Facultad de
Cs. de la Computación

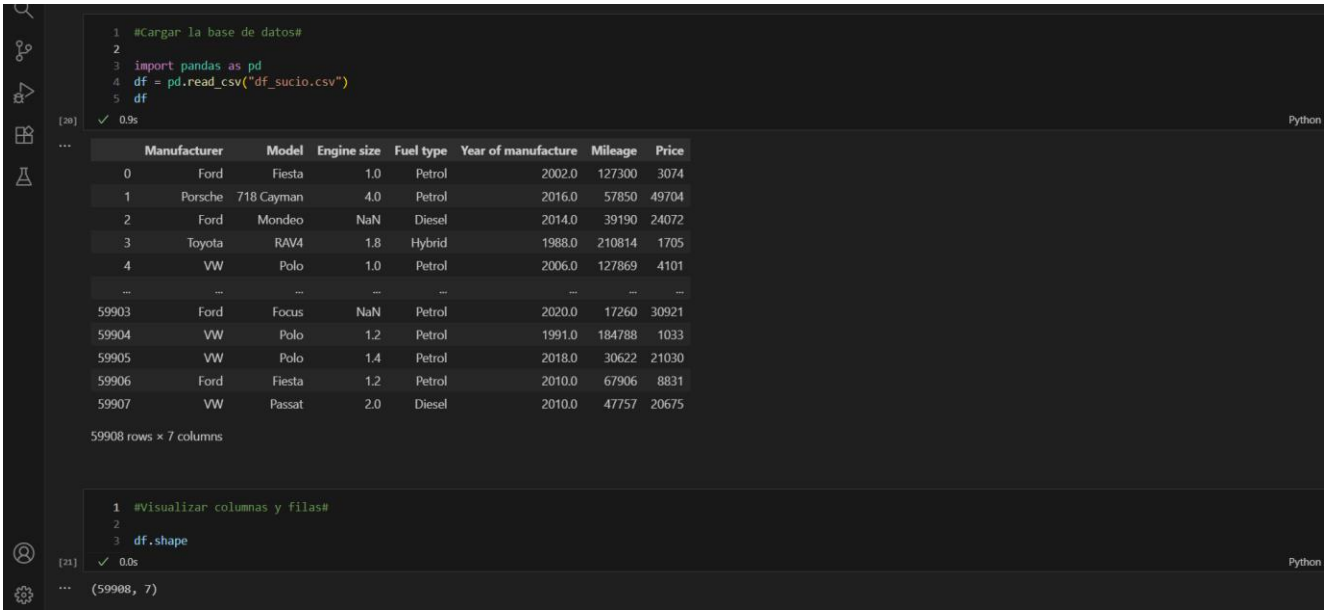
Introducción a la Ciencia de Datos.

Limpieza de datos.

Nombre del Alumno: Daniel Carvajal Garcia.
Nombre del Profesor: Jaime Alejandro Romero Sierra
Materia: Introducción a la Ciencia de datos.
Matrícula: 202502492
Licenciatura: Ingeniería en Ciencia de Datos
Benemérita Universidad Autónoma de Puebla
Fecha de entrega: 20 de Octubre 2025

Proceso de limpieza de datos de una base de datos sucia.

1. Se necesita cargar la base de datos, se utiliza la librería **Pandas** para leer archivos como **.csv** con **pd.read_csv()**, del mismo se utiliza el **df.shape()**, para visualizar columnas y filas del DataFrame.



```
#Cargar la base de datos#

import pandas as pd
df = pd.read_csv("df_sucio.csv")
df
```

✓ 0.9s — Python

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
0	Ford	Fiesta	1.0	Petrol	2002.0	127300	3074
1	Porsche	718 Cayman	4.0	Petrol	2016.0	57850	49704
2	Ford	Mondeo	NaN	Diesel	2014.0	39190	24072
3	Toyota	RAV4	1.8	Hybrid	1988.0	210814	1705
4	VW	Polo	1.0	Petrol	2006.0	127869	4101
...	...	...	...	...	...	...	...
59903	Ford	Focus	NaN	Petrol	2020.0	17260	30921
59904	VW	Polo	1.2	Petrol	1991.0	184788	1033
59905	VW	Polo	1.4	Petrol	2018.0	30622	21030
59906	Ford	Fiesta	1.2	Petrol	2010.0	67906	8831
59907	VW	Passat	2.0	Diesel	2010.0	47757	20675

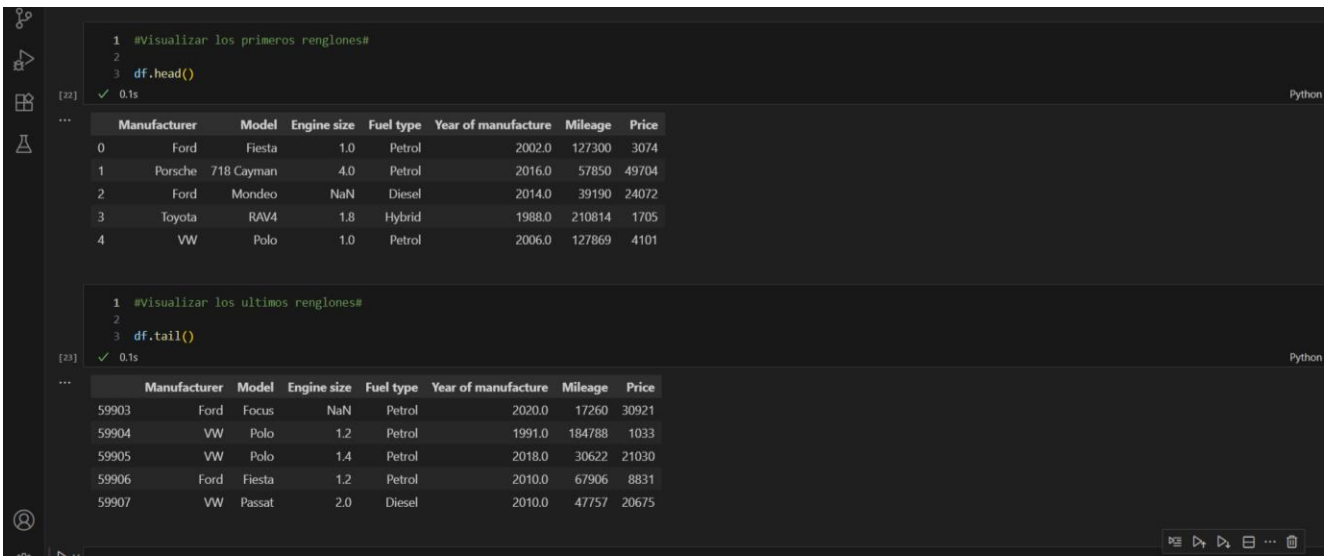
59908 rows × 7 columns

```
#Visualizar columnas y filas#

df.shape
```

✓ 0.0s — Python

(59908, 7)

2. Esto no es importante, sin embargo, lo agrego porque a mi criterio si llega a ser importante, uno es del **df.head()** sirve para visualizar los primeros renglones del dataframe, después en **df.tail()** sirve para visualizar los últimos renglones del DataFrame.



```
#Visualizar los primeros renglones#

df.head()
```

