P12_Metodos numéricos

July 24, 2024

El servicio de venta de autos usados Rusty Bargain está desarrollando una aplicación para atraer nuevos clientes. Gracias a esa app, puedes averiguar rápidamente el valor de mercado de tu coche. Tienes acceso al historial: especificaciones técnicas, versiones de equipamiento y precios. Tienes que crear un modelo que determine el valor de mercado. A Rusty Bargain le interesa: - la calidad de la predicción; - la velocidad de la predicción; - el tiempo requerido para el entrenamiento

0.1 Preparación de datos

[1]:
```
pip install category_encoders
```

```
Requirement already satisfied: category_encoders in
/opt/conda/envs/python3/lib/python3.9/site-packages (2.6.3)
Requirement already satisfied: numpy>=1.14.0 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from category_encoders)
(1.21.2)
Requirement already satisfied: scikit-learn>=0.20.0 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from category_encoders)
(0.24.1)
Requirement already satisfied: scipy>=1.0.0 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from category_encoders)
(1.10.1)
Requirement already satisfied: statsmodels>=0.9.0 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from category_encoders)
(0.13.2)
Requirement already satisfied: pandas>=1.0.5 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from category_encoders)
(1.2.4)
Requirement already satisfied: patsy>=0.5.1 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from category_encoders)
(0.5.6)
Requirement already satisfied: python-dateutil>=2.7.3 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from
pandas>=1.0.5->category_encoders) (2.9.0)
Requirement already satisfied: pytz>=2017.3 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from
pandas>=1.0.5->category_encoders) (2024.1)
Requirement already satisfied: six in
/opt/conda/envs/python3/lib/python3.9/site-packages (from
```

```
patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=0.11 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from scikit-
learn>=0.20.0->category_encoders) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from scikit-
learn>=0.20.0->category_encoders) (3.5.0)
Requirement already satisfied: packaging>=21.3 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from
statsmodels>=0.9.0->category_encoders) (24.0)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import category_encoders as ce
import math
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor
```

[3]:
```
data = pd.read_csv('/datasets/car_data.csv')
data.describe()
```

[3]:

	Price	RegistrationYear	Power	Mileage	\
count	354369.000000	354369.000000	354369.000000	354369.000000	
mean	4416.656776	2004.234448	110.094337	128211.172535	
std	4514.158514	90.227958	189.850405	37905.341530	
min	0.000000	1000.000000	0.000000	5000.000000	
25%	1050.000000	1999.000000	69.000000	125000.000000	
50%	2700.000000	2003.000000	105.000000	150000.000000	
75%	6400.000000	2008.000000	143.000000	150000.000000	
max	20000.000000	9999.000000	20000.000000	150000.000000	

	RegistrationMonth	NumberOfPictures	PostalCode
count	354369.000000	354369.0	354369.000000
mean	5.714645	0.0	50508.689087
std	3.726421	0.0	25783.096248
min	0.000000	0.0	1067.000000
25%	3.000000	0.0	30165.000000
50%	6.000000	0.0	49413.000000
75%	9.000000	0.0	71083.000000
max	12.000000	0.0	99998.000000

Podemos observar algunos datos y discrepancias en una primera revisión de los datos: * Existen vehículos con precio = 0 lo cual no es posible * Existen vehículos con año de registro = 9999 cómo máximo y 1000 cómo mínimo, estos datos tampoco son posibles * Nota: el primer automóvil con motor de combustión interna fue patentado en 1886 por Karl Benz, es considerado el primer atomóvil moderno * Existen vehículos con 0 caballos de vapor y con 20,000 CV, lo cual no es posible * Nota: El vehículo con mayor CV en la actualidad es el Rimac Nevera, con 1914 HP, equivalente a 1,941 CV. * Existen vehículos que tienen como mes de registro el mes 0, lo cual no es posible (debido a que el máximo es 12 y contamos de manera natural) * Ningún vehículo fue cargado con fotografías, el máximo es = 0

Trataremos estas particularidades más adelante

[4]:
```
data.columns
```

[4]:
```
Index(['DateCrawled', 'Price', 'VehicleType', 'RegistrationYear', 'Gearbox',
       'Power', 'Model', 'Mileage', 'RegistrationMonth', 'FuelType', 'Brand',
       'NotRepaired', 'DateCreated', 'NumberOfPictures', 'PostalCode',
       'LastSeen'],
      dtype='object')
```

[5]:
```
data.head()
```

[5]:

	DateCrawled	Price	VehicleType	RegistrationYear	Gearbox	Power	Model	Mileage	RegistrationMonth	FuelType	Brand	NotRepaired	DateCreated	NumberOfPictures	PostalCode	LastSeen
0	24/03/2016 11:52	480	NaN	1993	manual	0	golf	150000	0	petrol	volkswagen	NaN	24/03/2016 00:00	0	70435	07/04/2016 03:16
1	24/03/2016 10:58	18300	coupe	2011	manual	190	NaN	125000	5	gasoline	audi	yes	24/03/2016 00:00	0	66954	07/04/2016 01:46
2	14/03/2016 12:52	9800	suv	2004	auto	163	grand	125000	8	gasoline	jeep	NaN	14/03/2016 00:00	0	90480	05/04/2016 12:47
3	17/03/2016 16:54	1500	small	2001	manual	75	golf	150000	6	petrol	volkswagen	no	17/03/2016 00:00	0	91074	17/03/2016 17:40
4	31/03/2016 17:25	3600	small	2008	manual	69	fabia	90000	7	gasoline	skoda	no	31/03/2016 00:00	0	60437	06/04/2016 10:17

[6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 354369 entries, 0 to 354368
Data columns (total 16 columns):
```

