```
## Objetive
#Our goal is to predict the price of a BMW car considering their unique characteristics
```

```
#We will import google.colab as well as some libreries

from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder,MinMaxScaler
from sklearn import model_selection # model assesment and model selection strategies
from sklearn import metrics # model evaluation metrics

import folium
import plotly.express as px
```

```
Mounted at /content/drive
```

```
#Reading the .csv and take a first look into the dataset
df_bmw=pd.read_csv("/content/drive/MyDrive/ADB/Master Data Science/2. Archivos de clases/Proyecto #1 - BMW/bmw_pricing_v2.csv")
df_bmw.head()
```

	marca	modelo	km	potencia	fecha_registro	tipo_gasolina	color	tipo_coch
0	BMW	118	140411.0	100.0	2012-02-01	diesel	black	convertibl
1	BMW	M4	13929.0	317.0	2016-04-01	petrol	grey	convertibl
2	BMW	320	183297.0	120.0	2012-04-01	diesel	white	convertibl
3	BMW	420	128035.0	135.0	2014-07-01	diesel	red	convertibl
4	BMW	425	97097.0	160.0	2014-12-01	diesel	silver	convertibl

```
df_bmw.info()

# We´ll delete the column named "marca" due to there is any kind of information inside
# We´ll round the column "potencia", change the type of "fecha"
# Out TARGET will be "precio"
# We´ll change some variables from type object to boolean
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4843 entries, 0 to 4842
Data columns (total 18 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   marca                      4841 non-null   object 
 1   modelo                     4840 non-null   object 
 2   km                         4841 non-null   float64
 3   potencia                   4842 non-null   float64
 4   fecha_registro             4842 non-null   object 
 5   tipo_gasolina              4838 non-null   object 
 6   color                      4831 non-null   object 
 7   tipo_coche                 4834 non-null   object 
 8   volante_regulable          4839 non-null   object 
 9   aire_acondicionado         4841 non-null   object 
 10  camara_trasera             4841 non-null   object 
 11  asientos_traseros_plegables  4839 non-null   object 
 12  elevalunas_electrico       4841 non-null   object 
 13  bluetooth                  4839 non-null   object 
 14  gps                        4843 non-null   bool   
 15  alerta_lim_velocidad       4841 non-null   object 
 16  precio                     4837 non-null   float64
 17  fecha_venta                4842 non-null   object 
dtypes: bool(1), float64(3), object(14)
memory usage: 648.1+ KB
```

```
df_bmw.describe().T #Another first look into the dataset

#There are some "km" in negative, as well as with "1M"
#There are some "potential" with zero
#There are some prices really cheaper
```

	count	mean	std	min	25%	50
km	4841.0	140959.347862	60208.534313	-64.0	102884.0	141080.

```
df_bmw.isnull().sum() #There are some nulls that we´ll delete from the dataset
```

```
marca                         2
modelo                        3
km                            2
potencia                      1
fecha_registro                1
tipo_gasolina                 5
color                        12
tipo_coche                    9
volante_regulable             4
aire_acondicionado            2
camara_trasera                2
asientos_traseros_plegables   4
elevalunas_electrico          2
bluetooth                     4
gps                           0
alerta_lim_velocidad          2
precio                        6
fecha_venta                   1
dtype: int64
```

```
len(df_bmw[df_bmw.duplicated()]) #We´ll identify if there is some duplicated values
```

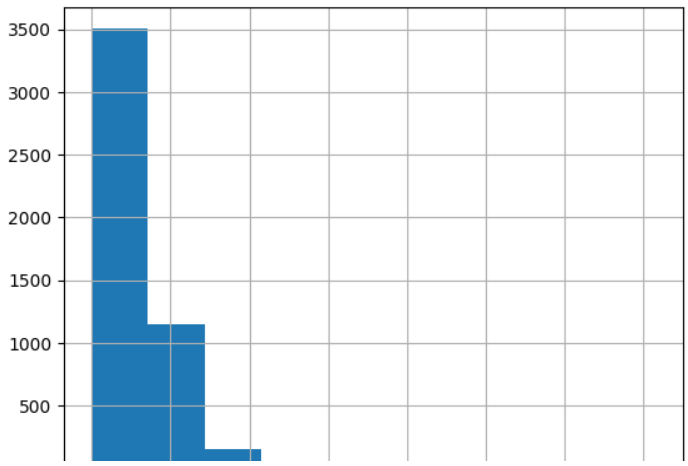
```
0
```

```
#### ANALYSIS EACH COLUMN ####
```

```
## PRICE ##
```

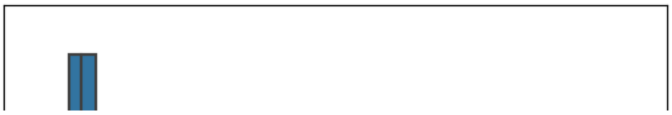
```
df_bmw['precio'].hist()
```

```
<Axes: >
```




```
sns.boxplot(x=df_bmw["precio"])
```

```
<Axes: xlabel='precio'>
```



```
len(df_bmw[df_bmw['precio'] < 500]) #There are 15 cars with value below from 500€
```

```
15
```

```
df_bmw[df_bmw['precio'] < 500]

#Models: 320, 318, 520, 116, 525, 316, X3
```

	marca	modelo	km	potencia	fecha_registro	tipo_gasolir
565	BMW	320	179358.0	120.0	2013-06-01	dies
630	BMW	318	147558.0	105.0	2014-11-01	Na
879	BMW	318	134156.0	105.0	2014-06-01	dies
1255	BMW	320	170381.0	135.0	2013-07-01	dies
1513	BMW	520	358332.0	100.0	2000-10-01	dies
1558	BMW	520	358333.0	100.0	2000-10-01	dies
1832	BMW	116	174524.0	85.0	2014-07-01	dies
2473	BMW	525	230578.0	85.0	1997-07-01	dies
2574	BMW	525	229880.0	85.0	1997-07-01	dies
2611	BMW	525	230264.0	85.0	1997-07-01	dies
2829	BMW	525	439060.0	105.0	1996-10-01	dies
3062	BMW	318	98097.0	85.0	1994-01-01	petr

```
df_bmw.drop(df_bmw[df_bmw['precio'] <500].index, inplace=True) #Eliminating cars with values < 500
```

