EDA_V1

May 22, 2025

1 *Motor Design Data Driven*

1.1 1. Análisis EDA

1.1.1 1.1. Librerías

[1]:
```python
# Librerías necesarias
import os
import re  # Import the regular expression module

import pandas as pd
import numpy as np
import math

import matplotlib
matplotlib.use('TKAgg')
import matplotlib.pyplot as plt
from matplotlib.ticker import ScalarFormatter
import seaborn as sns
```

1.1.2 1.2. Lectura de fichero

[2]:
```python
# Definir las rutas base y de las carpetas
base_path = os.getcwd()  # Se asume que el notebook se ejecuta desde la carpeta␣
↪'EDA'
db_path = os.path.join(base_path, "DB_EDA")
fig_path = os.path.join(base_path, "Figuras_EDA")

# Ruta al archivo de la base de datos
data_file = os.path.join(db_path, "design_DB_5000_Uniforme.csv")
print(data_file)

# Ruta al archivo de las figuras
figure_path = os.path.join(fig_path, "5000_MOT_Uniforme")
```

```
C:\Users\s00244\Documents\GitHub\MotorDesignDataDriven\Notebooks_TFM\1.EDA\DB_ED
A\design_DB_5000_Uniforme.csv
```

[3]:

```python
# Lectura del archivo CSV
try:
    df = pd.read_csv(data_file)
    print("Archivo cargado exitosamente.")
except FileNotFoundError:
    print("Error: Archivo no encontrado. Revisa la ruta del archivo.")
except pd.errors.ParserError:
    print("Error: Problema al analizar el archivo CSV. Revisa el formato del archivo.")
except Exception as e:
    print(f"Ocurrió un error inesperado: {e}")
```

```
Archivo cargado exitosamente.
```

1.1.3 1.3. Exploración inicial de datos

[4]:

```python
# Exploración inicial de datos

# Mostrar las primeras filas del DataFrame
print("\nPrimeras filas del DataFrame:")
display(df.head())
```

```
Primeras filas del DataFrame:

   x1::OSD  x2::Dint       x3::L     x4::tm     x5::hs2     x6::wt  x7::Nt  \
0    48.60   27.8640  14.800000  2.780311   6.312467  4.392325       6
1    54.60   23.1040  32.800001  3.080830  11.833245  2.379534      18
2    59.40   24.0560  29.200001  2.121244  10.249868  2.569301      12
3    54.72   32.0528  22.960001  2.456926   7.797124  2.123813      18
4    48.84   21.9616  25.120000  3.032073   6.972909  2.557345      14

   x8::Nh  m1::Drot    m2::Dsh  …  p2::Tnom  p3::nnom     p4::GFF  p5::BSP_T  \
0       4   26.8640  13.342235  …      0.11    3960.0  40.082719   0.170606
1       5   22.1040   9.341198  …      0.11    3960.0  49.664102   0.990486
2       3   23.0560  11.940368  …      0.11    3960.0  24.675780   0.412852
3       3   31.0528  16.981004  …      0.11    3960.0  42.652370   0.538189
4       3   20.9616   8.622712  …      0.11    3960.0  57.017278   0.380920

    p6::BSP_n  p7::BSP_Pm  p8::BSP_Mu  p9::BSP_Irms  p10::MSP_n  p11::UWP_Mu
0  17113.2350   305.74251   90.763857     10.070335  18223.3200    86.138152
1   2684.3461   278.42958   79.546525     12.589184   3576.9857          NaN
2   4913.5479   212.43125   87.076820      7.558136   5737.1407    88.799881
3   3806.5372   214.53262   83.929471      7.553457   4325.1237    83.402341
4   5161.0967   205.87507   87.040314      7.554095   6293.4336    91.343493

[5 rows x 25 columns]
```