```
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   DateCrawled        354369 non-null  object
 1   Price              354369 non-null  int64
 2   VehicleType        316879 non-null  object
 3   RegistrationYear   354369 non-null  int64
 4   Gearbox            334536 non-null  object
 5   Power              354369 non-null  int64
 6   Model              334664 non-null  object
 7   Mileage            354369 non-null  int64
 8   RegistrationMonth  354369 non-null  int64
 9   FuelType           321474 non-null  object
 10  Brand              354369 non-null  object
 11  NotRepaired        283215 non-null  object
 12  DateCreated        354369 non-null  object
 13  NumberOfPictures   354369 non-null  int64
 14  PostalCode         354369 non-null  int64
 15  LastSeen           354369 non-null  object
dtypes: int64(7), object(9)
memory usage: 43.3+ MB
```

0.1.1 Explorando columnas

0.1.2 Descripción de los datos

Contamos con 354,369 registros, 16 columnas de las cuales 5 de ellas cuentan con datos Nulos, entre ellas: * VehicleType * Gearbox * Model * FuelType * NotRepaired

Descripción de las columnas

- *DateCrawled* — fecha en la que se descargó el perfil de la base de datos
- *VehicleType* — tipo de carrocería del vehículo
- *RegistrationYear* — año de matriculación del vehículo
- *Gearbox* — tipo de caja de cambios
- *Power* — potencia (CV)
- *Model* — modelo del vehículo
- Mileage — kilometraje (medido en km de acuerdo con las especificidades regionales del conjunto de datos)
- *RegistrationMonth* — mes de matriculación del vehículo
- *FuelType* — tipo de combustible
- *Brand* — marca del vehículo
- *NotRepaired* — vehículo con o sin reparación
- *DateCreated* — fecha de creación del perfil
- *NumberOfPictures* — número de fotos del vehículo
- *PostalCode* — código postal del propietario del perfil (usuario)
- *LastSeen* — fecha de la última vez que el usuario estuvo activo

Por la relevancia y la información que proveen las siguientes columnas, hemos decidido que no son relevantes para el modelo: * DateCrawled * RegistrationMonth * LastSeen * PostalCode *

DateCreated * NumberOfPictures

Normalizando nombres de columnas

[7]:
```
data.columns = data.columns.str.lower()
data.rename(columns={'datecrawled':'date_crawled','vehicletype':'vehicle_type',
                     'registrationyear':'registration_year','registrationmonth':
 ↪'registration_month',
                    'fueltype':'fuel_type','notrepaired':'not_repaired',
                    'datecreated':'date_created','numberofpictures':
 ↪'number_of_pictures',
                    'postalcode':'postal_code','lastseen':
 ↪'last_seen'},inplace=True)
print(data.columns)
data.head(5)
```

```
Index(['date_crawled', 'price', 'vehicle_type', 'registration_year', 'gearbox',
       'power', 'model', 'mileage', 'registration_month', 'fuel_type', 'brand',
       'not_repaired', 'date_created', 'number_of_pictures', 'postal_code',
       'last_seen'],
      dtype='object')
```

[7]:

	date_crawled	price	vehicle_type	registration_year	gearbox	power	\
0	24/03/2016 11:52	480	NaN	1993	manual	0	
1	24/03/2016 10:58	18300	coupe	2011	manual	190	
2	14/03/2016 12:52	9800	suv	2004	auto	163	
3	17/03/2016 16:54	1500	small	2001	manual	75	
4	31/03/2016 17:25	3600	small	2008	manual	69	

	model	mileage	registration_month	fuel_type	brand	not_repaired	\
0	golf	150000	0	petrol	volkswagen	NaN	
1	NaN	125000	5	gasoline	audi	yes	
2	grand	125000	8	gasoline	jeep	NaN	
3	golf	150000	6	petrol	volkswagen	no	
4	fabia	90000	7	gasoline	skoda	no	

	date_created	number_of_pictures	postal_code	last_seen
0	24/03/2016 00:00	0	70435	07/04/2016 03:16
1	24/03/2016 00:00	0	66954	07/04/2016 01:46
2	14/03/2016 00:00	0	90480	05/04/2016 12:47
3	17/03/2016 00:00	0	91074	17/03/2016 17:40
4	31/03/2016 00:00	0	60437	06/04/2016 10:17

Eliminando filas duplicadas

[8]:
```
print(data[data.duplicated()]['price'].count())
data.drop_duplicates(inplace=True)
```

```
print("Comprobando si existen datos duplicados despúes de la operación :
  ↪",data[data.duplicated()]['price'].count())
```

```
262
Comprobando si existen datos duplicados despúes de la operación : 0
```

[9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 354107 entries, 0 to 354368
Data columns (total 16 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   date_crawled        354107 non-null  object
 1   price               354107 non-null  int64
 2   vehicle_type        316623 non-null  object
 3   registration_year   354107 non-null  int64
 4   gearbox             334277 non-null  object
 5   power               354107 non-null  int64
 6   model               334406 non-null  object
 7   mileage             354107 non-null  int64
 8   registration_month  354107 non-null  int64
 9   fuel_type           321218 non-null  object
 10  brand               354107 non-null  object
 11  not_repaired        282962 non-null  object
 12  date_created        354107 non-null  object
 13  number_of_pictures  354107 non-null  int64
 14  postal_code         354107 non-null  int64
 15  last_seen           354107 non-null  object
dtypes: int64(7), object(9)
memory usage: 45.9+ MB
```

Pre- procesando columnas en esta sección buscaremos discriminar registros que no aporten valor al modelo, por ejemplo: * Vehículos sin modelo ni tipo de vehículo * Vehículos sin precio ni tipo de vehículo

[10]:
```
data.head()
```

[10]:

	date_crawled	price	vehicle_type	registration_year	gearbox	power	model	mileage	registration_month	fuel_type	brand	not_repaired
0	24/03/2016 11:52	480	NaN	1993	manual	0	golf	150000	0	petrol	volkswagen	NaN
1	24/03/2016 10:58	18300	coupe	2011	manual	190						
2	14/03/2016 12:52	9800	suv	2004	auto	163						
3	17/03/2016 16:54	1500	small	2001	manual	75						
4	31/03/2016 17:25	3600	small	2008	manual	69						

```
1    NaN   125000                  5  gasoline        audi          yes
2  grand   125000                  8  gasoline        jeep          NaN
3   golf   150000                  6    petrol  volkswagen           no
4  fabia    90000                  7  gasoline       skoda           no

       date_created  number_of_pictures  postal_code         last_seen
0  24/03/2016 00:00                   0        70435  07/04/2016 03:16
1  24/03/2016 00:00                   0        66954  07/04/2016 01:46
2  14/03/2016 00:00                   0        90480  05/04/2016 12:47
3  17/03/2016 00:00                   0        91074  17/03/2016 17:40
4  31/03/2016 00:00                   0        60437  06/04/2016 10:17
```

Reemplazando los precios = 0 por la mediana (2700)

[11]:
```
data.loc[data['price']== 0,'price']= data['price'].median()
#Comprobando los cambios
data[data['price']==0]['price'].any()
```

[11]:
```
False
```

Trabajando con el año de registro de los vehículos ('registration_year')

primero estudiemos los valores únicos de años registrados

[12]:
```
data['registration_year'].sort_values().unique()
```

[12]:
```
array([1000, 1001, 1039, 1111, 1200, 1234, 1253, 1255, 1300, 1400, 1500,
       1600, 1602, 1688, 1800, 1910, 1915, 1919, 1920, 1923, 1925, 1927,
       1928, 1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938,
       1940, 1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950,
       1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961,
       1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972,
       1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983,
       1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994,
       1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005,
       2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016,
       2017, 2018, 2019, 2066, 2200, 2222, 2290, 2500, 2800, 2900, 3000,
       3200, 3500, 3700, 3800, 4000, 4100, 4500, 4800, 5000, 5300, 5555,
       5600, 5900, 5911, 6000, 6500, 7000, 7100, 7500, 7800, 8000, 8200,
       8455, 8500, 8888, 9000, 9229, 9450, 9996, 9999])
```

Observamos que: * El año más cercano al mínimo de la historia automotriz moderna(1886) es el 1910, lo tomaremos como un valor mínimo. * El año más cercano a la actualidad (2024) es el 2019, lo tomaremos como un valor máximo.

