

Resumen de comandos de R

Contents

Constantes y variables	2
Ayuda	3
Funciones	3
Paquetes	4
Operadores	4
Aritméticos	4
Relacionales	5
Lógicos	6
Números complejos	6
Tipos de datos	6
NA y NULL	8
Condicionales	8
if	8
ifelse	8
for	8
while	9
Break y next	10
Repeat	10
lapply y sapply	10
sapply	12

Estructuras de datos	12
Vector	12
Propiedades	13
Operaciones	13
Extracción de valores	14
Sucesiones	15
Repetición	15
Función all y any	16
Matriz	16
Propiedades	17
Listas	19
Lista de listas	20
Concatenar listas	21
Propiedades	22
Dataframe	22
Mostrando los datos	22
Cargando conjunto de datos	23
Accediendo a los datos del dataframe	23
Filtros	24
Ordenando los datos de un dataframe	24
Modificando los nombres de las columnas	25
Transformación de datos	25
Canalización	25
Función anidada	25
Gráficos	25
Histograma	29
Ruido	30
Bibliografía	34

Este notebook ha sido creado en RStudio, un entorno que incluye una consola para insertar código, historial, gráficos, paquetes y librerías.

Constantes y variables

En R, se usa `<-` para asignar valores a una variable. Los nombres de las variables pueden incluir letras, números, puntos y guiones bajos, sin embargo, siempre deben empezar con una letra.

```
# Asignandole 1 a la variable a
a<-1
```

```
# Imprimir a
print(a) # o simplemente a
```

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

```
# Imprimir un numero con un caracter usando paste
print(paste("Cantidad de personas:", a))
```

```
## [1] "Cantidad de personas: 1"
```

Ayuda

Es posible obtener la documentación de una función digitando un signo de interrogación (?) al inicio de esta o escribiendo `help(" ")` con el nombre de la función dentro de las comillas. Al hacerlo, en la pestaña de Help se mostrará cómo utilizar dicha función, sus parámetros y ejemplos.

```
?sum()
# o help("sum")
```

La documentación de un paquete se puede obtener con:

```
help(package = "datasets")
```

Funciones

En R, una función posee la siguiente sintaxis: **nombre_funcion()**. Dentro de los paréntesis, van los argumentos de la función. Algunas funciones básicas definidas de R son:

- `sum()`: suma.
- `mean()`: promedio.
- `max()`: máximo.
- `min()`: mínimo.
- `sqrt()`: raíz cuadrada.
- `sort()`: ordena los valores de menor a mayor.
- `unique()`: muestra valores únicos.

Las funciones también pueden ser definidas por el usuario usando la siguiente sintaxis:

```
nombre_función <- function(argumentos) {
  # codigo
  return(resultado)
}
```

Ejemplo:

```
cuadrado <- function(x){  
  resultado <- x^2  
  return(resultado)  
}  
  
cuadrado(4)
```

```
## [1] 16
```

Paquetes

En R, un paquete es una colección con funciones que no están en R base. CRAN es el repositorio de paquetes oficial de R, los cuales se pueden instalar mediante `install.packages()`. Por ejemplo:

```
install.packages("stats")
```

Luego de instalar el paquete, las funciones de este se podrán utilizar después de ejecutar `library()` con el nombre del paquete dentro de la función.

```
library(stats)
```

Cada vez que se inicia una nueva sesión y se requiera usar una función que pertenezca a un paquete, se debe ejecutar `library()`.

Para saber qué paquetes están instalados, se debe ejecutar `installed.packages()`, sin argumento.

Operadores

Aritméticos

Los operadores aritméticos de R base que se pueden utilizar con datos enteros y numéricos son:

- `+`: suma.
- `-`: resta.
- `*`: multiplicación.
- `/`: división.
- `^`: exponencial.
- `%%`: residuo de la división.
- `%/%`: cociente.
- `%*%`: multiplicación entre matrices.

Ejemplo:

```
3*5+2
```

```
## [1] 17
```

El orden de las operaciones es el siguiente:

1. ^.
2. *.
3. /.
4. +.
5. -.
6. <, >, <=, >=, ==, !=.
7. !.
8. &.
9. |.
10. <-.

Relacionales

Los operadores relacionales se utilizan para comparar un valor con otro. Siempre devuelven TRUE o FALSE (1 o 0, respectivamente).

- : mayor que.
- =: mayor o igual que.
- <: menor que.
- <=: menor o igual que.
- ==: igual.
- !=: distinto.

Ejemplo 1:

```
234 > 243
```

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

Ejemplo 2:

```
"manzana" == "MANZANA"
```

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

Ejemplo 3:

```
edad <- c(12, 35, 46, 2)
edad > 20
```

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

Lógicos

Los operadores lógicos se utilizan para crear condiciones. Devuelven TRUE o FALSE.

- `&`: y.
- `|`: o.
- `!`: not, negación lógica.

Ejemplo 1:

```
# Al usar &, si uno de los valores es FALSE, devuelve FALSE
# Como 234 es menor que 243, devuelve FALSE
234 & 2000 > 243
```

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

Ejemplo 2:

```
# Al usar |, si uno de los valores es TRUE, devuelve TRUE
# Como 2000 es mayor que 243, devuelve TRUE
234 | 2000 > 243
```

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

Números complejos

En R, los números complejos se representan mediante la forma $(a + bi)$, donde a y b son números reales e i es la unidad imaginaria. Se usa `complex()` para crear un número complejo de la siguiente forma:

```
# Creando un numero complejo
z_1 <- complex(real=14, imaginary=7)
z_1
```

```
## [1] 14+7i
```

Otras funciones:

- `as.complex()`: declara un número real como complejo.
- `Conj()`: conjugado.
- `Mod()`: módulo.
- `Arg()`: argumento en radianes.

Tipos de datos

Los tipos de datos más comunes en R son:

1. integer: entero. Ejemplo: 1.
2. double: decimales. Ejemplo: 1.7

3. numeric: real. Ejemplo 4.5.
4. character: cadena de texto. Ejemplo: "Hola mundo"
5. factor: se utiliza para representar variables categóricas. Ejemplo: Categoría de productos como A, B, C.
6. logical: lógico, booleano. Ejemplo: FALSE.

Para que devuelva el tipo de dato de una variable, se usa `typeof`:

```
typeof(1.5)
```

```
## [1] "double"
```

Se pueden convertir un tipo de dato en otro por medio de funciones que comienzan con `as..`

1. `as.integer()`: convertir a entero.
2. `as.double()`: convertir a decimal.
3. `as.character()`: convertir a cadena de texto.
4. `as.logical()`: convertir a booleano.

