CC216 - FUNDAMENTOS DE DATA SCIENCE

HOJA 6 – PREPROCESAMIENTO – LIMPIEZA DE DATOS CON R

TEMA:

En esta clase, veremos en la práctica:

- Como se identifican los 'ruidos' en los datos al explorar un conjunto de datos.
- Como aplicar ciertas técnicas de limpieza de datos para dejar preparados los datos para el análisis.

OBJETIVO PRINCIPAL

Explorar un conjunto de datos, para identificar el 'ruido' (valores inconsistentes o posibles errores) y solucionarlo a partir de la aplicación de tareas/técnicas de limpieza de datos utilizando R/RStudio.

COMPETENCIAS

- Aprender a identificar los principales errores o inconsistencias en los datos.
- Aprender a limpiar los datos que presenten inconsistencias y generar un nuevo conjunto de datos limpio.

ACTIVIDADES

1. Eliminar datos sin valor
2. Limpieza selectiva de datos sin valor
    - **Caso#1:** Limpiar datos NA de sólo una variable de un dataframe
    - **Caso#2:** Limpiar datos NA de todo un dataframe
    - **Caso#3:** Limpiar los valores 'cero' de una variable de un dataframe
    - **Caso#4:** Limpiar los valores NA con valores/métricas estadísticas y/o matemáticas
3. Reemplazo de datos NA con la media o extracción aleatoria de valores
    - **Caso#1:** Reemplazo de NA con la media de la población
    - **Caso#2:** Reemplazo de datos NA con un valor aleatorio simple
4. Evitar duplicación de observaciones

APLICANDO LIMPIEZA DE DATOS EN R/RSTUDIO

Pasos iniciales

1. Configuramos nuestro directorio de trabajo en R. Por ejemplo:

```
> setwd("E:/Patricia/developer/r-course/scripts")
```

2. Abrimos el dataset contenido en el CSV missing-data.csv. En este archivo hay datos que faltan en algunos atributos. Para resolver esto, rellenaremos dichos valores faltantes con el valor 'NA'

	A	B	C	D
1	Income	Phone_type	Car_type	
2	89800	Android	Luxury	
3	47500	Android	Non-Luxury	
4	45000	iPhone	Luxury	
5	44700		Luxury	
6	59500	iPhone	Luxury	
7		Android	Non-Luxury	
8	63300	iPhone	Non-Luxury	
9	52900	Android	Luxury	
10	78200	Android	Luxury	
11	145100	iPhone	Luxury	
12	88600	iPhone	Non-Luxury	
13	65600	iPhone	Luxury	
14		Android	Non-Luxury	
15	94600	Android	Luxury	
16	59400	iPhone	Luxury	
17	47300	iPhone	Non-Luxury	
18	72100		Luxury	
19	0	iPhone	Non-Luxury	
20	0	Android	Luxury	
21	83000	iPhone	Luxury	
22	64100	Android	Non-Luxury	
23	42100	iPhone	Non-Luxury	
24	0	iPhone	Luxury	
25	91500	iPhone	Non-Luxury	
26	51200	Android	Luxury	
27	13800	iPhone	Non-Luxury	
28	47500	iPhone	Non-Luxury	
29				

1. Eliminar datos sin valor

Creamos un nuevo script R y lo grabamos en nuestra carpeta de scripts con el nombre:

01-missing-data.R

En este script:

- Leemos el archivo csv con el parámetro na.string = "" para que durante la carga de los datos se rellene con NA los valores vacíos o faltantes.

```
> data <- read.csv("../data/tema01/missing-data.csv",  na.strings = "" )
```

El resultado será:

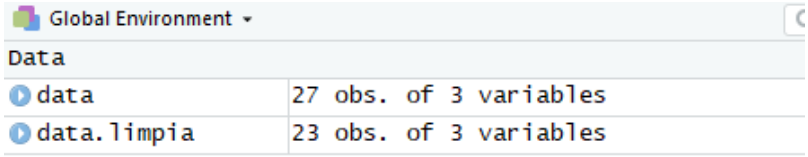
	Income	Phone_type	Car_type
1	89800	Android	Luxury
2	47500	Android	Non-Luxury
3	45000	iPhone	Luxury
4	44700	NA	Luxury
5	59500	iPhone	Luxury
6	NA	Android	Non-Luxury
7	63300	iPhone	Non-Luxury
8	52900	Android	Luxury
9	78200	Android	Luxury
10	145100	iPhone	Luxury
11	88600	iPhone	Non-Luxury
12	65600	iPhone	Luxury
13	NA	Android	Non-Luxury
14	94600	Android	Luxury
15	59400	iPhone	Luxury
16	47300	iPhone	Non-Luxury
17	72100	NA	Luxury
18	0	iPhone	Non-Luxury

Showing 1 to 19 of 27 entries

- Ahora debemos limpiar este dataframe con la instrucción **na.omit()**. Esta instrucción eliminará las observaciones (filas) que contengan en algún atributo de tipo string un NA.

```
> data.limpia <- na.omit(data)
```