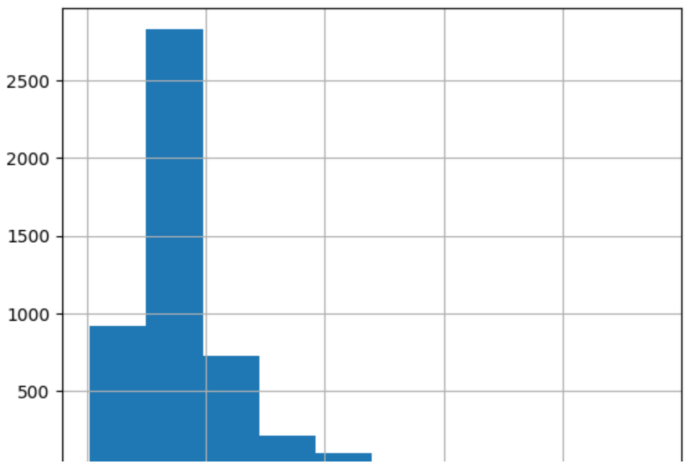
```
len(df_bmw[df_bmw['precio'] > 100000]) #There are 4 cars with value >  100.000€
```

```
2
```

```
df_bmw.drop(df_bmw[df_bmw['precio'] > 100000].index, inplace=True) #Eliminating cars with values > 100.000
```

```
df_bmw['precio'].hist() #Validamos que aún tiene una distribución a la derecha, pero luego normalizaremos el TARGET
```

```
<Axes: >
```




```
## MARCA ##
```

```
df_bmw['marca'].value_counts() #There is only one data
```

```
df_bmw['marca'].value_counts() #There is only one data
```

```
BMW    4824
Name: marca, dtype: int64
```

```
del(df_bmw['marca']) #Eliminating the column
```

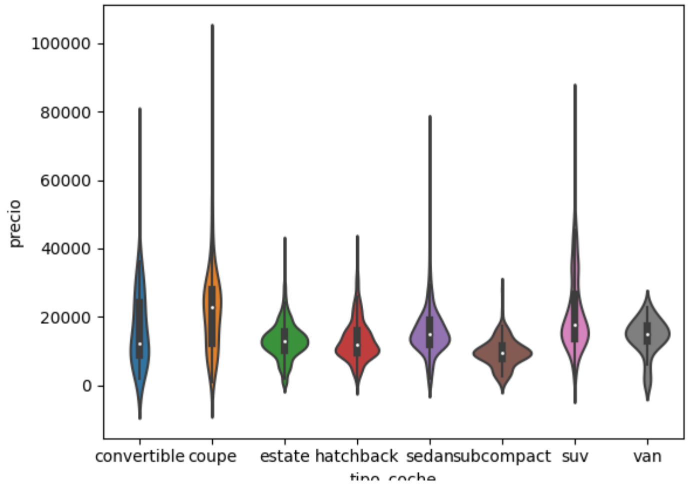
```
## Tipo coche ##
```

```
df_bmw['tipo_coche'].value_counts() #Taking a deep looking

#First we´ll identify the relation between price and type of car
```

```
estate         1598
sedan          1160
suv            1054
hatchback       698
subcompact      113
coupe           104
convertible      47
van              43
Name: tipo_coche, dtype: int64
```

```
sns.violinplot(x="tipo_coche", y="precio", data=df_bmw) #NO NUMERICAL VARIABLE

#We´ll hold on the analysis of this column
```

```
<Axes: xlabel='tipo_coche', ylabel='precio'>
```




```
## Modelo ##
```

```
df_bmw[df_bmw['modelo'].isnull()]
```

	modelo	km	potencia	fecha_registro	tipo_gasolina	colc
173	NaN	146338.0	105.0	2014-01-01	diesel	blac
4766	NaN	115566.0	105.0	2014-01-01	diesel	silv

```
len(df_bmw["modelo"].value_counts())
```

```
76
```

```
df_bmw["modelo"].value_counts()
```

```
320              750
520              631
318              565
X3               436
116              357
                ...
M135               1
225                1
i8                 1
630                1
```

```
214 Gran Tourer      1
Name: modelo, Length: 76, dtype: int64
```

```
df_bmw[(df_bmw["modelo"] == "i8") & (df_bmw["precio"]>50000)]

#Model i8 & price > 50.000 --> outlier
```

modelo	km	potencia	fecha_registro	tipo_gasolina	color

```
df_bmw = df_bmw.drop(df_bmw[(df_bmw["modelo"] == "i8") & (df_bmw["precio"] > 50000)].index)
```

```
df_bmw[(df_bmw["modelo"] == "X3") & (df_bmw["precio"]>50000)]
#Model X3 & price > 50.000 --> outlier
```

modelo	km	potencia	fecha_registro	tipo_gasolina	color	tipo_c

```
df_bmw = df_bmw.drop(df_bmw[(df_bmw["modelo"] == "X3") & (df_bmw["precio"] > 50000)].index)
```

```
df_bmw[(df_bmw["modelo"] == "i8") & (df_bmw["precio"]>1000)]
#Model i8 & price > 1.000 --> unique
```

modelo	km	potencia	fecha_registro	tipo_gasolina	color	tipo_c

```
df_bmw = df_bmw.drop(df_bmw[(df_bmw["modelo"] == "i8") & (df_bmw["precio"] > 1000)].index)
```

```
## Tipo gasolina ##
```

```
df_bmw[df_bmw['tipo_gasolina'].isnull()]
```