Ejemplo:

```
# Convirtiendo la variable b en una cadena de texto
b<-456 # integer
b<-as.character(b)
typeof(b)
```

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

```
# Como b es un texto, devuelve el resultado entre comillas
b
```

```
## [1] "456"
```

Si al aplicar el `as.` no se puede convertir el dato al tipo de dato deseado, retornará un NA.

```
c <- "abuela"
c <- as.integer(c)
```

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

Por otro lado, se puede determinar el tipo de dato de un dato con `is.:`

1. `is.integer()`: verifica si es entero.
2. `is.double()`: verifica si es decimal.
3. `is.character()`: verifica si es cadena de texto.
4. `is.logical()`: verifica si es booleano.

Ejemplo:

```
# Verificando si 432 es una cadena de texto.  
is.character(432)
```

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

NA y NULL

La diferencia entre NA y NULL es que NULL representa datos inexistentes y NA datos faltantes o no disponibles.

Es posible eliminar estos valores usando `drop_na()` de la librería tidy. Se puede usar en una sola columna o en todas las columnas del marco de datos:

```
drop_na(df, column = "nombre columna")
```

Si se especifica `column = NULL`, se eliminarán todas las filas que contengan un valor NA en el marco de datos.

Condicionales

if

```
if (condicion){  
  # operacion si la condicion es TRUE  
} else {  
  # operacion si la condicion es FALSE  
}
```

ifelse

```
ifelse(vector, valor_si_es_TRUE, valor_si_es_FALSE)
```

Ejemplo:

```
# El siguiente codigo imprimira en pantalla Par si el modulo entre el numero y 2 es cero, o impar si no  
n <- 1:6  
ifelse(n%%2==0,"Par", "Impar")
```

```
## [1] "Impar" "Par"   "Impar" "Par"   "Impar" "Par"
```

for

```
for (elemento in objeto){  
  # operacion con el elemento  
}
```

Ejemplo:


```

dado<-1:6
for (cara in dado){
  d<-dado^2
}
d

```

```
## [1] 1 4 9 16 25 36
```

while

```

while (condicion){
  # operaciones
  i <- i + 1 # se incrementa para evitar un loop infinito
}

```

Ejemplo 1:

```

num <- 1
while (num <10){
  print(num^2)
  num <- num+1
}

```

```

## [1] 1
## [1] 4
## [1] 9
## [1] 16
## [1] 25
## [1] 36
## [1] 49
## [1] 64
## [1] 81

```

Ejemplo 2:

```

auto <- 1
while (auto <10){
  print(paste("El numero de autos es", auto))
  auto <- auto + 1
}

```

```

## [1] "El numero de autos es 1"
## [1] "El numero de autos es 2"
## [1] "El numero de autos es 3"
## [1] "El numero de autos es 4"
## [1] "El numero de autos es 5"
## [1] "El numero de autos es 6"
## [1] "El numero de autos es 7"
## [1] "El numero de autos es 8"
## [1] "El numero de autos es 9"

```

Break y next

El comando **break** interrumpe un bucle, mientras que **next** avanza a la siguiente iteración del bucle saltándose la actual. Estos comandos se ocupan para **for** y **while**.

Ejemplo:

```
auto <- 1
bus <- 1
while (num <10){
  # Detiene el while cuando la suma de auto y bus es 8
  if(auto + bus == 8){
    break
  }
  print(paste("El numero de autos es", auto))
  print(paste("El numero de buses es", bus))
  bus <- bus + 1
  auto <- auto + 1
}
```

Repeat

repeat es un bucle que se repetirá un número específico de veces. Se usa **break** para detenerlo. Ejemplo:

```
valor<-0
repeat{
  valor<-valor+1
  if (valor ==5){
    break
  }
}
# Imprime el resultado
valor
```

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

lapply y sapply

lapply y **sapply** son funciones que sirven para simplificar operaciones de manipulación de datos en R. *##*
lapply

lapply tiene como función principal aplicar una función definida a cada elemento de una lista, donde la salida será una lista con los resultados.

Ejemplo 1:

```
lista <- list(pop = 1868,
              numeros = c(1,2,4,6,32,575,121,5))

# lapply
lapply(lista, class)
```

```
## $pop
## [1] "numeric"
##
## $numeros
## [1] "numeric"

# lapply
nombres <- c("Roberto", "Marta", "Matias", "Ignacio")
# devuelve el numero de letras de cada nombre
lapply(nombres, nchar)

## [[1]]
## [1] 7
##
## [[2]]
## [1] 6
##
## [[3]]
## [1] 6
##
## [[4]]
## [1] 7
```

Ejemplo 2:

```
# Creando un dataframe
df_ejemplo <- data.frame(
  col_1 = c(1, 2, 3),
  col_2 = c(4, 5, 6),
  col_3 = c(7, 8, 9)
)

# Aplicando el promedio (mean) a cada columna del dataframe
medias <- lapply(df_ejemplo, mean)
medias
```

```
## $col_1
## [1] 2
##
## $col_2
## [1] 5
##
## $col_3
## [1] 8
```

Para que devuelva un vector:

```
unlist(lapply(nombres, nchar))
```

```
## [1] 7 6 6 7
```

sapply

Por su parte, sapply también aplica una función a cada elemento de una lista, pero con la diferencia de que la salida es un formato más compacto, como un vector.

Ejemplo:

```
sapply(nombres, nchar)
```

```
## Roberto   Marta   Matias Ignacio  
##         7         6         6         7
```

Estructuras de datos

Una estructura de datos es una especie de contenedor que guarda datos en una posición específica. En R, las estructuras de datos más comunes son:

1. Vector
2. Matriz
3. Factores
4. Listas
5. DataFrame

Vector

Es la estructura de datos más básica en R. Almacena datos del mismo tipo y su única dimensión es el largo. Para crear un vector, se usa la función `c()` (concatenación).

Ejemplo:

```
# Creando un vector numerico  
v1<-c(1,2,3,4,5,6,7)  
v1
```

```
## [1] 1 2 3 4 5 6 7
```

```
# Vector numerico del 2 al 20  
v2<-c(2:20)  
v2
```

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

```
# Creando un vector de caracteres  
v3 <- c("r", "a", "j")  
v3
```

```
## [1] "r" "a" "j"
```

```
# Creando un vector logico
v4 <- c(TRUE, FALSE, FALSE, TRUE)
v4
```

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

Propiedades

```
# Vector
x<-c(3,4,5,4,3,2)
x
```

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

```
# Verifica si un vector es vector. Si es vector, devuelve TRUE
is.vector(x)
```

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

```
# Imprime el largo del vector
length(x)
```

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

```
# Imprime el tipo de vector
typeof(x)
```

```
## [1] "double"
```

```
# Imprime el tipo de dato
class(x)
```

```
## [1] "numeric"
```

```
# Nombra los elementos de un vector
names(x)<-c("a","b","c","d","e","f")
x
```

```
## a b c d e f
## 3 4 5 4 3 2
```

