Capítulo 7: *RStudio*

Hasta ahora la consola ha sido algo útil para empezar a familiarizarse con el lenguaje R, pero existen otras maneras más sencillas de visualizar los comandos, editarlos y recibir las respuestas.

RStudio es una interfaz o IDE (Entorno Integrado de Desarrollo) diseñado especificamente para R. *RStudio* nos va a permitir escribir archivos ejecutables en los que podemos incluir todo un programa y editarlo sin tener que ir linea por linea.

Aqui está la guía para instalarlo:

Descargar *RStudio*

1. Abre tu navegador y dirigite al sitio al que pertenence el siguiente link: https://posit.co/
2. Hacer click en **DOWNLOAD RSTUDIO**
3. En la sección llamada **RStudio Desktop** hacer click donde diga **Download RStudio**
4. Hacer click en **Download Rstudio** para el sistema operativo que maneje.
5. En caso de no aparecer, buscar en la lista *All Installers* y guardar el archivo ejecutable.
6. Correr el archivo que acaba de descargar (.dmg para *macOS* o .exe para *Windows*) y continuar con las instrucciones de instalación que le aparezcan en su dispositivo al abrir el archivo.

* *Verificar el disco y usuario donde se guarda* R. *

Capítulo 8: Listas

Las listas pueden entenderse como vectores más complejos, cada elemento de la lista puede tener una estructura diferente, para crear una lista se utiliza la funición `list()`.

```
(list(1,62,993,40) -> lista0)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 62
##
## [[3]]
## [1] 993
##
## [[4]]
## [1] 40
```

La función `lenght` nos da la longitud de mi lista:

```
length(lista0)
```

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

Las listas pueden llegar a ser muy complejas y por ende, utiles para recopilación de datos. Estas pueden usarse para almacenar vectores, matrices, data frames, variables, incluso puede almacenar una lista propiamente. Por ejemplo:

```
(lista1 <- list(c(1,62,993, 40)))
```

```
## [[1]]
## [1]   1  62 993  40
```

¿Cuál es la diferencia entre esta lista y la anterior?

```
(lista2 <- list(c(4,5,9), 3:8))
```

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

```
(lista3 <- list(mtcars, 1:10))
```

```
## [[1]]
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
```

```
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10
```

`mtcars` *es un data frame que retomaremos más adelante.*

```
(lista5<- list(1,2,"a","b", NA, T, FALSE))
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] "a"
##
## [[4]]
## [1] "b"
##
## [[5]]
## [1] NA
##
## [[6]]
## [1] TRUE
##
## [[7]]
## [1] FALSE
```

Notar que la lista puede almacenar cualquier forma de estructura de datos y por ende cualquier tipo de dato válido en cada estructura.

En su forma más basica la lista puede usarse como vector,pero, las respuestas de la consola son diferentes a las que generaría un vector. Las diferencias más notable son la separación entre elementos y la posición del elemento de la lista entre doble corchete.

Respecto a la posición del elemento, esa es una respuesta por default, pero se puede asignar nombre a los elementos de la lista de 2 formas, la primera sería con 'names()':

```
names(lista3) <- c("mi_tabla", "mi_vector")
lista3
```

```
## $mi_tabla
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
##
## $mi_vector
##  [1]  1  2  3  4  5  6  7  8  9 10
```

La segunda es asignarle nombre cuando se crea la lista:

```
lista4 <- list(objeto = 1:10, carros = mtcars  ) #note el uso '='
names(lista4)
```

```
## [1] "objeto" "carros"
```

`names()` por si sola nada más nos da el nombre de la lista que colquemos entre parentesis.

Listas vacias

```
elist <- vector(mode='list', length = 5)
elist
```

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

El elemento `NULL` nos indica que ese elemento de la lista está vacio.

Acceso a datos de las listas

Para modificar los datos que tenemos en nuestras listas haremos lo siguiente:

```
elist[[1]] #con este comando vamos a acceder al elemento 1 de la lista
```

```
## NULL
```

```
elist[[1]] <- c(999,888)
elist
```

```
## [[1]]
## [1] 999 888
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
```

Note que usamos doble corchete, en *R* vamos a poder acceder a los elementos de las listas de la siguiente manera:

```
#Creamos una lista
lista6 <- list(c(88,99, 11,13), c(T,F,T))
```

Para acceder al elemento 1 :

```
lista6[1] #puede ser así
```

```
## [[1]]
## [1] 88 99 11 13
```

```
lista6[[1]] #o con doble corchete
```

```
## [1] 88 99 11 13
```

Para acceder al elemento 1 del elemento 1:

```
lista6[[1]][1] #aquí si es importante el doble corchete
```

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

Para filtrar del elemento 2 al 4 del elemento 1 de la lista6:

```
lista6[[1]][2:4]
```

```
## [1] 99 11 13
```

