



BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

*Limpieza y Preprocesamiento de Datos sobre la
Donación de Órganos en México*

Introducción a la Ciencia de Datos

Autor: Jonathan Araiza Guzmán

Profesor: Jaime Alejandro Romero Sierra

Repositorio:

10/10/2025

DESCRIPCIÓN DE LA BASE DE DATOS

Se muestra la fuente de la base de datos, la descripción general del contenido y el significado de cada columna en una tabla para facilitar su comprensión.

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** 01_Análisis_Exploratorio.ipynb
- Path:** Proyecto_Data_Science_Sem01 > Proyecto (Donación de órganos) > 01_Análisis_Exploratorio.ipynb
- Cell 36 of 58:** Python 3.13.7
- Content:**
 - Descripción de inicial de la base de datos:**
 - Fuente de la base de datos: <https://www.kaggle.com/datasets/joshdoit/master>
 - Descripción general del contenido: La presente base de datos fue publicada por la CENATRA, recopilando datos abiertos del gobierno de México. Esta contiene 52501 diferentes entradas (filas) y 24 diferentes variables (columnas) de diferentes donadores en México, las cuales registran el proceso de la donación de órganos. La base abarca datos desde el año 2007 al primer semestre de 2019.
 - Significado de cada columna:
 - Table:** A table mapping categories to columns and their descriptions.

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** 01_Análisis_Exploratorio.ipynb
- Path:** Proyecto_Data_Science_Sem01 > Proyecto (Donación de órganos) > 01_Análisis_Exploratorio.ipynb
- Cell 36 of 58:** Python 3.13.7
- Content:**
 - Descripción de inicial de la base de datos:**
 - Fuente de la base de datos: <https://www.kaggle.com/datasets/joshdoit/master>
 - Descripción general del contenido: La presente base de datos fue publicada por la CENATRA, recopilando datos abiertos del gobierno de México. Esta contiene 52501 diferentes entradas (filas) y 24 diferentes variables (columnas) de diferentes donadores en México, las cuales registran el proceso de la donación de órganos. La base abarca datos desde el año 2007 al primer semestre de 2019.
 - Significado de cada columna:
 - Table:** A table mapping categories to columns and their descriptions.

PROCESO DE LIMPIEZA (EVIDENCIAS)

Para comenzar, debemos de explorar nuestro DataSet. Explorar el DataSet nos permitirá diagnosticar diferentes problemas que no son visibles: (datos nulos, duplicados, valores atípicos, conversión de tipos de datos, etc.).

2. Exploración y diagnóstico del DataFrame

Antes de modificar los datos de nuestro DataSet debemos de conocer su estructura. Conocer su composición nos permite diagnosticar problemas que no son visibles.

```
# Debemos conocer el total de filas y columnas de nuestro DataFrame.
print("El DataFrame tiene {} filas y {} columnas ---")
print("")
# Hacemos uso de .info() para realizar un diagnóstico más profundo.
# .info() nos permite ver el número de entradas, las columnas y cuantos valores "no nulos" tienen.
# .info() también nos permite conocer el tipo de dato (Dtype) de cada columna.
print("Diagnóstico con .info() ---")
df_main.info()
print("")
```

[3] 0.0s Python

```
...  
--- El DataFrame tiene 52501 filas y 24 columnas ---  
  
--- Diagnóstico con .info() ---  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 52501 entries, 0 to 52500  
Data columns (total 24 columns):  
 #   Column           Non-Null Count  Dtype     
---   --  
 0   SEXO              50396 non-null   object    
 1   CODIGO_SEXO       49382 non-null   float64  
 2   TIPO_DONANTE      50401 non-null   object    
 3   MUERTE             50401 non-null   object    
 4   ENTIDAD_FEDERATIVA 50401 non-null   object    
 5   CODIGO_ENTIDAD_FEDERATIVA 50401 non-null   object    
 6   ESTABLECIMIENTO    50401 non-null   object    
 7   INSTITUCION        50401 non-null   object    
 8   EDAD_ANIOS         49403 non-null   float64  
 9   FECHA_PROCURACION 50401 non-null   object    
 10  RINON_IZQUIERDO    50401 non-null   object    
 11  RINON_DERECHO      50401 non-null   float64  
 12  RINON_BLOCK         50401 non-null   object    
 13  PULMON_IZQUIERDO   50401 non-null   object    
 14  PULMON_DERECHO     50401 non-null   float64  
 ...  
 23  CORAZON_TEJIDOS    49388 non-null   float64  
dtypes: float64(12), object(12)  
memory usage: 9.6+ MB
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Observamos que los tipos de cada columna no son los que deberían ser. Esto nos permite saber que debemos de hacer una limpieza en cada columna de nuestro DataFrame.

```
# Encontramos los valores nulos de cada columna con .isnull() y .sum() para contar todos los valores nulos de cada uno.
print("Total de valores nulos ---")
df_main.isnull().sum()
```

[4] 0.0s Python

```
...  
--- Total de valores nulos ---  
  
...  SEXO              2105  
  CODIGO_SEXO          3119  
  TIPO_DONANTE         2100  
  MUERTE              2100  
  ENTIDAD_FEDERATIVA   2100  
  CODIGO_ENTIDAD_FEDERATIVA 2100  
  ESTABLECIMIENTO     2100  
  INSTITUCION          2100  
  EDAD_ANIOS           3098  
  FECHA_PROCURACION   2100  
  RINON_IZQUIERDO      2100  
  RINON_DERECHO         2100  
  RINON_BLOCK           2100  
  PULMON_IZQUIERDO     2100  
  PULMON_DERECHO        2100  
  CORAZON               2100  
  HIGADO               2100  
  PANCREAS              2100  
  INTESTINO              2100  
  CORNEA_IZQUIERDA     2100  
  CORNEA_DERECHA        2100  
  PIEL                  2100  
  HUELOS                2100  
  CORAZON_TEJIDOS        3113  
dtype: int64
```

Ahora, debemos de identificar el total de valores nulos de cada columna.

```
[4] # Encontramos los valores nulos de cada columna con .isnull() y .sum() para contar todos los valores nulos de cada uno.
print("Total de valores nulos ---")
df_main.isnull().sum()

[4]: 0.0s
Python
...
--- Total de valores nulos ---

...
SEXO 2105
CODIGO_SEXO 3119
TIPO_DONANTE 2100
MUERTE 2100
ENTIDAD_FEDERATIVA 2100
CODIGO_ENTIDAD_FEDERATIVA 2100
ESTABLECIMIENTO 2100
INSTITUCION 2100
EDAD_ANIOS 3098
FECHA_PROCURACION 2100
RINON_IZQUIERDO 2100
RINON_DERECHO 2100
RINON_BLOCK 2100
PULMONIZQUIERDO 2100
PULMONDERECHO 2100
CORAZON 2100
HIGADO 2100
PANCREAS 2100
INTESTINO 2100
CORNIAIZQUIERDA 2100
CORNIA_DERECHA 2100
PIEL 2100
HUESOS 2100
CORAZON_TEJIDOS 3113
dtype: int64
```

Para finalizar la exploración y diagnóstico del DataSet, debemos de investigar todos los valores de cada columna y posteriormente, transformamos los valores en base de nuestras necesidades.

Cabe mencionar que se clasificaron las columnas a partir del tipo del total de valores que contienen:

- **Columnas con pocos valores**
- **Columnas con varios valores**
- **Columnas con valores numéricos y binarios**

Iniciamos visualizando los valores de las columnas con pocos valores:

- ‘SEXO’
- ‘CODIGO_SEXO’
- ‘TIPO_DONANTE’
- ‘MUERTE’
- ‘ENTIDAD_FEDERATIVA’
- ‘CODIGO_ENTIDAD_FEDERATIVA’
- ‘INSTITUCION’

3.1 Visualización de columnas con pocos valores

A continuación, se clasificaron las columnas elegidas como columnas de pocos valores y se muestran los valores de cada una:

```
# Bloque de código para visualizar columnas con pocos valores.  
print("Los valores de 'SEXO' son: \n---> {{df_main['SEXO'].unique()}}")  
print("")  
print("Los valores de 'CODIGO_SEXO' son: \n---> {{df_main['CODIGO_SEXO'].unique()}}")  
print("")  
print("Los valores de 'TIPO_DONANTE' son: \n---> {{df_main['TIPO_DONANTE'].unique()}}")  
print("")  
print("Los valores de 'MUERTE' son: \n---> {{df_main['MUERTE'].unique()}}")  
print("")  
print("Los valores de 'ENTIDAD_FEDERATIVA' son: \n---> {{df_main['ENTIDAD_FEDERATIVA'].unique()}}")  
print("")  
print("Los valores de 'CODIGO_ENTIDAD_FEDERATIVA' son: \n---> {{df_main['CODIGO_ENTIDAD_FEDERATIVA'].unique()}}")  
print("")  
print("Los valores de 'INSTITUCION' son: \n---> {{df_main['INSTITUCION'].unique()}}")  
print("")
```

```
... Los valores de 'SEXO' son:  
---> [nan 'MASCULINO' 'FEMENINO' 'bbb']  
  
Los valores de 'CODIGO_SEXO' son:  
---> [ 2.  1. nan]  
  
Los valores de 'TIPO_DONANTE' son:  
---> ['CADÁVER' 'bbb' nan 'VIVO']  
  
Los valores de 'MUERTE' son:  
---> ['PARO CARDIORESPIRATORIO' 'MUERTE ENCEFÁLICA' nan 'bbb' 'NO APLICA']  
  
Los valores de 'ENTIDAD_FEDERATIVA' son:  
---> ['NUEVO LEÓN' 'CIUDAD DE MÉXICO' 'PUEBLA' 'QUERÉTARO' 'MÉXICO'  
'SAN LUIS POTOSÍ' 'GUANAJUATO' 'TABASCO' 'VERACRUZ' 'BAJA CALIFORNIA'  
'DURANGO' 'AGUASCALIENTES' 'SONORA' 'MICHOACÁN' 'COAHUILA' 'CHIHUAHUA'  
'SINALOA' nan 'MORELOS' 'ZACATECAS' 'COLIMA' 'BAJA CALIFORNIA SUR'  
'JALISCO' 'OAXACA' 'YUCATÁN' 'GUERRERO' 'TAMAULIPAS' 'QUINTANA ROO'  
'NAYARIT' 'TLAXCALA' 'HIDALGO' 'CHIAPAS' 'CAMPECHE']  
  
Los valores de 'CODIGO_ENTIDAD_FEDERATIVA' son:  
---> ['19' '9' '21' '22' '15' '24' '11' '27' '30' '2' 'bbb' '10' '1' '26' '16'  
'5' '8' '25' nan '17' '6' '3' '14' '20' '31' '12' '32' '28' '23' '18'  
'29' '13' '7' '4']  
  
Los valores de 'INSTITUCION' son:  
---> ['IMSS' 'SSE' 'PRIVADO' nan 'ISSSTE' 'SSA' 'bbb' 'PEMEX' 'SNTE' 'SEDENA'  
'SEMAR']
```

Comenzamos con la limpieza de esta sección de variables. Sin embargo, es necesario hacer dos cosas: realizar una copia del DataFrame principal, así nos aseguramos de no modificar algo de forma accidental y evitar problemas en el análisis posterior de la limpieza.

