

# Car Price Prediction

Qué tal amigos! 🙌😊

Bien dicen que la mejor forma de aprender algo nuevo es enseñar lo que sabes, y creo la mejor forma de enseñar es intentar demostrar los conceptos lo más simple posible, así que en este modelo es lo que intentaré hacer. Me parece un poco injusto para nosotros los latinos que casi no haya contenido de este estilo en español, así que quise intentarlo yo.

**Por favor, si puedo recibir feedback sobre esto, o algún otro comentario, lo agradecería.**

---

Bien, para mantener simple la explicación, el proceso de trabajo lo voy a dividir en 3 etapas:

1. Análisis Exploratorio de datos y limpieza
2. Ingenieria de Características (Feature Engineering)
3. Creación, Validación y Ajustes de Hiperparámetros del modelo

---

Antes de comenzar a Explorar los datos, necesitamos importar las librerías y ajustar algunas opciones:

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

# Estas lineas indican la longitud maxima por default en filas y columnas
pd.set_option('display.max_columns', 45)
pd.set_option('display.max_rows', 45)
# Así como el número de decimales
pd.options.display.float_format="{:,.4f}".format

import matplotlib.pyplot as plt
import seaborn as sns
# Esto solo asigna el estilo de gráficos por default
plt.style.use('fivethirtyeight')
```

```
In [4]: # el 'index_col=[0]' asigna la primera columna como indices, puesto que es el
df = pd.read_csv('../data/raw/CarPrice_Assignment.csv', index_col=[0])
```

## Análisis Exploratorio de Datos

Creo que para hacer un buen análisis exploratorio, se hay que comenzar con una meta para saber la dirección por la que se quiere llevar el enfoque. Por ahora el objetivo es crear un modelo en base a la variable objetivo -> **"price"**, para saber el precio que deberían tener

ciertos nuevos coches en el mercado.

Ya después lo haré conforme vayan surgiendo las preguntas.

In [5]: `df.head()`

Out[5]:

	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel
car_ID							

car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel
1	3	alfa-romero giulia	gas	std	two	convertible	rwd
2	3	alfa-romero stelvio	gas	std	two	convertible	rwd
3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd
4	2	audi 100 ls	gas	std	four	sedan	fwd
5	2	audi 100ls	gas	std	four	sedan	4wd

In [6]: `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
Index: 205 entries, 1 to 205
Data columns (total 25 columns):
#   Column                Non-Null Count  Dtype
---  -
0   symboling              205 non-null    int64
1   CarName                205 non-null    object
2   fueltype               205 non-null    object
3   aspiration              205 non-null    object
4   doornumber             205 non-null    object
5   carbody                205 non-null    object
6   drivewheel             205 non-null    object
7   enginelocation         205 non-null    object
8   wheelbase              205 non-null    float64
9   carlength              205 non-null    float64
10  carwidth                205 non-null    float64
11  carheight              205 non-null    float64
12  curbweight             205 non-null    int64
13  enginetype             205 non-null    object
14  cylindernumber         205 non-null    object
15  enginesize             205 non-null    int64
16  fuelsystem             205 non-null    object
17  boreratio              205 non-null    float64
18  stroke                 205 non-null    float64
19  compressionratio       205 non-null    float64
20  horsepower             205 non-null    int64
21  peakrpm                205 non-null    int64
22  citympg                205 non-null    int64
23  highwaympg            205 non-null    int64
24  price                  205 non-null    float64
dtypes: float64(8), int64(7), object(10)
memory usage: 41.6+ KB
```

Estos son las clases más comunes y más útiles creo yo para echar un primer vistazo a los datos. Con dos simples líneas de código me doy cuenta que es un dataset con:

- 205 datos
- 24 columnas
- 0 valores nulos
- dtypes de cada columna
- El contenido del df

Yo sé que es un df muy sencillo, y quizá en la vida real nunca te vas a encontrar información tan limpia, pero por motivos didácticos, quiero mantener esto lo más simple posible. La limpieza ya la haré un poco después de escoger las features de mi interés.

```
In [7]: df['CarName'].value_counts()
```

```
Out[7]: CarName
toyota corona      6
toyota corolla     6
peugeot 504        6
subaru dl          4
mitsubishi mirage g4  3
..
mazda glc 4        1
mazda rx2 coupe    1
maxda glc deluxe   1
maxda rx3          1
volvo 246          1
Name: count, Length: 147, dtype: int64
```

```
In [8]: # Aquí solo agrupamos aquellos valores repetidos y mostramos lo que contiene
df.groupby('CarName').get_group('toyota corona')
```

```
Out[8]:      symboling  CarName  fueltype  aspiration  doornumber  carbody  drivewheel  e
car_ID
```

152	1	toyota corona	gas	std	two	hatchback	fwd
159	0	toyota corona	diesel	std	four	sedan	fwd
161	0	toyota corona	gas	std	four	sedan	fwd
165	1	toyota corona	gas	std	two	hatchback	rwd
176	-1	toyota corona	gas	std	four	hatchback	fwd
180	3	toyota corona	gas	std	two	hatchback	rwd

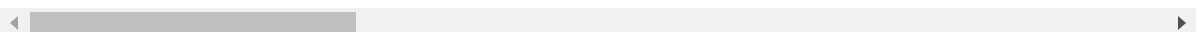
Podemos ver que el dataset contiene 205 modelos de distintas marcas de coches. La verdad es que no sé demasiado de coches. Me pregunto a que se deberá la variación en los datos. Quizá se deba a que son coches de distintos años, o quizá simplemente son distintos modelos, para estar seguro, haré lo siguiente:

In [9]: `df.drop_duplicates()`

Out[9]:

	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel
car_ID							
1	3	alfa-romero giulia	gas	std	two	convertible	rwd
2	3	alfa-romero stelvio	gas	std	two	convertible	rwd
3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd
4	2	audi 100 ls	gas	std	four	sedan	fwd
5	2	audi 100ls	gas	std	four	sedan	4wd
...	...	...	...	...	...	...	...
201	-1	volvo 145e (sw)	gas	std	four	sedan	rwd
202	-1	volvo 144ea	gas	turbo	four	sedan	rwd
203	-1	volvo 244dl	gas	std	four	sedan	rwd
204	-1	volvo 246	diesel	turbo	four	sedan	rwd
205	-1	volvo 264gl	gas	turbo	four	sedan	rwd

205 rows × 25 columns

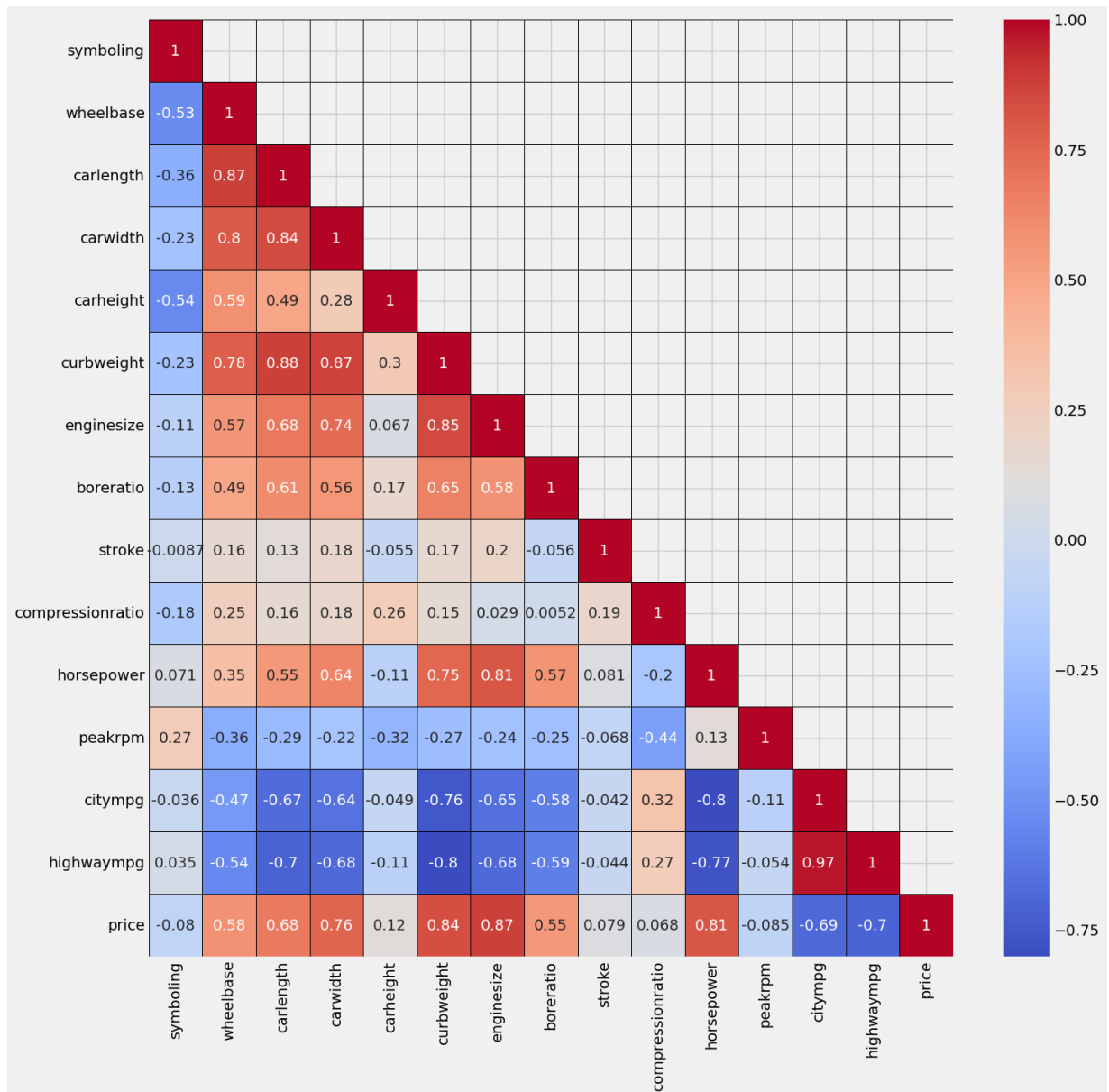


Bien, puesto que son distintos todos, voy a asumir que cada uno contribuye a la creación del modelo

La verdad no sé que es symboling. Hice una rápida búsqueda en Google, pero no logré encontrar una definición certera. Pudiera pasar tiempo buscando lo que es cada columna, pero prefiero hacer esto

In [17]: `# # Creamos una variable que seleccione solo los números, y haga una tabla de correlación  
# corr=df.select_dtypes(include='number').corr()  
# # y le escogemos un estilo con 2 decimales, paleta de colores "coolwarm"  
# corr.style.background_gradient(cmap='coolwarm').format("{:.2f}")  
plt.figure(figsize=(15, 15))  
corr = df.select_dtypes(include='number').corr()  
sns.heatmap(corr, cmap='coolwarm', annot=True, linewidths=0.5, linecolor='black')`

Out[17]: <Axes: >



Así a simple vista, comparado con la variable objetivo **price**, hay algunas que no tienen una correlación relevante. Quizá esto no sea lo más correcto, pero no creo influya demasiado en el modelo final. De esta forma me deshago de algunas columnas poco útiles.

```
In [9]: # Cada que hago modificaciones al df, prefiero crear uno nuevo para evitar problemas
df2 = df.drop(['symboling', 'stroke', 'compressionratio', 'peakrpm'], axis='col')
```

Si se preguntan porque no tiré 'carheight', es porque esta me servirá un poco después.

```
In [10]: df2.head()
```

Out[10]:

	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	engine loc
car_ID							
1	alfa-romero giulia	gas	std	two	convertible	rwd	
2	alfa-romero stelvio	gas	std	two	convertible	rwd	
3	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	
4	audi 100 ls	gas	std	four	sedan	fwd	
5	audi 100ls	gas	std	four	sedan	4wd	

Bien, ahora quiero lidiar con estas primeras columnas categóricas. Quiero saber como influyen en mi variable objetivo.