En el caso de que mi lista tenga un data frame o matriz como en la lista 3, vease el siguiente ejemplo para acceder a columnas y renglones:

```
lista3[[1]][1]#para acceder a la columna 1
```

```
##                      mpg
## Mazda RX4           21.0
## Mazda RX4 Wag       21.0
## Datsun 710          22.8
## Hornet 4 Drive      21.4
## Hornet Sportabout   18.7
## Valiant             18.1
## Duster 360          14.3
## Merc 240D           24.4
## Merc 230            22.8
## Merc 280            19.2
## Merc 280C           17.8
## Merc 450SE          16.4
## Merc 450SL          17.3
## Merc 450SLC         15.2
## Cadillac Fleetwood  10.4
## Lincoln Continental 10.4
## Chrysler Imperial   14.7
## Fiat 128            32.4
## Honda Civic         30.4
## Toyota Corolla      33.9
## Toyota Corona       21.5
## Dodge Challenger    15.5
## AMC Javelin         15.2
## Camaro Z28          13.3
## Pontiac Firebird    19.2
## Fiat X1-9           27.3
## Porsche 914-2       26.0
```

```
## Lotus Europa        30.4
## Ford Pantera L      15.8
## Ferrari Dino        19.7
## Maserati Bora       15.0
## Volvo 142E          21.4
```

```
lista3[[1]][, 1] #para acceder al renglón 1
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

También podemos acceder a los elementos de la lista de la siguiente manera:

Con el nombre que le hayamos puesto usando `$`:

```
lista3$mi_vector
```

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

O bien, como lo hemos estado haciendo, pero, en lugar de poner la posición del elemento, pondremos el nombre del elemento de la lista:

```
#con número de posición en la lista
lista3[[2]]
```

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

```
#con nombre del elemento
lista3[["mi_vector"]]
```

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

Por último veamos la función `unlist()`, la cual convierte una lista a un vector.

```
 unlist(lista0)
```

```
## [1]   1  62 993  40
```

Capítulo 9: Factores

En R , los factores se usan para trabajar con variables categóricas, es decir, variables que tienen un conjunto fijo y conocido de valores posibles. También son útiles cuando quieres mostrar vectores de caracteres en un orden no alfabético.

```
medallas <- c('Oro','Plata','Bronce','Oro','Plata','Plata','Plata','Oro')
factor(medallas)
```

```
## [1] Oro    Plata  Bronce Oro    Plata  Plata  Plata  Oro
## Levels: Bronce Oro Plata
```

Ahora el vector nos indica cuáles son los niveles del vector proporcionado. Esto puede ser útil para graficar o mucho más adelante, para entrenar modelos de clasificación.

```
medallas2 <- c('Oro','Plata','Oro','Plata','Plata','Plata','Oro')
fac_medallas <- factor(medallas2, levels = c('Oro','Plata','Bronce'))
fac_medallas
```

```
## [1] Oro   Plata Oro   Plata Plata Plata Oro
## Levels: Oro Plata Bronce
```

También le podemos indicar cuáles son los niveles de un factor, aunque el vector que estamos convirtiendo no los contenga. Esto lo hacemos con el argumento levels.

```
#Revisamos sólo los niveles
levels(fac_medallas)
```

```
## [1] "Oro"    "Plata"  "Bronce"
```

Incluso le podemos indicar a R que las categorías tienen orden y cuál es la jerarquía dentro de estas.

```
(fac_medallas_2 <- factor(medallas2, levels = c('Oro','Plata','Bronce'), ordered = T) )
```

```
## [1] Oro   Plata Oro   Plata Plata Plata Oro
## Levels: Oro < Plata < Bronce
```

Intuitivamente vemos que esa jerarquía no es correcta. Modifiquemosla con la función ordered().

```
ordered(fac_medallas_2, c('Bronce','Plata','Oro'))
```

```
## [1] Oro   Plata Oro   Plata Plata Plata Oro
## Levels: Bronce < Plata < Oro
```

Capítulo 10 : Data Frames

Los Data Frames son estructuras de datos fundamentales en R, especialmente útiles para trabajar con conjuntos de datos tabulares. Puedes pensar en ellos como tablas en una base de datos o hojas de cálculo en Excel que contiene renglones y columnas.

Ejemplo:

Primero vamos a crear vectores

```
x <- 10:1
y <- LETTERS[1:10]
z <- c("Actuaria", "Administración", "Ciencias Políticas",
       "Contaduría", "Ciencia de Datos", "Derecho", "Matemáticas",
       "Ingeniería Industrial", "Inteligencia Artificial", "Relaciones Internacionales" )
length(x)
```

```
## [1] 10
```

```
length(y)
```

```
## [1] 10
```

```
length(z)
```

```
## [1] 10
```

Despues, para definir un data frame usamos la función `data.frame()` y dentro del parentesis escribimos los vectores, así:

```
df <- data.frame(x,y,z)
df
```

```
##     x y                          z
## 1  10 A                   Actuaria
## 2   9 B             Administración
## 3   8 C          Ciencias Políticas
## 4   7 D                  Contaduría
## 5   6 E            Ciencia de Datos
## 6   5 F                    Derecho
## 7   4 G                 Matemáticas
## 8   3 H       Ingeniería Industrial
## 9   2 I     Inteligencia Artificial
## 10  1 J Relaciones Internacionales
```

