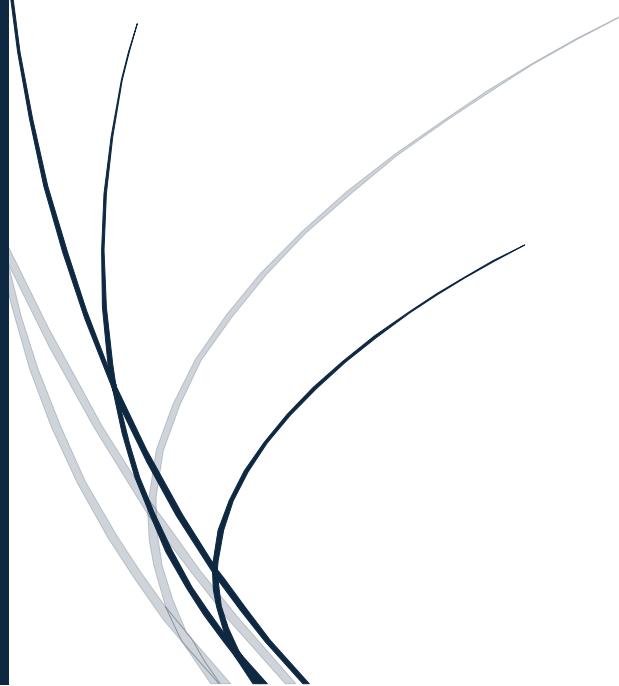


20-10-2025

# Limpieza de Datos

## Introducción a la Ciencia de Datos



Autor: Lic. Edgardo Flores Gutiérrez

PROFESOR: JAIME ALEJANDRO ROMERO SIERRA

REPOSITORIO GIT HUB:

[HTTPS://GITHUB.COM/EDGARDO2501/PROYECTO\\_CIENCIA\\_DE\\_DATOS](https://github.com/EDGARDO2501/PROYECTO_CIENCIA_DE_DATOS)

# Limpieza de datos

El siguiente archivo muestra el proceso para limpiar un dataset con valores nulos, valores de texto en columnas numéricicas, filas duplicadas y las transformaciones necesarias para realizar una limpieza exitosa.

## Acerca del conjunto de datos

Este conjunto de datos presenta el desempeño financiero de Deutsche Bank, incluidas métricas clave como el ingreso operativo, el ingreso neto y los indicadores del balance. Abarca 6800 registros a lo largo del tiempo, lo que lo hace adecuado para proyectos de análisis financiero, modelado y aprendizaje automático. El conjunto de datos se puede utilizar para tareas como evaluación del rendimiento, análisis de ratios y previsión.

- Fecha: La fecha correspondiente a cada registro financiero (a partir de enero de 2015).
- Resultado\_Operativo: Los ingresos generados por las operaciones comerciales principales del banco.
- Gastos: Costos totales incurridos durante las operaciones.
- Activos: Activos totales propiedad del banco (p. ej., efectivo, inversiones).
- Pasivos: El total de deudas y obligaciones contraídas.
- Patrimonio\_Neto: Patrimonio de los accionistas, que representa el valor neto de los activos menos los pasivos.
- Ingresos: Ingresos totales de todas las operaciones y actividades.
- Flujo\_efectivo: El efectivo neto generado o utilizado en las operaciones.
- Resultado\_Neto: Beneficio después de restar los gastos de los ingresos operativos.
- Ratio\_deuda\_patrimonio: Relación financiera que muestra la proporción de deuda en comparación con el capital.
- ROA (Return on Assets): Métrica de rentabilidad calculada como el ingreso neto dividido por los activos totales.

- Margen\_Utilidad: Un ratio que muestra el porcentaje de ingresos que permanece como beneficio.
- Gastos\_Financieros: Costos asociados a los préstamos o deudas del banco.
- Impuestos: El monto pagado como impuestos sobre las ganancias.
- Dividendos: La parte de las ganancias distribuida a los accionistas como dividendos.