Se verifica que ahora existen 23 observaciones, en lugar de las 27 observaciones originales:




```
> View(data.limpia)
```

	Income	Phone_type	Car_type
1	89800	Android	Luxury
2	47500	Android	Non-Luxury
3	45000	iPhone	Luxury
5	59500	iPhone	Luxury
7	63300	iPhone	Non-Luxury
8	52900	Android	Luxury
9	78200	Android	Luxury
10	145100	iPhone	Luxury
11	88600	iPhone	Non-Luxury
12	65600	iPhone	Luxury
14	94600	Android	Luxury
15	59400	iPhone	Luxury
16	47300	iPhone	Non-Luxury
18	0	iPhone	Non-Luxury
19	0	Android	Luxury
20	83000	iPhone	Luxury
21	64100	Android	Non-Luxury
22	42100	iPhone	Non-Luxury

Showing 1 to 19 of 23 entries

La función **na.omit()** utiliza internamente otra función llamada **is.na()**, que nos permite conocer si algún argumento que le pasemos es de tipo NA (Not Available o no disponible en castellano). La función **is.na()** cuando se aplica a un valor simple nos devuelve un valor True o False y cuando se aplica a una colección de valores (toda una columna, por ejemplo), nos devuelve un vector booleano.

Probemos con el dataframe original data:

- Consultemos si es NA el valor de la cuarta fila y segunda columna (atributo):

```
> is.na(data[4,2])
[1] TRUE
```

- ¿Es NA el valor de la cuarta fila y primera columna?

```
> is.na(data[4,1])
[1] FALSE
```

- Veamos, de toda una columna, por ejemplo, del atributo Income del dataframe data que valores contienen un NA.

```
> is.na(data$Income)
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FAL
SE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
[26] FALSE FALSE
```

Observamos que nos devuelve un vector booleano con dos coincidencias para valores NA.

Entonces, la función is.na() puede aplicarse a valores individuales como a colecciones de datos (toda una columna)

2. Limpieza selectiva de datos sin valor

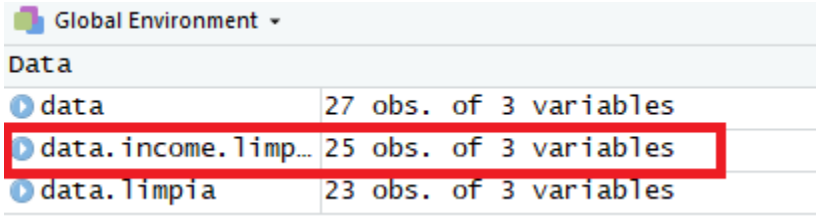
Caso#1: Limpiar datos NA de sólo una variable de un dataframe

En el dataframe original, llamado data, tenemos valores NA tanto en la variabe **Income** como en **Phone_type**. Si, solo queremos eliminar los NA de la variable Income haremos lo siguiente:

```
> data.income.limpio <- data[!is.na(data$Income),]
```

Estamos usando la negación de la función `!is.na` que significa: si no son NA los valores de Income, entonces, considera todas las filas "no NA" de todas las variables (porque después de la "," no se ha especificado que lo haga de alguna variable o columna en particular).

Como resultado, tenemos un nuevo dataframe llamado data.income.limpio que no considera NA en la columna Income. Este dataframe tiene 25 observaciones.

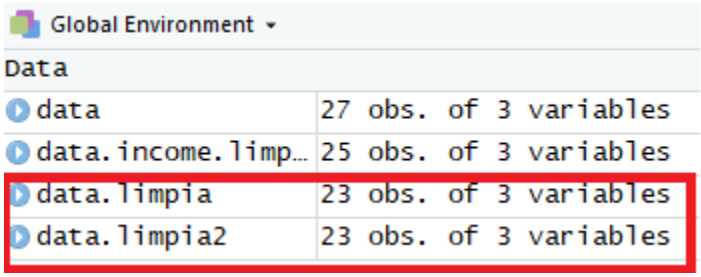


Caso#2: Limpiar datos NA de todo el dataframe

En lugar de utilizar la función **na.omit()**, podemos utilizar la funcion **complete.cases()** que nos retorna un vector booleano con los valores True o False si encuentra una valor NA.

```
> complete.cases(data)
 [1]  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TR
UE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE
[26]  TRUE  TRUE
```

```
> data.limpia2 <- data[complete.cases(data),]
```



Nos damos cuenta que el dataframe obtenido con na.omit() lo podemos generar también con la función complete.cases()

Caso#3: Limpiar los valores 'cero' de una variable de un dataframe

Es posible que el valor Income = 0 sea porque error, no se ingresó dicho valor o simplemente la persona no quiso dar ese dato. De cualquier forma, podemos completas NA para casos específicos.