```
# View(df) Es otra forma de ver la tabla en la consola o interfaz.
```

Es importante usar vectores de la misma longitud.

Se puede asignar nombre a las columnas de mi data frame, así como en las listas lo haciamos para los elementos:

```
m <- letters[1:3]
df1<- data.frame(columna1 = c(77,99,88), columna2 = m, columna3 = c(T,F,T) )
df1
```

```
##   columna1 columna2 columna3
## 1       77        a     TRUE
## 2       99        b    FALSE
## 3       88        c     TRUE
```

No importa si se define el vector antes o durante la creación del data frame.

Si le preguntamos la clase de la variable df1:

```
class(df1)
```

```
## [1] "data.frame"
```

Acceso a elementos del data frame

Para acceder a elementos especificos de mi data frame, se usa la siguiente estructura:

$$«Nombre\_del\_data\_frame»[renglon,\ columna]$$

Vease los siguientes ejemplos:

```
df[1:3, ] #los tres primeros renglones de mi df
```

```
##    x y                  z
## 1 10 A           Actuaria
## 2  9 B     Administración
## 3  8 C Ciencias Políticas
```

```
df[,c(2,3)] #las dos ultimas columnas de mi df
```

```
##    y                          z
## 1  A                   Actuaria
## 2  B             Administración
## 3  C         Ciencias Políticas
## 4  D                 Contaduría
## 5  E           Ciencia de Datos
## 6  F                    Derecho
## 7  G                Matemáticas
## 8  H      Ingeniería Industrial
## 9  I    Inteligencia Artificial
## 10 J Relaciones Internacionales
```

```
df[3] # la tercer columna de mi df
```

```
##                           z
## 1                   Actuaria
## 2             Administración
## 3         Ciencias Políticas
## 4                 Contaduría
## 5           Ciencia de Datos
```

```
## 6                    Derecho
## 7                Matemáticas
## 8       Ingeniería Industrial
## 9     Inteligencia Artificial
## 10 Relaciones Internacionales
```

```
df1[2:3,2] #los ultimos dos renglones de mi segunda columna
```

```
## [1] "b" "c"
```

R mantiene una serie de matrices de ejemplo cargadas en la memoria en todo momento. En nuestro caso de estudio, para la exploración de datos, emplearemos el data frame `mtcars`.

```
?mtcars
```

El signo de interrogación antes del nombre nos va a dar la información del data frame

Filtros

Ya hemos visto anteriormente como acceder a renglones y columnas especificas dentro del data frame, a eso se le puede decir que es un filtro. Los filtros dentro de nuestra estructura de datos van a ser utiles para explorar la propia estructura y centrarnos en lo que nos interese analizar. Ahora vamos a ver filtros mas especificos para el data frame:

Para analizar las dimensiones asi como el número de renglones y columnas del data frame vamos a usar las siguientes funciones:

- dim(). Me va a dar las dimensiones de mi matriz o data frame.
- ncol(). Me va a dar el número de columnas.
- nrow(). Me va a dar el número de filas o renglones.
- colnames(). Me va a dar el nombre de las columnas.
- summary(). Me va a generar resúmenes de datos a partir de estadísticos descriptivos.

Ejemplo

```
dim(mtcars)
```

```
## [1] 32 11
```

```
ncol(mtcars)
```

```
## [1] 11
```

```
nrow(mtcars)
```

```
## [1] 32
```

```
dim(mtcars)[1] #otra manera de acceder al número de renglones
```

```
## [1] 32
```

```
dim(mtcars)[2] #lo mismo para el número de columnas
```

```
## [1] 11
```

```
colnames(mtcars) #notar que la salida es en forma de vector
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

```r
summary(mtcars)
```

```
##       mpg             cyl             disp             hp
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
##       drat             wt             qsec             vs
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000
##        am              gear            carb
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
##  Median :0.0000   Median :4.000   Median :2.000
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

Note como la función summary() me da información muy importante de las columnas de mi data frame.

Para filtrar renglones y columnas:

```r
mtcars[1:5, ] #filtrar los primeros 5 renglones
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
```

```r
mtcars[28:32, ] #filtrar los ultimos 5 renglones
```

```
##                mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

```r
mtcars[ , 1:3] #filtrar las primeras 3 columnas
```

```
##                    mpg cyl  disp
## Mazda RX4          21.0   6 160.0
## Mazda RX4 Wag      21.0   6 160.0
## Datsun 710         22.8   4 108.0
## Hornet 4 Drive     21.4   6 258.0
## Hornet Sportabout  18.7   8 360.0
## Valiant            18.1   6 225.0
```

```
## Duster 360          14.3   8 360.0
## Merc 240D           24.4   4 146.7
## Merc 230            22.8   4 140.8
## Merc 280            19.2   6 167.6
## Merc 280C           17.8   6 167.6
## Merc 450SE          16.4   8 275.8
## Merc 450SL          17.3   8 275.8
## Merc 450SLC         15.2   8 275.8
## Cadillac Fleetwood  10.4   8 472.0
## Lincoln Continental 10.4   8 460.0
## Chrysler Imperial   14.7   8 440.0
## Fiat 128            32.4   4  78.7
## Honda Civic         30.4   4  75.7
## Toyota Corolla      33.9   4  71.1
## Toyota Corona       21.5   4 120.1
## Dodge Challenger    15.5   8 318.0
## AMC Javelin         15.2   8 304.0
## Camaro Z28          13.3   8 350.0
## Pontiac Firebird    19.2   8 400.0
## Fiat X1-9           27.3   4  79.0
## Porsche 914-2       26.0   4 120.3
## Lotus Europa        30.4   4  95.1
## Ford Pantera L      15.8   8 351.0
## Ferrari Dino        19.7   6 145.0
## Maserati Bora       15.0   8 301.0
## Volvo 142E          21.4   4 121.0
```