[5]:
```python
# Información general del DataFrame
print("\nInformación general del DataFrame:")
df.info()
```

```
Información general del DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5242 entries, 0 to 5241
Data columns (total 25 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   x1::OSD       5242 non-null   float64
 1   x2::Dint      5242 non-null   float64
 2   x3::L         5242 non-null   float64
 3   x4::tm        5242 non-null   float64
 4   x5::hs2       5242 non-null   float64
 5   x6::wt        5242 non-null   float64
 6   x7::Nt        5242 non-null   int64
 7   x8::Nh        5242 non-null   int64
 8   m1::Drot      5242 non-null   float64
 9   m2::Dsh       5242 non-null   float64
 10  m3::he        5242 non-null   float64
 11  m4::Rmag      5242 non-null   float64
 12  m5::Rs        5242 non-null   float64
 13  m6::GFF       5242 non-null   float64
 14  p1::W         4447 non-null   float64
 15  p2::Tnom      5242 non-null   float64
 16  p3::nnom      5242 non-null   float64
 17  p4::GFF       4447 non-null   float64
 18  p5::BSP_T     4447 non-null   float64
 19  p6::BSP_n     4447 non-null   float64
 20  p7::BSP_Pm    4447 non-null   float64
 21  p8::BSP_Mu    4447 non-null   float64
 22  p9::BSP_Irms  4447 non-null   float64
 23  p10::MSP_n    4447 non-null   float64
 24  p11::UWP_Mu   3761 non-null   float64
dtypes: float64(23), int64(2)
memory usage: 1024.0 KB
```

[6]:
```python
# Estadísticas descriptivas del DataFrame
print("\nEstadísticas descriptivas:")
display(df.describe())
```

Estadísticas descriptivas:

	x1::OSD	x2::Dint	x3::L	x4::tm	x5::hs2	\
count	5242.000000	5242.000000	5242.000000	5242.000000	5242.000000	
mean	55.847039	27.152434	24.947683	2.734945	8.682188	

```
std       3.439244     4.358982     8.751453     0.431730     2.287659
min      45.000960    21.204387    10.000384     2.000021     5.006201
25%      53.792861    23.525393    17.302182     2.361072     6.794815
50%      56.622163    26.299126    24.934452     2.732837     8.460640
75%      58.712525    29.916908    32.475884     3.111981    10.328053
max      59.999232    42.026714    39.998003     3.499768    14.946449

            x6::wt       x7::Nt       x8::Nh     m1::Drot      m2::Dsh  …  \
count  5242.000000  5242.000000  5242.000000  5242.000000  5242.000000  …
mean      3.309290    10.793781     5.039107    26.152434    12.924705  …
std       0.840682     5.317152     1.848228     4.358982     3.186535  …
min       2.000405     5.000000     3.000000    20.204387     8.007388  …
25%       2.588997     7.000000     3.000000    22.525393    10.385444  …
50%       3.251083     9.000000     5.000000    25.299126    12.318815  …
75%       3.987706    13.000000     6.000000    28.916908    14.972997  …
max       4.998505    30.000000     9.000000    41.026714    24.794923  …

       p2::Tnom  p3::nnom      p4::GFF     p5::BSP_T     p6::BSP_n  \
count   5242.00    5242.0  4447.000000  4447.000000   4447.000000
mean       0.11    3960.0    43.437674     0.540111   8235.127695
std        0.00       0.0    11.062318     0.294003   5658.244776
min        0.11    3960.0    20.937265     0.054076    761.280320
25%        0.11    3960.0    34.321520     0.321027   4266.651600
50%        0.11    3960.0    44.167043     0.477237   6815.689000
75%        0.11    3960.0    52.863005     0.703906  10557.782500
max        0.11    3960.0    66.633388     2.012437  38941.723000

         p7::BSP_Pm   p8::BSP_Mu  p9::BSP_Irms    p10::MSP_n  p11::UWP_Mu
count  4447.000000  4447.000000   4447.000000   4447.000000  3761.000000
mean    346.133033    87.690220     12.627435   9521.799554    88.320642
std     131.417512     4.343265      4.653808   6145.426096     2.934205
min     127.215630    65.162410      7.534755   1171.984600    70.238550
25%     227.242150    86.019082      7.554244   5222.421750    86.861221
50%     307.699100    89.006020     12.577613   7998.616600    89.029744
75%     442.864210    90.657291     15.107079  12100.442500    90.438868
max     706.812390    93.531236     22.702016  43867.109000    92.893227

[8 rows x 25 columns]
```