- Convertir los ceros de Income en NA

Verificamos cuantas filas tiene Income con valor cero.

```
> data$Income[data$Income == 0]
[1] NA NA  0  0  0
```

Si mantenemos los ceros en las variables Income y aplicáramos el promedio a esa variable, nos bajaría bastante el promedio, por tanto, es preferible tener ese valor como NA en lugar de cero.

```
> data$Income[data$Income == 0] <- NA
```

Luego de asignarle NA a los valores cero de la variable Income verificamos sus valores:

```
> data$Income[data$Income == 0]
[1] NA NA NA NA NA
```

Ahora ya no hay valores en cero para la variable Income

	Income	Phone_type	Car_type
1	89800	Android	Luxury
2	47500	Android	Non-Luxury
3	45000	iPhone	Luxury
4	44700	NA	Luxury
5	59500	iPhone	Luxury
6	NA	Android	Non-Luxury
7	63300	iPhone	Non-Luxury
8	52900	Android	Luxury
9	78200	Android	Luxury
10	145100	iPhone	Luxury
11	88600	iPhone	Non-Luxury
12	65600	iPhone	Luxury
13	NA	Android	Non-Luxury
14	94600	Android	Luxury
15	59400	iPhone	Luxury
16	47300	iPhone	Non-Luxury
17	72100	NA	Luxury
18	NA	iPhone	Non-Luxury

Showing 1 to 19 of 27 entries

Caso#4: Limpiar los valores NA con valores/métricas estadísticas y/o matemáticas

Si requerimos aplicar el promedio o la desviación estándar o sumar los valores de una variable (columna entera), y si esta contiene mínimo un NA, el resultado a obtener será NA. No podemos obtener el resultado a ninguna operación matemática aplicada a alguna variable, si esta contiene NAs.

En este caso, para aplicar la función matemática o estadística, debemos ignorar los NA.

```
> mean(data$Income)
[1] NA
```

Consideramos el argumento na.rm = True para evitar tomar los valores NA en las funciones matemáticas y estadísticas.

```
> mean(data$Income, na.rm=TRUE)
[1] 65763.64
```

```
> sd(data$Income)
[1] NA
> sd(data$Income, na.rm=TRUE)
[1] 26715.87
```

O una simple suma:

```
> sum(data$Income)
```

```
[1] NA
> sum(data$Income, na.rm=TRUE)
[1] 1446800
```

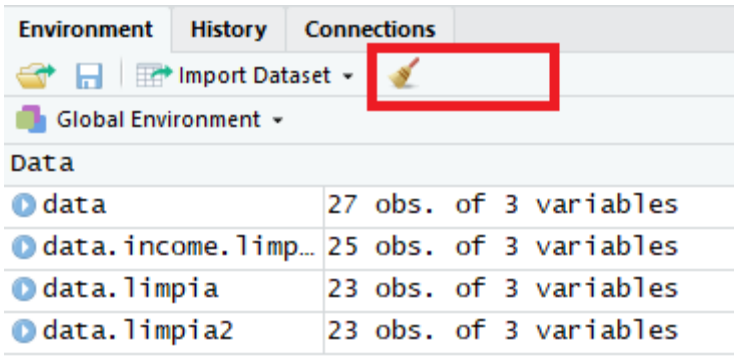
3. Reemplazo de datos NA con la media o la extracción aleatoria de valores

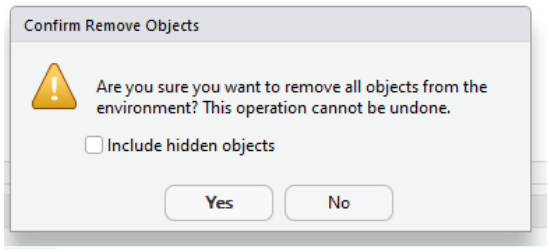
Cuando se descartan valores con NA, por lo general se estaría descartando datos que podrían ser de utilidad, entonces, haberlo hecho fácilmente con na.omit(), complete.cases() y na.rm = TRUE podría no resultarnos tan útil. En alguno u otro caso, nos interesaría reemplazar los NA con otro valor que calculemos, o que elijamos de forma aleatoria de los que ya te tenemos y que fuera más pertinente utilizar.

Caso#1: Reemplazo de datos NA con la media de la población

Una de las técnicas que se utilizan, es reemplazar los valores que no conocemos con el promedio o media (mean).

- Eliminamos los dataframes creados anteriormente (damos clic sobre la escoba y confirmamos la eliminación)



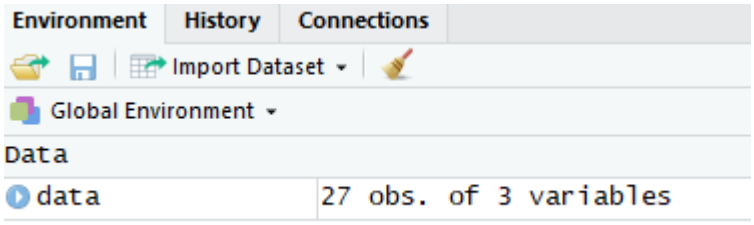


- Creamos un nuevo script R y lo grabamos en nuestra carpeta de scripts con el nombre:

01-missing-data2.R y en este script:

- Leemos nuevamente el archivo csv con el parámetro na.string = “” para que durante la carga de los datos se rellene con NA los valores vacíos o faltantes.

```
> data <- read.csv("../data/tema01/missing-data.csv",  na.strings = "" )
```



	Income	Phone_type	Car_type
1	89800	Android	Luxury
2	47500	Android	Non-Luxury
3	45000	iPhone	Luxury
4	44700	NA	Luxury
5	59500	iPhone	Luxury
6	NA	Android	Non-Luxury
7	63300	iPhone	Non-Luxury
8	52900	Android	Luxury
9	78200	Android	Luxury
10	145100	iPhone	Luxury
11	88600	iPhone	Non-Luxury
12	65600	iPhone	Luxury
13	NA	Android	Non-Luxury
14	94600	Android	Luxury
15	59400	iPhone	Luxury
16	47300	iPhone	Non-Luxury
17	72100	NA	Luxury
18	0	iPhone	Non-Luxury

Showing 1 to 19 of 27 entries

- Previamente a calcular la media, reemplazamos los valores de cero en la variable data$Income por NA.

```
data$Income[data$Income == 0] <- NA
```

	Income	Phone_type	Car_type
10	145100	iPhone	Luxury
11	88600	iPhone	Non-Luxury
12	65600	iPhone	Luxury
13	NA	Android	Non-Luxury
14	94600	Android	Luxury
15	59400	iPhone	Luxury
16	47300	iPhone	Non-Luxury
17	72100	NA	Luxury
18	0	iPhone	Non-Luxury
19	0	Android	Luxury
20	83000	iPhone	Luxury
21	64100	Android	Non-Luxury
22	42100	iPhone	Non-Luxury
23	0	iPhone	Luxury
24	91500	iPhone	Non-Luxury
25	51200	Android	Luxury
26	13800	iPhone	Non-Luxury
27	47500	iPhone	Non-Luxury

Showing 9 to 27 of 27 entries

	Income	Phone_type	Car_type
9	78200	Android	Luxury
10	145100	iPhone	Luxury
11	88600	iPhone	Non-Luxury
12	65600	iPhone	Luxury
13	NA	Android	Non-Luxury
14	94600	Android	Luxury
15	59400	iPhone	Luxury
16	47300	iPhone	Non-Luxury
17	72100	NA	Luxury
18	NA	iPhone	Non-Luxury
19	NA	Android	Luxury
20	83000	iPhone	Luxury
21	64100	Android	Non-Luxury
22	42100	iPhone	Non-Luxury
23	NA	iPhone	Luxury
24	91500	iPhone	Non-Luxury
25	51200	Android	Luxury
26	13800	iPhone	Non-Luxury

Showing 9 to 27 of 27 entries

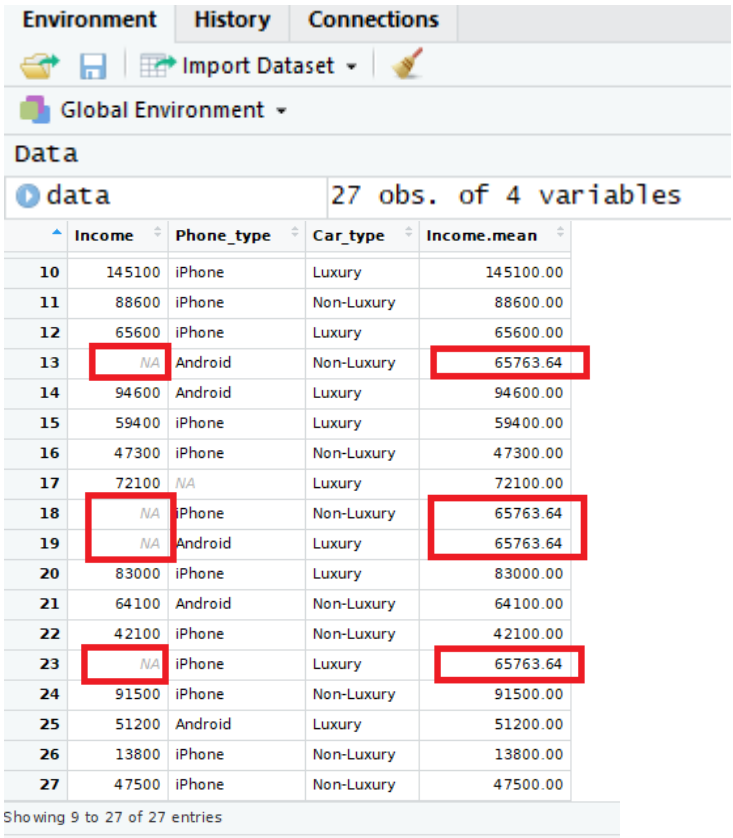
- Como no conocemos los ingresos de estas personas, cuyo ingreso figura como NA, podemos reemplazarlos por el valor promedio de la población (el promedio de ingresos de todo el dataset).