## 1. Extracción

```
In [22]: # Importar las bibliotecas necesarias  
  
import pandas as pd  
import numpy as np
```

```
In [23]: # Importar el conjunto de datos a limpiar  
df = pd.read_csv('df_sucio.csv')
```

## 2. Transformación

### 2.1 Exploracion del Dataframe

```
In [24]: # Visualizar el número de filas y columnas del DataFrame  
  
print(f"El conjunto de datos contiene {df.shape[0]} filas y {df.shape[1]} columnas.")
```

El conjunto de datos contiene 7033 filas y 15 columnas.

```
In [25]: # Contar los valores nulos en cada columna  
df.isnull().sum()
```

```
Out[25]: Date          0  
Operating_Income    486  
Expenses           351  
Assets              351  
Liabilities         351  
Equity              351  
Revenue             351  
Cash_Flow           351  
Net_Income          351  
Debt_to_Equity      484  
ROA                 351  
Profit_Margin        351  
Interest_Expense     351  
Tax_Expense          482  
Dividend_Payout      351  
dtype: int64
```

```
In [26]: # Crear una tabla que muestre el porcentaje de valores nulos en cada columna  
null_percentage = (df.isnull().sum() / len(df)) * 100  
null_percentage = null_percentage=null_percentage[null_percentage > 0].sort_values(ascending=False) # Filtrar solo las columnas con valores nulos y ordenar de mayor a menor  
  
# Mostrar la tabla de porcentaje de valores nulos  
null_percentage_table = pd.DataFrame({'% de valores nulos': null_percentage})  
print(null_percentage_table)
```

	% de valores nulos
Operating_Income	6.910280
Debt_to_Equity	6.881843
Tax_Expense	6.853405
Liabilities	4.990758
Assets	4.990758
Expenses	4.990758
Revenue	4.990758
Equity	4.990758
Net_Income	4.990758
Cash_Flow	4.990758
ROA	4.990758
Profit_Margin	4.990758
Interest_Expense	4.990758
Dividend_Payout	4.990758

```
In [27]: # Obtener información general sobre el DataFrame  
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 7033 entries, 0 to 7032  
Data columns (total 15 columns):  
 #   Column           Non-Null Count  Dtype     
---  --    
 0   Date             7033 non-null    object    
 1   Operating_Income 6547 non-null    float64   
 2   Expenses          6682 non-null    float64   
 3   Assets            6682 non-null    object    
 4   Liabilities       6682 non-null    object    
 5   Equity            6682 non-null    float64   
 6   Revenue           6682 non-null    float64   
 7   Cash_Flow         6682 non-null    float64   
 8   Net_Income        6682 non-null    object    
 9   Debt_to_Equity   6549 non-null    float64   
 10  ROA              6682 non-null    float64   
 11  Profit_Margin    6682 non-null    float64   
 12  Interest_Expense 6682 non-null    object    
 13  Tax_Expense       6551 non-null    float64   
 14  Dividend_Payout  6682 non-null    float64  
dtypes: float64(10), object(5)  
memory usage: 824.3+ KB
```

```
In [28]: # Visualizar el nombre de las columnas (variables)
df.columns
```

```
Out[28]: Index(['Date', 'Operating_Income', 'Expenses', 'Assets', 'Liabilities',
       'Equity', 'Revenue', 'Cash_Flow', 'Net_Income', 'Debt_to_Equity', 'ROA',
       'Profit_Margin', 'Interest_Expense', 'Tax_Expense', 'Dividend_Payout'],
      dtype='object')
```

```
In [29]: # Visualizar las estadísticas descriptivas del DataFrame
df.describe()
```