1.1.4 1.4. Visualización de los datos

[7]:
```python
# Estilo visual
sns.set(style='whitegrid')

# Agrupar columnas por prefijo
x_cols = [col for col in df.columns if col.startswith('x') and df[col].dtype in␣
↪['float64', 'int64']]
```

```
m_cols = [col for col in df.columns if col.startswith('m') and df[col].dtype in␣
 ↪['float64', 'int64']]
p_cols = [col for col in df.columns if col.startswith('p') and df[col].dtype in␣
 ↪['float64', 'int64']]

def plot_variable_group(columns, group_name):
    if not columns:
        print(f"No hay variables para el grupo '{group_name}'")
        return

    n = len(columns)
    cols = 3  # número de columnas de subplots
    rows = math.ceil(n / cols)

    fig, axes = plt.subplots(rows, cols, figsize=(cols * 6, rows * 4))
    axes = axes.flatten()

    for i, col in enumerate(columns):
        ax = axes[i]
        sns.histplot(df[col], kde=True, ax=ax, color='skyblue',␣
 ↪edgecolor='black')
        ax.set_title(f'Distribución de {col}', fontsize=12)
        ax.ticklabel_format(style='scientific', axis='x', scilimits=(0, 0))
        ax.set_xlabel(col, fontsize=10)
        ax.set_ylabel('Frecuencia', fontsize=10)

    # Eliminar ejes vacíos
    for j in range(i + 1, len(axes)):
        fig.delaxes(axes[j])

    fig.suptitle(f'Distribuciones del grupo "{group_name}"', fontsize=16)
    plt.tight_layout(rect=[0, 0, 1, 0.97])
    # Guardar la figura en la carpeta 'Figuras_EDA/(La carpeta que corresponda)'
    figure_file = os.path.join(figure_path, f"Distribuciones del␣
 ↪grupo_{group_name}.png")
    plt.savefig(figure_file, dpi =1080)
    plt.close()
    #plt.show()

# Generar subplots por grupo
plot_variable_group(x_cols, 'x')
plot_variable_group(m_cols, 'm')
plot_variable_group(p_cols, 'p')
```

[8]:
```
def plot_heatmap(subset_df, title, xlabel, ylabel):
    if subset_df.empty:
        print(f"No hay datos para {title}")
```

```python
        return

    plt.figure(figsize=(max(10, 0.5 * subset_df.shape[1]), max(6, 0.4 *␣
↪subset_df.shape[0])))
    sns.heatmap(subset_df, annot=True, fmt=".2f", cmap='coolwarm', linewidths=0.
↪5,
                cbar_kws={'label': 'Correlación'}, annot_kws={"size": 8})
    plt.title(title, fontsize=14, weight='bold')
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.xticks(rotation=45, ha='right')
    plt.yticks(rotation=0)
    plt.tight_layout()
    # Guardar la figura en la carpeta 'Figuras_EDA/(La carpeta que corresponda)'
    figure_file = os.path.join(figure_path, f"{title}.png")
    plt.savefig(figure_file, dpi =1080)
    plt.close()
    #plt.show()

# Correlación p-x
if p_cols and x_cols:
    corr_px = df[p_cols + x_cols].corr().loc[p_cols, x_cols]
    plot_heatmap(corr_px, 'Mapa de calor_Variables p vs x', 'x', 'p')

# Correlación p-m
if p_cols and m_cols:
    corr_pm = df[p_cols + m_cols].corr().loc[p_cols, m_cols]
    plot_heatmap(corr_pm, 'Mapa de calor_Variables p vs m', 'm', 'p')

# Correlación p-p
if p_cols:
    corr_pp = df[p_cols].corr()
    plot_heatmap(corr_pp, 'Mapa de calor_Variables p vs p', 'p', 'p')
```

