbitrarias.

8. Acceso a bases de datos en R

La gestión y el acceso a datos con **R**, supone el manejo de diferentes tipos de archivos. Aparte de los formatos más habituales (*.csv, etc.), en la propia gestión de proyectos más complejos es habitual el acceso a bases de datos. Este determinado formato requiere el uso de librerías específicas y, también, el uso del lenguaje SQL que es el estándar de gestión de bases de datos. En este apartado, básicamente, veremos cómo trabajar con bases de datos en **R**, desde la conexión a bases de datos externas hasta la consulta y manipulación de datos utilizando paquetes específicos. No es objetivo de este apartado el aprendizaje del lenguaje SQL, sino que solo se mostrarán ejemplos prácticos de su uso para mostrar el acceso y la gestión básica a base de datos con **R**.

8.1. Conexión a base de datos en R

En numerosas aplicaciones informáticas, los conjuntos de datos residen en sistemas gestores de bases de datos como es el caso de MySQL, PostgreSQL o SQLite. **R** ofrece diversas maneras de conectarse a estas bases de datos a través de paquetes generales como **ODBC** y **DBI** o más específicos como **RSQLite**.

El tipo de conexión genérica, **ODBC** (Open Database Connectivity) actúa como interfaz entre el lenguaje de programación (**R**, **Python**, etc.) y el sistema gestor de bases de datos. En este sentido, el paquete **RODBC** permite conectar **R** con PostgreSQL, MySQL o SQLite, entre otros.

Se recomienda consultar más detalles del paquete RODBC en: https://cran.r-project.org/web/packages/RODBC/RODBC.pdf

Existe otra forma de conectar **R** con sistemas gestores de bases de datos que es la llamada **DBI** (Database Interface) que tiene una funcionalidad similar al ODBC, pero un poco más sencilla y también permite la conexión con sistemas de gestión de bases de datos (DBMS) como MariaDB, además de los anteriormente mencionados. DBI actúa como una interfaz entre **R** y diversas bases de datos, permitiendo a los usuarios interactuar con bases de datos desde **R** de manera eficiente y estandarizada. Es interesante consultar https://dbi.r-dbi.org/ para profundizar en las posibilidades del paquete **DBI**.

Para la conexión a una base de datos desde **R**, es imprescindible la siguiente pauta: a. Instalación de paquetes de conexión a base de datos. b. Establecimiento de la conexión y configuración de los parámetros. c. Desconexión de la base de datos.

8.2. Consulta y manipulación de datos

Una vez realizada la conexión, comienza la gestión con la base de datos que puede suponer la consulta de información, así como diversas actualizaciones. Para ello, es básico el uso de SQL (Structured Query Language) para extraer información específica de la base de datos.

El paquete **dplyr** es una herramienta muy potente para manipular datos en **R**, pudiendo realizar la conexión a base de datos para realizar operaciones de filtrado, selección, agrupación y listados generales de la base de datos, sin necesidad de tener conocimientos exhaustivos de SQL. El paquete **dbplyr** es el soporte de la base de datos para **dplyr** que

permite utilizar tablas de bases de datos como si fueran conjuntos de datos (**datasets**) locales. *A priori*, no es necesario cargar dbplyr, ya que dplyr lo carga automáticamente.

Por otra parte, también el paquete **sqldf** nos permite realizar consultas sobre un sistema gestor de base de datos.

A continuación, instalaremos el paquete **DBI** (*install.packages("DBI")* o ***Tools > Install Packages*** y seleccionar *DBI*) y lo cargaremos al sistema. De forma análoga, hemos de instalar el paquete **RSQlite**.

```
library(DBI)
library(RSQLite)
```

La gestión a base de datos se realizará con **SQLite**; será necesario instalar dicho paquete y cargarlo al sistema y, seguidamente, accederemos a las bases de datos contenidas en uno de los ficheros de RSQLite llamado *datasetsDb*, para ello ejecutamos las instrucciones siguientes:

```
db<-"RSQLite"::datasetsDb()
dbListTables(db) ##Listamos los conjuntos de datos

##  [1] "BOD"             "CO2"             "ChickWeight"      "DNase"
##  [5] "Formaldehyde"    "Indometh"        "InsectSprays"
"LifeCycleSavings"
##  [9] "Loblolly"        "Orange"          "OrchardSprays"
"PlantGrowth"
## [13] "Puromycin"       "Theoph"          "ToothGrowth"      "USArrests"
## [17] "USJudgeRatings"  "airquality"      "anscombe"         "attenu"
## [21] "attitude"        "cars"            "chickwts"         "esoph"
## [25] "faithful"        "freeny"          "infert"           "iris"
## [29] "longley"         "morley"          "mtcars"           "npk"
## [33] "pressure"        "quakes"          "randu"            "rock"
## [37] "sleep"           "stackloss"       "swiss"            "trees"
## [41] "warpbreaks"      "women"
```

Ejemplo 32. Gestión de base de datos con el dataset 'airquality'