In [11]: `df2.select_dtypes(include='object').shape`

Out[11]: (205, 10)

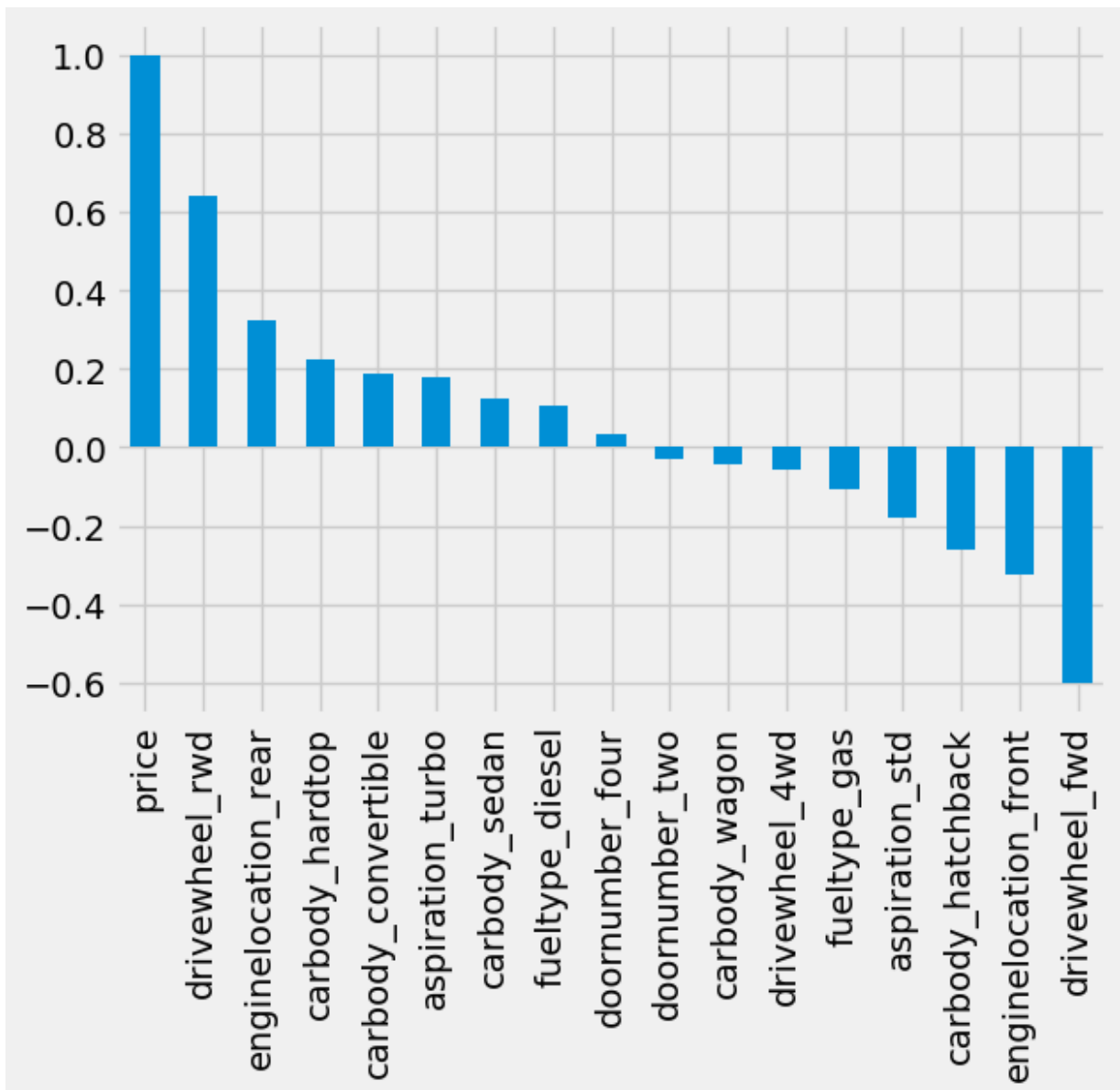
In [12]: '''Puede parecer un poco abrumador el código de las siguientes dos graficas, pero dejenme lo explico:  
la primera línea concatena la variable objetivo, y puesto que el modelo no s variables categóricas, hacemos One Hot Encoding, ya que cada columna contiene únicas.  
Asignamos este pequeño df a las variables "a" y "b", y creamos una gráfica c analizar la correlación con respecto a la variable objetivo "price". Ordenam de mayor a menor, y le damos el tamaño a la gráfica.  
  
Los separé entre a y b simplemente para una mejor visualización, pero en sí, pudieron haber estado juntas  
'''

```

a = pd.concat([pd.get_dummies(
    df2.select_dtypes(include='object')
    .iloc[:,1:7]), df2['price']], axis=1)
a.corr()['price'].sort_values(ascending=False).plot(kind='bar')
plt.figure(figsize=(15,5))

```

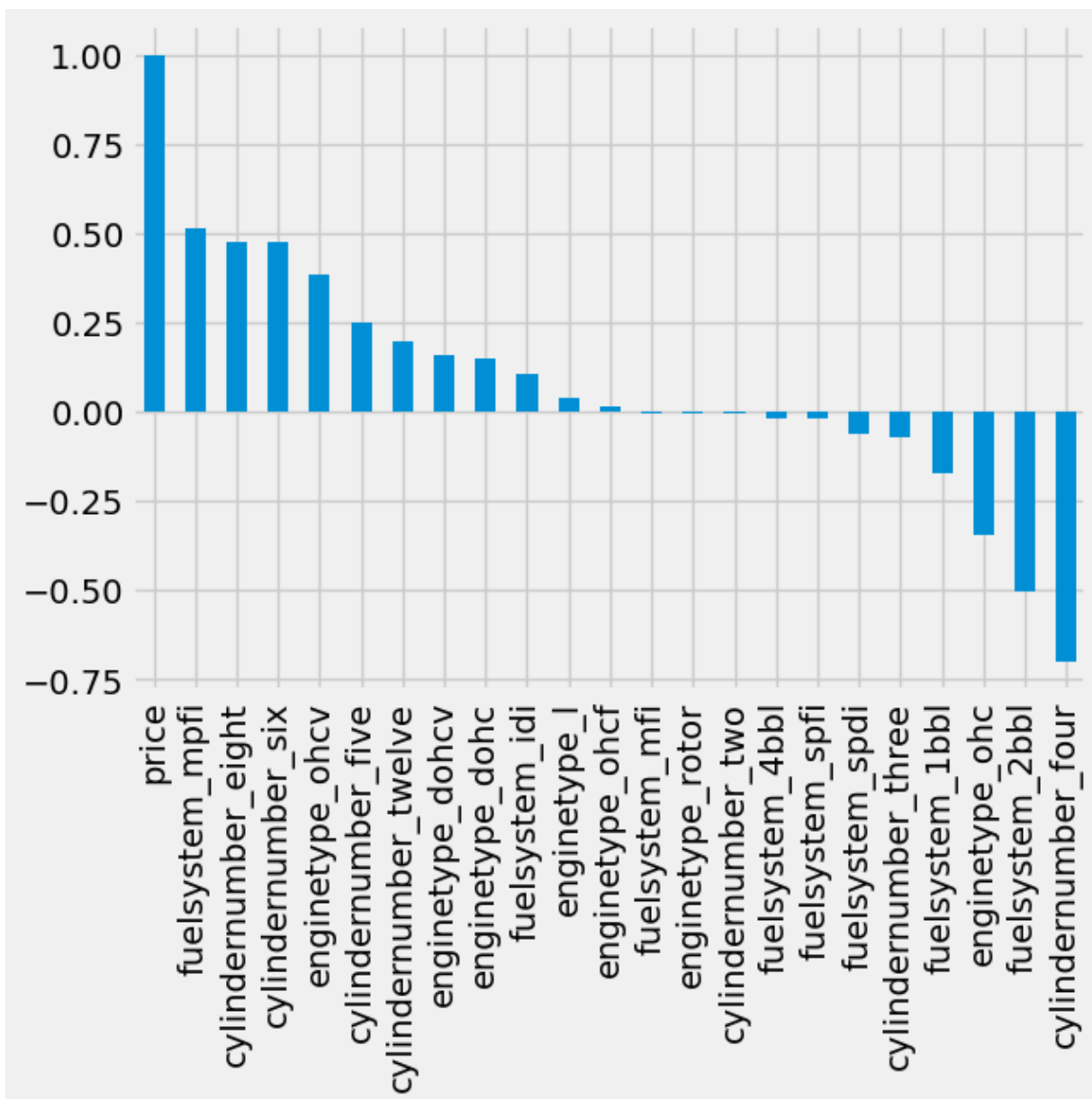
Out[12]: <Figure size 1500x500 with 0 Axes>



<Figure size 1500x500 with 0 Axes>

```
In [13]: b = pd.concat([pd.get_dummies(
    df2.select_dtypes(include='object')
    .iloc[:,7:]), df2['price']], axis=1)
b.corr()['price'].sort_values(ascending=False).plot(kind='bar')
plt.figure(figsize=(15,5))
```

Out[13]: <Figure size 1500x500 with 0 Axes>



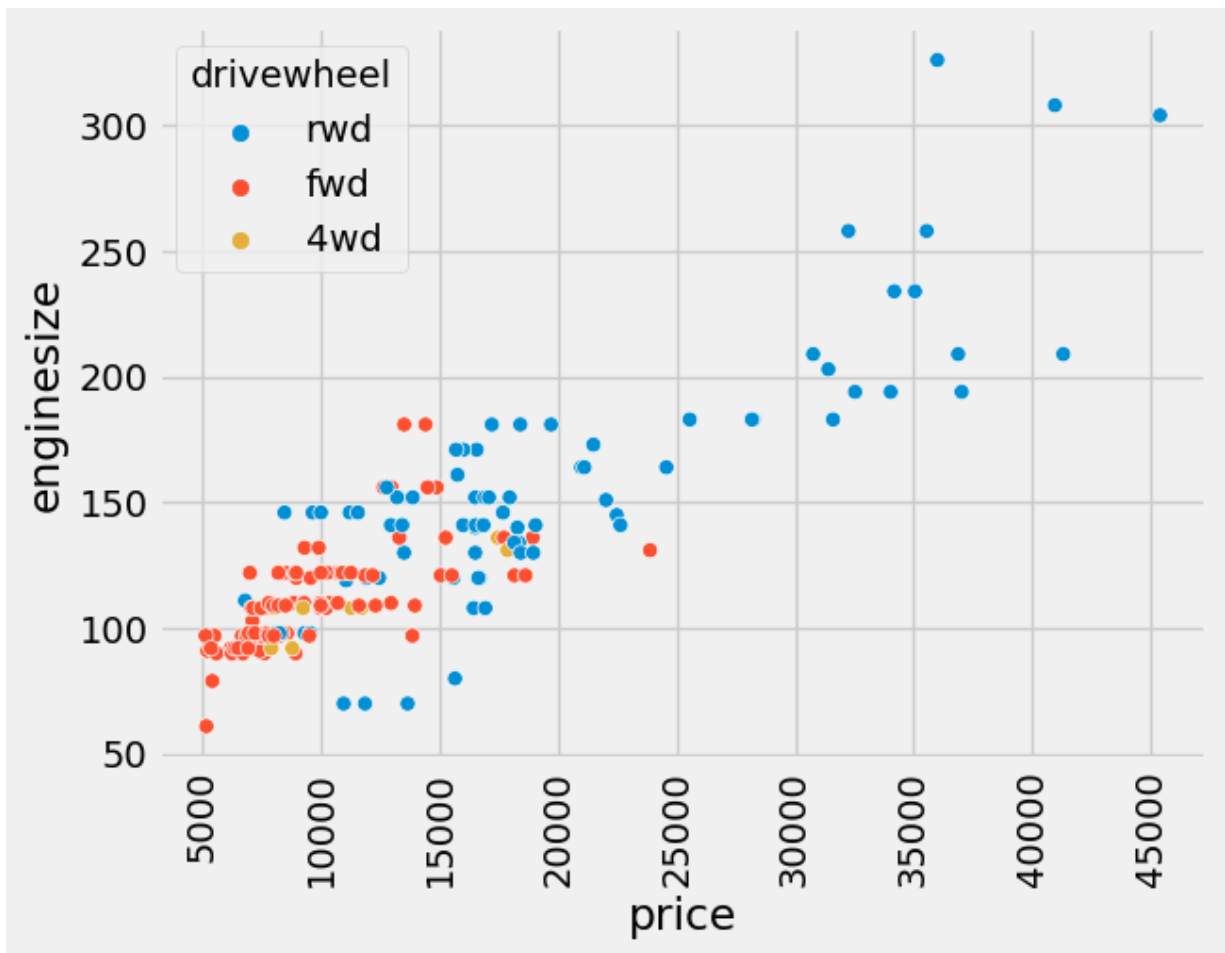
<Figure size 1500x500 with 0 Axes>

```
In [14]: # Tiraré de una vez el número de puertas puesto que no tiene una correlación
df2 = df2.drop('doornumber', axis='columns')
```