✓ 0.1s — Python

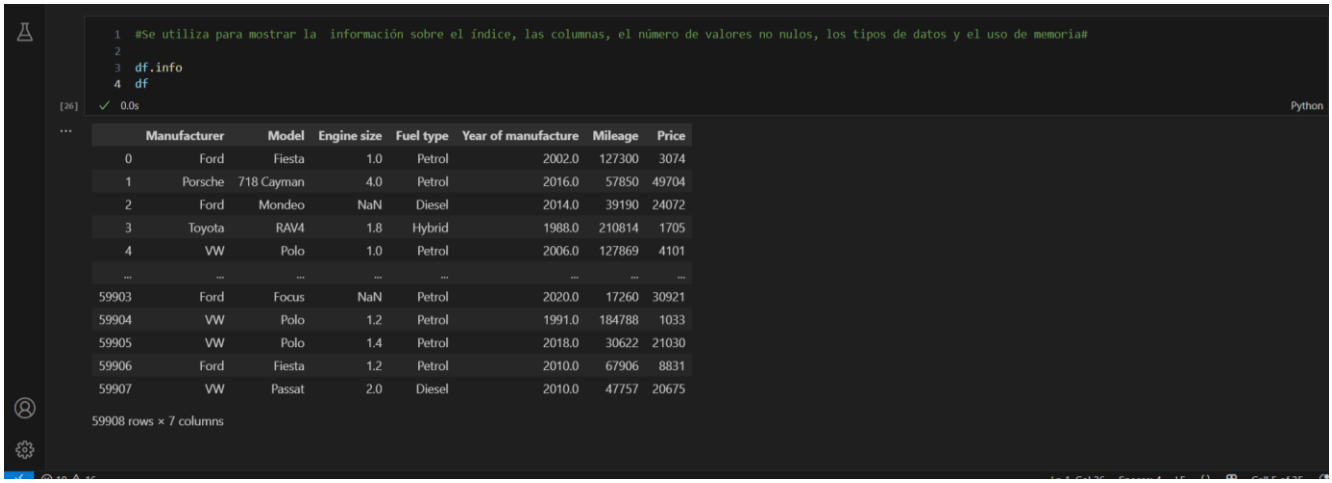
	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
0	Ford	Fiesta	1.0	Petrol	2002.0	127300	3074
1	Porsche	718 Cayman	4.0	Petrol	2016.0	57850	49704
2	Ford	Mondeo	NaN	Diesel	2014.0	39190	24072
3	Toyota	RAV4	1.8	Hybrid	1988.0	210814	1705
4	VW	Polo	1.0	Petrol	2006.0	127869	4101

```
#Visualizar los ultimos renglones#

df.tail()
```

✓ 0.1s — Python

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
59903	Ford	Focus	NaN	Petrol	2020.0	17260	30921
59904	VW	Polo	1.2	Petrol	1991.0	184788	1033
59905	VW	Polo	1.4	Petrol	2018.0	30622	21030
59906	Ford	Fiesta	1.2	Petrol	2010.0	67906	8831
59907	VW	Passat	2.0	Diesel	2010.0	47757	20675

3. Se utiliza el comando **df.info**, se usa para obtener un resumen conciso de un DataFrame de Pandas, proporciona información crucial como el número de entradas, los tipos de datos de cada columna, los valores no nulos y el uso de memoria, lo que es útil para comprender la estructura de los datos sin necesidad de verlos todos



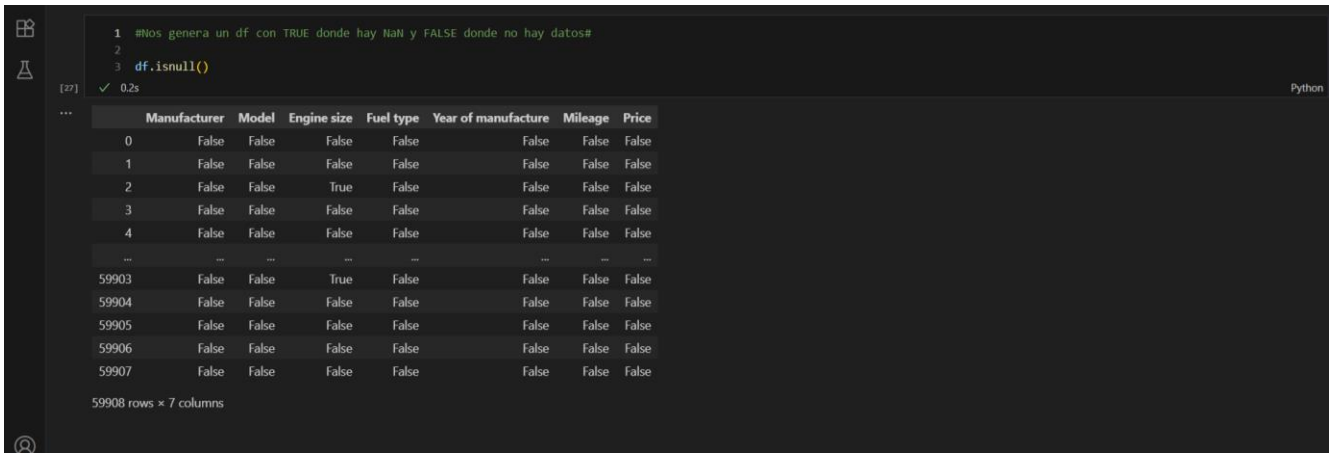
```
#Se utiliza para mostrar la  información sobre el índice, las columnas, el número de valores no nulos, los tipos de datos y el uso de memoria#

df.info
df
```

[26] ✓ 0.0s — Python

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
0	Ford	Fiesta	1.0	Petrol	2002.0	127300	3074
1	Porsche	718 Cayman	4.0	Petrol	2016.0	57850	49704
2	Ford	Mondeo	NaN	Diesel	2014.0	39190	24072
3	Toyota	RAV4	1.8	Hybrid	1988.0	210814	1705
4	VW	Polo	1.0	Petrol	2006.0	127869	4101
...	...	...	...	...	...	...	...
59903	Ford	Focus	NaN	Petrol	2020.0	17260	30921
59904	VW	Polo	1.2	Petrol	1991.0	184788	1033
59905	VW	Polo	1.4	Petrol	2018.0	30622	21030
59906	Ford	Fiesta	1.2	Petrol	2010.0	67906	8831
59907	VW	Passat	2.0	Diesel	2010.0	47757	20675

59908 rows × 7 columns

4. Ahora bien, un paso importante al momento de la limpieza de datos es el comando **df.isnull()**, se utiliza para detectar valores nulos (NaN) en un DataFrame, devuelve un DataFrame booleano del mismo tamaño que el original, con True donde haya un valor nulo y False en caso contrario.