Descartando los registros con año de registro menores a 1910 y mayores a 2019

[13]:
```
data = data.loc[(data['registration_year']>=1910) &␣
 ↪(data['registration_year']<=2019)]
```

```
#Verificando los cambios
print('min year: ',data['registration_year'].min())
print('max year: ',data['registration_year'].max())
```

```
min year:  1910
max year:  2019
```

Trabajando con la columna 'Power' El primer automóvil con motor de combustión interna fue patentado en 1886 por Karl Benz, es considerado el primer atomóvil moderno y contaba con 0.74 CV

Reemplazando los valores de los vehículos con Power=0 y Power > 1941 CV por la mediana

[14]:
```
data['power'].max()
```

[14]:
```
20000
```

[15]:
```
mask_min= data['power']<1
mask_max= data['power']>1940
median = data['power'].median().astype(int)

data.loc[mask_max,'power']=median
data.loc[mask_min,'power']=median
```

[16]:
```
print('min power: ',data['power'].min())
print('max power: ',data['power'].max())
```

```
min power:  1
max power:  1937
```

Trabajando con la columna 'vehicle_type'

[17]:
```
data['vehicle_type'].isna().sum()
```

[17]:
```
37313
```

Observamos que la columna cuenta con 37,313 registros donde el tipo de vehículo no fue cargado, lo vamos a reemplazar por 'unknown'

Reemplazando valores NaN en el tipo de vehículo por 'unknown'

[18]:
```
data['vehicle_type'] = data['vehicle_type'].fillna('unknown')
```

Trabajando con la columna 'model'

[19]:
```
print(data['model'].isna().sum())
print(round((data['model'].isna().sum()*100)/len(data),2))
```

```
19626
5.55
```

Podemos observar que hay 19,626 vehículos que no cuentan con el modelo, para propósitos de venta y de calcular el precio, se vuelve una característica indispensable, estos registros representan el 5.55% del total de los datos, por lo que decidimos eliminarlos

[20]:
```
data = data[~data['model'].isna()]
#Verificando los cambios
data['model'].isna().sum()
```

[20]:
```
0
```

Trabajado con la columna 'fuel_type'

[21]:
```
data['fuel_type'].isna().sum()
```

[21]:
```
25662
```

[22]:
```
#Realizamos una agrupación y transformación para sobreescribir los datos nulos␣
 ↪por la moda de cada modelo en su columna fuel_type
data['fuel_type'] = data['fuel_type'].fillna(data.groupby('model')['fuel_type'].
 ↪transform(lambda x: x.value_counts().idxmax()))
print("Cantidad de datos nulos en fuel_type : ",data['fuel_type'].isna().sum())
data['fuel_type'].unique()
```

```
Cantidad de datos nulos en fuel_type :  0
```

[22]:
```
array(['petrol', 'gasoline', 'lpg', 'other', 'hybrid', 'cng', 'electric'],
      dtype=object)
```

Trabajando con la columna 'not_repaired"

[23]:
```
#Verificando el precio promedio de los vehículos por su status de reparación␣
 ↪('yes', 'no')
print(data[data['not_repaired']=='yes']['price'].mean())
print(data[data['not_repaired']=='no']['price'].mean())
```

```
2094.9552930000596
5366.576604893258
```

Podemos observar que los vehículos reparados valen 4 veces más que los vehículos no reparados, con esto y en ausencia de la especificidad del significado de la columna, interpretamos que los vehículos "no" reparados son vehículos descompuestos de alguna manera y por eso su valor se reduce.

Vamos a asumir que los valores NaN son vehículos funcionales, por lo tanto vamos a reemplazar los valores nulos con 'yes'

[24]:
```
data['not_repaired'] = data['not_repaired'].fillna('yes')
#verificando los cambios
data['not_repaired'].isna().any()
```

[24]:
```
False
```

Trabajando con la columna 'gearbox'

[25]:
```
print(data['gearbox'].isna().any())
data[data['gearbox'].isna()].head()
```

```
True
```

[25]:

	date_crawled	price	vehicle_type	registration_year	gearbox	power	model	mileage	registration_month	fuel_type	brand	not_repaired	date_created	number_of_pictures	postal_code	last_seen
15	11/03/2016 21:39	450.0	small	1910	NaN	105	ka	5000	0	petrol	ford	yes	11/03/2016 00:00	0	24148	19/03/2016 08:46
16	01/04/2016 12:46	300.0	unknown	2016	NaN	60	polo	150000	0	petrol	volkswagen	yes	01/04/2016 00:00	0	38871	01/04/2016 12:46
32	15/03/2016 20:59	245.0	sedan	1994	NaN	105	golf	150000	2	petrol	volkswagen	no	15/03/2016 00:00	0	44145	17/03/2016 18:17
37	28/03/2016 17:50	1500.0	unknown	2016	NaN	105	kangoo	150000	1	gasoline	renault	no	28/03/2016 00:00	0	46483	30/03/2016 09:18
40	26/03/2016 22:06	2700.0	unknown	1990	NaN	105	corsa	150000	1	petrol	opel	yes	26/03/2016 00:00	0	56412	27/03/2016 17:43

Realizamos el mismo proceso de la columna 'vehicle_type' , vamos a sustituir los valores NaN por la moda del modelo del vehículo

[26]:
```
data['gearbox'] = data['gearbox'].fillna(data.groupby('model')['gearbox'].
↪transform(lambda x : x.value_counts().idxmax()))
```

[27]:
```
print(data['gearbox'].unique())
data.iloc[14:16]
```

```
['manual' 'auto']
```

[27]:

	date_crawled	price	vehicle_type	registration_year	gearbox	power	model	mileage	registration_month	fuel_type	brand	not_repaired
15	11/03/2016 21:39	450.0	small	1910	manual	105	ka	5000	0	petrol	ford	yes
16	01/04/2016 12:46	300.0	unknown	2016	manual	60	polo	150000	0	petrol	volkswagen	yes

```
        date_created  number_of_pictures  postal_code         last_seen
15  11/03/2016 00:00                   0        24148  19/03/2016 08:46
16  01/04/2016 00:00                   0        38871  01/04/2016 12:46
```

Trabajando con la columna 'price' Podemos observar que price tiene registros con valor = 0 así como valores muy bajos, esto bajo la conclusión de que pueden ser usuarios llenando datos de sus vehículos de manera ficticia.