	modelo	km	potencia	fecha_registro	tipo_gasolina	color
82	420	54993.0	135.0	2014-03-01	NaN	black
185	320	186697.0	135.0	2012-11-01	NaN	white
444	318	111622.0	100.0	2013-01-01	NaN	black

```
df_bmw["tipo_gasolina"].value_counts()
```

```
diesel           4618
petrol            188
hybrid_petrol       7
Diesel              5
electro             3
Name: tipo_gasolina, dtype: int64
```

```
df_bmw["tipo_gasolina"]=df_bmw["tipo_gasolina"].replace({'diesel':'Diesel'}) #group
df_bmw['tipo_gasolina'] = df_bmw['tipo_gasolina'].replace(['Hybrid Petrol', 'Electro'], 'petrol') #group
df_bmw['tipo_gasolina'] = df_bmw['tipo_gasolina'].replace(['hybrid_petrol', 'electro'], 'petrol') #group
```

```
df_bmw["tipo_gasolina"].value_counts()
```

```
Diesel    4623
petrol     198
Name: tipo_gasolina, dtype: int64
```

```
sns.scatterplot(x="tipo_gasolina", y="precio", data=df_bmw) #NUMERIC VARIABLE
```

```
<Axes: xlabel='tipo_gasolina', ylabel='precio'>
```


```
del(df_bmw['tipo_gasolina']) #we decided to eliminate the column due to we dont see any kind of important data or value that could impact
```

```
## Color ##
```

```
df_bmw[df_bmw['color'].isnull()]
```

	modelo	km	potencia	fecha_registro	color	tipo_coche
239	318	132731.0	100.0	2013-09-01	NaN	estate
834	318	148429.0	100.0	2013-06-01	NaN	estate
855	318	139736.0	100.0	2009-02-01	NaN	estate
864	318	157661.0	100.0	2013-06-01	NaN	estate
884	320	145981.0	122.0	2013-07-01	NaN	estate
904	320	126425.0	120.0	2013-07-01	NaN	estate
939	520	153102.0	140.0	2015-04-01	NaN	estate
1569	318	191804.0	100.0	2013-10-01	NaN	estate
1591	320	130624.0	120.0	2013-07-01	NaN	estate

```
df_bmw['color'].value_counts()
```

```
black     1627
grey      1166
blue       702
white      536
brown      341
silver     325
red         51
beige       41
green       18
orange       6
Name: color, dtype: int64
```

```
sns.scatterplot(x="color", y="precio", data=df_bmw)
```

```
<Axes: xlabel='color', ylabel='precio'>
```

```
del(df_bmw['color']) #we decided to eliminate the column due to we dont see any kind of important data or value that could impact on the

### COPY ###

df_bmw1 = df_bmw.copy() #we generate a copy so we can eliminate the nulls

df_bmw1.dropna(inplace = True)

df_bmw1.isnull().sum() #Checking if there are some nulls after the dropping
```

```
modelo                      0
km                          0
potencia                    0
fecha_registro              0
tipo_coche                  0
volante_regulable           0
aire_acondicionado          0
camara_trasera              0
asientos_traseros_plegables 0
elevalunas_electrico        0
bluetooth                   0
gps                         0
alerta_lim_velocidad        0
precio                      0
fecha_venta                 0
dtype: int64
```

```
### Fecha registro ###

# Change the type of the column to a date format
df_bmw1['fecha_registro']=pd.to_datetime(df_bmw1['fecha_registro'])

# Take the year from the column
df_bmw1['anio_registro']=df_bmw1['fecha_registro'].apply(lambda x:datetime.strftime(x,'%Y'))
df_bmw1['mes_registro'] = df_bmw1['fecha_registro'].apply(lambda x: datetime.strftime(x, '%m'))

df_bmw1["anio_registro"].value_counts() #Checking the format
```

```
2013    1527
2014    1270
2012     817
2015     310
2011     214
2010     103
2008     102
2009      85
2016      82
2006      67
2007      58
2005      52
2004      26
2001      17
2003      16
2017      11
2002       8
2000       8
1999       3
1998       2
1996       1
1995       1
1990       1
1997       1
Name: anio_registro, dtype: int64
```

```
df_bmw1["mes_registro"].value_counts()
```

```
03    464
01    452
10    446
07    445
06    442
05    435
09    416
04    406
02    393
11    343
08    331
```

```
12    209
Name: mes_registro, dtype: int64
```

```
 # Change the type of the column to a date format
df_bmw1['fecha_venta']=pd.to_datetime(df_bmw1['fecha_venta'])

# Take the year from the column
df_bmw1['anio_venta']=df_bmw1['fecha_venta'].apply(lambda x:datetime.strftime(x,'%Y'))
df_bmw1['mes_venta'] = df_bmw1['fecha_venta'].apply(lambda x: datetime.strftime(x, '%m'))
```

```
df_bmw1["anio_venta"].value_counts()
```

```
2018    4778
2007       1
2010       1
2009       1
2008       1
Name: anio_venta, dtype: int64
```

```
del(df_bmw1['anio_venta']) #We´ll eliminate the year of the sale due to its  irrelevant
```

```
df_bmw1["mes_venta"].value_counts()
```

```
05    804
03    726
04    688
06    599
07    532
08    519
02    490
09    220
01    204
Name: mes_venta, dtype: int64
```

```
del(df_bmw1['fecha_registro']) #Eliminating the column
```

```
del(df_bmw1['fecha_venta'])
```

```
### Booleans type ###
```

```
#We´ll change the type of the following variables
```

```
variables_boolenas = ["tipo_coche", "volante_regulable", "aire_acondicionado", "camara_trasera", "asientos_traseros_plegables", "elevacur
```

```
for i in range(len(variables_boolenas)):
    variables_boolenas[i] = bool(variables_boolenas[i])
```

```
### Volante_regulable ###
```

```
df_bmw1["volante_regulable"].value_counts() #We consider it a great variable, with a good distribution
```

```
True     2638
False    2144
Name: volante_regulable, dtype: int64
```

```
sns.violinplot(x="volante_regulable", y="precio", data=df_bmw1) #NO NUMERIC
```

```
<Axes: xlabel='volante_regulable', ylabel='precio'>
```

