

Introducción a Pandas

¿Qué es Pandas?

- Pandas es una biblioteca de Python especializada en la manipulación y análisis de datos. Ofrece estructuras de datos eficientes para manejar grandes volúmenes de información y herramientas para limpieza, transformación y visualización de datos.
- Pandas nos permite analizar big data y sacar conclusiones basadas en teorías estadísticas.
- Pandas puede limpiar conjuntos de datos desordenados y hacerlos legibles y relevantes.

Ventajas de Pandas.

- Permite trabajar con datos tabulares (similares a Excel o SQL)
- Facilita la manipulación y limpieza de datos.
- Compatible con otras Bibliotecas como NumPy y Matplotlib.
- Soporta multiples formatos de datos (CSV, Excel, SQL, JSON)

Instalación y uso de Pandas

- Para instalar Pandas podemos usar el siguiente comando desde la terminal o desde una celda de Jupyter.

```
In [4]: # Directo de la celda
%pip install pandas
#pip install pandas desde la terminal
```

Requirement already satisfied: pandas in c:\python\python311\lib\site-packages (2.2.1)Note: you may need to restart the kernel to use updated packages.

[notice] A new release of pip is available: 24.0 -> 25.0.1

[notice] To update, run: python.exe -m pip install --upgrade pip

Requirement already satisfied: numpy<2,>=1.23.2 in c:\python\python311\lib\site-packages (from pandas) (1.26.3)

Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\lalo\appdata\roaming\python\python311\site-packages (from pandas) (2.8.2)

Requirement already satisfied: pytz>=2020.1 in c:\python\python311\lib\site-packages (from pandas) (2024.1)

Requirement already satisfied: tzdata>=2022.7 in c:\python\python311\lib\site-packages (from pandas) (2024.1)

Requirement already satisfied: six>=1.5 in c:\users\lalo\appdata\roaming\python\python311\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)

Podemos instalar mas de una Libreria juntas. Desde la consola pip install pandas numpy matplotlib seaborn

Vamos a verificar la instalación.

```
In [6]: import pandas as pd
print(pd.__version__)
pd.show_versions()
```

2.2.1

```
c:\Python\Python311\Lib\site-packages\_distutils_hack\__init__.py:33: UserWarning:  
Setuptools is replacing distutils.  
  warnings.warn("Setuptools is replacing distutils.")
```

INSTALLED VERSIONS

```

-----
commit          : bdc79c146c2e32f2cab629be240f01658cfb6cc2
python          : 3.11.0.final.0
python-bits     : 64
OS              : Windows
OS-release      : 10
Version         : 10.0.19045
machine         : AMD64
processor        : Intel64 Family 6 Model 78 Stepping 3, GenuineIntel
byteorder       : little
LC_ALL          : None
LANG            : None
LOCALE          : Spanish_Mexico.1252


pandas          : 2.2.1
numpy           : 1.26.3
pytz            : 2024.1
dateutil        : 2.8.2
setuptools      : 65.5.0
pip             : 24.0
Cython          : None
pytest          : None
hypothesis      : None
sphinx          : None
blosc           : None
feather         : None
xlsxwriter      : None
lxml.etree      : 5.3.0
html5lib        : None
pymysql         : None
psycopg2        : None
jinja2          : 3.1.2
IPython         : 8.6.0
pandas_datareader : None
adbc-driver-postgresql : None
adbc-driver-sqlite : None
bs4             : 4.12.2
bottleneck      : None
dataframe-api-compat : None
fastparquet     : None
fsspec          : 2023.12.2
gcsfs           : None
matplotlib      : 3.6.3
numba           : None
numexpr         : None
odfpy           : None
openpyxl        : None
pandas_gbq      : None
pyarrow         : None
pyreadstat      : None
python-calamine : None
pyxlsb          : None
s3fs            : None
scipy           : 1.15.1
sqlalchemy      : None
tables          : None
tabulate        : None
xarray          : None
xlrd            : None
zstandard       : None
tzdata          : 2024.1
qtpy            : 2.3.1
pyqt5           : None

```

Estructuras de datos con Pandas.

- Pandas trabaja principalmente con Series y DataFrames.

Series.

- Una serie es una estructura de datos similar a una lista, pero con etiquetas de índice.

```
In [7]: import pandas as pd

# Crear una Series con una lista
datos = [10, 20, 30, 40]
serie = pd.Series(datos)

print(serie)
```

```
0    10
1    20
2    30
3    40
dtype: int64
```

```
In [8]: serie = pd.Series([10, 20, 30, 40], index=["a", "b", "c", "d"])
print(serie)
```

```
a    10
b    20
c    30
d    40
dtype: int64
```

DataFrame (Tablas bidimensionales)

- Un DataFrame es una estructura tabular similar a una hoja de cálculo de Excel o una tabla de SQL.

```
In [9]: # Crear un DataFrame a partir de un diccionario
datos = {
    "Nombre": ["Ana", "Luis", "Carlos"],
    "Edad": [25, 30, 35],
    "Ciudad": ["Madrid", "Barcelona", "Valencia"]
}

df = pd.DataFrame(datos)
print(df)
```

| | Nombre | Edad | Ciudad |
|---|--------|------|-----------|
| 0 | Ana | 25 | Madrid |
| 1 | Luis | 30 | Barcelona |
| 2 | Carlos | 35 | Valencia |

```
In [11]: print("\nAccedemos a una columna")
print(df['Nombre'])

print("\nAccedemos a una fila por indice")
print(df.loc[1])
```

```
Accedemos a una columna
0      Ana
1      Luis
2      Carlos
Name: Nombre, dtype: object
```

```
Accedemos a una fila por indice
Nombre      Luis
Edad        30
Ciudad      Barcelona
Name: 1, dtype: object
```

Operaciones Básicas en Pandas.

Creación de DataFrames y Series.

- Ya vimos cómo crear Series y DataFrames, pero ahora exploraremos más formas de hacerlo.
- Una Serie de Pandas es como una columna en una tabla.
- Es una matriz unidimensional que contiene datos de cualquier tipo.
- Si no se especifica nada más, los valores se etiquetan con su número de índice. El primer valor tiene índice 0, el segundo tiene índice 1.

Crear un DataFrame desde una lista de listas

```
In [13]: import pandas as pd

datos = [
    ["Ana", 25, "Mexico"],
    ["Luis", 30, "Canada"],
    ["Carlos", 35, "Cuba"]
]

df = pd.DataFrame(datos, columns=["Nombre", "Edad", "Ciudad"])
print(df)
```

```
   Nombre  Edad  Ciudad
0     Ana    25   Mexico
1    Luis    30   Canada
2  Carlos    35     Cuba
```

Indexación y Selección de Datos.

- Pandas ofrece múltiples formas de acceder a los datos dentro de un DataFrame o Serie.

```
In [14]: print("\nAcceder a una columna")
print(df["Nombre"])

print("\nAcceder a una fila por indice")
print(df.loc[1])

print("\nAcceder a una celda especifica")
print(df.at[1, "Edad"])
```

```
Acceder a una columna
0      Ana
1      Luis
2      Carlos
Name: Nombre, dtype: object
```

```
Acceder a una fila por indice
Nombre      Luis
Edad        30
Ciudad      Canada
Name: 1, dtype: object
```

```
Acceder a una celda especifica
30
```

Filtrado de datos.

```
In [19]: print("\nFiltrar filas según la condición")
df_filtrado = df[df["Edad"] > 28]
print(df_filtrado)

print("\nFiltrar filas con múltiples condiciones")
df_filtrado = df[(df["Edad"] > 25) & (df["Ciudad"] == "Canada")]
print(df_filtrado)
```

```
Filtrar filas según la condición
   Nombre  Edad  Ciudad
1    Luis    31  Canada
2  Carlos    36    Cuba
```

```
Filtrar filas con múltiples condiciones
   Nombre  Edad  Ciudad
1    Luis    31  Canada
```

Modificación de datos.

```
In [20]: print("\nModificar valores de una columna.")
df["Edad"] = df["Edad"] + 1
print(df)

print("\nModificar un valor específico.")
df.at[0, "Ciudad"] = "Paris"
print(df)
```

```
Modificar valores de una columna.
   Nombre  Edad  Ciudad
0     Ana    27   Paris
1    Luis    32  Canada
2  Carlos    37    Cuba
```

```
Modificar un valor específico.
   Nombre  Edad  Ciudad
0     Ana    27   Paris
1    Luis    32  Canada
2  Carlos    37    Cuba
```

Manipulación y limpieza de Datos con Pandas

- En este apartado aprenderemos a manejar valores nulos, eliminar duplicado, transformar datos y aplicar funciones para limpiar y preparar los datos antes del análisis.

Manejo de valores nulos.

- En Pandas, los valores nulos se representan con NaN (Not a Number). Podemos encontrarlos y tratarlos de varias formas.

```
In [21]: import pandas as pd
import numpy as np

print("\nIDENTIFICANDO VALORES NULOS")

datos = {
    "Nombre": ["Ana", "Luis", "Carlos", np.nan],
    "Edad": [25, np.nan, 35, 40],
    "Ciudad": ["Madrid", "Barcelona", np.nan, "Sevilla"]
}

df = pd.DataFrame(datos)
print(df)
print("\nValores nulos por columna:\n", df.isnull().sum())
```

```
IDENTIFICANDO VALORES NULOS
   Nombre  Edad  Ciudad
0     Ana  25.0   Madrid
1     Luis   NaN  Barcelona
2    Carlos  35.0      NaN
3      NaN  40.0   Sevilla
```

```
Valores nulos por columna:
Nombre    1
Edad      1
Ciudad    1
dtype: int64
```

```
In [22]: print("\nEliminar valores nulos")
df_sin_nulos = df.dropna() # Elimina las filas que contienen NaN
print(df_sin_nulos)

print("\nRellenar valores nulos")
df_relleno = df.fillna({"Nombre": "Desconocido", "Edad": df["Edad"].mean(), "Ciudad": df["Ciudad"].mode()[0]})
print(df_relleno)
```

```
Eliminar valores nulos
   Nombre  Edad  Ciudad
0     Ana  25.0   Madrid
```

```
Rellenar valores nulos
   Nombre      Edad  Ciudad
0     Ana  25.000000   Madrid
1     Luis  33.333333  Barcelona
2    Carlos  35.000000 No especificado
3 Desconocido  40.000000   Sevilla
```

Eliminación de duplicados.

- Podemos detectar y eliminar datos duplicados con estas funciones.

```
In [25]: print("\nEncontrando duplicados.")

df = pd.DataFrame({
    "Nombre": ["Ana", "Pao", "Lalo", "Ana"],
    "Edad": [25, 30, 35, 25]
})
print(df.duplicated())
```

```
print("\nEliminando duplicados.")
df_sin_dup = df.drop_duplicates()
print(df_sin_dup)
```

Encontrando duplicados.

```
0    False
1    False
2    False
3     True
dtype: bool
```

Eliminando duplicados.

| | Nombre | Edad |
|---|--------|------|
| 0 | Ana | 25 |
| 1 | Pao | 30 |
| 2 | Lalo | 35 |

Eliminar filas con muchos valores nulos.