Pero esta no es la única manera de filtrar los primeros o ultimos renglones del data frame.

Existen funciones que sirven para visualizar las primeras 6 lineas y las útlimas 6 lineas del data frame:

Función `head()`:

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
head(mtcars,3)
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

Notar que al agregar un número como segundo argumento de esa función me da el el número de las primeras n filas que quiera ver.

Luego esta la función `tail()`:

```
tail(mtcars)
```

```
##                  mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

```
tail(mtcars,3) #los últimos 3 renglones.
```

```
##                mpg cyl disp  hp drat   wt qsec vs am gear carb
## Ferrari Dino  19.7   6  145 175 3.62 2.77 15.5  0  1    5    6
## Maserati Bora 15.0   8  301 335 3.54 3.57 14.6  0  1    5    8
## Volvo 142E    21.4   4  121 109 4.11 2.78 18.6  1  1    4    2
```

También se puede consultar valores específicos:

```
mtcars[1,1]
```

```
## [1] 21
```

```
mtcars[2,5]
```

```
## [1] 3.9
```

```
mtcars[,c(1,2)]
```

```
##                     mpg cyl
## Mazda RX4          21.0   6
## Mazda RX4 Wag      21.0   6
## Datsun 710         22.8   4
## Hornet 4 Drive     21.4   6
## Hornet Sportabout  18.7   8
## Valiant            18.1   6
## Duster 360         14.3   8
## Merc 240D          24.4   4
## Merc 230           22.8   4
## Merc 280           19.2   6
## Merc 280C          17.8   6
## Merc 450SE         16.4   8
## Merc 450SL         17.3   8
## Merc 450SLC        15.2   8
## Cadillac Fleetwood 10.4   8
## Lincoln Continental 10.4  8
## Chrysler Imperial  14.7   8
## Fiat 128           32.4   4
## Honda Civic        30.4   4
## Toyota Corolla     33.9   4
## Toyota Corona      21.5   4
## Dodge Challenger   15.5   8
## AMC Javelin        15.2   8
## Camaro Z28         13.3   8
## Pontiac Firebird   19.2   8
## Fiat X1-9          27.3   4
## Porsche 914-2      26.0   4
```

```
## Lotus Europa        30.4   4
## Ford Pantera L      15.8   8
## Ferrari Dino        19.7   6
## Maserati Bora       15.0   8
## Volvo 142E          21.4   4
```

```
mtcars[c(1,2,3),]
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

```
mtcars[c(1,2,3),c(1,2)]
```

```
##                mpg cyl
## Mazda RX4     21.0   6
## Mazda RX4 Wag 21.0   6
## Datsun 710    22.8   4
```

Consultas más avanzadas

La estructura para hacer consultas más precisas dentro del data frame va a ser similar a la que haciamos para consultar renglones en específico.

Primero, para consultar columnas con nombres específicos lo que vamos a hacer es utilizar la siguiente estructura:

«Nombre_del_data_frame»$*«columna_del_data_frame»*

Ejemplo:

```
mtcars$mpg #columna de mpg y la respuesta es un vector
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

```
#Uso de tab para ver las columnas después del signo $
```

Pero si solo pudieran filtrarse sería impráctico, es por eso que vamos a recordar la estructura para filtrar renglones y columnas, pero ahora vamos con condiciones dentro del corchete:

```
mtcars[mtcars$cyl==4, ]
```

```
##                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

Note como la consola busca cumplir con la condición en los renglones y da de respuesta los TRUE que cumplen con la condición, en otras palabras, la respuesta es todos los renglones que cumplen esa condición.

Más ejemplos:

```
mtcars[mtcars$hp > 200, ]
```

```
##                      mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Duster 360          14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
## Cadillac Fleetwood  10.4   8  472 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8  440 230 3.23 5.345 17.42  0  0    3    4
## Camaro Z28          13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
## Ford Pantera L      15.8   8  351 264 4.22 3.170 14.50  0  1    5    4
## Maserati Bora       15.0   8  301 335 3.54 3.570 14.60  0  1    5    8
```

```
mtcars[mtcars$hp > 400, ]
```

```
##  [1] mpg  cyl  disp hp   drat wt   qsec vs   am   gear carb
## <0 rows> (or 0-length row.names)
```

```
#note la respuesta cuando ningún renglon cumple la condición
nrow(mtcars[mtcars$hp > 200, ]) #se puede combinar con funciones básicas de df
```

