

Rstudio – Ejercicio 1

1. Previsualizar el contenido con la función head().

```
RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function Addins
Console
> mtcars
```

	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

Head() te saca los primeros 5 registros del dataframe

```
> 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

2. Mirar el número de filas y columnas con nrow() y ncol().

```
> nrow(mtcars)
[1] 32
> ncol(mtcars)
[1] 11
> |
```

3. Crear un nuevo data frame con los modelos de coche que consumen menos de 15 millas/galon.

```
> Lowcosum <- data.frame(mtcars[mtcars$mpg < 15, ])  
> view(Lowcosum)
```

Explicación del código:

lowcosum esto es como vamos a llamar al dataframe

<- esto es en que consistirá el data frame

dataa.frame () esto es que declaramos la creación de un data frame

mtcars [] esto que utilizamos información del data frame mtcars

mtcars\$mpg la columna que se llama mpg

< 15 que sea menor de 15

, incluyendo todos los demas datos

Así lowcosum sera un data frame compuesto por los datos de mtcars donde mpg sea menor que 15 y contenga el resto de los datos

Resultado:

	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

4. Ordenar el data frame anterior por disp.

```
> Lowcosum[order(Lowcosum[, "disp"]), ]  
      mpg  cyl  disp  hp drat    wt  qsec vs  am gear carb  
Camaro Z28    13.3   8   350 245 3.73 3.840 15.41 0  0   3    4  
Duster 360    14.3   8   360 245 3.21 3.570 15.84 0  0   3    4  
Chrysler Imperial 14.7   8   440 230 3.23 5.345 17.42 0  0   3    4  
Lincoln Continental 10.4   8   460 215 3.00 5.424 17.82 0  0   3    4  
Cadillac Fleetwood 10.4   8   472 205 2.93 5.250 17.98 0  0   3    4
```

Explicación del código:

Lowcosum toma este dataframe

[order()] me lo ordenas

Lowcosum[, "disp"]) de lowcosum todas las lineas (el espacio y coma) por "disp" (entre comillas que es nombre de columna

,] cierro corchetes

5. Calcular la media de las marchas (gear) de los modelos del data frame anterior.

```
> mean(Lowcosum[, "gear"])  
[1] 3  
>
```

Explicación del código:

mean() haz la media

Lowcosum haz la media

[, "gear"]) para todas las lineas en la columna que se llama "gear"

,] cierro corchetes

6. Cambiar los nombres de las variables del data frame a var1, var2, ..., var11.

Pista: Mirar la documentación de la función paste y usarla para generar el vector ("var1", "var2", ..., "varN") donde N es el número de variables del data frame.

```
> ?paste  
--> La ayuda es criptica
```

```
> ncol(Lowcosum)  
[1] 11  
> newnames <- paste0("var", c(1:11))  
> newnames  
[1] "var1" "var2" "var3" "var4" "var5" "var6" "var7" "var8" "var9" "var10"  
[11] "var11"  
> names(Lowcosum) <- newnames  
> view(Lowcosum)  
> view(Lowcosum)  
>
```

	Var1	Var2	Var3	Var4	Var5	Var6	Var7	Var8	Var9	Var10	Var11
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

Explicación del código:

ncol(lowcosum) vemos cuantas columnas tiene nuestra lowcosum

newnames <- Generamos vector new name

paste0() esto te permite combinar elemento

"var" combina el texto VAR (por eso las comillas)

, c(1:11) con c cuando c es del 1 al 11

names() Los nombres de columnas

<- newnames asigne los que hemos creado en el vector newnames

Ejercicio con data frame iris

Con el data frame iris (viene cargado en R).

1. ¿Como está estructurado el data frame? (utilizar las funciones str() y dim()).

```
> str(iris)
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
> dim(iris)
[1] 150 5
> |
```

2. ¿De qué tipo es cada una de las variables del data frame?.

```
> str(iris)
'data.frame': 150 obs.
 $ Sepal.Length: num 5.
 $ Sepal.width : num 3.
 $ Petal.Length: num 1.
 $ Petal.width : num 0.
 $ Species : Factor
```

Son tipo numérico y factor

Tambien se puede ver:

```
> class(iris[,1])
[1] "numeric"
```

Explicación del código:

class() ver clase

iris[,1] del data frame iris , todas filas, columna 1.

3. Utilizar la función summary() para obtener un resumen de los estadísticos de las variables.

```
> summary(iris)
 Sepal.Length Sepal.width Petal.Length Petal.width Species
Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100 setosa :50
1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300 versicolor:50
Median :5.800 Median :3.000 Median :4.350 Median :1.300 virginica :50
Mean :5.843 Mean :3.057 Mean :3.758 Mean :1.199
3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800
Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500
>
```

4. Comprobar con las funciones mean(), range(), que se obtienen los mismos valores.