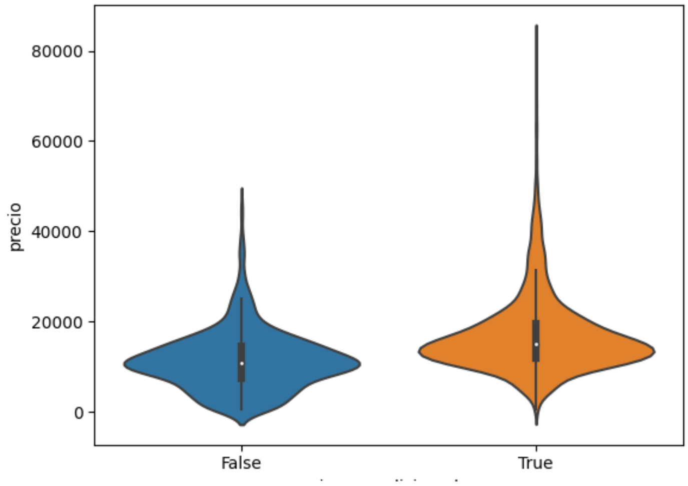

```
### Aire_acondicionado ###
```

```
df_bmw1["aire_acondicionado"].value_counts() #80% - 20%
```

```
True     3807
False     975
Name: aire_acondicionado, dtype: int64
```

```
sns.violinplot(x="aire_acondicionado", y="precio", data=df_bmw1) #NO NUMERIC
```

```
<Axes: xlabel='aire_acondicionado', ylabel='precio'>
```



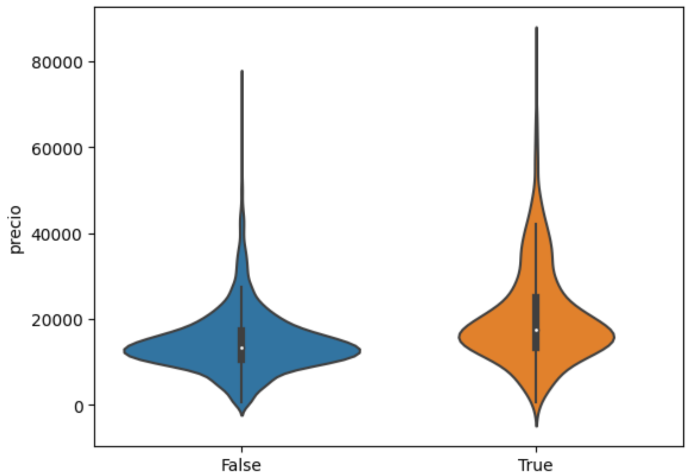

```
### Camara trasera ###
```

```
df_bmw1["camara_trasera"].value_counts() #80% - 20%
```

```
False    3817
True      965
Name: camara_trasera, dtype: int64
```

```
sns.violinplot(x="camara_trasera", y="precio", data=df_bmw1) #NO NUMERIC
```

```
<Axes: xlabel='camara_trasera', ylabel='precio'>
```




```
### Asientos_traseros_plegables ###
```

```
df_bmw1["asientos_traseros_plegables"].value_counts() #80-20
```

```
False    3824
True      958
Name: asientos_traseros_plegables, dtype: int64
```

```
### Elevalunas_electrico ###
```

```
df_bmw1["elevalunas_electrico"].value_counts() #50-50
```

```
False    2575
True     2207
Name: elevalunas_electrico, dtype: int64
```

```
### Bluetooth ###
```

```
df_bmw1["bluetooth"].value_counts() #70-30
```

```
False    3622
True     1160
Name: bluetooth, dtype: int64
```

```
### GPS ###
```

```
df_bmw1["gps"].value_counts() #90-10
```

```
True     4462
False     320
Name: gps, dtype: int64
```

```
### Alerta_lim_velocidad ###
```

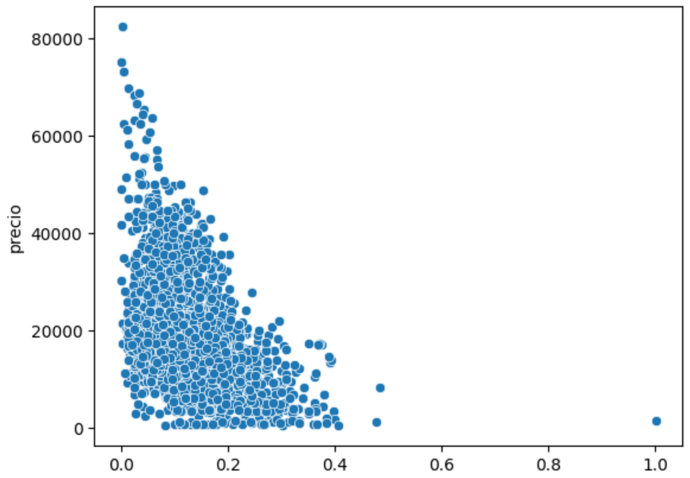
```
df_bmw1["alerta_lim_velocidad"].value_counts() #40-60
```

```
True     2588
False    2194
Name: alerta_lim_velocidad, dtype: int64
```

```
### KM ###
```

```
sns.scatterplot(x="km", y="precio", data=df_bmw1) #NUMERIC
```

```
<Axes: xlabel='km', ylabel='precio'>
```




```
df_bmw1[df_bmw1["km"]<100] #Looking for undervalued data

#We´ll eliminate the negative rows
```

	modelo	km	potencia	tipo_coche	volante_regulable	aire_ac

```
df_bmw1.drop(df_bmw1[df_bmw1['km'] <0].index, inplace=True)
```