- Si una fila tiene demasiados valores NaN, puede ser mejor eliminarla.

Eliminar filas con más del 50% de valores nulos.

```
In [26]: df = pd.DataFrame({
    "Nombre": ["Ana", "Luis", np.nan, "María", "Sofía"],
    "Edad": [25, np.nan, np.nan, 22, 22],
    "Salario": [5000, np.nan, np.nan, 4800, 5100]
})

df_limpio = df.dropna(thresh=2) # Al menos 2 valores no nulos
print(df_limpio)
```

| | Nombre | Edad | Salario |
|---|--------|------|---------|
| 0 | Ana | 25.0 | 5000.0 |
| 3 | María | 22.0 | 4800.0 |
| 4 | Sofía | 22.0 | 5100.0 |

Rellenar valores nulos de una columna basado en otra.

- Podemos asignar valores faltantes basándonos en otra columna.

Asignar la media de Salario según la Edad

```
In [27]: df["Salario"] = df.groupby("Edad")["Salario"].transform(lambda x: x.fillna(x.mean()))
print(df)
```

| | Nombre | Edad | Salario |
|---|--------|------|---------|
| 0 | Ana | 25.0 | 5000.0 |
| 1 | Luis | NaN | NaN |
| 2 | NaN | NaN | NaN |
| 3 | María | 22.0 | 4800.0 |
| 4 | Sofía | 22.0 | 5100.0 |

Reemplazar valores nulos con datos externos.

- Podemos usar un diccionario de valores predefinidos.

Rellenar con valores específicos según la columna.

```
In [28]: df.fillna({"Edad": 30, "Salario": 4000}, inplace=True)
print(df)
```


| | Nombre | Edad | Salario |
|---|--------|------|---------|
| 0 | Ana | 25.0 | 5000.0 |
| 1 | Luis | 30.0 | 4000.0 |
| 2 | NaN | 30.0 | 4000.0 |
| 3 | María | 22.0 | 4800.0 |
| 4 | Sofía | 22.0 | 5100.0 |

Transformaciones y Operaciones Matemáticas

- Pandas permite aplicar operaciones aritméticas y funciones sobre los datos de una DataFrame

```
In [26]: print("\nAplicando operaciones matemáticas a una columna.")
df["Edad"] = df["Edad"] + 1
print(df)

print("\nAplicar una función a una columna.")
df["Categoria_Edad"] = df["Edad"].apply(lambda x: "Joven" if x < 30 else "Adulto")
print(df)
```

Aplicando operaciones matemáticas a una columna.

| | Nombre | Edad |
|---|--------|------|
| 0 | Ana | 26 |
| 1 | Pao | 31 |
| 2 | Lalo | 36 |
| 3 | Ana | 26 |

Aplicar una función a una columna.

| | Nombre | Edad | Categoria_Edad |
|---|--------|------|----------------|
| 0 | Ana | 26 | Joven |
| 1 | Pao | 31 | Adulto |
| 2 | Lalo | 36 | Adulto |
| 3 | Ana | 26 | Joven |

Aplicación de funciones personalizadas.

- Podemos definir nuestras propias funciones y aplicarlas a un DataFrame.

```
In [27]: print("\nConvertir nombres a mayusculas.")
print("-----")
df["Nombre"] = df["Nombre"].apply(lambda x: x.upper())
print(df)
```

Convertir nombres a mayusculas.

| | Nombre | Edad | Categoria_Edad |
|---|--------|------|----------------|
| 0 | ANA | 26 | Joven |
| 1 | PAO | 31 | Adulto |
| 2 | LALO | 36 | Adulto |
| 3 | ANA | 26 | Joven |

Agrupación y Agregación de Datos en Pandas.

- En este apartado aprenderemos a agrupar, resumir y analizar datos usando funciones como groupby(), agg() y operaciones estadísticas.
- El método groupby() permite agrupar datos basado en una o más columnas

```
In [31]: import pandas as pd

# Crear un DataFrame
datos = {
```

```

"Producto": ["Laptop", "Mouse", "Laptop", "Teclado", "Mouse"],
"Categoria": ["Electrónica", "Accesorios", "Electrónica", "Accesorios", "Accesorios"],
"Precio": [1000, 50, 1200, 80, 55]
}
print("\nDataFrame")
print(datos)

df = pd.DataFrame(datos)

# Agrupar por "Producto" y obtener el precio promedio
print("\nAgrupar por producto y obtener el precio promedio")
agrupado = df.groupby("Producto")["Precio"].mean()
print(agrupado)

```

DataFrame

```

{'Producto': ['Laptop', 'Mouse', 'Laptop', 'Teclado', 'Mouse'], 'Categoria': ['Electrónica', 'Accesorios', 'Electrónica', 'Accesorios', 'Accesorios'], 'Precio': [1000, 50, 1200, 80, 55]}

```

Agrupar por producto y obtener el precio promedio

```

Producto
Laptop      1100.0
Mouse        52.5
Teclado      80.0
Name: Precio, dtype: float64

```

In [29]: `import pandas as pd`

```

df = pd.DataFrame({
    "Departamento": ["Ventas", "Ventas", "IT", "IT", "RRHH", "RRHH", "IT"],
    "Empleado": ["Ana", "Luis", "Carlos", "María", "Sofía", "Pedro", "Elena"],
    "Salario": [5000, 4500, 6000, 5200, 4800, 4700, 6100]
})

# Agrupar por "Departamento" y calcular el salario promedio
df_grouped = df.groupby("Departamento")["Salario"].mean()
print(df_grouped)

```

Departamento

```

IT          5766.666667
RRHH        4750.000000
Ventas      4750.000000
Name: Salario, dtype: float64

```

In [32]: `print("\nAgrupar por multiples columnas")`
`agrupado = df.groupby(["Categoria", "Producto"])["Precio"].mean()`
`print(agrupado)`

Agrupar por multiples columnas

```

Categoria  Producto
Accesorios  Mouse      52.5
            Teclado    80.0
Electrónica Laptop    1100.0
Name: Precio, dtype: float64

```

Funciones de agregación (sum(), mean(), count(), agg())

- Podemos aplicar funciones de agregación para obtener estadísticas sobre los datos agrupados.

In [35]: `print("\nContar productos por categoría.")`
`conteo = df.groupby("Categoria")["Producto"].count()`
`print(conteo)`

```
print("\nAplicando multiples funciones con agg.")
resumen = df.groupby("Categoria")["Precio"].agg(["sum", "mean", "max", "min"])
print(resumen)
```

Contar productos por categoría.

Categoría

Accesorios 3

Electrónica 2

Name: Producto, dtype: int64

Aplicando multiples funciones con agg.

| | sum | mean | max | min |
|-------------|------|-------------|------|------|
| Categoría | | | | |
| Accesorios | 185 | 61.666667 | 80 | 50 |
| Electrónica | 2200 | 1100.000000 | 1200 | 1000 |

Agrupar y filtrar datos con filter()

```
In [30]: df_filtrado = df.groupby("Departamento").filter(lambda x: len(x) > 2)
print(df_filtrado)
```

| | Departamento | Empleado | Salario |
|---|--------------|----------|---------|
| 2 | IT | Carlos | 6000 |
| 3 | IT | María | 5200 |
| 6 | IT | Elena | 6100 |

Obtener estadísticos generales con describe()

- Podemos obtener un resumen estadístico por grupo.

```
In [31]: df_grouped = df.groupby("Departamento")["Salario"].describe()
print(df_grouped)
```

| | count | mean | std | min | 25% | 50% | 75% | \ |
|--------------|-------|-------------|------------|--------|--------|--------|--------|---|
| Departamento | | | | | | | | |
| IT | 3.0 | 5766.666667 | 493.288286 | 5200.0 | 5600.0 | 6000.0 | 6050.0 | |
| RRHH | 2.0 | 4750.000000 | 70.710678 | 4700.0 | 4725.0 | 4750.0 | 4775.0 | |
| Ventas | 2.0 | 4750.000000 | 353.553391 | 4500.0 | 4625.0 | 4750.0 | 4875.0 | |

| | max |
|--------------|--------|
| Departamento | |
| IT | 6100.0 |
| RRHH | 4800.0 |
| Ventas | 5000.0 |

Aplicación de transform() y apply() en Agrupaciones.

```
In [ ]: print("\nCalculo de porcentaje del precio respecto al total por categoria")
df["Total_Categoria"] = df.groupby("Categoria")["Precio"].transform("sum")
df["Porcentaje"] = (df["Precio"] / df["Total_Categoria"]) * 100
print(df)
```

Calculo de el porcentaje del precio respecto al total por categoria

| | Producto | Categoria | Precio | Total_Categoria | Porcentaje |
|---|----------|-------------|--------|-----------------|------------|
| 0 | Laptop | Electrónica | 1000 | 2200 | 45.454545 |
| 1 | Mouse | Accesorios | 50 | 185 | 27.027027 |
| 2 | Laptop | Electrónica | 1200 | 2200 | 54.545455 |
| 3 | Teclado | Accesorios | 80 | 185 | 43.243243 |
| 4 | Mouse | Accesorios | 55 | 185 | 29.729730 |

Combinación y fusión de DataFrames en Pandas

- En este módulo aprenderemos a unir, concatenar y fusionar DataFrames utilizando funciones como merge(), concat() y join()

Concatenación vertical (Apilado de Datos)

```
In [37]: import pandas as pd

# Crear dos DataFrames
df1 = pd.DataFrame({"ID": [1, 2, 3], "Nombre": ["Ana", "Luis", "Carlos"]})
df2 = pd.DataFrame({"ID": [4, 5], "Nombre": ["Marta", "Pedro"]})

# Concatenación vertical
df_concatenado = pd.concat([df1, df2], ignore_index=True)
print(df_concatenado)
```

| | ID | Nombre |
|---|----|--------|
| 0 | 1 | Ana |
| 1 | 2 | Luis |
| 2 | 3 | Carlos |
| 3 | 4 | Marta |
| 4 | 5 | Pedro |

Concatenación horizontal (Añadir columnas)

```
In [38]: df3 = pd.DataFrame({"ID": [1, 2, 3], "Edad": [25, 30, 35]})

# Concatenación horizontal
df_combinado = pd.concat([df1, df3], axis=1)
print(df_combinado)
```

| | ID | Nombre | ID | Edad |
|---|----|--------|----|------|
| 0 | 1 | Ana | 1 | 25 |
| 1 | 2 | Luis | 2 | 30 |
| 2 | 3 | Carlos | 3 | 35 |

Fusión de DataFrames con merge()