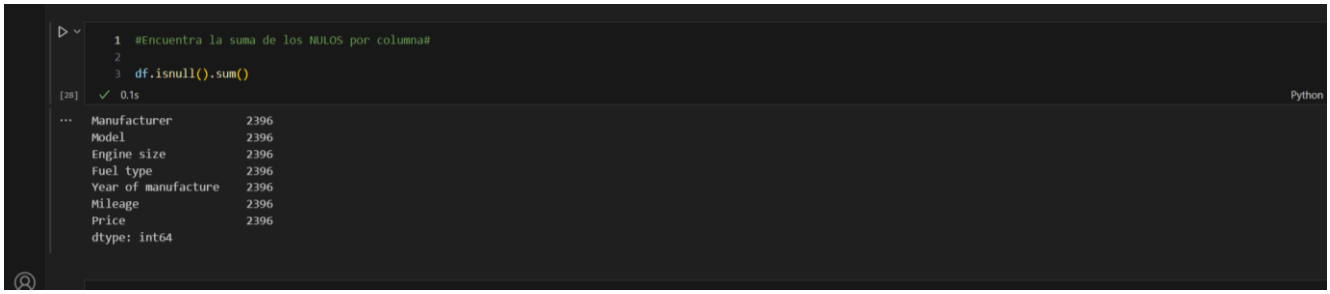
```
#Nos genera un df con TRUE donde hay NaN y FALSE donde no hay datos#

df.isnull()
```

[27] ✓ 0.2s — Python

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	True	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
59903	False	False	True	False	False	False	False
59904	False	False	False	False	False	False	False
59905	False	False	False	False	False	False	False
59906	False	False	False	False	False	False	False
59907	False	False	False	False	False	False	False

59908 rows × 7 columns

5. Ahora, utilice el comando **df.isnull().sum()**, se utiliza para contar la cantidad de valores faltantes (nulos o NaN) en cada columna de un DataFrame, la función **sum()** suma estos True para cada columna, devolviendo una serie con el recuento de nulos por columna.



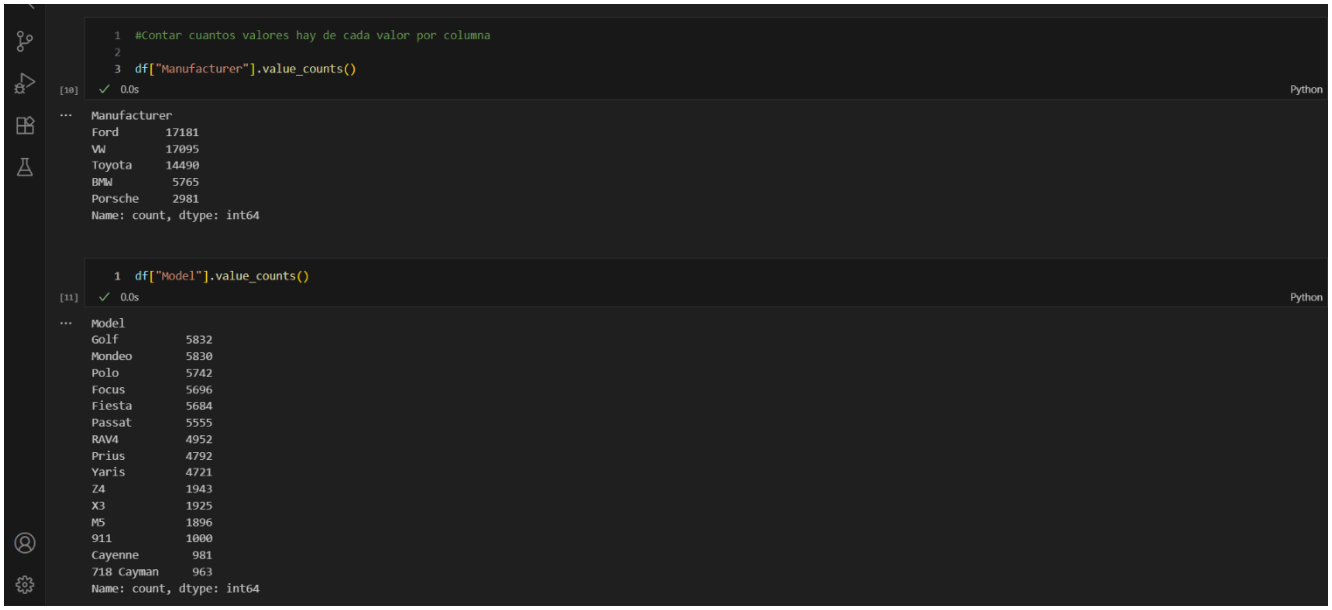
```
#Encuentra la suma de los NULOS por columna#

df.isnull().sum()
```

[28] ✓ 0.1s — Python

```
Manufacturer          2396
Model                 2396
Engine size           2396
Fuel type             2396
Year of manufacture   2396
Mileage               2396
Price                 2396
dtype: int64
```

6. En este punto, utilice el comando **df[].value_counts()**, me oriento para contar la frecuencia de valores únicos en una columna del DataFrame, porque devuelve una serie de Pandas que muestra cuántas veces aparece cada valor, ordenada por defecto de mayor a menor frecuencia, me funciono para identificar si dentro de los valores había

palabras extrañas, en este caso, pongo el ejemplo de 2 columnas de la tabla, sin embargo, se realizó el análisis de las 7 columnas de la tabla.



```
#Contar cuantos valores hay de cada valor por columna

df["Manufacturer"].value_counts()
```

```
Manufacturer
Ford       17181
VW         17095
Toyota     14490
BMW         5765
Porsche     2981
Name: count, dtype: int64
```

```
df["Model"].value_counts()
```

```
Model
Golf          5832
Mondeo        5830
Polo          5742
Focus         5696
Fiesta        5684
Passat        5555
RAV4          4952
Prius         4792
Yaris         4721
Z4            1943
X3            1925
M5            1896
911           1000
Cayenne        981
718 Cayman     963
Name: count, dtype: int64
```

7. Después utilice el comando **df[].unique()**, me sirvió para obtener una lista de todos los valores únicos en una columna (una Serie) del DataFrame, fue muy útil para explorar los datos, como encontrar todos los valores posibles en una categoría (por ejemplo, todos los modelos de un auto en una columna de "Model"), igual para detectar datos duplicados, y encontrar los NaN y nombres extraños dentro de una columna.

```
#Visualizar que valores unicos vienen en cada columna

df["Manufacturer"].unique()
```

```
array(['Ford', 'Porsche', 'Toyota', 'VW', nan, 'BMW'], dtype=object)
```

```
df["Model"].unique()
```

```
array(['Fiesta', '718 Cayman', 'Mondeo', 'RAV4', 'Polo', 'Focus', 'Prius',
       'Golf', 'Z4', 'Yaris', '911', nan, 'Passat', 'M5', 'Cayenne', 'X3'],
      dtype=object)
```

```
df["Engine size"].unique()
```

```
array([1. , 4. , nan, 1.8, 1.4, 1.2, 2. , 2.2, 1.6, 2.4, 2.6, 3.5, 4.4,
       3. , 5. ])
```

```
df["Fuel type"].unique()
```

```
array(['Petrol', 'Diesel', 'Hybrid', nan, 'ayuder$'], dtype=object)
```

8. Volví aplicar el comando **df.info** esta para obtener un resumen conciso que incluye el tipo de los índices, los tipos de datos de cada columna, el recuento de valores no nulos y el uso de memoria, igual me funciono para identificar cuál es la mejor opción para graficar cada columna, por ejemplo, las que dicen “object” son las que se pueden graficar, las que dicen “float64” la clave no está en el tipo de dato en sí, sino en que estén organizados de una manera adecuada para ser representados.

```
#Tipos de datos por columna#

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59908 entries, 0 to 59907
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Manufacturer        57512 non-null  object
 1   Model               57512 non-null  object
 2   Engine size         57512 non-null  float64
 3   Fuel type           57512 non-null  object
 4   Year of manufacture 57512 non-null  float64
 5   Mileage             57512 non-null  object
 6   Price               57512 non-null  object
dtypes: float64(2), object(5)
memory usage: 3.2+ MB
```