Operaciones

Se pueden realizar operaciones aritméticas con los vectores e incluso operaciones relacionales, devolviendo un TRUE o un FALSE en este caso.

```
v5<-c(2,4,6,8)
v6<-c(1,3,5,7)
```

```
# 1.
v5*2
```

```
## [1] 4 8 12 16
```

```
v6+2
```

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

```
# 2.
v6==3
```

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

```
# 3. Devuelve las posiciones de los valores que estan por sobre la media de v6
which (v6>=(mean(v6)))
```

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

```
# 4. Devuelve los valores que estan por sobre la media de v6
mayor_media <- which (v6>=(mean(v6)))
v6[mayor_media]
```

```
## [1] 5 7
```

```
# 5. Reemplaza un valor en el vector (el 8 por el 234)
v5[4] = 234
v5
```

```
## [1] 2 4 6 234
```

```
# 6. Repite tesis y agrega elementos de v6
paste("Tesis", v6)
```

```
## [1] "Tesis 1" "Tesis 3" "Tesis 5" "Tesis 7"
```

Para combinar dos vectores, se concatenan usando c():

```
vector_n <- c(vector_1, vector_2)
```

Extracción de valores

```
v5
```

```
## [1] 2 4 6 234
```

```
# Imprime el valor de la posicion 4 del vector v3  
v5[c(4)] # o v5[4]
```

```
## [1] 234
```

```
# Entrega un rango de valores del vector v3  
v5[1:3]
```

```
## [1] 2 4 6
```

```
# Elimina el valor de la posicion 1 (elimina el 2)  
v5[-1]
```

```
## [1] 4 6 234
```

```
# Elimina un rango de valores (elimina el 4 y 6)  
v5[-1:-3]
```

```
## [1] 234
```

Sucesiones

Para crear una sucesión en R se utiliza `seq(inicio, fin, by=numero)`:

```
# Sucesion que inicia en 1 y va de 2 en 2 hasta 20  
seq(1,20,by=2)
```

```
## [1] 1 3 5 7 9 11 13 15 17 19
```

```
# Sucesion que inicia en 3 y termina en 16  
seq(3:16)
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

Repetición

En R, se puede repetir un número n veces usando `rep(numero, num_de_repeticiones)`:

```
# Repite el 12 cinco veces  
rep(12, 5)
```

```
## [1] 12 12 12 12 12
```

```
# Repita casa 2 veces  
rep("casa", 2)
```

```
## [1] "casa" "casa"
```

```
# Repite la secuencia del 1 al 5 3 veces  
rep(1:5, 3)
```

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

Función all y any

Las funciones `all` y `any` se utilizan para hacer comprobaciones lógicas en vectores.

- `all()`: si todos los elementos de un vector son TRUE, devolverá TRUE. De lo contrario, si hay uno que sea FALSE, devolverá FALSE. Ejemplo:

```
# Comprueba si todos los elementos son mayores a 5  
y<-1:20  
all(y>5)
```

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

- `any()`: si al menos uno de los elementos es TRUE, devolverá TRUE. Si todos son FALSE, devolverá FALSE. Ejemplo:

```
# Comprueba si hay al menos un elemento menor a 5  
x<-1:10  
any(x<5)
```

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

Matriz

Una matriz es una estructura de datos que posee filas y columnas, donde la indexación se realiza mediante pares de índices, es decir, uno para la fila y otro para la columna. Además, contiene elementos del mismo tipo.

Para crear una matriz, se utiliza la función `matrix()`:

```
matrix(rango, nrow= n_filas, ncol=n_col)
```

En el rango, se puede insertar un vector.

Por ejemplo:

```
# Creando una matriz de datos NA  
z<-matrix(nrow=4,ncol=3)  
z
```



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

```
z1<-matrix(1:6, nrow=2, ncol=3)
z1
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Se pueden crear matrices con `rbind` y `cbind`:

```
# Vectores
vec1 <- 1:4
vec2 <- 5:8
# cbind usa cada vector como una columna
vec3 <- cbind(vec1, vec2)
vec3
```

```
##      vec1 vec2
## [1,]    1    5
## [2,]    2    6
## [3,]    3    7
## [4,]    4    8
```

```
# rbind usa cada vector como una fila
vec3 <- rbind(vec1, vec2)
vec3
```

```
##      [,1] [,2] [,3] [,4]
## vec1    1    2    3    4
## vec2    5    6    7    8
```

Propiedades

Usando las matrices anteriores, es posible cambiar el nombre de las filas y las columnas mediante los siguientes comandos.

Para ponerle nombre a las filas, se usa `rownames()`:

```
rownames(z1) <- c("Fila_1", "Fila_2")
z1
```

```
##      [,1] [,2] [,3]
## Fila_1    1    3    5
## Fila_2    2    4    6
```

Para nombrar las columnas se usa `colnames()`:

```
colnames(z1) <- c("Col_1", "Col_2", "Col_3")
z1
```

```
##           Col_1 Col_2 Col_3
## Fila_1      1     3     5
## Fila_2      2     4     6
```

```
m <- matrix(1:12, nrow=3, ncol=4)
m
```

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

```
m2 <- matrix(12:23, nrow=4, ncol=3)
m2
```

```
##      [,1] [,2] [,3]
## [1,]   12   16   20
## [2,]   13   17   21
## [3,]   14   18   22
## [4,]   15   19   23
```

```
# Devuelve la primera fila
m2[1,]
```

```
## [1] 12 16 20
```

```
# Devuelve la dimension de una matriz
dim(m)
```

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

```
# Transpone una matriz
t(m)
```

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

```
# Multiplicacion entre matrices
m %*% m2
```

```
##      [,1] [,2] [,3]
## [1,]  312  400  488
## [2,]  366  470  574
## [3,]  420  540  660
```

```
# Calcula la suma de los valores de cada fila  
rowSums(m2)
```

```
## [1] 48 51 54 57
```

```
# Calcula la media de las columnas  
colMeans(m2)
```

```
## [1] 13.5 17.5 21.5
```

Siendo A una matriz y B otra matriz:

```
# Producto cruzado  
crossprod(a,b) # t(A) %*% B  
tcrossprod(A,B) # A %*% t(B)
```

```
#Producto de Kronecker  
A %x% B
```

```
# Potencia  
A %>% 2
```

```
# Potencia elemento a elemento  
A^2
```

```
# Determinante  
det(A)
```

```
# Diagonal  
diag(A)
```

```
# Diagonal secundaria  
diag(apply(A,2,rev))
```

```
# Matriz diagonal  
diag(c(1,2,3,4,5))
```

```
# Matriz identidad  
diag(3)
```

```
# Autovalores  
# Se almacenan en el elemento values  
eigen(A)$values
```

```
# Autovectores  
# Se almacenan en el elemento vectors  
eigen(A)$vectors
```