	Operating_Income	Expenses	Equity	Revenue	Cash_Flow	Debt_to_Equity	ROA	Profit_Margin	Tax_Expe
<b>count</b>	6.547000e+03	6.682000e+03	6.682000e+03	6.682000e+03	6.682000e+03	6549.000000	6682.000000	6682.000000	6.551000e
<b>mean</b>	5.480086e+06	2.750231e+06	5.449851e+07	8.437142e+06	4.228204e+06	5.450289	0.013589	0.433249	7.659882e
<b>std</b>	2.628738e+06	1.323608e+06	2.590829e+07	3.720943e+06	2.143893e+06	5.244191	0.020944	0.613714	4.183861e
<b>min</b>	1.045554e+06	5.208441e+05	1.005881e+07	2.000399e+06	5.070756e+05	0.240000	-0.040000	-1.530000	5.034946e
<b>25%</b>	3.181688e+06	1.548318e+06	3.109830e+07	5.209141e+06	2.450360e+06	2.060000	0.000000	0.060000	3.998043e
<b>50%</b>	5.568493e+06	2.834771e+06	5.404533e+07	8.574058e+06	4.119509e+06	3.880000	0.010000	0.330000	7.503705e
<b>75%</b>	7.755533e+06	3.845787e+06	7.752392e+07	1.154675e+07	6.061746e+06	6.900000	0.020000	0.630000	1.147344e
<b>max</b>	9.997459e+06	4.997362e+06	9.996019e+07	1.497074e+07	7.990947e+06	35.450000	0.130000	3.590000	1.498851e

```
In [30]: # Visualizar cuantas filas hay duplicadas
df.duplicated().sum()
```

```
Out[30]: np.int64(33)
```

```
In [31]: # Contar las ocurrencias de cada fecha
conteo_fechas = df['Date'].value_counts()

# Filtrar para mostrar solo las fechas que aparecen más de una vez
fechas_duplicadas = conteo_fechas[conteo_fechas > 1]
```

```
print("Fechas duplicadas y su frecuencia:")
print(fechas_duplicadas)
```

Fechas duplicadas y su frecuencia:

Date

```
2020-04-28    3
2021-05-23    3
2027-08-25    3
2030-10-01    3
2015-01-03    2
...
2032-10-18    2
2032-11-04    2
2032-12-24    2
2033-04-15    2
2033-04-26    2
```

Name: count, Length: 229, dtype: int64

In [32]:

```
# Crear un filtro booleano para las filas donde la fecha está duplicada
filtro_fechas_dup = df['Date'].duplicated(keep=False)

# Aplicar el filtro para ver todas las filas involucradas
filas_con_fechas_duplicadas = df[filtro_fechas_dup]

print("Filas completas que tienen fechas duplicadas:")
print(filas_con_fechas_duplicadas.sort_values(by='Date'))
```

Filas completas que tienen fechas duplicadas:

	Date	Operating_Income	Expenses	Assets	Liabilities	\	
2	2015-01-03	7587945.48	3093297.62	151995381.0	175658980.1		
6830	2015-01-03	7587945.48	3093297.62	151995381.0	175658980.1		
9	2015-01-10	7372653.20	1009571.21	293005361.6	192520616.1		
6845	2015-01-10	7372653.20	1009571.21	293005361.6	192520616.1		
6932	2015-01-18	5722807.88		NaN	347827635.4	34818484.78	
...	...	...	...	...	...	...	
6842	2032-12-24	6179267.60	631612.77	403543228.1	43637225.94		
6865	2033-04-15	2365073.92	2090396.21	55267974.83	188689392.7		
6679	2033-04-15	2365073.92	2090396.21	55267974.83	188689392.7		
6855	2033-04-26	6971515.92		NaN	133338005.5	124944432.0	
6690	2033-04-26	6971515.92	4321380.43	133338005.5	124944432.0		
	Equity	Revenue	Cash_Flow	Net_Income	Debt_to_Equity	ROA	\
2	NaN	NaN	7320721.28	4494647.86	6.40	0.03	
6830	27447676.11	11882092.09	7320721.28	4494647.86	6.40	0.03	
9	64949207.95	2736845.62	6947340.09	6363081.99	2.96	0.02	
6845	64949207.95	2736845.62	6947340.09	6363081.99	2.96	0.02	
6932	10179292.17	13643969.28	6647186.52	3427295.72	3.42	0.01	
...	...	...	...	...	...	...	
6842	51362565.15	4402899.17	2193421.64	bbb	0.85	0.01	
6865	61655487.95	8282016.99	3057214.80	274677.71	3.06	0.00	
6679	61655487.95	8282016.99	3057214.80	274677.71	3.06	NaN	
6855	99786546.77	9037867.47	6856607.25	2650135.49	1.25	0.02	
6690	99786546.77	9037867.47		NaN	2650135.49	1.25	0.02
	Profit_Margin	Interest_Expense	Tax_Expense	Dividend_Payout			
2	0.38	337977.34		NaN	1603358.92		
6830	0.38	337977.34	345091.11		1603358.92		
9	2.32	972677.01	1414000.77		2404522.39		
6845	2.32	972677.01		NaN	2404522.39		
6932	0.25	562041.96	1377353.49		NaN		
...	...	...	...	...	...		
6842	1.26	1640776.87	465927.64		2450751.16		
6865	0.03	223308.63	1108339.22		2648755.32		
6679	0.03	223308.63	1108339.22		2648755.32		
6855	0.29	1964628.34	828445.76		2914937.28		
6690	0.29	1964628.34		NaN	2914937.28		