De las gráfica anteriores, podemos ver que existen algunas columnas que influyen más.  
Estas me gustaría visualizarlas dentro de una gráfica de dispersión:

```
In [15]: '''Bien, aquí utilicé enginesize en el eje y, puesto que era la que mejor cc
en el heatmap. xticks es solo para darle una rotación de 90 grados a los lab
sns.scatterplot(data=df2, x='price', y='enginesize', hue='drivewheel')
plt.xticks(rotation=90)
plt.show()'''
```





```
In [16]: df2['drivewheel'].value_counts()
```

```
Out[16]: drivewheel
fwd      120
rwd       76
4wd        9
Name: count, dtype: int64
```

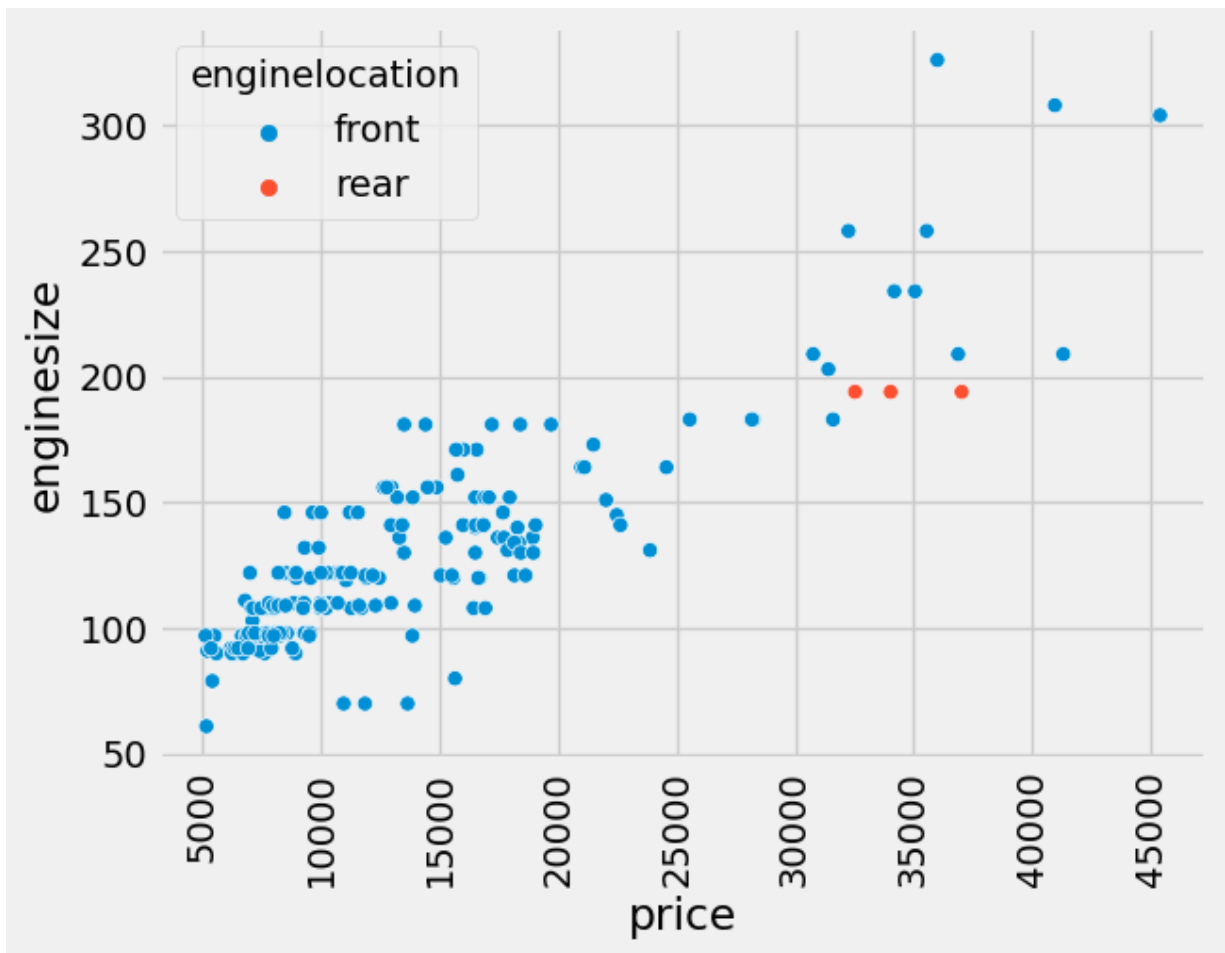
Nos podemos dar cuenta que, los coches con tracción en la llanta trasera ('rwd') son más caros, y a su vez tienen un enginesize más grande!

Dada la baja correlación en coches con tracción en 4 llantas('4wd'), su poca cantidad de datos y su semejanza en cuanto al precio, lo sumaré a los coches de tracción delantera, para ahorrarme el hacer OHE aquí, y así reducir la dimensionalidad en las columnas.

```
In [17]: df3 = df2.copy()
df3['drivewheel_rwd'] = df2['drivewheel'].apply(lambda x: 1 if x=='rwd' else 0)
```

```
In [18]: df3 = df3.drop('drivewheel', axis='columns')
```

```
In [19]: sns.scatterplot(data=df3, x='price', y='enginesize', hue='enginelocation')
plt.xticks(rotation=90)
plt.show()
```

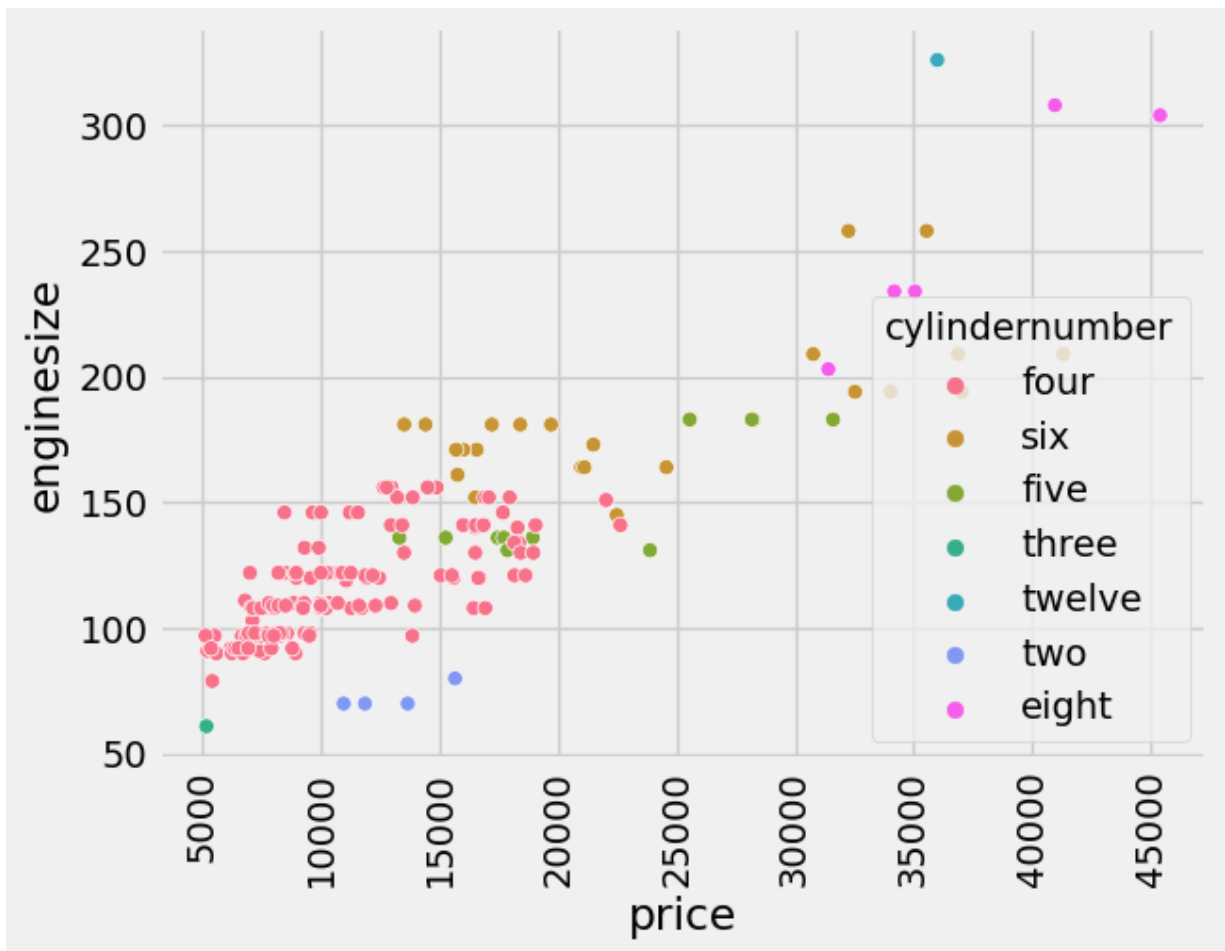


Bien, tener solo 3 valores no influirá en el modelo, así que esta columna se va.

```
In [20]: df3 = df3.drop('engine location', axis='columns')
```

Okey, volviendo a nuestra gráfica de correlación de barras, y a nuestro heatmap, me dan curiosidad ver dos cosas: Mientras menos millas por galón en carretera y la ciudad ("highwaympg" y "citympg"), mayor es el precio. También nos damos cuenta que el número de cilindros en un coche influye demasiado!

```
In [21]: sns.scatterplot(data=df3, x='price', y='enginesize', hue='cylindernumber')
plt.xticks(rotation=90)
plt.show()
```



Bien! Nos damos cuenta de algo que aunque suena obvio, al menos yo no era consciente antes de verlo en los datos: A mayor número de cilindros en un coche, mayor su precio, y a su vez mayor el enginesize. Entonces, en vez de dar un OHE a esta variable, podemos crear un Label Encoder, que transforme la variable categórica a numérica.

Bien, investigando un poco, me doy cuenta que sklearn no recomienda usar LabelEncoder para columnas, en vez, usar OrdinalEncoder. Otra observación es, que la mejor práctica es el utilizar One Hot Encoding desde sklearn y no con el metodo de pandas "get\_dummies", esto porque todo el preprocesamiento de datos es mejor hacerlo desde un pipeline para un mejor orden en la creación del modelo. Si soy sincero, **aún no tengo experiencia con el uso de Pipelines**, así que por este proyecto no lo usaré, pero seguiré aprendiendo...

```
In [22]: from sklearn.preprocessing import OrdinalEncoder

categorias_ordinales = ['two', 'three', 'four', 'five', 'six', 'eight', 'twelve']

# Crea una instancia de OrdinalEncoder y especifica las categorías
oe = OrdinalEncoder(categories=[categorias_ordinales])

# Ajusta y transforma tus datos
categorias_codificadas = oe.fit_transform(df3[['cylindernumber']])
```

```
In [23]: categorias_codificadas[:5]
```

```
Out[23]: array([[2.],
                [2.],
                [4.],
                [2.],
                [3.]])
```

```
In [24]: df3['oe_cylindernumber'] = categorias_codificadas
```

```
In [25]: df3.head()
```

```
Out[25]:
```

	CarName	fueltype	aspiration	carbody	wheelbase	carlength	carwidth	car_ID
--	---------	----------	------------	---------	-----------	-----------	----------	--------

1	alfa-romero giulia	gas	std	convertible	88.6000	168.8000	64.1000
2	alfa-romero stelvio	gas	std	convertible	88.6000	168.8000	64.1000
3	alfa-romero Quadrifoglio	gas	std	hatchback	94.5000	171.2000	65.5000
4	audi 100 ls	gas	std	sedan	99.8000	176.6000	66.2000
5	audi 100ls	gas	std	sedan	99.4000	176.6000	66.4000

Bien, ahora solo hay que recordar que los números no son equivalentes a el numero de cilindros, puesto que oe ordena los valores de 0 - (no\_valores\_unicos-1)

Este proyecto lo estoy haciendo a través de una jupyter notebook desde el buscador web, porque creí que así se vería más profesional, pero si es demasiada diferencia trabajar en tu propio entorno local como lo es en VSCode. No hay IDE que se asemeje, y el Workflow es mucho más rápido. De todas formas, terminaré de una vez con este proyecto ya aquí.

```
In [26]: df4 = df3.copy()
df4 = df4.drop('cylindernumber', axis='columns')
```

```
In [27]: df4['fueltype'].value_counts()
```

```
Out[27]: fueltype
gas      185
diesel   20
Name: count, dtype: int64
```

```
In [28]: df4['fuelsystem'].value_counts()
```

```
Out[28]: fuelsystem
mpfi      94
2bbl      66
idi       20
1bbl      11
spdi       9
4bbl       3
mfi        1
spfi       1
Name: count, dtype: int64
```


```
In [29]: '''Tenemos algunas variables con un solo valor! Si hago One Hot Encoding con
inútiles, así que, aquellos fuel system's que no estén dentro de 'mpfi' y '2
'''
df4['fuelsystem'] = df4['fuelsystem'].apply(lambda x: 'other' if x not in ['
```

```
In [30]: df4['aspiration'].value_counts()
```