```
[6] # Antes de proceder a limpiar los datos, es importante hacer una copia de seguridad del DataFrame original.  
df_backup = df_main.copy()  
print("\n--- Copia de seguridad del DataFrame creada ---\n")  
[6] ✓ 0.0s  
...  
--- Copia de seguridad del DataFrame creada ---
```

Ahora, eliminamos las entradas duplicadas. Al aplicar `.drop_duplicates(inplace=True)`, eliminamos todos estos valores, y pasamos de 52501 entradas, a 51658.

```
[7] # Procedemos a limpiar los datos de esta sección.  
# Eliminamos las filas duplicadas.  
df_backup.drop_duplicates(inplace=True)  
  
# Mostramos el número de filas y columnas antes de eliminar duplicados.  
print(f"\n--- Antes de eliminar duplicados, el DataFrame tenía {df_main.shape[0]} filas y {df_main.shape[1]} columnas ---")  
  
# Verificamos que se hayan eliminado las filas duplicadas.  
print(f"\n--- Después de eliminar duplicados, el DataFrame tiene {df_backup.shape[0]} filas y {df_backup.shape[1]} columnas ---")  
[7] ✓ 0.0s  
...  
--- Antes de eliminar duplicados, el DataFrame tenía 52501 filas y 24 columnas ---  
  
--- Después de eliminar duplicados, el DataFrame tiene 51658 filas y 24 columnas ---
```

A partir del diagnóstico exploratorio y de conocer los valores de las columnas con pocos valores, obtenemos los valores que debemos de transformar.

3.2 Limpieza y transformación de columnas con pocos valores

Después de conocer todos los valores de cada columna de esta sección, procedemos a transformar los que no nos sirven. A continuación, los valores que necesitamos transformar de cada columna:

- Para 'SEXO' los valores son: 'nan' y 'bbb'
- Para 'CODIGO_SEXO' los valores son: 'nan'
- Para 'TIPO_DONANTE' los valores son: 'nan' y 'bbb'
- Para 'MUERTE' los valores son: 'nan' y 'bbb'
- Para 'ENTIDAD_FEDERATIVA' los valores son: 'nan', 'NUEVO LEÓN', 'QUERÉTARO', 'MÉXICO', 'SAN LUIS POTOSÍ', 'MICHOACÁN' y 'YUCATÁN'
- Para 'CODIGO_ENTIDAD_FEDERATIVA' los valores son: 'nan' y 'bbb'
- Para 'INSTITUCION' los valores son: 'nan' y 'bbb'

Realizamos la limpieza de cada columna.

‘SEXO’ :

```
[8] # Limpieza de la columna 'SEXO'.  
print("\n--- Valores únicos en 'SEXO' antes de la limpieza ---")  
print(df_backup['SEXO'].unique())  
  
# Limpiamos los valores nulos en la columna 'SEXO' reemplazándolos con 'DESCONOCIDO'.  
df_backup.fillna({'SEXO': 'DESCONOCIDO'}, inplace=True)  
  
# Ahora, los valores 'bbb' los reemplazamos con 'DESCONOCIDO'.  
df_backup['SEXO'].replace('bbb', 'DESCONOCIDO', inplace=True)  
  
# Verificamos los valores únicos después de la limpieza.  
print("\n--- Valores únicos en 'SEXO' después de la limpieza ---")  
print(df_backup['SEXO'].unique())  
print("")  
[8] ✓ 0.0s  
...  
--- Valores únicos en 'SEXO' antes de la limpieza ---  
[nan 'MASCULINO' 'FEMENINO' 'bbb']  
--- Valores únicos en 'SEXO' después de la limpieza ---  
['DESCONOCIDO' 'MASCULINO' 'FEMENINO']  
  
C:\Users\pikag\AppData\Local\Temp\ipykernel_16408\230935811.py:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method((col: value), inplace=True)' or 'df[col] = df[col].method(value)' instead, to perform the operation inplace on  
df_backup['SEXO'].replace('bbb', 'DESCONOCIDO', inplace=True)
```

‘CODIGO_SEXO’:

```
# Limpieza de la columna 'CODIGO_SEXO'.
print("\n--- Valores únicos en 'CODIGO_SEXO' antes de la limpieza ---")
print(df_backup['CODIGO_SEXO'].unique())

# La columna 'CODIGO_SEXO' debe contener solo valores 'string' que representen números.
# Limpiamos los valores nulos en la columna 'CODIGO_SEXO' reemplazándolos con 'DESCONOCIDO'.
df_backup.fillna({'CODIGO_SEXO': 'DESCONOCIDO'}, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'CODIGO_SEXO' después de la limpieza ---")
print(df_backup['CODIGO_SEXO'].unique())
print("")
```

[9] ✓ 0.0s Python

...

```
--- Valores únicos en 'CODIGO_SEXO' antes de la limpieza ---
[ 2.  1. nan]
```

```
--- Valores únicos en 'CODIGO_SEXO' después de la limpieza ---
[2.0 1.0 'DESCONOCIDO']
```

‘TIPO_DONANTE’:

```
# Limpieza de la columna 'TIPO_DONANTE'.
print("\n--- Valores únicos en 'TIPO_DONANTE' antes de la limpieza ---")
print(df_backup['TIPO_DONANTE'].unique())

# Limpiamos los valores nulos en la columna 'TIPO_DONANTE' reemplazándolos con 'DESCONOCIDO'.
df_backup.fillna({'TIPO_DONANTE': 'DESCONOCIDO'}, inplace=True)

# Ahora, los valores 'bbb' los reemplazamos con 'DESCONOCIDO'.
df_backup['TIPO_DONANTE'].replace('bbb', 'DESCONOCIDO', inplace=True)

# Transformamos 'CADÁVER' a 'CADÁVER' para estandarizar.
df_backup['TIPO_DONANTE'].replace('CADÁVER', 'CADÁVER', inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'TIPO_DONANTE' después de la limpieza ---")
print(df_backup['TIPO_DONANTE'].unique())
print("")
```

[10] ✓ 0.0s Python

...

```
--- Valores únicos en 'TIPO_DONANTE' antes de la limpieza ---
['CADÁVER' 'bbb' nan 'VIVO']
```

```
--- Valores únicos en 'TIPO_DONANTE' después de la limpieza ---
['CADÁVER' 'DESCONOCIDO' 'VIVO']
```

‘MUERTE’:

```
# Limpieza para la columna 'MUERTE'.
print("\n--- Valores únicos en 'MUERTE' antes de la limpieza ---")
print(df_backup['MUERTE'].unique())

# Limpiamos los valores nulos en la columna 'MUERTE'.
df_backup.fillna({'MUERTE': 'DESCONOCIDO'}, inplace=True)

# Ahora, los valores 'bbb' los reemplazamos con 'DESCONOCIDO'.
df_backup['MUERTE'].replace('bbb', 'DESCONOCIDO', inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'MUERTE' después de la limpieza ---")
print(df_backup['MUERTE'].unique())
print("")
```

[11] ✓ 0.0s Python

...

```
--- Valores únicos en 'MUERTE' antes de la limpieza ---
['PARO CARDIORESPIRATORIO' 'MUERTE ENCEFÁLICA' nan 'bbb' 'NO APLICA']
```

```
--- Valores únicos en 'MUERTE' después de la limpieza ---
['PARO CARDIORESPIRATORIO' 'MUERTE ENCEFÁLICA' 'DESCONOCIDO' 'NO APLICA']
```

```
C:\Users\pikag\AopData\local\Temp\ipykernel_16408\3318898090.py:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or 'df[col] = df[col].method(value)' instead, to perform the operation inplace on
```

```
df_backup['MUERTE'].replace('bbb', 'DESCONOCIDO', inplace=True)
```

'ENTIDAD_FEDERATIVA': Aquí se transforman aquellas palabras que usan acentos. Esto por estandarización y para evitar futuros problemas en el posterior análisis.

```
# Limpieza de la columna 'ENTIDAD_FEDERATIVA'.
print("\n--- Valores únicos en 'ENTIDAD_FEDERATIVA' antes de la limpieza ---")
print(df_backup['ENTIDAD_FEDERATIVA'].unique())

# Limpiamos los valores nulos en la columna 'ENTIDAD_FEDERATIVA'.
df_backup.fillna('ENTIDAD_FEDERATIVA': 'DESCONOCIDO', inplace=True)

# Transformamos los valores con acentos a sin acentos.
# 'NUEVO LEÓN' tiene acento en la 'O'.
df_backup['ENTIDAD_FEDERATIVA'].replace('NUEVO LEÓN', 'NUEVO LEÓN', inplace=True)

# 'QUERETARO' tiene acento en la 'E'.
df_backup['ENTIDAD_FEDERATIVA'].replace('QUERETARO', 'QUERETARO', inplace=True)

# 'MÉXICO' tiene acento en la 'E'.
df_backup['ENTIDAD_FEDERATIVA'].replace('MÉXICO', 'MEXICO', inplace=True)

# 'SAN LUIS POTOSÍ' tiene acento en la 'I'.
df_backup['ENTIDAD_FEDERATIVA'].replace('SAN LUIS POTOSÍ', 'SAN LUIS POTOSI', inplace=True)

# 'MICHOACÁN' tiene acento en la 'A'.
df_backup['ENTIDAD_FEDERATIVA'].replace('MICHOACÁN', 'MICHOACAN', inplace=True)

# 'YUCATÁN' tiene acento en la 'A'.
df_backup['ENTIDAD_FEDERATIVA'].replace('YUCATÁN', 'YUCATAN', inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'ENTIDAD_FEDERATIVA' después de la limpieza ---")
print(df_backup['ENTIDAD_FEDERATIVA'].unique())
print("")
```