Listas

Para crear una lista, se usa `list`:

```
mi_lista <- list("elemento1", "elemento2", 3, 4)
mi_lista
```

```
## [[1]]
## [1] "elemento1"
##
## [[2]]
## [1] "elemento2"
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] 4
```

Extraer elementos de una lista:

```
# Extrayendo el primer elemento
mi_lista[1]
```

```
## [[1]]
## [1] "elemento1"
```

Número de elementos de una lista:

```
length(mi_lista)
```

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

Para borrar elementos de una lista:

```
# Borrando el elemento 2 de mi_lista
mi_lista[[2]] <- NULL # es equivalente a mi_lista[-2]
mi_lista
```

```
## [[1]]
## [1] "elemento1"
##
## [[2]]
## [1] 3
##
## [[3]]
## [1] 4
```

Lista de listas

Es posible crear una lista de listas usando `list` y agregando listas dentro de su paréntesis:

```
lista_de_lista <- list(lista_1, lista_2, lista_3)
```

Ejemplo de una lista de listas:

```

# Lista vacia
lista_de_listas <- list()
lista_de_listas <- vector("list", length = 5)
# Creando lista de 5 listas con 5 elementos
for (j in 1:5){
  lista_de_listas[[j]] <- c(12,34,654,76,23)
}
lista_de_listas

```

```

## [[1]]
## [1] 12 34 654 76 23
##
## [[2]]
## [1] 12 34 654 76 23
##
## [[3]]
## [1] 12 34 654 76 23
##
## [[4]]
## [1] 12 34 654 76 23
##
## [[5]]
## [1] 12 34 654 76 23

```

Para acceder a sus elementos

```

# Primera lista
lista_de_listas[[1]]

```

```

## [1] 12 34 654 76 23

```

```

# Extrayendo el segundo elemento de la tercera lista
lista_de_listas[[3]][2]

```

```

## [1] 34

```

Concatenar listas

Se usan los siguientes códigos para concatenar listas:

```

# Los elementos de la 2da lista se agregan a la 1ra
append(lista_1, lista_2)

```

```

# Concatenando con c
c(lista_1, lista_2)

```

```

# Con do.call, se le puede aplicar una funcion a las listas, en este caso c concatena las listas
do.call(c, list(lista_1, lista_2))

```

Propiedades

```
# Elementos comunes en dos listas
intersect(lista_1, lista_2)

# Elementos diferentes en dos listas
setdiff(lista_1, lista_2) # Responde a que esta en lista_1 que no esta en lista_2

# Comparar elementos iguales en dos listas:
lista_1 %in% lista_2
# o
compare.list(lista_1, lista_2)

# Convertir una lista en un vector
unlist(lista, use.names=FALSE) # Convierte toda la lista a vector
# o
unlist(lista[[1]], use.names=FALSE) # Convierte el primer elemento a vector

# Convertir una lista en dataframe
data.frame(matrix(unlist(lista), nrow=length(lista), byrow=TRUE))
# o
do.call(rbind.data.frame, lista)
```

Dataframe

Para crear un dataframe, se usa `data.frame` y los nombres de las columnas se pueden definir con vectores:

```
datos <- data.frame(nom_columna = vector_1, nombre_columna2 = vector_2)
```

Ejemplo:

```
df <- data.frame(
  Nombre = c("Jorge", "Marta", "Carlos"),
  Edad = c(23, 35, 18),
  Puntos = c(10, 6, 5)
)
df
```

```
##   Nombre Edad Puntos
## 1  Jorge   23     10
## 2  Marta   35      6
## 3 Carlos   18      5
```

Mostrando los datos

```
# Mostrando los datos
View(df)

# Tipo de datos
str(df)
```

```

# Devuelve solo las primeras 6 filas
head(df)

# Devuelve las ultimas 6 filas
tail(df)

# Devuelve el numero de columnas y filas del df
glimpse(df)

# Devuelve un resumen estadistico
summary(df)

#
skim(df)

#
skim_without_charts(df)

```

Cargando conjunto de datos

Para cargar datos de R:

```
data(conjunto_de_datos)
```

Accediendo a los datos del dataframe

Luego de cargar el conjunto de datos, se pueden acceder a los datos del dataframe con:

```

df$nom_columna
# o
df[,numero_columna]

```

Ejemplo:

```

# Seleccionando la columna edad del dataframe
df$Edad

```

```
## [1] 23 35 18
```

```

# Seleccionando los datos que tienen una edad mayor a 19
df[df$Edad>19,]

```

```

##   Nombre Edad Puntos
## 1  Jorge   23     10
## 2  Marta   35      6

```

```

# Calculando la media
mean(df$Edad)

```

```
## [1] 25.33333
```

También, se pueden agregar filas y columnas a un dataframe con:

```
df$columna <- df$nueva_columna
```

O eliminar columnas

```
df_2 <- df[,-c(1,2)]
```

Filtros

Se puede filtrar datos con `subset()`:

```
# Seleccionando solo los datos del df donde los autos sean toyota
subset(df, autos=="toyota")
```

```
# Seleccionando solo dos datos del df donde los autos sean toyota y su año mayor o igual a 2002
subset(df, autos == "toyota" & año >= 2002)
```

O `select()`:

```
# Especifica ciertas columnas o las excluye con un -
select(columna_1, columna_5, columna_9)
```

```
# Las excluye con
select(-columna)
```

Usando `filter()`:

```
filter(df, autos=="toyota")
```

Ordenando los datos de un dataframe

```
# Se usa para elegir la variable que se quiere ordenar
arrange(df, columna)
# Ordena de forma descendiente
arrange(df, desc(columna))
```

```
# Agrupa
group_by(columna)
```

```
# Calculando la media de una columna
summarise(df, nueva_columna = mean(columna))
```

```
# Crea subconjuntos
subset(df, criterio)
```

```
# Crea nuevas variables que se calculan por medio de otras
mutate(df, operacion)
```


Modificando los nombres de las columnas

```
# Cambia de nombre las columnas
rename(nueva_columna = antigua_columna)

# Pone nombres en mayusculas
rename_witch(df, toupper)
# En minusculas
rename_with(df, tolower)

# Verifica si los nombres de las columnas son exclusivos y coherentes
clean_names()

# Devuelve los nombres de las columnas del df
col_names(df)

# Cambia nombres de las columnas
col_names(df)[nombre_columna_a_cambiar] <- c("columna_nueva")
```