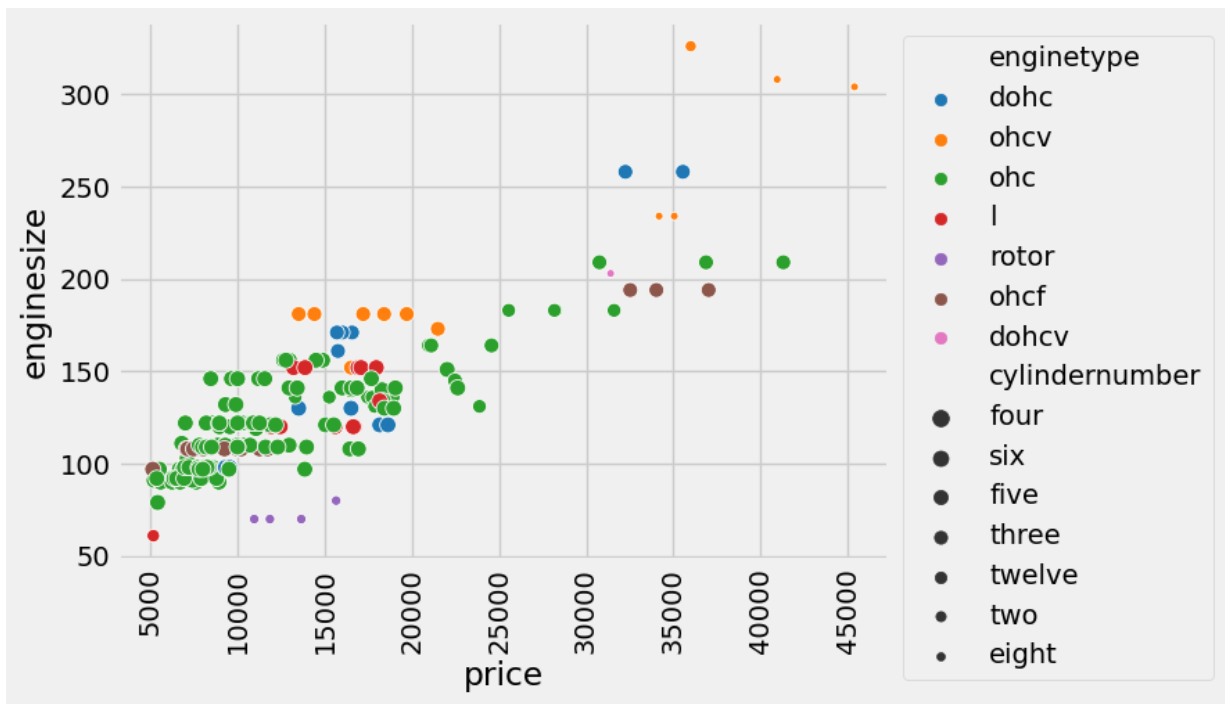
```
Out[30]: aspiration
std      168
turbo     37
Name: count, dtype: int64
```

```
In [31]: df['carbody'].value_counts()
```

```
Out[31]: carbody
sedan      96
hatchback  70
wagon      25
hardtop     8
convertible 6
Name: count, dtype: int64
```

Esta tambien contiene variables con muy pocos valores, pero no puedo reducirlas porque se encuentran muy dispersas en nuestra gráfica de correlación con respecto a la variable 'price' , y puesto que tenemos un dataset no muy grande, no creo que sea gran trabajo computacional el hacer OHE con estas 5 nuevas columnas

```
In [32]: """
        AQUI UTILIZAMOS LA VARIABLE OBJETIVO PRECIO, EL ENGINESIZE EN EL EJE Y,
        LOS DIVIDIMOS POR TAMAÑOS EN BASE AL NÚMERO DE CILINDROS, Y LE DIMOS UNA
        FUERA DEL GRÁFICO, Y LA MOSTRAMOS CON MATPLOTLIB
        """
sns.scatterplot(data=df3, x='price' ,y='enginesize', hue='enginetype',size='
plt.xticks(rotation=90)
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.show()
```



```
In [33]: df4['enginetype'].value_counts()
```

```
Out[33]: enginetype
ohc      148
ohcf     15
ohcv     13
dohc     12
l         12
rotor     4
dohcv     1
Name: count, dtype: int64
```

Bien, podemos ver que el enginetype tiene una alta correlación, pero no quiero que una columna como 'dohcv' influya en mi modelo. Podemos ver que los coches con dos cilindros manejan un tipo ingenieril (creo sería la traducción) de tipo rotor, y que los más comunes son los ohc, que creo son los que manejan la mayoría de coches con 4 cilindros.

no quiero cambiar 'rotor' por lo mismo, a pesar de tener muy pocos valores, pero 'dohcv' lo asignaré al grupo del 'ohcv' por su similitud con estos.

```
In [34]: # Antes de moverlo, quiero saber cuales son exactamente:
df4.query("enginetype=='dohcv' | enginetype=='rotor'")
```

Out [34]:

	CarName	fueltype	aspiration	carbody	wheelbase	carlength	carwidth	carheight
car_ID								
56	mazda 626	gas	std	hatchback	95.3000	169.0000	65.7000	49.6
57	mazda glc	gas	std	hatchback	95.3000	169.0000	65.7000	49.6
58	mazda rx-7 gs	gas	std	hatchback	95.3000	169.0000	65.7000	49.6
59	mazda glc 4	gas	std	hatchback	95.3000	169.0000	65.7000	49.6
130	porsche cayenne	gas	std	hatchback	98.4000	175.7000	72.3000	50.5

In [35]: `df4['enginetype'] = df4['enginetype'].apply(lambda x: 'ohcv' if x=='dohcv' else 'ohcv')`In [36]: `df4.head()`

Out [36]:

	CarName	fueltype	aspiration	carbody	wheelbase	carlength	carwidth	carheight
car_ID								
1	alfa-romero giulia	gas	std	convertible	88.6000	168.8000	64.1000	
2	alfa-romero stelvio	gas	std	convertible	88.6000	168.8000	64.1000	
3	alfa-romero Quadrifoglio	gas	std	hatchback	94.5000	171.2000	65.5000	
4	audi 100 ls	gas	std	sedan	99.8000	176.6000	66.2000	
5	audi 100ls	gas	std	sedan	99.4000	176.6000	66.4000	

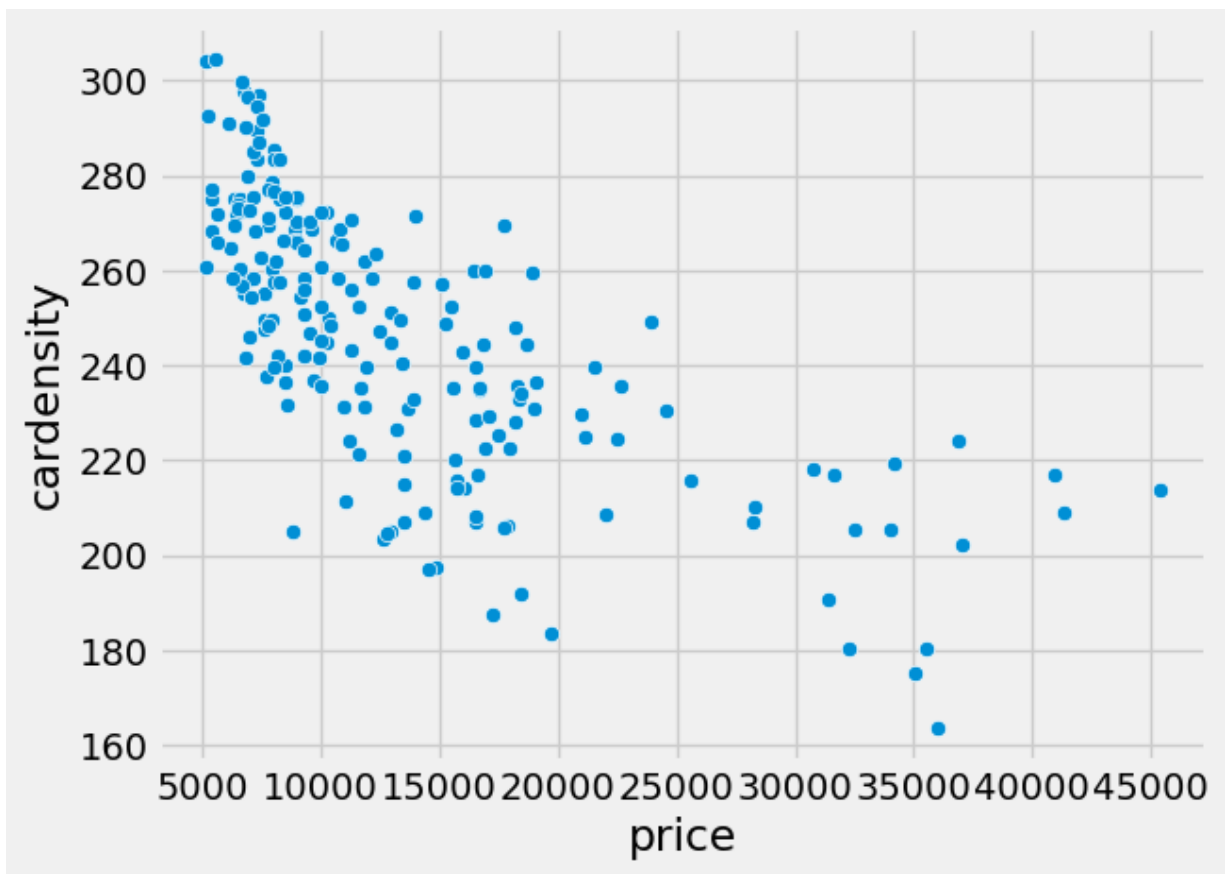
## 2. Feature Engineering

Bien! Recuerdan porque no eliminé la variable de 'carheight' al inicio?? Vamos a crear 2 nuevas columnas útiles creo yo, para nuestro modelo. Utilizaremos el volumen de un coche y su densidad para complementarlo.

In [37]: `df4['carvolume'] = df4['carlength'] * df4['carwidth'] * df4['carheight']  
df4['cardensity'] = df4['carvolume'] / df4['curbweight']`In [38]: `df4.head()`

Out[38]:

	CarName	fueltype	aspiration	carbody	wheelbase	carlength	carwidth	ca
car_ID								
1	alfa-romero giulia	gas	std	convertible	88.6000	168.8000	64.1000	
2	alfa-romero stelvio	gas	std	convertible	88.6000	168.8000	64.1000	
3	alfa-romero Quadrifoglio	gas	std	hatchback	94.5000	171.2000	65.5000	
4	audi 100 ls	gas	std	sedan	99.8000	176.6000	66.2000	
5	audi 100ls	gas	std	sedan	99.4000	176.6000	66.4000	

In [39]: `sns.scatterplot(data=df4, x='price', y='cardensity')`Out[39]: `<Axes: xlabel='price', ylabel='cardensity'>`

Podemos ver una relación! a menor densidad en el coche, mayor su precio, aunque llega un punto en la densidad, arriba de 220 peso/volumen aprox (no especifican unidades), donde la curva se estabiliza. Sin embargo, hay una correlacion lineal bajo este numero, y esto es lo que influirá en nuestro modelo :)

In [40]: `'''get_dummies es el metodo de pandas para hacer OHE. Creo no es la mejor pra  
df4.iloc[:, 1:] hace list slicing, donde incluyo todas las filas, y todas la  
'''`



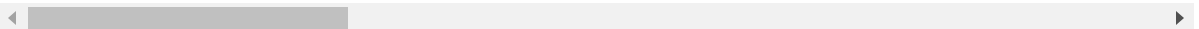
```
ohe = pd.get_dummies(df4.iloc[:, 1:].select_dtypes(include='object'))
ohe
```

Out[40]:

	fueltype_diesel	fueltype_gas	aspiration_std	aspiration_turbo	carbody_convertibl
car_ID					

1	False	True	True	False	Tru
2	False	True	True	False	Tru
3	False	True	True	False	Fals
4	False	True	True	False	Fals
5	False	True	True	False	Fals
...	...	...	...	...	
201	False	True	True	False	Fals
202	False	True	False	True	Fals
203	False	True	True	False	Fals
204	True	False	False	True	Fals
205	False	True	False	True	Fals