9. Antes de limpiar los datos, utilice **df.describe()**, para generar un resumen estadístico de las columnas del DataFrame, ofreciendo una visión general rápida de los datos numéricos. Es una herramienta fundamental para el análisis exploratorio de datos y ayuda a identificar rápidamente la tendencia central, la dispersión y la distribución de los datos.

```
#Antes de limpiar los datos, una visión general rápida de los datos numéricos#

df.describe()
df
```

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
0	Ford	Fiesta	1.0	Petrol	2002.0	127300	3074
1	Porsche	718 Cayman	4.0	Petrol	2016.0	57850	49704
2	Ford	Mondeo	NaN	Diesel	2014.0	39190	24072
3	Toyota	RAV4	1.8	Hybrid	1988.0	210814	1705
4	VW	Polo	1.0	Petrol	2006.0	127869	4101
...	...	...	...	...	...	...	...
59903	Ford	Focus	NaN	Petrol	2020.0	17260	30921
59904	VW	Polo	1.2	Petrol	1991.0	184788	1033
59905	VW	Polo	1.4	Petrol	2018.0	30622	21030
59906	Ford	Fiesta	1.2	Petrol	2010.0	67906	8831
59907	VW	Passat	2.0	Diesel	2010.0	47757	20675

59908 rows × 7 columns

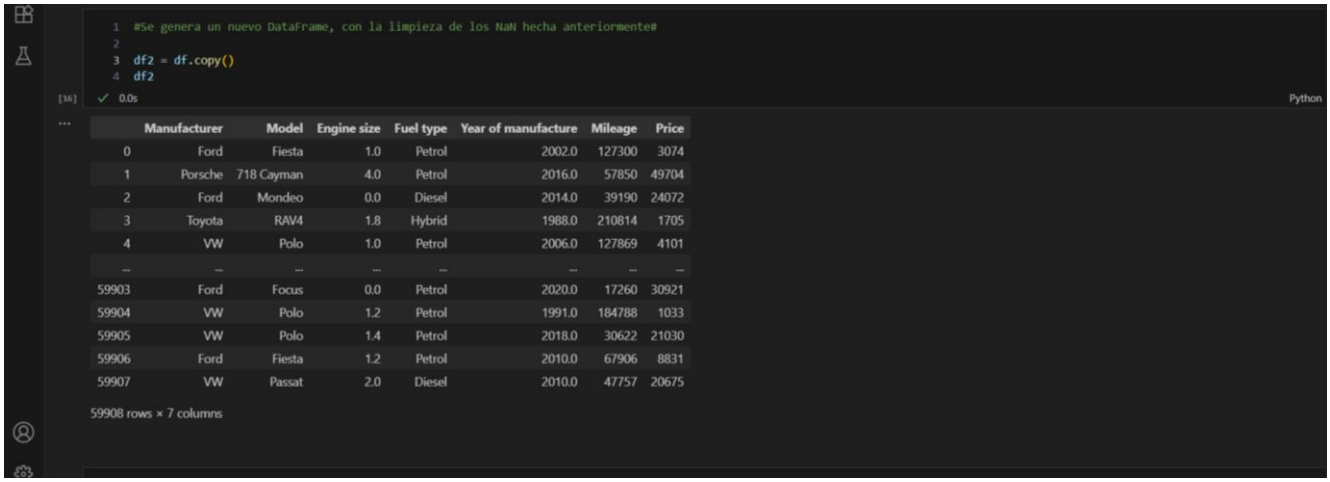
10. **Limpieza de datos:** Como primer paso utilice el comando **df.fillna(0, inplace=True),** el cual me funciono para reemplazar todos los valores nulos (NaN) en un DataFrame por el número (0), modificando el DataFrame original directamente en lugar de devolver uno nuevo. Esto es útil para la limpieza de datos, ya que sustituye los valores faltantes por un valor específico y numérico para facilitar el análisis posterior. Además, recordando que NO se puede utilizar dropna().

```
#Limpieza datos#

#Se utiliza en la biblioteca Pandas para reemplazar valores faltantes (representados como NaN) en un DataFrame#
#Sirve para limpiar datos y preparar un conjunto de datos para el análisis, ya sea reemplazando los valores nulos con un valor específico (como 0)#

df.fillna(0, inplace=True)
df
```

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
0	Ford	Fiesta	1.0	Petrol	2002.0	127300	3074
1	Porsche	718 Cayman	4.0	Petrol	2016.0	57850	49704
2	Ford	Mondeo	0.0	Diesel	2014.0	39190	24072
3	Toyota	RAV4	1.8	Hybrid	1988.0	210814	1705
4	VW	Polo	1.0	Petrol	2006.0	127869	4101
...	...	...	...	...	...	...	...
59903	Ford	Focus	0.0	Petrol	2020.0	17260	30921
59904	VW	Polo	1.2	Petrol	1991.0	184788	1033
59905	VW	Polo	1.4	Petrol	2018.0	30622	21030
59906	Ford	Fiesta	1.2	Petrol	2010.0	67906	8831
59907	VW	Passat	2.0	Diesel	2010.0	47757	20675

59908 rows × 7 columns

11. Después el comando df2=df.copy() el cual crea una copia independiente del DataFrame original para que las modificaciones hechas en df2 no afecten al original. Esto es crucial para el análisis de datos.



```
#Se genera un nuevo DataFrame, con la limpieza de los NaN hecha anteriormente#

df2 = df.copy()
df2
```

[16] ✓ 0.0s — Python

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
0	Ford	Fiesta	1.0	Petrol	2002.0	127300	3074
1	Porsche	718 Cayman	4.0	Petrol	2016.0	57850	49704
2	Ford	Mondeo	0.0	Diesel	2014.0	39190	24072
3	Toyota	RAV4	1.8	Hybrid	1988.0	210814	1705
4	VW	Polo	1.0	Petrol	2006.0	127869	4101
...	...	...	...	...	...	...	...
59903	Ford	Focus	0.0	Petrol	2020.0	17260	30921
59904	VW	Polo	1.2	Petrol	1991.0	184788	1033
59905	VW	Polo	1.4	Petrol	2018.0	30622	21030
59906	Ford	Fiesta	1.2	Petrol	2010.0	67906	8831
59907	VW	Passat	2.0	Diesel	2010.0	47757	20675

59908 rows × 7 columns

12. Para rectificar la limpieza de datos, con el comando **df2.isnull()**, se revisa si tenemos valores NaN dentro de los datos, hasta este momento se observo que ya no tenemos NaN porque no hay algún True dentro de la tabla.