```
...
--- Valores únicos en 'ENTIDAD_FEDERATIVA' antes de la limpieza ---
['NUEVO LEÓN' 'CIUDAD DE MEXICO' 'PUEBLA' 'QUERETARO' 'MÉXICO'
 'SAN LUIS POTOSÍ' 'GUANAJUATO' 'TABASCO' 'VERACRUZ' 'BAJA CALIFORNIA'
 'DURANGO' 'AGUASCALIENTES' 'SONORA' 'MICHOACÁN' 'COAHUILA' 'CHIHUAHUA'
 'SINALOA' nan 'MORELOS' 'ZACATECAS' 'COLIMA' 'BAJA CALIFORNIA SUR'
 'JALISCO' 'OAXACA' 'YUCATÁN' 'GUERRERO' 'TAMAULIPAS' 'QUINTANA ROO'
 'NAYARIT' 'TLAXCALA' 'HIDALGO' 'CHIAPAS' 'CAMPECHE']

--- Valores únicos en 'ENTIDAD_FEDERATIVA' después de la limpieza ---
['NUEVO LEÓN' 'CIUDAD DE MEXICO' 'PUEBLA' 'QUERETARO' 'MEXICO'
 'SAN LUIS POTOSI' 'GUANAJUATO' 'TABASCO' 'VERACRUZ' 'BAJA CALIFORNIA'
 'DURANGO' 'AGUASCALIENTES' 'SONORA' 'MICHOACAN' 'COAHUILA' 'CHIHUAHUA'
 'SINALOA' 'DESCONOCIDO' 'MORELOS' 'ZACATECAS' 'COLIMA'
 'BAJA CALIFORNIA SUR' 'JALISCO' 'OAXACA' 'YUCATAN' 'GUERRERO'
 'TAMAULIPAS' 'QUINTANA ROO' 'NAYARIT' 'TLAXCALA' 'HIDALGO' 'CHIAPAS'
 'CAMPECHE']

C:\Users\piitag\AppData\Local\Temp\ipykernel_16408\1494770713.py:10: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. This behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or 'df[col] = df[col].method(value)' instead, to perform the operation inplace on the DataFrame or Series directly.

df_backup['ENTIDAD_FEDERATIVA'].replace('NUEVO LEÓN', 'NUEVO LEON', inplace=True)
C:\Users\piitag\AppData\Local\Temp\ipykernel_16408\1494770713.py:13: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. This behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or 'df[col] = df[col].method(value)' instead, to perform the operation inplace on the DataFrame or Series directly.
```

'CODIGO_ENTIDAD_FEDERATIVA':

```
# Limpieza de la columna 'CODIGO_ENTIDAD_FEDERATIVA'.
print("\n--- Valores únicos en 'CODIGO_ENTIDAD_FEDERATIVA' antes de la limpieza ---")
print(df_backup['CODIGO_ENTIDAD_FEDERATIVA'].unique())

# Limpiamos los valores nulos en la columna 'CODIGO_ENTIDAD_FEDERATIVA'.
df_backup.fillna('CODIGO_ENTIDAD_FEDERATIVA': 'DESCONOCIDO', inplace=True)

# Ahora, los valores 'bbb' los reemplazamos con 'DESCONOCIDO'.
df_backup['CODIGO_ENTIDAD_FEDERATIVA'].replace('bbb', 'DESCONOCIDO', inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'CODIGO_ENTIDAD_FEDERATIVA' después de la limpieza ---")
print(df_backup['CODIGO_ENTIDAD_FEDERATIVA'].unique())
print("")
```

```
...
--- Valores únicos en 'CODIGO_ENTIDAD_FEDERATIVA' antes de la limpieza ---
['19' '9' '21' '22' '15' '24' '11' '27' '30' '2' 'bbb' '10' '1' '26' '16'
 '5' '8' '25' nan '17' '6' '3' '14' '20' '31' '12' '32' '28' '23' '18'
 '29' '13' '7' '4']

--- Valores únicos en 'CODIGO_ENTIDAD_FEDERATIVA' después de la limpieza ---
['19' '9' '21' '22' '15' '24' '11' '27' '30' '2' 'DESCONOCIDO' '10' '1'
 '26' '16' '5' '8' '25' '17' '6' '3' '14' '20' '31' '12' '32' '28' '23'
 '18' '29' '13' '7' '4']

C:\Users\piitag\AppData\Local\Temp\ipykernel_16408\1897424717.py:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. This behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or 'df[col] = df[col].method(value)' instead, to perform the operation inplace on the DataFrame or Series directly.
```

‘INSTITUCION’ :

```
[34] # Limpieza de la columna 'INSTITUCION'.
print("n--- Valores únicos en 'INSTITUCION' antes de la limpieza ---")
print(df_backup['INSTITUCION'].unique())

# Limpiamos los valores nulos en la columna 'INSTITUCION'.
df_backup.fillna({'INSTITUCION': 'DESCONOCIDO'}, inplace=True)

# Ahora, los valores 'bbb' los reemplazamos con 'DESCONOCIDO'.
df_backup['INSTITUCION'].replace('bbb', 'DESCONOCIDO', inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("n--- Valores únicos en 'INSTITUCION' después de la limpieza ---")
print(df_backup['INSTITUCION'].unique())
print("")
```

(34) ✓ 0.0s Python

```
... --- Valores únicos en 'INSTITUCION' antes de la limpieza ---
['IMSS' 'SSE' 'PRIVADO' nan 'ISSSTE' 'SSA' 'bbb' 'PEMEX' 'SNTE' 'SEDENA'
 'SEMAR']

--- Valores únicos en 'INSTITUCION' después de la limpieza ---
['IMSS' 'SSE' 'PRIVADO' 'DESCONOCIDO' 'ISSSTE' 'SSA' 'PEMEX' 'SNTE'
 'SEDENA' 'SEMAR']

C:\Users\pikap\AppData\Local\Temp\ipykernel_16408\1051672342.py:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on
```

```
df_backup['INSTITUCION'].replace('bbb', 'DESCONOCIDO', inplace=True)
```

Visualizamos los valores de las columnas con varios valores y, debido al gran tamaño de valores, no es posible visualizar todos los valores en una sola captura:

- ‘ESTABLECIMIENTO’
- ‘EDAD_ANIOS’
- ‘FECHA_PROCURACION’

3.3 Visualización de columnas con muchos valores

A continuación, se clasificaron las columnas elegidas como columnas con muchos valores:

```
[35] # Columna "ESTABLECIMIENTO".
print(f"Los valores de 'ESTABLECIMIENTO' son: \n--- {(df_main['ESTABLECIMIENTO'].unique())}")
print("")
```

(35) ✓ 0.0s Python

```
... 'INTER-HOSP, S.A DE C.V.' 'CENTRO MEDICO EXCEL S.C.'
'CENTRO DE ESPECIALIDADES MEDICA DE CELAYA S.A. DE C.V.'
'CENTRO HOSPITALARIO B.S.A.L.A.H., S.A DE C.V.'
'HOSPITAL STAR MEDICA CENTRO'
'SERVICIOS MEDICOS QUIRURGICOS DE TOLUCA S.A. DE C.V.'
'CENTRO MEDICO DE ESPECIALIDADES DE CIUDAD JUAREZ S.A. DE C.V.'
'HOSPITAL HISPANO AMERICANO S. A. DE C.V.'
'CLINICA HOSPITAL CULIACAN S.A. DE C.V.'
'MILLENIUM MEDICAL CENTER VERACRUZ, S.A. DE C.V.'
'HOSPITAL PRIVADO SAN JOSE DE CIUDAD OBREGON S.A DE C.V.'
'HOSPITAL SANTA ENGRACIA'
'HOSPITAL DE ALTA ESPECIALIDAD "DR. JUAN GRAHAM CASASUS"'
'HOSPITAL HMG COYOACAN' 'HOSPITAL MERLOS S.A. DE C.V.'
'CORPORATIVO HOSPITAL SATELITE S.A. DE C.V.'
'CLINICA SANTA CRUZ S.A. DE C.V.' 'SANATORIO HIDALGO DE TOLUCA, S. A.'
'HOSPITAL CENTRO MEDICO NOVA S.A. DE C.V.'
'HOSPITAL DE LA MUJER, S.A. DE C.V.'
'OPERADORA DE HOSPITALES ANGELES, S.A. DE C.V. Sucursal Chihuahua'
'HOSPITAL CARDIOLOGICA AGUASCALIENTES'
'HOSPITAL DE BENEFICENCIA ESPANOLA'
'HOSPITAL GUADALUPE CORP. S.A. DE C.V.' 'STAR MEDICA, S. A. DE C. V.'
'HOSPITAL REFORMA' 'HOSPITAL STAR MEDICA VERACRUZ'
'HOSPITAL GENERAL DE ESPECIALIDADES DR. TAVTER BIENETH OSORIO'
```

```
▷ ▶ # Total de valores de "ESTABLECIMIENTO" con .nunique().
print("Total de establecimientos únicos:", df_main['ESTABLECIMIENTO'].nunique())
(37) ✓ 0.0s Python
```

```
... Total de establecimientos únicos: 356
```

```
# Columna "EDAD_ANIOS".
print("Los valores de 'EDAD_ANIOS' son: \n---> {(df_main['EDAD_ANIOS'].unique())}")
print("")
```

(38) ✓ 0.0s

... 5.7000000e+01 6.0000000e+00 1.4000000e+01 1.2000000e+01
5.2000000e+01 2.0000000e+01 2.3000000e+01 2.1000000e+01
9.0000000e+01 2.5000000e+01 3.6000000e+01 3.5000000e+01
8.3000000e+01 7.9000000e+01 5.0000000e+01 8.0000000e+01
6.5000000e+01 1.0000000e+00 5.4000000e+01 2.9000000e+01
5.0000000e+00 6.6000000e+01 3.3000000e+01 7.8000000e+01
4.7000000e+01 4.0000000e+01 8.9000000e+01 7.6000000e+01
7.8000000e+00 6.2000000e+01 1.1000000e+01 6.9000000e+01
8.4000000e+01 7.5000000e+01 6.9999988e-01 7.7000000e+01
8.0000000e+00 7.4000000e+01 8.7000000e+01 1.0000000e+01
-2.0000000e+01 8.1000000e+01 9.2000000e+01 8.6000000e+01
9.2000000e+01 8.1800000e+02 8.5000000e+01 4.2900000e+03
-2.5000000e+01 9.6000000e+01 1.0999999e-01 8.8000000e+01
1.0000000e-01 9.2200000e+02 9.3000000e+01 9.9000000e+01
5.0000000e-01 8.9999976e-01 6.00000024e-01 2.0120000e+03
9.1000000e+01 1.7999995e+00 9.2500000e+02 3.6000000e+02
1.4500000e+02 3.2000000e+02 -3.0000000e+00 1.3100000e+02
1.8410000e+03 9.2900000e+02 -5.2000000e+01 9.4700000e+02
1.2600000e+02 -3.2000000e+01 9.3400000e+02 -9.5200000e+02
9.3800000e+02 -2.9620000e+03 -9.7200000e+02 9.3500000e+02
8.2000000e+02 9.3700000e+02 1.0210000e+03 1.8110000e+03
1.8220000e+03 1.0390000e+03 1.4200000e+02 1.0420000e+03
-9.4900000e+02 1.0000000e+02 9.4100000e+02 1.3500000e+02
1.1300000e+03 1.3600000e+02 1.8110000e+03 1.0340000e+03
8.1700000e+02 8.1400000e+02 1.0500000e+03 9.5200000e+02
9.4400000e+02 1.8160000e+03 1.0250000e+03 9.6100000e+02