205 rows × 18 columns



Me gusta!! Esto ya solo lo concateno a mis datos para comenzar a hacer el modelo:

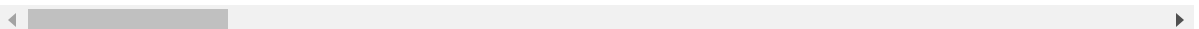
In [41]: `df5 = pd.concat([df4.select_dtypes(include='number'), ohe], axis='columns')`

In [42]: `df5.head()`

Out[42]:

	wheelbase	carlength	carwidth	carheight	curbweight	engineize	boreratio	ho
car_ID								

1	88.6000	168.8000	64.1000	48.8000	2548	130	3.4700	
2	88.6000	168.8000	64.1000	48.8000	2548	130	3.4700	
3	94.5000	171.2000	65.5000	52.4000	2823	152	2.6800	
4	99.8000	176.6000	66.2000	54.3000	2337	109	3.1900	
5	99.4000	176.6000	66.4000	54.3000	2824	136	3.1900	



In [43]: `df5.info()`

```

<class 'pandas.core.frame.DataFrame'>
Index: 205 entries, 1 to 205
Data columns (total 33 columns):
 #   Column                                Non-Null Count  Dtype
---  -
 0   wheelbase                            205 non-null    float64
 1   carlength                            205 non-null    float64
 2   carwidth                             205 non-null    float64
 3   carheight                           205 non-null    float64
 4   curbweight                           205 non-null    int64
 5   enginesize                           205 non-null    int64
 6   boreratio                            205 non-null    float64
 7   horsepower                           205 non-null    int64
 8   citympg                             205 non-null    int64
 9   highwaympg                          205 non-null    int64
10   price                               205 non-null    float64
11   drivewheel_rwd                      205 non-null    int64
12   oe_cylindernumber                   205 non-null    float64
13   carvolume                           205 non-null    float64
14   cardensity                          205 non-null    float64
15   fueltype_diesel                     205 non-null    bool
16   fueltype_gas                        205 non-null    bool
17   aspiration_std                      205 non-null    bool
18   aspiration_turbo                    205 non-null    bool
19   carbody_convertible                 205 non-null    bool
20   carbody_hardtop                     205 non-null    bool
21   carbody_hatchback                   205 non-null    bool
22   carbody_sedan                       205 non-null    bool
23   carbody_wagon                       205 non-null    bool
24   enginetype_dohc                     205 non-null    bool
25   enginetype_l                        205 non-null    bool
26   enginetype_ohc                      205 non-null    bool
27   enginetype_ohcf                     205 non-null    bool
28   enginetype_ohcv                     205 non-null    bool
29   enginetype_rotor                    205 non-null    bool
30   fuelsystem_2bbl                     205 non-null    bool
31   fuelsystem_mpf                      205 non-null    bool
32   fuelsystem_other                    205 non-null    bool
dtypes: bool(18), float64(9), int64(6)
memory usage: 29.2 KB

```

```

In [44]: X = df5.drop('price', axis='columns')
         y = df5['price']

```

### 3. Creación, validación y ajustes de hiperparámetros del modelo

Aquí tengo que ser sincero. No soy experto en esto aún, pero me he dado cuenta que lo más importante para este modelo ya lo hemos hecho. Cuando la gente se refiere a "construir un modelo", realmente se refiere a limpiar datos, crear columnas y seleccionar las columnas para tu modelo, porque el background matemático y las formulas necesarias ya se encuentran cargadas en sklearn y algunas librerías como XGBoost, donde tú solo tienes que ajustarlos con el método `fit`

```
In [45]: # Aquí separamos los datos en datos de entrenamiento y prueba
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran
```

```
In [46]: from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
from sklearn.ensemble import RandomForestRegressor

# Modelo de Regresión con Random Forest
rf_model = RandomForestRegressor(n_estimators=1000, max_depth=10, min_sample
rf_model.fit(X_train, y_train)
y_pred_rf_regression = rf_model.predict(X_test)
print(f"Random Forest Score = {r2_score(y_test,y_pred_rf_regression)*100}")
```

Random Forest Score = 95.8511607473852

```
In [47]: rf_model.score(X_test,y_test) * 100
```

Out[47]: 95.8511607473852

Wow!! casi 96% n.n Listo, para Random Forest esto fue todo! Estuve experimentando con los hiperparámetros para ver cual me daba mejor resultado, y creo que a 1000 arboles de decisión, profundidad de las ramas de 10, el modelo predice mejor

```
In [48]: # Hice una prueba para ver la diferencia entre el valor real y el predicho
pred_0 = rf_model.predict([X_test.iloc[0]])
print(f"el valor real del coche es: {y_test.iloc[0]}, y su valor predicho fu
```

el valor real del coche es: 30760.0, y su valor predicho fue: [34941.036]

/Users/daniboy/miniconda3/envs/ML/lib/python3.11/site-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but RandomForestRegressor was fitted with feature names  
warnings.warn(

```
In [49]: print("RMSE", np.sqrt(mean_squared_error(y_test,y_pred_rf_regression)))
```

RMSE 1809.7680862054306

```
In [50]: df['price'].mean()
```

Out[50]: 13276.710570731706

Personalmente, no me gusta utilizar modelos de regresión lineal y logística, puesto que rara vez nos serán los más útiles en análisis multivariantes. Son esenciales creo yo para entender como funciona overfitting y underfitting, pero una vez entiendes la relación entre sesgo y varianza creo puedes pasar de ellos. En cambio, prefiero los modelos que se

basan en decision trees, y aún más, aquellos que se basan en el error de los decision trees, como lo es XGBoost. Tiene demasiados hiperparámetros, pero es considerado uno de los mejores algoritmos de aprendizaje supervisado, es decir, cuando tenemos una variable objetivo, en este caso "price". Y si, estos pueden llevar a mayor coste computacional quizá, pero sabiendo controlar sus hiperparametros, y conociendo las capacidades de tu computadora, puedes lograr muy buenos resultados en muy poco tiempo. Al final, estas exprimiendo toda la matemática a que la haga tu computadora

```
In [51]: import xgboost as xgb

# Modelo de Regresión con XGBoost
xgboost_regressor = xgb.XGBRegressor(
    n_estimators=100, max_depth=3, learning_rate=0.1, subsample=1.0,
    colsample_bytree=0.5, gamma=0.0, random_state=42)
xgboost_regressor.fit(X_train, y_train)
y_pred_xgboost_regression = xgboost_regressor.predict(X_test)
mse_xgboost_regression = mean_squared_error(y_test, y_pred_xgboost_regression)
print("MAE (XGBoost Regresión):", mse_xgboost_regression)
```

MAE (XGBoost Regresión): 5358400.102837478

```
In [52]: print("RMSE (XGBoost Regresión):", np.sqrt(mse_xgboost_regression))
```

RMSE (XGBoost Regresión): 2314.821829609674

```
In [53]: y_pred_xgboost_regression = xgboost_regressor.predict(X_test)
print(f"XGBoost Score = {r2_score(y_test,y_pred_xgboost_regression)*100}")
```

XGBoost Score = 93.2124053941896

Bien, podemos ver un mejor score con Random Forest, pero la verdad aún no entiendo bien como modificar los hiperparametros en XGBoost. Así que aún no diré que fue mejor.

Okey, aquí ya estuve investigando como mejorar los hiperparámetros. Basicamente el algoritmo crea 1,000 arboles de decisión, y el `early_stopping_rounds=10` nos dice que cuando el 'rmse' (Root Mean Squared Error) permanezca aumentando después de 10 iteraciones más, termine el algoritmo con la menor. `n_jobs=2` es el número de cores del procesador con los que quieres que el algoritmo utilice la computación paralela, esto acelera el proceso, pero se recomienda usarlo con respecto a los cores del procesador de tu computadora, que en mi caso son dos. El learning rate son los baby steps para llegar al mínimo global, `max_depth=3` es la profundidad de los decisión trees generados, y el `random_state=42` es la semilla aleatoria para el algoritmo. Por convención se utiliza el 42, y fue el mismo que utilicé para los 3 algoritmos

Ya dentro del metodo `fit`, lo ajustamos con los datos de entrenamiento, y lo evaluamos con los de prueba. `verbose` nos imprime en pantalla cada 'n' veces el "rmse", o sea, los pasos que el algoritmo está dando en busca de ese mínimo global, que son un poco conceptos ya de Machine Learning. Para explicarlo mejor, "RMSE" refiere de ROOT MEAN SQUARED ERROR, que quizá es poco espacio para explicarlo aquí, pero en pocas palabras,

la raíz es, porque el modelo necesita elevar al cuadrado para darle mayor peso a los valores mayores y menores del error entre el valor predicho y el real.

```
In [54]: reg = xgb.XGBRegressor(n_estimators=1000, early_stopping_rounds=5, n_jobs=2,
                                learning_rate=0.03, max_depth=3, random_state=42)
reg.fit(X_train, y_train,
        eval_set=[(X_train, y_train), (X_test, y_test)],
        verbose=100)
```

```
[0]    validation_0-rmse:14884.79548    validation_1-rmse:15680.81953
[100]   validation_0-rmse:1700.90589    validation_1-rmse:2412.22755
[145]   validation_0-rmse:1240.53599    validation_1-rmse:2279.49714
```

```
Out[54]: XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=5,
              enable_categorical=False, eval_metric=None, feature_
types=None,
              gamma=None, gpu_id=None, grow_policy=None, importanc
e_type=None,
              interaction_constraints=None, learning_rate=0.03, ma
x_bin=None,
```

```
In [55]: # Obtener el último valor del RMSE en el conjunto de validación
final_rmse = reg.evals_result_['validation_1']['rmse'][-1]

print(f"RMSE en el conjunto de validación: {final_rmse}")
```

RMSE en el conjunto de validación: 2281.520712489974

```
In [56]: from sklearn.metrics import r2_score

y_pred = reg.predict(X_test)
r2 = r2_score(y_test, y_pred)

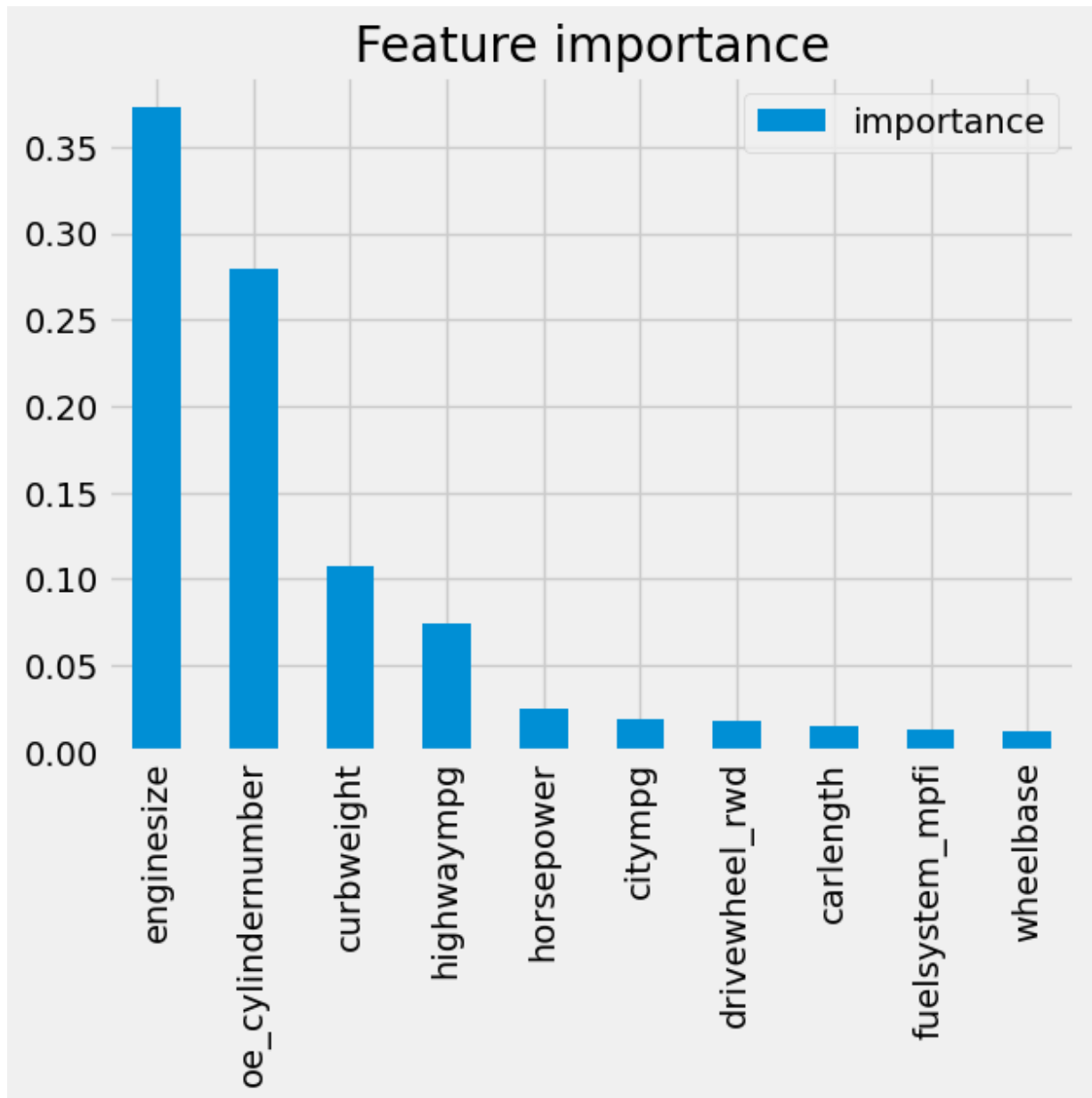
print(f"R2 en el conjunto de prueba: {r2}")
```