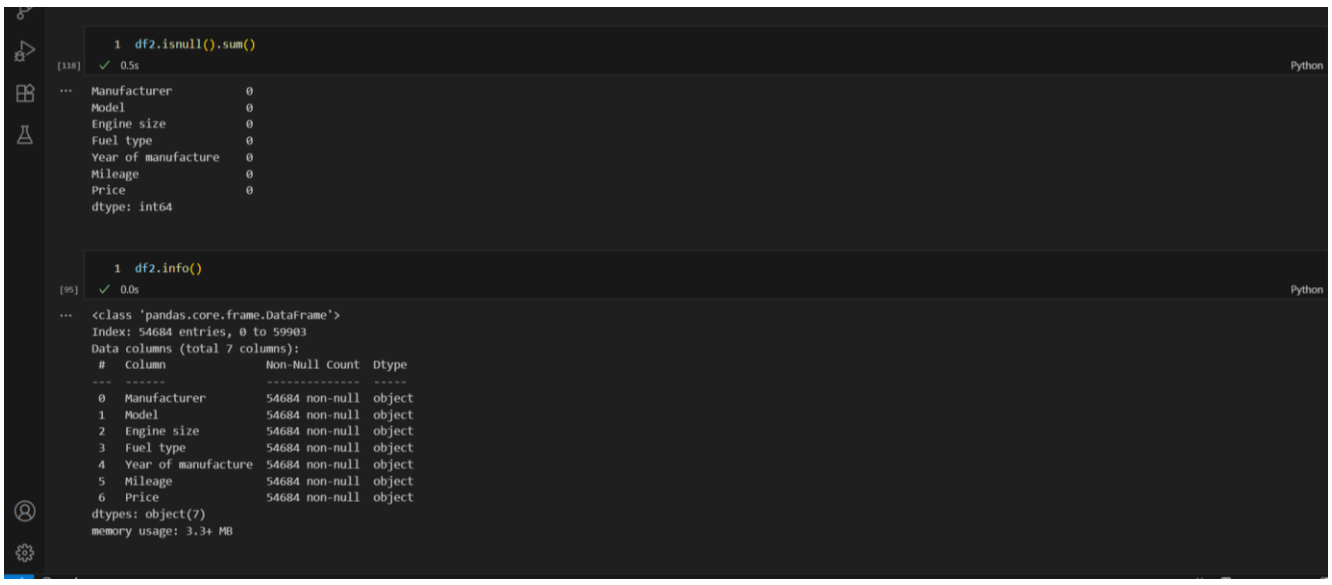
```
#Nos genera un df con True donde hay NaN y False donde hay datos, se puede observar que ya no tenemos NaN porque no aprece algún True en la tabla#

df2.isnull()
```

[18] ✓ 0.0s — Python

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
59903	False	False	False	False	False	False	False
59904	False	False	False	False	False	False	False
59905	False	False	False	False	False	False	False
59906	False	False	False	False	False	False	False
59907	False	False	False	False	False	False	False

59908 rows × 7 columns

13. Volví aplicar el comando **df2.isnull().sum()** esto para obtener si todavía tenemos valores NaN, lo cual se observa que ya tenemos valores NaN, sin embargo, al momento de aplicar **df2.info()**, todas las columnas me lanzan como “object”, lo cual no es correcto porque hay datos de valores numéricos.



```
df2.isnull().sum()
```

[118] ✓ 0.5s — Python

```
Manufacturer          0
Model                 0
Engine size           0
Fuel type             0
Year of manufacture   0
Mileage               0
Price                 0
dtype: int64
```

```
df2.info()
```

[95] ✓ 0.0s — Python

```
<class 'pandas.core.frame.DataFrame'>
Index: 54684 entries, 0 to 59903
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Manufacturer        54684 non-null  object
 1   Model               54684 non-null  object
 2   Engine size         54684 non-null  object
 3   Fuel type           54684 non-null  object
 4   Year of manufacture 54684 non-null  object
 5   Mileage             54684 non-null  object
 6   Price               54684 non-null  object
dtypes: object(7)
memory usage: 3.3+ MB
```

14. En este caso utilice el comando **df[] = pd.to_numeric(df[], errors='coerce')** lo utilice para convertir una columna del DataFrame a tipo numérico, reemplazando los valores que no se puedan convertir por NaN. Se usa comúnmente para limpiar datos donde una columna, que debería contener solo números, tiene valores inválidos, como texto, símbolos o cadenas que no se pueden convertir a números**.**

```
#Para datos atípicos se necesita el boxplot numérico, se convertira la columna en este caso a numéricas#

df2['Engine size'] = pd.to_numeric(df['Engine size'], errors='coerce').fillna(0)
df2['Year of manufacture'] = pd.to_numeric(df['Year of manufacture'], errors='coerce').fillna(0)
df2["Price"]=pd.to_numeric(df['Price'], errors='coerce').fillna(0)
df2["Mileage"]=pd.to_numeric(df['Mileage'], errors='coerce').fillna(0)
df2
```

[16] ✓ 0.2s — Python

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
0	Ford	Fiesta	1.0	Petrol	2002.0	127300.0	3074.0
1	Porsche	718 Cayman	4.0	Petrol	2016.0	57850.0	49704.0
2	Ford	Mondeo	0.0	Diesel	2014.0	39190.0	24072.0
3	Toyota	RAV4	1.8	Hybrid	1988.0	210814.0	1705.0
4	VW	Polo	1.0	Petrol	2006.0	127869.0	4101.0
...	...	...	...	...	...	...	...
59903	Ford	Focus	0.0	Petrol	2020.0	17260.0	30921.0
59904	VW	Polo	1.2	Petrol	1991.0	184788.0	1033.0
59905	VW	Polo	1.4	Petrol	2018.0	30622.0	21030.0
59906	Ford	Fiesta	1.2	Petrol	2010.0	67906.0	8831.0
59907	VW	Passat	2.0	Diesel	2010.0	47757.0	20675.0

59908 rows × 7 columns

15. Después de ese análisis, de nuevo aplique de nuevo df2.info() para rectificar que las columnas que tienen datos numéricos estén en float64 y que las demás columnas este en object.

```
df2.info()
```

[41] ✓ 0.0s — Python

```
<class 'pandas.core.frame.DataFrame'>
Index: 54650 entries, 0 to 59903
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Manufacturer        54650 non-null  object
 1   Model               54650 non-null  object
 2   Engine size         54650 non-null  float64
 3   Fuel type           54650 non-null  object
 4   Year of manufacture 54650 non-null  float64
 5   Mileage             54650 non-null  float64
 6   Price               54650 non-null  float64
dtypes: float64(4), object(3)
memory usage: 3.3+ MB
```

16. A partir de aquí utilice el comando **df2[col] = df2[col].str.strip().str.lower()** sirve para limpiar y estandarizar una columna de texto del DataFrame, eliminando espacios en blanco al principio y al final de cada cadena, luego convirtiendo todo el texto a minúsculas para un mejor manejo de los datos.

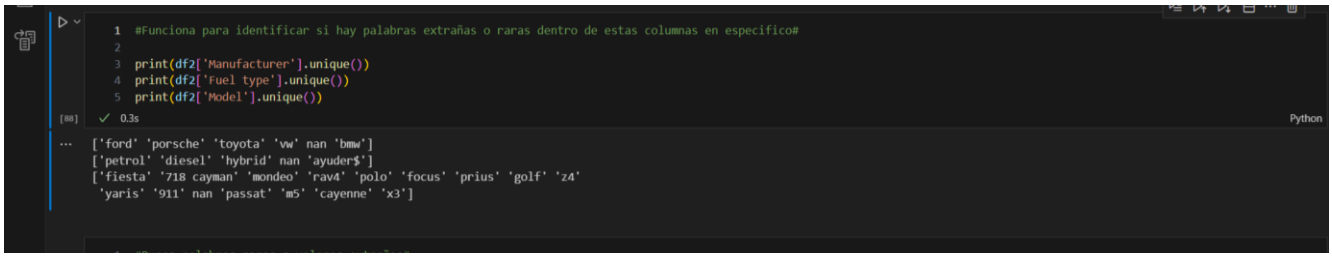
```
#Convertir las palabras a minicusculas, esto me sirve porque evita duplicados y mejora la consistencia al agrupar o contar categorías#

for col in ['Manufacturer', 'Model', 'Fuel type']:
    df2[col] = df2[col].str.strip().str.lower()
df2
```