  Añadimos una cuarta columna al dataframe **data** al que llamaremos data$**Income.mean** (si no existe la crea) y utilizando la función ifelse() le asignamos a esta nueva variable el valor medio de

ingresos de la población (reemplazamos los valores con NA), dejando el valor original de ingreso a las filas que tienen un valor distinto a NA.

```
> data$Income.mean <- ifelse(is.na(data$Income), mean(data$Income, na.rm = TRUE), data$Income)
```

En las siguientes imágenes, vemos que se creó la cuarta columna y los valores que le hemos asignado:



	Income	Phone_type	Car_type	Income.mean
10	145100	iPhone	Luxury	145100.00
11	88600	iPhone	Non-Luxury	88600.00
12	65600	iPhone	Luxury	65600.00
13	NA	Android	Non-Luxury	65763.64
14	94600	Android	Luxury	94600.00
15	59400	iPhone	Luxury	59400.00
16	47300	iPhone	Non-Luxury	47300.00
17	72100	NA	Luxury	72100.00
18	NA	iPhone	Non-Luxury	65763.64
19	NA	Android	Luxury	65763.64
20	83000	iPhone	Luxury	83000.00
21	64100	Android	Non-Luxury	64100.00
22	42100	iPhone	Non-Luxury	42100.00
23	NA	iPhone	Luxury	65763.64
24	91500	iPhone	Non-Luxury	91500.00
25	51200	Android	Luxury	51200.00
26	13800	iPhone	Non-Luxury	13800.00
27	47500	iPhone	Non-Luxury	47500.00

Showing 9 to 27 of 27 entries

Caso#2: Reemplazo de datos NA con un valor aleatorio simple

- Tengamos presente que NO PODEMOS CALCULAR EL PROMEDIO DE VARIABLES CATEGORICAS, SOLO DE VARIABLES NUMERICAS. Entonces, para reemplazar los NA de variables categóricas (en la variable Phone_type, por ejemplo), podemos elegir de forma aleatoria un valor entre la población que si tiene un valor diferente a NA en Phone_type y asignar dicho valor aleatorio a quien no lo tiene.
- Creamos un nuevo script R y lo grabamos en nuestra carpeta de scripts con el nombre:

  02-replace-missing-data.R

- Creamos una función que llamaremos **rand.valor()** para obtener un valor aleatorio de la(s) variable(s) del dataframe que queremos examinar (data en nuestro ejemplo). Esta técnica la llamamos **Muestra Aleatoria Simple (MAS).**

```
rand.valor <- function(x){
+   faltantes <- is.na(x)
+   tot.faltantes <- sum(faltantes)
+   x.obs <- x[!faltantes]
```

```
+   valorado <- x
+   valorado[faltantes] <- sample(x.obs, tot.faltantes, replace = TRUE)
+   return (valorado)
+ }
```

Esta función, hace lo siguiente:

a) Recibe en **x** un vector de valores
b) Coloca en el vector booleano **faltantes** TRUE o FALSE, dependiendo si cada valor de x es NA o no.
c) Suma en **tot.faltantes** el total de valores que resultaron TRUE (es decir, son NA) del vector faltantes. El valor de tot.faltantes es un numero entero.
d) El vector **x.obs** contendrá los valores de las observaciones o filas que tienen un valor diferente a NA.
e) El vector **valorado** inicialmente tiene los valores iniciales de x
f) Seleccionamos solo los elementos **faltantes** en el vector **valorado** y le asignamos mediante la función sample() un valor aleatorio obtenido del vector **x.obs** (que contiene los valores diferentes a NA), indicando la cantidad de valores aleatorios requeridos en **tot.faltantes** y señalamos que reemplace con dichos valores aleatorios los valores faltantes en el vector **valorado**.
g) Retornamos el vector **valorado**, el que contiene todos sus valores distintos a NA, porque los valores **faltantes** fueron reemplazados por un valor aleatorio tomado de vector original **x**.

- Creamos una nueva función que llamaremos **random.df**, que recibirá como parámetros un dataframe y los nombres de sus columnas que queremos pre-procesar.

```
> random.df <- function(df, cols){
+   nombres <- names(df)
+   for (col in cols)  {
+     nombre <- paste(nombres[col], "valorado", sep = ".")
+     df[nombre] <- rand.valor(df[,col])
+   }
+   df
+ }
```

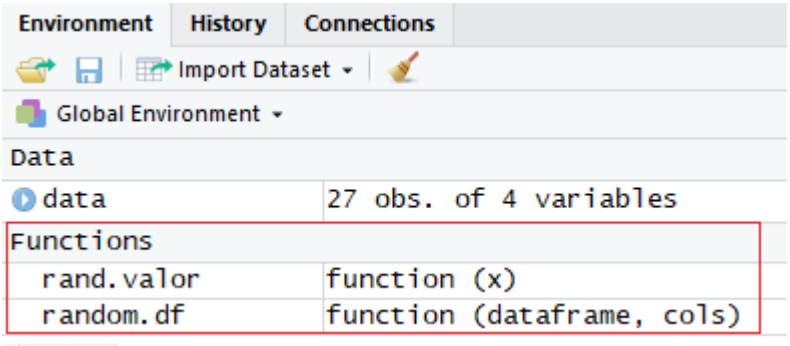
Esta función, hace lo siguiente:

a) Recibe en el parámetro **df** el dataframe que queremos evaluar y un vector de nombres columas de dicho dataframe.
b) En el vector llamado **nombre** obtenemos todos los nombres de las columnas del dataframe recibido en **df**.
c) Con la sentencia **for** recorremos desde la primera a la última columna del dataframe df y concatenamos en la variable nombre, un nuevo nombre de columna que resulta de la concatenación del nombre original de la columna y la palabra 'valorado'. Esto lo logramos con la función **paste(),** y la separación entre estas dos palabras será un punto ".".
d) Ejecutamos la función creada previamente rand.valor(). Creamos una nueva columna en el dataframe **df,** poniéndole como nombre de columna el valor obtenido en la variable **nombre** y a esta nueva columna le asignamos los valores random que nos devuelve la

función **rand.valor()** al que le pasamos como parámetro todas las filas del dataframe **df** y la columna **col** respectiva dentro del bucle.

e) Finalmente, devolvemos el nuevo dataframe **df** que contiene las nuevas columnas pre-procesadas con los valores random calculados.

- Ejecutamos ambas funciones y las vemos creadas ahora en nuestro ambiente global:



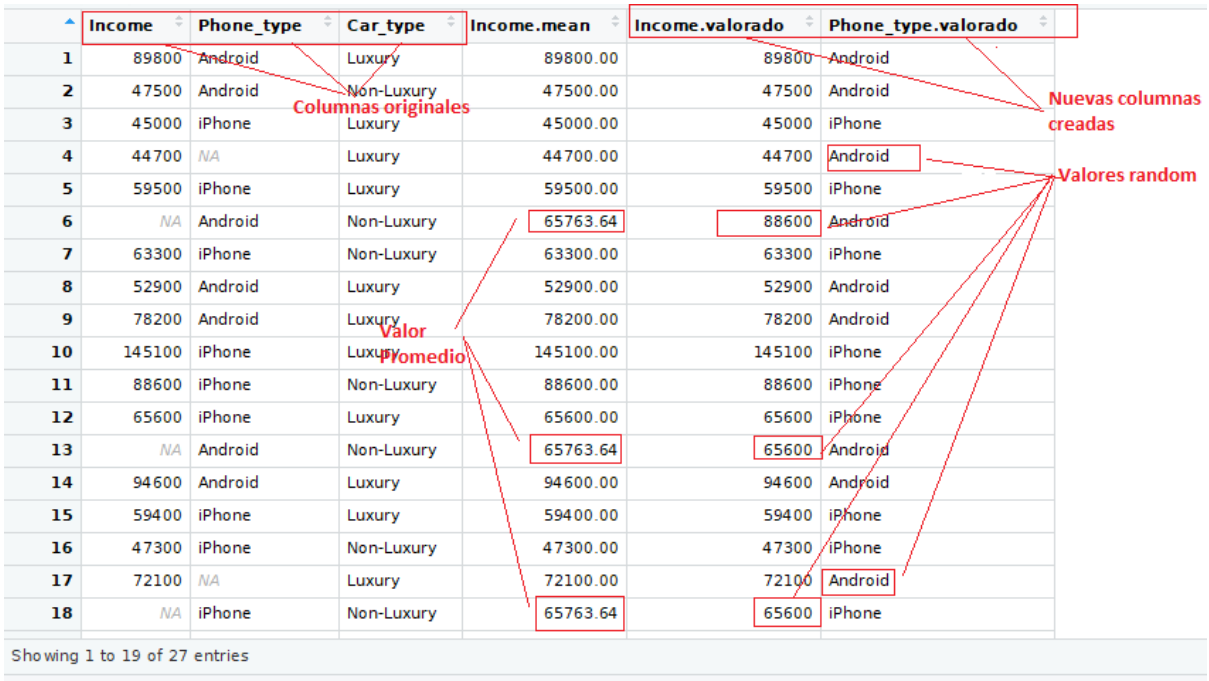
- Recordemos que en el dataframe **data** completamos previamente los valores faltantes y los de valor cero en Income y en Phone_Type con NA. Ahora, procedemos a pre-procesar los valores NA de dichas columnas (columnas 1 y 2 respectivamente) asignándoles de forma aleatoria un valor de los que no son NA.