Vamos a visualizar la información contenida en el conjunto de datos *airquality* que hace referencia a la calidad del aire hace algunos años (más información: https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/airquality.html):

```
dbReadTable(db,"airquality") #accedemos al conjunto de datos airquality y lo
guardamos en db

##     Ozone Solar.R Wind Temp Month Day
## 1      41     190  7.4   67     5   1
## 2      36     118  8.0   72     5   2
## 3      12     149 12.6   74     5   3
## 4      18     313 11.5   62     5   4
## 5      NA      NA 14.3   56     5   5
## 6      28      NA 14.9   66     5   6
```

```
## 7      23     299  8.6   65     5   7
## 8      19      99 13.8   59     5   8
## 9       8      19 20.1   61     5   9
## 10     NA     194  8.6   69     5  10
## 11      7      NA  6.9   74     5  11
## 12     16     256  9.7   69     5  12
## 13     11     290  9.2   66     5  13
## 14     14     274 10.9   68     5  14
## 15     18      65 13.2   58     5  15
## 16     14     334 11.5   64     5  16
## 17     34     307 12.0   66     5  17
## 18      6      78 18.4   57     5  18
## 19     30     322 11.5   68     5  19
## 20     11      44  9.7   62     5  20
## 21      1       8  9.7   59     5  21
## 22     11     320 16.6   73     5  22
## 23      4      25  9.7   61     5  23
## 24     32      92 12.0   61     5  24
## 25     NA      66 16.6   57     5  25
## 26     NA     266 14.9   58     5  26
## 27     NA      NA  8.0   57     5  27
## 28     23      13 12.0   67     5  28
## 29     45     252 14.9   81     5  29
## 30    115     223  5.7   79     5  30
## 31     37     279  7.4   76     5  31
## 32     NA     286  8.6   78     6   1
## 33     NA     287  9.7   74     6   2
## 34     NA     242 16.1   67     6   3
## 35     NA     186  9.2   84     6   4
## 36     NA     220  8.6   85     6   5
## 37     NA     264 14.3   79     6   6
## 38     29     127  9.7   82     6   7
## 39     NA     273  6.9   87     6   8
## 40     71     291 13.8   90     6   9
## 41     39     323 11.5   87     6  10
## 42     NA     259 10.9   93     6  11
## 43     NA     250  9.2   92     6  12
## 44     23     148  8.0   82     6  13
## 45     NA     332 13.8   80     6  14
## 46     NA     322 11.5   79     6  15
## 47     21     191 14.9   77     6  16
## 48     37     284 20.7   72     6  17
## 49     20      37  9.2   65     6  18
## 50     12     120 11.5   73     6  19
## 51     13     137 10.3   76     6  20
## 52     NA     150  6.3   77     6  21
## 53     NA      59  1.7   76     6  22
## 54     NA      91  4.6   76     6  23
## 55     NA     250  6.3   76     6  24
## 56     NA     135  8.0   75     6  25
```

```
## 57     NA     127  8.0   78     6  26
## 58     NA      47 10.3   73     6  27
## 59     NA      98 11.5   80     6  28
## 60     NA      31 14.9   77     6  29
## 61     NA     138  8.0   83     6  30
## 62    135     269  4.1   84     7   1
## 63     49     248  9.2   85     7   2
## 64     32     236  9.2   81     7   3
## 65     NA     101 10.9   84     7   4
## 66     64     175  4.6   83     7   5
## 67     40     314 10.9   83     7   6
## 68     77     276  5.1   88     7   7
## 69     97     267  6.3   92     7   8
## 70     97     272  5.7   92     7   9
## 71     85     175  7.4   89     7  10
## 72     NA     139  8.6   82     7  11
## 73     10     264 14.3   73     7  12
## 74     27     175 14.9   81     7  13
## 75     NA     291 14.9   91     7  14
## 76      7      48 14.3   80     7  15
## 77     48     260  6.9   81     7  16
## 78     35     274 10.3   82     7  17
## 79     61     285  6.3   84     7  18
## 80     79     187  5.1   87     7  19
## 81     63     220 11.5   85     7  20
## 82     16       7  6.9   74     7  21
## 83     NA     258  9.7   81     7  22
## 84     NA     295 11.5   82     7  23
## 85     80     294  8.6   86     7  24
## 86    108     223  8.0   85     7  25
## 87     20      81  8.6   82     7  26
## 88     52      82 12.0   86     7  27
## 89     82     213  7.4   88     7  28
## 90     50     275  7.4   86     7  29
## 91     64     253  7.4   83     7  30
## 92     59     254  9.2   81     7  31
## 93     39      83  6.9   81     8   1
## 94      9      24 13.8   81     8   2
## 95     16      77  7.4   82     8   3
## 96     78      NA  6.9   86     8   4
## 97     35      NA  7.4   85     8   5
## 98     66      NA  4.6   87     8   6
## 99    122     255  4.0   89     8   7
## 100    89     229 10.3   90     8   8
## 101   110     207  8.0   90     8   9
## 102    NA     222  8.6   92     8  10
## 103    NA     137 11.5   86     8  11
## 104    44     192 11.5   86     8  12
## 105    28     273 11.5   82     8  13
## 106    65     157  9.7   80     8  14
```

```
## 107    NA      64 11.5   79     8  15
## 108    22      71 10.3   77     8  16
## 109    59      51  6.3   79     8  17
## 110    23     115  7.4   76     8  18
## 111    31     244 10.9   78     8  19
## 112    44     190 10.3   78     8  20
## 113    21     259 15.5   77     8  21
## 114     9      36 14.3   72     8  22
## 115    NA     255 12.6   75     8  23
## 116    45     212  9.7   79     8  24
## 117   168     238  3.4   81     8  25
## 118    73     215  8.0   86     8  26
## 119    NA     153  5.7   88     8  27
## 120    76     203  9.7   97     8  28
## 121   118     225  2.3   94     8  29
## 122    84     237  6.3   96     8  30
## 123    85     188  6.3   94     8  31
## 124    96     167  6.9   91     9   1
## 125    78     197  5.1   92     9   2
## 126    73     183  2.8   93     9   3
## 127    91     189  4.6   93     9   4
## 128    47      95  7.4   87     9   5
## 129    32      92 15.5   84     9   6
## 130    20     252 10.9   80     9   7
## 131    23     220 10.3   78     9   8
## 132    21     230 10.9   75     9   9
## 133    24     259  9.7   73     9  10
## 134    44     236 14.9   81     9  11
## 135    21     259 15.5   76     9  12
## 136    28     238  6.3   77     9  13
## 137     9      24 10.9   71     9  14
## 138    13     112 11.5   71     9  15
## 139    46     237  6.9   78     9  16
## 140    18     224 13.8   67     9  17
## 141    13      27 10.3   76     9  18
## 142    24     238 10.3   68     9  19
## 143    16     201  8.0   82     9  20
## 144    13     238 12.6   64     9  21
## 145    23      14  9.2   71     9  22
## 146    36     139 10.3   81     9  23
## 147     7      49 10.3   69     9  24
## 148    14      20 16.6   63     9  25
## 149    30     193  6.9   70     9  26
## 150    NA     145 13.2   77     9  27
## 151    14     191 14.3   75     9  28
## 152    18     131  8.0   76     9  29
## 153    20     223 11.5   68     9  30
```

```
colnames(airquality) #mostramos las columnas del conjunto de datos
```

```
## [1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"

dbGetQuery(db,"Select * from airquality") #mostramos la información de db

##     Ozone Solar.R Wind Temp Month Day
## 1      41     190  7.4   67     5   1
## 2      36     118  8.0   72     5   2
## 3      12     149 12.6   74     5   3
## 4      18     313 11.5   62     5   4
## 5      NA      NA 14.3   56     5   5
## 6      28      NA 14.9   66     5   6
## 7      23     299  8.6   65     5   7
## 8      19      99 13.8   59     5   8
## 9       8      19 20.1   61     5   9
## 10     NA     194  8.6   69     5  10
## 11      7      NA  6.9   74     5  11
## 12     16     256  9.7   69     5  12
## 13     11     290  9.2   66     5  13
## 14     14     274 10.9   68     5  14
## 15     18      65 13.2   58     5  15
## 16     14     334 11.5   64     5  16
## 17     34     307 12.0   66     5  17
## 18      6      78 18.4   57     5  18
## 19     30     322 11.5   68     5  19
## 20     11      44  9.7   62     5  20
## 21      1       8  9.7   59     5  21
## 22     11     320 16.6   73     5  22
## 23      4      25  9.7   61     5  23
## 24     32      92 12.0   61     5  24
## 25     NA      66 16.6   57     5  25
## 26     NA     266 14.9   58     5  26
## 27     NA      NA  8.0   57     5  27
## 28     23      13 12.0   67     5  28
## 29     45     252 14.9   81     5  29
## 30    115     223  5.7   79     5  30
## 31     37     279  7.4   76     5  31
## 32     NA     286  8.6   78     6   1
## 33     NA     287  9.7   74     6   2
## 34     NA     242 16.1   67     6   3
## 35     NA     186  9.2   84     6   4
## 36     NA     220  8.6   85     6   5
## 37     NA     264 14.3   79     6   6
## 38     29     127  9.7   82     6   7
## 39     NA     273  6.9   87     6   8
## 40     71     291 13.8   90     6   9
## 41     39     323 11.5   87     6  10
## 42     NA     259 10.9   93     6  11
## 43     NA     250  9.2   92     6  12
## 44     23     148  8.0   82     6  13
## 45     NA     332 13.8   80     6  14
```

```
## 46     NA     322 11.5   79     6  15
## 47     21     191 14.9   77     6  16
## 48     37     284 20.7   72     6  17
## 49     20      37  9.2   65     6  18
## 50     12     120 11.5   73     6  19
## 51     13     137 10.3   76     6  20
## 52     NA     150  6.3   77     6  21
## 53     NA      59  1.7   76     6  22
## 54     NA      91  4.6   76     6  23
## 55     NA     250  6.3   76     6  24
## 56     NA     135  8.0   75     6  25
## 57     NA     127  8.0   78     6  26
## 58     NA      47 10.3   73     6  27
## 59     NA      98 11.5   80     6  28
## 60     NA      31 14.9   77     6  29
## 61     NA     138  8.0   83     6  30
## 62    135     269  4.1   84     7   1
## 63     49     248  9.2   85     7   2
## 64     32     236  9.2   81     7   3
## 65     NA     101 10.9   84     7   4
## 66     64     175  4.6   83     7   5
## 67     40     314 10.9   83     7   6
## 68     77     276  5.1   88     7   7
## 69     97     267  6.3   92     7   8
## 70     97     272  5.7   92     7   9
## 71     85     175  7.4   89     7  10
## 72     NA     139  8.6   82     7  11
## 73     10     264 14.3   73     7  12
## 74     27     175 14.9   81     7  13
## 75     NA     291 14.9   91     7  14
## 76      7      48 14.3   80     7  15
## 77     48     260  6.9   81     7  16
## 78     35     274 10.3   82     7  17
## 79     61     285  6.3   84     7  18
## 80     79     187  5.1   87     7  19
## 81     63     220 11.5   85     7  20
## 82     16       7  6.9   74     7  21
## 83     NA     258  9.7   81     7  22
## 84     NA     295 11.5   82     7  23
## 85     80     294  8.6   86     7  24
## 86    108     223  8.0   85     7  25
## 87     20      81  8.6   82     7  26
## 88     52      82 12.0   86     7  27
## 89     82     213  7.4   88     7  28
## 90     50     275  7.4   86     7  29
## 91     64     253  7.4   83     7  30
## 92     59     254  9.2   81     7  31
## 93     39      83  6.9   81     8   1
## 94      9      24 13.8   81     8   2
## 95     16      77  7.4   82     8   3
```

```
## 96     78      NA  6.9   86     8   4
## 97     35      NA  7.4   85     8   5
## 98     66      NA  4.6   87     8   6
## 99    122     255  4.0   89     8   7
## 100    89     229 10.3   90     8   8
## 101   110     207  8.0   90     8   9
## 102    NA     222  8.6   92     8  10
## 103    NA     137 11.5   86     8  11
## 104    44     192 11.5   86     8  12
## 105    28     273 11.5   82     8  13
## 106    65     157  9.7   80     8  14
## 107    NA      64 11.5   79     8  15
## 108    22      71 10.3   77     8  16
## 109    59      51  6.3   79     8  17
## 110    23     115  7.4   76     8  18
## 111    31     244 10.9   78     8  19
## 112    44     190 10.3   78     8  20
## 113    21     259 15.5   77     8  21
## 114     9      36 14.3   72     8  22
## 115    NA     255 12.6   75     8  23
## 116    45     212  9.7   79     8  24
## 117   168     238  3.4   81     8  25
## 118    73     215  8.0   86     8  26
## 119    NA     153  5.7   88     8  27
## 120    76     203  9.7   97     8  28
## 121   118     225  2.3   94     8  29
## 122    84     237  6.3   96     8  30
## 123    85     188  6.3   94     8  31
## 124    96     167  6.9   91     9   1
## 125    78     197  5.1   92     9   2
## 126    73     183  2.8   93     9   3
## 127    91     189  4.6   93     9   4
## 128    47      95  7.4   87     9   5
## 129    32      92 15.5   84     9   6
## 130    20     252 10.9   80     9   7
## 131    23     220 10.3   78     9   8
## 132    21     230 10.9   75     9   9
## 133    24     259  9.7   73     9  10
## 134    44     236 14.9   81     9  11
## 135    21     259 15.5   76     9  12
## 136    28     238  6.3   77     9  13
## 137     9      24 10.9   71     9  14
## 138    13     112 11.5   71     9  15
## 139    46     237  6.9   78     9  16
## 140    18     224 13.8   67     9  17
## 141    13      27 10.3   76     9  18
## 142    24     238 10.3   68     9  19
## 143    16     201  8.0   82     9  20
## 144    13     238 12.6   64     9  21
## 145    23      14  9.2   71     9  22
```

```
## 146    36     139 10.3   81     9  23
## 147     7      49 10.3   69     9  24
## 148    14      20 16.6   63     9  25
## 149    30     193  6.9   70     9  26
## 150    NA     145 13.2   77     9  27
## 151    14     191 14.3   75     9  28
## 152    18     131  8.0   76     9  29
## 153    20     223 11.5   68     9  30
```

Las anteriores instrucciones hacen referencia a la lectura del conjunto de datos y a la consulta genérica en SQL que permite mostrar toda la información de dicho conjunto de datos. Veamos algunos ejemplos de cómo podemos obtener más información basada en determinados criterios y utilizando el paquete **sqldf**.