[87] ✓ 0.3s — Python

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
0	ford	fiesta	1.0	petrol	2002.0	127300.0	3074.0
1	porsche	718 cayman	4.0	petrol	2016.0	57850.0	49704.0
2	ford	mondeo	0.0	diesel	2014.0	39190.0	24072.0
3	toyota	rav4	1.8	hybrid	1988.0	210814.0	1705.0
4	vw	polo	1.0	petrol	2006.0	127869.0	4101.0
...	...	...	...	...	...	...	...
59903	ford	focus	0.0	petrol	2020.0	17260.0	30921.0
59904	vw	polo	1.2	petrol	1991.0	184788.0	1033.0
59905	vw	polo	1.4	petrol	2018.0	30622.0	21030.0
59906	ford	fiesta	1.2	petrol	2010.0	67906.0	8831.0
59907	vw	passat	2.0	diesel	2010.0	47757.0	20675.0

59908 rows × 7 columns

17. En las columnas donde hay datos de texto, aplique sprint.unique() , porque devuelve todos los valores únicos de una columna de un DataFrame o Serie, en formato de un array NumPy. Me funciono para poder identificar palabras extrañas o raras dentro de la base de datos.

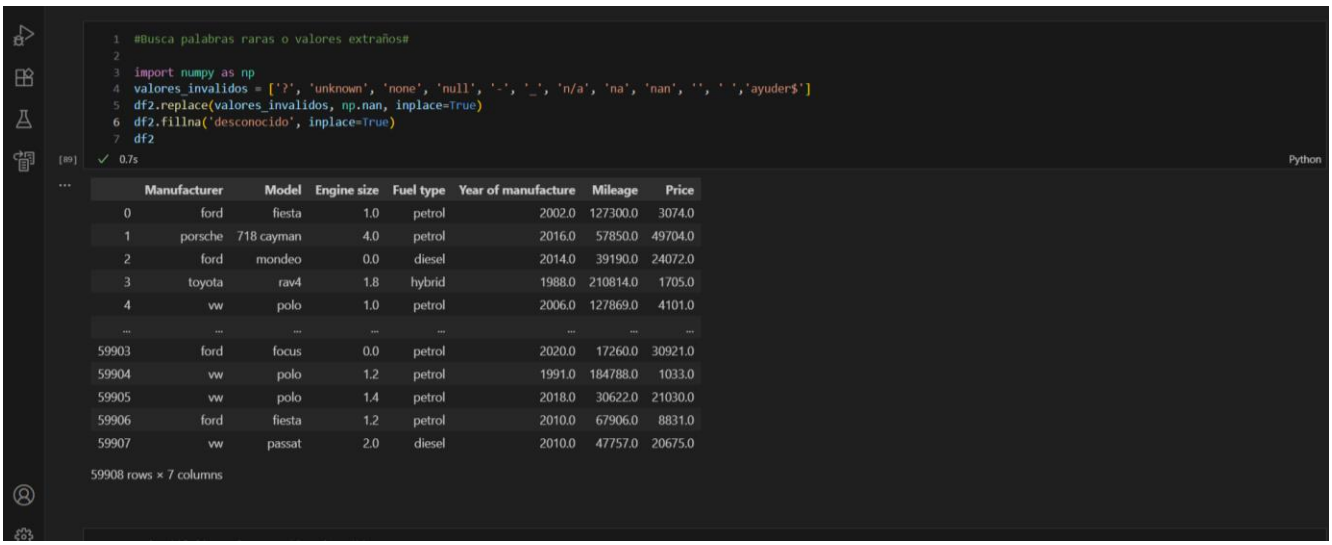


```
#Funciona para identificar si hay palabras extrañas o raras dentro de estas columnas en especifico#

print(df2['Manufacturer'].unique())
print(df2['Fuel type'].unique())
print(df2['Model'].unique())
```

0.3s — Python

```
['ford' 'porsche' 'toyota' 'vw' nan 'bmw']
['petrol' 'diesel' 'hybrid' nan 'ayuder$']
['fiesta' '718 cayman' 'mondeo' 'rav4' 'polo' 'focus' 'prius' 'golf' 'z4'
 'yaris' '911' nan 'passat' 'm5' 'cayenne' 'x3']
```

18. Para eliminar palabras extrañas o raras, primero se crea una lista con posibles palabras extrañas, de ahí junto con el comando **df2.replace(valores_invalidos, np.nan, inplace=True)** se utiliza para reemplazar valores específicos o no deseados del DataFrame con el valor np.nan (Not a Number), que representa datos faltantes o nulos, también **df2.fillna('desconocido', inplace=True)** sirve para rellenar todos los valores faltantes (NaN) del DataFrame con la cadena de texto 'desconocido'.



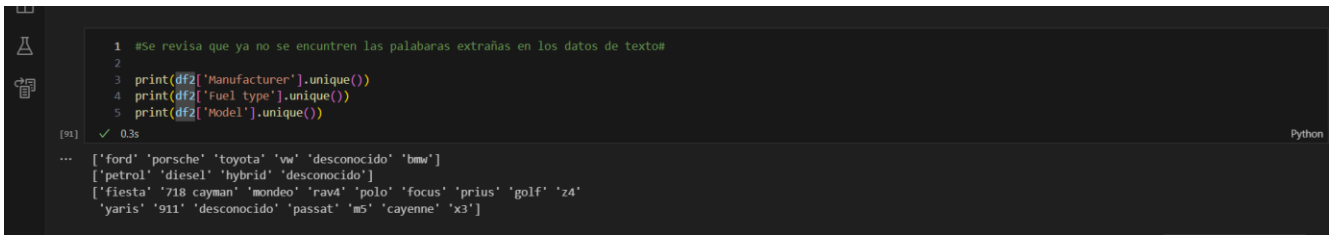
```
#Busca palabras raras o valores extraños#

import numpy as np
valores_invalidos = ['?', 'unknown', 'none', 'null', '-', '_', 'n/a', 'na', 'nan', '', ' ','ayuder$']
df2.replace(valores_invalidos, np.nan, inplace=True)
df2.fillna('desconocido', inplace=True)
df2
```

0.7s — Python

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
0	ford	fiesta	1.0	petrol	2002.0	127300.0	3074.0
1	porsche	718 cayman	4.0	petrol	2016.0	57850.0	49704.0
2	ford	mondeo	0.0	diesel	2014.0	39190.0	24072.0
3	toyota	rav4	1.8	hybrid	1988.0	210814.0	1705.0
4	vw	polo	1.0	petrol	2006.0	127869.0	4101.0
...	...	...	...	...	...	...	...
59903	ford	focus	0.0	petrol	2020.0	17260.0	30921.0
59904	vw	polo	1.2	petrol	1991.0	184788.0	1033.0
59905	vw	polo	1.4	petrol	2018.0	30622.0	21030.0
59906	ford	fiesta	1.2	petrol	2010.0	67906.0	8831.0
59907	vw	passat	2.0	diesel	2010.0	47757.0	20675.0