```
# Total de valores de "EDAD_ANIOS" con .unique().
print("Total de edades únicas:", df_main['EDAD_ANIOS'].unique())
```

(41) ✓ 0.0s

... Total de edades únicas: 161

```
# Columna "FECHA_PROCURACION".
print("Los valores de 'FECHA_PROCURACION' son: \n---> {(df_main['FECHA_PROCURACION'].unique())}")
print("")
```

(39) ✓ 0.0s

... Los valores de 'FECHA_PROCURACION' son:
---> ['2007-06-11' '2008-05-08' '2009-04-21' ... '2016-12-17' '2017-07-15'
'2019-06-08']

```
# Total de valores de "FECHA_PROCURACION" con .unique().
print("Total de fechas de procuración únicas:", df_main['FECHA_PROCURACION'].unique())
```

(43) ✓ 0.0s

... Total de fechas de procuración únicas: 4481

A partir del diagnóstico exploratorio y de conocer los valores de las columnas de la sección de las columnas con muchos valores, obtenemos los valores que debemos de transformar.

▼ **3.4 Limpieza y transformación de columnas con muchos valores**

Después de conocer todos los valores de cada columna de esta sección, procedemos a transformar los valores que no nos sirven. A continuación, los valores que necesitamos transformar de cada columna:

- Para 'ESTABLECIMIENTO' los valores son: '**nan**' y '**bbb**'. Además, es preferible cambiar palabras particulares.
- Para 'EDAD_ANIOS' los valores son: '**nan**' y '**bbb**'. Además, se transformará el tipo de dato a entero.
- Para 'FECHA_PROCURACIÓN' los valores son: '**nan**' y '**bbb**'. Además, se transformará el tipo de dato a fecha.

Realizamos la limpieza de cada columna.

'ESTABLECIMIENTO': Como se puede ver en la casilla de Markdown, es necesario cambiar palabras particulares, eliminar espacios y dejar más legible los valores.

```
# Limpieza de la columna 'ESTABLECIMIENTO'.
print("\n--- Valores únicos en 'ESTABLECIMIENTO' antes de la limpieza ---")
print(df_backup['ESTABLECIMIENTO'].unique())

# Limpiamos los valores nulos en la columna 'ESTABLECIMIENTO'.
df_backup.fillna({'ESTABLECIMIENTO': 'DESCONOCIDO'}, inplace=True)

# Ahora, los valores 'bbb' los reemplazamos con 'DESCONOCIDO'.
df_backup['ESTABLECIMIENTO'].replace("bbb", 'DESCONOCIDO', inplace=True)

# Simplificamos los nombres largos en la columna 'ESTABLECIMIENTO'.
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace('HOSPITAL', 'HOSP.', regex=True)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace('CENTRO MEDICO', 'C. MED.', regex=True)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace('CLINICA', 'CLIN.', regex=True)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace('UNIVERSITARIO', 'UNIV.', regex=True)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace('NACIONAL', 'NAC.', regex=True)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace('REGIONAL', 'REG.', regex=True)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace('GENERAL', 'GEN.', regex=True)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace('INSTITUTO', 'INST.', regex=True)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace('CENTRO DE SALUD', 'C. SALUD', regex=True)

# Eliminamos siglas como S.A. de C.V., S.A. y S.S. de R.L. que no aportan información relevante.
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace(' S.A. de C.V.', '', regex=False)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace(' S.A. de C.V.', '', regex=False)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace(' S.A. de C.V.', '', regex=False)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace(' S.A de C.V.', '', regex=False)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace(' S.A.C.V.', '', regex=False)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace(' S.A.C.W.', '', regex=False)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace(' S.A.', '', regex=False)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace(' S.S. de R.L.', '', regex=False)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace(' S. A. DE C. V.', '', regex=False)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace(' S. A DE C.V.', '', regex=False)
df_backup['ESTABLECIMIENTO'] = df_backup['ESTABLECIMIENTO'].str.replace(' S. A DE C. V.', '', regex=False)
```

```
# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'ESTABLECIMIENTO' después de la limpieza ---")
print(df_backup['ESTABLECIMIENTO'].unique())
print("")
```

```
--- Valores únicos en 'ESTABLECIMIENTO' antes de la limpieza ---
['U.M.A.E. HOSPITAL DE ESPECIALIDADES No. 25',
 'U.M.A.E. HOSPITAL DE CARDIOLOGIA DEL CENTRO MEDICO NACIONAL SIGLO XXI',
 'nan 'HOSPITAL GENERAL DE QUERETARO',
 'HOSPITAL LOMAS DE SAN LUIS INTERNACIONAL' "HOSPITAL GENERAL DE CELAYA",
 'HOSPITAL GENERAL DE ZONA C/U M.A. No. 46',
 'HOSPITAL CENTRAL "DR. IGNACIO MORONES PRIETO"',
 'HOSPITAL GENERAL REGIONAL DE LEON' "HOSPITAL GENERAL DE ZONA NO. 17",
 'HOSPITAL JUAREZ DE MEXICO ORGANISMO PUBLICO DESCENTRALIZADO',
 'U.M.A.E. HOSPITALES DE ESPECIALIDADES DEL CENTRO MEDICO NACIONAL',
 'HOSPITAL PARA EL NIÑO' "HOSPITAL FRAY JUNIPERO SERRA",
 'OPERADORA DE HOSPITALES ANGELES S.A DE C.V.',
 'HOSPITAL GENERAL DE VERACRUZ',
 'U.M.A.E. HOSPITAL DE ESPECIALIDADES C.M.N. GRAL. DE DIV. "MANUEL AVILA CAMACHO"',
 'HOSPITAL REGIONAL GRAL. IGNACIO ZARAGOZA',
 'HOSPITAL GENERAL DE MEXICO DR. EDUARDO LICEAGA',
 'HOSPITAL GENERAL DE DURANGO',
 'CRUZ ROJA MEXICANA DELEGACION AGUASCALIENTES',
 'HOSPITAL GENERAL DE PUEBLA DR. EDUARDO VAZQUEZ NAVARRO',
 'HOSPITAL GENERAL LA PERLA DE CD. NEZAHUALCOYOTL',
 'HOSP. DE ESPECIALIDADES NO. 2' "HOSPITAL GENERAL "DR. MIGUEL SILVA",
 'U.M.A.E. HOSPITAL DE ESPECIALIDADES NO. 71',
 'HOSPITAL UNIVERSITARIO "DR. JOSE E. GONZALEZ"',
 'HOSPITAL GENERAL DE CIUDAD JUAREZ',
 'HOSPITAL DE ESPECIALIDADES NO. 1 DEL CENTRO MEDICO NACIONAL DEL BAJIO',
 'THE AMERICAN BRITISH COMDRAY MEDICAL CENTER I.A.P (SANTA FE)',
 'BANCO DE ORGANOS Y TEJIDOS (CRUZ ROJA MEXICANA DISTRITO FEDERAL)',
 'HOSPITAL CIVIL DE CULIACAN']

--- Valores únicos en 'ESTABLECIMIENTO' después de la limpieza ---
['U.M.A.E. HOSP. DE ESPECIALIDADES No. 25',
 'U.M.A.E. HOSP. DE CARDIOLOGIA DEL C. MED. NAC. SIGLO XXI' 'DESCONOCIDO',
 'HOSP. GEN. DE QUERETARO' "HOSP. LOMAS DE SAN LUIS INTERNAC.",
 'HOSP. GEN. DE CELAYA" "HOSP. GEN. DE ZONA C/U M.A. No. 46',
 'HOSP. CENTRAL "DR. IGNACIO MORONES PRIETO"' "HOSP. GEN. REG. DE LEON",
 'HOSP. GEN. DE ZONA NO. 17',
 'HOSP. JUAREZ DE MEXICO ORGANISMO PUBLICO DESCENTRALIZADO',
 'U.M.A.E. HOSP.ES DE ESPECIALIDADES DEL C. MED. NAC.',
 'HOSP. PARA EL NIÑO' "HOSP. FRAY JUNIPERO SERRA",
 'OPERADORA DE HOSP.ES ANGELES.' "HOSP. GEN. DE VERACRUZ",
 'U.M.A.E. HOSP. DE ESPECIALIDADES C.M.N. GRAL. DE DIV. "MANUEL AVILA CAMACHO"',
 'HOSP. REG. GRAL. IGNACIO ZARAGOZA',
 'HOSP. GEN. DE MEXICO DR. EDUARDO LICEAGA' "HOSP. GEN. DE DURANGO",
 'CRUZ ROJA MEXICANA DELEGACION AGUASCALIENTES',
 'HOSP. GEN. DE PUEBLA DR. EDUARDO VAZQUEZ NAVARRO',
 'HOSP. GEN. LA PERLA DE CD. NEZAHUALCOYOTL',
 'HOSP. DE ESPECIALIDADES NO. 2' "HOSP. GEN. "DR. MIGUEL SILVA",
 'U.M.A.E. HOSP. DE ESPECIALIDADES NO. 71',
 'HOSP. UNIV. "DR. JOSE E. GONZALEZ"' "HOSP. GEN. DE CIUDAD JUAREZ",
 'HOSP. DE ESPECIALIDADES NO. 1 DEL C. MED. NAC. DEL BAJIO',
 'THE AMERICAN BRITISH COMDRAY MEDICAL CENTER I.A.P (SANTA FE)',
 'BANCO DE ORGANOS Y TEJIDOS (CRUZ ROJA MEXICANA DISTRITO FEDERAL)',
 'HOSP. CIVIL DE CULIACAN',
 'UNIDAD MEDICA DE ALTA ESPECIALIDAD DOCTOR GAUDENCIO GONZALEZ GARZA C. MED. NAC. LA RAZA',
 'HOSP. DE TRAUMATOLOGIA "MAGDALENA DE LAS SALINAS"
```

'EDAD_ANIOS': Transformamos a 0 los valores negativos, los valores que son atípicos y dejamos un máximo de 120.