- La función merge() es más flexible y permite combinar DataFrames con en SQL(INNER JOIN, LEFT JOIN, RIGHT JOIN, OUTER JOIN).

```
In [39]: df1 = pd.DataFrame({"ID": [1, 2, 3], "Nombre": ["Ana", "Luis", "Carlos"]})
df2 = pd.DataFrame({"ID": [1, 3, 4], "Edad": [25, 35, 40]})

# Fusionar por la columna "ID"
df_merge = pd.merge(df1, df2, on="ID", how="inner")
print(df_merge)
```

| | ID | Nombre | Edad |
|---|----|--------|------|
| 0 | 1 | Ana | 25 |
| 1 | 3 | Carlos | 35 |

LEFT JOIN (Mantener todos los datos de la izquierda)

```
In [40]: df_merge = pd.merge(df1, df2, on="ID", how="left")
print(df_merge)
```

| | ID | Nombre | Edad |
|---|----|--------|------|
| 0 | 1 | Ana | 25.0 |
| 1 | 2 | Luis | NaN |
| 2 | 3 | Carlos | 35.0 |

Unión de DataFrames con join()

- El método `join()` se usa cuando queremos unir DataFrames basados en el índice.

```
In [41]: print("\nUnir los DataFrames por el índice.")
df1 = pd.DataFrame({"Nombre": ["Ana", "Luis", "Carlos"]}, index=[1, 2, 3])
df2 = pd.DataFrame({"Edad": [25, 30, 35]}, index=[1, 2, 3])

df_unido = df1.join(df2)
print(df_unido)
```

Unir los DataFrames por el índice.

| | Nombre | Edad |
|---|--------|------|
| 1 | Ana | 25 |
| 2 | Luis | 30 |
| 3 | Carlos | 35 |

RIGHT JOIN (Mantener todos los datos de la derecha)

- Este tipo de fusión mantiene todas las filas de la derecha, aunque algunas no tengan coincidencia en la izquierda.

```
In [42]: df1 = pd.DataFrame({"ID": [1, 2, 3], "Nombre": ["Ana", "Luis", "Carlos"]})
df2 = pd.DataFrame({"ID": [1, 3, 4], "Edad": [25, 35, 40]})

df_merge = pd.merge(df1, df2, on="ID", how="right")
print(df_merge)
```

| | ID | Nombre | Edad |
|---|----|--------|------|
| 0 | 1 | Ana | 25 |
| 1 | 3 | Carlos | 35 |
| 2 | 4 | NaN | 40 |

OUTER JOIN (Unión completa de los datos)

- Aquí se combinan todos los datos de ambos DataFrames, incluso si no tienen coincidencias.

```
In [43]: df_merge = pd.merge(df1, df2, on="ID", how="outer")
print(df_merge)
```

| | ID | Nombre | Edad |
|---|----|--------|------|
| 0 | 1 | Ana | 25.0 |
| 1 | 2 | Luis | NaN |
| 2 | 3 | Carlos | 35.0 |
| 3 | 4 | NaN | 40.0 |

Concatenar DataFrames con diferentes columnas

- Si los DataFrames tienen columnas distintas `concat()` rellena con NaN los valores faltantes.

```
In [44]: df1 = pd.DataFrame({"ID": [1, 2], "Nombre": ["Ana", "Luis"]})
df2 = pd.DataFrame({"ID": [3, 4], "Edad": [25, 30]})

df_concatenado = pd.concat([df1, df2], ignore_index=True)
print(df_concatenado)
```

| | ID | Nombre | Edad |
|---|----|--------|------|
| 0 | 1 | Ana | NaN |
| 1 | 2 | Luis | NaN |
| 2 | 3 | NaN | 25.0 |
| 3 | 4 | NaN | 30.0 |

Fusionar con diferentes nombres de columnas (left_on, right_on)

- Podemos unir DataFrames que no tienen el mismo nombre de columna para la clave

```
In [45]: df_clientes = pd.DataFrame({"ID_Cliente": [1, 2, 3], "Nombre": ["Ana", "Luis", "Carlos"], "Edad": [25, 35, 40]})
df_compras = pd.DataFrame({"ID": [1, 3, 4], "Producto": ["Laptop", "Mouse", "Teclado"]})

df_merge = pd.merge(df_clientes, df_compras, left_on="ID_Cliente", right_on="ID", how="left")
print(df_merge)
```

| | ID_Cliente | Nombre | ID | Producto |
|---|------------|--------|-----|----------|
| 0 | 1 | Ana | 1.0 | Laptop |
| 1 | 2 | Luis | NaN | NaN |
| 2 | 3 | Carlos | 3.0 | Mouse |

Unir multiples DataFrames con reduce()

- Cuando necesitamos unir varios DataFrames, podemos hacerlo con reduce()

```
In [46]: from functools import reduce

df1 = pd.DataFrame({"ID": [1, 2, 3], "Nombre": ["Ana", "Luis", "Carlos"]})
df2 = pd.DataFrame({"ID": [1, 3, 4], "Edad": [25, 35, 40]})
df3 = pd.DataFrame({"ID": [1, 2, 4], "Salario": [2000, 2500, 3000]})

dfs = [df1, df2, df3]

df_final = reduce(lambda left, right: pd.merge(left, right, on="ID", how="outer"), dfs)
print(df_final)
```

| | ID | Nombre | Edad | Salario |
|---|----|--------|------|---------|
| 0 | 1 | Ana | 25.0 | 2000.0 |
| 1 | 2 | Luis | NaN | 2500.0 |
| 2 | 3 | Carlos | 35.0 | NaN |
| 3 | 4 | NaN | 40.0 | 3000.0 |

Manejo de Datos faltantes en Pandas

- En este módulo aprenderemos a manejar valores NaN (valores nulos o faltantes) en un DataFrame.
- Al trabajar con datos reales, es común encontrar valores ausentes, por lo que es importante saber cómo detectarlos, eliminarlos o reemplazarlos.

Detección de Datos faltantes.

- Para identificar valores faltantes en un DataFrame, usamos isna() o isnull() (ambas hacen lo mismo)

Identificar valores nulos en un DataFrame

- isna() devuelve un DataFrame de True/false indicando dónde hay valores nulos.
- sum() cuenta la cantidad de valores nulos en cada columna.

```
In [2]: import pandas as pd
import numpy as np

# Crear un DataFrame con valores faltantes
df = pd.DataFrame({
    "Nombre": ["Ana", "Luis", "Carlos", np.nan, "Marta"],
    "Edad": [25, np.nan, 35, 40, np.nan],
    "Ciudad": ["Mexico", "Canada", "EU", "Panama", np.nan]
})

# Mostrar el DataFrame
print(df)

# Detectar valores nulos
print(df.isna())

# Contar valores nulos por columna
print(df.isna().sum())
```

| | Nombre | Edad | Ciudad |
|---|--------|------|--------|
| 0 | Ana | 25.0 | Mexico |
| 1 | Luis | NaN | Canada |
| 2 | Carlos | 35.0 | EU |
| 3 | NaN | 40.0 | Panama |
| 4 | Marta | NaN | NaN |

| | Nombre | Edad | Ciudad |
|---|--------|-------|--------|
| 0 | False | False | False |
| 1 | False | True | False |
| 2 | False | False | False |
| 3 | True | False | False |
| 4 | False | True | True |

| | |
|--------|---|
| Nombre | 1 |
| Edad | 2 |
| Ciudad | 1 |

dtype: int64

Eliminación de datos faltantes (dropna())

- Podemos eliminar filas o columnas con datos nulos usando dropna()
- Eliminar filas con valores nulos.

```
In [3]: df_sin_nulos = df.dropna()
print(df_sin_nulos)
```

| | Nombre | Edad | Ciudad |
|---|--------|------|--------|
| 0 | Ana | 25.0 | Mexico |
| 2 | Carlos | 35.0 | EU |

Eliminar columnas con valores nulos.

```
In [4]: df_sin_columnas_nulas = df.dropna(axis=1)
print(df_sin_columnas_nulas)
```

Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4]

Relleno de datos faltantes (fillna())

- En lugar de eliminar datos, podemos reemplazar valores nulos con fillna()
- Rellenar valores nulos con un valor específico.

```
In [5]: df_rellenado = df.fillna("Desconocido")
print(df_rellenado)
```

| | Nombre | Edad | Ciudad |
|---|-------------|-------------|-------------|
| 0 | Ana | 25.0 | Mexico |
| 1 | Luis | Desconocido | Canada |
| 2 | Carlos | 35.0 | EU |
| 3 | Desconocido | 40.0 | Panama |
| 4 | Marta | Desconocido | Desconocido |

Rellenar valores nulos con la media de la columna.

```
In [6]: media_edad = df["Edad"].mean()
df["Edad"] = df["Edad"].fillna(media_edad)
print(df)
```

| | Nombre | Edad | Ciudad |
|---|--------|-----------|--------|
| 0 | Ana | 25.000000 | Mexico |
| 1 | Luis | 33.333333 | Canada |
| 2 | Carlos | 35.000000 | EU |
| 3 | NaN | 40.000000 | Panama |
| 4 | Marta | 33.333333 | NaN |

Interpolación de datos.

- Si los datos siguen una tendencia, interpolate() puede estimar valores faltantes.
- Interpolación de valores numéricos.

```
In [7]: df_numerico = pd.DataFrame({"Año": [2000, 2001, 2002, 2003, 2004],
                                   "Ventas": [100, np.nan, 200, np.nan, 400]})

df_interpolado = df_numerico.interpolate()
print(df_interpolado)
```

| | Año | Ventas |
|---|------|--------|
| 0 | 2000 | 100.0 |
| 1 | 2001 | 150.0 |
| 2 | 2002 | 200.0 |
| 3 | 2003 | 300.0 |
| 4 | 2004 | 400.0 |

Manejo de fechas y tiempos en Pandas

- El manejo de fechas es crucial en análisis de datos, especialmente en series temporales.
- Pandas proporciona herramientas poderosas para trabajar con fechas, como pd.to_datetime(), dt y resample().