Debido a que no podemos tratar esta columna con ningúna otra métrica estadística para "estimar" los precios, decidimos eliminar los registros con precios menores a 10 euros

[28]:
```
data.head()
```

[28]:
```
        date_crawled   price vehicle_type  registration_year gearbox  power  \
0  24/03/2016 11:52   480.0      unknown               1993  manual    105
2  14/03/2016 12:52  9800.0          suv               2004    auto    163
3  17/03/2016 16:54  1500.0        small               2001  manual     75
4  31/03/2016 17:25  3600.0        small               2008  manual     69
5  04/04/2016 17:36   650.0        sedan               1995  manual    102

   model  mileage  registration_month fuel_type       brand not_repaired  \
0   golf   150000                   0    petrol  volkswagen          yes
2  grand   125000                   8  gasoline        jeep          yes
3   golf   150000                   6    petrol  volkswagen           no
4  fabia    90000                   7  gasoline       skoda           no
5    3er   150000                  10    petrol         bmw          yes

        date_created  number_of_pictures  postal_code         last_seen
0  24/03/2016 00:00                   0        70435  07/04/2016 03:16
2  14/03/2016 00:00                   0        90480  05/04/2016 12:47
3  17/03/2016 00:00                   0        91074  17/03/2016 17:40
4  31/03/2016 00:00                   0        60437  06/04/2016 10:17
5  04/04/2016 00:00                   0        33775  06/04/2016 19:17
```

[29]:
```
mask_price = data['price']>10
data=data[mask_price]
data
```

[29]:
```
            date_crawled   price vehicle_type  registration_year gearbox  \
0       24/03/2016 11:52   480.0      unknown               1993  manual
2       14/03/2016 12:52  9800.0          suv               2004    auto
3       17/03/2016 16:54  1500.0        small               2001  manual
4       31/03/2016 17:25  3600.0        small               2008  manual
5       04/04/2016 17:36   650.0        sedan               1995  manual
…                    …      …            …                  …      …
354363  27/03/2016 20:36  1150.0          bus               2000  manual
```

```
354364  21/03/2016 09:50  2700.0      unknown               2005  manual
354366  05/03/2016 19:56  1199.0  convertible               2000    auto
354367  19/03/2016 18:57  9200.0          bus               1996  manual
354368  20/03/2016 19:41  3400.0        wagon               2002  manual

        power        model  mileage  registration_month fuel_type       brand  \
0         105         golf   150000                   0    petrol  volkswagen
2         163        grand   125000                   8  gasoline        jeep
3          75         golf   150000                   6    petrol  volkswagen
4          69        fabia    90000                   7  gasoline       skoda
5         102          3er   150000                  10    petrol         bmw
…         …          …      …                       …        …          …
354363    105       zafira   150000                   3    petrol        opel
354364    105         colt   150000                   7    petrol  mitsubishi
354366    101       fortwo   125000                   3    petrol       smart
354367    102  transporter   150000                   3  gasoline  volkswagen
354368    100         golf   150000                   6  gasoline  volkswagen

       not_repaired      date_created  number_of_pictures  postal_code  \
0               yes  24/03/2016 00:00                   0        70435
2               yes  14/03/2016 00:00                   0        90480
3                no  17/03/2016 00:00                   0        91074
4                no  31/03/2016 00:00                   0        60437
5               yes  04/04/2016 00:00                   0        33775
…             …                …                     …          …
354363           no  27/03/2016 00:00                   0        26624
354364          yes  21/03/2016 00:00                   0         2694
354366           no  05/03/2016 00:00                   0        26135
354367           no  19/03/2016 00:00                   0        87439
354368          yes  20/03/2016 00:00                   0        40764

              last_seen
0       07/04/2016 03:16
2       05/04/2016 12:47
3       17/03/2016 17:40
4       06/04/2016 10:17
5       06/04/2016 19:17
…                  …
354363  29/03/2016 10:17
354364  21/03/2016 10:42
354366  11/03/2016 18:17
354367  07/04/2016 07:15
354368  24/03/2016 12:45

[333267 rows x 16 columns]
```

Trabajando con la columna 'registration_month'

[30]:
```
data.groupby('registration_month')['date_crawled'].count()
```

[30]:
```
registration_month
0     30324
1     21973
2     20352
3     33010
4     27894
5     27768
6     29992
7     25882
8     21629
9     22912
10    25050
11    23274
12    23207
Name: date_crawled, dtype: int64
```

En esta columna podemos observar que el mes 0 tiene la segunda mayor cantidad de registros, podríamos pensar que los usuarios seleccionaron el primer mes en el selector, esto nos trae problemas debido a que no podríamos utilizar la moda de los modelos, o el promedio del mes seleccionado en el resto de los registros, si bien, saber hace cuantos meses (y años) fue registrado el vehículo puede ser útil para que el comprador tome una decisión, para la predicción del precio del vehículo podríamos tomar por practicidad solamente el año de publicación del vehículo.

Con estas conclusiones reafirmamos la decisión inicial de eliminar esta columna

Eliminando columnas innecesarias para el modelo

[31]:
```
data.columns
```

[31]:
```
Index(['date_crawled', 'price', 'vehicle_type', 'registration_year', 'gearbox',
       'power', 'model', 'mileage', 'registration_month', 'fuel_type', 'brand',
       'not_repaired', 'date_created', 'number_of_pictures', 'postal_code',
       'last_seen'],
      dtype='object')
```

[32]:
```
data.drop(['date_crawled','registration_month',
           'date_created','number_of_pictures',
          'postal_code','last_seen'],axis=1,inplace=True)
```

0.1.3 Descripción de los datos después de los cambios y normalización

[33]:
```
data.head()
```

[33]:

	price	vehicle_type	registration_year	gearbox	power	model	mileage	\
0	480.0	unknown	1993	manual	105	golf	150000	
2	9800.0	suv	2004	auto	163	grand	125000	

```
3  1500.0        small               2001  manual     75   golf   150000
4  3600.0        small               2008  manual     69  fabia    90000
5   650.0        sedan               1995  manual    102    3er   150000

  fuel_type       brand not_repaired
0    petrol  volkswagen          yes
2  gasoline        jeep          yes
3    petrol  volkswagen           no
4  gasoline       skoda           no
5    petrol         bmw          yes
```

[34]:
```
data.describe()
```

[34]:
```
                price  registration_year          power        mileage
count  333267.000000      333267.000000  333267.000000  333267.000000
mean     4588.222962        2003.210375     119.327461  128626.251624
std      4482.252175           7.136372      58.636867   37090.767887
min        11.000000        1910.000000       1.000000    5000.000000
25%      1290.000000        1999.000000      82.000000  125000.000000
50%      2850.000000        2003.000000     105.000000  150000.000000
75%      6500.000000        2008.000000     143.000000  150000.000000
max     20000.000000        2019.000000    1937.000000  150000.000000
```