```
# Limpieza de la columna 'EDAD_ANIOS'.
print("\n--- Valores únicos en 'EDAD_ANIOS' antes de la limpieza ---")
print(df_backup['EDAD_ANIOS'].unique())

# Transformamos los valores de 'EDAD_ANIOS' a enteros, manejando errores.
# Iniciamos con convertir a string para eliminar espacios en blanco.
df_backup['EDAD_ANIOS'] = df_backup['EDAD_ANIOS'].astype(str).str.strip()
# Luego convertimos a numérico.
df_backup['EDAD_ANIOS'] = pd.to_numeric(df_backup['EDAD_ANIOS'], errors='coerce')

# Limpiamos valores negativos a 0 en la columna 'EDAD_ANIOS'.
df_backup.loc[df_backup['EDAD_ANIOS'] < 0, 'EDAD_ANIOS'] = 0

# Ahora, convertimos a enteros, manejando los valores nulos.
df_backup['EDAD_ANIOS'] = df_backup['EDAD_ANIOS'].fillna(0).astype(int)

# Convertimos edades mayores a 120 a 120, ya que es un valor más realista.
df_backup.loc[df_backup['EDAD_ANIOS'] > 120, 'EDAD_ANIOS'] = 120

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'EDAD_ANIOS' después de la limpieza ---")
print(df_backup['EDAD_ANIOS'].unique())
print("")
```

[21] ✓ 0.s Python

...
--- Valores únicos en 'EDAD_ANIOS' antes de la limpieza ---
[4.9000000e+01 nan 6.1000000e+01 7.1000000e+01
 6.3000000e+01 4.1000000e+01 5.8000000e+01 3.1000000e+01
 6.0000000e+01 4.5000000e+01 7.0000000e+01 6.4000000e+01
 4.0000000e+00 2.0000000e+00 5.5000000e+01 3.7000000e+01
 5.3000000e+01 1.7000000e+01 5.1000000e+01 7.2000000e+01
 4.8000000e+01 5.6000000e+01 9.0000000e+00 0.0000000e+00
 4.2000000e+01 5.9000000e+01 2.6000000e+01 3.0000000e+00
 1.8000000e+01 3.8000000e+01 1.3000000e+01 1.5000000e+01
 3.0000000e+01 1.9000000e+01 2.2000000e+01 1.6000000e+01
 7.3000000e+01 6.7000000e+01 3.9000000e+01 4.4000000e+01
 6.8000000e+01 4.3000000e+01 2.4000000e+01 3.2000000e+01
 3.4000000e+01 2.7000000e+01 4.5000000e+01 2.8000000e+01
 5.7000000e+01 6.0000000e+00 1.4000000e+01 1.2000000e+01
 5.2000000e+01 2.8000000e+01 2.3000000e+01 2.1000000e+01
 9.0000000e+01 2.5000000e+01 3.6000000e+01 3.5000000e+01
 8.3000000e+01 7.9000000e+01 5.0000000e+01 8.0000000e+01
 6.5000000e+01 1.8000000e+00 5.4000000e+01 2.9000000e+01
 5.0000000e+00 6.6000000e+01 3.3000000e+01 7.8000000e+01
 4.7000000e+01 4.0000000e+01 8.9000000e+01 7.6000000e+01
 7.0000000e+00 6.2000000e+01 1.1000000e+01 6.9000000e+01
 8.4000000e+01 7.5000000e+01 6.9999998e-01 7.7000000e+01
 8.0000000e+00 7.4000000e+01 8.7000000e+01 1.0000000e+01
 -2.0000000e+01 8.1000000e+01 9.2000000e+01 8.6000000e+01
 8.2000000e+01 8.1800000e+02 8.5000000e+01 -4.2900000e+03
 -2.9000000e+01 9.6000000e+01 1.0999999e-01 8.8000000e+01
 1.0000000e-01 9.7200000e+02 9.3000000e+01 9.9000000e+01
 5.0000000e-01 8.9999976e-01 6.00000024e-01 2.0120000e+01
 9.1000000e+01 1.79999995e+00 9.2500000e+02 3.6000000e+02

--- Valores únicos en 'EDAD_ANIOS' después de la limpieza ---
[49 0 61 71 63 41 58 31 60 45 70 64 4 2 55 37 53 17
 51 72 48 56 9 42 59 26 3 18 38 13 15 30 19 22 16 20 23
 67 39 44 68 43 24 32 34 27 46 28 57 6 14 12 52 20 23
 21 90 25 36 35 83 79 50 80 65 1 54 29 5 66 33 78 47
 40 89 76 7 62 11 69 84 75 77 8 74 87 10 81 92 86 82
 120 85 96 88 93 99 91 100]

'FECHA_PROCURACION': Además de eliminar los valores nulos, convertimos esta variable a formato 'datetime'.

```
# Limpieza de la columna 'FECHA_PROCURACION'.
print("\n--- Valores únicos en 'FECHA_PROCURACION' antes de la limpieza ---")
print(df_backup['FECHA_PROCURACION'].unique())

# Primero, reemplazamos los valores 'bbb' con NAs antes de convertir.
df_backup['FECHA_PROCURACION'] = df_backup['FECHA_PROCURACION'].replace('bbb', pd.NA)

# Convertimos la columna 'FECHA_PROCURACION' a formato datetime, manejando errores.
df_backup['FECHA_PROCURACION'] = pd.to_datetime(df_backup['FECHA_PROCURACION'], errors='coerce')

# Ahora llenamos los valores NaT (Not a Time) con la fecha ficticia '1900-01-01'.
df_backup['FECHA_PROCURACION'] = df_backup['FECHA_PROCURACION'].fillna(pd.Timestamp('1900-01-01'))

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'FECHA_PROCURACION' después de la limpieza ---")
print(df_backup['FECHA_PROCURACION'].unique())
print("")
```

[22] ✓ 0.0s Python

...

```
...
--- Valores únicos en 'FECHA_PROCURACION' antes de la limpieza ---
['2007-06-11' '2008-05-08' '2009-04-21' ... '2016-12-17' '2017-07-15'
 '2019-06-08']

--- Valores únicos en 'FECHA_PROCURACION' después de la limpieza ---
<DatetimeArray>
[ '2007-06-11 00:00:00', '2008-05-08 00:00:00', '2009-04-21 00:00:00',
 '2010-04-29 00:00:00', '2011-09-16 00:00:00', '2012-03-04 00:00:00',
 '2013-07-12 00:00:00', '2014-08-27 00:00:00', '2015-05-20 00:00:00',
 '2016-03-21 00:00:00',
 ...
 '2016-06-05 00:00:00', '2016-08-06 00:00:00', '2017-03-11 00:00:00',
 '2016-11-06 00:00:00', '2016-12-10 00:00:00', '2016-12-18 00:00:00',
 '2016-12-11 00:00:00', '2016-12-17 00:00:00', '2017-07-15 00:00:00',
 '2019-06-08 00:00:00']
Length: 4482, dtype: datetime64[ns]
```

Visualizamos los valores de las columnas con valores binarios:

- ‘RINON_IZQUIERDO’
- ‘RINON_DERECHO’
- ‘RINON_BLOCK’
- ‘PULMON_IZQUIERDO’
- ‘PULMON_DERECHO’
- ‘CORAZON’
- ‘HIGADO’
- ‘PANCREAS’
- ‘INTESTINO’
- ‘CORNEA_IZQUIERDA’
- ‘CORENA_DERECHA’
- ‘PIEL’
- ‘CORAZON_TEJIDOS’
- ‘HUESOS’

```
# Bloque de código para visualizar columnas con el total de órganos donados de cada columna.  
# Se incluye la columna "HUESOS" a pesar de tener un máximo de 14, ya que es importante verificar que no existan valores mayores a 14.  
# El máximo de riñón izquierdo es 1, por lo que, si notamos algún valor mayor a 1, es un valor erróneo.  
print(f"Los valores de 'RINON_IZQUIERDO' son: \n---> {({df_main['RINON_IZQUIERDO'].unique()})}")  
print("")  
  
# El máximo de riñón derecho es 1, por lo que, si notamos algún valor mayor a 1, es un valor erróneo.  
print(f"Los valores de 'RINON_DERECHO' son: \n---> {({df_main['RINON_DERECHO'].unique()})}")  
print("")  
  
# El máximo de bloque de riñón es 1, por lo que, si notamos algún valor mayor a 1, es un valor erróneo.  
print(f"Los valores de 'RINON_BLOCK' son: \n---> {({df_main['RINON_BLOCK'].unique()})}")  
print("")  
  
# El máximo de pulmón izquierdo es 1, por lo que, si notamos algún valor mayor a 1, es un valor erróneo.  
print(f"Los valores de 'PULMON_IZQUIERDO' son: \n---> {({df_main['PULMON_IZQUIERDO'].unique()})}")  
print("")  
  
# El máximo de pulmón derecho es 1, por lo que, si notamos algún valor mayor a 1, es un valor erróneo.  
print(f"Los valores de 'PULMON_DERECHO' son: \n---> {({df_main['PULMON_DERECHO'].unique()})}")  
print("")  
  
# El máximo de corazón es 1, por lo que, si notamos algún valor mayor a 1, es un valor erróneo.  
print(f"Los valores de 'CORAZON' son: \n---> {({df_main['CORAZON'].unique()})}")  
print("")  
  
# El máximo de hígado es 1, por lo que, si notamos algún valor mayor a 1, es un valor erróneo.  
print(f"Los valores de 'HIGADO' son: \n---> {({df_main['HIGADO'].unique()})}")  
print("")  
  
# El máximo de páncreas es 1, por lo que, si notamos algún valor mayor a 1, es un valor erróneo.  
print(f"Los valores de 'PANCREAS' son: \n---> {({df_main['PANCREAS'].unique()})}")  
print("")  
  
# El máximo de intestino es 1, por lo que, si notamos algún valor mayor a 1, es un valor erróneo.  
print(f"Los valores de 'INTESTINO' son: \n---> {({df_main['INTESTINO'].unique()})}")  
print("")  
  
# El máximo de córnea izquierda es 1, por lo que, si notamos algún valor mayor a 1, es un valor erróneo.  
print(f"Los valores de 'CORNEA_IZQUIERDA' son: \n---> {({df_main['CORNEA_IZQUIERDA'].unique()})}")  
print("")  
  
# El máximo de córnea derecha es 1, por lo que, si notamos algún valor mayor a 1, es un valor erróneo.  
print(f"Los valores de 'CORNEA_DERECHA' son: \n---> {({df_main['CORNEA_DERECHA'].unique()})}")  
print("")  
  
# El máximo de piel es 1, por lo que, si notamos algún valor mayor a 1, es un valor erróneo.  
print(f"Los valores de 'PIEL' son: \n---> {({df_main['PIEL'].unique()})}")  
print("")  
  
# El máximo de tejidos de corazón es 1, por lo que, si notamos algún valor mayor a 1, es un valor erróneo.  
print(f"Los valores de 'CORAZON_TEJIDOS' son: \n---> {({df_main['CORAZON_TEJIDOS'].unique()})}")  
print("")  
  
# El máximo de huesos es 14, por lo que, si notamos algún valor mayor a 14, es un valor erróneo.  
print(f"Los valores de 'HUESOS' son: \n---> {({df_main['HUESOS'].unique()})}")  
print("")
```

```

... Los valores de 'RINONIZQUIERDO' son:
---> [ '0' '1' nan '2' 'bbb' '4']

Los valores de 'RINONDERECHO' son:
---> [ 0. 1. 2. nan 4. 3.]