  Dataframe **data** actual:

	Income	Phone_type	Car_type	Income.mean
1	89800	Android	Luxury	89800.00
2	47500	Android	Non-Luxury	47500.00
3	45000	iPhone	Luxury	45000.00
4	44700	NA	Luxury	44700.00
5	59500	iPhone	Luxury	59500.00
6	NA	Android	Non-Luxury	65763.64
7	63300	iPhone	Non-Luxury	63300.00
8	52900	Android	Luxury	52900.00
9	78200	Android	Luxury	78200.00
10	145100	iPhone	Luxury	145100.00
11	88600	iPhone	Non-Luxury	88600.00
12	65600	iPhone	Luxury	65600.00
13	NA	Android	Non-Luxury	65763.64
14	94600	Android	Luxury	94600.00
15	59400	iPhone	Luxury	59400.00
16	47300	iPhone	Non-Luxury	47300.00
17	72100	NA	Luxury	72100.00
18	NA	iPhone	Non-Luxury	65763.64

- Creamos un nuevo dataframe que llamaremos **data.limpio** y allí crearemos las nuevas columnas con los datos limpios proveniente de las columnas 1 y 2 Income y Phone_type, a las hemos llamado Income.valorado y Phone_type.valorado respectivamente, como se muestra a continuación:

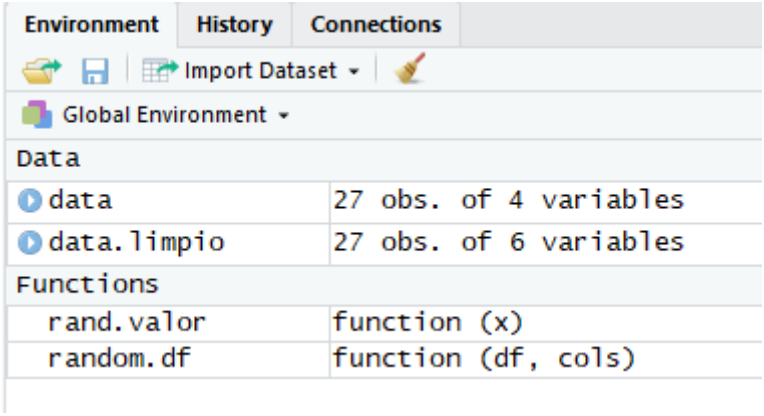
```
> data.limpio <- random.df(data, c(1,2))
> View(data.limpio)
```



	Income	Phone_type	Car_type	Income.mean	Income.valorado	Phone_type.valorado
1	89800	Android	Luxury	89800.00	89800	Android
2	47500	Android	Non-Luxury	47500.00	47500	Android
3	45000	iPhone	Luxury	45000.00	45000	iPhone
4	44700	NA	Luxury	44700.00	44700	Android
5	59500	iPhone	Luxury	59500.00	59500	iPhone
6	NA	Android	Non-Luxury	65763.64	88600	Android
7	63300	iPhone	Non-Luxury	63300.00	63300	iPhone
8	52900	Android	Luxury	52900.00	52900	Android
9	78200	Android	Luxury	78200.00	78200	Android
10	145100	iPhone	Luxury	145100.00	145100	iPhone
11	88600	iPhone	Non-Luxury	88600.00	88600	iPhone
12	65600	iPhone	Luxury	65600.00	65600	iPhone
13	NA	Android	Non-Luxury	65763.64	65600	Android
14	94600	Android	Luxury	94600.00	94600	Android
15	59400	iPhone	Luxury	59400.00	59400	iPhone
16	47300	iPhone	Non-Luxury	47300.00	47300	iPhone
17	72100	NA	Luxury	72100.00	72100	Android
18	NA	iPhone	Non-Luxury	65763.64	65600	iPhone

Showing 1 to 19 of 27 entries

- Y los datos y funciones creadas se visualizan en nuestro ambiente global de trabajo:



- No olvidar guardar el script 02-replace-missing-data.R

4. Evitar duplicación de observaciones

En este ejemplo, utilizaremos la función unique() para filtrar las observaciones únicas en un dataset y la función duplicates() para identificas que observaciones son las duplicadas.

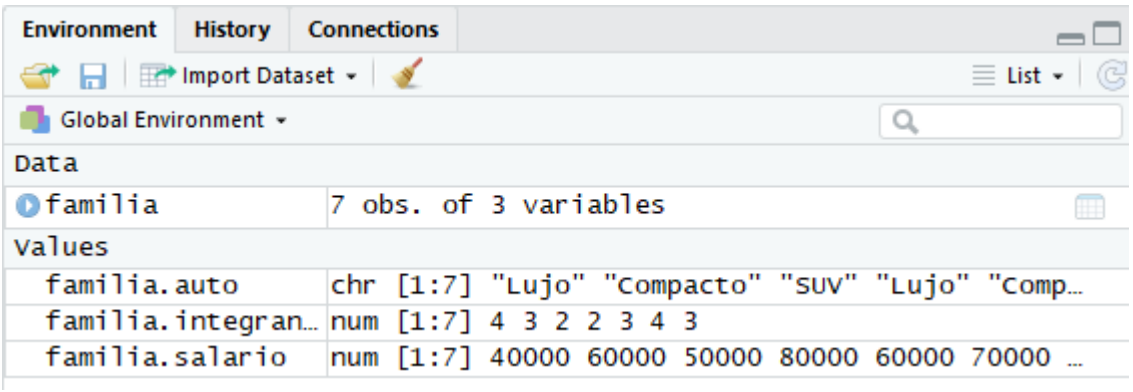
- Creamos un script en R llamado 03-remove-duplicates.R
- A continuación, no cargaremos los datos, sino que crearemos los propios. Crearemos nuestro propio dataframe a partir de datos de una familia (sueldo, numero de integrantes y tipo de auto).