[35]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 333267 entries, 0 to 354368
Data columns (total 10 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   price              333267 non-null  float64
 1   vehicle_type       333267 non-null  object
 2   registration_year  333267 non-null  int64
 3   gearbox            333267 non-null  object
 4   power              333267 non-null  int64
 5   model              333267 non-null  object
 6   mileage            333267 non-null  int64
 7   fuel_type          333267 non-null  object
 8   brand              333267 non-null  object
 9   not_repaired       333267 non-null  object
dtypes: float64(1), int64(3), object(6)
memory usage: 28.0+ MB
```

Podemos observar que tenemos 10 columnas con 333,267 registros no nulos,que representan el 94.04% de los datos originales, este porcentaje nos representa una buena integridad de los datos iniciales

- Los precios oscilan entre los 11 y 200,000 euros

- El año de registro de los vehículos oscila entre 1910 y 2019
- La potencia (CV) de los vehículos oscila entre 1 y 1937 caballos de vapor
- El millaje de los vehículos oscila entre los 5,000 y 150,000 millas

0.1.4 Preparación de los datos para el modelo

En esta sección vamos a realizar el preparado de los datos específicamente para el modelo: * Codificación de características categóricas * Segmentación de datos de prueba y entrenamiento * Escalamiento de los datos * Definición de características y Objetivo

Codificando características categóricas mediante Binary Encoder

[36]:
```
data2= data.copy()
```

[37]:
```
data2
```

[37]:
```
         price vehicle_type  registration_year gearbox  power        model  \
0        480.0      unknown               1993  manual    105         golf
2       9800.0          suv               2004    auto    163        grand
3       1500.0        small               2001  manual     75         golf
4       3600.0        small               2008  manual     69        fabia
5        650.0        sedan               1995  manual    102          3er
…          …            …                  …       …      …           …
354363  1150.0          bus               2000  manual    105       zafira
354364  2700.0      unknown               2005  manual    105         colt
354366  1199.0  convertible               2000    auto    101       fortwo
354367  9200.0          bus               1996  manual    102  transporter
354368  3400.0        wagon               2002  manual    100         golf

        mileage fuel_type       brand not_repaired
0        150000    petrol  volkswagen          yes
2        125000  gasoline        jeep          yes
3        150000    petrol  volkswagen           no
4         90000  gasoline       skoda           no
5        150000    petrol         bmw          yes
…           …         …           …           …
354363   150000    petrol        opel           no
354364   150000    petrol  mitsubishi          yes
354366   125000    petrol       smart           no
354367   150000  gasoline  volkswagen           no
354368   150000  gasoline  volkswagen          yes

[333267 rows x 10 columns]
```

[38]:
```
#definiendo categorías únicas
unique_cat = data['model'].unique()
cat_num= len(unique_cat)
```

```
#Obteniendo el logaritmo base 2 para obtener la cantidad de bits necesarios␣
 ↪para cubrir las 250 categorías
num_bits = math.ceil(math.log2(cat_num))
categorical =␣
 ↪['vehicle_type','gearbox','model','fuel_type','brand','not_repaired']
```

[39]:
```
#Creando una instancia de Binary Encoder
binary_enc =ce.BinaryEncoder(cols=data[categorical])
#Codificando columnas categóricas
data_b_enc= binary_enc.fit_transform(data[categorical])
#Concatenando columnas nuevas con columnas existentes y eliminando las columnas␣
 ↪categóricas originales
#Se crea un nuevo dataframe 'data_encoded'
data_encoded = pd.concat([data.
 ↪drop(data[categorical],axis=1),data_b_enc],axis=1)
```

Segmentando datos de entrenamiento y validación

- Escalando características numéricas

[40]:
```
#Segmentando datos de validación y entrenamiento
df_train, df_valid = train_test_split(data_encoded, test_size = 0.
 ↪3,random_state=12345)
df_train_scaled= df_train.copy()
df_valid_scaled = df_valid.copy()

# Segmentando datos de validación y entrenamiento para  CatBoost
df_train, df_valid = train_test_split(data2, test_size = 0.3,random_state=12345)
df_train2= df_train.copy()
df_valid2 = df_valid.copy()

#Escalando características numéricas
numeric =['registration_year','power','mileage']
scaler = StandardScaler()
scaler.fit(df_train[numeric])

df_train_scaled.loc[:,numeric] = scaler.transform(df_train_scaled[numeric])
df_valid_scaled.loc[:,numeric] = scaler.transform(df_valid_scaled[numeric])


print(df_train_scaled.shape)
print(df_valid_scaled.shape)
```

```
(233286, 29)
(99981, 29)
```

[41]:
```
df_valid_scaled.head()
```

[41]:

	price	registration_year	power	mileage	vehicle_type_0	\
84837	1200.0	0.250576	-0.245609	0.575511	0	
348627	6450.0	1.089438	-0.245609	-1.313432	0	
248206	2900.0	-0.029045	-0.951659	-0.773734	0	
137921	2700.0	-0.029045	0.202130	0.575511	0	
224745	6900.0	-3.803926	-1.468281	-2.122979	0	

	vehicle_type_1	vehicle_type_2	vehicle_type_3	gearbox_0	gearbox_1	\
84837	0	0	1	1	0	
348627	0	1	1	0	1	
248206	0	1	1	0	1	
137921	1	0	0	0	1	
224745	0	1	1	0	1	

	…	fuel_type_1	fuel_type_2	brand_0	brand_1	brand_2	brand_3	\
84837	…	1	0	0	0	0	1	
348627	…	0	1	0	0	0	1	
248206	…	0	1	0	0	0	0	
137921	…	1	0	0	1	0	0	
224745	…	0	1	0	0	0	0	

	brand_4	brand_5	not_repaired_0	not_repaired_1
84837	0	0	0	1
348627	1	0	1	0
248206	0	1	1	0
137921	1	1	1	0
224745	0	1	1	0

[5 rows x 29 columns]

Definiendo características y objetivo

[42]:
```
#Definiendo características y objetivo
target = 'price'
features = [i for i in data_encoded.columns if i not in target]
print(len(features))

#Definiendo características para CatBoostRegressor
features2 = [i for i in data2.columns if i not in target]
print(len(features2))
```

```
28
9
```

0.2 Entrenamiento del modelo

0.2.1 Modelo de regresión lineal (modelo base)

[43]:
```
%%time
model_lr = LinearRegression()
model_lr.fit(df_train_scaled[features],df_train_scaled[target])
```

```
CPU times: user 362 ms, sys: 158 ms, total: 519 ms
Wall time: 497 ms
```

[43]:
```
LinearRegression()
```

[44]:
```
%%time
predictions_valid = model_lr.predict(df_valid_scaled[features])
```