Instalamos dicho paquete de la forma habitual, es decir, *install.packages("sqldf")* o bien ***Tools > Install packages*** y seleccionar *sqldf*, y cargamos el paquete al sistema:

```
library(sqldf)

## Loading required package: gsubfn

## Loading required package: proto
```

Nota.- Es posible que, según la versión instalada de **RStudio** se produzca algún requerimiento de instalación previa de otros paquetes.

Ejemplo 33. Gestión de bases de datos con sqldf()

A continuación, realizaremos diversas consultas a *airquality*:

1) Mostrar las siete primeras filas del conjunto de datos.

```
##Utilizamos sqldf para ejecutar las sentencias SQL
##Mostramos las primeras siete filas de airquality
sqldf("SELECT * FROM airquality LIMIT 7")

##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
## 7    23     299  8.6   65     5   7
```

2) Mostrar las cinco primeras filas de la columna Wind.

```
sqldf("SELECT Wind FROM airquality LIMIT 5")

##   Wind
## 1  7.4
## 2  8.0
```

```
## 3 12.6
## 4 11.5
## 5 14.3
```

3) Mostrar la tabla ordenada por la columna Month de forma ascendente (por defecto).

```
##Tabla ordenada por el campo Month de forma ascendente
sqldf("SELECT * FROM airquality ORDER BY Month ASC LIMIT 5")
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
```

4) Mostrar los datos de los registros cuyo valor de la columna Wind sea inferior a diez.

```
#Datos de airquality cuyo factor Wind es inferior a diez
sqldf("SELECT * FROM airquality where Wind<10 LIMIT 5")
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    23     299  8.6   65     5   7
## 4    NA     194  8.6   69     5  10
## 5     7      NA  6.9   74     5  11
```

5) Mostrar los datos del factor Ozone cuyo valor de Temp sea superior a setenta y Wind inferior a diez.

```
#Datos del factor Ozone cuyo factor Temp es superior setenta y Wind inferior
a diez
sqldf("SELECT Ozone FROM airquality where Wind<10 AND Temp>70 LIMIT 5")
```

```
##   Ozone
## 1    36
## 2     7
## 3   115
## 4    37
## 5    NA
```

6) Mostrar los datos del factor Ozone y Temp, teniendo en cuenta que Temp ha de estar en un rango entre setenta y noventa.

```
#Datos del factor Ozone y Temp, cuyo factor Temp esté en un rango entre
setenta y noventa
sqldf("SELECT Ozone,Temp FROM airquality where Temp in (70,90) LIMIT 5")
```

```
##   Ozone Temp
## 1    71   90
## 2    89   90
```

```
## 3   110   90
## 4    30   70
```

7) Mostrar los datos asociados al máximo y el mínimo de las temperaturas (Temp) registradas y también la media de temperaturas.

```
#Máximo y mínimo del factor Wind
sqldf("SELECT *,MAX(Wind) FROM airquality")

##   Ozone Solar.R Wind Temp Month Day MAX(Wind)
## 1    37     284 20.7   72     6  17      20.7

sqldf("SELECT *,MIN(Wind) FROM airquality")

##   Ozone Solar.R Wind Temp Month Day MIN(Wind)
## 1    NA      59  1.7   76     6  22       1.7

#Media de temperaturas registradas
sqldf("SELECT AVG(Temp) FROM airquality")

##   AVG(Temp)
## 1  77.88235
```

Además, del uso del paquete *sqldf* para gestionar el conjunto de datos, también podríamos utilizar las siguientes instrucciones que veremos en los siguientes ejemplos.

Ejemplo 34. Gestión bases de datos

```
library(dplyr)

##
## Attaching package: 'dplyr'

## The following object is masked from 'package:MASS':
##
##     select

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(dbplyr)

##
## Attaching package: 'dbplyr'

## The following objects are masked from 'package:dplyr':
##
##     ident, sql
```

```
library(DBI)

##conectamos RSQLite
db_airq<-DBI::dbConnect(RSQLite::SQLite(),":memory:") #cargamos en memoria
copy_to(db_airq,airquality) #copiamos la información en una variable
db_airq

## <SQLiteConnection>
##   Path: :memory:
##   Extensions: TRUE

tb_airq<-tbl(db_airq,"airquality") #guardamos la información de db_airq
head(tb_airq) #mostramos información

## # Source:   SQL [6 x 6]
## # Database: sqlite 3.41.2 [:memory:]
##   Ozone Solar.R  Wind  Temp Month   Day
##   <int>   <int> <dbl> <int> <int> <int>
## 1    41     190   7.4    67     5     1
## 2    36     118   8      72     5     2
## 3    12     149  12.6    74     5     3
## 4    18     313  11.5    62     5     4
## 5    NA      NA  14.3    56     5     5
## 6    28      NA  14.9    66     5     6

#Mostramos datos de airquality cuyo valor de Ozone es superior a 120
high_ozone<-tb_airq %>%
filter(Ozone>120)
head(high_ozone)

## # Source:   SQL [3 x 6]
## # Database: sqlite 3.41.2 [:memory:]
##   Ozone Solar.R  Wind  Temp Month   Day
##   <int>   <int> <dbl> <int> <int> <int>
## 1   135     269   4.1    84     7     1
## 2   122     255   4      89     8     7
## 3   168     238   3.4    81     8    25

#Mostramos los datos de ozono superior a 120 agrupados por Month
high_ozone_month<-tb_airq %>%
group_by(Month)
head(high_ozone_month)

## # Source:   SQL [6 x 6]
## # Database: sqlite 3.41.2 [:memory:]
## # Groups:   Month
##   Ozone Solar.R  Wind  Temp Month   Day
##   <int>   <int> <dbl> <int> <int> <int>
## 1    41     190   7.4    67     5     1
## 2    36     118   8      72     5     2
## 3    12     149  12.6    74     5     3
## 4    18     313  11.5    62     5     4
```

```
## 5    NA      NA  14.3    56     5     5
## 6    28      NA  14.9    66     5     6
```

De forma análoga a los ejemplos vistos anteriormente, veamos las siguientes consultas realizadas al conjunto de datos *esoph* (https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/esoph.html) utilizando la sintaxis de acceso a bases de datos.

Ejemplo 35. Gestión base de datos a dataset 'esoph'

Inicialmente, listamos los conjuntos de tablas existentes:

```
db<-"RSQLite"::datasetsDb()
dbListTables(db) ##Mostramos los conjuntos de datos
```

```
##  [1] "BOD"              "CO2"              "ChickWeight"      "DNase"
##  [5] "Formaldehyde"     "Indometh"         "InsectSprays"
"LifeCycleSavings"
##  [9] "Loblolly"         "Orange"           "OrchardSprays"
"PlantGrowth"
## [13] "Puromycin"        "Theoph"           "ToothGrowth"      "USArrests"
## [17] "USJudgeRatings"   "airquality"       "anscombe"         "attenu"
## [21] "attitude"         "cars"             "chickwts"         "esoph"
## [25] "faithful"         "freeny"           "infert"           "iris"
## [29] "longley"          "morley"           "mtcars"           "npk"
## [33] "pressure"         "quakes"           "randu"            "rock"
## [37] "sleep"            "stackloss"        "swiss"            "trees"
## [41] "warpbreaks"       "women"
```

1) Mostrar el número de filas y de columnas del conjunto de datos *esoph*.

```
ncol(esoph)
```

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

```
nrow(esoph)
```

```
## [1] 88
```

2) Mostrar los registros del conjunto de datos *esoph*.

```
#mostrar todos los datos de la tabla esoph
head(sqldf("SELECT * FROM esoph"))
```

```
##   agegp     alcgp    tobgp ncases ncontrols
## 1 25-34 0-39g/day 0-9g/day      0        40
## 2 25-34 0-39g/day    10-19      0        10
## 3 25-34 0-39g/day    20-29      0         6
## 4 25-34 0-39g/day      30+      0         5
## 5 25-34     40-79 0-9g/day      0        27
## 6 25-34     40-79    10-19      0         7
```

3) Mostrar las edades de aquellos pacientes que han realizado más de treinta controles.

```
#mostrar las edades de aquellos pacientes que han realizado más de treinta
controles
head(sqldf("SELECT agegp FROM esoph where ncontrols>30"))

##   agegp
## 1 25-34
## 2 35-44
## 3 35-44
## 4 45-54
## 5 45-54
## 6 55-64
```

4) Mostrar la media de los controles realizados.

```
#mostrar la media de controles realizados
sqldf("SELECT AVG(ncontrols) FROM esoph")

##   AVG(ncontrols)
## 1       8.806818
```