  Ejecutamos las siguientes instrucciones:

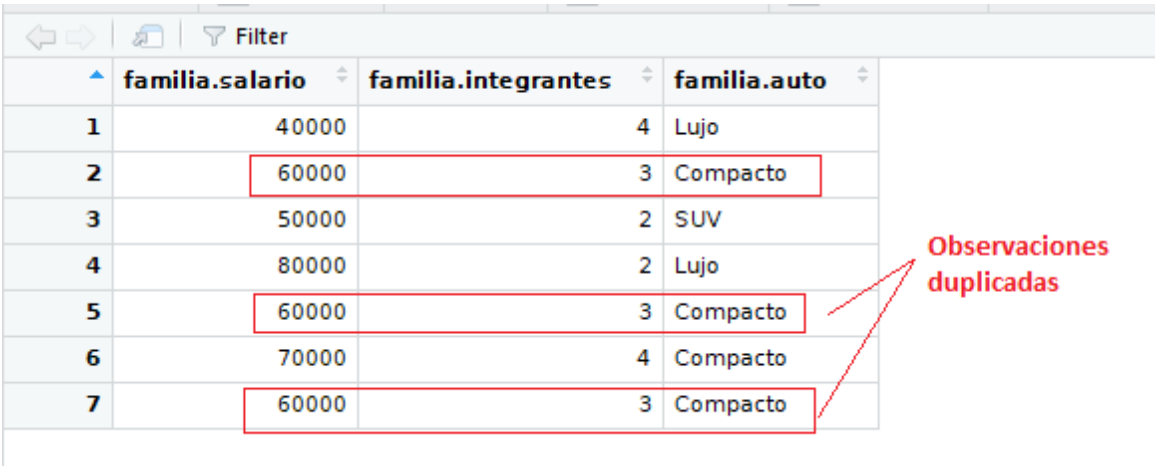
```
> familia.salario <- c(40000,60000,50000, 80000, 60000, 70000, 60000)
> familia.integrantes <- c(4,3,2,2,3,4,3)
```

```
> familia.auto <- c("Lujo", "Compacto", "SUV", "Lujo", "Compacto", "Compac
to", "Compacto")
> familia <- data.frame(familia.salario, familia.integrantes, familia.auto
)
```

- Al ejecutar estas instrucciones hemos creado el dataframe familia que consta de 7 observaciones en 3 variables.




```
View(familia)
```



- Observamos que existen filas (observaciones) duplicadas. Entonces, si queremos consultar que observaciones son las duplicadas, utilizaremos la función **duplicated()**.

```
> duplicated(familia)
[1] FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE
```

Y obtenemos un vector booleano que solo considera como TRUE dos coincidencias (la fila 5 y 7), lógicamente porque la fila 2 es la que considera la original o no duplicada.

- Consultamos que observaciones son las duplicadas, al recuperar del dataframe familia, las filas duplicadas para todas sus variables, con la siguiente instrucción:

```
> familia[duplicated(familia),]
```

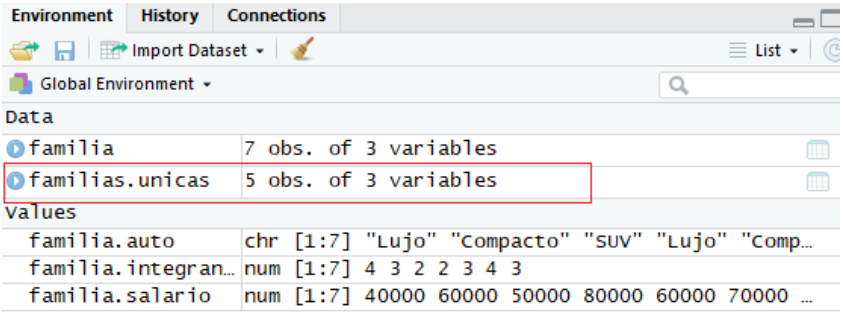
```
  familia.salario familia.integrantes familia.auto
5           60000                   3     Compacto
7           60000                   3     Compacto
```

- Si los que deseamos es recuperar las observaciones “únicas” en el dataframe, utilizaremos la función unique(), que nos devuelve un dataframe sin duplicados.

```
> unique(familia)
  familia.salario familia.integrantes familia.auto
1           40000                   4         Lujo
2           60000                   3     Compacto
3           50000                   2          SUV
4           80000                   2         Lujo
6           70000                   4     Compacto
```

- Finalmente, este dataframe con observaciones únicas, lo podemos almacenar en un nuevo dataframe llamado familias.unicas

```
> familias.unicas <- unique(familia)
```




```
> View(familias.unicas)
```

	familia.salario	familia.integrantes	familia.auto
1	40000	4	Lujo
2	60000	3	Compacto
3	50000	2	SUV
4	80000	2	Lujo
6	70000	4	Compacto

- No existen observaciones duplicadas ahora en el dataframe familias.unicas.