Convertir fechas con pd.to_datetime()

```
In [22]: import pandas as pd

df = pd.DataFrame({
    "fecha" : ["2025-01-01", "2025-02-01", "2025-03-01"],
    "ventas": [100, 150, 200]
})

df["fecha"] = pd.to_datetime(df["fecha"])
print(df.dtypes)
print("Ahora fecha es de tipo datetime64, lo que permite hacer operaciones avanzadas")
```


fecha datetime64[ns]
ventas int64
dtype: object
Ahora fecha es de tipo datetime64, lo que permite hacer operaciones avanzadas

Extracción de componentes dde fechas.(dt)

- Podemos extraer año, mes, día, etc.

```
In [12]: df["Año"] = df["fecha"].dt.year  
df["Mes"] = df["fecha"].dt.month  
df["dia"] = df["fecha"].dt.day  
df["Dia_Semana"] = df["fecha"].dt.day_name()  
print(df)
```

| | fecha | ventas | Año | Mes | dia | Dia_Semana |
|---|------------|--------|------|-----|-----|------------|
| 0 | 2025-01-01 | 100 | 2025 | 1 | 1 | Wednesday |
| 1 | 2025-02-01 | 150 | 2025 | 2 | 1 | Saturday |
| 2 | 2025-03-01 | 200 | 2025 | 3 | 1 | Saturday |

Creación de rango de fechas (pd.date_range())

- Si necesitamos generar una serie de fechas pd.date_range() nos ayuda.
- Generar un rango de fechas diaria.

```
In [13]: rango_fechas = pd.date_range(start="2025-02-02", end="2025-03-03", freq="D")  
print(rango_fechas)
```

```
DatetimeIndex(['2025-02-02', '2025-02-03', '2025-02-04', '2025-02-05',  
               '2025-02-06', '2025-02-07', '2025-02-08', '2025-02-09',  
               '2025-02-10', '2025-02-11', '2025-02-12', '2025-02-13',  
               '2025-02-14', '2025-02-15', '2025-02-16', '2025-02-17',  
               '2025-02-18', '2025-02-19', '2025-02-20', '2025-02-21',  
               '2025-02-22', '2025-02-23', '2025-02-24', '2025-02-25',  
               '2025-02-26', '2025-02-27', '2025-02-28', '2025-03-01',  
               '2025-03-02', '2025-03-03'],  
              dtype='datetime64[ns]', freq='D')
```

Operaciones con fechas.

- Podemos calcular diferencias, agregar días y más

```
In [14]: df["Dias desde inicio"] = (df["fecha"] - df["fecha"].min()).dt.days  
print(df)
```

| | fecha | ventas | Año | Mes | dia | Dia_Semana | Dias desde inicio |
|---|------------|--------|------|-----|-----|------------|-------------------|
| 0 | 2025-01-01 | 100 | 2025 | 1 | 1 | Wednesday | 0 |
| 1 | 2025-02-01 | 150 | 2025 | 2 | 1 | Saturday | 31 |
| 2 | 2025-03-01 | 200 | 2025 | 3 | 1 | Saturday | 59 |

Re-muestreo de datos temporales (resample())

- Podemos agrupar datos por días, semanas o meses con resample()

```
In [23]: df.set_index("fecha", inplace=True)  
df_mensual = df.resample("M").sum()  
print(df_mensual)
```

| fecha | ventas |
|------------|--------|
| 2025-01-31 | 100 |
| 2025-02-28 | 150 |
| 2025-03-31 | 200 |

C:\Users\Lalo\AppData\Local\Temp\ipykernel_20152\2306358148.py:2: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME' instead.

```
df_mensual = df.resample("M").sum()
```

Filtrar datos por fechas.

- Podemos seleccionar datos en un rango de fechas.

```
In [24]: df = pd.DataFrame({
    "Fecha": pd.date_range(start="2024-01-01", periods=10, freq="D"),
    "Ventas": [100, 150, 120, 180, 200, 140, 160, 180, 220, 190]
})

# Filtrar datos entre el 3 y el 7 de enero
filtro = (df["Fecha"] >= "2024-01-03") & (df["Fecha"] <= "2024-01-07")
df_filtrado = df[filtro]
print(df_filtrado)
```

| | Fecha | Ventas |
|---|------------|--------|
| 2 | 2024-01-03 | 120 |
| 3 | 2024-01-04 | 180 |
| 4 | 2024-01-05 | 200 |
| 5 | 2024-01-06 | 140 |
| 6 | 2024-01-07 | 160 |

Reemplazar el formato fecha.

- Podemos cambiar el formato de fechas con .dt.strftime()

%d/%m/%y da formato día/mes/año.

%Y da formato año completo.

%Y-%m-%d da formato 2025-02-02.

%B %d, %y da formato Febrero 13, 2025.

%A Monday es día de la semana.

```
In [25]: df["Fecha Formateada"] = df["Fecha"].dt.strftime("%d/%m/%Y")
print(df[["Fecha", "Fecha Formateada"]])
```

| | Fecha | Fecha Formateada |
|---|------------|------------------|
| 0 | 2024-01-01 | 01/01/2024 |
| 1 | 2024-01-02 | 02/01/2024 |
| 2 | 2024-01-03 | 03/01/2024 |
| 3 | 2024-01-04 | 04/01/2024 |
| 4 | 2024-01-05 | 05/01/2024 |
| 5 | 2024-01-06 | 06/01/2024 |
| 6 | 2024-01-07 | 07/01/2024 |
| 7 | 2024-01-08 | 08/01/2024 |
| 8 | 2024-01-09 | 09/01/2024 |
| 9 | 2024-01-10 | 10/01/2024 |

Análisis exploratorio de datos en Pandas (EDA)

- El análisis exploratorio de datos (EDA, Exploratory Data Analysis) es un proceso clave en ciencia de datos. Permite comprender, limpiar y visualizar la información antes de aplicar modelos o tomar decisiones.
- Pandas ofrece muchas herramientas para realizar EDA de manera eficiente.

Cargar datos y obtener información general.

- Antes de analizar datos, necesitamos cargarlos y revisar su estructura.

Cargar un dataset y obtener información básica.

- Este dataset que vamos a cargar para hacer EDA lo tenemos de forma local y fue descargado desde Kaggle
- link <https://www.kaggle.com/datasets/sameerk2004/space-missions-dataset>
- Pueden hacer uso de cualquier dataset que sea de su agrado.

```
In [33]: import pandas as pd

# Cargar un dataset desde un archivo CSV
df = pd.read_csv("space_missions_dataset.csv")

# Información general sobre las columnas y tipos de datos
print("Información General sobre el dataset")
print(df.info())
```

Información General sobre el dataset

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 500 entries, 0 to 499

Data columns (total 15 columns):

| # | Column | Non-Null Count | Dtype |
|----|-----------------------------------|----------------|---------|
| 0 | Mission ID | 500 non-null | object |
| 1 | Mission Name | 500 non-null | object |
| 2 | Launch Date | 500 non-null | object |
| 3 | Target Type | 500 non-null | object |
| 4 | Target Name | 500 non-null | object |
| 5 | Mission Type | 500 non-null | object |
| 6 | Distance from Earth (light-years) | 500 non-null | float64 |
| 7 | Mission Duration (years) | 500 non-null | float64 |
| 8 | Mission Cost (billion USD) | 500 non-null | float64 |
| 9 | Scientific Yield (points) | 500 non-null | float64 |
| 10 | Crew Size | 500 non-null | int64 |
| 11 | Mission Success (%) | 500 non-null | float64 |
| 12 | Fuel Consumption (tons) | 500 non-null | float64 |
| 13 | Payload Weight (tons) | 500 non-null | float64 |
| 14 | Launch Vehicle | 500 non-null | object |

dtypes: float64(7), int64(1), object(7)

memory usage: 58.7+ KB

None

Inspección de los datos.

```
In [36]: print("\nVamos a mostrar los primeros 5 filas")
df.head()
```

Vamos a mostrar los primeros 5 filas

Out[36]:

| | Mission ID | Mission Name | Launch Date | Target Type | Target Name | Mission Type | Distance from Earth (light-years) | Mission Duration (years) | Mission Cost (billion USD) | Sci (p |
|---|------------|--------------|-------------|-------------|-------------|--------------|-----------------------------------|--------------------------|----------------------------|--------|
| 0 | MSN-0001 | Mission-1 | 2025-01-01 | Star | Titan | Colonization | 7.05 | 5.2 | 526.68 | |
| 1 | MSN-0002 | Mission-2 | 2025-01-08 | Exoplanet | Betelgeuse | Colonization | 41.76 | 23.0 | 234.08 | |
| 2 | MSN-0003 | Mission-3 | 2025-01-15 | Asteroid | Mars | Exploration | 49.22 | 28.8 | 218.68 | |
| 3 | MSN-0004 | Mission-4 | 2025-01-22 | Exoplanet | Titan | Colonization | 26.33 | 17.8 | 232.89 | |
| 4 | MSN-0005 | Mission-5 | 2025-01-29 | Exoplanet | Proxima b | Mining | 8.67 | 9.2 | 72.14 | |

In [37]: `print("\nVamos a mostrar las ultimas 5 filas")
df.tail()`

Vamos a mostrar las ultimas 5 filas

Out[37]:

| | Mission ID | Mission Name | Launch Date | Target Type | Target Name | Mission Type | Distance from Earth (light-years) | Mission Duration (years) | Mission Cost (billion USD) | Sci (p |
|-----|------------|--------------|-------------|-------------|-------------|--------------|-----------------------------------|--------------------------|----------------------------|--------|
| 495 | MSN-0496 | Mission-496 | 2034-06-28 | Planet | Betelgeuse | Colonization | 48.26 | 25.3 | 274.82 | |
| 496 | MSN-0497 | Mission-497 | 2034-07-05 | Asteroid | Betelgeuse | Exploration | 0.35 | 4.1 | 379.71 | |
| 497 | MSN-0498 | Mission-498 | 2034-07-12 | Planet | Ceres | Exploration | 47.60 | 26.6 | 296.45 | |
| 498 | MSN-0499 | Mission-499 | 2034-07-19 | Planet | Betelgeuse | Research | 31.99 | 18.0 | 457.38 | |
| 499 | MSN-0500 | Mission-500 | 2034-07-26 | Planet | Io | Exploration | 43.41 | 23.5 | 449.40 | |

In [38]: `print("\nVamos a mostrar 5 filas de forma aleatorias")
df.sample(5)`

Vamos a mostrar 5 filas de forma aleatorias

Out[38]:

| | Mission ID | Mission Name | Launch Date | Target Type | Target Name | Mission Type | Distance from Earth (light-years) | Mission Duration (years) | Mission Cost (billion USD) | Scientific Yield (points) |
|-----|------------|--------------|-------------|-------------|-------------|--------------|-----------------------------------|--------------------------|----------------------------|---------------------------|
| 361 | MSN-0362 | Mission-362 | 2031-12-03 | Star | Proxima b | Research | 6.72 | 7.9 | 479.66 | |
| 474 | MSN-0475 | Mission-475 | 2034-02-01 | Asteroid | Titan | Mining | 33.22 | 20.1 | 81.24 | |
| 46 | MSN-0047 | Mission-47 | 2025-11-19 | Moon | Io | Colonization | 29.79 | 17.2 | 426.74 | |
| 220 | MSN-0221 | Mission-221 | 2029-03-21 | Moon | Proxima b | Mining | 35.45 | 22.5 | 228.78 | |
| 110 | MSN-0111 | Mission-111 | 2027-02-10 | Planet | Mars | Exploration | 27.02 | 17.0 | 47.56 | |

In [39]: `print("\nVamos a ver la cantidad de columnas")`
`df.shape`

Out[39]: Vamos a ver la cantidad de columnas
(500, 15)