R2 en el conjunto de prueba: 0.9342391658172655

```
In [57]: fi = pd.DataFrame(data=reg.feature_importances_,
                           index=reg.feature_names_in_,
                           columns=['importance'])
```

```
In [58]: # Esto nos muestra el peso que tomó el algoritmo para crear los árboles de d
fi.sort_values('importance', ascending=False).head(10).plot(kind='bar')
plt.title('Feature importance')
```

```
Out[58]: Text(0.5, 1.0, 'Feature importance')
```



Bien, había encontrado un modelo con casi 96% de accuracy, cierto?? Pues, ahora crearé una estructura más estandarizada para encontrar el mejor modelo. Es la primera vez que pongo en practica librerias como "LGBMRegressor", "GridSearchCV", e intentaré explicarlas para una mejor comprensión.

```
In [59]: from sklearn.model_selection import GridSearchCV
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
```

```
In [60]: """
        AQUI HAY QUE TENER CUIDADO, PORQUE POR CADA PARAMETRO EXTRA QUE AGREGUEM
        HACIENDO QUE SEA MUCHA CARGA PARA EL PROCESADOR, ASI QUE EN ESTE CASO UT
        EL MINIMO DE SEPARACIONES, Y EL MINIMO DE HOJAS, LOS DEJE EN SOLO UN VAL
        FOREST TUVO EL MAYOR SCORE, EMPEZARÉ POR ESTE
        """
```

```
param_grid_rf = {  
    'n_estimators': [100, 500],  
    'max_depth': [10, 30, 50],  
    'min_samples_split': [1,2],  
    'min_samples_leaf': [1,2],  
}  
  
# 1 Random Forest Regressor() X 2 valores para el numero de arboles del modelo  
# Esto multiplicado 5 veces los K-folds que utilizo el Cross Validation para  
# 1 X 2 X 3 X 1 X 1 X 5 = 30 RandomForestRegressor()  
grid_search_rf = GridSearchCV(  
    estimator=RandomForestRegressor(),  
    param_grid=param_grid_rf,  
    scoring='neg_mean_squared_error',  
    cv=5,  
    verbose=5  
)
```

```
In [61]: # Nuevamente ajustamos el modelo, pero con cada uno de los modelos  
grid_search_rf.fit(X_train, y_train)
```

Fitting 5 folds for each of 24 candidates, totalling 120 fits

```
[CV 1/5] END max_depth=10, min_samples_leaf=1, min_samples_split=1, n_estimators=100;; score=nan total time= 0.0s
[CV 2/5] END max_depth=10, min_samples_leaf=1, min_samples_split=1, n_estimators=100;; score=nan total time= 0.0s
[CV 3/5] END max_depth=10, min_samples_leaf=1, min_samples_split=1, n_estimators=100;; score=nan total time= 0.0s
[CV 4/5] END max_depth=10, min_samples_leaf=1, min_samples_split=1, n_estimators=100;; score=nan total time= 0.0s
[CV 5/5] END max_depth=10, min_samples_leaf=1, min_samples_split=1, n_estimators=100;; score=nan total time= 0.0s
[CV 1/5] END max_depth=10, min_samples_leaf=1, min_samples_split=1, n_estimators=500;; score=nan total time= 0.0s
[CV 2/5] END max_depth=10, min_samples_leaf=1, min_samples_split=1, n_estimators=500;; score=nan total time= 0.0s
[CV 3/5] END max_depth=10, min_samples_leaf=1, min_samples_split=1, n_estimators=500;; score=nan total time= 0.0s
[CV 4/5] END max_depth=10, min_samples_leaf=1, min_samples_split=1, n_estimators=500;; score=nan total time= 0.0s
[CV 5/5] END max_depth=10, min_samples_leaf=1, min_samples_split=1, n_estimators=500;; score=nan total time= 0.0s
[CV 1/5] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100;; score=-4702676.362 total time= 0.4s
[CV 2/5] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100;; score=-2134280.717 total time= 0.4s
[CV 3/5] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100;; score=-8745591.480 total time= 0.5s
[CV 4/5] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100;; score=-4160356.747 total time= 0.4s
[CV 5/5] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100;; score=-10220286.032 total time= 0.4s
[CV 1/5] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=500;; score=-4722302.161 total time= 1.7s
[CV 2/5] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=500;; score=-2208608.667 total time= 1.2s
[CV 3/5] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=500;; score=-8898086.425 total time= 1.3s
[CV 4/5] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=500;; score=-4180523.297 total time= 1.1s
[CV 5/5] END max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=500;; score=-9770311.451 total time= 1.1s
[CV 1/5] END max_depth=10, min_samples_leaf=2, min_samples_split=1, n_estimators=100;; score=nan total time= 0.0s
[CV 2/5] END max_depth=10, min_samples_leaf=2, min_samples_split=1, n_estimators=100;; score=nan total time= 0.0s
[CV 3/5] END max_depth=10, min_samples_leaf=2, min_samples_split=1, n_estimators=100;; score=nan total time= 0.0s
[CV 4/5] END max_depth=10, min_samples_leaf=2, min_samples_split=1, n_estimators=100;; score=nan total time= 0.0s
[CV 5/5] END max_depth=10, min_samples_leaf=2, min_samples_split=1, n_estimators=100;; score=nan total time= 0.0s
[CV 1/5] END max_depth=10, min_samples_leaf=2, min_samples_split=1, n_estimators=500;; score=nan total time= 0.0s
[CV 2/5] END max_depth=10, min_samples_leaf=2, min_samples_split=1, n_estimators=500;; score=nan total time= 0.0s
[CV 3/5] END max_depth=10, min_samples_leaf=2, min_samples_split=1, n_estimators=500;; score=nan total time= 0.0s
```



```
tors=500;; score=nan total time= 0.0s
[CV 4/5] END max_depth=10, min_samples_leaf=2, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 5/5] END max_depth=10, min_samples_leaf=2, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 1/5] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estima
tors=100;; score=-5176286.057 total time= 0.2s
[CV 2/5] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estima
tors=100;; score=-2482237.062 total time= 0.2s
[CV 3/5] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estima
tors=100;; score=-9096091.548 total time= 0.2s
[CV 4/5] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estima
tors=100;; score=-3731734.402 total time= 0.2s
[CV 5/5] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estima
tors=100;; score=-9872464.916 total time= 0.2s
[CV 1/5] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estima
tors=500;; score=-5036976.065 total time= 1.2s
[CV 2/5] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estima
tors=500;; score=-2593666.152 total time= 1.0s
[CV 3/5] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estima
tors=500;; score=-9336165.969 total time= 0.9s
[CV 4/5] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estima
tors=500;; score=-3644898.635 total time= 0.9s
[CV 5/5] END max_depth=10, min_samples_leaf=2, min_samples_split=2, n_estima
tors=500;; score=-9700703.326 total time= 1.2s
[CV 1/5] END max_depth=30, min_samples_leaf=1, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 2/5] END max_depth=30, min_samples_leaf=1, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 3/5] END max_depth=30, min_samples_leaf=1, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 4/5] END max_depth=30, min_samples_leaf=1, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 5/5] END max_depth=30, min_samples_leaf=1, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 1/5] END max_depth=30, min_samples_leaf=1, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 2/5] END max_depth=30, min_samples_leaf=1, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 3/5] END max_depth=30, min_samples_leaf=1, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 4/5] END max_depth=30, min_samples_leaf=1, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 5/5] END max_depth=30, min_samples_leaf=1, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 1/5] END max_depth=30, min_samples_leaf=1, min_samples_split=2, n_estima
tors=100;; score=-4989034.880 total time= 0.4s
[CV 2/5] END max_depth=30, min_samples_leaf=1, min_samples_split=2, n_estima
tors=100;; score=-2428693.531 total time= 0.3s
[CV 3/5] END max_depth=30, min_samples_leaf=1, min_samples_split=2, n_estima
tors=100;; score=-9268862.960 total time= 0.3s
[CV 4/5] END max_depth=30, min_samples_leaf=1, min_samples_split=2, n_estima
tors=100;; score=-4268049.797 total time= 0.3s
[CV 5/5] END max_depth=30, min_samples_leaf=1, min_samples_split=2, n_estima
tors=100;; score=-9651763.680 total time= 0.2s
[CV 1/5] END max_depth=30, min_samples_leaf=1, min_samples_split=2, n_estima
```

```
tors=500;; score=-4770788.735 total time= 1.6s
[CV 2/5] END max_depth=30, min_samples_leaf=1, min_samples_split=2, n_estima
tors=500;; score=-2148143.438 total time= 1.8s
[CV 3/5] END max_depth=30, min_samples_leaf=1, min_samples_split=2, n_estima
tors=500;; score=-9068478.118 total time= 3.2s
[CV 4/5] END max_depth=30, min_samples_leaf=1, min_samples_split=2, n_estima
tors=500;; score=-4378002.127 total time= 2.9s
[CV 5/5] END max_depth=30, min_samples_leaf=1, min_samples_split=2, n_estima
tors=500;; score=-10112048.587 total time= 3.0s
[CV 1/5] END max_depth=30, min_samples_leaf=2, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 2/5] END max_depth=30, min_samples_leaf=2, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 3/5] END max_depth=30, min_samples_leaf=2, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 4/5] END max_depth=30, min_samples_leaf=2, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 5/5] END max_depth=30, min_samples_leaf=2, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 1/5] END max_depth=30, min_samples_leaf=2, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 2/5] END max_depth=30, min_samples_leaf=2, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 3/5] END max_depth=30, min_samples_leaf=2, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 4/5] END max_depth=30, min_samples_leaf=2, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 5/5] END max_depth=30, min_samples_leaf=2, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 1/5] END max_depth=30, min_samples_leaf=2, min_samples_split=2, n_estima
tors=100;; score=-5096499.191 total time= 0.6s
[CV 2/5] END max_depth=30, min_samples_leaf=2, min_samples_split=2, n_estima
tors=100;; score=-2627057.601 total time= 0.4s
[CV 3/5] END max_depth=30, min_samples_leaf=2, min_samples_split=2, n_estima
tors=100;; score=-9594191.815 total time= 0.3s
[CV 4/5] END max_depth=30, min_samples_leaf=2, min_samples_split=2, n_estima
tors=100;; score=-3714874.958 total time= 0.3s
[CV 5/5] END max_depth=30, min_samples_leaf=2, min_samples_split=2, n_estima
tors=100;; score=-9728350.544 total time= 0.7s
[CV 1/5] END max_depth=30, min_samples_leaf=2, min_samples_split=2, n_estima
tors=500;; score=-4891012.149 total time= 2.0s
[CV 2/5] END max_depth=30, min_samples_leaf=2, min_samples_split=2, n_estima
tors=500;; score=-2804561.333 total time= 2.2s
[CV 3/5] END max_depth=30, min_samples_leaf=2, min_samples_split=2, n_estima
tors=500;; score=-9238688.884 total time= 1.8s
[CV 4/5] END max_depth=30, min_samples_leaf=2, min_samples_split=2, n_estima
tors=500;; score=-3795554.302 total time= 1.9s
[CV 5/5] END max_depth=30, min_samples_leaf=2, min_samples_split=2, n_estima
tors=500;; score=-10141052.114 total time= 2.4s
[CV 1/5] END max_depth=50, min_samples_leaf=1, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 2/5] END max_depth=50, min_samples_leaf=1, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 3/5] END max_depth=50, min_samples_leaf=1, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 4/5] END max_depth=50, min_samples_leaf=1, min_samples_split=1, n_estima
```

```
tors=100;; score=nan total time= 0.0s
[CV 5/5] END max_depth=50, min_samples_leaf=1, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 1/5] END max_depth=50, min_samples_leaf=1, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 2/5] END max_depth=50, min_samples_leaf=1, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 3/5] END max_depth=50, min_samples_leaf=1, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 4/5] END max_depth=50, min_samples_leaf=1, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 5/5] END max_depth=50, min_samples_leaf=1, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 1/5] END max_depth=50, min_samples_leaf=1, min_samples_split=2, n_estima
tors=100;; score=-4650003.181 total time= 0.3s
[CV 2/5] END max_depth=50, min_samples_leaf=1, min_samples_split=2, n_estima
tors=100;; score=-2245936.457 total time= 0.2s
[CV 3/5] END max_depth=50, min_samples_leaf=1, min_samples_split=2, n_estima
tors=100;; score=-9266065.473 total time= 0.5s
[CV 4/5] END max_depth=50, min_samples_leaf=1, min_samples_split=2, n_estima
tors=100;; score=-4671414.876 total time= 0.3s
[CV 5/5] END max_depth=50, min_samples_leaf=1, min_samples_split=2, n_estima
tors=100;; score=-9540729.213 total time= 0.4s
[CV 1/5] END max_depth=50, min_samples_leaf=1, min_samples_split=2, n_estima
tors=500;; score=-4788771.253 total time= 1.6s
[CV 2/5] END max_depth=50, min_samples_leaf=1, min_samples_split=2, n_estima
tors=500;; score=-2042083.971 total time= 1.6s
[CV 3/5] END max_depth=50, min_samples_leaf=1, min_samples_split=2, n_estima
tors=500;; score=-9038915.088 total time= 1.6s
[CV 4/5] END max_depth=50, min_samples_leaf=1, min_samples_split=2, n_estima
tors=500;; score=-4328669.751 total time= 1.3s
[CV 5/5] END max_depth=50, min_samples_leaf=1, min_samples_split=2, n_estima
tors=500;; score=-10196372.621 total time= 1.5s
[CV 1/5] END max_depth=50, min_samples_leaf=2, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 2/5] END max_depth=50, min_samples_leaf=2, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 3/5] END max_depth=50, min_samples_leaf=2, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 4/5] END max_depth=50, min_samples_leaf=2, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 5/5] END max_depth=50, min_samples_leaf=2, min_samples_split=1, n_estima
tors=100;; score=nan total time= 0.0s
[CV 1/5] END max_depth=50, min_samples_leaf=2, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 2/5] END max_depth=50, min_samples_leaf=2, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 3/5] END max_depth=50, min_samples_leaf=2, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 4/5] END max_depth=50, min_samples_leaf=2, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 5/5] END max_depth=50, min_samples_leaf=2, min_samples_split=1, n_estima
tors=500;; score=nan total time= 0.0s
[CV 1/5] END max_depth=50, min_samples_leaf=2, min_samples_split=2, n_estima
tors=100;; score=-4915168.810 total time= 0.2s
[CV 2/5] END max_depth=50, min_samples_leaf=2, min_samples_split=2, n_estima
```

```
tors=100;; score=-2275289.892 total time= 0.2s
[CV 3/5] END max_depth=50, min_samples_leaf=2, min_samples_split=2, n_estimators=100;; score=-9827942.668 total time= 0.2s
[CV 4/5] END max_depth=50, min_samples_leaf=2, min_samples_split=2, n_estimators=100;; score=-4177940.143 total time= 0.3s
[CV 5/5] END max_depth=50, min_samples_leaf=2, min_samples_split=2, n_estimators=100;; score=-10125910.430 total time= 0.3s
[CV 1/5] END max_depth=50, min_samples_leaf=2, min_samples_split=2, n_estimators=500;; score=-5074648.502 total time= 1.0s
[CV 2/5] END max_depth=50, min_samples_leaf=2, min_samples_split=2, n_estimators=500;; score=-2863415.791 total time= 1.0s
[CV 3/5] END max_depth=50, min_samples_leaf=2, min_samples_split=2, n_estimators=500;; score=-9149090.112 total time= 1.2s
[CV 4/5] END max_depth=50, min_samples_leaf=2, min_samples_split=2, n_estimators=500;; score=-3670006.067 total time= 1.1s
[CV 5/5] END max_depth=50, min_samples_leaf=2, min_samples_split=2, n_estimators=500;; score=-9636778.219 total time= 1.1s
```

```

/Users/daniboy/miniconda3/envs/ML/lib/python3.11/site-packages/sklearn/model
_selection/_validation.py:378: FitFailedWarning:
60 fits failed out of a total of 120.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting err
or_score='raise'.