[462 rows x 15 columns]

```
In [33]: # Visualizar cuántos valores únicos hay en cada columna  
valores_unicos = df.nunique()  
print("Número de valores únicos en cada columna:")  
print(valores_unicos)
```

Número de valores únicos en cada columna:

```
Date           6800  
Operating_Income    800  
Expenses         800  
Assets            801  
Liabilities       801  
Equity             800  
Revenue            800  
Cash_Flow          800  
Net_Income         801  
Debt_to_Equity     568  
ROA                 18  
Profit_Margin      230  
Interest_Expense    801  
Tax_Expense          800  
Dividend_Payout      800  
dtype: int64
```

## 2.2 Transformación

### 2.2.1. Eliminar filas duplicadas

```
In [34]: # Eliminar filas duplicadas basadas en la columna 'Date', manteniendo la primera ocurrencia con keep='first'  
df2 = df.drop_duplicates(subset=['Date'], keep='first')
```

```
In [35]: # Verificar si aún hay filas duplicadas  
df2.duplicated().sum()
```

```
Out[35]: np.int64(0)
```

```
In [36]: # Verificar el número de columnas y filas después de eliminar duplicados  
  
print(f"El conjunto de datos contiene {df2.shape[0]} filas y {df2.shape[1]} columnas.")
```

El conjunto de datos contiene 6800 filas y 15 columnas.

## 2.2.2 Coercionar las filas a formato numérico y sustituir los valores de texto por el valor promedio de la columna

In [43]:

```
# Crear lista de columnas de formato 'object' (contienen texto) pero deberían ser numéricas (float)
columnas = ['Assets', 'Liabilities', 'Net_Income', 'Interest_Expense']

# Iterar sobre cada columna para limpiarla
for col in columnas:
    # Forzar la conversión a número. Los valores que no se puedan convertir
    # se transformarán en NaN.
    df2.loc[:, col] = pd.to_numeric(df2[col], errors='coerce') # errors='coerce' convierte errores en NaN

    # Calcular el promedio de la columna (ignora los NaN automáticamente)
    mean_value = df2[col].mean()

    # Rellenar los valores NaN con el promedio calculado
    df2.loc[:, col] = df2[col].fillna(mean_value)
```

In [38]:

```
# Verificar los tipos de datos después de la conversión
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6800 entries, 0 to 6799
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Date              6800 non-null    object  
 1   Operating_Income  6332 non-null    float64 
 2   Expenses          6466 non-null    float64 
 3   Assets            6800 non-null    float64 
 4   Liabilities       6800 non-null    float64 
 5   Equity             6465 non-null    float64 
 6   Revenue           6463 non-null    float64 
 7   Cash_Flow         6454 non-null    float64 
 8   Net_Income        6800 non-null    float64 
 9   Debt_to_Equity   6333 non-null    float64 
 10  ROA               6459 non-null    float64 
 11  Profit_Margin    6465 non-null    float64 
 12  Interest_Expense 6800 non-null    float64 
 13  Tax_Expense       6334 non-null    float64 
 14  Dividend_Payout  6461 non-null    float64 
dtypes: float64(14), object(1)
memory usage: 850.0+ KB
```