5) Mostrar los datos de *esoph* agrupados por franjas de edad.

```
#mostrar los datos de esoph agrupados por franjas de edad
sqldf("SELECT * FROM esoph GROUP BY agegp")

##   agegp     alcgp    tobgp ncases ncontrols
## 1 25-34 0-39g/day 0-9g/day      0        40
## 2 35-44 0-39g/day 0-9g/day      0        60
## 3 45-54 0-39g/day 0-9g/day      1        45
## 4 55-64 0-39g/day 0-9g/day      2        47
## 5 65-74 0-39g/day 0-9g/day      5        43
## 6   75+ 0-39g/day 0-9g/day      1        17
```

6) Mostrar los registros ordenados por la columna ncases de forma descendente.

```
#ordenar los registros por la variable ncases de forma descendente
head(sqldf("SELECT * FROM esoph ORDER BY ncases DESC"))

##   agegp  alcgp    tobgp ncases ncontrols
## 1 65-74  40-79 0-9g/day     17        17
## 2 55-64  40-79 0-9g/day      9        31
## 3 55-64 80-119 0-9g/day      9         9
## 4 55-64 80-119    10-19      8         7
## 5 45-54  40-79 0-9g/day      6        32
## 6 45-54 80-119    10-19      6         8
```

7) Mostrar los registros cuya columna ncontrols sea inferior a la media de controles y cuyo ncases esté entre cinco y quince.

```
#mostrar los registros cuyo ncontrols sea inferior a la media de controles y
cuyo ncases esté entre cinco y quince
```

```
sqldf("SELECT * FROM esoph where ncontrols < (SELECT AVG(ncontrols) FROM
esoph) AND ncases IN (5,15)")
```

```
##   agegp alcgp    tobgp ncases ncontrols
## 1 45-54 40-79      30+      5         2
## 2 55-64  120+ 0-9g/day      5         5
## 3 55-64  120+      30+      5         1
## 4 65-74 40-79    20-29      5         4
```

Los ejemplos anteriores nos muestran cómo acceder a un conjunto de datos, en particular tablas, y cómo mostrar la información que contienen mediante instrucciones de acceso a bases de datos. En otros casos, necesitaremos acceder a una base de datos con diversas tablas e, incluso, obtener información de diversas tablas relacionadas. A continuación, visualizaremos cómo establecer una conexión con una determinada base de datos que guardaremos en memoria, en este caso utilizaremos el paquete **tidyr** y cómo mostrar algunas consultas específicas a partir de las tablas que contiene.

Nota.- El paquete **tidyr** contiene registros de tuberculosis de la OMS; se recomienda consultar https://cran.r-project.org/web/packages/datos/readme/README.html para obtener más información.

Inicialmente, hemos de disponer de la instalación de los paquetes necesarios, para ello, ejecutamos las siguientes instrucciones:

```
#Instalamos y cargamos las librerías necesarias
if (!requireNamespace("tidyverse", quietly = TRUE)) {
  install.packages("tidyverse")
}
if (!requireNamespace("DBI", quietly = TRUE)) {
  install.packages("DBI")
}
if (!requireNamespace("RSQLite", quietly = TRUE)) {
  install.packages("RSQLite")
}
if (!requireNamespace("tidyr", quietly = TRUE)) {
  install.packages("tidyr")
}
```

A continuación, cargamos todos los paquetes:

```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse
2.0.0 ──
## ✓ forcats   1.0.0     ✓ readr     2.1.4
## ✓ ggplot2   3.4.3     ✓ stringr   1.5.0
## ✓ lubridate 1.9.2     ✓ tibble    3.2.1
## ✓ purrr     1.0.2     ✓ tidyr     1.3.0
## ── Conflicts ────────────────────────────────────────
tidyverse_conflicts() ──
```

```
## ✖ dplyr::filter() masks stats::filter()
## ✖ dbplyr::ident() masks dplyr::ident()
## ✖ dplyr::lag()    masks stats::lag()
## ✖ dplyr::select() masks MASS::select()
## ✖ dbplyr::sql()   masks dplyr::sql()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force
all conflicts to become errors

library(DBI)
library(RSQLite)
library(tidyr)
```

Definimos la conexión y creamos una base de datos SQLite en memoria:

```
#Se conecta con el comando de dbconnect y se especifica el driver según la
base de datos que se utiliza, en este caso SQLite.
con <- dbConnect(RSQLite::SQLite(), ":memory:")
con

## <SQLiteConnection>
##   Path: :memory:
##   Extensions: TRUE
```

A continuación, definiremos las tablas en las que guardaremos la información de cada una de las tablas de **tidyr**:

```
dbWriteTable(con, "table1", tidyr::table1) #1a variante
dbWriteTable(con, "table2", tidyr::table2) #2a variante
dbWriteTable(con, "table3", tidyr::table3) #3a variante
dbWriteTable(con, "table4a", tidyr::table4a) #4a variante
dbWriteTable(con, "table4b", tidyr::table4b) #4b variante
dbWriteTable(con, "table5", tidyr::table5) #5a variante
dbWriteTable(con, "who", tidyr::who) #Datos tuberculosis OMS
```

Nota.- Al escribir 'tidyr::' aparece el listado de elementos que contiene, entre ellos, las tablas que aparecen anteriormente.

Seguidamente, realizaremos algunas consultas sobre las tablas anteriores:

1) Mostrar las primeras filas de la tabla correspondiente a la 1a variante de la tuberculosis.

```
head(table1)

## # A tibble: 6 × 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
```

```
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583

Variante1 <- dbGetQuery(con, "SELECT * FROM table1")
head(Variante1)

##       country year  cases population
## 1 Afghanistan 1999    745   19987071
## 2 Afghanistan 2000   2666   20595360
## 3      Brazil 1999  37737  172006362
## 4      Brazil 2000  80488  174504898
## 5       China 1999 212258 1272915272
## 6       China 2000 213766 1280428583
```

2) Mostrar el total de casos por país del año 1999.

```
query1 <- "
  SELECT country, SUM(cases) AS total_cases
  FROM table1
  WHERE year = 1999
  GROUP BY country"
result1 <- dbGetQuery(con, query1)
print(result1)

##       country total_cases
## 1 Afghanistan         745
## 2      Brazil       37737
## 3       China      212258
```

3) Calcular la tasa de incidencia de casos por país para el año 2000.

```
query2 <- "
  SELECT country, (SUM(cases) / SUM(population)) * 1000 AS incidence_rate
  FROM table1
  WHERE year = 2000
  GROUP BY country"
result2 <- dbGetQuery(con, query2)
print(result2)

##       country incidence_rate
## 1 Afghanistan      0.1294466
## 2      Brazil      0.4612363
## 3       China      0.1669488

head(table2)

## # A tibble: 6 × 4
##   country      year type           count
##   <chr>       <dbl> <chr>          <dbl>
## 1 Afghanistan  1999 cases            745
## 2 Afghanistan  1999 population  19987071
## 3 Afghanistan  2000 cases           2666
```

```
## 4 Afghanistan  2000 population  20595360
## 5 Brazil       1999 cases          37737
## 6 Brazil       1999 population 172006362

head(table2)

## # A tibble: 6 × 4
##   country      year type           count
##   <chr>       <dbl> <chr>          <dbl>
## 1 Afghanistan  1999 cases            745
## 2 Afghanistan  1999 population  19987071
## 3 Afghanistan  2000 cases           2666
## 4 Afghanistan  2000 population  20595360
## 5 Brazil       1999 cases          37737
## 6 Brazil       1999 population 172006362
```

4) Mostrar los países con la mayor cantidad de casos por año.

```
query2 <- "
  SELECT country, year,
         SUM(CASE WHEN type = 'cases' THEN count ELSE 0 END) AS total_cases
  FROM table2
  WHERE type = 'cases'
  GROUP BY country, year
  ORDER BY total_cases DESC"
result2 <- dbGetQuery(con, query2)
print(result2)

##       country year total_cases
## 1       China 2000      213766
## 2       China 1999      212258
## 3      Brazil 2000       80488
## 4      Brazil 1999       37737
## 5 Afghanistan 2000        2666
## 6 Afghanistan 1999         745
```

Seleccionamos las columnas «country» y «year» de la tabla 1 (t1.country y t1.year).

Luego, calculamos la suma de los casos (total_cases) y la suma de la población (total_population) de la tabla 2 (t2), pero solo para los registros donde el tipo es 'cases' o 'population'. Utilizamos una cláusula CASE para realizar esta suma condicional.

Calculamos la tasa de incidencia (incidence_rate) dividiendo la suma de casos (total_cases) por la suma de población (total_population) y multiplicando por 1.000 para obtener la tasa por 1.000 personas.

Agrupamos los resultados por país (country) y año (year) utilizando la cláusula GROUP BY.