```

Below are more details about the failures:

-----  
 ----  
 60 fits failed with the following error:

Traceback (most recent call last):

```

File "/Users/daniboy/miniconda3/envs/ML/lib/python3.11/site-packages/sklea
rn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)

```

```

File "/Users/daniboy/miniconda3/envs/ML/lib/python3.11/site-packages/sklea
rn/ensemble/_forest.py", line 340, in fit
    self._validate_params()

```

```

File "/Users/daniboy/miniconda3/envs/ML/lib/python3.11/site-packages/sklea
rn/base.py", line 600, in _validate_params
    validate_parameter_constraints(

```

```

File "/Users/daniboy/miniconda3/envs/ML/lib/python3.11/site-packages/sklea
rn/utils/_param_validation.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(

```

```

sklearn.utils._param_validation.InvalidParameterError: The 'min_samples_spli
t' parameter of RandomForestRegressor must be an int in the range [2, inf) o
r a float in the range (0.0, 1.0]. Got 1 instead.

```

```

warnings.warn(some_fits_failed_message, FitFailedWarning)

```

```

/Users/daniboy/miniconda3/envs/ML/lib/python3.11/site-packages/sklearn/model
_selection/_search.py:952: UserWarning: One or more of the test scores are n
on-finite: [
nan
nan -5992638.26747694 -5955966.4
0007379

```

```

nan
nan -6071762.79684481 -6062482.02944572
nan
nan -6121280.9695087 -6095492.20090572
nan
nan -6152194.82165734 -6174173.75651788
nan
nan -6074829.83989169 -6078962.53684951
nan
nan -6264450.38879523 -6078787.73815057]

```

```

warnings.warn(

```

Out[61]:

```

└─ GridSearchCV
  └─ estimator: RandomForestRegressor
    └─ RandomForestRegressor

```

In [62]:

```

# Encontramos el mejor modelo con el metodo '.best_estimator_'
best_model_rf = grid_search_rf.best_estimator_

```

In [63]:

```

best_model_rf.fit(X_train, y_train)

```

```
Out[63]: ▼ RandomForestRegressor
RandomForestRegressor(max_depth=10, n_estimators=500)
```

```
In [64]: # Y calculamos el porcentaje de error para el mejor Random Forest es:
print(f"El porcentaje de error para el mejor Random Forest es: {best_model_r
```

El porcentaje de error para el mejor Random Forest es: 95.84086066172479

Mejoramos un poco nuestro modelo!! 🤗

```
In [65]: """
HACEMOS LO MISMO PARA LOS OTROS DOS MODELOS:
... Si se preguntan porque estan comentados, es porque el metodo 'gridse
y los multiplica! Haciendo que se creen demasiados modelos (como en nues
tardar dependiendo de la maquina y el procesador. Aún así, si quieres ve
lineas
"""

# param_grid_xgb = {
#     'n_estimators': [100, 500],
#     'max_depth': [10, 50],
#     'learning_rate': [0.01, 0.1],
#     'min_child_weight': [1, 5],
# }

# grid_search_xgb = GridSearchCV(
#     estimator=XGBRegressor(),
#     param_grid=param_grid_xgb,
#     scoring='neg_mean_squared_error',
#     cv=5,
#     verbose=5
# )

# grid_search_xgb.fit(X_train, y_train)
```

```
Out[65]: "\n    HACEMOS LO MISMO PARA LOS OTROS DOS MODELOS:\n    ... Si se pregunta
n porque estan comentados, es porque el metodo 'gridsearchcv' itera por cad
a uno de los parametros\n    y los multiplica! Haciendo que se creen demasi
ados modelos (como en nuestro ejemplo con RandomForest), y esto puede\n
tardar dependiendo de la maquina y el procesador. Aún así, si quieres ver c
omo funciona, solo descomenta las siguientes\n    lineas\n"
```

```
In [66]: # best_model_xgb = grid_search_xgb.best_estimator_
# best_model_xgb.fit(X_train, y_train)
```

```
In [67]: ## Y calculamos el porcentaje de error para el mejor Random Forest es:
# print(f"El porcentaje de error para el mejor XGBRegressor es: {best_model_
```

```
In [68]: # param_grid_lgbm = {
#     'n_estimators': [100, 500],
#     'max_depth': [10],
#     'num_leaves': [2, 4],
```

```
# 'learning_rate': [0.01, 0.1],
# 'min_child_weight': [1, 5],
# }

# grid_search_lgbm = GridSearchCV(
#     estimator=LGBMRegressor(),
#     param_grid=param_grid_lgbm,
#     scoring='neg_mean_squared_error',
#     cv=5,
#     verbose=5
# )

# grid_search_lgbm.fit(X_train, y_train)
```

**NOTA:** Para la creacion de los dos modelos extra, me ahorre el paso y simplemente se lo pedí a ChatGPT

```
In [69]: # best_model_lgbm = grid_search_lgbm.best_estimator_
# best_model_lgbm.fit(X_train, y_train)
```

```
In [70]: ## Y calculamos el porcentaje de error para el mejor Random Forest es:
# print(f"El porcentaje de error para el mejor XGBRegressor es: {best_model_
```

---

Indiscutiblemente podemos ver que para este caso Random Forest fue el más adecuado. XGBoost mejora notablemente su rendimiento con datasets mas grandes, me di cuenta cuando aumenté train split de datos de 70% a 80%. LightGB fue el de menor rendimiento, para su defensa, aún no sé bien como ajustar sus hiperparámetros.

```
In [71]: print(f"Aunque aumentó solo un poco: {( 95.87059566993466-95.8511607473852
```

Aunque aumentó solo un poco: 0.02%, ahora tenemos un algoritmo más sólido y rígido puesto que fue expuesto a un cross-validation con 5 k-folds

```
In [72]: y_pred = best_model_rf.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

```
In [73]: rmse
```

```
Out[73]: 1812.0131974193896
```

Esto quiere decir, que el error promedio de nuestros datos varia por 1805!

---

Ahora, demos unas conclusiones en base a nuestro modelo

```
In [74]: print(f"El porcentaje de error para el mejor Random Forest con datos de prue
{best_model_rf.score(X_train, y_train) * 100}")
```

El porcentaje de error para el mejor Random Forest con datos de prueba es: 98.53509518778463

```
In [75]: print(f"El porcentaje de error para el mejor Random Forest es: {best_model_r
```

El porcentaje de error para el mejor Random Forest es: 95.84086066172479

Wow! Tenemos casi 98.5% de score en nuestro modelo de entrenamiento. Esto no significa el valor real de nuestro modelo puesto que puede estar **overfitted**, pero la relación con el score en el test set, me da muy buena impresión

Tener un algoritmo con 98.44% de predicción es realmente bueno, puesto que es el conjunto de prueba, nos demuestra que tenemos muy buena 'bias' -> sesgo. Si quiero mejorar el gap entre este y el score ya con `X_test`, que es cerca del 3%, conseguir más datos ayudaría al modelo, aunque podría intentar incrementar la regularización (`lambda`)

ChatGPT -> como incremento la regularización en un algoritmo de random forest

En los algoritmos Random Forest, la regularización se controla principalmente mediante los hiperparámetros relacionados con la complejidad del árbol, como `max_depth`, `min_samples_split`, `min_samples_leaf`, `max_features`, entre otros. Aumentar la regularización generalmente se refiere a limitar la profundidad del árbol y aumentar el número mínimo de muestras requeridas para dividir un nodo o formar una hoja.

Aquí hay algunos hiperparámetros que puedes ajustar para aumentar la regularización en un algoritmo de Random Forest:

- `max_depth` : Este hiperparámetro controla la profundidad máxima de los árboles en el bosque. Reducir `max_depth` limitará la profundidad de los árboles y aumentará la regularización.
- `min_samples_split` : Este hiperparámetro establece el número mínimo de muestras requeridas para dividir un nodo interno del árbol. Aumentar este valor hará que los árboles sean menos profundos y aumentará la regularización.
- `min_samples_leaf` : Este hiperparámetro establece el número mínimo de muestras requeridas para formar una hoja del árbol. Aumentar este valor generará hojas más pequeñas y aumentará la regularización.
- `max_features` : Este hiperparámetro controla la cantidad máxima de características a considerar al dividir un nodo. Reducir este valor puede aumentar la regularización al limitar la selección de características en cada división.