59908 rows × 7 columns

19. Después de identifica que las palabras extrañas queden nombradas como “Desconocido”.



```
#Se revisa que ya no se encuntren las palabaras extrañas en los datos de texto#

print(df2['Manufacturer'].unique())
print(df2['Fuel type'].unique())
print(df2['Model'].unique())
```

0.3s — Python

```
['ford' 'porsche' 'toyota' 'vw' 'desconocido' 'bmw']
['petrol' 'diesel' 'hybrid' 'desconocido']
['fiesta' '718 cayman' 'mondeo' 'rav4' 'polo' 'focus' 'prius' 'golf' 'z4'
 'yaris' '911' 'desconocido' 'passat' 'm5' 'cayenne' 'x3']
```

20. Para limpiar por completo los datos de texto utilice el comando
for col in df2.select_dtypes(include='object').columns:
df2[col] = df2[col].apply(limpiar_texto)
El cual me elimina caracteres raros o símbolos, elimina espacios sobrantes y convierte todo el texto a minúscula, así los datos quedan más uniformes y ordenados para analizarlos después.

```python
#Elimina caracteres raros o símbolos#
#Quita espacios sobrantes#
#Convierte todo a minúsculas#

import re

def limpiar_texto(texto):
    if isinstance(texto, str):
        texto = re.sub(r'[^A-Za-z0-9\s\-]', '', texto)  # deja letras, números y guiones
        texto = re.sub(r'\s+', ' ', texto).strip()
        return texto.lower()
    return texto

for col in df2.select_dtypes(include='object').columns:
    df2[col] = df2[col].apply(limpiar_texto)
df2
```

[92] ✓ 1.3s — Python

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
0	ford	fiesta	1.0	petrol	2002.0	127300.0	3074.0
1	porsche	718 cayman	4.0	petrol	2016.0	57850.0	49704.0
2	ford	mondeo	0.0	diesel	2014.0	39190.0	24072.0
3	toyota	rav4	1.8	hybrid	1988.0	210814.0	1705.0
4	vw	polo	1.0	petrol	2006.0	127869.0	4101.0
...	...	...	...	...	...	...	...
59903	ford	focus	0.0	petrol	2020.0	17260.0	30921.0
59904	vw	polo	1.2	petrol	1991.0	184788.0	1033.0
59905	vw	polo	1.4	petrol	2018.0	30622.0	21030.0
59906	ford	fiesta	1.2	petrol	2010.0	67906.0	8831.0
59907	vw	passat	2.0	diesel	2010.0	47757.0	20675.0

59908 rows × 7 columns

21. Para finalizar la limpieza de datos, se utiliza el comando **df2.duplicate()**, para identificar filas duplicadas del DataFrame. Devuelve una serie booleana (True/False) donde True indica que la fila es un duplicado y False que es única. Esta función es una herramienta esencial para la limpieza de datos, ya que permite verificar la presencia de duplicados antes de eliminarlos usando

```python
#Con este comando se crea un booleano que muestra renglones duplicados#

df2.duplicated()
```

[94] ✓ 0.2s — Python

```
0        False
1        False
2        False
3        False
4        False
         ...
59903    False
59904     True
59905     True
59906     True
59907     True
Length: 59908, dtype: bool
```

22. Teniendo esto en cuenta, para sumar todos los datos duplicados se utiliza el comando **df2.duplicated().sum()** se utiliza para contar el número total de filas duplicadas del DataFrame, combina el método duplicated() que marca las filas duplicadas como True con el método sum() que cuenta cuántos True hay en esa serie booleana, del mismo modo el comando **df2.duplicated(subset=[]).sum(),** cuenta el número total de filas duplicadas del DataFrame, considerando solo las columnas especificadas en subset, en este caso las columnas.

```python
df2.duplicated().sum()
```

[95] ✓ 0.0s — Python

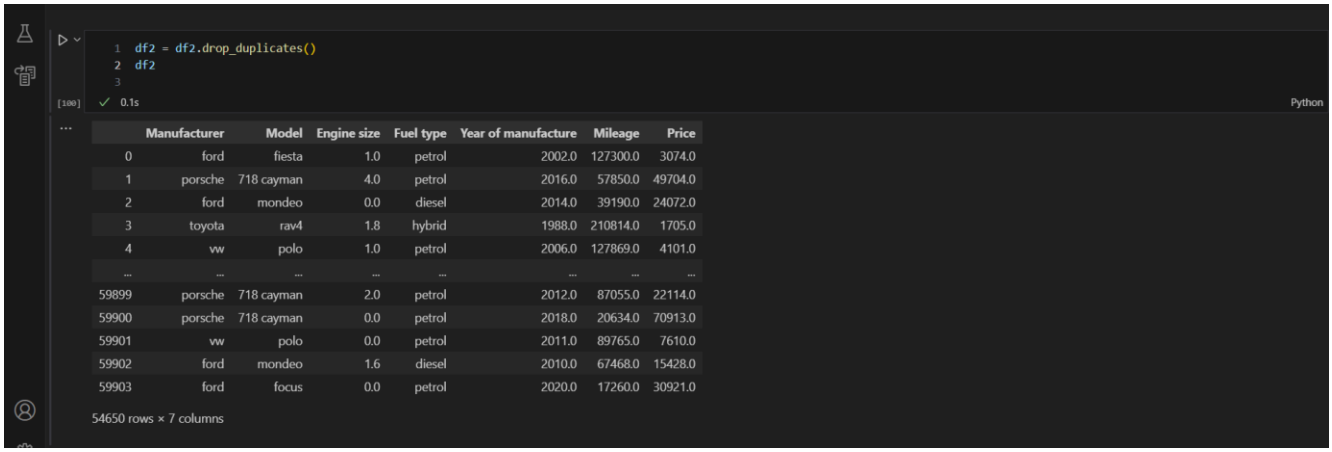
```
np.int64(5258)
```

```python
#Compara solo las columnas 'CustomerId', 'Surname' por lo que no

df.duplicated(subset=['Manufacturer', 'Model','Engine size', 'Fuel type','Year of manufacture', 'Mileage','Price',]).sum()
```

[96] ✓ 0.1s — Python

```
np.int64(5224)
```

23. Para eliminar las filas duplicadas se utiliza el comando **df2= df2.drop_duplicates()** sirve para eliminar las filas que están completamente duplicadas del DataFrame.



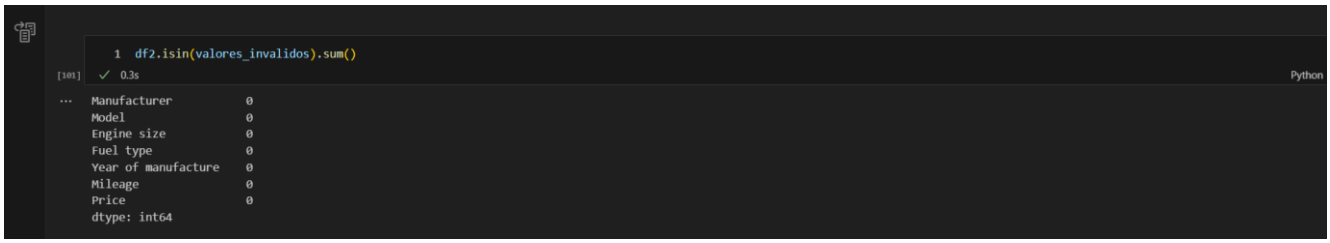
```
df2 = df2.drop_duplicates()
df2
```

[100] ✓ 0.1s — Python

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
0	ford	fiesta	1.0	petrol	2002.0	127300.0	3074.0
1	porsche	718 cayman	4.0	petrol	2016.0	57850.0	49704.0
2	ford	mondeo	0.0	diesel	2014.0	39190.0	24072.0
3	toyota	rav4	1.8	hybrid	1988.0	210814.0	1705.0
4	vw	polo	1.0	petrol	2006.0	127869.0	4101.0
...	...	...	...	...	...	...	...
59899	porsche	718 cayman	2.0	petrol	2012.0	87055.0	22114.0
59900	porsche	718 cayman	0.0	petrol	2018.0	20634.0	70913.0
59901	vw	polo	0.0	petrol	2011.0	89765.0	7610.0
59902	ford	mondeo	1.6	diesel	2010.0	67468.0	15428.0
59903	ford	focus	0.0	petrol	2020.0	17260.0	30921.0