En resumen, esta consulta relaciona las dos tablas (tabla 1 y tabla 2) utilizando las columnas «country» y «year». Luego, calcula la tasa de incidencia de casos por país y año utilizando los datos de casos y población de la tabla 2. La consulta utiliza sumas condicionales (CASE) para

obtener los totales correctos de casos y población para cada país y año. La tasa de incidencia se calcula como la suma de casos dividida por la suma de población, multiplicada por 1.000 para obtener una tasa por 1.000 personas.

```
query <- "
  SELECT t1.country, t1.year,
         SUM(CASE WHEN t2.type = 'cases' THEN t2.count ELSE 0 END) AS
total_cases,
         SUM(CASE WHEN t2.type = 'population' THEN t2.count ELSE 0 END) AS
total_population,
         (SUM(CASE WHEN t2.type = 'cases' THEN t2.count ELSE 0 END) /
SUM(CASE WHEN t2.type = 'population' THEN t2.count ELSE 0 END)) * 1000 AS
incidence_rate
  FROM table1 AS t1
  INNER JOIN table2 AS t2
  ON t1.country = t2.country AND t1.year = t2.year
  WHERE (t2.type = 'cases' OR t2.type = 'population')
  GROUP BY t1.country, t1.year"
result <- dbGetQuery(con, query)
print(result)
```

```
##       country year total_cases total_population incidence_rate
## 1 Afghanistan 1999         745         19987071      0.0372741
## 2 Afghanistan 2000        2666         20595360      0.1294466
## 3      Brazil 1999       37737        172006362      0.2193930
## 4      Brazil 2000       80488        174504898      0.4612363
## 5       China 1999      212258       1272915272      0.1667495
## 6       China 2000      213766       1280428583      0.1669488
```

```
head(table1)
```

```
## # A tibble: 6 × 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Para finalizar y dado que hemos cargado la base de datos en memoria, realizamos la desconexión:

```
#Cerramos la conexión a la base de datos
dbDisconnect(db)
```

9. Ejercicios y casos prácticos en R

Ejercicio 1:

A partir de los datos siguientes de los pacientes de un hospital:

Nombre	Edad	Género	Presión arterial	Colesterol
Paciente1	45	Femenino	120/80	180
Paciente2	32	Masculino	130/85	200
Paciente3	28	Femenino	110/70	150
Paciente4	55	Masculino	140/90	220
Paciente5	40	Femenino	125/75	190

Cread un data frame con los datos de los pacientes anteriores y: 1. Filtrad los pacientes con niveles de colesterol por encima de 200. 2. Calculad las estadísticas descriptivas de los niveles de colesterol de los pacientes.

Ejercicio 2:

El porcentaje de grasa corporal óptima de un individuo depende de la edad y del género. Por ello, en general: Porcentaje masa corporal.

Edad (años) Mujer (%) Hombre (%) 10-30 20-26 12-18 31-40 21-27 13-19 41-50 22-28 14-20 51-60 22-30 16-20 61 o más 22-31 17-21 Nota.- Estos valores pueden variar ligeramente según el origen de la publicación. (Fuente: http://nutricionymultinivel.blogspot.com/p/tablas-de-valores-de-la-organizacion.html). Definid variables que representen la edad y el género de un individuo y, en función del valor que les asignéis, mostrad el porcentaje de masa corporal óptima que les correspondería con base en la tabla anterior.

Ejercicio 3:

A partir del conjunto de datos *anorexia* del paquete *MASS*, calculad el porcentaje de pacientes que han realizado cada tipo de tratamiento.

Ejercicio 4:

A partir del conjunto de datos *birthwt* del paquete *MASS*, calculad cuántos recién nacidos con un peso inferior a 2.600 gramos corresponden a madres menores de veinte años y fumadoras.

Ejercicio 5:

Supongamos que queremos generar un registro médico de pacientes que incluya información como nombre, edad, diagnóstico y niveles de glucosa en sangre, con los siguientes datos de pacientes:

nombres <- c(«Ana», «Luis», «María», «Carlos», «Elena») edades <- c(45, 32, 28, 55, 40) diagnósticos <- c(«Diabetes», «Hipertensión», «Sano», «Hipertensión», «Asma») glucosa <- c(125, 140, 95, 160, 110).

Definid una función llamada *crear_registro_médico* que tome los cuatro vectores anteriores como argumentos (nombres, edades, diagnósticos y glucosa) y cree un data frame que representa un registro médico de pacientes. Finalmente, el resultado por pantalla deberá ser un data frame que contiene información detallada sobre cada paciente, incluyendo su nombre, edad, diagnóstico y niveles de glucosa en sangre.

Ejercicio 6:

A partir del conjunto de datos *birthwt* del paquete *MASS*, guardad en un vector los pesos de los recién nacidos cuyas madres tengan menos de 20 años. Calculad la media y representad gráficamente este vector (por ejemplo, un diagrama de puntos).

Ejercicio 7:

A partir de los ejemplos trabajados de definición de funciones, definid dos funciones, una que calcule el IMC de un individuo y otra que calcule el porcentaje de peso corporal óptimo en función de la edad y el género. En ambas funciones, es necesario definir los parámetros que correspondan.

Ejercicio 8:

Si consideramos el conjunto de datos *birthwt* del paquete *MASS*, cread una función que muestre los valores de peso de los recién nacidos cuyas madres sean de raza blanca y cuyo peso esté por debajo de la media de los pesos observados.

Ejercicio 9:

A partir del conjunto de datos Melanoma del paquete *MASS* que hace referencia a los datos de supervivencia de melanoma maligno, se pide resolver las siguientes cuestiones: a) Definid un vector que guarde las edades de los individuos observados y calculad la media, el máximo y el mínimo de esta distribución. b) Definid una función que recorra el conjunto de datos y muestre cuántos de los individuos observados murieron a causa del melanoma, cuántos sobrevivieron y cuántos murieron por diferentes causas. c) Definid una función que recorra el conjunto de datos y muestre la proporción de hombres y de mujeres que murieron por melanoma.

Ejercicio 10:

A partir del conjunto de datos anterior *airquality* trabajado anteriormente, resolved las siguientes consultas:

1) Mostrad la media del factor Ozone.
2) Mostrad los datos de Ozone, Solar.R y Wind de los meses (Month) 5, 6 y 7.
3) Mostrad la información anterior ordenada por Month.

4) Mostrad los datos de Wind agrupados por Month. Nota: Utilizad la sintaxis adecuada para mostrar por pantalla solamente los cinco primeros registros.

Caso práctico 1. La serie de Fibonacci en las moléculas de ADN

Según las investigaciones de Mark E. Curtis, cada giro completo de la doble hélice del ADN mide 21 Å de ancho por 34 Å de largo; 34 y 21 son dos números consecutivos en la sucesión de Fibonacci. Por lo tanto, su cociente, 1.6190476, aproxima de cerca la Razón Áurea φ=1.6180339… (Fuente: http://www.sacred-geometry.es/?q=es/content/phi-en-la-estructura-del-adn).

Se pide:

a) Implementad en **R** una función que, dado un valor n, muestre los valores de la serie de Fibonacci hasta ese valor determinado.
b) ¿Para qué valores de n localizamos, en la serie de Fibonacci, las medidas de la doble hélice de la molécula de ADN?
c) Complementad el código del apartado a) añadiendo una nueva función (o las instrucciones que correspondan en la función creada anteriormente) de manera que, para un determinado valor n, muestre las proporciones sucesivas de los valores de la serie de Fibonacci. ¿A partir de qué valor de n, esta proporción se va aproximando a la Razón Áurea?

Caso práctico 2. Secuencias de ADN

La secuencia de ADN de un individuo contiene información genética de este. Actualmente, existen técnicas que permiten realizar esta secuenciación y aportar datos para investigar el comportamiento de diferentes tipos de organismos biológicos. (Fuente: https://es.wikipedia.org/wiki/Secuenciaci%C3%B3n_del_ADN). Se pide generar una secuencia de ADN y calcular su inverso y su complementario.

Caso práctico 3. Gestión de base de datos de pacientes

Supongamos que pretendemos gestionar la información de una base de datos de pacientes de un hospital. En principio, disponemos de dos tablas, *pacientes* y *registros_médicos*.

De la tabla *pacientes*, disponemos de los siguientes datos: pacientes (nombre, edad, género, condición_médica) (‘Paciente1’, 45, ‘Femenino’, ‘Diabetes’), (‘Paciente2’, 32, ‘Masculino’, ‘Hipertensión’), (‘Paciente3’, 28, ‘Femenino’, ‘Sano’), (‘Paciente4’, 55, ‘Masculino’, ‘Hipertensión’), (‘Paciente5’, 40, ‘Femenino’, ‘Asma’).

De la tabla *registros_médicos*, disponemos de los siguientes datos: registros_médicos (paciente_id, fecha, nivel_glucosa) (1, ‘2023-08-01’, 125.5), (2, ‘2023-08-01’, 140.2), (3, ‘2023-08-01’, 95.8), (4, ‘2023-08-01’, 160.6), (5, ‘2023-08-01’, 110.9).

Se pide utilizar las instrucciones de acceso a bases de datos para resolver las siguientes cuestiones:

1) Cread la base de datos (*Hospital*), las tablas y los registros correspondientes y la conexión desde **R**.
2) Mostrad pacientes con una condición médica específica (por ejemplo, diabetes).
3) Calculad estadísticas descriptivas de los niveles de glucosa en sangre.
4) Obtened los registros médicos de pacientes con hipertensión y sus estadísticas de glucosa.
5) Calculad la edad promedio de pacientes con asma por género.
6) Cread un gráfico de barras para mostrar la distribución de género en pacientes con diabetes.

Es importante recordar cargar los paquetes *RSQLite*, *dplyr* y, también, *ggplot2*.

Solución de los ejercicios y casos prácticos en R

Solución del ejercicio 1:

```
#Cargar la biblioteca dplyr si aún no está instalada
# install.packages("dplyr")
library(dplyr)

#Crear un data frame con datos de pacientes
datos_pacientes <- data.frame(
  Nombre = c("Paciente1", "Paciente2", "Paciente3", "Paciente4",
"Paciente5"),
  Edad = c(45, 32, 28, 55, 40),
  Género = c("Femenino", "Masculino", "Femenino", "Masculino", "Femenino"),
  Presión arterial = c("120/80", "130/85", "110/70", "140/90", "125/75"),
  Colesterol = c(180, 200, 150, 220, 190)
)