In [40]: `print("\nListar nombres de las columnas")`
`df.columns`

Out[40]: Listar nombres de las columnas
Index(['Mission ID', 'Mission Name', 'Launch Date', 'Target Type',
 'Target Name', 'Mission Type', 'Distance from Earth (light-years)',
 'Mission Duration (years)', 'Mission Cost (billion USD)',
 'Scientific Yield (points)', 'Crew Size', 'Mission Success (%)',
 'Fuel Consumption (tons)', 'Payload Weight (tons)', 'Launch Vehicle'],
 dtype='object')

In [41]: `print("\nVer indices del dataframe")`
`df.index`

Out[41]: Ver indices del dataframe
RangeIndex(start=0, stop=500, step=1)

In [42]: `print("\nListar tipos de datos de las columnas")`
`df.dtypes`

Listar tipos de datos de las columnas

```
Out[42]: Mission ID          object
Mission Name        object
Launch Date         object
Target Type         object
Target Name         object
Mission Type        object
Distance from Earth (light-years) float64
Mission Duration (years) float64
Mission Cost (billion USD) float64
Scientific Yield (points) float64
Crew Size           int64
Mission Success (%) float64
Fuel Consumption (tons) float64
Payload Weight (tons) float64
Launch Vehicle      object
dtype: object
```

```
In [47]: print("\nInformación general sobre el dataframe")
df.info()
```

```
Información general sobre el dataframe
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 15 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Mission ID                           500 non-null   object
1   Mission Name                         500 non-null   object
2   Launch Date                         500 non-null   object
3   Target Type                         500 non-null   object
4   Target Name                         500 non-null   object
5   Mission Type                        500 non-null   object
6   Distance from Earth (light-years)    500 non-null   float64
7   Mission Duration (years)            500 non-null   float64
8   Mission Cost (billion USD)          500 non-null   float64
9   Scientific Yield (points)            500 non-null   float64
10  Crew Size                           500 non-null   int64
11  Mission Success (%)                 500 non-null   float64
12  Fuel Consumption (tons)             500 non-null   float64
13  Payload Weight (tons)               500 non-null   float64
14  Launch Vehicle                     500 non-null   object
dtypes: float64(7), int64(1), object(7)
memory usage: 58.7+ KB
```

```
In [ ]: print("\nEstadísticas basics de columnas numéricas")
df.describe()
```

```
Estadísticas basics de columnas numericas
```

Out[]:

| | Distance from Earth (light- years) | Mission Duration (years) | Mission Cost (billion USD) | Scientific Yield (points) | Crew Size | Mission Success (%) | Fuel Consumption (tons) | P |
|-------|---|--------------------------------|-------------------------------------|---------------------------------|------------|---------------------------|-------------------------------|------|
| count | 500.000000 | 500.000000 | 500.000000 | 500.000000 | 500.000000 | 500.000000 | 500.000000 | 500. |
| mean | 25.483060 | 15.736800 | 277.300280 | 55.223400 | 50.118000 | 92.616600 | 2543.522140 | 50. |
| std | 14.942128 | 7.578316 | 141.137422 | 26.446232 | 27.660989 | 9.391094 | 1492.964489 | 28. |
| min | 0.350000 | 1.400000 | 13.320000 | 10.000000 | 1.000000 | 66.000000 | 18.060000 | 1. |
| 25% | 11.750000 | 8.900000 | 149.960000 | 33.775000 | 27.000000 | 85.500000 | 1177.315000 | 25. |
| 50% | 26.185000 | 16.400000 | 282.170000 | 54.400000 | 50.000000 | 98.600000 | 2597.985000 | 50. |
| 75% | 38.570000 | 22.200000 | 399.995000 | 79.025000 | 74.000000 | 100.000000 | 3859.355000 | 74. |
| max | 49.900000 | 29.500000 | 538.320000 | 99.800000 | 99.000000 | 100.000000 | 5018.600000 | 99. |

```
In [ ]: print("\nEstadísticas de todas las columnas")
df.describe(include="all")
```

Estadísticas de todas las columnas

Out[]:

| | Mission ID | Mission Name | Launch Date | Target Type | Target Name | Mission Type | Distance from Earth (light- years) | Mission Duration (years) | Mission Cost (billion USD) |
|--------|---------------|-----------------|----------------|----------------|----------------|-----------------|---|--------------------------------|-------------------------------------|
| count | 500 | 500 | 500 | 500 | 500 | 500 | 500.000000 | 500.000000 | 500.000000 |
| unique | 500 | 500 | 500 | 5 | 7 | 4 | NaN | NaN | NaN |
| top | MSN-0001 | Mission-1 | 2025-01-01 | Star | Proxima b | Research | NaN | NaN | NaN |
| freq | 1 | 1 | 1 | 112 | 81 | 132 | NaN | NaN | NaN |
| mean | NaN | NaN | NaN | NaN | NaN | NaN | 25.483060 | 15.736800 | 277.300280 |
| std | NaN | NaN | NaN | NaN | NaN | NaN | 14.942128 | 7.578316 | 141.137422 |
| min | NaN | NaN | NaN | NaN | NaN | NaN | 0.350000 | 1.400000 | 13.320000 |
| 25% | NaN | NaN | NaN | NaN | NaN | NaN | 11.750000 | 8.900000 | 149.960000 |
| 50% | NaN | NaN | NaN | NaN | NaN | NaN | 26.185000 | 16.400000 | 282.170000 |
| 75% | NaN | NaN | NaN | NaN | NaN | NaN | 38.570000 | 22.200000 | 399.995000 |
| max | NaN | NaN | NaN | NaN | NaN | NaN | 49.900000 | 29.500000 | 538.320000 |

```
In [46]: print("\nVer el consumo de la memoria")
df.memory_usage
```

Ver el consumo de la memoria

```

Out[46]: <bound method DataFrame.memory_usage of
rget Type Target Name Mission Type \
0 MSN-0001 Mission-1 2025-01-01 Star Titan Colonization
1 MSN-0002 Mission-2 2025-01-08 Exoplanet Betelgeuse Colonization
2 MSN-0003 Mission-3 2025-01-15 Asteroid Mars Exploration
3 MSN-0004 Mission-4 2025-01-22 Exoplanet Titan Colonization
4 MSN-0005 Mission-5 2025-01-29 Exoplanet Proxima b Mining
.. ...
495 MSN-0496 Mission-496 2034-06-28 Planet Betelgeuse Colonization
496 MSN-0497 Mission-497 2034-07-05 Asteroid Betelgeuse Exploration
497 MSN-0498 Mission-498 2034-07-12 Planet Ceres Exploration
498 MSN-0499 Mission-499 2034-07-19 Planet Betelgeuse Research
499 MSN-0500 Mission-500 2034-07-26 Planet Io Exploration

Distance from Earth (light-years) Mission Duration (years) \
0 7.05 5.2
1 41.76 23.0
2 49.22 28.8
3 26.33 17.8
4 8.67 9.2
.. ...
495 48.26 25.3
496 0.35 4.1
497 47.60 26.6
498 31.99 18.0
499 43.41 23.5

Mission Cost (billion USD) Scientific Yield (points) Crew Size \
0 526.68 64.3 21
1 234.08 84.4 72
2 218.68 98.6 16
3 232.89 36.0 59
4 72.14 96.5 31
.. ...
495 274.82 91.2 64
496 379.71 82.6 61
497 296.45 98.6 29
498 457.38 77.9 39
499 449.40 45.4 88

Mission Success (%) Fuel Consumption (tons) Payload Weight (tons) \
0 100.0 731.88 99.78
1 89.6 4197.41 45.72
2 98.6 4908.00 36.12
3 90.0 2569.05 40.67
4 73.2 892.76 12.40
.. ...
495 96.2 4837.16 44.97
496 100.0 62.59 67.88
497 100.0 4794.01 51.38
498 100.0 3102.16 86.28
499 100.0 4302.93 86.74

Launch Vehicle
0 SLS
1 Starship
2 Starship
3 Starship
4 Starship
.. ...
495 Ariane 6
496 SLS
497 Falcon Heavy
498 SLS

```


[500 rows x 15 columns]>

Manejo de valores nulos

```
In [50]: print("\nSe muestra el numero de valores nulos en cada columna.")
df.isnull().sum()
```

```
Out[50]: Se muestra el numero de valores nulos en cada columna.
Mission ID                0
Mission Name              0
Launch Date              0
Target Type              0
Target Name              0
Mission Type             0
Distance from Earth (light-years)  0
Mission Duration (years)  0
Mission Cost (billion USD)  0
Scientific Yield (points)  0
Crew Size                0
Mission Success (%)      0
Fuel Consumption (tons)  0
Payload Weight (tons)    0
Launch Vehicle           0
dtype: int64
```

-Podemos eliminar calores nulos. pero como podemos ver este dataframe esta completo asi que solo comparto las instrucciones en caso de que un data si tenga valores nulos.

- `df_sin_nulos = df.dropna()`
- `df_sin_columnas_nulas = df.dropna(axis=1)`

Estas lineas son para rellenar valores faltantes.

- `df["Target Type"].fillna(df["Target Type"].mean(), inplace=True)` # Rellenar con la media
- `df["Mission Type"].fillna("Desconocido", inplace=True)` # Rellenar con un valor fijo

Analisis de frecuencias.

- Podemos contar cuantas veces aparece cada valor en una columna categorica.

```
In [51]: print(df["Mission Name"].value_counts())
```

```
Mission Name
Mission-1      1
Mission-330    1
Mission-343    1
Mission-342    1
Mission-341    1
..
Mission-162    1
Mission-161    1
Mission-160    1
Mission-159    1
Mission-500    1
Name: count, Length: 500, dtype: int64
```

```
In [52]: print(df["Mission Duration (years)"].value_counts())
```

```

Mission Duration (years)
26.6    7
19.0    6
25.8    6
8.9     6
26.1    6
..
8.1     1
29.5    1
18.8    1
12.6    1
17.3    1
Name: count, Length: 231, dtype: int64

```

Instrucciones completas para trabajar con valores nulos.

- `df.isnull().sum()` # Contar valores nulos en cada columna.
- `df.notnull().sum()` # Contar valores NO nulos.
- `df.dropna()` # Eliminar filas con valores nulos.
- `df.dropna(axis=1)` # Eliminar columnas con valores nulos.
- `df.fillna(0)` # Reemplazar valores nulos con 0.
- `df.fillna(df.mean())` # Rellenar con la media.
- `df.fillna(df.median())` # Rellenar con la mediana.
- `df.fillna(method="ffill")` # Rellenar con el valor anterior.
- `df.fillna(method="bfill")` # Rellenar con el siguiente valor.
- `df.interpolate()` # Rellenar valores nulos interpolando.

Buscaremos un dataframe que tenga valores nulos y así poder probar esta lista de instrucciones.