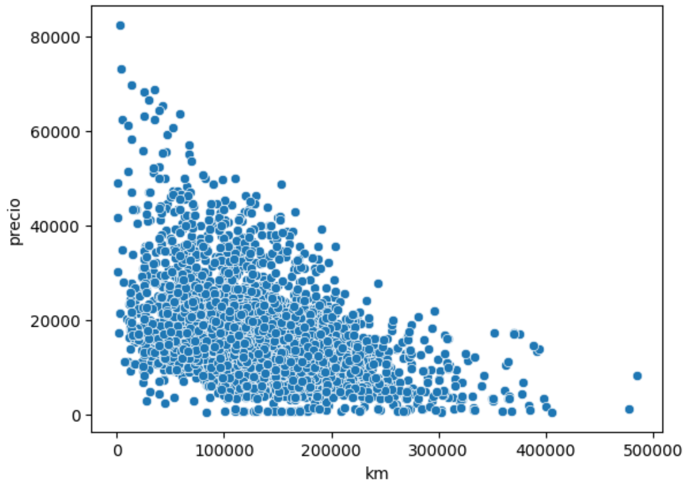
```
df_bmw1[df_bmw1["km"]>400000] #Looking for overvalued data

#We´ll eliminate KM >1.000.000
```

	modelo	km	potencia	tipo_coche	volante_regulable	air
557	520	484615.0	120.0	estate	True	
1573	320	400654.0	110.0	estate	False	
2350	318	477571.0	85.0	hatchback	False	
3198	320	405816.0	100.0	sedan	False	

```
df_bmw1.drop(df_bmw1[df_bmw1['km'] >1000000].index, inplace=True)
```

```
sns.scatterplot(x="km", y="precio", data=df_bmw1) #NUMERIC

#Here we have a correlation: - KM + price
```

<Axes: xlabel='km', ylabel='precio'>




```
### POWER ###
```

```
df_bmw2 = df_bmw1.copy() #Generating a copy
```

```
sns.scatterplot(x="potencia", y="precio", data=df_bmw2) #NUMERIC

#There could be a correlation: + price + power
```

<Axes: xlabel='potencia', ylabel='precio'>

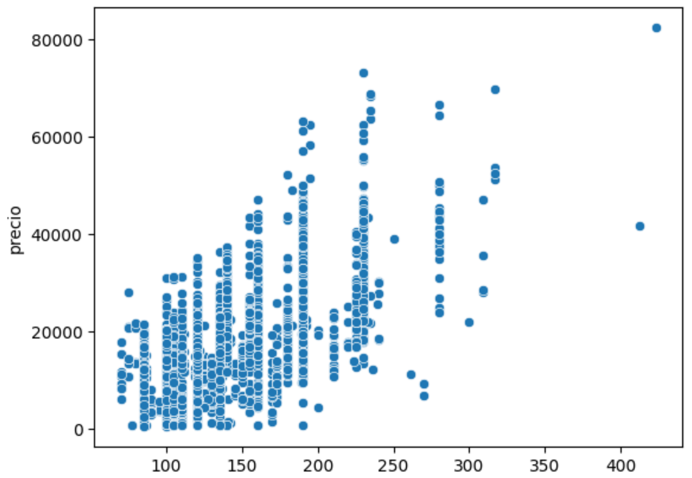
```
df_bmw2[df_bmw2["potencia"]<70] #There are 5 rows with power < 70
```

	modelo	km	potencia	tipo_coche	volante_regulable	aire
1796	i3	152328.0	25.0	hatchback	False	
1925	i3	152470.0	25.0	hatchback	False	
2390	318	170529.0	66.0	hatchback	False	
2771	316	146951.0	66.0	sedan	False	

```
df_bmw2.drop(df_bmw2[df_bmw2["potencia"] < 70].index, inplace=True)
```

```
sns.scatterplot(x="potencia", y="precio", data=df_bmw2)
```

<Axes: xlabel='potencia', ylabel='precio'>




```
df_bmw2[df_bmw2["potencia"]>350] #Looking for outliers
```

	modelo	km	potencia	tipo_coche	volante_regulable	aire
3601	M5	150187.0	412.0	sedan	True	

```
df_bmw2[df_bmw2["modelo"] == "X6 M"] #We´ll eliminate the row with power = 423
#Posible outlier if we compare it with the rest of the rows with same model
```

	modelo	km	potencia	tipo_coche	volante_regulable	aire
3829	X6 M	39725.0	280.0	suv	False	
3986	X6 M	115569.0	280.0	suv	True	
4109	X6 M	67798.0	190.0	suv	True	
4146	X6 M	2970.0	423.0	suv	True	
4166	X6 M	53221.0	180.0	suv	True	
4282	X6 M	90157.0	190.0	suv	True	

```
df_bmw2.drop(df_bmw2[df_bmw2["potencia"] > 420].index, inplace=True)
```

```
df_bmw2[df_bmw2["potencia"]>350] #Checking
```

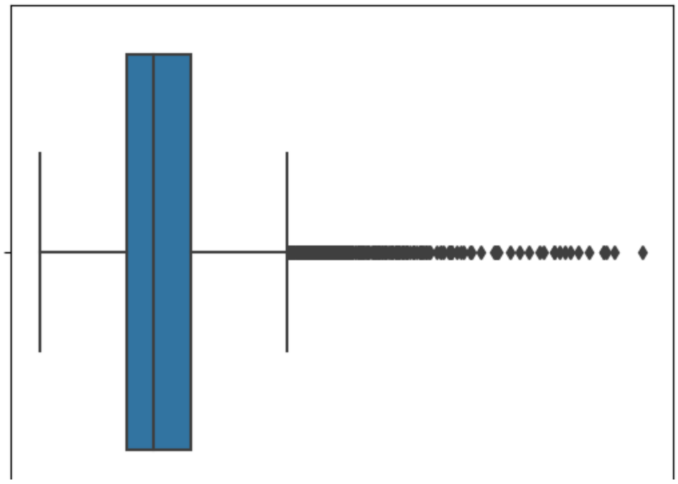
	modelo	km	potencia	tipo_coche	volante_regulable	aire

```
### TARGET ###
```

```
df_bmw3 = df_bmw2.copy()
```