#Filtrar pacientes con niveles de colesterol por encima de 200
pacientes_ colesterol alto <- datos_pacientes %>%
  filter(Colesterol > 200)

#Calcular estadísticas descriptivas de los niveles de colesterol de los
pacientes
estadísticas_colesterol <- pacientes_alto_colesterol %>%
  summarize(
    media_colesterol = mean(Colesterol),
    desviación_estándar_colesterol = sd(Colesterol),
    máximo_colesterol = max(Colesterol),
    mínimo_colesterol = min(Colesterol)
  )

#Mostrar el data frame de pacientes con colesterol alto y estadísticas
descriptivas
print(pacientes_alto_colesterol)

##      Nombre Edad    Género Presión arterial Colesterol
## 1 Paciente4   55 Masculino          140/90        220

print("Estadísticas de niveles de colesterol en pacientes con colesterol
alto:")

## [1] "Estadísticas de niveles de colesterol en pacientes con colesterol
alto:"

print(estadísticas_colesterol)

##   media_colesterol desviación_estándar_colesterol máximo_colesterol
## 1              220                             NA               220
```

```
##   mínimo_colesterol
## 1               220
```

Solución del ejercicio 2:

```
##definimos las variables sexo y edad, asignando unos determinados valores
sexo<-"mujer"
edad<-25
##Evaluamos los valores de la masa corporal óptima según género y edad
if(sexo=='mujer') {
if(edad<10)
 mc_opt<-"No registrado"
else
if(edad<=30)
 mc_opt<-'20-26'
else
if(edad<=40)
 mc_opt<-'21-27'
else
if(edad<=50)
 mc_opt<-'22-28'
else
if(edad<=60)
 mc_opt<-'22-30'
else
 mc_opt<-'22-31'
}else {
if(edad<10)
 mc_opt<-"No registrado"
else
if(edad<=30)
 mc_opt<-'12-18'
else
if(edad<=40)
 mc_opt<-'13-19'
else
if(edad<=50)
 mc_opt<-'14-20'
else
if(edad<=60)
 mc_opt<-'16-20'
else
 mc_opt<-'17-21'
}
#a partir de los valores iniciales de edad y género, mostramos el valor de la
masa corporal óptima
print(mc_opt)

## [1] "20-26"
```

Solución del ejercicio 3:

Una posible solución sería la siguiente:

```
#cargamos los datos de anorexia
library(MASS)
data("anorexia")
#definimos los contadores de cada tipo de tratamiento
p_cont<-0
p_cbt<-0
p_ft<-0
tratamientos<-c(anorexia$Treat)
#definimos la variable que utilizaremos para recorrer los registros
for (i in 1:length(tratamientos)) {
if (anorexia$Treat[i]=="Cont")
 p_cont<-p_cont+1
else
if (anorexia$Treat[i]=="CBT")
 p_cbt<-p_cbt+1
else
 p_ft<-p_ft+1
}
#calculamos los porcentajes de cada tratamiento
porc_cont<-p_cont/length(tratamientos)
porc_cbt<-p_cbt/length(tratamientos)
porc_ft<-p_ft/length(tratamientos)
#mostramos los valores de los porcentajes
print(porc_cont)

## [1] 0.3611111

print(porc_cbt)

## [1] 0.4027778

print(porc_ft)

## [1] 0.2361111
```

Solución del ejercicio 4:

Una posible solución es la siguiente:

```
#cargamos los datos de birthwt
library(MASS)
data("birthwt")
#definimos las variables
c_inf<-0
bwt<-c(birthwt$bwt)
#recorremos los registros del conjunto de datos
for(i in 1:length(bwt))
#comprobamos los criterios establecidos en el enunciado
```

```
if (birthwt$bwt[i]<2600)
if (birthwt$age[i]<20)
if (birthwt$smoke[i]==1)
 c_inf<-c_inf+1
#mostramos el resultado
print (c_inf)

## [1] 7
```

Solución del ejercicio 5:

```
nombres <- c("Ana", "Luis", "María", "Carlos", "Elena")
edades <- c(45, 32, 28, 55, 40)
diagnósticos <- c("Diabetes", "Hipertensión", "Sano", "Hipertensión", "Asma")
glucosa <- c(125, 140, 95, 160, 110)

#Función para crear un registro médico de pacientes
crear_registro_médico <- function(nombres, edades, diagnósticos, glucosa) {
  pacientes <- data.frame(
    Nombre = nombres,
    Edad = edades,
    Diagnóstico = diagnósticos,
    Glucosa = glucosa
  )
  return(pacientes)
}

#Datos de pacientes
nombres <- c("Ana", "Luis", "María", "Carlos", "Elena")
edades <- c(45, 32, 28, 55, 40)
diagnósticos <- c("Diabetes", "Hipertensión", "Sano", "Hipertensión", "Asma")
glucosa <- c(125, 140, 95, 160, 110)

#Llamada a la función para crear el registro médico
registro_médico <- crear_registro_médico(nombres, edades, diagnósticos,
glucosa)

#Mostrar el registro médico
print(registro_medico)

##   Nombre Edad  Diagnóstico Glucosa
## 1    Ana   45     Diabetes     125
## 2   Luis   32 Hipertensión     140
## 3  María   28         Sano      95
## 4 Carlos   55 Hipertensión     160
## 5  Elena   40         Asma     110
```

Solución del ejercicio 6:

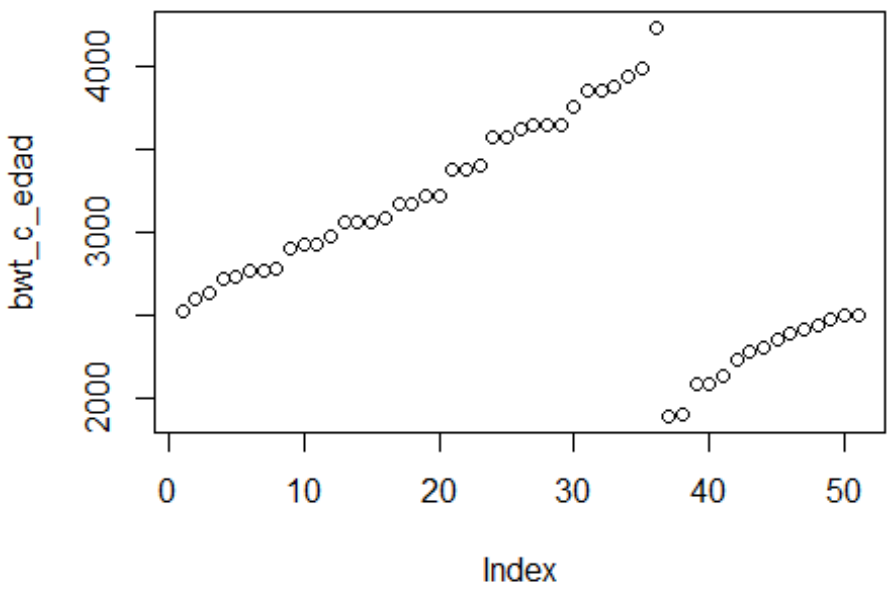
```
#cargamos los datos de birthwt
library(MASS)
data("birthwt")
#definimos el vector que guardará los pesos
bwt_c<-c()
j<-1
#definimos la función
F_bwt<-function(){
#recorremos los valores de pesos y guardamos en un vector aquellos cuya edad
de las madres sea <20
for (i in 1:length(birthwt$bwt))
if (birthwt$age[i]<20){
 bwt_c[j]<-birthwt$bwt[i]
 j<-j+1
 }
#calculamos la media
return (bwt_c)
}
bwt_c_edad<-F_bwt()
bwt_c_edad
```

```
##  [1] 2523 2600 2637 2722 2733 2769 2769 2778 2906 2920 2920 2977 3062 3062
3062
## [16] 3090 3175 3175 3225 3225 3374 3374 3402 3572 3572 3629 3643 3651 3651
3756
## [31] 3856 3860 3884 3941 3997 4238 1885 1899 2082 2084 2125 2225 2282 2296
2353
## [46] 2381 2414 2438 2466 2495 2495
```

```
#calculamos la media y un gráfico de puntos
mean(bwt_c_edad)
```

```
## [1] 2973.529
```

```
plot(bwt_c_edad)
```



Solución del ejercicio 7:

```
#Definimos la función que calcule el índice de masa corporal (IMC) a partir
del peso y la altura
f_IMC<-function(peso,altura) {
return (peso/(altura*altura))
}
v_IMC<-f_IMC(45,1.65)
print(v_IMC)

## [1] 16.52893

## [1] 16.52893
#Definimos la función que calcule el peso corporal óptimo (PCO) a partir del
sexo y la edad
f_PCO<-function(sexo,edad){
if(sexo=='mujer') {
if(edad<10)
 mc_opt<-"No registrado"
else
if(edad<=30)
 mc_opt<-'20-26'
else
if(edad<=40)
 mc_opt<-'21-27'
else
if(edad<=50)
 mc_opt<-'22-28'
else
if(edad<=60)
 mc_opt<-'22-30'
else
 mc_opt<-'22-31'
```