Duplicados

```

In [55]: print("\nConteo de filas duplicadas")
df.duplicated().sum()

print("\nEliminar duplicados")
df.duplicated().sum()

print("\nEliminar duplicados en una columna")
df.drop_duplicates(subset=["Target Name"])

```

Conteo de filas duplicadas

Eliminar duplicados

Eliminar duplicados en una columna

Out[55]:

| | Mission ID | Mission Name | Launch Date | Target Type | Target Name | Mission Type | Distance from Earth (light-years) | Mission Duration (years) | Mission Cost (billion USD) | Scientific Yield (points) |
|----|------------|--------------|-------------|-------------|-------------|--------------|-----------------------------------|--------------------------|----------------------------|---------------------------|
| 0 | MSN-0001 | Mission-1 | 2025-01-01 | Star | Titan | Colonization | 7.05 | 5.2 | 526.68 | 10 |
| 1 | MSN-0002 | Mission-2 | 2025-01-08 | Exoplanet | Betelgeuse | Colonization | 41.76 | 23.0 | 234.08 | 15 |
| 2 | MSN-0003 | Mission-3 | 2025-01-15 | Asteroid | Mars | Exploration | 49.22 | 28.8 | 218.68 | 12 |
| 4 | MSN-0005 | Mission-5 | 2025-01-29 | Exoplanet | Proxima b | Mining | 8.67 | 9.2 | 72.14 | 8 |
| 5 | MSN-0006 | Mission-6 | 2025-02-05 | Moon | Ceres | Colonization | 13.69 | 8.8 | 452.42 | 18 |
| 15 | MSN-0016 | Mission-16 | 2025-04-16 | Moon | Io | Exploration | 0.65 | 3.1 | 495.63 | 20 |
| 21 | MSN-0022 | Mission-22 | 2025-05-28 | Exoplanet | Europa | Mining | 33.29 | 18.9 | 201.81 | 14 |

In [91]: `df = pd.read_csv("space_missions_dataset.csv")`

Filtrado de datos.

In [92]: `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 15 columns):
 #   Column                                Non-Null Count  Dtype
---  -
 0   Mission ID                           500 non-null   object
 1   Mission Name                         500 non-null   object
 2   Launch Date                          500 non-null   object
 3   Target Type                          500 non-null   object
 4   Target Name                          500 non-null   object
 5   Mission Type                         500 non-null   object
 6   Distance from Earth (light-years)    500 non-null   float64
 7   Mission Duration (years)             500 non-null   float64
 8   Mission Cost (billion USD)           500 non-null   float64
 9   Scientific Yield (points)             500 non-null   float64
10   Crew Size                            500 non-null   int64
11   Mission Success (%)                  500 non-null   float64
12   Fuel Consumption (tons)              500 non-null   float64
13   Payload Weight (tons)                500 non-null   float64
14   Launch Vehicle                       500 non-null   object
dtypes: float64(7), int64(1), object(7)
memory usage: 58.7+ KB
```

In [93]: `print("\nFiltrar valores mayores a 100")
df[df["Crew Size"] > 50]`

Filtrar valores mayores a 100

Out[93]:

| | Mission ID | Mission Name | Launch Date | Target Type | Target Name | Mission Type | Distance from Earth (light-years) | Mission Duration (years) | Mission Cost (billion USD) |
|-----|------------|--------------|-------------|-------------|-------------|--------------|-----------------------------------|--------------------------|----------------------------|
| 1 | MSN-0002 | Mission-2 | 2025-01-08 | Exoplanet | Betelgeuse | Colonization | 41.76 | 23.0 | 234.08 |
| 3 | MSN-0004 | Mission-4 | 2025-01-22 | Exoplanet | Titan | Colonization | 26.33 | 17.8 | 232.89 |
| 6 | MSN-0007 | Mission-7 | 2025-02-12 | Asteroid | Ceres | Research | 1.02 | 5.0 | 220.38 |
| 8 | MSN-0009 | Mission-9 | 2025-02-26 | Asteroid | Betelgeuse | Exploration | 5.98 | 4.4 | 361.35 |
| 11 | MSN-0012 | Mission-12 | 2025-03-19 | Asteroid | Mars | Research | 27.75 | 15.5 | 38.80 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 490 | MSN-0491 | Mission-491 | 2034-05-24 | Planet | Proxima b | Research | 15.88 | 10.0 | 112.99 |
| 492 | MSN-0493 | Mission-493 | 2034-06-07 | Star | Io | Colonization | 2.17 | 3.4 | 202.50 |
| 495 | MSN-0496 | Mission-496 | 2034-06-28 | Planet | Betelgeuse | Colonization | 48.26 | 25.3 | 274.82 |
| 496 | MSN-0497 | Mission-497 | 2034-07-05 | Asteroid | Betelgeuse | Exploration | 0.35 | 4.1 | 379.71 |
| 499 | MSN-0500 | Mission-500 | 2034-07-26 | Planet | Io | Exploration | 43.41 | 23.5 | 449.40 |

249 rows × 15 columns

In [100...]

```
print("\nFiltrar por misiones seleccionadas.")
misiones_seleccionadas = df[["Mission Name", "Launch Date"]]
print(misiones_seleccionadas)
```

| | Mission Name | Launch Date |
|-----|--------------|-------------|
| 0 | Mission-1 | 2025-01-01 |
| 1 | Mission-2 | 2025-01-08 |
| 2 | Mission-3 | 2025-01-15 |
| 3 | Mission-4 | 2025-01-22 |
| 4 | Mission-5 | 2025-01-29 |
| .. | ... | ... |
| 495 | Mission-496 | 2034-06-28 |
| 496 | Mission-497 | 2034-07-05 |
| 497 | Mission-498 | 2034-07-12 |
| 498 | Mission-499 | 2034-07-19 |
| 499 | Mission-500 | 2034-07-26 |

[500 rows x 2 columns]

In [130...]

```
print("\nFiltrado por misiones Exitosas")
misiones_exitosas = df[df["Mission Name"] == "Mission Success (%)"]
print("\n",misiones_exitosas)
```

Filtrado por misiones Exitosas

Empty DataFrame

Columns: [Mission ID, Mission Name, Launch Date, Target Type, Target Name, Mission Type, Distance from Earth (light-years), Mission Duration (years), Mission Cost (billion USD), Scientific Yield (points), Crew Size, Mission Success (%), Fuel Consumption (tons), Payload Weight (tons), Launch Vehicle]
Index: []

In [105...

```
print("\nFiltrar por misiones lanzadas despues del 2000")
df["Launch Date"] = pd.to_datetime(df["Launch Date"])
misiones_post_2000 = df[df["Launch Date"].dt.year > 2000]
print(misiones_post_2000)
```

Filtrar por misiones lanzadas despues del 2000

| | Mission ID | Mission Name | Launch Date | Target Type | Target Name | Mission Type \ |
|-----|------------|--------------|-------------|-------------|-------------|----------------|
| 0 | MSN-0001 | Mission-1 | 2025-01-01 | Star | Titan | Colonization |
| 1 | MSN-0002 | Mission-2 | 2025-01-08 | Exoplanet | Betelgeuse | Colonization |
| 2 | MSN-0003 | Mission-3 | 2025-01-15 | Asteroid | Mars | Exploration |
| 3 | MSN-0004 | Mission-4 | 2025-01-22 | Exoplanet | Titan | Colonization |
| 4 | MSN-0005 | Mission-5 | 2025-01-29 | Exoplanet | Proxima b | Mining |
| .. | ... | ... | ... | ... | ... | ... |
| 495 | MSN-0496 | Mission-496 | 2034-06-28 | Planet | Betelgeuse | Colonization |
| 496 | MSN-0497 | Mission-497 | 2034-07-05 | Asteroid | Betelgeuse | Exploration |
| 497 | MSN-0498 | Mission-498 | 2034-07-12 | Planet | Ceres | Exploration |
| 498 | MSN-0499 | Mission-499 | 2034-07-19 | Planet | Betelgeuse | Research |
| 499 | MSN-0500 | Mission-500 | 2034-07-26 | Planet | Io | Exploration |

| | Distance from Earth (light-years) | Mission Duration (years) \ |
|-----|-----------------------------------|----------------------------|
| 0 | 7.05 | 5.2 |
| 1 | 41.76 | 23.0 |
| 2 | 49.22 | 28.8 |
| 3 | 26.33 | 17.8 |
| 4 | 8.67 | 9.2 |
| .. | ... | ... |
| 495 | 48.26 | 25.3 |
| 496 | 0.35 | 4.1 |
| 497 | 47.60 | 26.6 |
| 498 | 31.99 | 18.0 |
| 499 | 43.41 | 23.5 |

| | Mission Cost (billion USD) | Scientific Yield (points) | Crew Size \ |
|-----|----------------------------|---------------------------|-------------|
| 0 | 526.68 | 64.3 | 21 |
| 1 | 234.08 | 84.4 | 72 |
| 2 | 218.68 | 98.6 | 16 |
| 3 | 232.89 | 36.0 | 59 |
| 4 | 72.14 | 96.5 | 31 |
| .. | ... | ... | ... |
| 495 | 274.82 | 91.2 | 64 |
| 496 | 379.71 | 82.6 | 61 |
| 497 | 296.45 | 98.6 | 29 |
| 498 | 457.38 | 77.9 | 39 |
| 499 | 449.40 | 45.4 | 88 |

| | Mission Success (%) | Fuel Consumption (tons) | Payload Weight (tons) \ |
|-----|---------------------|-------------------------|-------------------------|
| 0 | 100.0 | 731.88 | 99.78 |
| 1 | 89.6 | 4197.41 | 45.72 |
| 2 | 98.6 | 4908.00 | 36.12 |
| 3 | 90.0 | 2569.05 | 40.67 |
| 4 | 73.2 | 892.76 | 12.40 |
| .. | ... | ... | ... |
| 495 | 96.2 | 4837.16 | 44.97 |
| 496 | 100.0 | 62.59 | 67.88 |
| 497 | 100.0 | 4794.01 | 51.38 |
| 498 | 100.0 | 3102.16 | 86.28 |
| 499 | 100.0 | 4302.93 | 86.74 |

| | Launch Vehicle |
|-----|----------------|
| 0 | SLS |
| 1 | Starship |
| 2 | Starship |
| 3 | Starship |
| 4 | Starship |
| .. | ... |
| 495 | Ariane 6 |
| 496 | SLS |
| 497 | Falcon Heavy |
| 498 | SLS |

499 Falcon Heavy

[500 rows x 15 columns]

Trabajando con fechas.

- Vamos a trabajar con la columnas Launch Date ya que este data si contiene fechas.

In [112...