Los valores de 'RINON_BLOCK' son:
---> [ '0' nan '1' 'bbb' '4' '2']

Los valores de 'PULMONIZQUIERDO' son:
---> [ '0' nan '1' 'bbb' '2']

Los valores de 'PULMONDERECHO' son:
---> [ 0. nan 1. 2. ]

Los valores de 'CORAZON' son:
---> [ '0' nan '1' 'bbb' '2']

Los valores de 'HIGADO' son:
---> [ 0. nan 1. 2. ]

Los valores de 'PANCREAS' son:
---> [ 0. nan 1. 2. ]

Los valores de 'INTESTINO' son:
---> [ 0. nan 1. ]

Los valores de 'CORNEAIZQUIERDA' son:
---> [ 1. 0. nan 2. 3. 4. ]

```

```

... Los valores de 'PULMONDERECHO' son:
---> [ 0. nan 1. 2. ]

Los valores de 'CORAZON' son:
---> [ '0' nan '1' 'bbb' '2']

Los valores de 'HIGADO' son:
---> [ 0. nan 1. 2. ]

Los valores de 'PANCREAS' son:
---> [ 0. nan 1. 2. ]

Los valores de 'INTESTINO' son:
---> [ 0. nan 1. ]

Los valores de 'CORNEAIZQUIERDA' son:
---> [ 1. 0. nan 2. 3. 4. ]

Los valores de 'CORNEADERECHA' son:
---> [ 1. nan 0. 2. 3. 4. ]

Los valores de 'PIEL' son:
---> [ 0. 1. nan 2. ]

Los valores de 'CORAZONTEJIDOS' son:
---> [ 0. nan 1. 2. ]

Los valores de 'HUESOS' son:
---> [ 0. 1. 2. nan 9. 5. 6. 12. 8. 14. 4. ]

```

A partir del diagnóstico exploratorio y de conocer los valores de las columnas de la sección de las columnas con valores numéricicos y binarios, obtenemos los valores que debemos de transformar.

3.6 Limpieza y transformación de columnas con valores binarios (0 y 1)

Después de conocer todos los valores de cada columna de esta sección, procedemos a transformar los que no nos sirven. A continuación, los valores que necesitamos transformar de cada columna:

- Para 'RINONIZQUIERDO' los valores son: 'nan', 'bbb', '2' y '4'
- Para 'RINONDERECHO' los valores son: 'nan', '2', '3' y '4'
- Para 'RINON_BLOCK' los valores son: 'nan', 'bbb', '2' y '4'
- Para 'PULMONIZQUIERDO' los valores son: 'nan', 'bbb' y '2'
- Para 'PULMONDERECHO' los valores son: 'nan' y '2'
- Para 'CORAZON' los valores son: 'nan', 'bbb' y '2'
- Para 'HIGADO' los valores son: 'nan' y '2'
- Para 'PANCREAS' los valores son: 'nan' y '2'
- Para 'INTESTINO' los valores son: 'nan'
- Para 'CORNEAIZQUIERDA' los valores son: 'nan', '2', '3' y '4'
- Para 'CORNEADERECHA' los valores son: 'nan', '2', '3' y '4'
- Para 'PIEL' los valores son: 'nan' y '2'
- Para 'CORAZONTEJIDOS' los valores son: 'nan' y '2'
- Para 'HUESOS' los valores son: 'nan'

IMPORTANTE:

1. Se considera la columna 'HUESOS' miembro de este grupo a pesar de tener más de 2 valores debido a su parecido para señalar sus valores: indicar el total de órganos donados (en este caso, huesos).
2. Para evitar posibles errores, convertimos todos los valores a enteros.

Realizamos la limpieza de cada columna.

‘RINON_IZQUIERDO’ :

```
# Limpieza columna 'RINON_IZQUIERDO'.
print("\n--- Valores únicos en 'RINON_IZQUIERDO' antes de la limpieza ---")
print(df_backup['RINON_IZQUIERDO'].unique())

# Convertimos todos los valores a enteros.
df_backup['RINON_IZQUIERDO'] = pd.to_numeric(df_backup['RINON_IZQUIERDO'], errors='coerce').fillna(0).astype(int)

# Limpiamos los valores nulos en la columna 'RINON_IZQUIERDO' reemplazándolos con 0.
df_backup['RINON_IZQUIERDO'].fillna(0, inplace=True)

# Ahora, los valores 'bbb' los remplazamos con 0.
df_backup['RINON_IZQUIERDO'].replace('bbb', 0, inplace=True)

# Se realiza lo mismo con 2 y 3.
df_backup['RINON_IZQUIERDO'].replace(2, 0, inplace=True)
df_backup['RINON_IZQUIERDO'].replace(3, 0, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'RINON_IZQUIERDO' después de la limpieza ---")
print(df_backup['RINON_IZQUIERDO'].unique())
print("")
```

[24] ✓ 0.0s Python

```
... --- Valores únicos en 'RINON_IZQUIERDO' antes de la limpieza ---
[0 '1' nan '2' 'bbb' '4']

--- Valores únicos en 'RINON_IZQUIERDO' después de la limpieza ---
[0 1]
```

‘RINON_DERECHO’ :

```
# Limpieza de la columna 'RINON_DERECHO'.
print("\n--- Valores únicos en 'RINON_DERECHO' antes de la limpieza ---")
print(df_backup['RINON_DERECHO'].unique())

# Convertimos todos los valores a enteros.
df_backup['RINON_DERECHO'] = pd.to_numeric(df_backup['RINON_DERECHO'], errors='coerce').fillna(0).astype(int)

# Limpiamos los valores nulos en la columna 'RINON_DERECHO' reemplazándolos con 0.
df_backup['RINON_DERECHO'].fillna(0, inplace=True)

# Se realiza lo mismo con 2, 3 y 4.
df_backup['RINON_DERECHO'].replace(2, 0, inplace=True)
df_backup['RINON_DERECHO'].replace(3, 0, inplace=True)
df_backup['RINON_DERECHO'].replace(4, 0, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'RINON_DERECHO' después de la limpieza ---")
print(df_backup['RINON_DERECHO'].unique())
print("")
```

[25] ✓ 0.0s Python

```
... --- Valores únicos en 'RINON_DERECHO' antes de la limpieza ---
[ 0.  1.  2. nan  4.  3.]

--- Valores únicos en 'RINON_DERECHO' después de la limpieza ---
[0 1]
```

‘RINON_BLOCK’ :

```
# Limpieza de la columna 'RINON_BLOCK'.
print("\n--- Valores únicos en 'RINON_BLOCK' antes de la limpieza ---")
print(df_backup['RINON_BLOCK'].unique())

# Convertimos todos los valores a enteros.
df_backup['RINON_BLOCK'] = pd.to_numeric(df_backup['RINON_BLOCK'], errors='coerce').fillna(0).astype(int)

# Limpiamos los valores nulos en la columna 'RINON_BLOCK' reemplazándolos con 0.
df_backup['RINON_BLOCK'].fillna(0, inplace=True)

# Ahora, los valores 'bbb' los remplazamos con 0.
df_backup['RINON_BLOCK'].replace('bbb', 0, inplace=True)

# Se realiza lo mismo con 2 y 3.
df_backup['RINON_BLOCK'].replace(2, 0, inplace=True)
df_backup['RINON_BLOCK'].replace(3, 0, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'RINON_BLOCK' después de la limpieza ---")
print(df_backup['RINON_BLOCK'].unique())
print("")
```

[26] ✓ 0.0s Python

```
... --- Valores únicos en 'RINON_BLOCK' antes de la limpieza ---
[ '0' nan '1' 'bbb' '4' '2']

--- Valores únicos en 'RINON_BLOCK' después de la limpieza ---
[0 1]
```

‘PULMON_IZQUIERDO’ :

```
# Limpieza de la columna 'PULMON_IZQUIERDO'.
print("\n--- Valores únicos en 'PULMON_IZQUIERDO' antes de la limpieza ---")
print(df_backup['PULMON_IZQUIERDO'].unique())

# Convertimos todos los valores a enteros.
df_backup['PULMON_IZQUIERDO'] = pd.to_numeric(df_backup['PULMON_IZQUIERDO'], errors='coerce').fillna(0).astype(int)

# Limpiamos los valores nulos en la columna 'PULMON_IZQUIERDO' reemplazándolos con 0.
df_backup['PULMON_IZQUIERDO'].fillna(0, inplace=True)

# Ahora, los valores 'bbb' los remplazamos con 0.
df_backup['PULMON_IZQUIERDO'].replace('bbb', 0, inplace=True)

# Se realiza lo mismo con 2.
df_backup['PULMON_IZQUIERDO'].replace(2, 0, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'PULMON_IZQUIERDO' después de la limpieza ---")
print(df_backup['PULMON_IZQUIERDO'].unique())
print("")
```

[27] ✓ 0.0s Python

...