## 2.2.3 Convertir la columna "Date" a formato datetime

```
In [40]: # Convertir la columna de fecha al tipo datetime
df2.loc[:, 'Date'] = df2['Date'].astype('datetime64[ns]')
```

```
In [41]: # Verificar los tipos de datos después de la conversión
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6800 entries, 0 to 6799
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Date              6800 non-null    datetime64[ns]
 1   Operating_Income  6332 non-null    float64 
 2   Expenses          6466 non-null    float64 
 3   Assets            6800 non-null    float64 
 4   Liabilities       6800 non-null    float64 
 5   Equity             6465 non-null    float64 
 6   Revenue           6463 non-null    float64 
 7   Cash_Flow         6454 non-null    float64 
 8   Net_Income        6800 non-null    float64 
 9   Debt_to_Equity   6333 non-null    float64 
 10  ROA               6459 non-null    float64 
 11  Profit_Margin    6465 non-null    float64 
 12  Interest_Expense 6800 non-null    float64 
 13  Tax_Expense       6334 non-null    float64 
 14  Dividend_Payout  6461 non-null    float64 
dtypes: datetime64[ns](1), float64(14)
memory usage: 850.0 KB
```

## 2.2.4 Sustituir los valores nulos por el valor promedio de la columna

```
In [ ]: # Crear lista de columnas
lista_col=df.columns

# Iterar sobre cada columna para limpiarla
for col in lista_col:
    # Calcular el promedio de la columna
    mean_value = df2[col].mean()

    # Rellenar los valores NaN con el promedio calculado
    df2[col].fillna(mean_value, inplace=True)
```

```
In [60]: # Verificar si aún hay valores nulos
df2.isnull().sum()
```

```
Out[60]: Date      0
          Operating_Income 0
          Expenses      0
          Assets        0
          Liabilities   0
          Equity         0
          Revenue       0
          Cash_Flow     0
          Net_Income    0
          Debt_to_Equity 0
          ROA           0
          Profit_Margin 0
          Interest_Expense 0
          Tax_Expense    0
          Dividend_Payout 0
          dtype: int64
```

## 2.2.5 Traducir los nombres de las columnas al Español

```
In [61]: df.columns
```

```
Out[61]: Index(['Date', 'Operating_Income', 'Expenses', 'Assets', 'Liabilities',
       'Equity', 'Revenue', 'Cash_Flow', 'Net_Income', 'Debt_to_Equity', 'ROA',
       'Profit_Margin', 'Interest_Expense', 'Tax_Expense', 'Dividend_Payout'],
       dtype='object')
```

```
In [ ]: # Traducir Los nombres de las columnas al español
df2=df2.rename(columns={'Date': 'Fecha', 'Operating_Income': 'Resultado_Operativo', 'Expenses': 'Gastos', 'Assets': 'Activos',
                      'Liabilities': 'Pasivos', 'Equity': 'Patrimonio_Neto', 'Revenue': 'Ingresos',
                      'Cash_Flow': 'Flujo_efectivo', 'Net_Income': 'Resultado_Neto', 'Debt_to_Equity': 'Ratio_deuda_patrimonio',
                      'Profit_Margin': 'Margen_Utilidad', 'Interest_Expense': 'Gastos_Financieros',
                      'Tax_Expense': 'Impuestos', 'Dividend_Payout': 'Dividendos'})

# Visualizar Los nombres de las columnas después del cambio
df2
```

Out[ ]:

	Fecha	Resultado_Operativo	Gastos	Activos	Pasivos	Patrimonio_Neto	Ingresos	Flujo_efectivo	Resultado_Net
0	2015-01-01	4.370861e+06	3682573.85	1.363403e+08	2.095012e+08	5.980341e+07	9.435946e+06	1428845.20	688287.20
1	2015-01-02	9.556429e+06	1186425.69	1.955172e+08	4.725052e+07	5.528192e+07	8.435807e+06	1029017.28	8370003.08
2	2015-01-03	7.587945e+06	3093297.62	1.519954e+08	1.756590e+08	5.442243e+07	8.435807e+06	7320721.28	4494647.80
3	2015-01-04	5.483306e+06	3230217.71	2.097483e+08	1.382626e+08	8.729351e+07	4.000699e+06	1925965.75	3157708.61
4	2015-01-05	2.404168e+06	2408588.02	8.124073e+07	2.773058e+08	7.092467e+07	3.940243e+06	6659376.16	-4420.26
...	...	...	...	...	...	...	...	...	...
6795	2033-08-09	1.228157e+06	1507179.27	4.983505e+08	1.522763e+08	9.778088e+07	3.616496e+06	5005941.12	-279022.58
6796	2033-08-10	6.086199e+06	857387.16	6.145149e+07	3.530014e+08	5.442243e+07	1.218110e+07	5618837.98	5228811.49
6797	2033-08-11	5.217946e+06	2712175.46	2.352945e+08	3.479339e+07	5.595227e+07	3.819836e+06	4622879.41	2505770.31
6798	2033-08-12	6.680248e+06	4230119.63	1.817699e+08	1.673065e+08	6.282429e+07	4.469170e+06	1409772.49	2450128.00
6799	2033-08-13	4.267741e+06	3544310.24	2.436923e+08	4.925220e+07	5.955970e+07	9.135825e+06	3431880.70	723430.41

6800 rows × 15 columns



In [63]:

```
# Visualizar todo el DataFrame Limpio
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6800 entries, 0 to 6799
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Fecha            6800 non-null    datetime64[ns]
 1   Resultado_Operativo  6800 non-null    float64 
 2   Gastos           6800 non-null    float64 
 3   Activos          6800 non-null    float64 
 4   Pasivos          6800 non-null    float64 
 5   Patrimonio_Neto  6800 non-null    float64 
 6   Ingresos         6800 non-null    float64 
 7   Flujo_efectivo  6800 non-null    float64 
 8   Resultado_Neto   6800 non-null    float64 
 9   Ratio_deuda_patrimonio  6800 non-null    float64 
 10  ROA              6800 non-null    float64 
 11  Margen_Utilidad 6800 non-null    float64 
 12  Gastos_Financieros  6800 non-null    float64 
 13  Impuestos        6800 non-null    float64 
 14  Dividendos       6800 non-null    float64 
dtypes: datetime64[ns](1), float64(14)
memory usage: 850.0 KB
```

## 2.2.6 Tratamiento de valores atípicos

```
In [ ]: # Revisar los valores atípicos en cada columna
for col in df2.select_dtypes(include=[np.number]).columns:
    Q1 = df2[col].quantile(0.25)
    Q3 = df2[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df2[(df2[col] < lower_bound) | (df2[col] > upper_bound)]
    print(f"Columna: {col}, Número de valores atípicos: {outliers.shape[0]}")
```

Los valores atípicos son casos de estudio dentro del dataset, ya que podría representar eventos reales o excepcionales que podrían indicar información valiosa. Por lo que, en el análisis se estará realizando el tratamiento específicos a estos valores.

## 3. Carga del Dataframe

Exportar el dataframe limpio a formato csv para su análisis.

```
In [65]: # Guardar el DataFrame Limpio en un nuevo archivo CSV  
df2.to_csv('df_deutsche_bank.csv', index=False)
```

## Conclusiones

El proceso de limpieza del dataset fue riguroso ya que existían valores nulos, texto en columnas, filas duplicadas y valores nulos. Con un análisis exploratorio previo se pudo localizar todos estos hallazgos, lo que me permitió tener claridad de los pasos a seguir para realizar una limpieza exitosa. Las técnicas usadas para realizar la limpieza fueron:

1. Eliminar filas duplicadas con "drop\_duplicates"
2. Coercionar las filas a formato numérico (a excepción de la columna "Date")
3. Sustituir los valores de texto por el valor promedio de la columna
4. Sustituir los valores nulos por el valor promedio de la columna
5. Traducir los nombres de las columnas al español
6. Visualizar los outliers para su tratamiento en la fase de análisis

Durante el proceso aprendí a realizar una limpieza ordenada y estructurada que me permitió diseñar el proceso a seguir para lograr limpiar el dataset de manera correcta y lista para su análisis.