54650 rows × 7 columns

24. Para rectificar que se realizó correctamente el comando se utiliza el comando **df2.isin(valores_invalidos).sum()** sirve para contar el número de valores no deseados o inválidos en cada columna del DataFrame. Esta operación es especialmente útil durante el proceso de limpieza de datos para identificar y cuantificar la presencia de datos erróneos, se puede rectificar de que ya NO existe filas duplicadas.

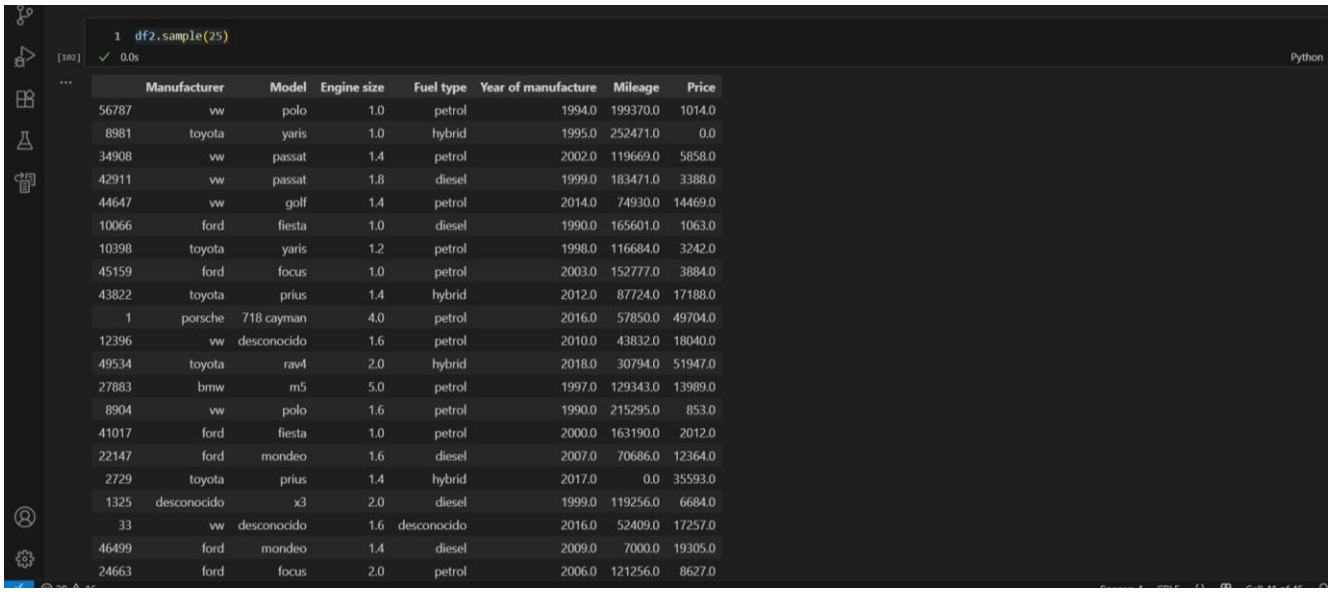


```
df2.isin(valores_invalidos).sum()
```

[101] ✓ 0.3s — Python

```
Manufacturer          0
Model                 0
Engine size           0
Fuel type             0
Year of manufacture   0
Mileage               0
Price                 0
dtype: int64
```

25. Para verificar que la tabla de datos está limpia, se utiliza el comando **df2.sample()**. Es útil para obtener un subconjunto de datos, por ejemplo, para crear conjuntos de entrenamiento y prueba, realizar análisis exploratorios o trabajar con grandes conjuntos de datos de manera más eficiente.



```
df2.sample(25)
```

[102] ✓ 0.0s — Python

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
56787	vw	polo	1.0	petrol	1994.0	199370.0	1014.0
8981	toyota	yaris	1.0	hybrid	1995.0	252471.0	0.0
34908	vw	passat	1.4	petrol	2002.0	119669.0	5858.0
42911	vw	passat	1.8	diesel	1999.0	183471.0	3388.0
44647	vw	golf	1.4	petrol	2014.0	74930.0	14469.0
10066	ford	fiesta	1.0	diesel	1990.0	165601.0	1063.0
10398	toyota	yaris	1.2	petrol	1998.0	116684.0	3242.0
45159	ford	focus	1.0	petrol	2003.0	152777.0	3884.0
43822	toyota	prius	1.4	hybrid	2012.0	87724.0	17188.0
1	porsche	718 cayman	4.0	petrol	2016.0	57850.0	49704.0
12396	vw	desconocido	1.6	petrol	2010.0	43832.0	18040.0
49534	toyota	rav4	2.0	hybrid	2018.0	30794.0	51947.0
27883	bmw	m5	5.0	petrol	1997.0	129343.0	13989.0
8904	vw	polo	1.6	petrol	1990.0	215295.0	853.0
41017	ford	fiesta	1.0	petrol	2000.0	163190.0	2012.0
22147	ford	mondeo	1.6	diesel	2007.0	70686.0	12364.0
2729	toyota	prius	1.4	hybrid	2017.0	0.0	35593.0
1325	desconocido	x3	2.0	diesel	1999.0	119256.0	6684.0
33	vw	desconocido	1.6	desconocido	2016.0	52409.0	17257.0
46499	ford	mondeo	1.4	diesel	2009.0	7000.0	19305.0
24663	ford	focus	2.0	petrol	2006.0	121256.0	8627.0

Conclusiones.

Problemas principales que presentaba la base:

La base contenía valores faltantes (NaN), textos con caracteres extraños, mayúsculas y minúsculas mezcladas, además de espacios innecesarios y nombres inconsistentes entre registros. Esto hacía difícil analizar o comparar los datos correctamente.

Técnicas aplicadas para solucionarlos:

Se reemplazaron los valores NaN, se aplicó una función de limpieza usando expresiones regulares para eliminar símbolos no deseados, normalizar el texto a minúsculas y quitar espacios sobrantes. También se verificó que las columnas quedaran coherentes y con formatos uniformes.

Aprendizaje del proceso:

Aprendí la importancia de la limpieza de datos antes de analizarlos, ya que una base ordenada permite obtener resultados más confiables. Además, comprendí cómo usar Python (con pandas y re) para automatizar la depuración y estandarización de la información.

Link de repositorio de GitHub:

https://github.com/Daniel-Carvajal224/Base-Limpia-Proyecto-Ciencia-de-Datos-Daniel-Carvajal