```
CPU times: user 4.52 ms, sys: 19 ms, total: 23.5 ms
Wall time: 19.1 ms
```

[45]:
```
mean_lrmodel = predictions_valid.mean()
mse = mean_squared_error(df_valid[target],predictions_valid)
rmse = mse **.5
print(f'el precio promedio predicho de los vehículos es:␣
 ↪{round(mean_lrmodel,2)} euros')
print(f'el error medio cuadrático para la estimaciones es: {round(rmse,2)}␣
 ↪euros')
```

```
el precio promedio predicho de los vehículos es: 4598.69 euros
el error medio cuadrático para la estimaciones es: 3080.16 euros
```

Podemos observar que la regresión lineal nos arrojó una predicción promedio de 4,598 euros y un RMSE de 3,080 euros, el desempeño del modelo no es muy bueno, sin embargo, lo importante del modelo de regresión lineal (para este ejercicio) es utilizarlo para hacer una prueba de cordura de otros métodos.

0.2.2 Random Forest Regressor sin ajuste de hiperparámetros (modelo base)

[46]:
```
%%time
model_rf= RandomForestRegressor(n_estimators=30, random_state=12345)
model_rf.fit(df_train_scaled[features],df_train_scaled[target])
```

```
CPU times: user 37.1 s, sys: 241 ms, total: 37.3 s
Wall time: 37.4 s
```

[46]:
```
RandomForestRegressor(n_estimators=30, random_state=12345)
```

[47]:
```
%%time
predictions_rf = model_rf.predict(df_valid_scaled[features])
```

```
CPU times: user 1.31 s, sys: 7.86 ms, total: 1.32 s
Wall time: 1.33 s
```

[48]:
```
mean_rfmodel = predictions_rf.mean()
mse = mean_squared_error(df_valid[target],predictions_rf)
rmse = mse **.5
print(f'el precio promedio predicho de los vehículos es:␣
 ↪{round(mean_rfmodel,2)} euros')
print(f'el error medio cuadrático para la estimaciones es: {round(rmse,2)}␣
 ↪euros')
```

```
el precio promedio predicho de los vehículos es: 4611.18 euros
el error medio cuadrático para la estimaciones es: 1719.13 euros
```

[49]:
```
#Borrando variables para evitar kernel se muera.
del model_rf
del mse
del rmse
```

0.2.3 Random Forest Regressor con ajuste de hiperparámetros

[50]:
```
%%time
param_grid = {'n_estimators': [50,60],
              'max_features': ['log2', 'sqrt']}

grid_search = GridSearchCV(RandomForestRegressor(random_state=12345),
                           param_grid=param_grid,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1)
```

```
CPU times: user 83 µs, sys: 4 µs, total: 87 µs
Wall time: 89.2 µs
```

[51]:
```
%%time
grid_search.fit(df_train_scaled[features],df_train_scaled[target])
# Wall time: 5min 44s
```

```
CPU times: user 5min 22s, sys: 5.7 s, total: 5min 28s
Wall time: 5min 28s
```

[51]:
```
GridSearchCV(estimator=RandomForestRegressor(random_state=12345), n_jobs=-1,
             param_grid={'max_features': ['log2', 'sqrt'],
                         'n_estimators': [50, 60]},
             scoring='neg_root_mean_squared_error')
```

[52]:
```
%%time
best_model_rf = grid_search.best_estimator_
```

```
CPU times: user 5 µs, sys: 0 ns, total: 5 µs
Wall time: 9.06 µs
```

[53]:
```
print(grid_search.best_params_)
```

```
{'max_features': 'sqrt', 'n_estimators': 60}
```

[54]:
```
%%time
final_rf_predicitons= best_model_rf.predict(df_valid_scaled[features])
mse_cv_rf = mean_squared_error(df_valid[target],final_rf_predicitons)
final_rf_rmse = mse_cv_rf**.5
print(f'el precio promedio predicho de los vehículos es:␣
 ↪{round(final_rf_predicitons.mean(),2)} euros')
print(f'el error medio cuadrático para la estimaciones es:␣
 ↪{round(final_rf_rmse,2)} euros')
```

```
el precio promedio predicho de los vehículos es: 4580.58 euros
el error medio cuadrático para la estimaciones es: 1683.79 euros
CPU times: user 2.63 s, sys: 8.01 ms, total: 2.64 s
Wall time: 2.67 s
```

[55]:
```
del best_model_rf
del final_rf_predicitons
del mse_cv_rf
del final_rf_rmse
```

0.2.4 LightGBM sin ajuste de hiperparámetros (modelo base)

[56]:
```
%%time
lgbm_b_model= LGBMRegressor(random_state=12345)
lgbm_b_model.fit(df_train_scaled[features],df_train_scaled[target])
```

```
CPU times: user 4.81 s, sys: 67.6 ms, total: 4.88 s
Wall time: 4.87 s
```

[56]:
```
LGBMRegressor(random_state=12345)
```

[57]:
```
%%time
lgbm_b_predictions = lgbm_b_model.predict(df_valid_scaled[features])
lgbm_b_mean = lgbm_b_predictions.mean()
lgbm_b_mse= mean_squared_error(df_valid_scaled[target],lgbm_b_predictions)
lgbm_b_rmse= lgbm_b_mse **.5
```

```
CPU times: user 888 ms, sys: 7.92 ms, total: 896 ms
Wall time: 893 ms
```

[58]:
```
print(f'el precio promedio predicho de los vehículos es: {round(lgbm_b_mean,2)}␣
↪euros')
print(f'el error medio cuadrático para la estimaciones es:␣
↪{round(lgbm_b_rmse,2)} euros')
```

```
el precio promedio predicho de los vehículos es: 4589.93 euros
el error medio cuadrático para la estimaciones es: 1823.87 euros
```

[59]:
```
del lgbm_b_model
del lgbm_b_predictions
del lgbm_b_mean
del lgbm_b_mse
del lgbm_b_rmse
```

0.2.5 LightGBM con ajuste de hiperparámetros

[60]:
```
param_grid={'num_leaves':[50,100],
       'max_depth':[3,5],
       'learning_rate':[None,.01],
}
```

[61]:
```
gridsearch_lgbm = GridSearchCV(LGBMRegressor(random_state=12345),
                               ␣
↪param_grid=param_grid,n_jobs=-1,scoring='neg_root_mean_squared_error')
```

[62]:
```
%%time
gridsearch_lgbm.fit(df_train_scaled[features],df_train_scaled[target])
```

```
CPU times: user 4min 5s, sys: 1.98 s, total: 4min 7s
Wall time: 4min 8s
```

[62]:
```
GridSearchCV(estimator=LGBMRegressor(random_state=12345), n_jobs=-1,
             param_grid={'learning_rate': [None, 0.01], 'max_depth': [None, 10],
                         'num_leaves': [50, 100]},
             scoring='neg_root_mean_squared_error')
```

[63]:
```
gridsearch_lgbm_best= gridsearch_lgbm.best_estimator_
```

[64]:
```
print(gridsearch_lgbm.best_params_)
# {'learning_rate': None, 'max_depth': None, 'num_leaves': 100}
```