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

```
#esta en particular me da el número de columnas que cumple la condición.
mtcars[mtcars$cyl <6,]
```

```
##                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

También hay otra forma de consultar las columnas por nombre:

```
mtcars[,c("cyl", "mpg")]
```

```
##                   cyl  mpg
## Mazda RX4           6 21.0
## Mazda RX4 Wag       6 21.0
## Datsun 710          4 22.8
## Hornet 4 Drive      6 21.4
## Hornet Sportabout   8 18.7
## Valiant             6 18.1
## Duster 360          8 14.3
## Merc 240D           4 24.4
```

```
## Merc 230              4 22.8
## Merc 280              6 19.2
## Merc 280C             6 17.8
## Merc 450SE            8 16.4
## Merc 450SL            8 17.3
## Merc 450SLC           8 15.2
## Cadillac Fleetwood    8 10.4
## Lincoln Continental   8 10.4
## Chrysler Imperial     8 14.7
## Fiat 128              4 32.4
## Honda Civic           4 30.4
## Toyota Corolla        4 33.9
## Toyota Corona         4 21.5
## Dodge Challenger      8 15.5
## AMC Javelin           8 15.2
## Camaro Z28            8 13.3
## Pontiac Firebird      8 19.2
## Fiat X1-9             4 27.3
## Porsche 914-2         4 26.0
## Lotus Europa          4 30.4
## Ford Pantera L        8 15.8
## Ferrari Dino          6 19.7
## Maserati Bora         8 15.0
## Volvo 142E            4 21.4
```

Hasta ahora, hemos visto condiciones relativamente faciles de la exploración y manipulación de datos, sin embargo los siguientes ejemplos van a explorar otras funciones para la manipulación de datos y texto en las siguientes secciones:

Manipulación de datos

Hasta ahora ya hemos visto las maneras de acceder a un valor específico de una columna en nuestro data frame. La siguiente linea de código, por ejemplo, nos devuelve el vector correspondiente a la columna 'mpg'.

```
mtcars$mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

Al tratarse de un vector, podemos aplicar todas las operaciones y técnicas conocidas. Si quisiéramos acceder al segundo elemento, escribiríamos:

```
mtcars$mpg[2]
```

```
## [1] 21
```

```
mtcars[2,1] #Otra alternativa
```

```
## [1] 21
```

A continuación, crearemos una nueva columna. Utilizamos la notación $ con un nombre de columna inexistente y el operador de asignación <- para asignar valores a esta nueva columna:

```
mtcars$like <- rep(0, nrow(mtcars))
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb like
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4    0
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4    0
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1    0
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1    0
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2    0
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1    0
```

En este ejemplo, usamos la función `rep()` para repetir el valor 0 tantas veces como filas tenga mtcars. Sin embargo, en R, también podemos asignar un único valor que se reciclará a lo largo de toda la columna:

```
mtcars$like <- 0
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb like
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4    0
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4    0
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1    0
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1    0
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2    0
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1    0
```

Para modificar valores específicos de esta columna, seguimos el mismo procedimiento utilizado para vectores. Seleccionamos el elemento deseado y le asignamos un nuevo valor:

```
mtcars$like[18] <- 1
mtcars$like[12] <- 1
mtcars$like[2] <- 1
mtcars$like[28] <- 1
mtcars$like[20] <- 1
mtcars$like[21] <- 1
mtcars$like
```

```
##  [1] 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0
```

Dado que la columna es un vector, podemos aplicar las mismas operaciones que conocemos para vectores:

```
sum(mtcars$like)
```

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

```
max(mtcars$cyl)
```

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

La primera línea suma los valores de la columna 'like', mientras que la segunda encuentra el valor máximo en la columna 'cyl'.

Para aplicar condiciones lógicas y filtrar datos, podemos ejecutar:

```
mtcars$cyl >=8
```

```
##  [1] FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE
## [13]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
## [25]  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
```

Esto ya lo habíamos visto antes, es un filtro para la columna `cyl`. Recordemos que esto devuelve un vector booleano indicando si cada elemento de mtcars$cyl cumple con la condición. Podemos usar este vector booleano para filtrar filas como en ejemplos anteriores:

```
mtcars[mtcars$cyl >= 8, ]
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb like
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2    0
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4    0
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3    1
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3    0
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3    0
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4    0
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4    0
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4    0
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2    0
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2    0
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4    0
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2    0
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4    0
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8    0
```

Para seleccionar columnas específicas junto con esta condición, usamos un vector de índices o nombres de columnas:

```
#Un vector de índices columnas
mtcars[mtcars$cyl >=8 & mtcars$disp > 400, c(1,4,5)]
```

```
##                      mpg  hp drat
## Cadillac Fleetwood  10.4 205 2.93
## Lincoln Continental 10.4 215 3.00
## Chrysler Imperial   14.7 230 3.23
```

```
#Un rango de índices columnas
mtcars[mtcars$cyl >=8 & mtcars$disp > 400, 2:5]
```

```
##                     cyl disp  hp drat
## Cadillac Fleetwood    8  472 205 2.93
## Lincoln Continental   8  460 215 3.00
## Chrysler Imperial     8  440 230 3.23
```

```
#Un vector con nombres de columnas
mtcars[mtcars$cyl >=8 & mtcars$disp > 400, c('mpg','cyl', 'disp')]
```

```
##                      mpg cyl disp
## Cadillac Fleetwood  10.4   8  472
## Lincoln Continental 10.4   8  460
## Chrysler Imperial   14.7   8  440
```