1.1.5 1.5. Preprocesado de los datos

[9]:
```python
# Verificar y corregir tipos de datos incorrectos usando expresiones regulares
def correct_dtype_regex(df):
    numeric_regex = re.compile(r'^-?\d+(\.\d+)?$')
    for col in df.columns:
        if df[col].dtype == 'object':
            if df[col].dropna().apply(lambda x: bool(numeric_regex.
↪match(str(x)))).all():
                df[col] = pd.to_numeric(df[col])
                print(f"Columna '{col}' convertida a tipo numérico exitosamente␣
↪usando regex.")
            else:
```

```
                print(f"Columna '{col}' no puede ser convertida directamente a␣
↪numérico.")
    return df

df = correct_dtype_regex(df)
display(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5242 entries, 0 to 5241
Data columns (total 25 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   x1::OSD        5242 non-null   float64
 1   x2::Dint       5242 non-null   float64
 2   x3::L          5242 non-null   float64
 3   x4::tm         5242 non-null   float64
 4   x5::hs2        5242 non-null   float64
 5   x6::wt         5242 non-null   float64
 6   x7::Nt         5242 non-null   int64
 7   x8::Nh         5242 non-null   int64
 8   m1::Drot       5242 non-null   float64
 9   m2::Dsh        5242 non-null   float64
 10  m3::he         5242 non-null   float64
 11  m4::Rmag       5242 non-null   float64
 12  m5::Rs         5242 non-null   float64
 13  m6::GFF        5242 non-null   float64
 14  p1::W          4447 non-null   float64
 15  p2::Tnom       5242 non-null   float64
 16  p3::nnom       5242 non-null   float64
 17  p4::GFF        4447 non-null   float64
 18  p5::BSP_T      4447 non-null   float64
 19  p6::BSP_n      4447 non-null   float64
 20  p7::BSP_Pm     4447 non-null   float64
 21  p8::BSP_Mu     4447 non-null   float64
 22  p9::BSP_Irms   4447 non-null   float64
 23  p10::MSP_n     4447 non-null   float64
 24  p11::UWP_Mu    3761 non-null   float64
dtypes: float64(23), int64(2)
memory usage: 1024.0 KB
```

```
None
```

[10]:
```
# Optimización del uso de memoria reduciendo tamaño de tipos de datos
for col in df.select_dtypes(include=['int64', 'float64']).columns:
    if df[col].dtype == 'int64':
        df[col] = df[col].astype('int32')
        print(f"Columna '{col}' convertida de int64 a int32.")
    elif df[col].dtype == 'float64':
```

```python
        df[col] = df[col].astype('float32')
        print(f"Columna '{col}' convertida de float64 a float32.")
display(df.info())
```

```
Columna 'x1::OSD' convertida de float64 a float32.
Columna 'x2::Dint' convertida de float64 a float32.
Columna 'x3::L' convertida de float64 a float32.
Columna 'x4::tm' convertida de float64 a float32.
Columna 'x5::hs2' convertida de float64 a float32.
Columna 'x6::wt' convertida de float64 a float32.
Columna 'x7::Nt' convertida de int64 a int32.
Columna 'x8::Nh' convertida de int64 a int32.
Columna 'm1::Drot' convertida de float64 a float32.
Columna 'm2::Dsh' convertida de float64 a float32.
Columna 'm3::he' convertida de float64 a float32.
Columna 'm4::Rmag' convertida de float64 a float32.
Columna 'm5::Rs' convertida de float64 a float32.
Columna 'm6::GFF' convertida de float64 a float32.
Columna 'p1::W' convertida de float64 a float32.
Columna 'p2::Tnom' convertida de float64 a float32.
Columna 'p3::nnom' convertida de float64 a float32.
Columna 'p4::GFF' convertida de float64 a float32.
Columna 'p5::BSP_T' convertida de float64 a float32.
Columna 'p6::BSP_n' convertida de float64 a float32.
Columna 'p7::BSP_Pm' convertida de float64 a float32.
Columna 'p8::BSP_Mu' convertida de float64 a float32.
Columna 'p9::BSP_Irms' convertida de float64 a float32.
Columna 'p10::MSP_n' convertida de float64 a float32.
Columna 'p11::UWP_Mu' convertida de float64 a float32.
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5242 entries, 0 to 5241
Data columns (total 25 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   x1::OSD       5242 non-null   float32
 1   x2::Dint      5242 non-null   float32
 2   x3::L         5242 non-null   float32
 3   x4::tm        5242 non-null   float32
 4   x5::hs2       5242 non-null   float32
 5   x6::wt        5242 non-null   float32
 6   x7::Nt        5242 non-null   int32
 7   x8::Nh        5242 non-null   int32
 8   m1::Drot      5242 non-null   float32
 9   m2::Dsh       5242 non-null   float32
 10  m3::he        5242 non-null   float32
 11  m4::Rmag      5242 non-null   float32
 12  m5::Rs        5242 non-null   float32
 13  m6::GFF       5242 non-null   float32
```

```
 14  p1::W         4447 non-null   float32
 15  p2::Tnom      5242 non-null   float32
 16  p3::nnom      5242 non-null   float32
 17  p4::GFF       4447 non-null   float32
 18  p5::BSP_T     4447 non-null   float32
 19  p6::BSP_n     4447 non-null   float32
 20  p7::BSP_Pm    4447 non-null   float32
 21  p8::BSP_Mu    4447 non-null   float32
 22  p9::BSP_Irms  4447 non-null   float32
 23  p10::MSP_n    4447 non-null   float32
 24  p11::UWP_Mu   3761 non-null   float32
dtypes: float32(23), int32(2)
memory usage: 512.0 KB

None
```