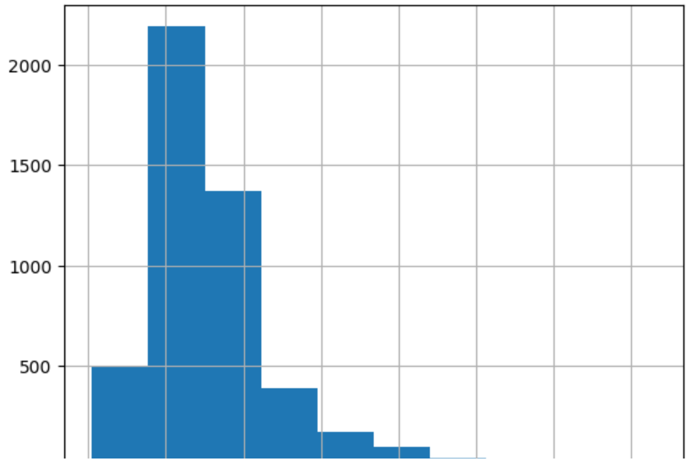
```
sns.boxplot(x=df_bmw3["precio"])

#Looking for outliers
```

```
<Axes: xlabel='precio'>
```



```
df_bmw3["precio"].hist()
```

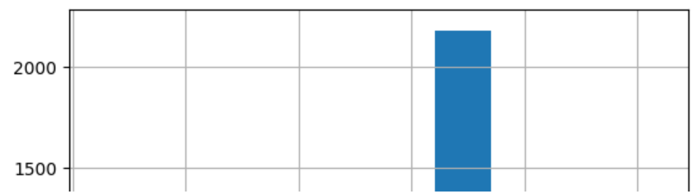
```
<Axes: >
```




```
df_bmw3['precioLN'] = df_bmw3['precio'].apply(lambda x: np.log1p(x))
TARGET_LN = 'precioLN'
```

```
df_bmw3["precioLN"].hist()
```

<Axes: >




```
del(df_bmw3["precioLN"])
```

```
### TRANSFORMATIONS: OHE & MINMAXSCALER ###
```

```
df_bmw4 = df_bmw3.copy()
```

```
df_bmw4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4774 entries, 0 to 4841
Data columns (total 16 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   modelo                     4774 non-null   object 
 1   km                         4774 non-null   float64
 2   potencia                   4774 non-null   float64
 3   tipo_coche                 4774 non-null   object 
 4   volante_regulable          4774 non-null   object 
 5   aire_acondicionado         4774 non-null   object 
 6   camara_trasera             4774 non-null   object 
 7   asientos_traseros_plegables  4774 non-null   object 
 8   elevalunas_electrico       4774 non-null   object 
 9   bluetooth                  4774 non-null   object 
 10  gps                        4774 non-null   bool   
 11  alerta_lim_velocidad       4774 non-null   object 
 12  precio                     4774 non-null   float64
 13  anio_registro              4774 non-null   object 
 14  mes_registro               4774 non-null   object 
 15  mes_venta                  4774 non-null   object 
dtypes: bool(1), float64(3), object(12)
memory usage: 601.4+ KB
```

```
df_bmw4["potencia"] = MinMaxScaler().fit_transform(df_bmw4["potencia"].values.reshape(-1,1)) #We´ll replace with MMS the power column
```

```
df_bmw4["km"] = MinMaxScaler().fit_transform(df_bmw4["km"].values.reshape(-1,1)) #Same action with KM
```

```
# We create a boolean list
variables_boolenas = ["tipo_coche", "volante_regulable", "aire_acondicionado", "camara_trasera", "asientos_traseros_plegables", "elevalur

for column in variables_boolenas:
    # OHE
    encoded_columns = pd.get_dummies(df_bmw4[column], prefix=column)

    # Concat the DataFrame with the transformed column
    df_bmw4 = pd.concat([df_bmw4, encoded_columns], axis=1)

# Eliminating the original boolean columns
df_bmw4 = df_bmw4.drop(variables_boolenas, axis=1)
```

```
df_bmw4.head()
```

	km	potencia	precio	tipo_coche_convertible	tipo_coche_co
0	0.289039	0.087719	11300.0	1	
1	0.027787	0.722222	69700.0	1	
2	0.377621	0.146199	10200.0	1	
3	0.263476	0.190058	25100.0	1	
4	0.199573	0.263158	33400.0	1	

```
### MODELING ###
```

```
#We decide to use aleatory to split the model

p_dev = 0.70 # % of train

df_bmw4['is_train'] = np.random.uniform(0, 1, len(df_bmw4)) <= p_dev
dev_df_bmw4, val_df_bmw4 = df_bmw4[df_bmw4['is_train']==True], df_bmw4[df_bmw4['is_train']==False]
df_bmw4 = df_bmw4.drop('is_train', 1)

print("Ejemplos usados para entrenar: ", len(dev_df_bmw4))
print("Ejemplos usados para validación: ", len(val_df_bmw4))
```

```
Ejemplos usados para entrenar:  3277
Ejemplos usados para validación:  1497
<ipython-input-117-55dee41ae9a5>:7: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the arg
  df_bmw4 = df_bmw4.drop('is_train', 1)
```

```
### Asignación de atributos y target a las variables X e Y ###


dev_df_bmw4_X = dev_df_bmw4.drop('precio', axis=1)
dev_df_bmw4_y = dev_df_bmw4[['precio']]


val_df_bmw4_X = val_df_bmw4.drop('precio', axis=1)
val_ddf_bmw4_y = val_df_bmw4[['precio']]


### Random hold out ###


X_train, X_test, y_train, y_test = model_selection.train_test_split(
                                        dev_df_bmw4_X, # X
                                        dev_df_bmw4_y, # y
                                        test_size = 0.20,
                                        random_state = 1279
                                     )


### Previous checks ###


dev_df_bmw4_X.head().T
```