Transformación de datos

```
# Divide los datos en columnas separadas
separate(df, columna_a_separar, into = c("nombre_nueva_columna", ...), sep= ",")

# Fusiona columnas
unite(df, "nombre_columna", columna_a_combinar, columna_a_combinar2, sep=",")

# Une dos dataframe, donde df_1 es el primer df a unir y df2 el segundo, by
# es o son las variables que se van a usar para unirlos y all.x indica si
# todos los registros del primer df deben incluirse en la union.
# Lo mismo para all.y, pero con el segundo df
merge(df_1, df_2, by = "variable", all.x = TRUE, all.y =TRUE, ...)
```

Canalización

Una canalización toma el resultado de una instrucción y lo convierte en la entrada de la siguiente instrucción:

```
Primero() %>% Segundo() %>%
  Tercero()
```

Función anidada

Es una función que está contenida dentro de otra función. Se leen desde adentro hacia afuera:

```
Tercero(Segundo(Primero(x)))
```

Gráficos

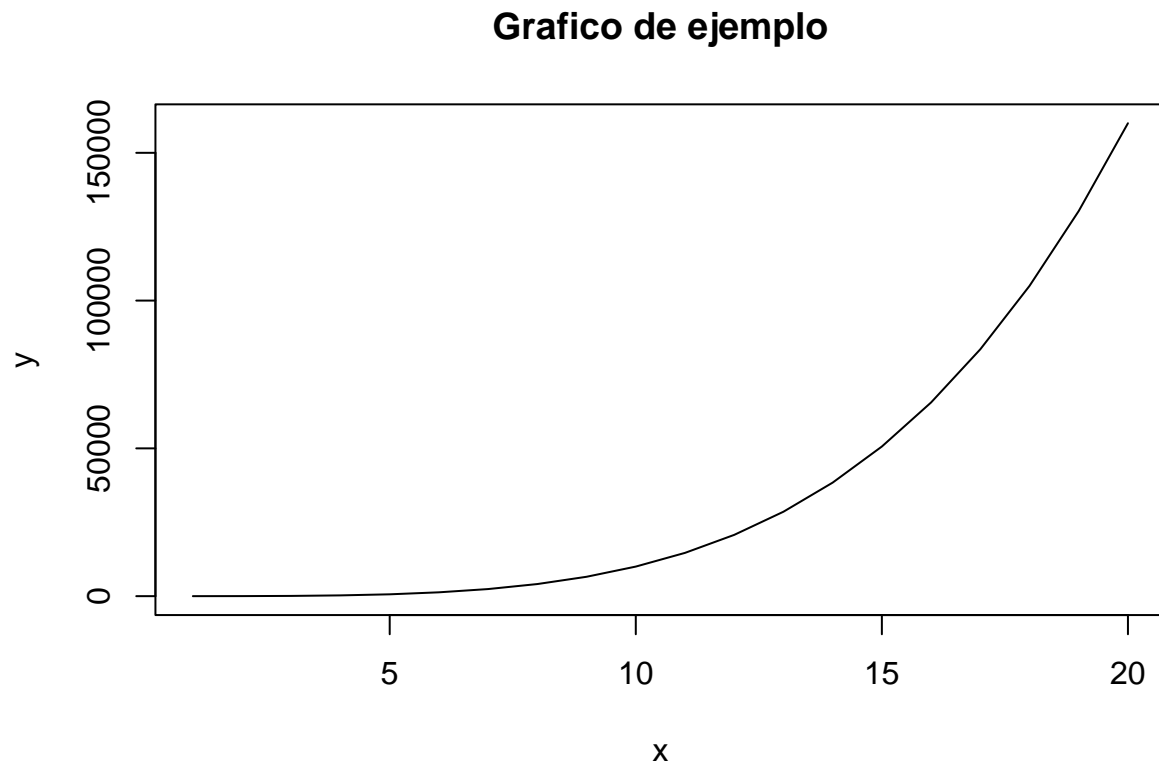
R tiene la función `plot(x,y,...)` para graficar. Recibe más argumentos, como: - `type()`: tipo de gráfica. p para puntos, l para líneas y h para histogramas. - `main`: título del gráfico. - `sub`: subtítulo. - `xlab`: título del eje x. - `ylab`: título del eje y.

Ejemplo 1:

```
plot(x,y,type="l", main="Grafico de ejemplo", xlab="meses", ylab="goles")
```

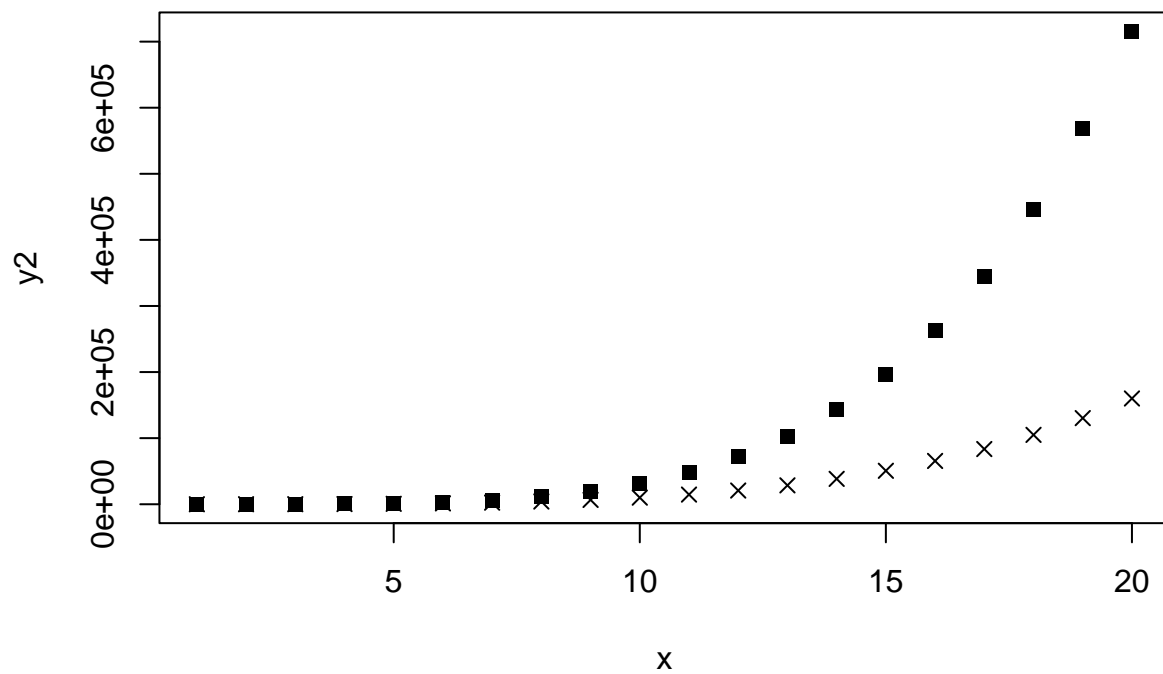
Ejemplo 2:

```
x<-1:20  
y<-x^4  
plot(x, y, main="Grafico de ejemplo", type="l")
```