[11]:

```python
# Verificación de valores faltantes y duplicados
print("\nValores faltantes por columna:")
display(df.isnull().sum())
```

```
Valores faltantes por columna:

x1::OSD            0
x2::Dint           0
x3::L              0
x4::tm             0
x5::hs2            0
x6::wt             0
x7::Nt             0
x8::Nh             0
m1::Drot           0
m2::Dsh            0
m3::he             0
m4::Rmag           0
m5::Rs             0
m6::GFF            0
p1::W            795
p2::Tnom           0
p3::nnom           0
p4::GFF          795
p5::BSP_T        795
p6::BSP_n        795
p7::BSP_Pm       795
p8::BSP_Mu       795
p9::BSP_Irms     795
p10::MSP_n       795
p11::UWP_Mu     1481
dtype: int64
```

[12]:
```python
print("\nCantidad de filas duplicadas:")
display(df.duplicated().sum())
```

```
Cantidad de filas duplicadas:
```

```
np.int64(0)
```

[13]:
```python
# Identifica las filas con valores NaN en cualquier matriz
rows_with_nan = df[df.isnull().any(axis=1)].index

# Obtiene el conjunto de todos los índices con NaN
all_nan_indices = set(rows_with_nan)
all_nan_indices = sorted(list(all_nan_indices))

# Elimina las filas con valores NaN.
df_cleaned = df.drop(index=all_nan_indices)

display(df_cleaned.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3761 entries, 0 to 5241
Data columns (total 25 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   x1::OSD       3761 non-null   float32
 1   x2::Dint      3761 non-null   float32
 2   x3::L         3761 non-null   float32
 3   x4::tm        3761 non-null   float32
 4   x5::hs2       3761 non-null   float32
 5   x6::wt        3761 non-null   float32
 6   x7::Nt        3761 non-null   int32
 7   x8::Nh        3761 non-null   int32
 8   m1::Drot      3761 non-null   float32
 9   m2::Dsh       3761 non-null   float32
 10  m3::he        3761 non-null   float32
 11  m4::Rmag      3761 non-null   float32
 12  m5::Rs        3761 non-null   float32
 13  m6::GFF       3761 non-null   float32
 14  p1::W         3761 non-null   float32
 15  p2::Tnom      3761 non-null   float32
 16  p3::nnom      3761 non-null   float32
 17  p4::GFF       3761 non-null   float32
 18  p5::BSP_T     3761 non-null   float32
 19  p6::BSP_n     3761 non-null   float32
 20  p7::BSP_Pm    3761 non-null   float32
 21  p8::BSP_Mu    3761 non-null   float32
 22  p9::BSP_Irms  3761 non-null   float32
 23  p10::MSP_n    3761 non-null   float32
```

```
 24  p11::UWP_Mu   3761 non-null   float32
dtypes: float32(23), int32(2)
memory usage: 396.7 KB
```