```
{'learning_rate': None, 'max_depth': None, 'num_leaves': 100}
```

Esto indica que los mejores parámetros fueron:

- **learning_rate= 0.1**
- **max_depth = -1**
- **num_leaves= 100**

[65]:
```
%%time
final_lgbm_predicitons= gridsearch_lgbm_best.predict(df_valid_scaled[features])
mse_cv_lgbm = mean_squared_error(df_valid[target],final_lgbm_predicitons)
final_lgbm_rmse = mse_cv_lgbm**.5
print(f'el precio promedio predicho de los vehículos es:␣
 ↪{round(final_lgbm_predicitons.mean(),2)} euros')
print(f'el error medio cuadrático para la estimaciones es:␣
 ↪{round(final_lgbm_rmse,2)} euros')
```

```
el precio promedio predicho de los vehículos es: 4592.53 euros
el error medio cuadrático para la estimaciones es: 1718.73 euros
CPU times: user 1.26 s, sys: 8.06 ms, total: 1.26 s
Wall time: 1.29 s
```

[66]:
```
del gridsearch_lgbm
del gridsearch_lgbm_best
del final_lgbm_predicitons
del mse_cv_lgbm
del final_lgbm_rmse
```

0.2.6 Catboost sin ajuste de hiperparámetros (modelo base)

[67]:
```
#Definiendo características categóricas
cat_features=['vehicle_type','gearbox','model','fuel_type','brand','not_repaired']
```

[68]:
```
%%time
cb_model= CatBoostRegressor(random_state=12345, loss_function='RMSE',␣
 ↪iterations=150)
cb_model.
 ↪fit(df_train2[features2],df_train2[target],verbose=100,cat_features=cat_features)
```

```
Learning rate set to 0.453048
0:      learn: 3372.4091298     total: 217ms    remaining: 32.4s
100:    learn: 1717.9328863     total: 15.6s    remaining: 7.55s
149:    learn: 1681.9248020     total: 23.3s    remaining: 0us
CPU times: user 23.4 s, sys: 136 ms, total: 23.5 s
Wall time: 24 s
```

[68]:
```
<catboost.core.CatBoostRegressor at 0x7fd951b0d0d0>
```

[69]:
```
%%time
cb_predictions = cb_model.predict(df_valid2[features2])
cb_mean = cb_predictions.mean()
cb_mse= mean_squared_error(df_valid2[target],cb_predictions)
cb_rmse= cb_mse **.5
```

```
print(f'el precio promedio predicho de los vehículos es: {round(cb_mean,2)}␣
 ↪euros')
print(f'el error medio cuadrático para la estimaciones es: {round(cb_rmse,2)}␣
 ↪euros')
```

```
el precio promedio predicho de los vehículos es: 4594.54 euros
el error medio cuadrático para la estimaciones es: 1743.44 euros
CPU times: user 175 ms, sys: 3.99 ms, total: 179 ms
Wall time: 178 ms
```

[70]:
```
del cb_model
del cb_predictions
del cb_mean
del cb_mse
del cb_rmse
```

0.2.7 Catboost con ajuste de hiperparámetros

[71]:
```
param_grid={
           'learning_rate':[0.5],
    'depth':[5,10],
    'early_stopping_rounds': [7]
           }
```

En el paso anterior probamos las siguientes combinaciones:

Configuración 1: * 'iterations' * [100,1000] * 'learning_rate':[0.1,.01] * RECM: aproximado de 1657 euros * Wall time: 51min 2s

Configuración 2: * 'iterations' * [100,500] * 'learning_rate':[0.1,.01] * RECM: 1721.26 euros * Wall time: 27min 2s

Configuración 3: * 'iterations' * [100,1000] * 'learning_rate':[0.1] * RECM: aproximado de * Wall time:

decidimos conservar la configuración 3 por su desempeño tanto en el estimador como en el tiempo de procesamiento.

[72]:
```
cb_gridsearch=␣
 ↪GridSearchCV(CatBoostRegressor(random_state=12345,loss_function='RMSE',iterations=150),para
 ↪n_jobs=-1)
```

[73]:
```
%%time
cb_gridsearch.
 ↪fit(df_train2[features2],df_train2[target],verbose=100,cat_features=cat_features)
# Wall time: 51min 2s
```

```
0:      learn: 3314.2450892     total: 119ms    remaining: 17.7s
100:    learn: 1755.8005997     total: 10.7s    remaining: 5.2s
```

```
149:    learn: 1719.7409511     total: 15.9s    remaining: 0us
0:      learn: 3329.7553209     total: 119ms    remaining: 17.7s
100:    learn: 1752.7349594     total: 10.6s    remaining: 5.15s
149:    learn: 1718.7820876     total: 15.7s    remaining: 0us
0:      learn: 3322.8128032     total: 116ms    remaining: 17.3s
100:    learn: 1757.1143035     total: 10.7s    remaining: 5.18s
149:    learn: 1721.4998362     total: 15.8s    remaining: 0us
0:      learn: 3328.7134088     total: 117ms    remaining: 17.4s
100:    learn: 1762.9185007     total: 10.7s    remaining: 5.18s
149:    learn: 1726.6128477     total: 15.8s    remaining: 0us
0:      learn: 3288.2150254     total: 122ms    remaining: 18.1s
100:    learn: 1762.3299598     total: 10.7s    remaining: 5.17s
149:    learn: 1728.2869439     total: 15.8s    remaining: 0us
0:      learn: 3081.7762402     total: 228ms    remaining: 33.9s
100:    learn: 1510.3925690     total: 22.5s    remaining: 10.9s
149:    learn: 1456.0737623     total: 33.5s    remaining: 0us
0:      learn: 3070.1642003     total: 235ms    remaining: 34.9s
100:    learn: 1500.5652713     total: 22.5s    remaining: 10.9s
149:    learn: 1446.1404155     total: 33.5s    remaining: 0us
0:      learn: 3097.7268146     total: 233ms    remaining: 34.7s
100:    learn: 1509.0714260     total: 22.6s    remaining: 11s
149:    learn: 1451.2808945     total: 33.4s    remaining: 0us
0:      learn: 3093.1557822     total: 233ms    remaining: 34.7s
100:    learn: 1513.6070417     total: 22.9s    remaining: 11.1s
149:    learn: 1457.4395034     total: 33.8s    remaining: 0us
0:      learn: 3075.1502695     total: 249ms    remaining: 37.1s
100:    learn: 1518.4626910     total: 22.6s    remaining: 11s
149:    learn: 1462.6270468     total: 33.6s    remaining: 0us
0:      learn: 3090.5125278     total: 291ms    remaining: 43.4s
100:    learn: 1518.6174617     total: 28.1s    remaining: 13.6s
149:    learn: 1467.6785242     total: 41.6s    remaining: 0us
CPU times: user 4min 53s, sys: 806 ms, total: 4min 53s
Wall time: 4min 55s
```

[73]:
```
GridSearchCV(estimator=<catboost.core.CatBoostRegressor object at
0x7fd951b11af0>,
             n_jobs=-1,
             param_grid={'depth': [5, 10], 'early_stopping_rounds': [7],
                         'learning_rate': [0.5]})
```

[74]:
```
cb_gridsearch_best = cb_gridsearch.best_estimator_
```

[75]:
```
cb_gridsearch.best_params_
```

[75]:
```
{'depth': 10, 'early_stopping_rounds': 7, 'learning_rate': 0.5}
```

[76]:
```
%%time
cb_f_predictions = cb_gridsearch_best.predict(df_valid2[features2])
cb_f_mean = cb_f_predictions.mean()
cb_f_mse= mean_squared_error(df_valid2[target],cb_f_predictions)
cb_f_rmse= cb_f_mse **.5