Para poder hacer dos gráficas en una misma, se usa `points()`:

```
x<-1:20  
y2<-x^4.5  
plot(x,y2, pch=15)  
points(x,y, pch=4)
```



Con `pch` se pueden diferenciar las gráficas mediante símbolos. Mediante `col`, se le agregan colores:

```
x<-1:20
y2<-x^4.5
plot(x,y2, pch=15, col=50)
points(x,y, pch=4, col=60)
```

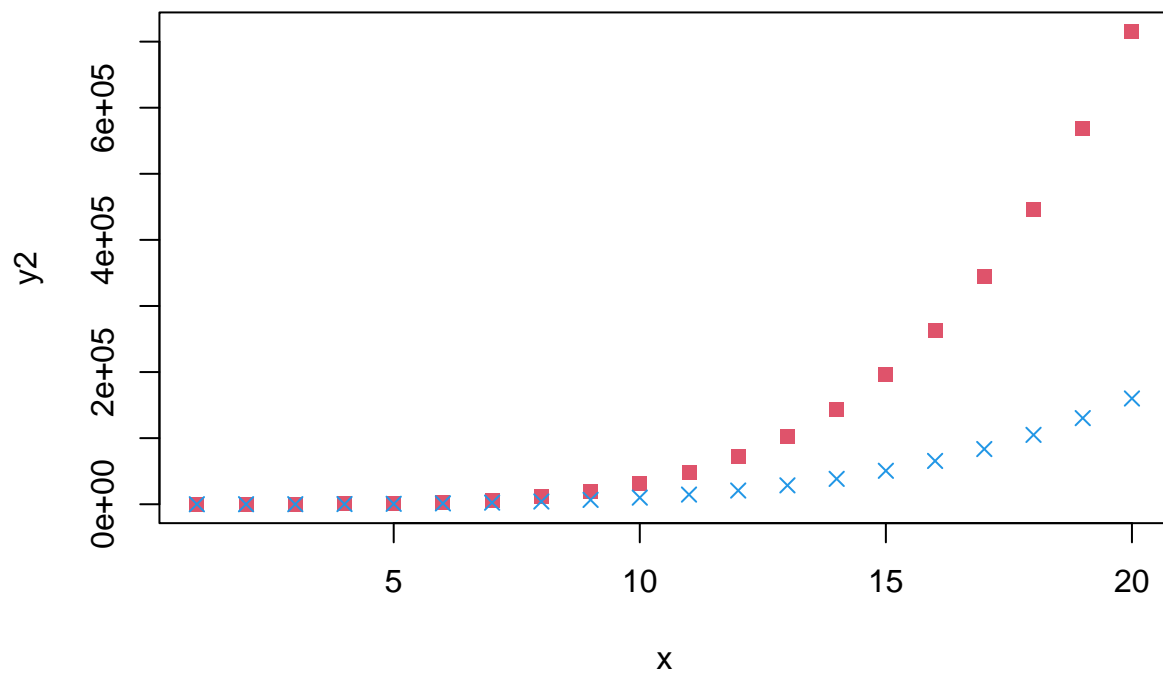
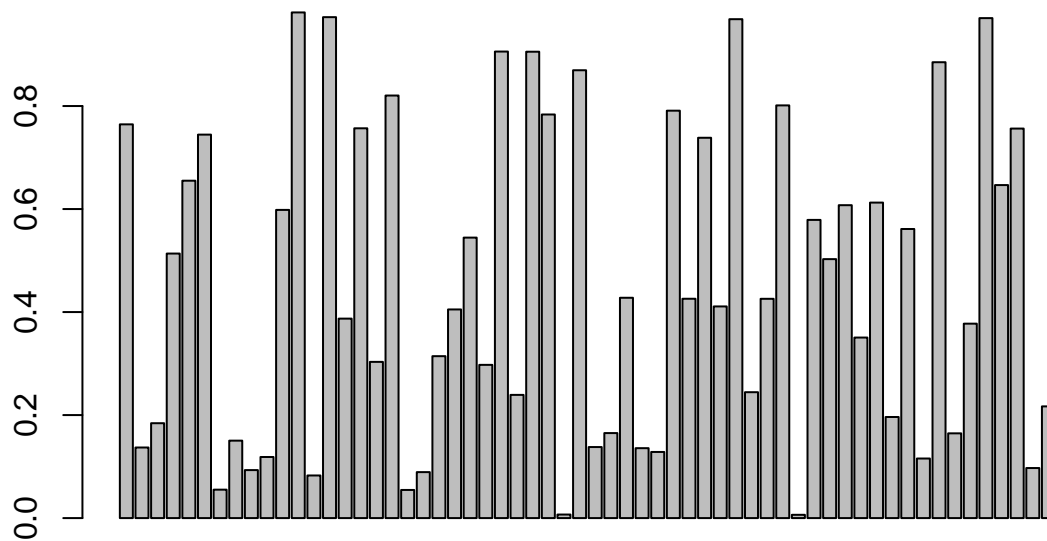


Gráfico de barras Los gráficos de barras se usan para representar la frecuencia de datos en una variable discreta. En R, se crean con `barplot()`

```
# Generando 60 numeros aleatorios
n1 <- runif(60)

barplot(n1, main="Ejemplo grafico de barras")
```