None

[14]:

```python
# Tabla de estadísticas descriptivas
print("\nTabla de estadísticas descriptivas finales:")
display(df_cleaned.describe().T)
```

```
Tabla de estadísticas descriptivas finales:

              count          mean          std          min          25%  \
x1::OSD      3761.0     55.744644     3.460788    45.003456    53.590309
x2::Dint     3761.0     26.694340     4.074372    21.204388    23.338999
x3::L        3761.0     23.823900     8.494957    10.000384    16.565634
x4::tm       3761.0      2.732233     0.432805     2.000021     2.354096
x5::hs2      3761.0      8.798923     2.204605     5.006847     7.025496
x6::wt       3761.0      3.304939     0.833983     2.000405     2.591386
x7::Nt       3761.0      9.603031     4.167259     5.000000     6.000000
x8::Nh       3761.0      5.273597     1.860813     3.000000     4.000000
m1::Drot     3761.0     25.694340     4.074372    20.204388    22.338999
m2::Dsh      3761.0     12.602922     2.997488     8.007388    10.256752
m3::he       3761.0      5.726226     1.810047     3.500325     4.261017
m4::Rmag     3761.0     12.164112     2.035033     9.361403    10.492299
m5::Rs       3761.0     22.146091     2.109504    15.755375    20.641384
m6::GFF      3761.0     37.111980     9.556508    20.004059    29.085491
p1::W        3761.0      0.569902     0.155213     0.255234     0.445454
p2::Tnom     3761.0      0.110000     0.000000     0.110000     0.110000
p3::nnom     3761.0   3960.000000     0.000000  3960.000000  3960.000000
p4::GFF      3761.0     42.554726    11.028567    20.937265    33.193748
p5::BSP_T    3761.0      0.474520     0.235081     0.110104     0.298206
p6::BSP_n    3761.0   9024.212891  5188.172852  2347.505127  5247.996582
p7::BSP_Pm   3761.0    367.535553   129.422104   138.840652   259.329926
p8::BSP_Mu   3761.0     88.922539     2.745711    73.908340    87.576187
p9::BSP_Irms 3761.0     13.267383     4.689486     7.534754    10.066794
p10::MSP_n   3761.0  10409.014648  5660.459961  3977.654297  6298.291992
p11::UWP_Mu  3761.0     88.320641     2.934205    70.238548    86.861221

                     50%          75%          max
x1::OSD        56.496574    58.670208    59.999233
x2::Dint       25.801891    29.232868    41.778736
x3::L          23.265970    30.747520    39.998001
x4::tm          2.727570     3.106093     3.499768
x5::hs2         8.556318    10.381292    14.946449
x6::wt          3.249449     3.977364     4.998168
x7::Nt          9.000000    12.000000    30.000000
x8::Nh          5.000000     7.000000     9.000000
```

```
m1::Drot        24.801891     28.232868     40.778736
m2::Dsh         11.985586     14.483794     24.794922
m3::he           5.296264      6.792925     13.000177
m4::Rmag        11.708392     13.434603     19.883490
m5::Rs          22.277868     23.764368     26.461054
m6::GFF         37.159924     45.031456     54.999233
p1::W            0.555020      0.683692      1.005128
p2::Tnom         0.110000      0.110000      0.110000
p3::nnom      3960.000000   3960.000000   3960.000000
p4::GFF         42.848991     51.901443     66.633385
p5::BSP_T        0.433830      0.592823      1.657419
p6::BSP_n     7634.479980  11194.800781  38941.722656
p7::BSP_Pm     351.486145    459.191101    706.812378
p8::BSP_Mu      89.512253     90.878654     93.531235
p9::BSP_Irms    12.587963     17.621677     22.702017
p10::MSP_n    8791.044922  12816.920898  43867.109375
p11::UWP_Mu     89.029747     90.438866     92.893227
```

1.1.6 1.6. Almacenar el preprocesado

[15]:
```python
# Guardar DataFrame preprocesado
print("\nDataFrame después del preprocesamiento:")
# Ruta al archivo de la base de datos
data_cleaned_file = os.path.join(db_path, 'design_DB_preprocessed.csv')
df_cleaned.to_csv(data_cleaned_file, index=False)
# Confirmación de preprocesamiento
print("\nPreprocesamiento completado exitosamente. Archivo 'datos_preprocesados.
 ↪csv' guardado.")
```

```
DataFrame después del preprocesamiento:

Preprocesamiento completado exitosamente. Archivo 'datos_preprocesados.csv'
guardado.
```