```
In [76]: best_model_rf
```



Out [76]:

▼ RandomForestRegressor

RandomForestRegressor(max\_depth=10, n\_estimators=500)

Puesto que nuestro modelo fue de max\_depth=30 y n\_estimators=500, vamos a experimentar nuevamente con sus hiperparametros pero tomando en cuenta la regularización que nos proporcionó ChatGPT.

In [77]:

```
# param_grid_rf_reg = {
#     'n_estimators': [100],
#     'max_depth': [30,35],
#     'min_samples_split': [1,2],
#     'min_samples_leaf': [2,3,4],
#     'max_features': [5,10]
# }

## 1 Random Forest Regressor() X 1 valor para el numero de arboles del modelo
## 2 ramas, 3 hojas y 2 maximo numero de caracteristicas. Esto multiplicado
## Cross Validation para encontrar un algoritmo más rígido. De nuevo, terminamos
## 1 X 1 X 2 X 2 X 3 X 2 X 5 = 120 RandomForestRegressor() distintos para evaluar
# grid_search_rf_reg = GridSearchCV(
#     estimator=RandomForestRegressor(),
#     param_grid=param_grid_rf_reg,
#     scoring='neg_mean_squared_error',
#     cv=5,
#     verbose=5
# )
```

In [78]:

```
# grid_search_rf_reg.fit(X_train,y_train)
```

In [79]:

```
# best_model_rf_reg = grid_search_rf_reg.best_estimator_
```

In [80]:

```
# best_model_rf_reg
```

In [81]:

```
# best_model_rf_reg.score(X_train,y_train)
```

In [82]:

```
# best_model_rf_reg.score(X_test,y_test)
```

## Qué pasó aquí??

Bien, como ya expliqué, creé varios modelos para encontrar el que tuviera el mayor score, y puesto que quise que el modelo encontrara un mayor porcentaje con respecto al visto en el score con el "Training-set", tuve de dos: Encontrar mas datos, o aumentar su regularización. Trágicamente no pude, como les dije aún no tengo gran experiencia con esto, y es por eso que utilizaremos el mejor modelo que ya habíamos creado pasos atrás.

In [83]:

```
best_model_rf.score(X_test,y_test)
```

Out[83]: 0.9584086066172479

```
In [84]: y_pred = best_model_rf.predict(X_test)
y_pred
```

```
Out[84]: array([34654.165      , 18165.58361984,  9139.32568077, 13010.93816786,
        28499.46338242,  6309.77579127,  7820.63078977,  7986.15285131,
        9963.68960863,  8114.90702627, 14793.06619292,  7920.65273305,
        14980.45356528, 10969.55625271,  40071.893      ,  6301.98923088,
        5660.48634286, 13864.65283311,  8552.76638205,  9646.13605047,
        10403.96816825, 15340.28446818,  7050.05703822,  5720.69273333,
        7395.89291578,  34597.706      ,  9450.74765264, 16512.37842907,
        7175.40836369, 16270.03675209, 28574.88988242,  6374.00916342,
        8060.772808   , 18916.63064172,  8042.56179683, 28368.50825742,
        10989.29306416, 12262.60104242,  7582.55660446, 14581.87246089,
        8529.31266258])
```

```
In [85]: y_test
```

```
Out[85]: car_ID
16      30,760.0000
10      17,859.1670
101     9,549.0000
133     11,850.0000
69      28,248.0000
96       7,799.0000
160     7,788.0000
163     9,258.0000
148     10,198.0000
183     7,775.0000
192     13,295.0000
165     8,238.0000
66      18,280.0000
176     9,988.0000
74      40,960.0000
153     6,488.0000
19       5,151.0000
83      12,629.0000
87       8,189.0000
144     9,960.0000
61       8,495.0000
102     13,499.0000
99       8,249.0000
31       6,479.0000
26       6,692.0000
17      41,315.0000
169     9,639.0000
196     13,415.0000
98       7,999.0000
195     12,940.0000
68      25,552.0000
121     6,229.0000
155     7,898.0000
203     21,485.0000
80       7,689.0000
70      28,176.0000
146     11,259.0000
56      10,945.0000
46       8,916.5000
85      14,489.0000
147     7,463.0000
Name: price, dtype: float64
```

```
In [86]: Resultados_Modelo = pd.DataFrame({'Precio_Predicho':y_pred, 'Precio_Real': y
```

```
In [87]: Resultados_Modelo
```

Out[87]:

	Precio_Predicho	Precio_Real
car_ID		
10	18,165.5836	17,859.1670
16	34,654.1650	30,760.0000
17	34,597.7060	41,315.0000
19	5,660.4863	5,151.0000
26	7,395.8929	6,692.0000
31	5,720.6927	6,479.0000
46	7,582.5566	8,916.5000
56	12,262.6010	10,945.0000
61	10,403.9682	8,495.0000
66	14,980.4536	18,280.0000
68	28,574.8899	25,552.0000
69	28,499.4634	28,248.0000
70	28,368.5083	28,176.0000
74	40,071.8930	40,960.0000
80	8,042.5618	7,689.0000
83	13,864.6528	12,629.0000
85	14,581.8725	14,489.0000
87	8,552.7664	8,189.0000
96	6,309.7758	7,799.0000
98	7,175.4084	7,999.0000
99	7,050.0570	8,249.0000
101	9,139.3257	9,549.0000
102	15,340.2845	13,499.0000
121	6,374.0092	6,229.0000
133	13,010.9382	11,850.0000
144	9,646.1361	9,960.0000
146	10,989.2931	11,259.0000
147	8,529.3127	7,463.0000
148	9,963.6896	10,198.0000
153	6,301.9892	6,488.0000
155	8,060.7728	7,898.0000
160	7,820.6308	7,788.0000
163	7,986.1529	9,258.0000

	Precio_Predicho	Precio_Real
car_ID		
165	7,920.6527	8,238.0000
169	9,450.7477	9,639.0000
176	10,969.5563	9,988.0000
183	8,114.9070	7,775.0000
192	14,793.0662	13,295.0000
195	16,270.0368	12,940.0000
196	16,512.3784	13,415.0000
203	18,916.6306	21,485.0000

In [88]: `df.shape`

Out[88]: (205, 25)

Bien! Como pueden ver, solo tengo 41 rows, te un dataset que tenía 205. Esto es porque cuando hice split con el train / test, solo incluí el 20%. Ahora, no quise hacer el valor predicho del modelo en mis datos de prueba porque sería un valor injusto a los nuevos datos, como sí lo es nuestro 'y\_pred'. Ahora quiero que vean la correlación entre ambos n.n

In [89]: `Resultados_Modelo.corr()`

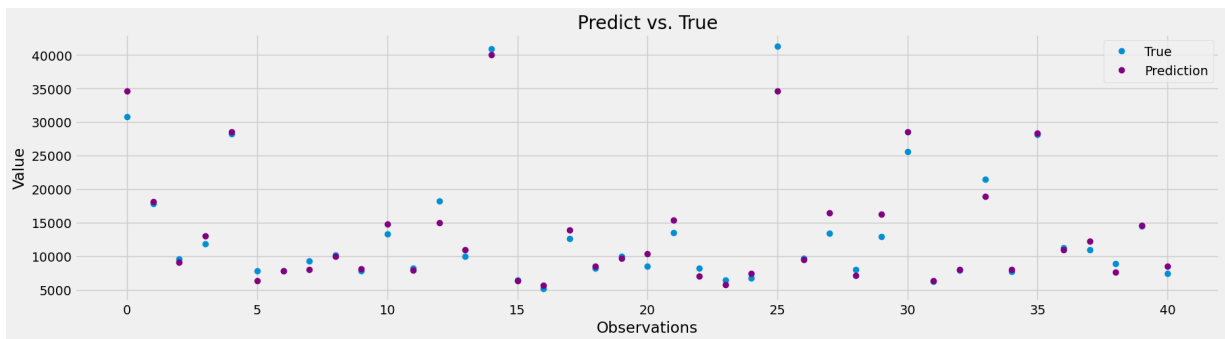
Out[89]:

	Precio_Predicho	Precio_Real
Precio_Predicho	1.0000	0.9791
Precio_Real	0.9791	1.0000

```
In [93]: plt.figure(figsize=(20, 5))

t = pd.DataFrame({"y_pred": y_pred, "y_test": y_test})

plt.plot(t["y_test"].to_list(), label="True", marker="o", linestyle="none")
plt.plot(
    t["y_pred"].to_list(),
    label="Prediction",
    marker="o",
    linestyle="none",
    color="purple",
)
plt.ylabel("Value")
plt.xlabel("Observations")
plt.title("Predict vs. True")
plt.legend()
plt.show()
```



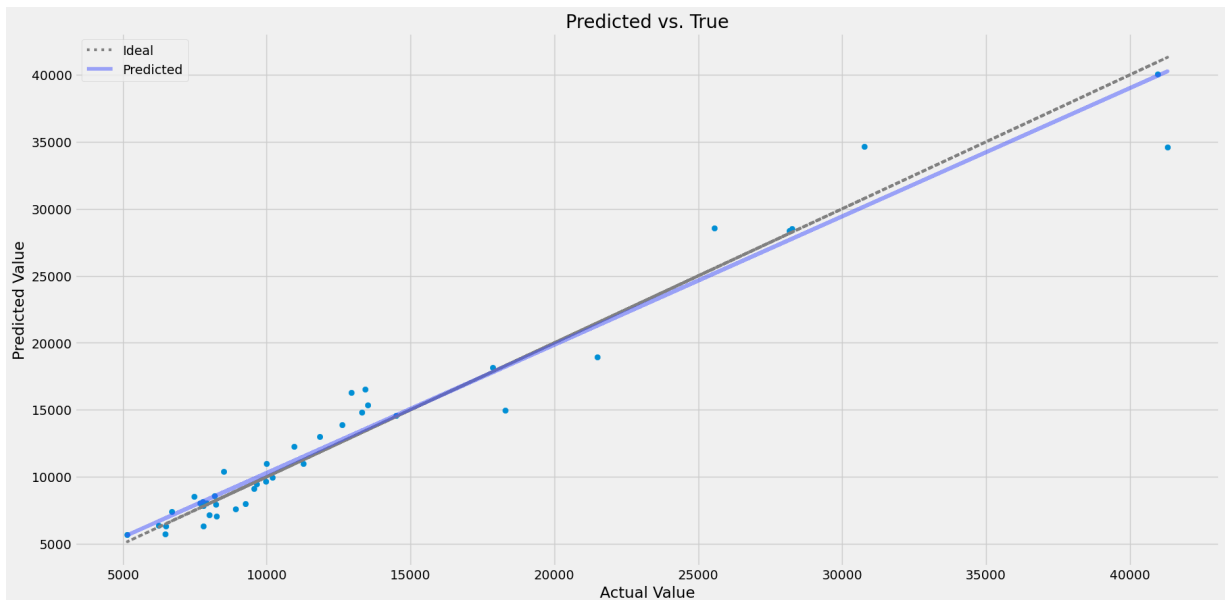
```
In [94]: # Create canvas
plt.figure(figsize=(20, 10))

# Plot scatter
plt.scatter(y_test, y_pred)

# Plot diagonal line (perfect fit)
z = np.polyfit(y_test, y_test, 1)
p = np.poly1d(z)
plt.plot(
    y_test, p(y_test), color="gray", linestyle="dotted", linewidth=3, label=
)

# Overlay the regression line
z = np.polyfit(y_test, y_pred, 1)
p = np.poly1d(z)
plt.plot(y_test, p(y_test), color="#4353ff", label="Predicted", alpha=0.5)

plt.xlabel("Actual Value")
plt.ylabel("Predicted Value")
plt.title("Predicted vs. True")
plt.legend()
plt.show()
```



**NOTA:** No terminé de exportar el modelo, pero es porque seguiré practicando próximamente, pero en scripts de python. Para el próximo proyecto ya estaré más fuerte.

## Y eso fue todo!! 😊

Gracias por haber llegado hasta acá. Si lo hiciste, qué opinas?? Crees que pude haber hecho algo mejor? Dejaló en los comentarios, quizá así aprendamos todos. Quizá para alguien que apenas está entrando esto le parezca abrumador, pero solo es constancia a través del tiempo. Seguiré desarrollando este superpoder 🦸‍♂️ Quizá el próximo proyecto lo haga en algo aplicable en la vida real 🤖...