Ejemplo grafico de barras



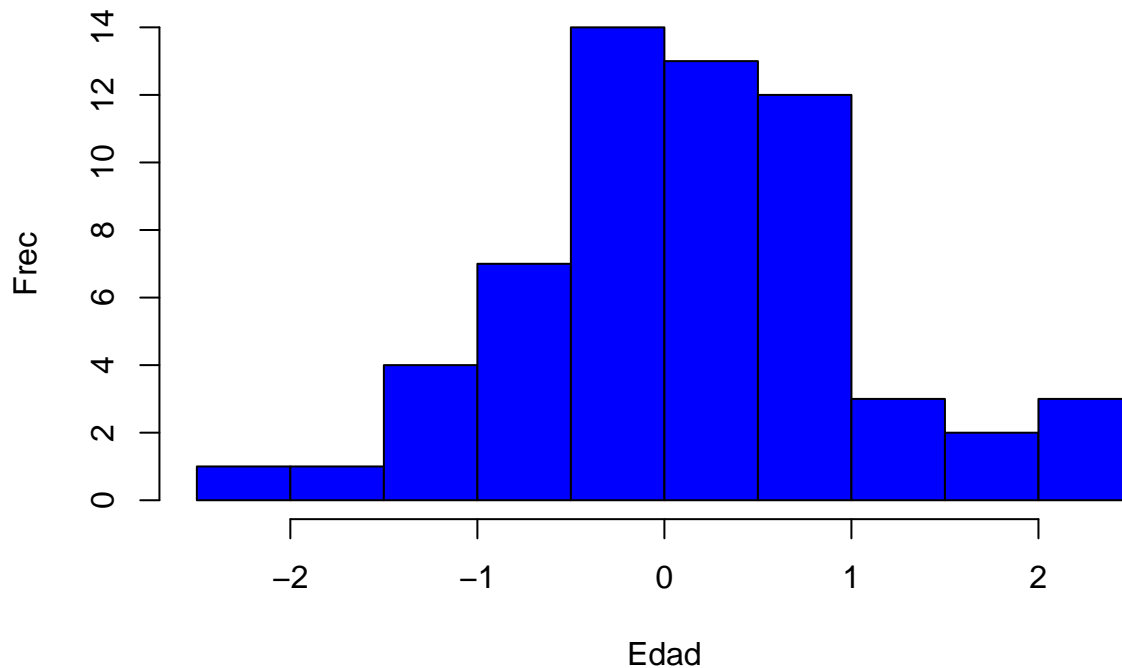
Histograma

Los histogramas se utilizan para visualizar la distribución de los datos, el cual lo hace por medio de barras que representan la frecuencia de datos. Para crearlo, se utiliza la función `hist()`:

```
# Generando 60 numeros aleatorios con distribucion normal
n2 <- rnorm(60)

hist(n2, main="Ejemplo de histograma", xlab="Edad", ylab="Frec", col="blue", breaks=10)
```

Ejemplo de histograma



`breaks` representa el número de barras del histograma. Es opcional.

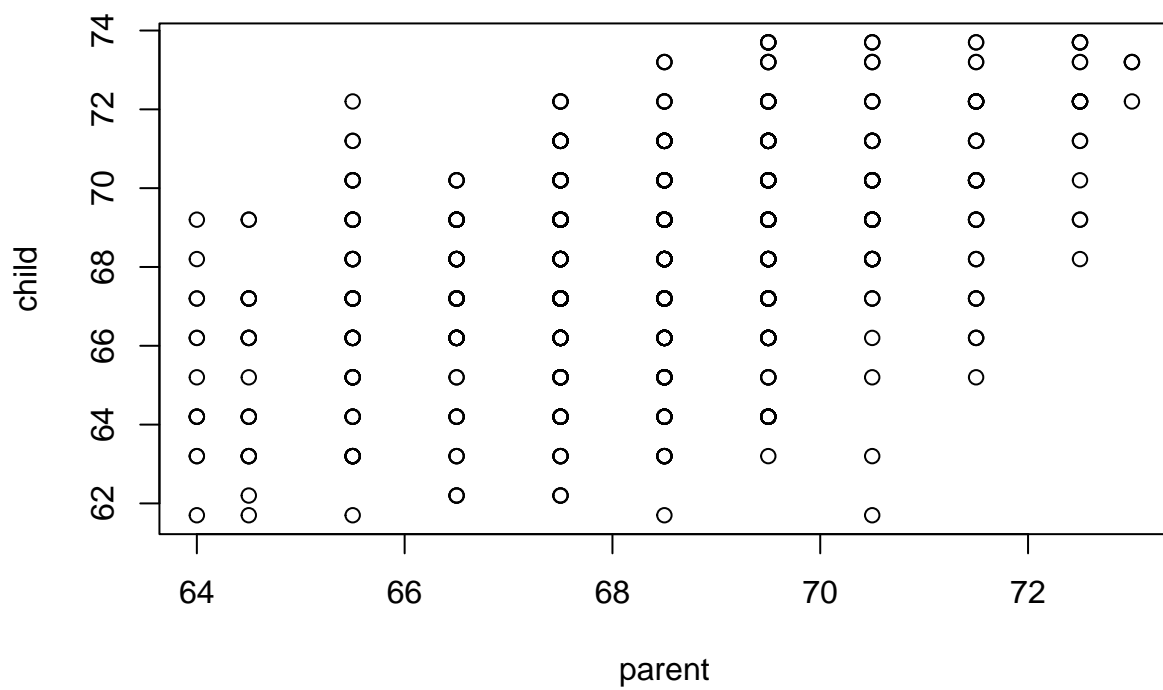
Ruido

La función `jitter` agrega una cantidad pequeña de ruido a las gráficas para evitar la superposición de puntos. Mediante el dataset Galton de R, se visualizará su funcionamiento:

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

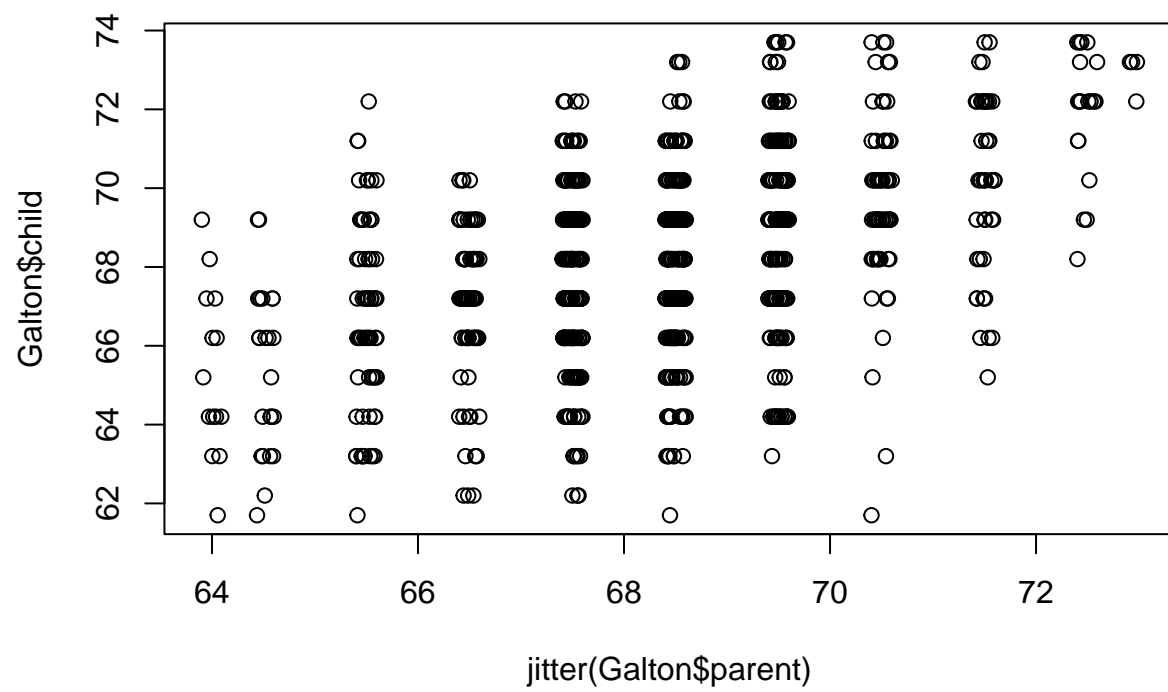
```
## Warning: package 'HistData' was built under R version 4.3.3
```

```
data(Galton)  
plot(Galton)
```