```
--- Valores únicos en 'PULMON_IZQUIERDO' antes de la limpieza ---
[0 nan 1 'bbb' 2]

--- Valores únicos en 'PULMON_IZQUIERDO' después de la limpieza ---
[0 1]
```

‘PULMON_DERECHO’ :

```
# Limpieza de la columna 'PULMON_DERECHO'.
print("\n--- Valores únicos en 'PULMON_DERECHO' antes de la limpieza ---")
print(df_backup['PULMON_DERECHO'].unique())

# Convertimos todos los valores a enteros.
df_backup['PULMON_DERECHO'] = pd.to_numeric(df_backup['PULMON_DERECHO'], errors='coerce').fillna(0).astype(int)

# Limpiamos los valores nulos en la columna 'PULMON_DERECHO' reemplazándolos con 0.
df_backup['PULMON_DERECHO'].fillna(0, inplace=True)

# Se realiza lo mismo con 2.
df_backup['PULMON_DERECHO'].replace(2, 0, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'PULMON_DERECHO' después de la limpieza ---")
print(df_backup['PULMON_DERECHO'].unique())
print("")
```

[28] ✓ 0.0s Python

...

```
--- Valores únicos en 'PULMON_DERECHO' antes de la limpieza ---
[ 0. nan 1. 2.]

--- Valores únicos en 'PULMON_DERECHO' después de la limpieza ---
[0 1]
```

‘CORAZON’ :

```
# Limpieza de la columna 'CORAZON'.
print("\n--- Valores únicos en 'CORAZON' antes de la limpieza ---")
print(df_backup['CORAZON'].unique())

# Convertimos todos los valores a enteros.
df_backup['CORAZON'] = pd.to_numeric(df_backup['CORAZON'], errors='coerce').fillna(0).astype(int)

# Limpiamos los valores nulos en la columna 'CORAZON' reemplazándolos con 0.
df_backup['CORAZON'].fillna(0, inplace=True)

# Ahora, los valores 'bbb' los remplazamos con 0.
df_backup['CORAZON'].replace('bbb', 0, inplace=True)

# Se realiza lo mismo con 2.
df_backup['CORAZON'].replace(2, 0, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'CORAZON' después de la limpieza ---")
print(df_backup['CORAZON'].unique())
print("")
```

[29] ✓ 0.0s Python

...

```
--- Valores únicos en 'CORAZON' antes de la limpieza ---
[0 nan 1 'bbb' 2]

--- Valores únicos en 'CORAZON' después de la limpieza ---
[0 1]
```

‘HIGADO’ :

```
# Limpieza de la columna 'HIGADO'.
print("\n--- Valores únicos en 'HIGADO' antes de la limpieza ---")
print(df_backup['HIGADO'].unique())

# Convertimos todos los valores a enteros.
df_backup['HIGADO'] = pd.to_numeric(df_backup['HIGADO'], errors='coerce').fillna(0).astype(int)

# Limpiamos los valores nulos en la columna 'HIGADO' reemplazándolos con 0.
df_backup['HIGADO'].fillna(0, inplace=True)

# Se realiza lo mismo con 2.
df_backup['HIGADO'].replace(2, 0, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'HIGADO' después de la limpieza ---")
print(df_backup['HIGADO'].unique())
print("")
```

[30] ✓ 0s Python

...
--- Valores únicos en 'HIGADO' antes de la limpieza ---
[0. nan 1. 2.]
--- Valores únicos en 'HIGADO' después de la limpieza ---
[0 1]

‘PANCREAS’ :

```
# Limpieza de la columna 'PANCREAS'.
print("\n--- Valores únicos en 'PANCREAS' antes de la limpieza ---")
print(df_backup['PANCREAS'].unique())

# Convertimos todos los valores a enteros.
df_backup['PANCREAS'] = pd.to_numeric(df_backup['PANCREAS'], errors='coerce').fillna(0).astype(int)

# Limpiamos los valores nulos en la columna 'PANCREAS' reemplazándolos con 0.
df_backup['PANCREAS'].fillna(0, inplace=True)

# Se realiza lo mismo con 2.
df_backup['PANCREAS'].replace(2, 0, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'PANCREAS' después de la limpieza ---")
print(df_backup['PANCREAS'].unique())
print("")
```

[31] ✓ 0s Python

...
--- Valores únicos en 'PANCREAS' antes de la limpieza ---
[0. nan 1. 2.]
--- Valores únicos en 'PANCREAS' después de la limpieza ---
[0 1]

‘INTESTINO’ :

```
# Limpieza de la columna 'INTESTINO'.
print("\n--- Valores únicos en 'INTESTINO' antes de la limpieza ---")
print(df_backup['INTESTINO'].unique())

# Convertimos todos los valores a enteros.
df_backup['INTESTINO'] = pd.to_numeric(df_backup['INTESTINO'], errors='coerce').fillna(0).astype(int)

# Limpiamos los valores nulos en la columna 'INTESTINO' reemplazándolos con 0.
df_backup['INTESTINO'].fillna(0, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'INTESTINO' después de la limpieza ---")
print(df_backup['INTESTINO'].unique())
print("")
```

[32] ✓ 0s Python

...
--- Valores únicos en 'INTESTINO' antes de la limpieza ---
[0. nan 1.]
--- Valores únicos en 'INTESTINO' después de la limpieza ---
[0 1]

‘CORNEA_IZQUIERDA’ :

```
# Limpieza de la columna 'CORNEA_IZQUIERDA'.
print("\n--- Valores únicos en 'CORNEA_IZQUIERDA' antes de la limpieza ---")
print(df_backup['CORNEA_IZQUIERDA'].unique())

# Convertimos todos los valores a enteros.
df_backup['CORNEA_IZQUIERDA'] = pd.to_numeric(df_backup['CORNEA_IZQUIERDA'], errors='coerce').fillna(0).astype(int)

# Limpiamos los valores nulos en la columna 'CORNEA_IZQUIERDA' reemplazándolos con 0.
df_backup['CORNEA_IZQUIERDA'].fillna(0, inplace=True)

# Se realiza lo mismo con 2,3 y 4.
df_backup['CORNEA_IZQUIERDA'].replace(2, 0, inplace=True)
df_backup['CORNEA_IZQUIERDA'].replace(3, 0, inplace=True)
df_backup['CORNEA_IZQUIERDA'].replace(4, 0, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'CORNEA_IZQUIERDA' después de la limpieza ---")
print(df_backup['CORNEA_IZQUIERDA'].unique())
print("")
```

[33] ✓ 0.0s

--- Valores únicos en 'CORNEA_IZQUIERDA' antes de la limpieza ---
[1. 0. nan 2. 3. 4.]

--- Valores únicos en 'CORNEA_IZQUIERDA' después de la limpieza ---
[1 0]

Python

‘CORNEA_DERECHA’ :

```
# Limpieza de la columna 'CORNEA_DERECHA'.
print("\n--- Valores únicos en 'CORNEA_DERECHA' antes de la limpieza ---")
print(df_backup['CORNEA_DERECHA'].unique())

# Convertimos todos los valores a enteros.
df_backup['CORNEA_DERECHA'] = pd.to_numeric(df_backup['CORNEA_DERECHA'], errors='coerce').fillna(0).astype(int)

# Limpiamos los valores nulos en la columna 'CORNEA_DERECHA' reemplazándolos con 0.
df_backup['CORNEA_DERECHA'].fillna(0, inplace=True)

# Se realiza lo mismo con 2, 3 y 4.
df_backup['CORNEA_DERECHA'].replace(2, 0, inplace=True)
df_backup['CORNEA_DERECHA'].replace(3, 0, inplace=True)
df_backup['CORNEA_DERECHA'].replace(4, 0, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'CORNEA_DERECHA' después de la limpieza ---")
print(df_backup['CORNEA_DERECHA'].unique())
print("")
```

[34] ✓ 0.0s

--- Valores únicos en 'CORNEA_DERECHA' antes de la limpieza ---
[1. nan 0. 2. 3. 4.]

--- Valores únicos en 'CORNEA_DERECHA' después de la limpieza ---
[1 0]

Python

‘PIEL’ :

```
# Limpieza de la columna 'PIEL'.
print("\n--- Valores únicos en 'PIEL' antes de la limpieza ---")
print(df_backup['PIEL'].unique())

# Convertimos todos los valores a enteros.
df_backup['PIEL'] = pd.to_numeric(df_backup['PIEL'], errors='coerce').fillna(0).astype(int)

# Limpiamos los valores nulos en la columna 'PIEL' reemplazándolos con 0.
df_backup['PIEL'].fillna(0, inplace=True)

# Se realiza lo mismo con 2.
df_backup['PIEL'].replace(2, 0, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'PIEL' después de la limpieza ---")
print(df_backup['PIEL'].unique())
print("")
```

[35] ✓ 0.0s

--- Valores únicos en 'PIEL' antes de la limpieza ---
[0. 1. nan 2.]

--- Valores únicos en 'PIEL' después de la limpieza ---
[0 1]

Python

‘CORAZON_TEJIDOS’:

```
[36] # Limpieza de la columna 'CORAZON_TEJIDOS'.
print("\n--- Valores únicos en 'CORAZON_TEJIDOS' antes de la limpieza ---")
print(df_backup['CORAZON_TEJIDOS'].unique())

# Convertimos todos los valores a enteros.
df_backup['CORAZON_TEJIDOS'] = pd.to_numeric(df_backup['CORAZON_TEJIDOS'], errors='coerce').fillna(0).astype(int)

# Limpiamos los valores nulos en la columna 'CORAZON_TEJIDOS' reemplazándolos con 0.
df_backup['CORAZON_TEJIDOS'].fillna(0, inplace=True)

# Se realiza lo mismo con 2.
df_backup['CORAZON_TEJIDOS'].replace(2, 0, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'CORAZON_TEJIDOS' después de la limpieza ---")
print(df_backup['CORAZON_TEJIDOS'].unique())
print("")
```

[36] ✓ 0.0s Python

```
... --- Valores únicos en 'CORAZON_TEJIDOS' antes de la limpieza ---
[ 0. nan  1.  2.] --- Valores únicos en 'CORAZON_TEJIDOS' después de la limpieza ---
[0 1]
```

‘HUESOS’:

```
[37] # Limpieza de la columna 'HUESOS'.
print("\n--- Valores únicos en 'HUESOS' antes de la limpieza ---")
print(df_backup['HUESOS'].unique())

# Convertimos todos los valores a enteros.
df_backup['HUESOS'] = pd.to_numeric(df_backup['HUESOS'], errors='coerce').fillna(0).astype(int)

# Limpiamos los valores nulos en la columna 'HUESOS' reemplazándolos con 0.
df_backup['HUESOS'].fillna(0, inplace=True)

# Verificamos los valores únicos después de la limpieza.
print("\n--- Valores únicos en 'HUESOS' después de la limpieza ---")
print(df_backup['HUESOS'].unique())
print("")
```

[37] ✓ 0.0s Python

```
... --- Valores únicos en 'HUESOS' antes de la limpieza ---
[ 0.  1.  2. nan  9.  5.  6. 12.  8. 14.  4.] --- Valores únicos en 'HUESOS' después de la limpieza ---
[ 0  1  2  9  5  6 12  8 14  4]
```

RESULTADOS DE LA LIMPIEZA Y CREACIÓN DEL NUEVO DATASET

Se comparan las dimensiones y valores nulos del DataFrame original con el que fue limpiado.