print(f'el precio promedio predicho de los vehículos es: {round(cb_f_mean,2)}␣
 ↪euros')
print(f'el error medio cuadrático para la estimaciones es: {round(cb_f_rmse,2)}␣
 ↪euros')
```

```
el precio promedio predicho de los vehículos es: 4585.31 euros
el error medio cuadrático para la estimaciones es: 1683.11 euros
CPU times: user 225 ms, sys: 10 µs, total: 225 ms
Wall time: 254 ms
```

0.3 Conclusiones

0.3.1 Conclusiones de los modelos

Resumamos los resultados obtenidos:

Regresión lineal sin ajuste de hiperparámetros (modelo benchmark): * Tiempo de entrenamiento : 579 ms * Tiempo de predicción : 1.42 s * Precio promedio en la predicción = 4598.69 euros * Error cuadrático medio = 3080.16

Un error muy grande si lo comparamos con la media

Random Forest Regressor sin ajuste de hiperparámetros * Tiempo de entrenamiento : 37.1 s * Tiempo de predicción : 1.42 s * Precio promedio en la predicción = 4611.18 euros * Error cuadrático medio = 1719.13

Podemos observar la primera iteración de mejora del RMSE

Random Forest Regressor con ajuste de hiperparámetros mediante GridSearchCV * Tiempo de entrenamiento : 5 min 45 segundos * Tiempo de predicción : 1.42 s * Precio promedio en la predicción = 4580.58 euros * Error cuadrático medio = 1683.79 * Mejores Hiperparámetros: * max_features = sqrt * n_estimators = 60

Podemos observar 2da iteración de mejora del RMSE

LightGBM sin ajuste de hiperparámetros * Tiempo de entrenamiento : 4.97 s * Tiempo de predicción : 823 ms * Precio promedio en la predicción = 4589.93 euros * Error cuadrático medio = 1823.87

Observamos un RMSE mejor que el modelo benchmark, pero no mejor que Random Forest, sin embargo el tiempo de entrenamiento y predicción fue notablemente mejor que el modelo de bosque aleatorio, veamos como se comporta con mejores hiperparámetros.

LightGBM con ajuste de hiperparámetros * Tiempo de entrenamiento : 4 min 5 segundos * Tiempo de predicción : 1.29 s * Precio promedio en la predicción = 4592.53 euros * Error cuadrático medio = 1718.73 * Mejores Hiperparámetros: * learning_rate = None (por default : 0.1) * max_depth = None (por default : -1) * num_leaves = 100

Podemos observar otra ronda de mejora del RMSE en relación al modelo benchmark y al modelo LightGBM sin ajuste de hiperparámetros, aumenta el tiempo de entrenamiento y de predicción, pero mejora sustancialmente el evaluador RMSE.

Catboost sin ajuste de hiperparámetros * Tiempo de entrenamiento : 24 s * Tiempo de predicción : 178 ms * Precio promedio en la predicción = 4594.54 euros * Error cuadrático medio = 1743.44

Podemos observar un tiempo de entrenamiento y predicción muy aceptable, el RMSE mejor que el modelo benchmark pero no mejor que LightGBM ni RandomForest, pensando en el tiempo de entrenamiento, podríamos pensar que es un modelo "equilibrado".

Catboost con ajuste de hiperparámetros * Tiempo de entrenamiento : 27 min 59 s * Tiempo de predicción : 1.07 ms * Precio promedio en la predicción = 4585.31 euros * Error cuadrático medio = 1683.11 * Mejores Hiperparámetros: * learning_rate = 0.5 * depth = 10 * early_stopping_rounds = 7 * iterations = 150

Podemos observar el mejor RMSE en relación a todos los modelos, sin embargo el tiempo de entrenamiento aumenta sustancialmente, podríamos concluir que es un modelo muy exacto pero con un costo computacional alto.

0.3.2 Conclusión general

Pudimos observar que el mejor resultado (desde el punto de vista de la predicción y RMSE) fue obtenido por **Catboost** con: * Precio promedio en la predicción = 4585.31 euros * Error cuadrático medio = 1683.11

Si embargo su costo computacional fue alto, el tiempo de predicción de 30 minutos es muy notorio, podemos mejorar la exactitud iterando más hiperparámetros mediante GridSearchCV (por ejemplo iterations = 1000) sin embargo el RMSE no baja "demasiado" para que compense un costo computacional alto (mayor a 51 min según las pruebas realizadas).

Codificación de datos:

En este punto decidimos utilizar un codificador **Binary-Encoder** debido a que la propuesta de utilizar **OneHot Encoder** nos generaba +300 columnas categóricas, esto nos traía una alta dimensionalidad para los modelos, lo cual no es deseable, aunado a que constantemente mataba el kernel.

En conclusión:

Los modelos que utilizan potenciación de gradiente nos ofrecen una mayor exactitud en las métricas de evaluación sin embargo aumentan su complejidad y *pueden* aumentar su costo computacional, dentro de los modelos evaluados pudimos observar un modelo *equilibrado*, en este caso fue el LightGBM con ajuste de hiperparámetros, nos ofreció las bondades de un RMSE aceptable (en relación al resto de los modelos) y un tiempo de entrenamiento aceptable.

Me parece importante destacar que la potenciación del gradiente nos ofrece bondades sustanciales al momento de buscar un mejor desempeño de los modelos.

0.3.3 Notas del equipo de ciencia de datos

Decidimos brindar un orden al notebook en el cual observamos una secuencia de “entrenamiento -> ejecución-> eliminación de variables” para cada modelo debido a que el kernel constantemente se moría cuando intentabamos conservar todas los modelos “vivos” y entrenados para despúes realizar las predicciones.