	1	3	4	5	7
km	0.027787	0.263476	0.199573	0.313703	0.23770
potencia	0.722222	0.190058	0.263158	0.453216	0.10233
tipo_coche_convertible	1	1	1	1	1
tipo_coche_coupe	0	0	0	0	C
tipo_coche_estate	0	0	0	0	C
...	...	...	...	...	...
modelo_X6	0	0	0	0	C
modelo_X6 M	0	0	0	0	C
modelo_Z4	0	0	0	0	C

```
X_train.info(verbose=False)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2621 entries, 3207 to 1464
Columns: 146 entries, km to is_train
dtypes: bool(1), float64(2), uint8(143)
memory usage: 430.0 KB
```

```
X_test.info(verbose=False)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 656 entries, 1347 to 846
Columns: 146 entries, km to is_train
dtypes: bool(1), float64(2), uint8(143)
memory usage: 107.6 KB
```

```
X_train.describe().T.head()
```

	count	mean	std	min	25%
km	2621.0	0.287988	0.120934	0.000281	0.208839
potencia	2621.0	0.171648	0.110721	0.000000	0.087719
tipo_coche_convertible	2621.0	0.011446	0.106392	0.000000	0.000000
tipo_coche_coupe	2621.0	0.020221	0.140783	0.000000	0.000000

```
X_test.describe().T.head()
```

	count	mean	std	min	25%
km	656.0	0.290215	0.117748	0.000475	0.220167
potencia	656.0	0.173602	0.123289	0.000000	0.087719
tipo_coche_convertible	656.0	0.006098	0.077908	0.000000	0.000000
tipo_coche_coupe	656.0	0.030488	0.172056	0.000000	0.000000

```
y_train.describe().T.head()
```

count	mean	std	min	25%	50%

```
y_test.describe().T.head()
```

count	mean	std	min	25%	50%

```
### MODEL DEFINITION - REGRESSION ###
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics

from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
import graphviz
import pydotplus

!conda install python-graphviz -y
!conda install pydot -y
```

```
/bin/bash: line 1: conda: command not found
/bin/bash: line 1: conda: command not found
```

```
X_train, X_test, y_train, y_test = train_test_split(dev_df_bmw4_X,dev_df_bmw4_y,test_size=0.3,random_state=23)
```

```
lin_reg= LinearRegression()
lin_reg.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
y_pred = lin_reg.predict(X_test)
```

```
y_test.head()
```