```
print("\nOrdenamos por el nombre la misión en descendente")
df_ordenado_mision = df.sort_values(by="Mission Name", ascending=False)
print(df_ordenado_mision)
```

Ordenamos por el nombre la misión en descendente

| | Mission ID | Mission Name | Launch Date | Target Type | Target Name | Mission Type \ |
|-----|------------|--------------|-------------|-------------|-------------|----------------|
| 98 | MSN-0099 | Mission-99 | 2026-11-18 | Moon | Titan | Colonization |
| 97 | MSN-0098 | Mission-98 | 2026-11-11 | Moon | Io | Mining |
| 96 | MSN-0097 | Mission-97 | 2026-11-04 | Exoplanet | Ceres | Colonization |
| 95 | MSN-0096 | Mission-96 | 2026-10-28 | Asteroid | Europa | Exploration |
| 94 | MSN-0095 | Mission-95 | 2026-10-21 | Planet | Europa | Exploration |
| .. | ... | ... | ... | ... | ... | ... |
| 101 | MSN-0102 | Mission-102 | 2026-12-09 | Planet | Proxima b | Exploration |
| 100 | MSN-0101 | Mission-101 | 2026-12-02 | Star | Proxima b | Exploration |
| 99 | MSN-0100 | Mission-100 | 2026-11-25 | Planet | Ceres | Research |
| 9 | MSN-0010 | Mission-10 | 2025-03-05 | Exoplanet | Betelgeuse | Colonization |
| 0 | MSN-0001 | Mission-1 | 2025-01-01 | Star | Titan | Colonization |

| | Distance from Earth (light-years) | Mission Duration (years) \ |
|-----|-----------------------------------|----------------------------|
| 98 | 16.02 | 9.3 |
| 97 | 42.24 | 24.7 |
| 96 | 43.12 | 26.2 |
| 95 | 23.76 | 14.0 |
| 94 | 47.22 | 27.1 |
| .. | ... | ... |
| 101 | 29.85 | 16.2 |
| 100 | 1.95 | 5.8 |
| 99 | 41.46 | 25.4 |
| 9 | 28.87 | 18.9 |
| 0 | 7.05 | 5.2 |

| | Mission Cost (billion USD) | Scientific Yield (points) | Crew Size \ |
|-----|----------------------------|---------------------------|-------------|
| 98 | 283.51 | 10.8 | 89 |
| 97 | 511.73 | 96.2 | 74 |
| 96 | 217.39 | 29.0 | 43 |
| 95 | 309.23 | 35.1 | 80 |
| 94 | 433.28 | 82.4 | 76 |
| .. | ... | ... | ... |
| 101 | 360.96 | 84.6 | 68 |
| 100 | 123.33 | 70.9 | 95 |
| 99 | 310.28 | 99.8 | 20 |
| 9 | 265.98 | 23.9 | 34 |
| 0 | 526.68 | 64.3 | 21 |

| | Mission Success (%) | Fuel Consumption (tons) | Payload Weight (tons) \ |
|-----|---------------------|-------------------------|-------------------------|
| 98 | 96.1 | 1648.62 | 54.57 |
| 97 | 100.0 | 4238.98 | 95.84 |
| 96 | 99.4 | 4231.21 | 42.00 |
| 95 | 90.0 | 2414.29 | 60.26 |
| 94 | 100.0 | 4732.24 | 78.66 |
| .. | ... | ... | ... |
| 101 | 100.0 | 3125.73 | 62.94 |
| 100 | 82.3 | 230.04 | 22.83 |
| 99 | 100.0 | 4125.24 | 60.90 |
| 9 | 99.3 | 2856.43 | 44.76 |
| 0 | 100.0 | 731.88 | 99.78 |

| | Launch Vehicle |
|-----|----------------|
| 98 | Falcon Heavy |
| 97 | Falcon Heavy |
| 96 | Ariane 6 |
| 95 | Falcon Heavy |
| 94 | Falcon Heavy |
| .. | ... |
| 101 | Falcon Heavy |
| 100 | Falcon Heavy |
| 99 | Starship |
| 9 | Starship |

0 SLS

[500 rows x 15 columns]

In [115...

```
print("\nOrdenar por fecha de lanzamiento.")
df_ordenado_fecha = df.sort_values(by="Launch Date")
df_ordenado_fecha
```

Ordenar por fecha de lanzamiento.

Out[115]:

| | Mission ID | Mission Name | Launch Date | Target Type | Target Name | Mission Type | Distance from Earth (light-years) | Mission Duration (years) | Mission Cost (billion USD) |
|-----|------------|--------------|-------------|-------------|-------------|--------------|-----------------------------------|--------------------------|----------------------------|
| 0 | MSN-0001 | Mission-1 | 2025-01-01 | Star | Titan | Colonization | 7.05 | 5.2 | 526.68 |
| 1 | MSN-0002 | Mission-2 | 2025-01-08 | Exoplanet | Betelgeuse | Colonization | 41.76 | 23.0 | 234.08 |
| 2 | MSN-0003 | Mission-3 | 2025-01-15 | Asteroid | Mars | Exploration | 49.22 | 28.8 | 218.68 |
| 3 | MSN-0004 | Mission-4 | 2025-01-22 | Exoplanet | Titan | Colonization | 26.33 | 17.8 | 232.89 |
| 4 | MSN-0005 | Mission-5 | 2025-01-29 | Exoplanet | Proxima b | Mining | 8.67 | 9.2 | 72.14 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 495 | MSN-0496 | Mission-496 | 2034-06-28 | Planet | Betelgeuse | Colonization | 48.26 | 25.3 | 274.82 |
| 496 | MSN-0497 | Mission-497 | 2034-07-05 | Asteroid | Betelgeuse | Exploration | 0.35 | 4.1 | 379.71 |
| 497 | MSN-0498 | Mission-498 | 2034-07-12 | Planet | Ceres | Exploration | 47.60 | 26.6 | 296.45 |
| 498 | MSN-0499 | Mission-499 | 2034-07-19 | Planet | Betelgeuse | Research | 31.99 | 18.0 | 457.38 |
| 499 | MSN-0500 | Mission-500 | 2034-07-26 | Planet | Io | Exploration | 43.41 | 23.5 | 449.40 |

500 rows x 15 columns

Agregaciones estadísticas.

- Calculamos estadísticas y realizamos agregaciones para obtener insights.

In [127...

```
print("\nContar misiones por destino")
misiones_por_destino = df["Mission Type"].value_counts()
print("Misiones por destino:")
print(misiones_por_destino)
```

```
Contar misiones por destino
Misiones por destino:
Mission Type
Research      132
Exploration   127
Colonization  125
Mining        116
Name: count, dtype: int64
```

```
In [ ]: print('\nCalcular la duración promedio de las misiones (Años luz)')
        if "Distance from Earth (light-years)" in df.columns:
            duracion_promedio = df["Distance from Earth (light-years)"].mean()
            print("Duración promedio de las misiones:", duracion_promedio)
```

Calcular la duración promedio de las misiones (si existe una columna de duración)
Duración promedio de las misiones: 25.48306

```
In [126... print("\nAgrupar por agencia espacial y contar misiones")
            misiones_por_agencia = df.groupby("Mission Type")["Mission Name"].count()
            print("Misiones por agencia:")
            print(misiones_por_agencia)
```

```
Agrupar por agencia espacial y contar misiones
Misiones por agencia:
Mission Type
Colonization    125
Exploration     127
Mining          116
Research        132
Name: Mission Name, dtype: int64
```

Separacion de columnas por Numericas y Categoricas.

- Pandas nos permite filtrar columnas por tipo de datos usando `select_dtypes`.

```
In [133... # Seleccionar columnas numéricas
df_numericas = df.select_dtypes(include=['int64', 'float64'])
df_numericas
```

Out[133]:

| | Distance from Earth (light- years) | Mission Duration (years) | Mission Cost (billion USD) | Scientific Yield (points) | Crew Size | Mission Success (%) | Fuel Consumption (tons) | Payload Weight (tons) |
|-----|---|--------------------------------|-------------------------------------|---------------------------------|--------------|---------------------------|-------------------------------|-----------------------------|
| 0 | 7.05 | 5.2 | 526.68 | 64.3 | 21 | 100.0 | 731.88 | 99.78 |
| 1 | 41.76 | 23.0 | 234.08 | 84.4 | 72 | 89.6 | 4197.41 | 45.72 |
| 2 | 49.22 | 28.8 | 218.68 | 98.6 | 16 | 98.6 | 4908.00 | 36.12 |
| 3 | 26.33 | 17.8 | 232.89 | 36.0 | 59 | 90.0 | 2569.05 | 40.67 |
| 4 | 8.67 | 9.2 | 72.14 | 96.5 | 31 | 73.2 | 892.76 | 12.40 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 495 | 48.26 | 25.3 | 274.82 | 91.2 | 64 | 96.2 | 4837.16 | 44.97 |
| 496 | 0.35 | 4.1 | 379.71 | 82.6 | 61 | 100.0 | 62.59 | 67.88 |
| 497 | 47.60 | 26.6 | 296.45 | 98.6 | 29 | 100.0 | 4794.01 | 51.38 |
| 498 | 31.99 | 18.0 | 457.38 | 77.9 | 39 | 100.0 | 3102.16 | 86.28 |
| 499 | 43.41 | 23.5 | 449.40 | 45.4 | 88 | 100.0 | 4302.93 | 86.74 |

500 rows × 8 columns

In [134]:

```
# Seleccionar columnas categóricas (objetos y fechas)
df_categoricas = df.select_dtypes(include=['object'])
df_categoricas
```

Out[134]:

| | Mission ID | Mission Name | Target Type | Target Name | Mission Type | Launch Vehicle |
|-----|------------|--------------|-------------|-------------|--------------|----------------|
| 0 | MSN-0001 | Mission-1 | Star | Titan | Colonization | SLS |
| 1 | MSN-0002 | Mission-2 | Exoplanet | Betelgeuse | Colonization | Starship |
| 2 | MSN-0003 | Mission-3 | Asteroid | Mars | Exploration | Starship |
| 3 | MSN-0004 | Mission-4 | Exoplanet | Titan | Colonization | Starship |
| 4 | MSN-0005 | Mission-5 | Exoplanet | Proxima b | Mining | Starship |
| ... | ... | ... | ... | ... | ... | ... |
| 495 | MSN-0496 | Mission-496 | Planet | Betelgeuse | Colonization | Ariane 6 |
| 496 | MSN-0497 | Mission-497 | Asteroid | Betelgeuse | Exploration | SLS |
| 497 | MSN-0498 | Mission-498 | Planet | Ceres | Exploration | Falcon Heavy |
| 498 | MSN-0499 | Mission-499 | Planet | Betelgeuse | Research | SLS |
| 499 | MSN-0500 | Mission-500 | Planet | Io | Exploration | Falcon Heavy |

500 rows × 6 columns

Manejo de datos en diversos formatos.

Archivos CSV (Comma-Separated Values)

- El formato csv es uno de los más comunes para almacenar y compartir datos.
- Para leer un archivo CSV con Pandas usamos `pd.read_csv()`

Algunos parametros utiles en read_csv()

```
In [139... #import pandas as pd

# Leer un archivo CSV
#df_csv = pd.read_csv("space_missions.csv")

...
df_csv = pd.read_csv("space_missions.csv",
                    sep=",",      # Separador de columnas (por defecto es ",")
                    encoding="utf-8", # Codificación del archivo
                    skiprows=1,    # Omitir la primera fila (si es necesario)
                    na_values=["N/A", "NA", "None"]) # Tratar ciertos valores como nulos
...
```