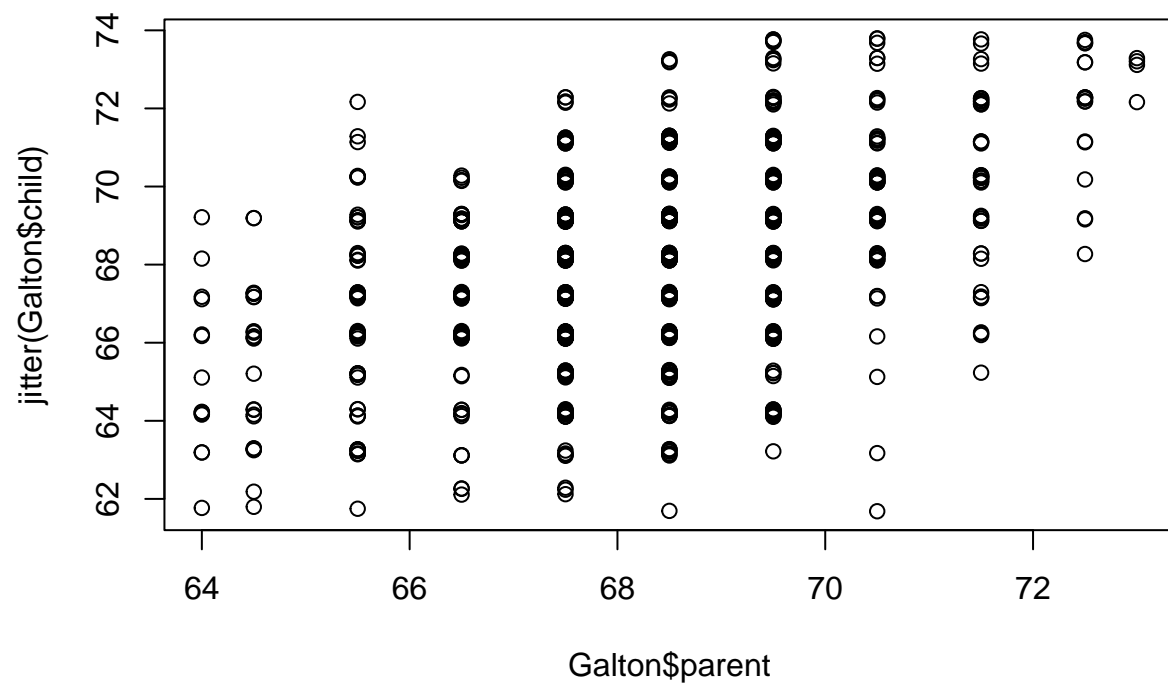
Agregando ruido a la columna parent:

```
plot(jitter(Galton$parent), Galton$child)
```



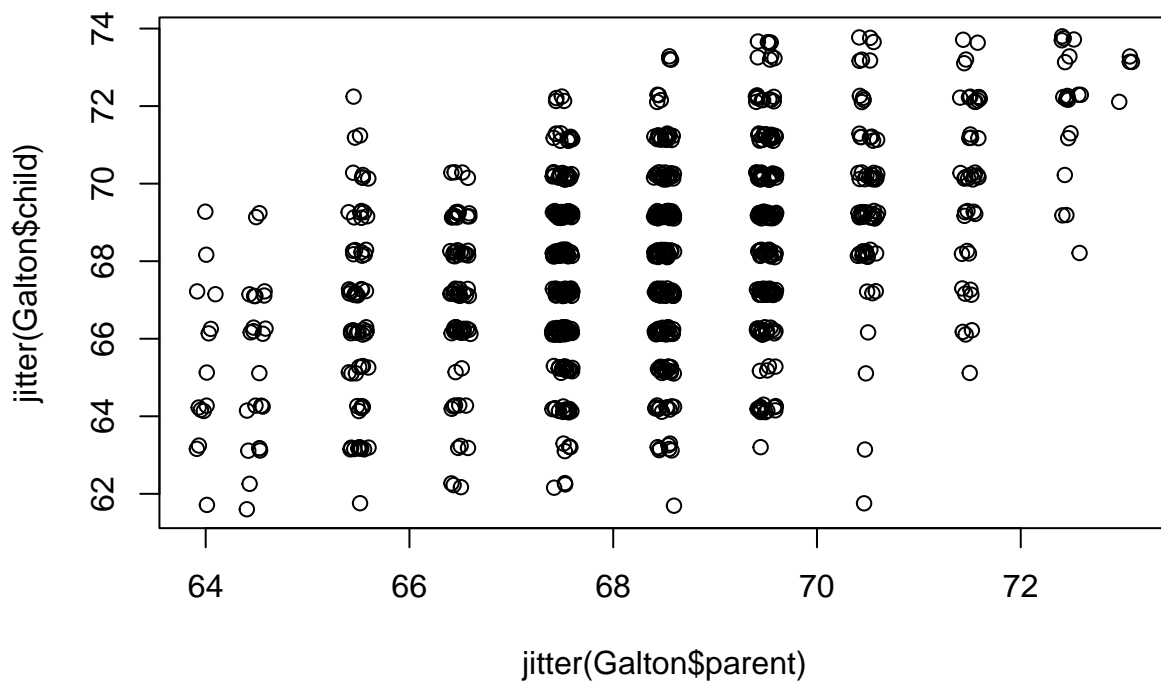
Agregando ruido a la columna child:

```
plot(Galton$parent, jitter(Galton$child))
```

Agregando ruido a ambas columnas:

```
plot(jitter(Galton$parent), jitter(Galton$child))
```



Importar y exportar datos

Para importar archivos en R, se usa el paquete `readr`:

```
library(readr)
```

Importar un archivo csv local

```
df <- read_csv("ruta_del_archivo.csv")
```

También, se pueden importar datos desde Import Dataset en Environment.

Para exportar datos:

Exportar un df

```
write.csv(nombre_df, file="nombre_archivo.csv")
```

Exportar un txt

```
write.table(datos, file="nombre_tabla.txt")
```

Exportar un archivo docx

```
capture.output(datos, file="nombre_archivo.doc")
```

Bibliografía

1. <https://r-coder.com/inicio/>
2. <https://bookdown.org/jboscomendoza/r-principiantes4/>