```
}else {
if(edad<10)
 mc_opt<-"No registrado"
else
if(edad<=30)
 mc_opt<-'12-18'
else
if(edad<=40)
 mc_opt<-'13-19'
else
if(edad<=50)
 mc_opt<-'14-20'
else
if(edad<=60)
 mc_opt<-'16-20'
else
 mc_opt<-'17-21'
}
return (mc_opt)
 }
v_PCO<-f_PCO('mujer',25)
print(v_PCO)
```

```
## [1] "20-26"
```

Solución del ejercicio 8:

Una posible solución sería la siguiente:

```
media_peso<-function(){
bwt<-c(birthwt$bwt)
race<-c(birthwt$race)
mean(bwt)
for (i in 1:length(bwt))
  if (race[i]==1)
    if (bwt[i]<mean(bwt))
      print(bwt[i])
}
media_peso()
```

```
## [1] 2557
## [1] 2594
## [1] 2600
## [1] 2637
## [1] 2663
## [1] 2665
## [1] 2769
## [1] 2769
## [1] 2782
## [1] 2821
## [1] 2835
```

```
## [1] 2836
## [1] 2877
## [1] 2906
## [1] 2920
## [1] 2922
## [1] 1021
## [1] 1790
## [1] 1818
## [1] 1885
## [1] 1928
## [1] 1928
## [1] 1936
## [1] 2082
## [1] 2084
## [1] 2084
## [1] 2100
## [1] 2187
## [1] 2225
## [1] 2296
## [1] 2353
## [1] 2353
## [1] 2410
## [1] 2410
## [1] 2414
## [1] 2424
## [1] 2466
## [1] 2466
## [1] 2495
```

Solución del ejercicio 9:

```
#cargamos los datos de Melanoma
library(MASS)
data("Melanoma")
#mostramos los datos de Melanoma
head(Melanoma)
```

```
##   time status sex age year thickness ulcer
## 1   10      3   1  76 1972      6.76     1
## 2   30      3   1  56 1968      0.65     0
## 3   35      2   1  41 1977      1.34     0
## 4   99      3   0  71 1968      2.90     0
## 5  185      1   1  52 1965     12.08     1
## 6  204      1   1  28 1971      4.84     1
```

```
## time status sex age year thickness ulcer
## 1 10 3 1 76 1972 6.76 1
## 2 30 3 1 56 1968 0.65 0
## 3 35 2 1 41 1977 1.34 0
## 4 99 3 0 71 1968 2.90 0
## 5 185 1 1 52 1965 12.08 1
```

```
## 6 204 1 1 28 1971 4.84 1
#a) Vector que guarde las edades de los individuos observados y calculad la
media, el máximo y el mínimo de esta distribución.
edades_m<-c(Melanoma$age)
mean(edades_m) #media

## [1] 52.46341

## [1] 52.46341
max(edades_m) #max

## [1] 95

## [1] 95
min(edades_m) #min

## [1] 4

## [1] 4
#b) Función que recorra el conjunto de datos y muestre cuántos de los
individuos observados murieron a causa del melanoma, cuántos
#sobrevivieron y cuántos murieron por diferentes causas.

F_diagnóstico<-function() {
#definimos contadores
muerte_m<-0
vivo<-0
muerte_oc<-0
status<-Melanoma$status
for (i in 1:length(Melanoma$status))
if(status[i]==1)
 muerte_m<-muerte_m+1
else
if(status[i]==2)
 vivo<-vivo+1
else
 muerte_oc<-muerte_oc+1
return (c(muerte_m,vivo,muerte_oc))
}
print(F_diagnóstico())

## [1]  57 134  14

## [1] 57 134 14
#c) Función que recorra el conjunto de datos y muestre la proporción de
hombres y de mujeres que murieron por melanoma.
F_diagnóstico_porc<-function(){
 muerte_m<-0
 muerte_m_m<-0
 muerte_m_f<-0
 status<-Melanoma$status
 sex<-Melanoma$sex
```

```
for (i in 1:length(Melanoma$status))
if(status[i]==1){
 muerte_m<-muerte_m+1
if (sex[i]==1)
 muerte_m_m<-muerte_m_m+1
else
 muerte_m_f<-muerte_m_f+1
 }
 porc_m_m<-muerte_m_m/muerte_m
 porc_m_f<-muerte_m_f/muerte_m
return (c(porc_m_m,porc_m_f))
}
print(F_diagnóstico_porc())
```

```
## [1] 0.5087719 0.4912281
```

```
## [1] 0.5087719 0.4912281
```

Solución del ejercicio 10:

1) Cargamos los paquetes necesarios para trabajar con las instrucciones correspondientes al acceso a bases de datos.

```
library(sqldf)
library(RSQLite)
library(dplyr)
```

```
db<-"RSQLite"::datasetsDb()
dbListTables(db) ##Mostramos los conjuntos de datos
```

```
##  [1] "BOD"              "CO2"              "ChickWeight"      "DNase"
##  [5] "Formaldehyde"     "Indometh"         "InsectSprays"
"LifeCycleSavings"
##  [9] "Loblolly"         "Orange"           "OrchardSprays"
"PlantGrowth"
## [13] "Puromycin"        "Theoph"           "ToothGrowth"      "USArrests"
## [17] "USJudgeRatings"   "airquality"       "anscombe"         "attenu"
## [21] "attitude"         "cars"             "chickwts"         "esoph"
## [25] "faithful"         "freeny"           "infert"           "iris"
## [29] "longley"          "morley"           "mtcars"           "npk"
## [33] "pressure"         "quakes"           "randu"            "rock"
## [37] "sleep"            "stackloss"        "swiss"            "trees"
## [41] "warpbreaks"       "women"
```

```
#Media del factor *Ozone*.
sqldf("SELECT AVG(Ozone) FROM airquality")
```

```
##   AVG(Ozone)
## 1   42.12931
```

2)

```
#Datos de *Ozone*, *Solar.R* y *Wind* de los meses(*Month*) 5, 6 y 7.
head(sqldf("SELECT Ozone,[Solar.R],Wind FROM airquality where Month IN
(5,6,7)"))
```

```
##   Ozone Solar.R Wind
## 1    41     190  7.4
## 2    36     118  8.0
## 3    12     149 12.6
## 4    18     313 11.5
## 5    NA      NA 14.3
## 6    28      NA 14.9
```

3)

```
#Ordenar por *Month*
head(sqldf("SELECT Ozone,[Solar.R],Wind, Month FROM airquality where
Month IN (5,6,7) ORDER BY Month DESC"))
```

```
##   Ozone Solar.R Wind Month
## 1   135     269  4.1     7
## 2    49     248  9.2     7
## 3    32     236  9.2     7
## 4    NA     101 10.9     7
## 5    64     175  4.6     7
## 6    40     314 10.9     7
```

4)

```
#d) Mostrad los datos de *Wind*, agrupados por *Month*.
sqldf("SELECT Wind,Month FROM airquality GROUP BY Month")
```

```
##   Wind Month
## 1  7.4     5
## 2  8.6     6
## 3  4.1     7
## 4  6.9     8
## 5  6.9     9
```

Solución caso práctico 1:

a)

```
##definimos la función
f_fibonacci <- function(n){
    num <- c(0, 1)
     i = 2
     for (i in i:(n-1)){
         num[i+1] <- num[i]+num[i-1]
     }
num <-num[1:length(num)]
     return(num)
```

```
}
f_fibonacci(5)

## [1] 0 1 1 2 3

f_fibonacci(8)

## [1]  0  1  1  2  3  5  8 13

f_fibonacci(15)

##  [1]   0   1   1   2   3   5   8  13  21  34  55  89 144 233 377
```

b)

```
f_fibonacci(9)

## [1]  0  1  1  2  3  5  8 13 21

f_fibonacci(10)

##  [1]  0  1  1  2  3  5  8 13 21 34
```

c)

```
f_Áurea<-function(n){
  valores<-f_fibonacci(n)
  razón<-c(1)
  for(i in 2:n){
    razón[i]<-valores[i]/valores[i-1]
  }
  return(razón)
}
```

Comprobamos para distintos valores de n:

```
f_Áurea(7)

## [1] 1.000000      Inf 1.000000 2.000000 1.500000 1.666667 1.600000

f_Áurea(15)

##  [1] 1.000000      Inf 1.000000 2.000000 1.500000 1.666667 1.600000
1.625000
##  [9] 1.615385 1.619048 1.617647 1.618182 1.617978 1.618056 1.618026
```

Comprobamos que, a medida que va aumentando el valor de n, más se ajusta el valor obtenido al valor de la razón áurea.

Solución caso práctico 2:

```
#Función para generar una secuencia de ADN aleatoria
generar_secuencia_adn <- function(longitud) {
  bases <- c("A", "T", "C", "G")
  secuencia <- sample(bases, longitud, replace = TRUE)
  return(paste(secuencia, collapse = ""))
}

#Función para obtener el inverso de una secuencia de ADN
obtener_inverso_adn <- function(secuencia) {
  inverso <- rev(strsplit(secuencia, "")[[1]])
  return(paste(inverso, collapse = ""))
}

#Función para obtener el complementario de una secuencia de ADN
obtener_complementario_adn <- function(secuencia) {
  complementario <- chartr("ATCG", "TAGC", secuencia)
  return(complementario)
}

#Generar una secuencia de ADN aleatoria
secuencia_original <- generar_secuencia_adn(20)

#Obtener el inverso de la secuencia
secuencia_inversa <- obtener_inverso_adn(secuencia_original)

#Obtener el complementario de la secuencia
secuencia_complementaria <- obtener_complementario_adn(secuencia_original)