Para seleccionar una sola columna, utilizamos el operador $ después de los corchetes:

```
mtcars[mtcars$cyl >=8 & mtcars$disp > 400,]$mpg
```

```
## [1] 10.4 10.4 14.7
```

Si queremos asignar estos resultados a una nueva variable, lo hacemos de la siguiente manera:

```
cars_4_cyl <- mtcars[mtcars$cyl == 4, ]
head(cars_4_cyl)
```

```
##                 mpg cyl  disp hp drat    wt  qsec vs am gear carb like
## Datsun 710     22.8   4 108.0 93 3.85 2.320 18.61  1  1    4    1    0
## Merc 240D      24.4   4 146.7 62 3.69 3.190 20.00  1  0    4    2    0
## Merc 230       22.8   4 140.8 95 3.92 3.150 22.90  1  0    4    2    0
## Fiat 128       32.4   4  78.7 66 4.08 2.200 19.47  1  1    4    1    1
## Honda Civic    30.4   4  75.7 52 4.93 1.615 18.52  1  1    4    2    0
## Toyota Corolla 33.9   4  71.1 65 4.22 1.835 19.90  1  1    4    1    1
```

Hasta ahora, estas operaciones no han modificado el data frame original `mtcars`. Las modificaciones ocurren sólo cuando hacemos asignaciones.

Ahora, vease el siguiente ejemplo que usa herramientas sobre la creación de columnas y números pseudoaleatorios para agregar una columna "tank", que indicará el tamaño del tanque de gasolina de los coches:

```
set.seed(13)
cars_4_cyl$tank <- round(rnorm(nrow(cars_4_cyl), 18, 5))
cars_4_cyl
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb like tank
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1    0   21
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2    0   17
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2    0   27
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1    1   19
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2    0   24
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1    1   20
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1    1   24
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1    0   19
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2    0   16
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2    1   24
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2    0   13
```

Aquí, `rnorm()` genera números con una media de 18 y una desviación estándar de 5, y `round()` redondea estos números. La cantidad de números generados corresponde al número de filas en cars_4_cyl. Estas dos últimas funciones se retomaran más adelante.

Podemos ver un resumen de esta nueva columna:

```
summary(cars_4_cyl$tank)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   13.00   18.00   20.00   20.36   24.00   27.00
```

Como las columnas de un data frame son vectores del mismo tamaño, podemos realizar operaciones entre ellas. Para calcular la distancia máxima de cada coche, multiplicamos la columna mpg por la columna tank:

```
cars_4_cyl$distancia_maxima <- cars_4_cyl$mpg * cars_4_cyl$tank
summary(cars_4_cyl$distancia_maxima)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   278.2   447.4   518.7   544.6   646.8   729.6
```

Finalmente, podemos filtrar el data frame con múltiples condiciones. Por ejemplo, para encontrar filas donde mpg sea mayor a 30 y la distancia_maxima sea menor a 400:

```
cars_4_cyl[cars_4_cyl$mpg > 30 & cars_4_cyl$distancia_maxima < 400, ]
```

```
##  [1] mpg              cyl              disp             hp
##  [5] drat             wt               qsec             vs
##  [9] am               gear             carb             like
## [13] tank             distancia_maxima
## <0 rows> (or 0-length row.names)
```

Esto devolverá un data frame sin filas, ya que ninguna cumple ambas condiciones.

Para asignar los nombres de marca y modelo a una nueva columna:

```
cars_4_cyl$marca_modelo <- rownames(cars_4_cyl)
head(cars_4_cyl)
```

```
##                 mpg cyl  disp hp drat    wt  qsec vs am gear carb like tank
## Datsun 710     22.8   4 108.0 93 3.85 2.320 18.61  1  1    4    1    0   21
## Merc 240D      24.4   4 146.7 62 3.69 3.190 20.00  1  0    4    2    0   17
## Merc 230       22.8   4 140.8 95 3.92 3.150 22.90  1  0    4    2    0   27
## Fiat 128       32.4   4  78.7 66 4.08 2.200 19.47  1  1    4    1    1   19
## Honda Civic    30.4   4  75.7 52 4.93 1.615 18.52  1  1    4    2    0   24
## Toyota Corolla 33.9   4  71.1 65 4.22 1.835 19.90  1  1    4    1    1   20
##                distancia_maxima   marca_modelo
## Datsun 710                478.8     Datsun 710
## Merc 240D                 414.8      Merc 240D
## Merc 230                  615.6       Merc 230
## Fiat 128                  615.6       Fiat 128
## Honda Civic               729.6    Honda Civic
## Toyota Corolla            678.0 Toyota Corolla
```

Otra forma de ver el data frame completo es con la función `View()`, que abre una ventana separada para explorar los datos:

```
View(cars_4_cyl)
```

Ahora podemos ver que nuestra nueva columna con los nombres de marca y modelo está incluida en el data frame.