```
> summary(iris)
  Sepal.Length      Sepa
Min.      :4.300   Min.
1st Qu.   :5.100   1st Q
Median    :5.800   Media
Mean      :5.843   Mean
3rd Qu.   :6.400   3rd Q
Max.      :7.900   Max.
> mean(iris[,1])    Pues va a ser que si.
[1] 5.843333
> range(iris[,1])
[1] 4.3 7.9
> |
```

5. Cambia los valores de las variables Sepal.Length Sepal.Width de las 5 primeras observaciones por NA.

```
> noaplica <- NA
> noaplica -> iris[1:5,1:4]
> iris[1:5,1:5]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           NA           NA           NA           NA   setosa
2           NA           NA           NA           NA   setosa
3           NA           NA           NA           NA   setosa
4           NA           NA           NA           NA   setosa
5           NA           NA           NA           NA   setosa
```

Explicación del código:

noaplica <- NA Aquí decimos que no aplica es igual a NA

naplica -> iris esto indica que llevamos lo que hay en noaplica al dataframe iris.

[1:5, 1:4] concretamente a las 4 primeras filas y las 4 primeras columnas

6. ¿Qué pasa si usamos ahora las funciones mean(), range() con las variables Sepal.Length y Sepal.Width? ¿Tiene el mismo problema la función summary()?

```
> mean(iris[,1])
[1] NA
> |
```

Pues nos aparece que no es posible

```
> summary(iris)
  Sepal.Length      Sepal.Width      Petal.Length      Petal.Width      Species
Min.      :4.300   Min.      :2.00   Min.      :1.000   Min.      :0.100   setosa      :50
1st Qu.   :5.200   1st Qu.   :2.80   1st Qu.   :1.600   1st Qu.   :0.400   versicolor:50
Median    :5.800   Median    :3.00   Median    :4.400   Median    :1.300   virginica  :50
Mean      :5.877   Mean      :3.05   Mean      :3.839   Mean      :1.234
3rd Qu.   :6.400   3rd Qu.   :3.30   3rd Qu.   :5.100   3rd Qu.   :1.800
Max.      :7.900   Max.      :4.40   Max.      :6.900   Max.      :2.500
NA's      :5       NA's      :5       NA's      :5       NA's      :5
> |
```

No tiene el mismo problema y abajo aparecen los NA que hay

7. Ver la documentación de mean(), range(), etc. ¿Qué parámetro habría que cambiar para arreglar el problema anterior?.

```
> ?mean
> mean(iris[,1], na.rm=TRUE)
[1] 5.877241
> |
```

Explicación del código:

mean() <- Media aritmetica de
iris[,1] -> de iris, todas las filas, columna 1
, na.rm=True sin tener en cuenta los NA

8. Visto lo anterior, ¿por qué es importante codificar los missing values como NA y no como 0, por ejemplo?

En R cuando falta algun valor, pues pone NA. Y muchas de las funciones estan preparadas para ello.

9. Eliminar los valores NA usando na.omit().

```
> na.omit(iris)
  Sepal.Length Sepal.width Petal.Length Petal.width  Species
6           5.4          3.9          1.7          0.4    setosa
7           4.6          3.4          1.4          0.3    setosa
8           5.0          3.4          1.5          0.2    setosa
9           4.4          2.9          1.4          0.2    setosa
10          4.9          3.1          1.5          0.1    setosa
11          5.4          3.7          1.5          0.2    setosa
```

Fijarse que empieza en la linea 6 y es que esto no borra las 5 filas en la base de datos:

```
> iris[1:5,]
  Sepal.Length Sepal.width Petal.Length Petal.width Species
1           NA          NA          NA          NA    setosa
2           NA          NA          NA          NA    setosa
3           NA          NA          NA          NA    setosa
4           NA          NA          NA          NA    setosa
5           NA          NA          NA          NA    setosa
```

10. Calcular la media de la variable Petal.Length para cada uno de las distintas especies (Species). Pista: usar la función tapply().

```
> tapply(iris$Petal.Length, iris$Species, mean, na.rm = TRUE)
setosa versicolor virginica 
1.468889  4.260000  5.552000
```

Explicación del código:

tapply() <- calcular función por rangos
iris\$Petal.length es para el data frame iris \$ nombre es decir para petal.length
iris\$Species es para el data frame iris para la columna con nombre Species
mean hazme la media agrupado por la selección anterior
na.rm=True Tambien se puede poner solo T de true. Esto para que no tenga en cuenta los NA