#Mostrar los resultados
cat("Secuencia original: ", secuencia_original, "\n")

## Secuencia original:  GGTTTTGCTTCTAAAATAGG

cat("Secuencia inversa: ", secuencia_inversa, "\n")

## Secuencia inversa:  GGATAAAATCTTCGTTTTGG

cat("Secuencia complementaria: ", secuencia_complementaria, "\n")

## Secuencia complementaria:  CCAAAACGAAGATTTTATCC
```

Puede visualizarse un ejemplo más exhaustivo en http://manuals.bioinformatics.ucr.edu/home/programming-in-r#TOC-R-Programming-Exercises

Solución caso práctico 3:

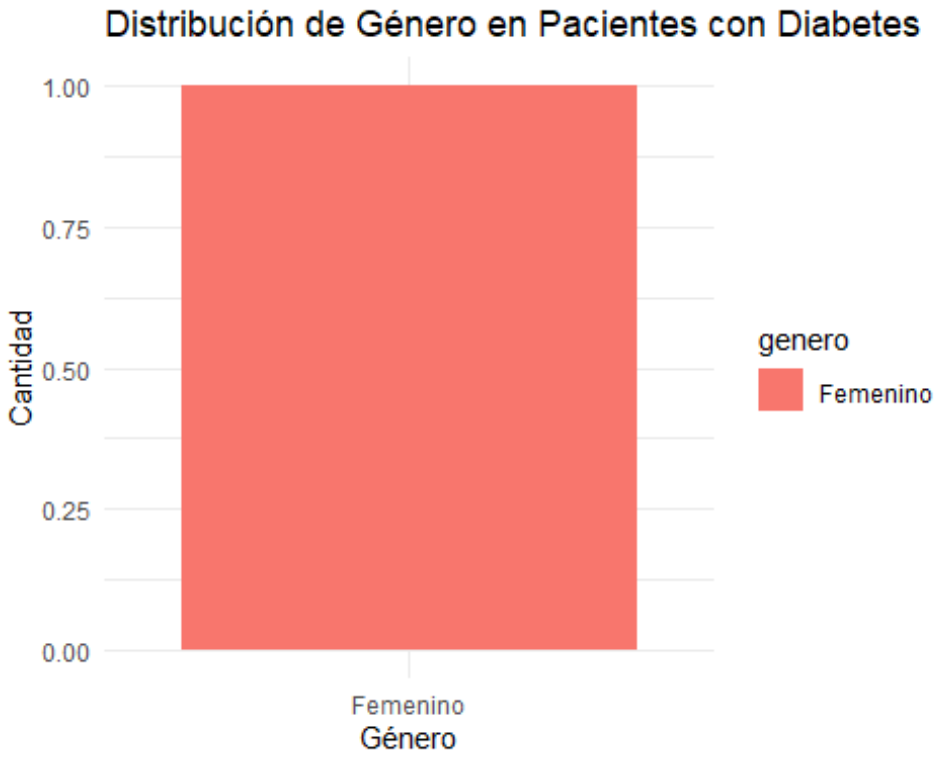
Una posible solución a los requerimientos definidos es la siguiente:

```
#Instalar y cargar los paquetes necesarios
#install.packages("RSQLite")
#install.packages("dplyr")
#install.packages("ggplot2")
library(RSQLite)
library(dplyr)
library(ggplot2)

#Paso 1: Crear la base de datos, las tablas y los registros correspondientes
y la conexión desde R

#Conectar a la base de datos SQLite
con <- dbConnect(SQLite(), "hospital.db")

#Crear la tabla "pacientes" y agregar registros
pacientes <- data.frame(
  nombre = c('Paciente1', 'Paciente2', 'Paciente3', 'Paciente4',
'Paciente5'),
  edad = c(45, 32, 28, 55, 40),
  género = c('Femenino', 'Masculino', 'Femenino', 'Masculino', 'Femenino'),
  condición_médica = c('Diabetes', 'Hipertensión', 'Sano', 'Hipertensión',
'Asma')
)

dbWriteTable(con, "pacientes", pacientes, row.names = FALSE, overwrite =
TRUE)

#Crear la tabla "registros_médicos" y agregar registros
registros_médicos <- data.frame(
  paciente_id = c(1, 2, 3, 4, 5),
  fecha = as.Date('2023-08-01'),
  nivel_glucosa = c(125.5, 140.2, 95.8, 160.6, 110.9)
)

dbWriteTable(con, "registros_médicos", registros_médicos, row.names = FALSE,
overwrite = TRUE)

#Paso 2: Mostrar pacientes con una condición médica específica (por ejemplo,
diabetes)
pacientes_diabetes <- dbGetQuery(con, "SELECT * FROM pacientes WHERE
condición_médica = 'Diabetes'")
print(pacientes_diabetes)
```

```
##      nombre edad   genero condición_médica
## 1 Paciente1   45 Femenino         Diabetes
```

```
#Paso 3: Calcular estadísticas descriptivas de los niveles de glucosa en
sangre
estadísticas_glucosa <- dbGetQuery(con, "SELECT AVG(nivel_glucosa) AS
media_glucosa,
                                         MAX(nivel_glucosa) AS
máximo_glucosa,
                                         MIN(nivel_glucosa) AS
mínimo_glucosa
                                         FROM registros_médicos")
print(estadísticas_glucosa)

##   media_glucosa máximo_glucosa mínimo_glucosa
## 1         126.6          160.6           95.8

#Paso 4: Obtener los registros médicos de pacientes con hipertensión y sus
estadísticas de glucosa
registros_hipertensión <- dbGetQuery(con, "SELECT r.*
                                          FROM registros_médicos r
                                          JOIN pacientes p ON r.paciente_id
= p.rowid
                                          WHERE p.condición_médica =
'Hipertensión'")

estadísticas_glucosa_hipertensión <- dbGetQuery(con, "SELECT
AVG(nivel_glucosa) AS media_glucosa,
                                                      MAX(nivel_glucosa)
AS máximo_glucosa,
                                                      MIN(nivel_glucosa)
AS mínimo_glucosa
                                                      FROM
registros_médicos
                                                      WHERE paciente_id
IN (SELECT rowid

FROM pacientes

WHERE condición_médica = 'Hipertensión')")
print(registros_hipertension)

##   paciente_id fecha nivel_glucosa
## 1           2 19570         140.2
## 2           4 19570         160.6

print(estadísticas_glucosa_hipertensión)

##   media_glucosa máximo_glucosa mínimo_glucosa
## 1         150.4          160.6          140.2

#Paso 5: Calcular la edad promedio de pacientes con asma por género
edad_promedio_asma <- dbGetQuery(con, "SELECT género, AVG(edad) AS
edad_promedio
```

```
                                    FROM pacientes
                                    WHERE condición_médica = 'Asma'
                                    GROUP BY género")
print(edad_promedio_asma)

##     género edad_promedio
## 1 Femenino            40

#Paso 6: Crear un gráfico de barras para mostrar la distribución de género en
pacientes con diabetes
género_diabetes <- dbGetQuery(con, "SELECT género, COUNT(*) AS cantidad
                                    FROM pacientes
                                    WHERE condición_médica = 'Diabetes'
                                    GROUP BY género")

#Crear un gráfico de barras
ggplot(género_diabetes, aes(x = género, y = cantidad, fill = género)) +
  geom_bar(stat = "identity") +
  labs(title = "Distribución de Género en Pacientes con Diabetes", x =
"Género", y = "Cantidad") +
  theme_minimal()
```



Información adicional

Algunos de los recursos materiales utilizados y recomendados para trabajar este **LAB3** son los siguientes:

- Web oficial de R: https://www.r-project.org/
- Tutorial de R: https://cran.r-project.org/doc/contrib/R-intro-1.1.0-espanol.1.pdf
- A first Course in Statistical Programming with R (Capítulos 1 y 2). Autores: John Braun, W. and Ducan J. Murdoch. Editorial: Cambridge University. 2010.
- Data manipulation with R (Capítulos 1 y 2). Autor: Jaynal Abedin. Editorial: Packt Publishing. +
- The R Book (Capítulos 1, 2, 3, 4 y 5). Autores: Michael J. Crawley (Capítulos 1, 2 y 3). Editorial: Wiley. 2009. + R Cookbook. Autor: Paul Teetor. Editorial: O'Reilly.
- Estadística básica con R y RCommander. Autores: A. J. Arriaza Gómez, F. Fernández Palacín, M. A. López Sánchez, M. Muñoz Márquez, S. Pérez Plaza, A. Sánchez Navas. Editorial: Servicio de publicaciones. Universidad de Cádiz. (http://knuth.uca.es/moodle/mod/url/view.php?id=1126)
- Métodos estadísticos con R y RCommander. Autor: Antonio Jesús Sáez Castillo. (http://www4.ujaen.es/~ajsaez/recursos.htm)
- R for data science (https://r4ds.had.co.nz/). Ver capítulo 19. + R para profesionales de los datos (https://datanalytics.com/libro_r/_main.pdf). Ver Capítulo 12.
- Bioinformatics with R Cookbook. Autor: Paurush Praveen Sinha. Editorial: Packt Publishing.
- http://manuals.bioinformatics.ucr.edu/home/programming-in-r#TOC-R-Programming-Exercises
- Databases using dplyr: https://db.rstudio.com/dplyr/
- Dplyr for Databases: https://rpubs.com/jladams/dbplyr + SQL databases en R: https://datacarpentry.org/R-ecology-lesson/05-r-and-databases.html