Manipulación de texto

La manipulación de texto en R es una tarea común y se puede realizar utilizando diversas funciones y paquetes. Tiene diversos usos, pero por ahora nos centraremos en las herramientas que vienen por defecto.

Convertir nombre de filas a columnas

Primero, podemos convertir los nombres de las filas de mtcars a una columna para facilitar la manipulación del texto:

```
mtcars$car_name <- rownames(mtcars)
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb like
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4    0
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4    1
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1    0
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1    0
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2    0
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1    0
##                             car_name
## Mazda RX4                  Mazda RX4
## Mazda RX4 Wag          Mazda RX4 Wag
## Datsun 710                Datsun 710
## Hornet 4 Drive        Hornet 4 Drive
## Hornet Sportabout Hornet Sportabout
## Valiant                     Valiant
```

Convertir mayúsculas a minúsculas

Podemos convertir los nombres de coches a mayúsculas o minúsculas usando `toupper()`o `tolower()`

```
mtcars$car_name_upper <- toupper(mtcars$car_name)
mtcars$car_name_lower <- tolower(mtcars$car_name)
head(mtcars[, c("car_name", "car_name_upper", "car_name_lower")])
```

```
##                             car_name    car_name_upper    car_name_lower
## Mazda RX4                  Mazda RX4         MAZDA RX4         mazda rx4
## Mazda RX4 Wag          Mazda RX4 Wag     MAZDA RX4 WAG     mazda rx4 wag
## Datsun 710                Datsun 710        DATSUN 710        datsun 710
## Hornet 4 Drive        Hornet 4 Drive    HORNET 4 DRIVE    hornet 4 drive
## Hornet Sportabout Hornet Sportabout HORNET SPORTABOUT hornet sportabout
## Valiant                     Valiant           VALIANT           valiant
```

Susitituit texto

Usamos `gsub()` para reemplazar partes de los nombres de los coches. Por ejemplo, podemos reemplazar "Mazda" por "Toyota":

```
mtcars$car_name_replaced <- gsub("Mazda", "Toyota", mtcars$car_name)
head(mtcars[, c("car_name", "car_name_replaced")])
```

```
##                             car_name car_name_replaced
## Mazda RX4                  Mazda RX4        Toyota RX4
## Mazda RX4 Wag          Mazda RX4 Wag    Toyota RX4 Wag
```

```
## Datsun 710               Datsun 710        Datsun 710
## Hornet 4 Drive       Hornet 4 Drive    Hornet 4 Drive
## Hornet Sportabout Hornet Sportabout Hornet Sportabout
## Valiant                     Valiant           Valiant
```

Dvidir y unir texto

Podemos dividir los nomnbres de los coches en palabras individuales utilizando `strsplit()` y unirls d nuevo con `paste()`.

```r
# Dividir nombres en palabras
split_names <- strsplit(mtcars$car_name, " ")
split_names[1:5]
```

```
## [[1]]
## [1] "Mazda" "RX4"
##
## [[2]]
## [1] "Mazda" "RX4"   "Wag"
##
## [[3]]
## [1] "Datsun" "710"
##
## [[4]]
## [1] "Hornet" "4"      "Drive"
##
## [[5]]
## [1] "Hornet"     "Sportabout"
```

```r
# Unir palabras con un guion
mtcars$car_name_hyphen <- sapply(split_names, function(x) paste(x, collapse = "-"))
head(mtcars[, c("car_name", "car_name_hyphen")])
```

```
##                           car_name   car_name_hyphen
## Mazda RX4                 Mazda RX4         Mazda-RX4
## Mazda RX4 Wag         Mazda RX4 Wag     Mazda-RX4-Wag
## Datsun 710               Datsun 710        Datsun-710
## Hornet 4 Drive       Hornet 4 Drive    Hornet-4-Drive
## Hornet Sportabout Hornet Sportabout Hornet-Sportabout
## Valiant                     Valiant           Valiant
```

Extraer subcadenas

Podemos extraer ciertas partes de los nombres usando substr(). Por ejemplo, extraer los primeros 5 caracteres de cada nombre de coche:

```r
mtcars$car_name_substr <- substr(mtcars$car_name, 1, 5)
head(mtcars[, c("car_name", "car_name_substr")])
```

```
##                           car_name car_name_substr
## Mazda RX4                 Mazda RX4           Mazda
## Mazda RX4 Wag         Mazda RX4 Wag           Mazda
## Datsun 710               Datsun 710           Datsu
## Hornet 4 Drive       Hornet 4 Drive           Horne
## Hornet Sportabout Hornet Sportabout           Horne
## Valiant                     Valiant           Valia
```

Veremos ahora una serie de funciones para manipular texto (o vectores de texto). Estas son especialmente útiles para la limpieza de columnas de datos. La primera que analizaremos es `grepl()`.