```
4. Resultados de la limpieza del DataFrame y creación del nuevo DataSet

Ahora, visualizamos el total de valores nulos que hay en el DataFrame df_backup y lo comparamos con df_main.
```

```
148 [✓] 0Ds Python
    # Dimensiones del DataFrame original y del DataFrame limpio.
    print("---- Dimensiones del DataFrame original: (df_main.shape) ---")
    print("---- Dimensiones del DataFrame limpio: (df_backup.shape) ---")

    # Valores nulos del DataFrame original y del DataFrame limpio.
    print("---- Valores nulos en el DataFrame original ---")
    print(df_main.isnull().sum())
    print("---- Valores nulos en el DataFrame limpio ---")
    print(df_backup.isnull().sum())

---- Dimensiones del DataFrame original: (52501, 24) ---
---- Dimensiones del DataFrame limpio: (51658, 24) ---

---- Valores nulos en el DataFrame original ---
SEXO           2100
CODIGO_SEXO     3119
TIPO_DONANTE   2100
MUERTE         2100
ENTIDAD_FEDERATIVA 2100
CODIGO_ENTIDAD_FEDERATIVA 2100
ESTABLECIMIENTO 2100
INSTITUCION    2100
EDAD_ANIOS     3698
FECHA_PROCURACION 2100
RINON_IZQUIERDO 2100
RINON_DERECHO   2100
RINON_BLOCK    2100
PULMON_IZQUIERDO 2100
PULMON_DERECHO  2100
CORAZON        2100
HIGADO          2100
PANCREAS       2100
INTESTINO       2100
CORNEA_IZQUIERDA 2100
CORNEA_DERECHA  2100
PIEL            2100
HUESOS          2100
CORAZON_TEJIDOS 3113

---- Valores nulos en el DataFrame limpio ---
SEXO           0
CODIGO_SEXO     0
TIPO_DONANTE   0
MUERTE         0
ENTIDAD_FEDERATIVA 0
CODIGO_ENTIDAD_FEDERATIVA 0
ESTABLECIMIENTO 0
INSTITUCION    0
EDAD_ANIOS     0
FECHA_PROCURACION 0
RINON_IZQUIERDO 0
RINON_DERECHO   0
RINON_BLOCK    0
PULMON_IZQUIERDO 0
PULMON_DERECHO  0
CORAZON        0
HIGADO          0
PANCREAS       0
INTESTINO       0
CORNEA_IZQUIERDA 0
CORNEA_DERECHA  0
PIEL            0
HUESOS          0
CORAZON_TEJIDOS 0
dtype: int64

print("\n--- Diagnóstico con .info() ---")
df_backup.info()
print("")
```

```
149 [✓] 0Ds Python
    ... Data columns (total 24 columns):
    # Column      Non-Null Count  Dtype  
    # ---        
    0  SEXO        51658 non-null object 
    1  CODIGO_SEXO 51658 non-null object 
    2  TIPO_DONANTE 51658 non-null object 
    3  MUERTE      51658 non-null object 
    4  ENTIDAD_FEDERATIVA 51658 non-null object 
    5  CODIGO_ENTIDAD_FEDERATIVA 51658 non-null object 
    6  ESTABLECIMIENTO 51658 non-null object 
    7  INSTITUCION  51658 non-null object 
    8  EDAD_ANIOS   51658 non-null int64  
    9  FECHA_PROCURACION 51658 non-null datetime64[ns]
    10 RINON_IZQUIERDO 51658 non-null int64  
    11 RINON_DERECHO  51658 non-null int64  
    12 RINON_BLOCK   51658 non-null int64  
    13 PULMON_IZQUIERDO 51658 non-null int64  
    14 PULMON_DERECHO 51658 non-null int64  
    15 CORAZON       51658 non-null int64  
    16 HIGADO        51658 non-null int64  
    17 PANCREAS      51658 non-null int64  
    18 INTESTINO     51658 non-null int64  
    19 CORNEA_IZQUIERDA 51658 non-null int64  
    20 CORNEA_DERECHA 51658 non-null int64  
    21 PIEL           51658 non-null int64  
    22 HUESOS         51658 non-null int64  
    23 CORAZON_TEJIDOS 51658 non-null int64  
    dtypes: datetime64[ns](1), int64(15), object(8)
    memory usage: 9.9+ MB
```

```
Solo queda crear el nuevo DataSet para su posterior análisis.

# Exportamos el DataFrame limpio a un nuevo archivo CSV.
df_backup.to_csv('data/donantes_organos_limpio.csv', index=False)

[4]: 02s
```

Python

CONCLUSIONES DEL PROYECTO

Este proyecto se centró en la fase inicial y fundamental de todo análisis de datos: la limpieza y preprocesamiento. Partiendo de un conjunto de datos "sucio" (dataset_sucio.csv), el objetivo fue transformar los datos crudos en un conjunto íntegro, coherente y confiable, listo para el análisis exploratorio (EDA) y la visualización.

Problemas Encontrados:

El diagnóstico inicial reveló que el dataset de 52,501 filas y 24 columnas no era viable para un análisis directo. Los problemas principales fueron:

- **Valores nulos y faltantes:** Prácticamente todas las columnas presentaban miles de valores nulos (NaN).
- **Datos contaminados :** Se detectó la presencia recurrente de datos basura, como la cadena de texto 'bbb', en columnas tanto categóricas (SEXO, TIPO_DONANTE) como numéricas (RINONIZQUIERDO).
- **Tipos de datos incorrectos:** Columnas que debían ser puramente numéricas y binarias (como RINONIZQUIERDO o CORAZON) estaban almacenadas como tipo object, impidiendo cualquier cálculo matemático.
- **Datos numéricos inválidos:** La columna EDAD_ANIOS contenía datos imposibles, como edades negativas (ej. -20, -4290) y valores atípicos extremos (ej. 818, 2012).
- **Valores binarios inconsistentes:** Las columnas de donación de órganos, que debían ser 0 (No) o 1 (Sí), contenían valores como 2 y 4.
- **Inconsistencia categórica:** Columnas de texto como ENTIDAD_FEDERATIVA tenían inconsistencias por acentos (ej. 'NUEVO LEÓN' vs. 'NUEVO LEON'), y ESTABLECIMIENTO presentaba nombres excesivamente largos y variantes.
- **Filas duplicadas:** Se identificaron y eliminaron filas que eran copias exactas de otras.

Técnicas de Limpieza Utilizadas:

Para resolver estos problemas, se aplicó un proceso de limpieza metódico utilizando la biblioteca Pandas:

■ **Diagnóstico:** Se usaron las funciones .info(), .isnull().sum() y .unique() para identificar la naturaleza y magnitud de los problemas.

■ **Respaldo de datos:** Se creó una copia del DataFrame (df_backup = df_main.copy()) para asegurar que el conjunto de datos original permaneciera intacto.

■ **Manejo de nulos y basura:**

- En columnas categóricas (SEXO, INSTITUCION), los valores NaN y 'bbb' se reemplazaron por la cadena 'DESCONOCIDO' usando .fillna() y .replace().

- En columnas numéricas y binarias (RINONIZQUIERDO, EDAD_ANIOS), los valores no numéricos se trataron en el siguiente paso.

■ **Transformación y estandarización de tipos:**

- **Binarios (Órganos):** Se utilizó pd.to_numeric(errors='coerce') para forzar la conversión a número (lo que convierte 'bbb' y otros textos a NaN). Posteriormente, se usó .fillna(0) para llenar esos nulos y se reemplazaron valores inválidos (como 2 y 4) por 0. Finalmente, se convirtió la columna a tipo int.

- **Numéricos (Edad):** Se siguió un proceso similar con pd.to_numeric(errors='coerce'). Se usó .loc[] para corregir valores atípicos, estableciendo las edades negativas a 0 y las edades imposibles (mayores a 120) a un límite de 120.

- **Fechas:** Se usó pd.to_datetime(errors='coerce') para convertir la columna FECHA_PROCURACION al formato datetime64[ns], y los valores nulos se imputaron con una fecha de referencia (1900-01-01).

■ **Estandarización de texto:**

- Se eliminaron acentos en ENTIDAD_FEDERATIVA (ej. 'QUERÉTARO' a 'QUERETARO').

- Se simplificaron los nombres en ESTABLECIMIENTO usando .str.replace() para abreviar (ej. 'HOSPITAL' a 'HOSP.') y eliminar texto legal ('S.A. de C.V.').

■ **Verificación final:** Tras la limpieza, se ejecutaron .info() y .isnull().sum() para confirmar la eliminación total de valores nulos y la correcta asignación de tipos de datos.

■ **Exportación:** El DataFrame limpio y verificado se exportó a un nuevo archivo, donantes_organos_limpio.csv.

Aprendizajes del Proceso

El desarrollo de este notebook consolidó varios aprendizajes clave sobre la ciencia de datos:

- + *La limpieza no es un paso previo, es la mitad del proyecto. Ningún análisis puede ser confiable si se basa en datos "sucios". El tiempo invertido en esta fase garantiza la validez de todas las conclusiones futuras.*
- + *Los problemas rara vez son obvios. Los datos "basura" pueden esconderse de muchas formas (texto en columnas numéricas, valores imposibles, inconsistencias sutiles). Las herramientas de diagnóstico como `.unique()` son indispensables.*
- + *La importancia de la conversión de tipos (`Dtype`). El mayor obstáculo para el análisis era que las columnas binarias y numéricas eran de tipo `object`. Su conversión a `int` y `datetime` fue el paso más crítico, ya que habilita la capacidad de sumar, promediar y filtrar datos.*
- + *Trabajar de forma no destructiva. Crear un `df_backup` es una práctica fundamental que permite experimentar con la limpieza sin temor a perder la información original.*