```
Out[139]: '\ndf_csv = pd.read_csv("space_missions.csv", \n                    sep=",",
# Separador de columnas (por defecto es ",")\n                    encoding="utf-
8", # Codificación del archivo\n                    skiprows=1, # Omitir la pr
imera fila (si es necesario)\n                    na_values=["N/A", "NA", "Non
e"]) # Tratar ciertos valores como nulos\n'
```

Vamos a instalar librerías necesarias para hacer la conexión a MySQL, para poder hacer un EDA la Base de Datos.

- pymysql Permite la conexión con MySQL desde Python.
- SQLAlchemy Proporciona una interfaz para interactuar con base de datos SQL desde pandas

```
In [145... %pip install pymysql sqlalchemy pandas
```

```

Collecting pymysql
  Downloading PyMySQL-1.1.1-py3-none-any.whl.metadata (4.4 kB)
Collecting sqlalchemy
  Downloading SQLAlchemy-2.0.38-cp311-cp311-win_amd64.whl.metadata (9.9 kB)
Requirement already satisfied: pandas in c:\python\python311\lib\site-packages (2.2.1)
Collecting greenlet!=0.4.17 (from sqlalchemy)
  Downloading greenlet-3.1.1-cp311-cp311-win_amd64.whl.metadata (3.9 kB)
Collecting typing-extensions>=4.6.0 (from sqlalchemy)
  Downloading typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0 kB)
Requirement already satisfied: numpy<2,>=1.23.2 in c:\python\python311\lib\site-packages (from pandas) (1.26.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\lalo\appdata\roaming\python\python311\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\python\python311\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\python\python311\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: six>=1.5 in c:\users\lalo\appdata\roaming\python\python311\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Downloading PyMySQL-1.1.1-py3-none-any.whl (44 kB)
----- 45.0/45.0 kB 739.0 kB/s eta 0:00:00
Downloading SQLAlchemy-2.0.38-cp311-cp311-win_amd64.whl (2.1 MB)
----- 2.1/2.1 MB 6.1 MB/s eta 0:00:00
Downloading greenlet-3.1.1-cp311-cp311-win_amd64.whl (298 kB)
----- 298.9/298.9 kB 6.3 MB/s eta 0:00:00
Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Installing collected packages: typing-extensions, pymysql, greenlet, sqlalchemy
  Attempting uninstall: typing-extensions
    Found existing installation: typing_extensions 4.5.0
    Uninstalling typing_extensions-4.5.0:
      Successfully uninstalled typing_extensions-4.5.0
Successfully installed greenlet-3.1.1 pymysql-1.1.1 sqlalchemy-2.0.38 typing-extensions-4.12.2
Note: you may need to restart the kernel to use updated packages.

ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
tensorflow-intel 2.15.0 requires keras<2.16,>=2.15.0, but you have keras 3.0.4 which is incompatible.

[notice] A new release of pip is available: 24.0 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip

```

Conectar Pandas con MySQL

- Para conectar Pandas con una base de datos MySQL, necesitamos los datos de la conexión.

```

In [ ]: import pandas as pd
        from sqlalchemy import create_engine

        # Datos de conexión
        usuario = "" # Tu usuario de MySQL
        password = "" # Tu contraseña de MySQL
        host = "localhost" # Dirección del servidor (si está en la nube, usa la IP o URL)
        puerto = "3306" # Puerto por defecto de MySQL
        base_de_datos = "agendacontactos" # Nombre de tu base de datos

        # Crear la conexión con SQLAlchemy
        engine = create_engine(f"mysql+pymysql://{usuario}:{password}@{host}:{puerto}/{base_de_datos}")

        # Verificar conexión

```

```
try:
    with engine.connect() as connection:
        print(" Conexión exitosa a MySQL")
except Exception as e:
    print(f" Error al conectar: {e}")
```

✓ Conexión exitosa a MySQL

Leer datos desde MySQL

- Si ya tenemos una tabla en la base de datos, podemos leerla con Pandas usando `pd.read_sql()`

```
In [147... # Leer la tabla desde MySQL
df_sql = pd.read_sql("SELECT * FROM contacto", con=engine)

# Mostrar las primeras filas
print(df_sql.head())
```

| | id | celular | email | fecha_nacimiento | fecha_registro | \ |
|---|----|------------|-------------------|------------------|---------------------|---|
| 0 | 1 | 65123863 | karen@hotmail.com | 1991-05-30 | 2024-03-04 12:04:11 | |
| 1 | 2 | 78965422 | msf@gmail.com | 1968-01-19 | 2024-03-04 12:06:01 | |
| 2 | 3 | 9632488 | edi@gmail.com | 1988-03-13 | 2024-03-04 12:07:16 | |
| 3 | 4 | 97531896 | pao@gmail.com | 1998-07-22 | 2024-03-04 12:09:30 | |
| 4 | 6 | 5151515151 | lalo@gmail.com | 1990-02-02 | 2024-03-04 12:45:22 | |

| | nombre |
|---|------------|
| 0 | karen Aide |
| 1 | mario |
| 2 | edgar |
| 3 | paola |
| 4 | lalo |

Guardar un DataFrame en MySQL

- Si queremos guardar un DataFrame en una tabla de MySQL usamos `to_sql()`

```
In [ ]: df_sql.to_sql(name="pruebaPandas", con=engine, if_exists="replace", index=False)

print(" Datos guardados en la tabla 'pruebaPandas'")
```

✓ Datos guardados en la tabla 'pruebaPandas'

C:\Users\Lalo\AppData\Local\Temp\ipykernel_20152\970577762.py:1: UserWarning: The provided table name 'pruebaPandas' is not found exactly as such in the database after writing the table, possibly due to case sensitivity issues. Consider using lower case table names.

```
df_sql.to_sql(name="pruebaPandas", con=engine, if_exists="replace", index=False)
```

Verificamos que se guardo el data en la base de datos de MySQL.

```
In [152... df_sql = pd.read_sql("SHOW TABLES", con=engine)
df_sql
```

Out[152]: **Tables_in_agendacontactos**

| | |
|---|--------------|
| 0 | contacto |
| 1 | pruebapandas |

Fusión y combinación de DataFrames

- Cuando trabajamos con datos, es común que la información este dividida en multiples fuentes.
- Pandas nos proporciona herramientas para fusionar, conectar y combinar DataFrames de manera eficiente.

Concatenación por filas (axis=0)

```
In [153... import pandas as pd

# Crear dos DataFrames de misiones espaciales
df1 = pd.DataFrame({
    "Mission_Name": ["Apollo 11", "Apollo 12"],
    "Agency": ["NASA", "NASA"],
    "Year": [1969, 1969]
})

df2 = pd.DataFrame({
    "Mission_Name": ["Voyager 1", "Voyager 2"],
    "Agency": ["NASA", "NASA"],
    "Year": [1977, 1977]
})

# Concatenar los DataFrames (apilando filas)
df_concat = pd.concat([df1, df2], ignore_index=True)
print(df_concat)
```

| | Mission_Name | Agency | Year |
|---|--------------|--------|------|
| 0 | Apollo 11 | NASA | 1969 |
| 1 | Apollo 12 | NASA | 1969 |
| 2 | Voyager 1 | NASA | 1977 |
| 3 | Voyager 2 | NASA | 1977 |

Concatenación por columnas (axis=1)

- Si tenemos DataFrames con el mismo número de filas, podemos unirlos por columnas.

```
In [154... df3 = pd.DataFrame({
    "Rocket_Type": ["Saturn V", "Saturn V", "Titan IIIE", "Titan IIIE"]
})

# Concatenar columnas
df_concat_col = pd.concat([df_concat, df3], axis=1)
print(df_concat_col)
```

| | Mission_Name | Agency | Year | Rocket_Type |
|---|--------------|--------|------|-------------|
| 0 | Apollo 11 | NASA | 1969 | Saturn V |
| 1 | Apollo 12 | NASA | 1969 | Saturn V |
| 2 | Voyager 1 | NASA | 1977 | Titan IIIE |
| 3 | Voyager 2 | NASA | 1977 | Titan IIIE |

Fusión de DataFrames (pd.merge())

- LA función pd.merge() ns permite combinar DataFrames basándonos en una columna clave (similar a un JOIN)

```
In [155... df_misiones = pd.DataFrame({
    "Mission_ID": [1, 2, 3],
    "Mission_Name": ["Apollo 11", "Voyager 1", "Curiosity"],
    "Agency": ["NASA", "NASA", "NASA"]
})
```

```
df_costos = pd.DataFrame({
    "Mission_ID": [1, 2, 3],
    "Cost_Million": [355, 865, 2500]
})

# Fusionar usando la columna "Mission_ID" como clave
df_merge = pd.merge(df_misiones, df_costos, on="Mission_ID")
print(df_merge)
```

| | Mission_ID | Mission_Name | Agency | Cost_Million |
|---|------------|--------------|--------|--------------|
| 0 | 1 | Apollo 11 | NASA | 355 |
| 1 | 2 | Voyager 1 | NASA | 865 |
| 2 | 3 | Curiosity | NASA | 2500 |

Unión de DataFrames con (df.join())

- El método df.join() es útil para combinar DataFrames basándose en el índice.

In [156..

```
df_info = pd.DataFrame({
    "Mission_Name": ["Apollo 11", "Voyager 1", "Curiosity"],
    "Launch_Year": [1969, 1977, 2011]
}).set_index("Mission_Name")

df_agencias = pd.DataFrame({
    "Mission_Name": ["Apollo 11", "Voyager 1", "Curiosity"],
    "Agency": ["NASA", "NASA", "NASA"]
}).set_index("Mission_Name")

# Unir por índice
df_join = df_info.join(df_agencias)
print(df_join)
```

| | Launch_Year | Agency |
|--------------|-------------|--------|
| Mission_Name | | |
| Apollo 11 | 1969 | NASA |
| Voyager 1 | 1977 | NASA |
| Curiosity | 2011 | NASA |

Tipos de funciones en merge()

- Dependiendo de cómo queramos combinar los datos, usamos diferentes métodos.
1. Inner Join (how="inner") Solo mantiene los valores que existen en ambas tablas.
 2. Left Join (how="left") Mantiene todos los valores de la izquierda y completa con NaN los faltantes.
 3. Right Join (how="right") Mantiene todos los valores de la derecha.
 4. Full Outer Join (how="outer") Mantiene todos los valores de ambas tablas y completa con NaN los faltantes.

Integración con otras Librerías en Pandas

- Integración con NumPy: Pandas está construido sobre NumPy, lo que permite usar funciones y operaciones de arrays.
- Integración con Matplotlib y Seaborn: Pandas facilita la integración de datos mediante Matplotlib y Seaborn.
- Integración con SciPy: Permite aplicar funciones estadísticas avanzadas a DataFrames.
- Integración con Scikit-learn: Permite aplicar algoritmos de aprendizaje para entrenar modelos.

Por: Eduardo Soto.

Ing de Software UACM