Esta sirve para buscar un patrón de caracteres en un vector. La sintaxis es `gsub(patrón, vector)`

```
grepl('Fiat', mtcars)
```

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

Obtenemos un vector booleano que por sí solo no nos es muy útil. Sin embargo, este se puede escribir dentro de los corchetes de indexación de un arreglo para obtener un resultado más útil.

```
 mtcars[grepl('Fiat', mtcars), ]
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb like
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3    0
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4    0
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4    0
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1    1
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8    0
##                                car_name      car_name_upper      car_name_lower
## Merc 450SL                   Merc 450SL          MERC 450SL          merc 450sl
## Lincoln Continental Lincoln Continental LINCOLN CONTINENTAL lincoln continental
## Chrysler Imperial     Chrysler Imperial   CHRYSLER IMPERIAL   chrysler imperial
## Fiat 128                     Fiat 128            FIAT 128            fiat 128
## Maserati Bora            Maserati Bora       MASERATI BORA       maserati bora
##                       car_name_replaced     car_name_hyphen car_name_substr
## Merc 450SL                   Merc 450SL          Merc-450SL           Merc
## Lincoln Continental Lincoln Continental Lincoln-Continental           Linco
## Chrysler Imperial     Chrysler Imperial   Chrysler-Imperial           Chrys
## Fiat 128                     Fiat 128            Fiat-128           Fiat
## Maserati Bora            Maserati Bora       Maserati-Bora           Maser
```

Ahora vemos que la función nos es útil para buscar datos específicos dentro de una cadena en un arreglo, no solo el dato completo de la columna (Esto se lograría con arreglo[dato == buscado]).

Vectorización de funciones

Para resolver el problema anterior primero tenemos que definir una función que haga lo que queremos. Ya analizamos el problema en el paso anterior entonces lo podemos aplicar de manera casi idéntica. Con la única excepción de que ahora lo pensamos como si la función recibiera un solo elemento a la vez.

```
extrae_apellido <- function(nombre_completo){
  strsplit(nombre_completo, " ")[[1]][2]
}
```

Verificamos que funcione como queremos:

```
extrae_apellido("Juan Pérez")
```

```
## [1] "Pérez"
```

Ahora utilizaremos la familia de funciones `apply()` para aplicar esta función a todos los elementos de un vector. Primero analizaremos `sapply()` y `lapply()`. La sintaxis para estas es: `apply(vector, función)`

```r
nombre_apellidos <- c("Alexis Zúñiga", "Juan Pérez", "José Hernández")

sapply(nombre_apellidos, extrae_apellido) #La función se escribe sin paréntesis
```

```
##  Alexis Zúñiga     Juan Pérez José Hernández
##       "Zúñiga"        "Pérez"    "Hernández"
```

```r
lapply(nombre_apellidos, extrae_apellido)
```

```
## [[1]]
## [1] "Zúñiga"
##
## [[2]]
## [1] "Pérez"
##
## [[3]]
## [1] "Hernández"
```

La función `sapply()` nos regresa un vector con los resultados que además tiene nombres y estos son los elementos del vector original. Por otro lado, `lapply()` regresa una lista donde cada resultado está en su propio vector.

Declaremos ahora una función más compleja y por lo tanto más útil que la que tenemos actualmente. Esta nos va a permitir indicarle qué caracter usar para el separador y también qué parte del nombre completo queremos extraer. Nótese que vamos a establecer valores predeterminados para estos parámetros.

```r
extrae_de_nombre <- function(nombre_completo, separador = " ", posicion = 2){
  strsplit(nombre_completo, separador)[[1]][posicion]
}
# Usando los valores default
extrae_de_nombre("Elba Laso")
```

```
## [1] "Laso"
```

```r
# Si los especificamos
extrae_de_nombre("Pepe Roni", " ", 2)
```

```
## [1] "Roni"
```

```r
# Extraer el nombre
extrae_de_nombre("Alan Brito ", posicion =1)
```

```
## [1] "Alan"
```

```r
# Con diferente separador
extrae_de_nombre("Aitor-Tilla", separador = "-")
```

```
## [1] "Tilla"
```

Para usar parámetros con las funciones `lapply()` o `sapply()` se usa la siguiente sintaxis: `apply(vector, función, parámetro = valor)`.

```
sapply(nombre_apellidos, extrae_de_nombre, posicion = 1)
```

```
##  Alexis Zúñiga     Juan Pérez José Hernández
##       "Alexis"         "Juan"         "José"
```

Sin embargo estas funciones solo reciben un parámetro por función. En este caso funciona porque nuestra función tiene valores default para los argumentos que no especificamos. Sin embargho, si queremos declarar todos individualmente, es necesario utilizar la función `mapply()`. Funciona igual que las demás funciones de la familia apply pero en este caso recibe argumentos infinitos y por lo tanto cambia la sintaxis: `mapply(función, arg1,arg2,... ,vector)`

```
nombres2 <- c("Ernesto-J-Barrios", "Alexis-X-Zúñiga", "Ana-L-Melo")
mapply(extrae_de_nombre, separador = "-", posicion = 3, nombres2)
```

```
##         -      <NA>      <NA>
## "Barrios"  "Zúñiga"    "Melo"
```