	precio
1717	8900.0

```
df_resultados = pd.DataFrame({'Actual': y_test, 'Predicted':y_pred})
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent
call last)
<ipython-input-137-fa470504ef3a> in <cell line: 1>()
----> 1 df_resultados = pd.DataFrame({'Actual': y_test,
'Predicted':y_pred})

3 frames
/usr/local/lib/python3.10/dist-
packages/pandas/core/internals/construction.py in
_extract_index(data)
    651                     raw_lengths.append(len(val))
    652                 elif isinstance(val, np.ndarray) and val.ndim >
1:
```

```
print ("MAE", metrics.mean_absolute_error(y_test, y_pred))
```

```
MAE 16745316344196.914
```

```
print ("MSE", metrics.mean_squared_error(y_test, y_pred))
```

```
MSE 3.917628648694983e+28
```

```
print("RMSE",np.sqrt(metrics.mean_squared_error(y_test, y_pred, squared = False)))
```

```
RMSE 14068759.860373845
```

```
df_resultados["dif"]=df_resultados["Predicted"]-df_resultados["Actual"]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent
call last)
<ipython-input-141-5d3445ad96f7> in <cell line: 1>()
----> 1 df_resultados["dif"]=df_resultados["Predicted"]-
df_resultados["Actual"]
```

```
df_resultados.sort_values(by="dif")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent
call last)
<ipython-input-142-e57d28c6b154> in <cell line: 1>()
----> 1 df_resultados.sort_values(by="dif")

NameError: name 'df_resultados' is not defined
```

```
df_resultados.hist("dif")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent
call last)
<ipython-input-143-1bbb1a882813> in <cell line: 1>()
----> 1 df_resultados.hist("dif")

NameError: name 'df_resultados' is not defined
```

```
### Model definition - RandomForestRegression ###
```

```
from sklearn.ensemble import RandomForestRegressor
```

```
modelo_rf = RandomForestRegressor(n_estimators = 20, random_state = 2022)
modelo_rf.fit(dev_df_bmw4_X, dev_df_bmw4_y)
```

```
<ipython-input-146-43a17aca18df>:2: DataConversionWarning: A column-
  modelo_rf.fit(dev_df_bmw4_X, dev_df_bmw4_y)
▾                 RandomForestRegressor
```

```
predicciones_rf = modelo_rf.predict(X_test)
predicciones_rf
```

```
       26085., 19245., 13975., 10120., 39870.,  5240., 13320.,  8875.,
       15915.,  9920., 19860.,  9665., 39855., 13900.,  4455., 14710.,
       12715., 10385., 10480.,  6855., 26345., 16800., 50715., 10365.,
        7655., 13470., 14080., 14110., 19210., 13130., 19530., 13600.,
       11110., 18145., 14840., 17675., 12420., 16540., 22160., 10035.,
       13400., 29695., 19515., 10310., 36895., 15140., 26260., 19290.,
       18890., 11695., 12815., 11110., 37820., 10795.,  3710., 13790.,
       14215., 19820.,  9905., 10760., 16990.,  6435., 14345., 22205.,
       14150., 12805., 14830., 20645., 11545., 10010.,  3335.,  6455.,
        7655., 19730., 19160., 19480.,  5165., 15625., 16565., 15335.,
        8670., 25415., 12975., 22820., 13245., 15620., 17375., 13330.,
       13350., 11220., 14340.,  8510., 10705., 15690., 14595., 27955.,
       12800., 12450., 12945., 11825., 29825., 13965., 12395., 28330.,
       14090., 13135., 13840., 10110., 21360., 21200., 13085., 17295.,
       14395., 19770., 18580.,  9120., 23505., 13235., 14495., 14705.,
       12475., 19200., 13760., 12760., 20850., 10155.,  8735.,  7070.,
        4615.,  1915., 12370.,  7790., 15185., 12985., 12945., 31170.,
       29970., 12715., 17100., 37575.,  9110., 15390.,  9595., 13775.,
       11170.,  9725., 14405., 13130., 12930., 12165., 17385., 24815.,
       20805.,  8865., 12715., 26480., 20145., 12865., 26555., 14880.,
       13810.,  1335., 20205., 23720., 11330., 16375., 14330., 18360.,
       18495., 19575.,  6325.,  7880., 10515., 24380., 18380., 22600.,
       22090., 14530., 11525., 15340., 11825., 14985., 15925., 31365.,
       15355., 13645., 19425., 20235., 11635., 11705., 17740., 30445.,
       12650., 15540., 11065., 10240.,  1300., 17960., 12355.,  8320.,
       12160.,  8670., 20490., 27325., 11680., 13925., 19875., 10710.,
       19950., 25175., 14215.,  9835., 12775., 18375., 18180., 17205.,
       22360.,  7035., 11310.,  4320., 11685., 29230., 17005., 15445.,
       12680., 12960., 20155., 38475., 18790.,  9850., 10685., 15255.,
       19290., 20675., 10295., 17825., 18250., 15135., 15130., 33940.,
       12985., 13570., 10930., 36620., 14590., 17690., 20065., 30270.,
       47785., 25890., 15260., 18135.,  5415., 11805., 11265.,  9080.,
        5055., 10775., 12095., 24760., 14320., 22610., 30135., 17190.,
       28440.,  1290., 26175., 15705., 22170., 13235., 10020., 11040.,
       20260.,  9505., 14375., 10385., 28690., 11320., 15950., 13675.,
        6690., 11305., 18085., 15530., 11920.,  5335., 28005.,  9245.,
       23395., 12590., 11380., 22165., 14405., 13105., 13720., 11965.,
       10555., 15540.,  6030.,  9215.,  8545., 20955., 38705., 29915.,
       14940.,  9185., 17160., 10660., 23890., 14625., 14720.,  8890.,
       21100., 29570., 28585., 11905., 33115.,  7830., 17820.,  9630.,
       25705., 14170., 18400., 11780., 28270., 20885., 12515., 11330.,
       14240., 16855., 16585.,  9940., 22375., 10950., 11735., 18595.,
        9965., 13075., 11725.,  8525., 12035., 13635.,  9250., 24090.,
       22160., 22310., 17460., 13375., 12070., 18000., 11085., 14495.,
       18150., 16960., 16000.,  7720.,  9375., 18185., 17460., 18340.,
        9950.,  6865.,  4225., 18040., 16525., 21835., 22105., 13225.,
       16595., 12155., 13105., 19630., 21740., 16100., 12615.,  5610.,
       14865., 16540.,  6200., 19360., 14665., 11880., 20855., 14445.,
       19045., 21085., 13320., 13040., 18555., 14195., 11915., 39200.,
       11300.,  4635., 12925., 17595., 16875., 19670., 26295., 12405.,
        8350., 11975.,  9365., 14030.,  9430., 11685., 12460., 16220.,
       11595., 18805., 13210., 11825.,  3300.,  7690., 12475., 13795.,
       19815., 33670., 10980., 10450., 13140., 41315., 11665., 21435.,
       12875., 14360., 11975., 11640., 14885.,  9535.,  9335., 14060.,
       12405., 16480., 15720., 14965., 10275.,  5380., 10230., 35120.,
       10120., 20100., 17010., 54620., 17005., 12470., 25780.,  9415.,
        7775., 14365., 10790., 13260., 14300., 13895., 13250.,  9990.,
       29705., 25690., 17995., 11020., 22305., 13010., 12340., 21065.,
       13455., 11280.,  5250.,  9935., 26680., 12560., 25575., 12405.,
```

```
comparaciones = pd.DataFrame(X_test)
comparaciones = comparaciones.assign(Precio_Real = y_test)
comparaciones = comparaciones.assign(Precio_Prediccion = predicciones_rf.flatten().tolist())
print(comparaciones)
```

```
...        ...       ...                    ...              ...
4691  0.188138  0.248538                      0                0
3007  0.373422  0.190058                      0                0
1496  0.238442  0.146199                      0                0
2245  0.188894  0.087719                      0                0
3590  0.371552  0.321637                      0                0

      tipo_coche_estate  tipo_coche_hatchback  tipo_coche_sedan  \
1717                  1                     0                 0
2594                  0                     0                 1
```

```
2245                0                   1               0
3590                0                   0               1

      tipo_coche_subcompact  tipo_coche_suv  tipo_coche_van  ...  modelo_X5  \
1717                      0               0               0  ...          0
2594                      0               0               0  ...          0
1880                      0               0               0  ...          0
2027                      0               0               0  ...          0
3293                      0               0               0  ...          0
...                     ...             ...             ...  ...        ...
4691                      0               1               0  ...          1
3007                      0               0               0  ...          0
1496                      0               0               0  ...          0
2245                      0               0               0  ...          0
3590                      0               0               0  ...          0

      modelo_X5 M  modelo_X5 M50  modelo_X6  modelo_X6 M  modelo_Z4  \
1717            0              0          0            0          0
2594            0              0          0            0          0
1880            0              0          0            0          0
2027            0              0          0            0          0
3293            0              0          0            0          0
...           ...            ...        ...          ...        ...
4691            0              0          0            0          0
3007            0              0          0            0          0
1496            0              0          0            0          0
2245            0              0          0            0          0
3590            0              0          0            0          0

      modelo_i3  is_train  Precio_Real  Precio_Prediccion
1717          0      True       8900.0             9045.0
2594          0      True      11300.0            11740.0
1880          0      True      17300.0            18060.0
2027          0      True      10200.0            10015.0
3293          0      True      12300.0            12030.0
...         ...       ...          ...                ...
4691          0      True      23800.0            23625.0
3007          0      True      15400.0            16060.0
1496          0      True      12700.0            10865.0
2245          0      True      12300.0            13065.0
3590          0      True      18200.0            17970.0
```

```
### RMSE modelo de ar ###


from sklearn.metrics import mean_squared_error, r2_score


rmse_rf = mean_squared_error(
        y_true  = y_test,
        y_pred  = predicciones_rf,
        squared = False
       )
print(f"El error (rmse) de test es: {rmse_rf}")
```

```
El error (rmse) de test es: 1335.4106562907753
```

✓ 0 s completado a las 8:59